

AWESOME GAME CREATION: NO PROGRAMMING REQUIRED

THIRD EDITION

- Provides a completely updated, start-to-finish guide for creating a variety of fun games—no programming required
- Teaches how to create several games from a 2D flying game, to an FPS, and a space shoot-em up
- Includes a CD with demo versions of the game engines used in the book, including Clickteam's *The Game Factory 2*, *GameMaker*, and *FPS Creator*, making game creation easy and affordable



AWESOME GAME CREATION: NO PROGRAMMING REQUIRED

THIRD EDITION

JASON DARBY



CHARLES RIVER MEDIA
Boston, Massachusetts

Copyright 2008 Career & Professional Group, a division of Thomson Learning, Inc.
Published by Charles River Media, an imprint of Thomson Learning Inc. All rights reserved.

No part of this publication may be reproduced in any way, stored in a retrieval system of any type, or transmitted by any means or media, electronic or mechanical, including, but not limited to, photocopy, recording, or scanning, without prior permission in writing from the publisher.

CHARLES RIVER MEDIA
25 Thomson Place
Boston, Massachusetts 02210
617-757-7900
617-757-7951 (FAX)
crm.info@thomson.com
www.charlesriver.com

This book is printed on acid-free paper.

Jason Darby. *Awesome Game Creation: No Programming Required, Third Edition*

ISBN-10: 1-58450-534-6

ISBN-13: 978-1-58450-534-1

eISBN-10: 1-58450-603-2

Library of Congress Catalog Card Number: 2007904967

All brand names and product names mentioned in this book are trademarks or service marks of their respective companies. Any omission or misuse (of any kind) of service marks or trademarks should not be regarded as intent to infringe on the property of others. The publisher recognizes and respects all marks used by companies, manufacturers, and developers as a means to distinguish their products.

Library of Congress Cataloging-in-Publication Data

Printed in the United States of America

08 09 10 11 12 TW 10 9 8 7 6 5 4 3 2 1

CHARLES RIVER MEDIA titles are available for site license or bulk purchase by institutions, user groups, corporations, etc. For additional information, please contact the Special Sales Department at 800-347-7707. Requests for replacement of a defective CD-ROM/DVD must be accompanied by the original disc, your mailing address, telephone number, date of purchase and purchase price. Please state the nature of the problem, and send the information to CHARLES RIVER MEDIA, 25 Thomson Place, Boston, Massachusetts 02210. CRM's sole obligation to the purchaser is to replace the disc, based on defective materials or faulty workmanship, but not on the operation or functionality of the product.

Publisher and General Manager, Charles River Media: Stacy L. Hiquet

Associate Director of Marketing: Sarah O'Donnell

Manager of Editorial Services: Heather Talbot

Acquisitions Editor: Heather Hurley

Marketing Assistant: Adena Flitt

Project Editor: Dan Foster, Scribe Tribe

PTR Editorial Services Coordinator: Erin Johnson

Copy Editors: Ruth Saavedra and Beth Roberts

Interior Layout Tech: Judy Littlefield

Cover Designer: Tyler Creative Services

DVD-ROM Producer: Brandon Penticuff

Indexer: Jerilyn Sproston

Proofreader: Sue Boshers

Image Credits

Figure No.	Copyright
2.3	Copyright Castle Software and Teddys Day Ltd
2.4	Copyright Clickteam.com
2.5	Copyright Clickteam.com
2.6	Copyright Castle Software and Teddys Day Ltd
2.8	Castle Software and Teddys Day Ltd
2.9	Copyright Teddys Day Ltd
2.10	Copyright Clickteam.com
2.11	Copyright Clickteam.com
2.12	Copyright Microsoft Corp.
2.13	Copyright Microsoft Corp.
2.14	Copyright Firefly Studios
2.15	Copyright Firefly Studios
2.16	Copyright Microsoft Corp.
2.17	Copyright Microsoft Corp.
2.18	Copyright Clickteam.com
2.19	Copyright Jason Darby and Castle Software Ltd
3.17–3.20	Copyright Jason Darby
8.1	Copyright Jason Darby and Teddys Day Ltd
10.1	Copyright Jason Darby and Teddys Day Ltd
12.1	Copyright Jason Darby and Teddys Day Ltd
13.11	Copyright Jason Darby and Teddys Day Ltd
23.1	Copyright Jason Darby and Castle Software Ltd
23.2	Copyright Jason Darby and Castle Software Ltd
23.3	Copyright Jason Darby and Castle Software Ltd
23.4	Copyright Clickteam.com
23.5	Copyright Empire Interactive
23.6	Copyright Jason Darby and Castle Software Ltd
23.7	Copyright Caligari

To my wonderful family,
Alicia, Jared, Kimberley and Lucas,
for all their support.

CONTENTS

	FOREWORD	xi
	ACKNOWLEDGMENTS	xii
	ABOUT THE AUTHOR	xiii
	INTRODUCTION	xv
CHAPTER 1	INTRODUCTION TO GAME DEVELOPMENT	1
	Setting Up a Game Studio	2
	Chapter Summary	9
CHAPTER 2	THE HISTORY OF GAME DEVELOPMENT	11
	Silicon Circuits	12
	Spacewar	12
	Assembly Language	14
	A Computer on a Chip	16
	Advances in Graphics	16
	It's a Polygon World	17
	Making Programming Languages Easier	17
	Game Consoles Shape the Future	19
	The Future of Game Development	20
	Game Genres	21
	Chapter Summary	34
CHAPTER 3	GRAPHICS: THE BASIC BUILDING BLOCKS OF A GAME	35
	Sights	36
	Basic Elements of an Image	38

Manipulating Images	45
Advanced Image Manipulation	50
Chapter Summary	57

CHAPTER 4 SOUND AND MUSIC 59

Why Sound and Music Are Important	60
Types of Sound	60
Obtaining or Creating Sounds and Music	61
Recording Sounds	62
Creating Music	67
ACID XPress	69
Dance eJay 7	79
Chapter Summary	87

CHAPTER 5 ELEMENTS OF DESIGNING A GAME 89

Introduction	90
Game Elements	91
Game Market	101
Technical Information and Associated Risks	102
Required Resources and Scheduling	103
Chapter Summary	103

CHAPTER 6 INTRODUCTION TO GAME MAKER 105

Installation	106
System Requirements	109
Game Maker Interface	109
Resource Explorer	110
Menus and Toolbar	112
Chapter Summary	116

CHAPTER 7 YOUR FIRST GAME MAKER PROJECT 117

Game Maker Basics	118
Creating a Simple Program	120
Save and Run	126
Chapter Summary	127

CHAPTER 8	2D SPACE SHOOTER—END OF THE EARTH	129
	Setting Up the Game	131
	Programming Objects	147
	Adding Sound Using a Script	168
	Adding a Help File	169
	Creating an Executable File	170
	Chapter Summary	171
CHAPTER 9	INTRODUCTION TO THE GAMES FACTORY 2	173
	About TGF2	174
	TGF2 Requirements	175
	Installation of TGF2	176
	Starting TGF2 for the First Time	179
	A Quick Introduction to TGF2	180
	Chapter Summary	183
CHAPTER 10	BEHIND THE SCENES OF THE GAMES FACTORY 2	185
	About Alien Wars	186
	Loading Alien Wars	187
	Alien Wars: The Storyboard Editor	188
	Alien Wars: The Frame Editor	191
	Alien Wars: The Event Editor	193
	Chapter Summary	201
CHAPTER 11	ALIEN WARS	203
	Library	204
	Initial Setup	205
	Event Programming	217
	Chapter Summary	253
CHAPTER 12	LITTER BUG	255
	Introduction	256
	Library	256
	Initial Setup	258
	Event Programming	269
	Chapter Summary	295

CHAPTER 13	ADVANCED GAME OVERVIEW	297
	Advanced Games	298
	Chapter Summary	315
CHAPTER 14	ADVANCED CONTROL OF OBJECTS	317
	Using Objects in Your Games	318
	Active Objects	319
	Backdrop and Quick Backdrop Objects	321
	Hi-Score Object	323
	Text Objects	325
	Lives Object	326
	Score Object	328
	Movement	328
	Multiple Movements	338
	Chapter Summary	339
CHAPTER 15	WORKING WITH PICTURES AND ANIMATIONS IN TGF2	341
	The Picture Editor	342
	The Animation Tool	354
	Chapter Summary	359
CHAPTER 16	INTRODUCTION TO FPS CREATOR	361
	Introduction	362
	Installation Walkthrough	363
	FPS Creator Terminology	365
	FPS Creator Creation Process	366
	FPS Creator Walkthrough	367
	Chapter Summary	374
CHAPTER 17	CREATING A BASIC GAME WITH FPS	375
	Creating Your First Room	376
	Testing Your First Level	380
	Player Starting Position	382
	Adding a Weapon	383
	Adding an Enemy Player	385
	Creating a Corridor	386
	Chapter Summary	394

CHAPTER 18	FPS CREATOR: NEXT STEP	395
	Adding Windows	396
	Creating Door Switches	398
	Lighting Rooms and Corridors	400
	World Effects: Smoke and Fire	401
	Making Your World More Visually Exciting	405
	Chapter Summary	406
CHAPTER 19	TAKING FPS CREATOR TO THE NEXT LEVEL	407
	Stairs, Elevators, and Teleporters	408
	Creating Enemy Patrols Using Waypoints	415
	Zones	417
	Chapter Summary	424
CHAPTER 20	FPS CREATOR ADVANCED OPTIONS	425
	Performance Checking	426
	Building an Executable	428
	Creating a Multiplayer Online Game	432
	Chapter Summary	436
CHAPTER 21	THE 3D GAMEMAKER	437
	System Requirements	438
	Installation	438
	Creating a Game with The 3D Gamemaker	441
	Saving the Game	445
	Playing the Game	445
	Chapter Summary	447
CHAPTER 22	GAMESPACE LITE	449
	System Requirements	450
	Installation	451
	gameSpace Lite Interface	454
	Creating Primitives	456
	A Simple 3D Example	459
	Exporting the Model	462
	Chapter Summary	462

CHAPTER 23	GAME MAKING WEBSITES	463
	Useful Websites	464
	Chapter Summary	470
APPENDIX A	DESIGN DOCUMENT: FIRST-PERSON SHOOTER	471
	Design History	471
	Game Overview	471
	Features	472
	The Game World	473
	Graphics	473
	Game Characters	473
	Weapons	474
	Music and Sound Effects	474
	Appendix ABC	474
APPENDIX B	THE KEY POSITIONS IN A DEVELOPMENT TEAM	475
	Designer	475
	Programmer	476
	Audio-Related Positions	477
	Art-Related Positions	477
	Producer	479
	Secondary Positions	479
APPENDIX C	ABOUT THE DVD	481
	General Minimum System Requirements	481
	ACID XPress (www.acidplanet.com) Trial	481
	Dance eJay 7 (www.ejay.co.uk) Trial	482
	Game Maker 7.0 Lite (www.yoyogames.com) Trial	482
	Games Factory 2.0 (www.clickteam.com) Trial	483
	FPS Creator (www.fpscreator.com) Trial	483
	3D Game-Maker (www.thegamecreators.com) Trial	484
	gameSpace Lite (www.caligari.com) Trial	484
	Folders	484
	INDEX	487

FOREWORD

I grew up in the golden age of video games. I remember the long lines of people in the local pizza parlor waiting in amazement when the first commercial pong game was installed, when Pac-Man fever swept the world and Donkey Kong introduced everyone to the legendary Mario.

Every kid I grew up with had a head full of ideas for making the next big video game, but the bar of learning computer programming kept the vast majority of people from turning dreams and ideas into reality. Only a select few ever attempted to put their ideas into action.

Fast forward a couple of decades and it's a completely different world. No longer is video game creation the sole domain of hard-core programmers; with many advanced visual tools, just about anyone can create commercial quality video games.

These new tools focus on the visual and logical flow of the game and do not require the mechanics of traditional programming. You're free to experiment and develop your ideas without a hassle or a time-consuming process. I believe we have already begun to see a new renaissance in video game creation with the massive success in the casual game market. As more and more "nonprogrammers" are given creation tools, we will see more and more new ideas come to life.

My good friend Jason has put together a wonderful outline of some of the popular and easy-to-use visual creation products. I encourage you to try them all and find one that suits your style.

Video game creation changed my life—and it can change yours.

Jeff Vance

Flyin V Interactive—Independent Game Developer

ACKNOWLEDGMENTS

I would like to thank a number of people who were involved in the creation of this book, without whose help it would not have come to fruition.

To my wife Alicia, and my children, Jared, Kimberley, and Lucas, who supported me throughout this project.

To Raymond of Teddysday Ltd, who created some amazing graphics and games for the book, and the front cover image. Without his help, I would have taken a lot longer to complete the book.

To my good friends Yves Lamoureux and Jeff Vance, who provided help and support to ensure the book is as complete as it can be.

To the professional and very friendly staff at Thomson Learning, who again provided excellent support throughout the entire process.

ABOUT THE AUTHOR

Jason Darby has been working in the IT industry for the past decade, writing user and systems documentation for users with little or no knowledge of the programs they are using. For a number of years he has been the director of his own company, Castle Software, Ltd., working in the games and application creation market, where he makes games, applications, and DVD demos. Jason is the author of *Make Amazing Games in Minutes* and *Power Users Guide to Windows Development*, which are also published by Charles River Media.

He has also had a number of articles published in the UK press including several in *Retro Gamer*[®] and *PC Format*[®], both leading magazines in their field.

This page intentionally left blank

INTRODUCTION

Welcome to *Awesome Game Creation: No Programming Required, Third Edition*. This book is aimed at anyone who wants to make exciting and fun games for Windows. This book will show you how to make a number of games in different drag-and-drop and no-programming-required game creation systems. You will learn to make games in four different game making packages as well as use other tools that will aid you in your game creations.

Audience

If you've purchased this book or are reading it in a bookstore, we can assume you're interested in developing games for the Windows platform. You may be an indie developer looking to make shareware programs, a multimedia designer, or a marketing manager looking at making games to advertise a service or Web site. You may be a skilled graphic artist who does not want to learn a programming language to create games or have to find a programmer to help realize your ideas. You might even be an educator or someone working for a software company who is looking at making games without the need for skilled programmers or reskilling your current staff. Whichever group you are from, you're reading the right book.

Aim of the Book

The aim of the book is to allow anyone with no programming background (or in fact even if you are a programming professional) to create a whole range of games for the Windows Operating System quickly and easily.

Some of the things that are covered in this book are:

- Understanding the history of games
- Learning about game genres
- Game design and storyboarding
- Jobs in the games industry
- Learning how to use Game Maker 7
- Learning how to use The Games Factory 2 (TGF2)
- Learning how to use FPS Creator

- Learning about objects in The Games Factory 2
- Learning about creating your own sounds
- Learning how to create your own music
- Learning how to use eJay
- Learning how to use ACID XPress
- Creating a space shoot-'em-up
- Creating a space invaders-type game
- Creating a 2D collection game

We've tried to include everything that we feel would be useful to anyone wanting to make their own games, from designing them to finding the right tool, to creating them. By the end of the book you should be very comfortable with the software tools available in this book and know which one will best suit your goals. We hope you will then be able to make your own ideas a reality.

This book does not:

- Teach more complex programming languages such as C++, C#, or Java. This book is aimed at those who want to make games easily without needing to learn those more complex languages. If you are interested in C++, then consider *C++ Programming Fundamentals*, by Chuck Easttom.
- Teach how to be a graphic artist or music creator. Look at *Composing Music for Video Games*, by Andrew Clark, or *3D Graphics Tutorial Collection*, by Shamms Mortier.
- Show you how to become an indi developer or build a team. If you want more information on being an indi developer read *The Indi Game Development Survival Guide*, by David Michael.
- Assume you are an expert at game creation. This book is aimed at those with little or no knowledge of game creation but also those who might have an idea of how things are put together but need more information.
- Show you how to make Windows-based applications. This book is totally geared to games and game creation. If you want more information on making your own Windows applications, read *Power Users Guide to Windows Development*, by Jason Darby.
- Concentrate on a single product but covers many different tools so you can get as much knowledge about the programs available to you. You can then make an informed choice on which to use for a particular project. If you are looking for more information on TGF2 then read *Make Amazing Games in Minutes*, by Jason Darby.

Chapter Overview

This book runs in a simple yet effective order to allow you to get the most out of reading it. It is possible to skip certain chapters, but it is recommended that you read through every chapter in order. Different products are used throughout the book, and using each of them will give you more knowledge about the game creation genres and overall what makes a good game.

- Chapter 1: Introduction to Game Development.** The book begins with some general advice on the type of equipment available to the budding game developer and those wanting to create their own game studio.
- Chapter 2: The History of Game Development.** A look back into the past at how the games industry started and what happened after that.
- Chapter 3: Graphics: The Basic Building Blocks of a Game.** Provides information on different graphic settings, techniques, and features found in most paint packages.
- Chapter 4: Sound and Music.** Reasons for using sound and music in your creations and how to create and record your own.
- Chapter 5: Elements of Designing a Game.** Things you need to look at when designing your own games, including the technology involved, the team you need to create your game, and the target audience.
- Chapter 6: Introduction to Game Maker.** An introduction to the first game tool that we will use in the book. We will install Game Maker 7.0 Lite and have a tour of the program and be ready to begin creating our first game.
- Chapter 7: Your First Game Maker Project.** You make your first game, a simple example that shows you all the important aspects of the Game Maker functionality, including how to save and run the program.
- Chapter 8: 2D Space Shooter—End of the Earth.** It's time to create a stunning shoot-'em-up game and learn in depth about the Game Maker features.
- Chapter 9: Introduction to The Games Factory 2.** An introduction to the second game making tool used in the book, TGF2. You will learn about its requirements and how to install it and have a quick introduction to the basic terminology of the program.
- Chapter 10: Behind the Scenes of The Games Factory 2.** Before you begin to make your first game in TGF2, you will get a walkthrough of the main editors and screens used in the program by looking at the game you will make in Chapter 11, called Alien Wars.
- Chapter 11: Alien Wars.** It's time to make your first game with TGF2, a space invaders-type game. Making this game, you will learn a lot about the Event Editor and the Frame Editor, two of the most commonly used editors in the program.
- Chapter 12: Litter Bug.** Now that you have completed your first game, you can attempt the second game in TGF2, called Litter Bug. You play the part of a robot cleaning machine. In this chapter you will learn many new techniques, including how to make your own movement engine.
- Chapter 13: Advanced Game Overview.** In this chapter you will be introduced to two advanced games made in TGF2. You get to take a tour of how they were put together and see new features and functionality that you could include in your own games. The first game is a card game called Black Jack, and the second game is a side scrolling game involving a fire-breathing dragon.

Chapter 14: Advanced Control of Objects. In this chapter you will look at the additional objects you can use in TGF2 to increase the power of your games. You will also be introduced to the different built-in movements that are available with TGF2 out of the box.

Chapter 15: Working with Pictures and Animations in TGF2. In this chapter you will look at how to create pictures and animations using The Games Factory's built-in picture editor. You will learn how to import or draw your own pictures and then animate them.

Chapter 16: Introduction to FPS Creator. We take a look at the third game-creating tool in the book, the FPS Creator. This program allows anyone to make first-person shooters without any programming knowledge. We begin with a basic walkthrough of the terminology and the interface.

Chapter 17: Creating a Basic Game with FPS. Here we create our first FPS game with a gun and a single enemy player.

Chapter 18: FPS Creator: Next Step. Now that you have created your first game in FPS Creator, you will be taken through some additional functionality to start building upon what you have learned. You will learn to create windows, doors, switches, smoke and fire effects, and much more.

Chapter 19: Taking FPS Creator to the Next Level. In this chapter you will learn about adding special features that will make your game stand out, including stairs, elevators, and teleporters.

Chapter 20: FPS Creator Advanced Options. In the final chapter about FPS Creator you get to learn how to make advanced changes to your games and are walked through the creation of an online multiplayer version of your game.

Chapter 21: The 3D Game-Maker. The final game making tool in the book is The 3D Game Maker, a simple and easy-to-use product. It's not the most recent of game creation engines, but it provides good insight into the different 3D game genres that are available. You will learn how to install the product and make a simple 3D game by the end of the chapter.

Chapter 22: gameSpace Lite. In this chapter we explore a product called gameSpace from Caligari. This is a product for making your own 3D models, which you can then use in your games.

Chapter 23: Game Making Web Sites. A quick look at some of the useful websites you can visit to help you in your game making.

Appendix A: Design Document: First-Person Shooter. A design document detailing the making of an FPS game. It gives you a document template that you can use in the design of your own games.

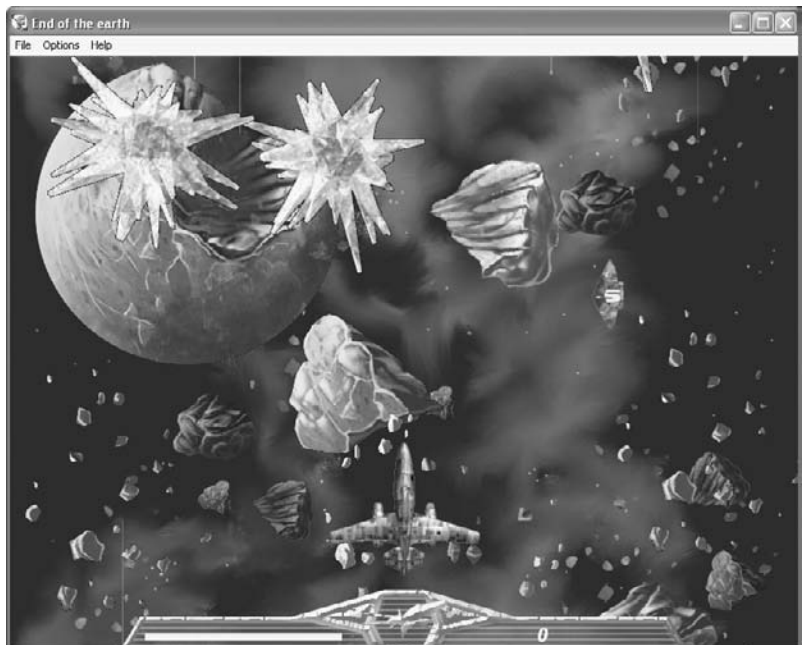
Appendix B: The Key Positions in a Development Team. This appendix details the types of job roles that are available in most game development environments.

Appendix C: About the DVD. This final appendix provides additional information about the DVD-ROM included with this book.

INTRODUCTION TO GAME DEVELOPMENT

In This Chapter

- Setting Up a Game Studio



Developing a computer game is a unique production, in which you combine a wide range of elements into what you hope will be an enjoyable experience for the end user. Games consist of a variety of components, which can seem overwhelming for a new developer. In this chapter, you'll look at what you need to set up your own development studio. Later chapters will introduce you to the various components that make up a game project and will walk you through the creation of several complete games.

SETTING UP A GAME STUDIO

Before you can make anything, you need to have the proper equipment. While it may sound expensive, setting up a game development studio doesn't have to be. With Moore's Law continuing to hold true (i.e., the processing power of computers doubles every 18 months), the cost of computers continues to plummet. Great deals for relatively powerful computers are everywhere. As many families are embracing the digital age and have many different types of digital-based equipment at home, you may find that you have some or most of the equipment already.

To go along with inexpensive computers, the variety of software designed for small game developers has greatly increased in the past couple of years. With these tools, you can now develop games without doing any programming.

When setting up your game studio, several factors help determine the type of equipment you need. Fortunately, you may already have the essentials of a game studio—a computer and this book. In this chapter, you'll learn how to determine if what you have is enough, and what else you may need.

As an aspiring game developer, you have a wide range of computers from which to choose, and trying to decide which system you need can be a daunting task. One way to look at this problem is to compare it to the purchase of other items, such as an automobile. For instance, if you were driving six kids to school, driving in a road race, or driving into combat, what vehicle would you choose? Computers are similar to vehicles in this respect. While a minivan, a racecar, and a jeep all have four wheels, each is designed for very different purposes.

So the big question for you is, what will you be doing with your computer? This book will help you answer this question, by giving you a chance to try the different types of things you will have to do on your computer as a game developer. After you have worked a bit with the various applications and learned their specific requirements and your needs as a developer, you will know what kind of system you need.

The first thing to consider while working on your current system is the system requirements for the applications you will be using or intend to use, which appear on the box, in ads, and on the home pages of the product. The system requirements are usually broken down into minimum and recommended.



Usually, the minimum system requirements are just that, the bare minimum to run the application. A minimum system will usually not be the most comfortable or even the most usable system to run the application. Moreover, the minimum requirements do not take into account other applications you may be running at the same time.

The more things you expect your computer to do, the more strain on the minimum requirements. Modern operating systems require much larger hard disks, processors, and memory amounts just to be able to run without needing to consider everything else you might have running. Let's say that the minimum RAM requirement for your art application is 64 MB. However, as a game developer, you also need to run other applications at the same time, such as a level editor, game engine, word processor, and 3D application. You might also have other programs running in the background to help protect your system from viruses, and a software firewall to protect your computer when you are on the Internet. These programs will severely tax your system and cause it to run poorly, if at all. And the minimum system requirements usually do not take into account the files with which you will be working. If you have experience with image editing applications such as Photoshop or Paint Shop Pro, you know that files can range from a few hundred kilobytes to over 50 megabytes, depending on what you're working on. While you can open and close applications that are not in use, this takes time (especially with slow, RAM-deficient machines) and will severely cut into your productivity and workflow.

Another area you should watch is the recommended amount of hard drive space for installing the application. This number includes only the application itself; it does not take into account the files you create with the application. Therefore, you also need to ensure that you have room for your files. Processor speed is another variable you should look at, which again only includes the speed to run the application and does not take into account larger files.

System and Equipment

The equipment you will need to create a computer game depends on the type and scope of your project. The right setup can range from a minimal investment of a few hundred dollars, to tens of thousands of dollars for the latest and most powerful computer and peripheral setup. To get started, you need to own a basic computer setup with a few important peripherals.

Computer

A computer is obviously a necessary item for game development. As previously mentioned, you can get many great deals these days for a minimal investment. Unless the requirements for your software indicate that you need a high-end system, a general-purpose off-the-shelf system is sufficient.

When purchasing your system, consider the work and applications you will run. The operating system (OS) is important (Windows XP or above is recommended for the tools in this book). New systems usually ship with the latest version of the biggest OS on the market at the time. The minimal system today usually has a 17-inch monitor, lots of RAM, and a fairly large hard drive. You should have no problem with an off-the-shelf or mail-order system from a reputable company.



See the end of this chapter for tips on buying equipment.

Processor

The processor can often be very difficult to upgrade. With this in mind, you should try to buy the fastest system you can afford. There are two main manufacturers of processors on the Windows side of things: Intel (Pentium), and AMD (Athlon). We won't get into a big discussion or try to decide which processor you should buy; you can simply assume they are comparable.

One reason you should buy the fastest processor you can is that it's harder to upgrade the processor than to upgrade other components. Getting the fastest chip possible makes sense if you are purchasing a system for general work. It's even more important for you as a game developer; you'll be pushing your system harder than most users and will need the speed. But don't worry if your system isn't the latest and greatest. You can still design and develop games with a minimal system, as long as it can run the specific applications you are using.

RAM

Along with the fastest processor you can afford, you should get as much RAM as possible. RAM stands for *random access memory* and is measured in megabytes. The computer uses RAM as *temporary* storage for the applications you run. When you turn off the system or the power goes out, the information in RAM is lost. Although RAM is cheap and very easy to upgrade, the prices are so low now that it's often a good idea to purchase a system that has a slightly slower processor, but more RAM. This results in overall better performance at less cost. RAM is definitely the most important thing you can have.

Graphic (Video) Cards and 3D Cards

Having a quality video or graphics card is becoming more important as time goes on. These cards allow images to appear on your monitor. A video card usually controls how big the image is on your screen, how much detail the image can have, and how many colors are displayed (in Chapter 3, "Graphics: The Basic Building Blocks of a Game," you'll learn about the specific elements of an image).

Many applications only display simple pictures, but if you are interested in doing 3D-related games, it makes sense to consider buying one of these cards. Most new

systems will have a hardware-accelerated card, but the type of card and the amount of graphics memory it has will affect your performance. Two manufacturers are head and shoulders above the rest—NVIDIA® with its GeForce line of cards, and ATI with the Radeon line. Regardless of the type of card you get, it will take the tasks of 3D rendering away from the processor by handling textures, effects, and geometric calculations.

Other Peripherals

Other peripherals you will need are standard on most computers: a modem, a CD-ROM or DVD-ROM drive, and a sound card. Your system might come with a modem—the most common type of which is DSL—that can allow you to transfer files with speed across the Internet. Your system will probably have a CD-ROM or DVD-ROM drive; simply choose the type that benefits you the most. The sound card sends sound output to a set of speakers. There are many manufacturers of, and many options for, these cards. Again, choose a sound card that meets your requirements.

Last, you will want to consider several other peripherals if you have the extra funds.

Scanner

A scanner works like a copy machine; it converts your flat document or image into a digital image that can be manipulated in the computer, as described in Chapter 3. This can be very useful for creating game art, Web sites, and logos.

Digital Camera

The next item is a digital camera. Digital cameras work like standard cameras, but instead of using film, they produce digital images, as a scanner does. The major difference is that a scanner requires flat images that have already been created on paper, while you can use a digital camera to take a picture of anything. Digital cameras come in many different sizes and are generally judged by the number of megapixels they can create. The megapixel size isn't the only measure of a good camera, but it does give you an idea of its potential quality. A camera with a megapixel count of three or above is good enough to take quality pictures. Today, you can purchase many cameras for less than a few hundred dollars.

Modem

Over the last few years, there has been an explosion in the use of cheap DSL Internet access. These DSL modems and lines have taken over from the 56 K modem speeds, which seem very inadequate into today's video, music, and downloading Internet experience. DSL stands for *digital subscriber line* and has been around for a number of years. It gives better speed access than the 56 K PSTN (standard telephone) line modems, and because the actual line is split in two, you can still receive

and make outgoing telephone calls when using the Internet. Due to growing competition in the marketplace and the number of people using the service, the prices have continued to drop, and the monthly fee for DSL is much cheaper than a telephone call to use the Internet. The Internet is such an invaluable resource, especially to game developers, that paying for high-speed access is a worthwhile investment. Some of the large downloads you will be making are images, game demos, sound files, development tools, and animation files.

Backup Devices

The next item is rapidly becoming an affordable necessity. That's because the prices of recordable CD-ROM drives and media are now very low, and nearly all systems now come with them as a standard item. There are a number of different types of recordable drives: a CD-Recordable (CD-R) drive that can write to a given CD only once, and a CD-Rewritable (CD-RW) drive that can write (and erase) the media many times. After you start creating content for your games, you will need a way to back it up. A CD-R or CD-RW drive is perfect for this. Working in the same way as CD-R and CD-RW is the DVD writable format, which allows users to place much more than the 700 MB of a CD-R disk onto its media; in fact, it can store around 4.7 GB of data. Unfortunately, this is where it becomes slightly more complex than the CD-R format, as a number of manufacturers were in competition to create their own formats. Initially, these formats were split into two groups: "+" and "-." So you could purchase a DVD-R, DVD-RW drive, or a DVD+R, DVD+RW drive. You also had to make sure you purchased the correct format media, as a "+" disc wouldn't work in a "-" drive. Fortunately, the market decided that it was too confusing and came up with a solution: multiformat drive support. When considering a DVD writer, many will support both the "+" and "-" formats. One area in which you might find a problem is that some multiformat drives will support both formats on single recordable discs, but may only support one format on rewritable media.

One area of further development in the basic DVD drive arena is the concept of dual layer, the capability to write information to two different layers on the disc. This doubled the capacity of the standard discs to 8.55 GB. Again, a special drive and media are required to support this type of device.

A new format war is currently happening between two new DVD formats: BD (Blu Ray Disc) and HD DVD (High Definition). HD can hold 15 GB of data per layer, and Blu Ray can contain 25 GB per layer. Currently, the cost of a writer and media is prohibitively expensive, but as the formats become more established the cost will reduce considerably—and there are already talks of multiformat supported drives. It is very unlikely that your game will require such space for backup, but it will allow you to back up multiple projects and other files to one disc if required.

Besides CD, DVD, and the high-definition writable drives, you have several other options for backing up and storing your content. Drives such as the Iomega Zip® drive can store data to around 70 GB per Zip disk, and other options for tape backup drives

can hold several gigabytes of data. A popular storage device is the external hard disk drive; for less than a couple hundred dollars, you can plug in a USB-based drive, back up, and then take the drive with you. This is still using standard hard disk technology, so unlike the CD or Zip formats that can easily be protected on the move, a hard drive is still susceptible to being dropped or knocked.

Some computers may still contain a floppy disk drive, but these only hold 1.44 MB of data, and are now being phased out of most PCs.

Graphic Tablets

Graphic tablets are a hardware device that contains a board and a pen, and allows artists to draw more naturally on the computer. The user moves the pen over the board, and can replicate drawing on paper on to the computer screen. Over the last few years, these once expensive devices have become relatively cheap, and although far from a necessity, they can be useful for people who prefer drawing with a pen, rather than the mouse. The less expensive graphic tablets are good for basic sketching, but may lack the fine control an artist needs.

Network

A network is another very important item that allows multiple computers to communicate with each other. While this sounds like an expensive proposition and a complex undertaking, it is a very achievable goal. You can purchase a good SOHO (Small Office Home Office) network system for under \$100. It comes in a kit with everything you need, dramatically extending your computing capabilities. One benefit of a home network is being able to share peripherals and resources. You can have one scanner, printer, or other device on the network, and have it be available from multiple computers. This can be useful because most computers (especially older PCs) can only be connected to a limited number of devices. In addition, having many devices installed on a system tends to slow down the system's boot-up and response times. Having a network also lets you easily back up data on multiple PCs. During development of a project, having a network is almost essential, because multiple team members can simultaneously update code and resources.

Ethernet Network

Ethernet was the most common home-networking system and the easiest to hook multiple computers into. A typical system for two computers uses two cards, called Ethernet cards, and a special cable called a crossover cable. If you have three or more computers, you need a *hub*. You plug all the computers into the hub, and it routes, or directs, the traffic. The software portion of a network can range from simply finding the other computers on the network and accessing the data on their drives, to setting up special software that operates peripherals and adds security, chatting, and other advanced functions.

Wireless Network

The most popular type of network today is the wireless network. The popularity of the wireless network came about when people began moving to DSL Internet connections and didn't want to put Ethernet cables all around their house. A wireless network allows you to share information between computers without physically connecting them. The biggest problem with wireless networks is that they do require a little more technical knowledge to get them up and running, and more importantly, if not set up correctly are more susceptible to computer hackers. Much wireless equipment that connects to the Internet is not locked down by default and many users forget to configure it, which means anyone in close proximity may be able to find your network and try to access it.

A Good Chair and Desk

One last suggestion is to buy a good chair and desk. You will be sitting for long periods of time, so this will prove to be an invaluable investment.

Tips for Buying Equipment

Now that you have some idea about the type of hardware you'll need to purchase, here are a few common sense tips to keep in mind:

- **If you are not paying by cash, use a credit card.** You should use a credit card, especially when buying online. With a credit card, you have the credit card company and usually more rights as a consumer. Many countries have less risk to the buyer if you pay by credit card; for example, if you purchase online and the company goes bankrupt before it ships your goods, you could lose your money. If you are using a credit card, in many cases it allows you to claim your money back. Always ensure you are purchasing from a reputable company before entering your credit card details online. Check with your credit card company Web site for more information on your rights as a consumer.
- **Don't try to cut costs.** Avoid the so-called "budget" computers unless you *really* know what you are getting into. In some cases, these systems may not include components that meet your needs (such as a larger hard drive and a quality monitor). Expect to pay about \$1,500–\$2,500 for a computer with all the key features. Depending on your experience, it may be a good idea to get an extended warranty, although many new systems come with three-year warranties. (After three years, your system will probably be behind the times and need replacing.)
- **Protect everything.** Buy a recordable CD-ROM drive or a Zip drive. Try to back up data daily to a Zip drive (or to another computer on your network), and monthly to a CD-ROM. You can never be too safe. If you want to protect from power surges or power cuts, buy a battery-operated surge protector or UPS (uninterruptible power supply). For about \$100, you can get a UPS that will protect several components, including your computer, monitor, and key equipment. The UPS will also allow you plenty of time to save your work and shut down your

computer if the power goes out. You should also get a surge protector, which are easy to use; you just plug them in and plug your computer into them. They will protect your computer from power spikes and shutdowns. A surge protector will actually blow a fuse or circuit if a surge of electricity from lightning or bad wiring hits it, which keeps your computer's innards safe. Of course, the best protection is to turn off your computer and unplug it during thunderstorms.

- **Research.** Above all, learn about computers. If possible, try the applications you expect to run on a few systems first. See how those systems handle massive graphic files and huge levels. And remember, most people are very biased about their own systems, so be careful when you ask others for their opinions. No matter how many opinions you get, you'll need to make up your own mind.

CHAPTER SUMMARY

In this chapter, you learned about the basic components you will need to create a development studio. Once you have assembled your game development studio and have it up and running (whether it is an off-the-shelf special or the latest and greatest system money can buy), you will have made a huge step toward becoming a game developer. The next step is to learn about the history of computer games, and then the basic building blocks of a game.

This page intentionally left blank

THE HISTORY OF GAME DEVELOPMENT

In This Chapter

- Silicon Circuits
- Spacewar
- Assembly Language
- A Computer on a Chip
- Advances in Graphics
- It's a Polygon World
- Making Programming Languages Easier
- Game Consoles Shape the Future
- The Future of Game Development
- Game Genres



In this chapter, we'll look at the history of computer game development. Understanding the history of something helps you appreciate where you are and what you are working with. We'll look at how the computer gaming industry began and how the industry has evolved into what it is today.

The game and interactive developer has come a long way from the days when one had to memorize complicated codes and numbers to work on a game. Basically, you had to be a programmer, and the focus was on the code, not the art. Currently, anyone can make a game 2D and 3D. The doors have been opened for great artists to contribute to a game, and even for the lowliest newcomers to try their hands at game design and development. Let's look at how far we have come.

SILICON CIRCUITS

In 1959, Jack St. Kirby at Texas Instruments, and Robert Noyce and Jean Hoerni at Fairchild Semiconductor Corporation independently devised a way to shrink much of the redundant and sluggish elements on an electronic circuit board and place them all onto a tiny square of silicon. It was called the integrated circuit; you know it as the microchip.

The year 1959 laid the path that led to the computer as we know it today, but there were a number of hurdles to jump before there was a PC on every desk. When the microchip appeared, it was hampered by high prices and very small stock, much like when any new technology debuts. Did this stop the advent and evolution of computer games? Of course not. The computer had already been invented. "How," you may ask, "did they do it?"

Before the microprocessor, everything was "solid-state." This refers to a circuit board full of electrical components that provide a system of computing power and temporary memory. The capacitor played the lead to this troupe. A capacitor could hold electric charges, negative or positive, for a variety of purposes. It was, in a room-sized nutshell, the world's first RAM. Indeed, it was on a "solid-state digital computer" that the first computer game would be written.

SPACEWAR

In November 1960, Digital Equipment Corporation (DEC) debuted the first of a widely successful computer line, the Programmed Data Processor (PDP). The first PDP—PDP-1—showed up at The Hingham Institute in Cambridge, Massachusetts, where J. Martin Graetz and his colleagues awaited it. Everything the group had read about the PDP-1 told them it would be the world's first useful computer, the world's first "toy computer," as Graetz put it in a 1981 issue of *Creative Computing* magazine.

In Graetz's words, "The PDP-1 would be faster than the Tixo, more compact, and *available*." (The Tixo, a nickname for TX-0, was an earlier computer, also at Hingham.) He adds, "It was the first computer that did not require one to have an Electronics Engineering degree and the patience of Buddha to start it up in the

morning; you could turn it on any time by flipping one switch, and when you were finished you could turn it off. We had never seen anything like that before.”

It was in the Institute’s “kludge room,” next to the Tixo, that the PDP-1 resided. Graetz, a published author, along with Stephen R. “Slug” Russell, an artificial intelligence specialist, and Wayne Witanen, a mathematician, had all experimented with coding on the Tixo for months, showing off such things as “Bouncing Ball,” which was advanced for the late 1950s. When they sat around the PDP-1, they transferred Tixo code and rewrote it for the PDP to get a feel for the new “toy.”

They wrote and rewrote, trying new experiments. They tried various ideas. A particular favorite was lines intersecting one another, or the “Minskytron.” Soon enough, two spaceships appeared, a sun, and then a star field—images fueled by a recent, healthy dose of 1950s pulp science fiction in the form of E.E. “Doc” Smith’s Lensman novels. They added features such as a way to rotate the ships, a thrust, and torpedoes. The result was Spacewar. On the PDP’s so-called “Precision CRT Type 40” monitor, two spaceships drifted against a backdrop of silent stars. In the middle, a larger star grew and shrank, tugging the ships toward it, as seen in Figure 2.1.



FIGURE 2.1 A screen from the original Spacewar on the PDP-1.

In Spacewar, players flipped console switches to control their spacecraft—one for clockwise rotation, another for counterclockwise, one to shoot “torpedoes,” and the last for thrust. The game looked very much like the arcade game Asteroids; white outlines and dots made up the figures against a “black” background.

(This graphics style would come to be known as “vector graphics.”) The PDP-1 also allowed two users to operate the computer simultaneously. That’s right, the world’s first death match!

Immediately, Institute members and students from nearby MIT took a liking to Spacewar. And as more computers showed up on campuses around the country, Spacewar was often copied. You can still find Spacewar on the Internet. It has been ported (or re-coded) to other computer languages, such as Java, and can be played in your Web browser.

Soon after Spacewar, new and different games started to appear. Adventure, the world’s first computer text adventure, was created shortly thereafter, as were Lunar Lander, Hammurabi (the first simulated (sim) world), and many others. Spacewar had made a single, clear point: computer games were fun and cool.

ASSEMBLY LANGUAGE

As cool as Spacewar was, it was difficult to program at least by today’s standards. The programmers had to write the game in a proprietary code only the PDP-1 could understand. Indeed, this language could be called a form of *assembly language*.



Machine language speaks directly to the computer hardware and tells it what to do. Assembly language is a level above this in ease of use, and above Assembly are the high-level languages such as C++, Java, VB, .Net, and Python (see Figure 2.2).

As new programming languages appear, the rules for what is and isn’t a high-level programming language are redefined. You may come across languages such as FORTRAN, C, and Pascal; in certain circles, these are still used and are considered some of the earlier high-level languages.

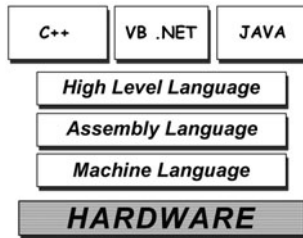


FIGURE 2.2 The hierarchy of computer languages.

At the most basic levels, computers can only process a low-level code called machine language. All computers understand machine language, but do humans? Forget about it. It consists entirely of numbers.

Assembly language is one step toward what are known as “human-readable” languages. Instead of numbers, it uses labels, or codes, that tell the processor to perform different functions. Assembly language is “readable” in the sense that the different codes have a structure and format people can understand. Well, at least that’s the way the theory goes.

Here is an example of assembler code:

```

.$13e3 [26 61 ] ro1 $61
.$13e5 [26 62 ] ro1 $62
.$13e7 [26 63 ] ro1 $63
    
```

```
.$13e9 [26 52 ] rol $52
.$13eb [26 53 ] rol $53
.$13ed [26 54 ] rol $54
.$13ef [a5 54 ] lda $54
```

Assembly language was, and is, difficult to master. It's about the closest a programmer can come to understanding a processor's native tongue. Master it, and you can speak to your processor. But remember: your processor can only understand one particular dialect. Maybe you've learned the assembly language for an 80x86 or Pentium processor, and now you want to write assembly language for a 6502 processor. You'll have to learn another dialect.

Spacewar was written in assembly language (many early games were) and it remained a staple of game programming for nearly 20 years. If someone wanted to port Spacewar to another processor, he had to rewrite it in that processor's assembly language. This is still true today. However, it became especially significant in the early 1970s, when another invention changed everything.



*Later in this book, we'll cover *The Games Factory 2 (TGF2)* and *Game Maker*. Using these programs, you can create a drag-and-drop game in minutes that is technically far beyond Spacewar, with textures, sounds, and much more (see Figure 2.3). You'll see that when you can drag and drop to create a game instead of hand coding everything, your creativity can really take off.*

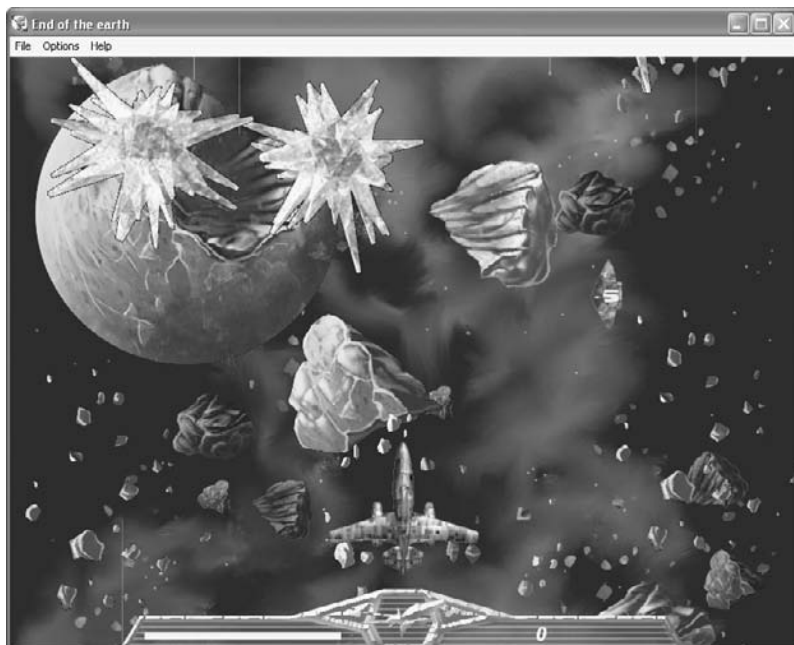


FIGURE 2.3 A game that is similar to Spacewar and technically superior, due to modern tools.

A COMPUTER ON A CHIP

In the late 1960s, science writers in various trade and consumer science magazines theorized about a so-called “computer-on-a-chip.” The microchip was still fresh in many scientists’ minds, and many wondered about that next step. Everyone concluded that integrated circuits weren’t where they needed to be. Certainly, many more experiments would have to be conducted before the “computer on a chip” could become a reality. It would take years! People weren’t waiting and holding their breath.

By the late 1960s, Intel had invented a so-called “MOS technology” (metal oxide semiconductor), which used the inherent properties of silicon to create gates that insulated conducting channels from nonconducting ones. This, theorized Ted Hoff, Stanley Mazor, and Federico Faggin, would make a single-chip CPU possible.

Suddenly, it wasn’t theory anymore. In 1971, Intel officially announced the first microprocessor, the Intel 4004, a single chip as powerful as ENIAC, the giant first electronic computer that filled an entire room. The 4004 was more of a technological curiosity than anything else; however, it did spur development of other microprocessors. Rockwell introduced the 6502 microprocessor series in the mid-1970s, which would power the Atari 2600 and the Commodore 64. General Instruments developed the 1610, which Mattel used in the Intellivision. And, of course, Intel developed the 8088, and later, the 80x86 family (80186, 80286, 80386, 80486, and Pentium). Motorola achieved great things with its 68000 and PowerPC® series of chips.

Throughout the life span of each processor, games were developed for it—some simple, some bad, some ingenious. Each tried to squeeze just a little more out of each processor. That squeezing continues today.

But at some point, the microprocessor itself, while certainly remaining at the forefront of developers’ minds, became a little less important. The speed was there, and it continued to accelerate as new and better processors came out. Then, many developers turned their attention to graphics. The question was, “How do we make better pictures?”

ADVANCES IN GRAPHICS

It’s a perpetual battle: gamers want better graphics. They see the games in the arcade and they want to bring all that color and explosive sound home. Developers want to give consumers all that and more, because they like the same graphics gamers do. In the middle lies the hardware, pulled at from both ends. Developers had to forge a compromise.

The first compromise was vector graphics, which consist of light stretched into lines or squeezed into points. Remember the original Asteroids? Tempest? Battlezone? All vector. In its earliest days, colored gels were physically placed on the screen to color the light.

However, vector graphics didn’t appear outside of the arcade very often. The world’s only vector video game console, the Vectrex, is now a highly sought-after collector’s item. But for the millions of people who owned Atari 2600s, Apple IIs,

Spectrum 48Ks, and Commodore 64s, the developer/processor/gamer compromise was *sprites*.

A sprite is a graphic image that can move within a larger image. Remember Pac-Man®? Pac was a sprite, as were the ghosts and the dots. Even the maze walls were sprites. Each sprite could be animated to move about a game board or “world,” or stay in one place, acting as a border or barrier. Usually, the character or machine you controlled in a game (Pac) was a sprite. It could be decoration. It could collide and react to other sprites, as we’ll see later when we look at TGF2.

By the time the first round of home computer systems debuted, the ability to draw sprites on the screen was available in just about every computer language. Some processors were even created with them in mind. As such, sprites ruled games for more than a decade.

However, sprites had one inherent flaw: they were 2D flat, without depth. You could paint a sprite anyway you wanted, but it was still flat 2D. The advent of 3D would make games so much better.

IT’S A POLYGON WORLD

In 1984, a new game made its way to the arcade. It never progressed much beyond it as a mere few hundred machines were produced, but it paved the way for Quake, Kingpin, and all those death matches you’ve fragged around in.

The game was *I, Robot* from Atari. It was somewhat based on the Isaac Asimov story of the same name. In *I, Robot*, you guide a robot around a “world,” looking for and walking onto red squares. Once in contact with the red, the robot lasers a foreboding red eye at the other end of the world. Touch all the red, the eye dies. After a small fly-through-space-shoot-objects game, you reach another world with more red, and another eye. Yeah, it’s simple, but fun. What is more important is that *I, Robot* was the first game to use polygons.

You’ve probably heard of polygons; they’re the latest buzzword in game advertising. “Each world consists of 40 bazillion polygons, all rendered on-the-fly!” Polygons are the key to 3D games and those more “realistic” worlds developers want to create and gamers want to play in.

In 1984, *I, Robot* was the impetus to develop 3D worlds. For a long time, such amazing stuff would stay in the arcade in games like *Hard Drivin’* and *Virtua Fighter*. The processors in home computers couldn’t handle all the necessary computations to draw polygons and what they represented in three-dimensional graphics. However, it was only a matter of time before this would change.

MAKING PROGRAMMING LANGUAGES EASIER

In the meantime, many programmers tried to get away from the opaque complexity of assembly language. Some programmers used other languages, but by and large, assembly language was the most powerful choice. However, for all its power, it was a pain. You couldn’t port a game to a different processor easily. And it was a bit of a

memory hog. Or, at least, assembly language programming handled precious memory resources inefficiently.

In answer to these and other problems, Dennis Ritchie and Brian Kernighan at Bell Labs introduced a “flexible” programming language, C, in the late 1970s. They called it a “high-level” programming language and it quickly became very popular. It took up less memory, was much easier to learn, and was more “human-readable.”

“Human-readable” is not a difficult concept to grasp. Remember our assembly example? Well, some programmers wanted to get beyond those cryptic codes to something humans could “read.”

Here’s an example of things not being “human readable” from the early days of word processing programs. In WordStar for DOS, if you wanted to boldface or italicize text, you had to insert a marker before and after that text. For bold, you pressed CONTROL-P and then CONTROL-B, typed the text, and then pressed CONTROL-P, CONTROL-B again. For italics, you pressed CONTROL-P, CONTROL-Y (yes, “Y”) before and after the text. Here’s how it looked onscreen:

The last three words here are ^B^Ybold and italic^Y^B.

Here’s how it printed out:

The last three words here are ***bold and italic.***

This is also very similar to HTML, or HyperText Markup Language, which is what Web browsers use to display a site’s pages.

```
<FONT COLOR="#000000" FACE="Times New Roman,Times,Times New-Roman"> The last three words here are <B><I>bold and italic</I></B></FONT>
```

These commands tell the Web browser what font (style of letter) to use, and the color of the font. And you can see the and <I> commands for bold and italic before and after the last three words.

The same point applies to programming languages. Assembly language required programmers to remember and use arcane codes. However, with the newest tools for making games, you can “point and click” to get the effects you want.

In many respects, the push toward “human-readable” languages parallels the push toward WYSIWYG (What You See Is What You Get) interfaces. From codes in word processors came buttons that quickly and easily formatted the words in the document and showed text onscreen exactly as it would appear in print.

Home computers were not as plentiful or prevalent as they are now when C first debuted. It found success, but only with tinkerers, hobbyist programmers, and some business folk. It never gained the popularity of its more human-readable second-generation version (called C++), which Bell Labs debuted in the late 1980s.

C++ revolutionized programming and game development in two distinct ways. First, it took advantage of a newly created programming structure called “Object-Oriented Programming,” (OOP). OOP, in a nutshell, takes functions, and the data those functions operate on, and places them in separate, independent structures that float inside a larger house program. This structure is the “object.” Once an object is created, the main program can call the object to perform its function. The data created is then served up to the main program, or even to other objects, which have

their own specific functions. Objects are portable; they can be moved to, and used in, any other C++ program.

By the time C++ came about, computers, especially IBM clones, had become affordable. Consequently, hobbyists and even professional programmers spread C++ objects and code throughout online bulletin boards, and later, the Internet. Any programmer who knew C++ could use these objects. Programmers didn't have to reinvent the wheel every time they wrote a new program. Do you need a routine that creates sounds? If an object for this exists on the Web, it's easy: download, modify a bit, and presto.

C++'s portability exploded beyond anything developers imagined, and brought on the second revolution in game programming. Developers created whole 3D engines for games like Doom, Quake, and Unreal, which they would then sell to other developers to use in other projects. Can't afford to buy a 3D engine? That's cool, because free 3D engines started to appear on the Web along with code for sound cards, objects for polygon calculation, and so much more—all of it nearly plug-and-play.

The ease of use and portability of C++ revolutionized game development, and better hardware support took it to the next level. Sound, calculations, sprite and polygon rendering, player control, collision detection; these are just a few of the things that, just a few years ago, you could only do painfully, in assembly language that would run on only one processor.

GAME CONSOLES SHAPE THE FUTURE

Back in the 1970s, when Atari and others debuted their game consoles, processors were expensive, so consoles such as the 2600 and Intellivision had to rely on less processor power to do all the necessary tasks. In 1984, Nintendo was creating its new video game console: the Family Computer (or Famicom), which later became the Nintendo Entertainment System (NES) in the United States. By this time, chips were more affordable and easier to come by. Therefore, when it created the Famicom, Nintendo gave it several processors, each with a specific task.

The breakdown went like this: the main CPU, the 6502, controlled the larger functions, such as math calculations, floating-point instructions, and system management. Another chip controlled the creation and administration of how graphics appeared on the screen, and what sprites would do when they collided with one another. Yet another chip managed, created, and played sounds. This whole system created a looser, more efficient structure, allowing programs to harness the power of each processor individually. The first Famicom games appeared in Japan in 1985, a year when Atari's 2600 was still king in America. But if you've ever seen a 2600 and Nintendo game side by side, you know that NES games blow 2600 games to smithereens.

A few computers of that day also used separate processors. The Commodore 64 had separate video, audio, and main processing chips. But Nintendo really showed the advantages of this type of system. The word was out: this was the way to better

games. Commodore's Amiga series and Atari's ST series took the model to an extreme by including amazing graphics and audio processors, even while they had mediocre CPUs.

While game console makers flocked to multiprocessor systems, PC manufacturers approached the idea with more caution. For a long time, IBM and the clone-makers thought their customers didn't want graphics. Computers used for businesses didn't need high-powered graphics or polygons or full-on surround-sound stereo. What businesses needed was a "real" computer like a PC, right?

Some third-party manufacturers thought otherwise. Graphics card manufacturers, like Creative, created cards with better video processors, and included their own RAM. The cards took on many time-intensive tasks, and the main CPU became free to do other things. CGA, the first PC graphics standard, turned to EGA, and then to VGA and SuperVGA. Soon, "graphics accelerator cards" like the 3Dfx and ATI series appeared. These cards assisted the video card and gave enhanced performance to graphics-intensive applications (games!). They rendered polygons into the tens of thousands and applied textures for a "real-life" look.

The same push for improvement happened in the sound world, too, although the battle ended early. Two cards, the Ad-Lib and the Sound Blaster®, appeared in the early 1990s. By the middle of the decade, the Sound Blaster was an unofficial standard. Today, it comes in nearly every new PC and creates and plays sound unimaginable a decade ago, full stereo music and effects, sounds that even rival real life. The screeches in Grand Theft Auto sound as if they're outside your door.

For game developers, cards made programming even easier. Each card came with drivers and libraries that could be inserted into new games. A new game could look and sound fantastic right out of the box, with no need to rewrite basic sound and graphics routines. Flat, 2D games with tinny sound gave way to fully rendered 3D worlds filled with music and sounds around every darkened corner. In Unreal, the growls fall from the platforms above. In Quake 3: Arena, you can actually hear the sound of someone getting fragged two rooms away. Quick! Run! Frag the fragger!

THE FUTURE OF GAME DEVELOPMENT

It may seem like we've come to the end of game development's road of progress—but this is by no means a dead end. There will always be room for advancement and improvement. Some developers believe easier programming and game development tools make for worse games. This is not true. Sure, there are many more games out there because they are so easy to crank out, and that ease of creation has caused some poorly done games. But better games are also appearing, because real artists can work on a game and create a great game, not a degraded version of it. Game makers can also focus more on the production values of the game, and not on the technical details.

The typewriter didn't create bad writing, just more of it, both good and bad. In any artistic endeavor, it's the output that is to be judged, not the tools that made it.

Previously, it was necessary to learn the specific language of a processor before trying to write a game for it. Now, with tools like TGF2 or FPS Creator, game makers can create games in a matter of hours. Easier programming tools afford professionals a broader range of talent to pull from, and give amateur developers stronger tools to hone their skills.

Games and game development are becoming more popular due to the increasing ease of entry into the game development field (you no longer have to be a programmer). There is a demand for artists, animators, and designers. As a result, there are larger and more diverse teams working on games.

GAME GENRES

To design and develop computer games, you'll find that, as in most professions, you will need a common vocabulary to communicate with all the people involved in the life of a game title. Among the most important terms is "genre." Genres in computer games, as in movies and books, help the designers form a unified vision; help businesspersons sell the games, and help the audience know what they are getting.

The concept of the genre in computer games starts simply, but gets rather complex. The field of game development has more forces and influences at work on its product in the computer field than in any other medium. In printed fiction, genres started simple, like the thriller, and then branched off into subgenres, such as the "legal thriller" or "psychological thriller." Having subgenres branch off main genres is simple to understand for everyone involved, from the writer to the reader. However, in computer games, many factors create many genre hybrids and combinations. Things are moving so fast that there is barely time to develop a consensus on how genres should be divided and labeled.

In the following sections, we will look at the many genres, subgenres, and hybrids of computer games. You will need to know the genre of your game before you design it, but chances are, if you have an idea for a game, it already fits into one of the genres discussed next. Genre is important at this point, because it will help determine the amount of art, technology, time, and money you will need for your game. And if you plan to get your game published, you will need to be able to quickly and clearly position your title in the publisher's mind by comparing it to other games, and discussing your game in terms of its genre.

Maze Games

Maze games have been around almost longer than any other genre. These are the very familiar games like Pac-Man and Ms. Pac-Man. In maze games, you simply run around a maze, usually eating or gathering something, while being chased by something. Maze games started in 2D with an overhead view of the maze. You can easily make maze games with The Games Factory 2.

Many people don't realize that from a design point of view, the modern high-tech full-blown 3D games are simply a case of the player being brought into the maze. Players still chase, are chased, gather power, and die in a maze.

Board Games

When a traditional board game like Monopoly, Cluedo, or Sorry is recreated on the computer, it still keeps its original genre classification of “board game.” The game usually looks much like the original game, with no innovation in game play, no original use of computer technology other than to make the game function as it does in real life, and usually no artistic improvements on the original game board and pieces. Initially, the challenge of programming enough artificial intelligence for the computer to play the game was enough to keep developers busy, so new innovations in art and game play had to wait.

More recently, board games in the computer world have been moving away from straight copies of their 2D ancestors to newer 3D versions. These newer games sport a 3D look, as the pieces move and have animated cut scenes at highlights or low points (victory and defeat points) of the game. Still, they are not usually innovative; they are simply more lavish productions. Some players and designers argue that this takes away from the game itself, as the animation and videos in many cases slow the game.

Figures 2.4 and 2.5 are pictures of two board games converted to the PC using previous versions of The Games Factory 2 software.

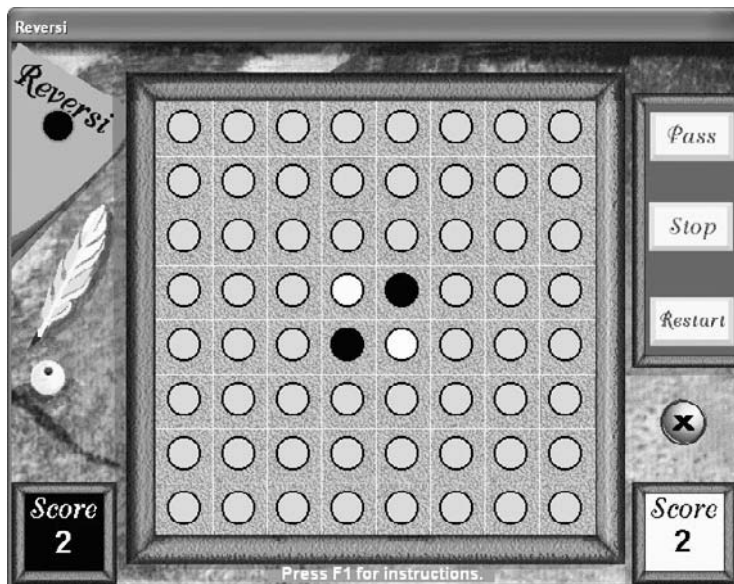


FIGURE 2.4 Reversi board game.

Card Games

Computer card games like Solitaire, Poker, Hearts, and Strip Poker are a huge genre. And, like board games, these titles have so far seen little innovation in most ways.

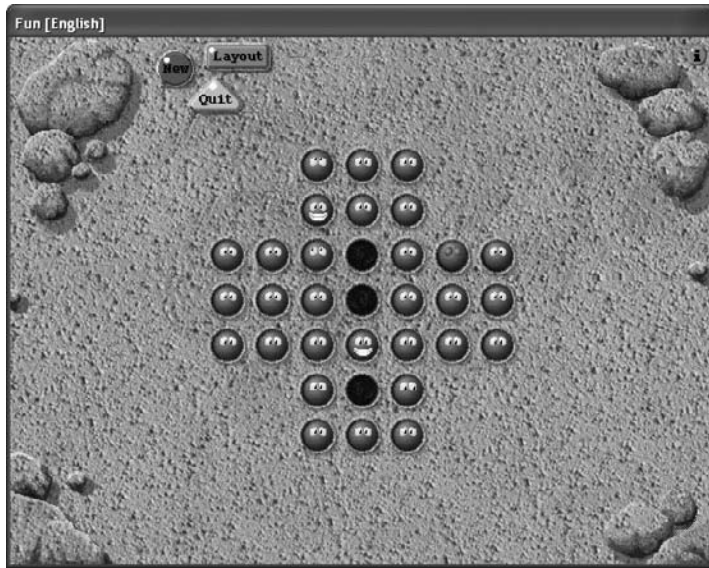


FIGURE 2.5 Solitaire board game.

In some respects, many people who play card games are not after fancy 3D or effects, but are looking for a challenging game they can play now and again. A complex story with video sequences is not what the usual card game player is looking for. Figure 2.6 shows a card game made with The Games Factory 2.



FIGURE 2.6 Blackjack card game.

Battle Card Games

Battle card games came about with the Magic The Gathering craze, which spawned such card games as Spellfire, Legends of the Five Rings, and Pokémon. Battle card games play very much like traditional card games, only with pretty pictures and an emphasis on being collectible. Naturally, the decks are open-ended; if users buy more cards, they become more powerful. Their move to the computer has been much like traditional card games' move to the computer, with little real innovation.

Quiz Games

Quiz games are big, especially online, and TGF2 makes them easy to create. The logic behind these so-called “multiple-choice games” is easy: all the games have to do is display a question and three or four answers. The hard part is researching and organizing all the content, questions, and answers. Figure 2.7 shows a typical quiz game interface.



FIGURE 2.7 A typical quiz game interface.

Puzzle Games

Puzzle games include Tetris, Dr. Mario, and others. Usually, there are pieces falling from above, which players have to line up before they hit bottom. The player must fit them all together in the most efficient manner, to leave no open spaces between the pieces. The pieces become more complex and fall faster as the game progresses.

Shoot 'em Ups

Space Invaders, Asteroids, Sinistar, Space Battle, and the original Spacewar are examples of this genre (later in the book you will make your own shoot 'em up game). These are the 2D games where you are in a ship in space and you shoot things before they hit you—aliens, missiles, and such. An example of a shoot 'em up game can be seen in Figure 2.8.

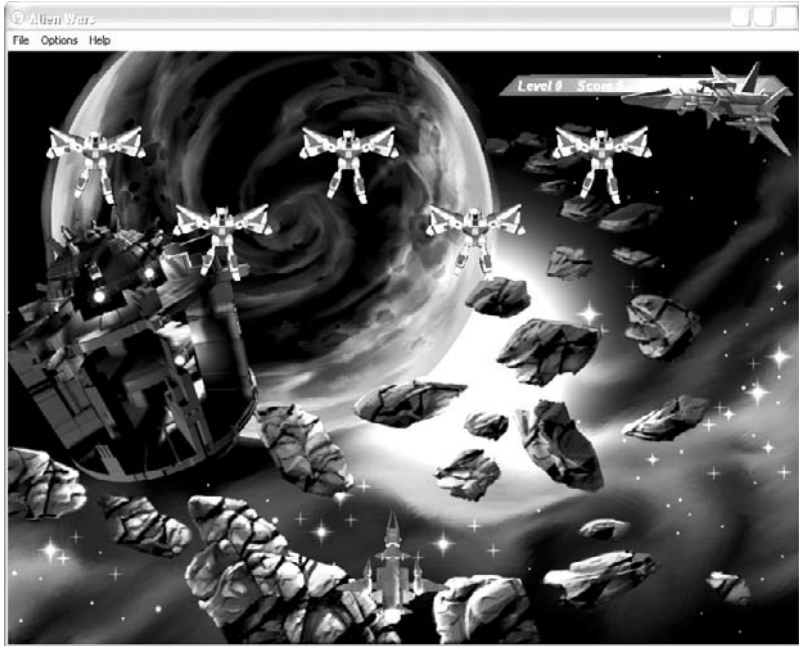


FIGURE 2.8 A shoot 'em up game made in TGF2.

Side Scrollers

Side scrollers are what made id software big. Remember Commander Keen in the "Invasion of the Vorticons?" The original Duke Nukem', Prince of Persia 1, and Zeb are also examples. Zeb was made with the original version of The Games Factory. Side scrollers usually have the hero running along platforms, jumping from one to the next, while trying not to fall into lava or get hit by projectiles. A typical side-scroller interface is shown in Figure 2.9.

Fighting Games

There are many fighting games; examples include Street Fighter 2, Samurai Show-down, Martial Champion, Virtua Fighter, Killer Instinct, Battle Arena Toshinden,



FIGURE 2.9 Jack in the Forest side-scrolling game.

Smash Brothers, and Kung Fu (see Figure 2.10). Fighting games started as flat 2D interfaces and now feature full 3D arenas and animated characters. The focus in a fighting game is the almost endless fighting moves and special moves you can use against your opponent.



FIGURE 2.10 The Kung Fu game is a fighting game made with the original Games Factory software.

Racing Games

Racing games center on the concept of driving fast around different tracks. Wipeout, Destruction Derby, Mario Kart, and South Park Derby, to name a few, are all racing games. Some 2D racing games have a scrolling road and the sprite of the car moving over the surface. One of the most popular 2D racing car games is Micro Machines, where you race mini cars over interesting backgrounds such as a table and outside.

You can see an example racing car game made with The Games Factory in Figure 2.11, and a 3D racing game called RalliSport Challenge by Microsoft Games in Figure 2.12.

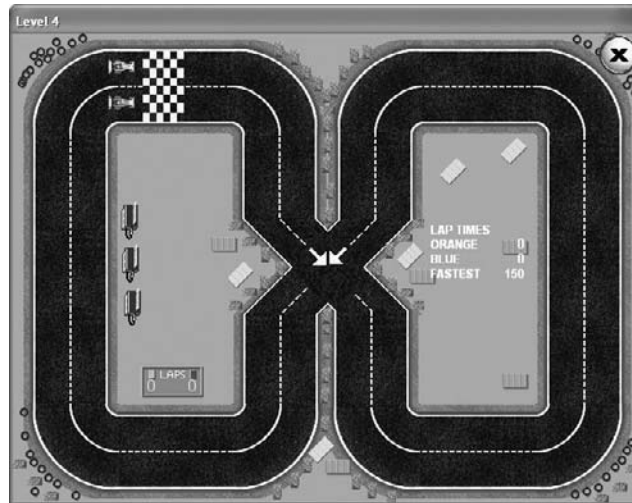


FIGURE 2.11 Racing line game.



FIGURE 2.12 3D racing game called RalliSport Challenge.

Flight Sims

A flight simulator (sim) attempts to simulate real flying conditions by giving you control over such things as fuel, wind speed, and other instruments, and control over the flaps and wings of your craft. A sim will respond with the same limits as a real plane, as opposed to a simpler flying game, where you can't control much. Wing Commander, X-Wing, and Microsoft Flight Simulator are all flight sims. A screen shot from a flight sim is shown in Figure 2.13.



FIGURE 2.13 Microsoft Flight Simulator X.

Turn-Based Strategy Games

In games such as Breach, Paladin, Empire, Civilization, Stellar Conflict, and Masters of Orion, players take turns making moves. These games require a great deal of strategic thought and planning, much like chess.

Real-Time Strategy Games

Command and Conquer 3, Age of Empires, Supreme Commander, and Caesar 4 are a few popular real-time strategy games. In these games, you don't have forever to take your turn before the next person moves. Faster players can make many moves in a short period of time. These games are also a bit like sims, since you are usually overseeing a large battle or war, and the building of towns and outposts. Resource management is important such as in Command and Conquer, where you have to

harvest (mine) ore called tiberium to be able to build more structures and soldiers. Two of this author's all-time favorite strategy games are CivCity Rome and Stronghold shown in Figures 2.14 and 2.15, respectively.



FIGURE 2.14 A popular real-time strategy game, CivCity Rome.



FIGURE 2.15 Castle building strategy game—Stronghold.

Sims

Sim City, Sim Earth, Sim Ant, and the most popular of the simulation games—The Sims. In these games, you run a simulation of a town, world, ant colony, or a group of people, making decisions and managing resources. These are often called “God games,” because you are playing the part of God in the game world.



In the discussion that follows, the terms first person and third person refer to the point of view of the player. Just as in literature, where you can write in first person (“I shot the rocket”) or third person (“she shot the rocket”), there are points of view in gaming as well.

First-Person Shooter (FPS) 3D Games

These games include Dark Messiah, Half Life 2, Halo, Doom 3, Quake, Far Cry, and Elder Scrolls IV Oblivion. The focus in these games is on technology and atmosphere. These games attempt to put you into the action, as you are literally looking out of the eyes of the character, seeing and hearing what the character sees or hears. As shown in Figure 2.16, the point of view is from a person on the street.



FIGURE 2.16 Screen shot from Halo 2, a first-person 3D game.

First-Person 3D Vehicle-Based Games

These games are very much like the first-person shooter games, except that they put you in a vehicle, such as a tank, ship, or giant robot. This genre is more similar to an FPS shooter game than a racing game, because you are not simply driving as fast as you can to cross a finish line. Your goals are more similar to the ones in an FPS game—

kill or be killed. In most cases, the vehicle-based games are part of an FPS game. For example, in *Battlefield 2142* you look out from your character in first-person mode, but also have the ability to jump into a plane or tank to attack the enemy.

Third-Person 3D Games

Tomb Raider, *Dark Vengeance*, *Zelda*, and *Fable* are all third-person games. Although there are games in which you can switch from first- to third-person perspective, most are designed primarily to be one or the other. *Tomb Raider* in first person is not as much fun, since it is designed around seeing Lara Croft jump, roll, and tumble. In first person, you would not see these acrobatics. Figure 2.17 shows a third-person game. Notice how you can see the spell effects you cast (the protection circle) when in third-person mode. Likewise, when playing a first-person shooter, like *Quake 3 Arena*, you depend on speed and accuracy in battle to win, which is the point of the game. If you were able to play *Quake 3* in third-person mode, you would die an awful lot since you would not be able to run, aim, shoot, and run some more as quickly.



FIGURE 2.17 Screen shot from *Fable* in third-person mode.

RPGs (Role-Playing Games)

Ultima, *Neverwinter Nights*, and *Dungeon Siege* are all RPGs. These games emulate the traditional pen-and-paper games in which you play characters that have many significant attributes, such as health, intelligence, strength, and areas of knowledge and skill. RPGs are like a simulation of an adventure.

Adventure Games

Broken Sword, CSI: Dark Motives, Sherlock Holmes and the Case of the Missing Earring, and Runaway 2: the Dream of the Turtle are all adventure games. In an adventure game, you walk around and try to fulfill a quest or unravel a mystery. You typically collect information and items. Battle is light and not the focus of this game type. The game is usually played using the mouse to point and click.

Full-Motion Video Games (FMVs)

FMV, or full-motion video, games include MYST, RIVEN, and . . . well, no other FMV game is worth mentioning. These games require a lot of art and animation or video production, and little of anything else. There is simply no room for effects, because FMV is a limiting genre at present. In an FMV, you mostly watch a movie and then select what portion of the movie to watch next, much like a computerized version of the “choose your own adventure” books.

Educational and Edutainment Games

Some games or interactive products fall into this genre. Whether a game fits into this category depends mostly on its purpose, rather than on its content or use of technology. A first-person game would be an edutainment title if its intention were to educate and entertain, as would a quiz game. These genres are instructional and informative. The edutainment variety attempts to make learning fun, while the educational variety is straightforward learning. An example of an educational game can be seen in Figure 2.18.



FIGURE 2.18 Magic Maths educational game.

Sports Games

Sports is a huge-selling genre all by itself, but also another genre label that doesn't completely convey the technology, game play, interface, or other aspects of the game. In fiction, a thriller that takes place at a football game may be called a "sports thriller." An inspirational nonfiction book with a sports theme may be called "self-help/sports." But in games, people often don't say "quiz game/sports" or "quiz game/football" or "third-person football simulation"—everything is lumped under "sports." You will find most popular sports are covered in the sports genre, including football, American football, basketball, ice hockey, and horse riding.

Screen Savers/Desktop Toys

While not games, and not even very interactive for that matter, these products are generally entertaining, so they are usually lumped in with games and interactive products. These are fairly popular products you can make with The Games Factory 2, like the screen saver in Figure 2.19, which was made for downloading a trial version and purchasing the full version for a small fee.



FIGURE 2.19 Micro Animals Fish screensaver.

Genre Madness

Even with all the genres just discussed, many games cross over and combine the genres. Generally, a good game in one genre will have elements of other genres, such as puzzle solving in a 3D game. Breakouts into new genres often occur where technology permits. For example, many fighting games started out as side scrollers for the 2D platform and evolved into 3D shooters or 3D games. Another example of a game traversing genres is Grand Theft Auto. In the first version, the game was a top-down 2D game, where the player could control a person and jump into cars and other vehicles. In the second version, the game became 3D, allowing the player to drive in a 3D world. Later releases brought the ability to play arcade machines and billiards within the game.

CHAPTER SUMMARY

When designing your title, keep genre in mind. It is the first step in clearly communicating your vision to all involved. Once you have a clear idea of your game (“it’s a first-person adventure game with shades of military simulation”), you can describe it in visual terms on paper, and then break it down into the elements that will comprise the design document. In the next chapter, we’ll look at the elements of design.

GRAPHICS: THE BASIC BUILDING BLOCKS OF A GAME

In This Chapter

- Sights
- Basic Elements of an Image
- Manipulating Images
- Advanced Image Manipulation



To create games, you will need to learn, and perhaps even master, the fundamental elements that make up a game—sights, sounds, and interactivity. Although interactivity (the ability to interact with a computer to play a game) is important, the basics of this interactivity depend on the game type and the application you are using to develop the game. You will learn more about interactivity in the tutorials later in the book, as you make several different types of games, using various tools. This chapter discusses the core building blocks that exist in virtually every game’s sights and sounds.

This simple approach will help you break down and understand a game at its most fundamental level. You can apply this knowledge to many areas beyond game development as well, since it is the core of graphic design, Web layout, and almost all interactive computing.

SIGHTS

When talking about sights, we are obviously talking about what you see on the screen during game play. In any major production, from a Web site to a game, the layout of the screens and the graphic images to make them are very important. In a large development team, a number of people—including a designer, producer, art director, and others—usually work on them. In a one- or two-person development effort, you will need to wear several hats and perform all of these roles. Your 2D art assets need to look good, and fit in with the audience, technology, and atmosphere for which you are designing. We’ll talk about this again later when we look at marketing a game.

Creating the assets you will use to make interface elements requires the use of many software tools and techniques. These assets are often first sketched on paper or mocked up on the computer. Some of the tools you can use are 2D paint programs that work only with flat images, 3D programs that allow you to build and render objects that realistically recreate a 3D environment or object, and even digital photographs and scans. To create the images, you will need to have an understanding of the concepts of the images, and a grasp of the tools you will be using.

The 2D art assets you will create include, but are not limited to:

Menu screens. Look at the toolbar in your word processor, browser, or your favorite game, and you will see art that was created by an artist.

Credit screens. These screens often contain art such as logos, images, and even fonts or special letters unique to the product, people, and company they represent.

Logos for companies, products, and services. Logos can be simple letters, 2D masterpieces, or fully rendered 3D scenes. Look on the Web and you will see logos that range from clip art to actual pieces of art.

User interfaces. These are broken down into background images, buttons, cursors, and other art objects a user must click or interact with.

In-game assets. These include the textures on the walls, the floors, and the characters. Even the 3D models and objects have 2D art applied to them.

Early computers did not display graphics; they were limited to alphanumeric characters, such as letters, punctuation marks, and numbers. Surprisingly, games were still made on these primitive machines. When computers started including graphics cards, games started their move toward the amazing graphics we see today. It can be argued that games have pushed the development of the computer as gamers demanded (and were willing to pay for) faster chips, better graphics cards, and better sound. However, even as the technology advanced, it was common for the artist on any given project to primarily be a programmer. This was because it was still very difficult to get decent art into a computer format, and an understanding of technology was necessary to do so. Today, we can almost ignore the technology we are working with.

Let's look at the core technology a computer artist deals with every day. In computer graphics today, there are two basic types of art: 2D and 3D. All 2D, or two-dimensional, art is a flat image with no depth; 3D, or three-dimensional, art shows depth, as illustrated in Figure 3.1.

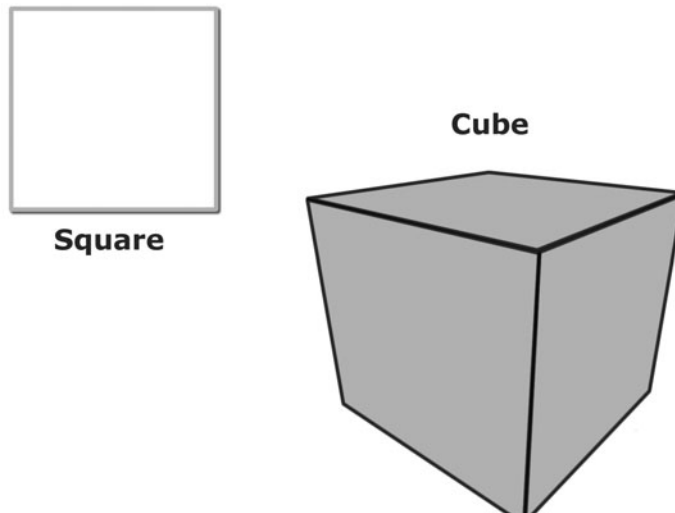


FIGURE 3.1 A square is 2D, while a cube is 3D.

The three dimensions in 3D art are described in the Cartesian coordinate system, using x , y , and z coordinates. This may be one of the most surprising aspects of game development; you can actually use some of the math topics you learned in school! In fact, algebra, geometry, and physics all play a role in game making. Simply stated, in the Cartesian coordinate system, x represents the distance along a horizontal line (or axis), y represents the distance along a vertical line, and z represents the distance backward and forward (see Figures 3.2, 3.3, and 3.4).

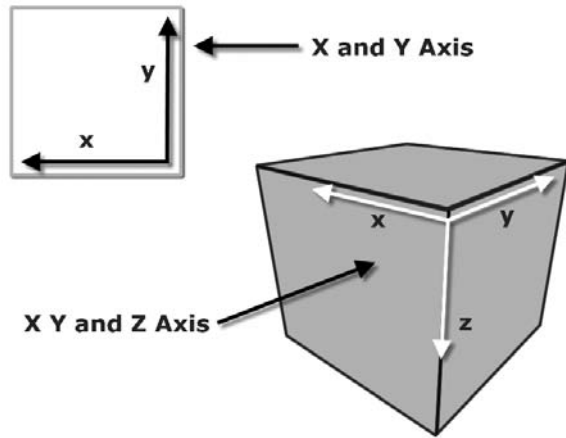


FIGURE 3.2 The Cartesian coordinate system. The x, y, and z axis.

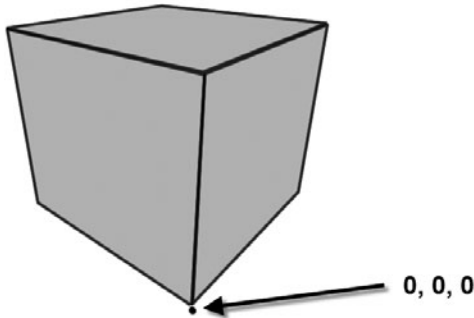


FIGURE 3.3 A cube and the xyz value of its location in space.

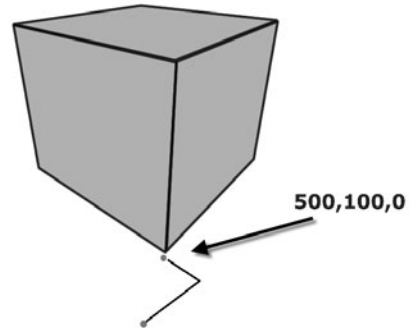


FIGURE 3.4 Another cube in a different xyz position.

BASIC ELEMENTS OF AN IMAGE

To properly understand 2D images, you must understand a few things about the basic elements that compose those images, as discussed next.

Pixel

We'll start with the most fundamental of fundamentals, the most basic element of an image—the *pixel*, or picture element. A pixel is a colored dot on the screen. A computer image is made up of pixels arranged in rows and columns. See Figure 3.5 for an illustration of a pixel. No matter how big and fancy a computer image is or what has been done to it, it's all just a bunch of pixels.

Once an image has been created with a particular number of pixels, the maximum detail is set and cannot be increased. The image can be enlarged and the number of pixels can be increased by a mathematical process called *interpolation*, as illustrated in Figures 3.6 and 3.7. However, this does not increase the detail; it simply adds extra pixels to smooth the transition between the original pixels.

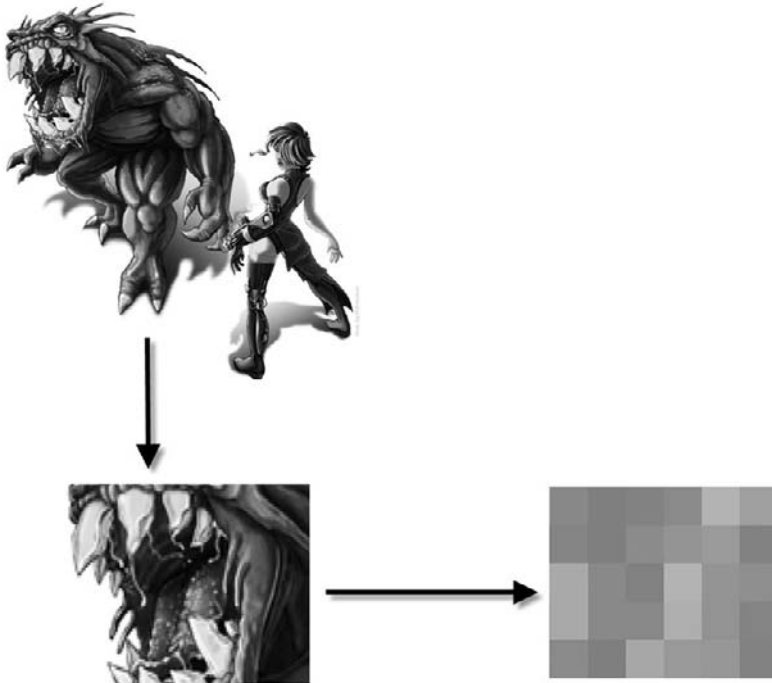


FIGURE 3.5 A pixel is the smallest unit of a computer image—simply colored dots.



FIGURE 3.6 An area of the image before enlarging.



FIGURE 3.7 The same area enlarged with pixels interpolated.

Resolution

Resolution is the number of pixels displayed (width \times height) in an image. A typical computer monitor displays 75 to 90 dpi (dots per inch, which refers to the number of pixels per inch in an image). A printed image usually needs to be 300 dpi or more to look good in print. Often, when computer people receive an image from a person who is used to working in print, they are surprised when the one-inch icon they requested takes up a *huge* number of bytes, but the image is still one inch by one inch. The reason for the enormous size is that a print person is used to using, and saving, images at a higher dpi. Some of the most common screen resolutions are 800×600 , 1024×768 , 1152×864 , and 1280×1024 . An 800×600 resolution means that the screen is 800 pixels wide (horizontal) and 600 pixels high (vertical). See the examples in Figures 3.8, 3.9, and 3. 10.

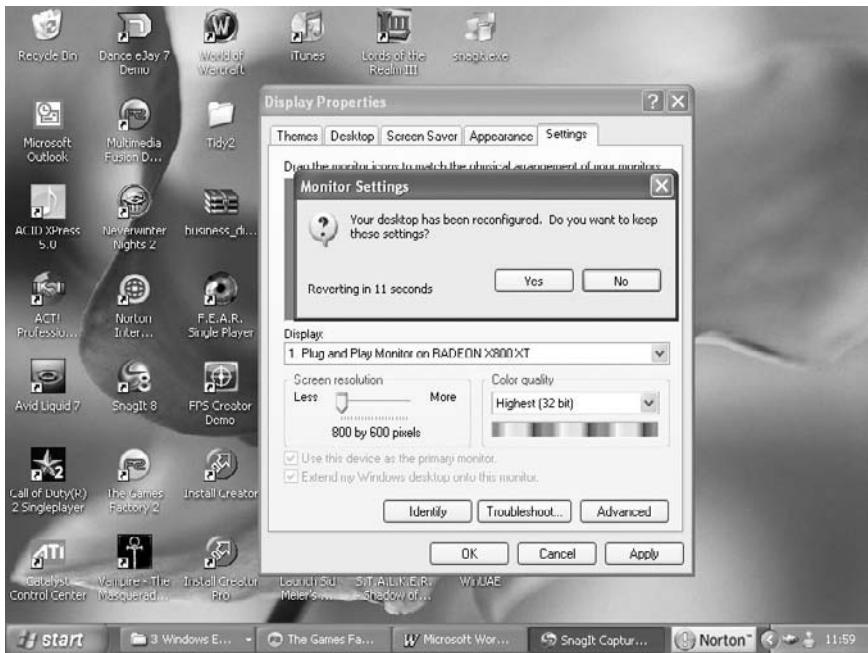


FIGURE 3.8 The Windows desktop at 800×600 dots per inch.

Aspect Ratio

Until recently, most computer monitors displayed in an aspect ratio of 4:3, also known as 1.33:1 format. If you take a standard computer monitor size of 800×600 and divide its height by its width, you get 1.33. The popularity of widescreen television sets has begun to revolutionize the computer monitor market and now a 16:9

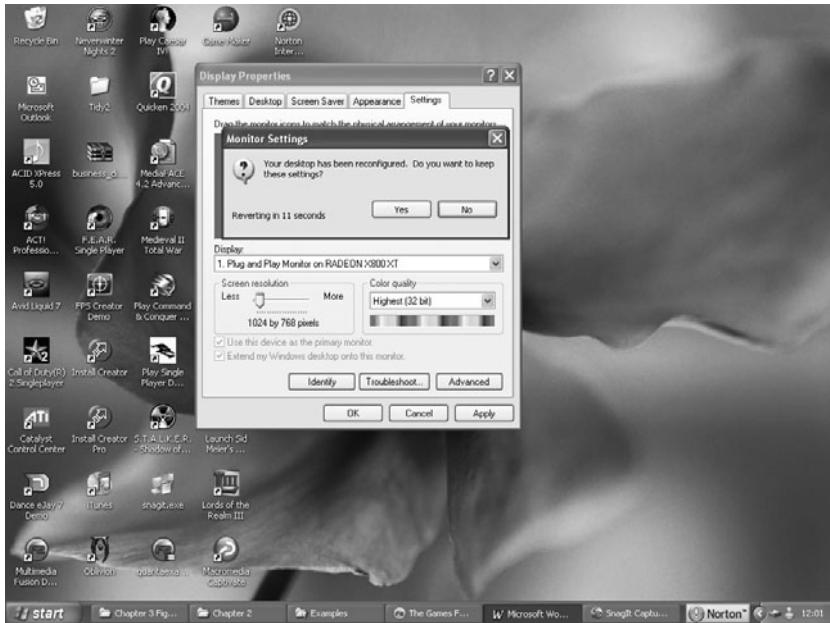


FIGURE 3.9 The Windows desktop at 1024 × 768 dots per inch.

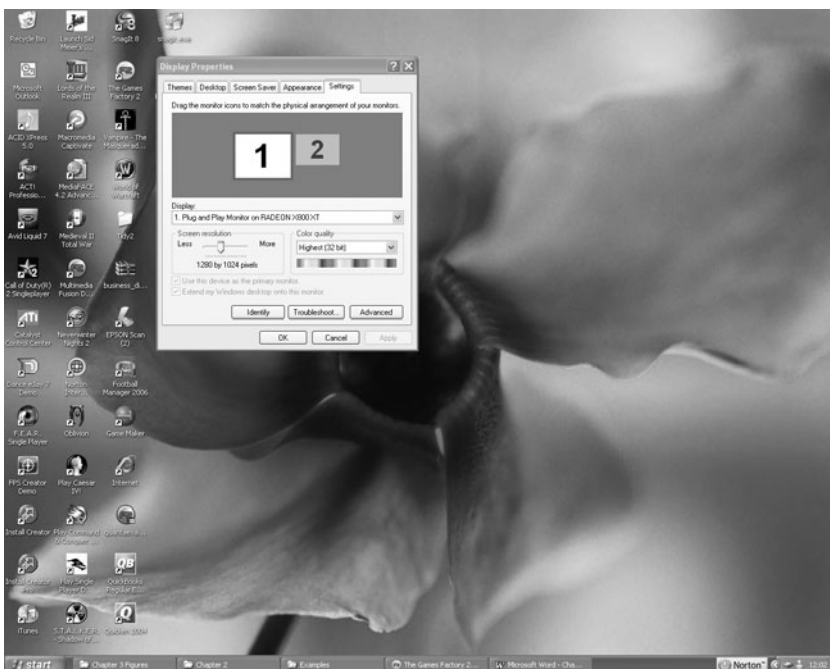


FIGURE 3.10 The Windows desktop at 1280 × 1024 dots per inch.

ratio has started to appear. This means that a large number of display resolutions and monitor sizes are available for computer users.

When working with widescreen aspect ratio, ensure any graphics will look correct on a 4:3-based monitor. A wide screen monitor will stretch any images across the monitor. This can mean that you are creating an image that may not look correct on a standard monitor and aspect ratio (see Figure 3.11).

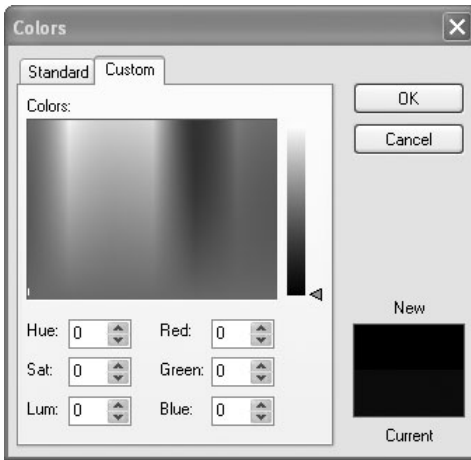
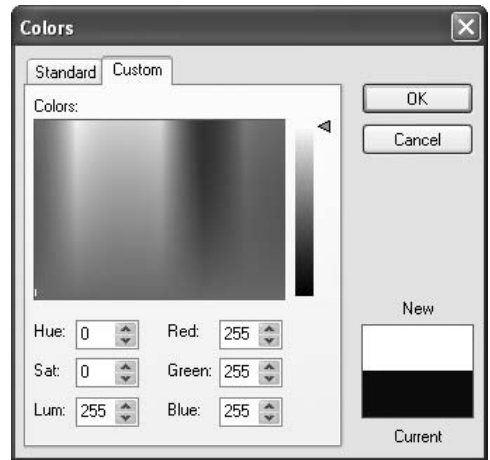
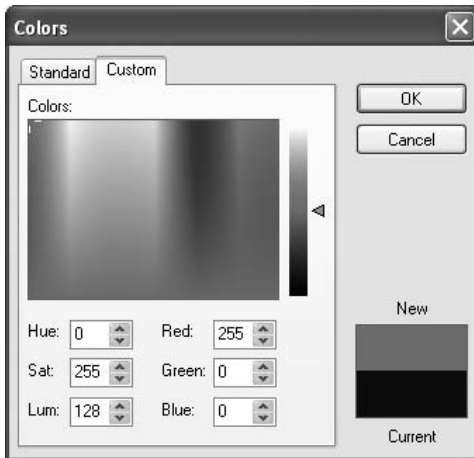
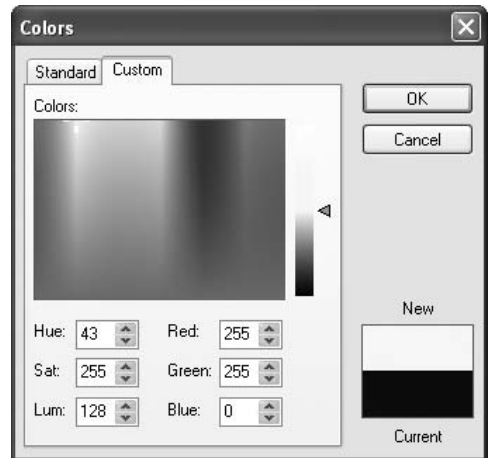


FIGURE 3.11 Two images created in two different ratios.

Colors

When working with most interactive content, you need to understand how color works in the computer. In some situations, such as games and Web sites, you will need precise control over your colors to achieve the effects you want. Colors are usually specified as RGB values, and artists may sometimes give you the specific value to use for a color in an image. An RGB value (also known as the indexed color value) is the mixture of Red, Green, and Blue to make other colors, just as in art class when you mixed red and yellow paint to make orange.

The first number represents Red, the second represents Green, and the third represents Blue. These values range from 0 to 255. For example, 255,0,0 means you have all Red and no Green or Blue; Black would be 0,0,0 (no colors at all); and white would be 255,255,255 (all colors at their highest intensity). In Figures 3.12 through 3.16, you can see the RGB values of the color, and even though the images are in black and white, you can see the position of the marker in the color palette.

**FIGURE 3.12** The RGB color palette for black.**FIGURE 3.13** The RGB color palette for white.**FIGURE 3.14** The RGB color palette for red.**FIGURE 3.15** The RGB color palette for yellow.

You will also hear color referred to as CMYK. CMYK is a mode used by traditional printing processes and stands for Cyan, Magenta, Yellow, and Black. You'll almost certainly never use CMYK color in game and computer content creation—you'll always deal in RGB or indexed color.

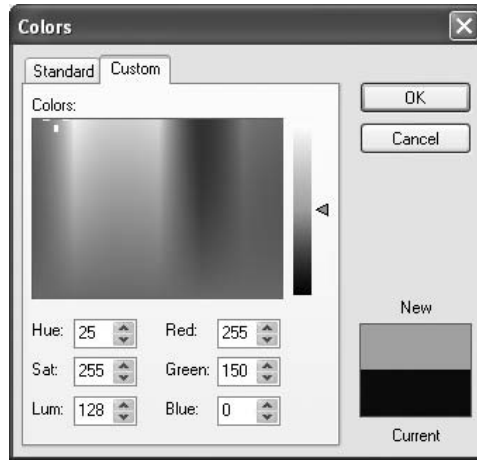


FIGURE 3.16 The RGB color palette for orange.

Number of Colors

A computer video card can display a certain number of colors at a time—16 or 256 at the low end, or even thousands, or millions at the high end (see Figures 3.17, 3.18, 3.19, and 3.20). *Color depth* describes how many colors your screen can display at once, in terms of *bits*, and refers to the amount of memory used to represent a single pixel. The most common values are 8-bit, 16-bit, 24-bit, and 32-bit color; the more bits, the wider the range of displayable colors.



FIGURE 3.17 An image in eight colors. See the images located in the color figures folders on the companion DVD to see a color version of this image.



FIGURE 3.18 An image in 16 colors. See the images located in the color figures folders on the companion DVD to see a color version of this image.





FIGURE 3.19 An image in 256 colors. See the images located in the color figures folders on the companion DVD to see a color version of this image.



FIGURE 3.20 An image in millions of colors. See the images located in the color figures folders on the companion DVD to see a color version of this image.



The interesting thing about these four images is that the colors used in the images will determine the number of pixels required. Visibly there is no difference between Figures 3.19 and 3.20, but there is about a 3.5 MB difference in file size.

True Color (24-bit color) can display about 16.8 million colors for each pixel on the screen. The human eye cannot distinguish the difference between that many colors. High Color (16-bit color) displays between 32,000 and 64,000 colors. However, this is still a very impressive range of colors, and enough for most work. The 256 Color setting is more limited. It stores its color information in a palette. Each palette can be set to contain any of thousands or millions of different color values, but the screen can't show more than 256 different colors at once.

Very few games still use this more limited palette because, as with increased resolution, having more colors means more data must be pumped to the screen. Therefore, if you can get away with only 256 colors, you can render (or draw) the game pictures to the screen faster. Most games use thousands or millions of colors, as the hardware permits.



The word render is used in games, especially real-time 3D games, because the computer and software literally render or build an image instantly, based on where a user is in the 3D world—hence the term interactive. In a movie, you watch a series of unchangeable frames as the moviemaker created them. However, when you play a 3D game, you control how each frame looks by where you choose to go in the world and what you do. Each frame of your gaming experience is made for you “on the fly” or as your experience is happening.

MANIPULATING IMAGES

During the development of your project, you will have to manipulate images to get them to fit your needs. The basics of image manipulation are similar to the text editing

you may have done in your word processor. Commands such as Cut, Copy, and Paste are common. We will also look at Skew, Rotate, Resize, Crop, and Flip.

Cut. If you cut an image, you remove it from the scene, as shown in Figure 3.21. But don't worry, you can paste it back or undo your action.

Copy. Copy does not alter your image; it creates a copy in the memory of your computer that you can paste somewhere else, as shown in Figure 3.21.



FIGURE 3.21 Cutting and copying sections of an image. Copying does not affect the image.

Paste. As mentioned previously, after cutting or copying an image, you can paste it somewhere else, as shown in Figure 3.22.

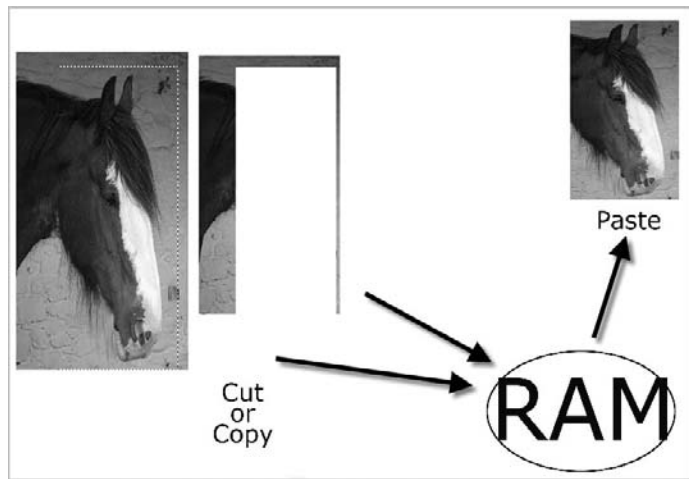


FIGURE 3.22 Pasting a section of an image.

Skew. Some image manipulation programs allow you to skew (slant, deform, or distort) an image, as shown in Figure 3.23.



FIGURE 3.23 Skewing an image.

Rotate. Rotating is self-explanatory; you can free rotate an image or rotate it precisely a certain amount, as shown in Figure 3.24.



FIGURE 3.24 Rotating an image.

Resize. Resizing an image is useful, but be careful. Any severe manipulation of an image degrades it, and resizing can do a lot of damage. Caution: If you reduce an image and then enlarge it again, you will seriously degrade it. This is because, in effect, you are enlarging a small image. The degradation takes place when you reduce an image, and when you enlarge it (see Figures 3.25, 3.26, and 3.27).

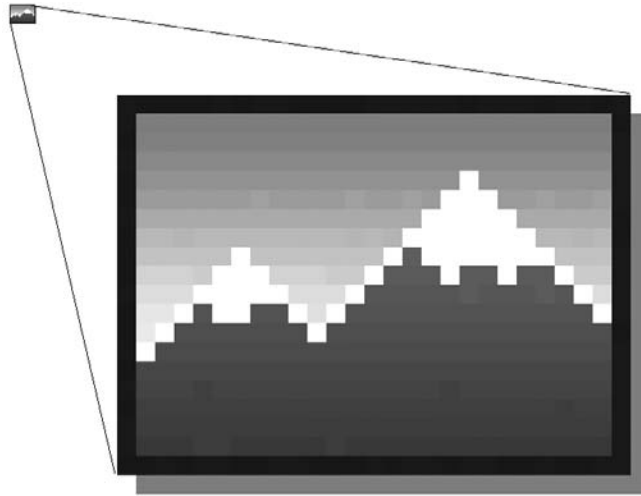


FIGURE 3.25 A smaller image blown up; pixel rip.



FIGURE 3.26 An image reduced.

Crop. Cropping cuts an image to a smaller area you define, as shown in Figures 3.28 and 3.29.



FIGURE 3.27 The same image enlarged to its original size. Notice what this has done.



FIGURE 3.28 Cropping an image. The crop outline.



FIGURE 3.29 The image cropped. Everything outside the crop outline is now gone.

Flip (horizontal and vertical). You can flip images horizontally and vertically (see Figures 3.30, 3.31, and 3.32).



FIGURE 3.30 The image.



FIGURE 3.31 The image flipped horizontally.



FIGURE 3.32 The image flipped vertically.

ADVANCED IMAGE MANIPULATION

In the last section, we looked at some basic image editing operations, and only scratched the surface of what you'll need to do to create graphics for a game. Here are some more advanced operations.

Sprites

A sprite is a graphic image that can move within a larger image. In your games, these might be characters, buttons, and other items. Notice that the sprite image in Figure 3.33 has a single color border around it, and in Figure 3.34, the surrounding color part is not seen.

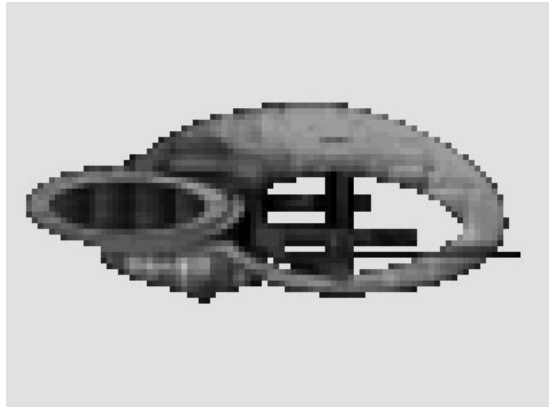


FIGURE 3.33 A sprite image. Notice the solid color part surrounding the image.



FIGURE 3.34 A sprite image in a game. Notice that the solid part is not displayed; you can see the background.

Sprite animation is done just like cartoon animation. A series of images is played in sequence to make it appear that a character is walking or a logo is spinning, for instance. Figures 3.35 and 3.36 show examples of sprite frames.

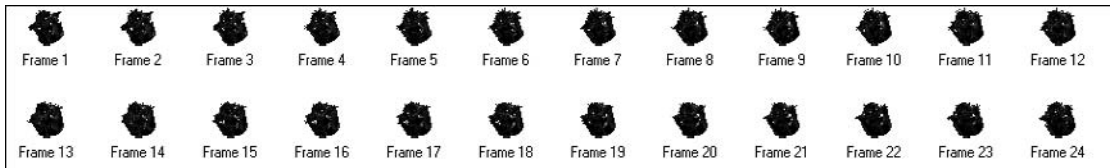


FIGURE 3.35 A series of sprite images for a game animation.

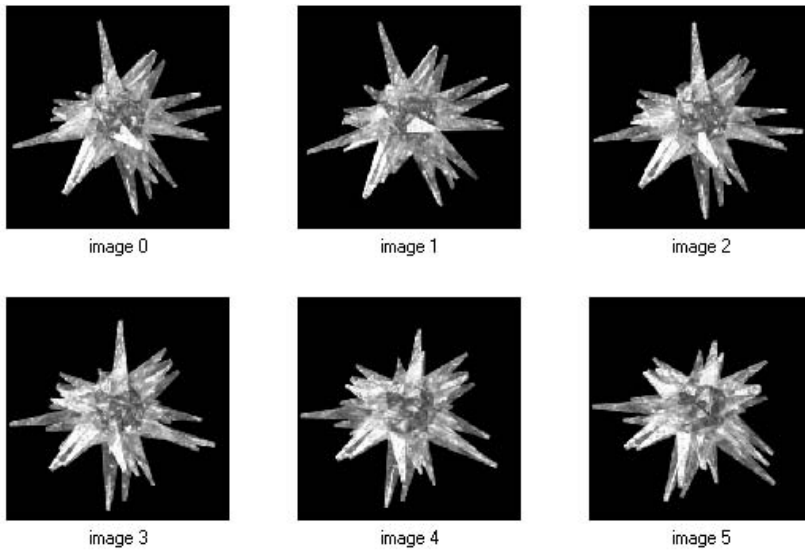


FIGURE 3.36 A series of sprite images for a spinning graphic object.

Masking

A mask is a special image that is used to “mask” off portions of another image. A mask works like a stencil—it lets you paste a nonrectangular image into another image. When you paste a mask into another image, it overlays whatever was in the image at the spot where the mask is pasted (see Figures 3.37, 3.38, and 3.39).

Color Masking

You can also use masking to specify that a specific color should be rendered as clear or transparent. Game programmers usually choose something like an ugly green or purple they most likely will not use anywhere else in the game art.



FIGURE 3.37 An image of a card dealer.

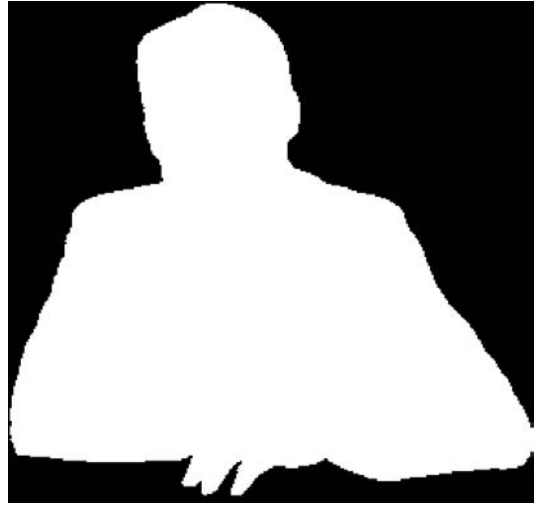


FIGURE 3.38 The mask for the card dealer image.



FIGURE 3.39 The mask and image combined in a scene.

Palette or Positional Masking

The last kind of masking lets you use a specific position on the color palette to determine what color to render as clear or transparent. Remember, the computer sees color numbers, not the colors themselves. In this kind of masking, the computer looks at the position on the palette, not the color, to determine transparency. Usually,

the last color place on the palette is used. Whenever the color with that number is called for, the computer will render it as clear, rather than using that color.

Opacity

You can also display images in games as opaque halfway between solid and clear (like our ghost image). To determine opacity, the computer looks at each pixel in the image and at the pixel directly under it. It then creates a new pixel that is a blended value of the original pixels (see Figures 3.40 and 3.41).



FIGURE 3.40 The masked card dealer image with opacity set at 50%.



FIGURE 3.41 A close-up detail of the image.

Anti-Aliasing

Look closely at the computer-generated images in Figures 3.42, 3.43, and 3.44. See those jagged edges on the letters? They look jagged, as if they are all a solid color. However, by using various shades of a color and gradually blending the edge color with the background color, the computer can make the transition smooth and fool the eye from a distance. Yes, this is similar to opacity.

This technique, *anti-aliasing*, is one of the reasons why images with more colors look better. With more colors, you can blend them more gradually. This is also the reason why using a higher resolution (more pixels) makes an image look better—the blending is smoother between pixels.



FIGURE 3.42 This image has no anti-aliasing.



FIGURE 3.43 This image has anti-aliasing.



FIGURE 3.44 Here is a close-up of both of the image's edges.

Graphic Formats

Graphic images are stored in many different formats, for many reasons. In business, this may be for technical support, product design, and for competitive and security reasons. However, the main reason is image quality and usefulness. Some image formats produce very large files, because they retain a lot of image data, while some formats can compress an image and strip out data for a smaller file size. Still other formats degrade images (in an acceptable way) so they can be very small, for uses such as Web sites. Figures 3.45 and 3.46 show two versions of an image. The degradation is not that bad (see Figure 3.47), considering that the file size of the BMP image is almost 20 times the file size of the JPG image.



FIGURE 3.45 This 640 × 480 image is in BMP format and is 900K.



FIGURE 3.46 This 640 × 480 image is a compressed JPEG and is only 68K.

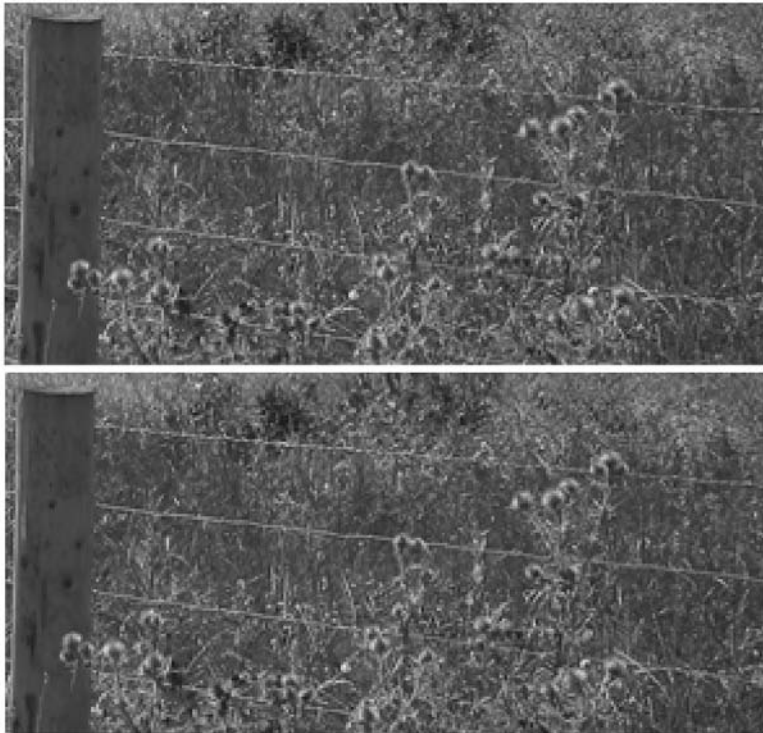


FIGURE 3.47 A close-up of the same area of both images.

CHAPTER SUMMARY

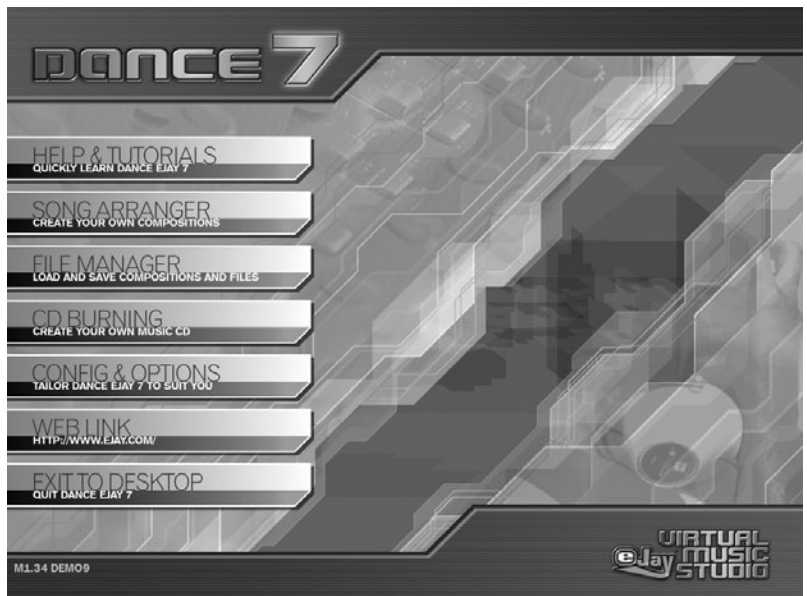
This chapter looked at the basic elements of images and how they are created. Now that you are familiar with graphics, you are almost ready to start creating content for a game. In the next chapter, you'll learn how to create music and sound effects.

This page intentionally left blank

SOUND AND MUSIC

In This Chapter

- Why Sound and Music Are Important
- Types of Sound
- Obtaining or Creating Sounds and Music
- Recording Sounds
- Creating Music
- ACID XPress
- Dance eJay 7



Of the many components that go into making a video game, perhaps none gets less attention than music and sound effects. Adding quality music and sound effects is one of the best ways to add production value to your games. A tremendous array of software and low-cost hardware is available to aid single developers or small teams in this process. Believable sound effects and music will greatly enhance the game player's emotional experience.

WHY SOUND AND MUSIC ARE IMPORTANT

There are many parallels between making a movie and developing a game. Hollywood has long realized the benefits of music and sound effects to the moviegoer. Over the past decade, filmmakers have spent a tremendous amount of time and resources on improving these aspects of a movie. During that time, we have seen the use of surround sound in both theatrical and home movie releases.

The long and varied history of the movie industry offers a tremendous amount of guidance. While you will find very little documentation on the creation of music and sound effects for games, there is a great deal of information available for the moviemaker, both professional and amateur. Many books have been written over the years, and numerous resources are available at Web sites, not to mention in the movies themselves, which can often provide inspiration and ideas.

As the music and sound of video games improves, the video game industry has started to receive recognition for its work. Beginning with the 42nd Annual Grammy Awards, the NARAS (National Academy of Recording Arts and Sciences) approved three new award categories, including music written for "Other Visual Media," the term they are using to include the music from games.

TYPES OF SOUNDS

What you hear in a game can range from recorded (or sampled) sounds such as voices and music; menu sounds like beeps and button clicks; and other effects, such as explosions, weapons fire, footsteps, and a long list of other in-game sound effects, both subtle and deafening.

Sound and music can be very important to a game, for atmospheric reasons alone. With the lights off and the sound turned up, players can really become immersed in a game. Sound and music can set the mood, and change it (think of the *Indiana Jones* score or Darth Vader theme). Among the many cues it gives, sound can clue players in about the threat of enemies. It greatly influences the level of satisfaction they get from the game, and can deliver a strong message to players about the quality of your game, and your company.

And sound is more immersive than graphics. Let's say that again. Sound is more immersive than graphics. While graphics will draw players into a scene, the sound going on in the background and all around has a mental effect on a player that graphics alone cannot achieve. This is probably because real-life sounds can be re-

produced on a computer much better than real-life visuals can. A dinosaur's roar can sound real on a computer, but a visual of a dinosaur doesn't look nearly as real.

Visually, players are looking into another world through a tiny window, and can feel safe from that world. But when they hear that world all around them, they really feel they are in it.

A good example of sound effectiveness is the movie *Jaws*. Who can forget the sound that announces that the shark is coming? In gaming, the sounds in *Trespasser* are incredible. They make the game a terribly tense and scary experience. One of the authors of this book actually had to turn the sound off, to avoid scaring his dog! But without the sound, the game is laughable. It looks just like that, a game. Puppet-like raptors stumble about and float like balloons. The tension is gone.

What makes playing the game with and without sound such a different experience? If you pull out one by one the various sound effects, ambient noises, and music that make the game scary, you'll find out. The raptors' footfalls top the list. This shows in the "distraction factor." With the sound off, you can actually play better, because the footfalls or raptor screams won't distract you.

Sound can reinforce a physical feeling and create a physical sensation. Have you ever hit a rollover button with your speakers all the way up? You feel it roll, baby! Sound can be important, even for menu buttons. In a menu, sound can convey a solid feeling like steel switches moving, or a light feeling like paper pages turning with a ruffle. This adds a lot to your production values. This is the same principle car manufacturers use. The sensation you get when slamming a car door is important. If the door gives a solid *thunk* and doesn't rattle, the car must be safe and solid, right?

OBTAINING OR CREATING SOUNDS AND MUSIC

Sound is everywhere in our daily life, so it's obvious why it would be so important to a game player. Sound effects often take on meaning in a game. A dark alley with a strange noise coming from behind an overflowing dumpster delivers a message of fear more than the dark alley would by itself. People yelling loudly can draw our attention to an area or make us want to flee in the opposite direction. You can also use sound effects to establish a time and a place. For instance, hearing crickets in the background or waves beating on a shore can add a great deal to a setting, without visually displaying anything.

Sound effects can also convey actions, such as firing a gun or a car colliding with a wall. It is this part of sound effects, the part that adds emotion or action to a scene, in which game programmers are most interested.

You don't have to come up with all the sounds yourself. Just as musicians buy CDs with loops, you can buy sound effects libraries. These libraries include sounds that will work directly, or with modification, with the vast majority of sound effects.

Creating unique sounds, or doing sounds yourself, is often a very simple process. If you have a personal digital assistant (PDA) or a portable recorder of some sort, you can often record the sounds yourself. For instance, if you have a game with animals, a visit to a pet store or local zoo is often all you will need to add the appropriate

noises. If you are creating a sports title, visiting a local sporting event will give you all the crowd and background noises you would ever need.

A word of caution: If you visit local areas to record sounds, keep in mind that you often need more than you think you do. For example, the sounds may not be the quality you need, or after editing, you may only have a few usable minutes from a 10-minute segment. Always try to get more material than you think you'll need.

The other basic type of sound effect for a game is the effects that occur when some type of action occurs. These sounds can take a great deal of time to produce and may require a tremendous amount of specialized equipment. Fortunately, as you'll see next, with a little effort and common items you can use some very simple ideas to record these types of sounds for your games.

RECORDING SOUNDS

It doesn't really matter what type of device you use to record sounds; ultimately, you have to get the data into the computer. For this setup, assume you are using a tape recorder, a digital recorder, or a PDA. In this section, you'll see how to connect these devices to the sound or microphone inputs on the computer's sound card, and how to change the sounds into a digital form.

How to Record

The first step is creating the recordings. You'll be creating several games in this book, and with this in mind, you'll need to create effects for a variety of sounds, such as gunshots, footsteps, and perhaps collision noises. These are actually quite easy to record.

The following table lists several types of actions you can easily record with common household items. You can use this list, or change it, so you can come up with sounds for many types of games.

SOUND TYPE	HOW TO RECORD
Car Crash	Fill a box with scrap metal and chunks of wood. Shake vigorously.
Fire	Take a piece of cellophane and crinkle it with your hands.
Door Slamming	Place the recording device near the door hinges and slam the door. While you're at it, open and close the door slowly, if you need that type of noise.
Body Collisions	Strike an item such as a pumpkin or watermelon with a piece of wood or a rubber mallet. Try various methods to get just the right sound. Watch out, though—this can be very messy! Another method is to wrap wet towels around wood planks and then strike them together. Or drop the planks a short distance to a concrete or hardwood surface.

SOUND TYPE	HOW TO RECORD
Rain	Record the sound of rain on a roof or metal sheet. Or if you don't want to wait for rain, simulate the effect. Cut the bottoms of five plastic cups into different shapes, such as a square, star, or ellipse. Then tape the cups together. Pour uncooked rice into the top of the cups. This will sound like rain falling.
Thunder	As with rain, record a thunderstorm, or simulate it, as follows. Make a simple "thunder sheet" by getting a piece of sheet metal cut to approximately 18" x 50". Then, fit 1" x 2" boards on one end (to use as a handle) and cut several holes in the other end. Hang the sheet by the holes from a ceiling or beam. To simulate thunder, shake the end with the handle. This can take some practice to master, so be patient if it doesn't sound realistic at first.
Footsteps	<p>The best way is to record the real thing. For outdoor simulations, walk on a gravel area. For indoor simulations, walk on a hardwood floor with a hard-heeled shoe. If you don't have a hardwood floor, build a 3" x 3" wooden box and use it to step in place. You can then flip it over to record stepping noises, or fill it with things like straw or newspaper to vary the sounds.</p> <p>To simulate walking in snow, press a shoe on an old strawberry container, a sofa cushion, or something similar. Doing this at an approximate stepping rhythm will simulate footsteps very well.</p> <p>You can also simulate animal footsteps. For a horse, strike together small squares of wood, or the two halves of a coconut with all the pulp removed. Or put sand in the box you made for human footsteps, and strike the box with the coconut halves.</p>
Machines	If possible, record the actual machine noises. For instance, if you are creating a car racing game, go to a race and record the sounds. Additional sounds that work well in games include saws, drills, and hammers.
Gunshots	Hit a leather seat with a thin wooden stick, such as a yardstick or ruler. For different types of sounds, experiment by hitting other materials with the wooden stick. If you want to record gunshots hitting another object, cut plywood into thin strips and then break them. It will sound as if shots are splintering the wood.

For ideas about experimentation, consider gunshots. As mentioned previously, you can hit a leather seat with a thin wooden stick. Strike various objects, and use sticks of varying strengths. Creating sound effects is very much trial and error, so spend time finding several objects that sound good, and record all of them.

Using a PDA

The next step is to connect your recorder to the computer. If you are using a PocketPC or Windows CE-based PDA, you can simply connect it to the computer and

transfer the recordings, which will already be in WAV format. If you are using this method, skip the next section, “Using a Recording Device.”

Depending on the sound quality of your PDA, the sounds may or may not be of value. If they are not good quality, you will probably have to use one of the methods discussed in subsequent sections to record your sounds.

Using a Recording Device

If you are using a tape recorder, mini-disc recorder, or other recording device, you will have to attach it to your computer’s sound card. Most sound cards have four connectors: Line In, Line Out, Microphone, and a MIDI/Game Port. Figure 4.1 shows the layout of a typical sound card.

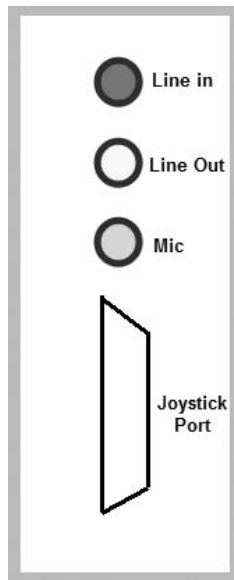


FIGURE 4.1 The layout of a typical sound card.

Most modern sound cards may not have a midi port or game port, as many devices now use USB connections.

The following list explains the various connectors:

MIDI/Game Port: The port most commonly used to connect a game paddle or joystick to the computer. This port also lets you connect a MIDI (Musical Instrument Digital Interface) device, such as a keyboard, to the computer.

Line In: A connector that lets you connect a sound source to the computer. Examples include CD players, tape recorders, and other recording devices.

Line Out: A connector that lets you connect the computer to anything that accepts sound input. Used most commonly for speakers or headphones.

Microphone: A connector that lets you connect a microphone to record your own sound files. If necessary, you can also connect a recording device to this port.

After you have located the Line In or Mic (microphone) connection, attach your device to the computer. Depending on the device, you may need different types of cables and connectors. The majority of sound cards use 1/8" (miniplug) jacks for Mic and Line In.

Using Your Sound Card's Mixer Panel

After connecting the device, open your sound card's mixer panel. The type of sound card and software you have will determine if you can access this from the system tray or from the Sound and Audio option in the Control Panel. The icon on the system tray will look something like Figure 4.2. Once you double left-click on it or access the option through the Control Panel, you will get a standard sound card mixer panel as in Figure 4.3.



The Sound and Audio option is available in the Control Panel for Windows XP users. For Vista, you will need to access the Control Panel and then the option Hardware and Sound, which looks very different from Windows XP.



FIGURE 4.2 The sound control icon shown in the system tray.

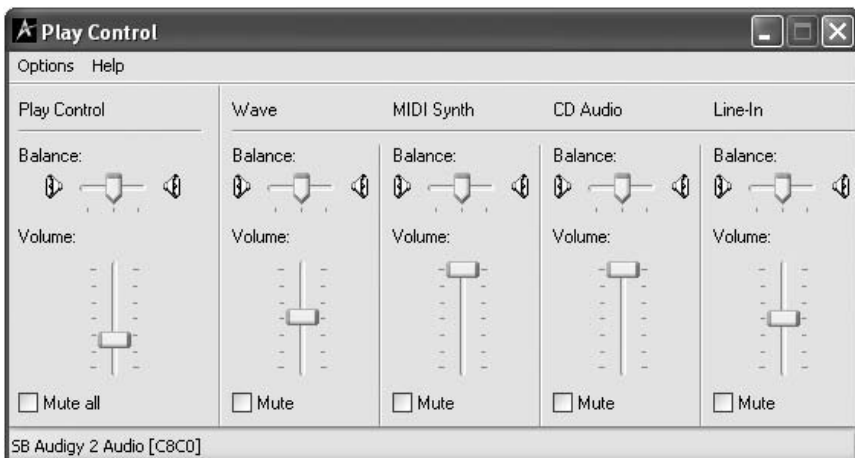


FIGURE 4.3 The mixer panel allows you to choose options related to the sound card.

When you first open the mixer, you will see all the possible playback volumes. Set the volumes as follows.

1. If the Wave Balance (the second slider from the left in Figure 4.3) is checked, click the Mute box to uncheck it.
2. Make sure the Wave Balance slider and the Play Control Balance slider (the leftmost one in Figure 4.3) are both at least halfway up.
3. Next, set the sound card's recording devices.
4. Choose Options | Properties.
5. In the box labeled "Adjust volume for," select Recording. Each of the devices from which your sound card can record will be listed in the window.
6. Click OK. This will display the Recording Controls window.
7. Make sure the volume of the category you plan to use is halfway up. For instance, if you are using the Microphone, ensure Microphone is halfway up. Figure 4.4 shows the recording control as it should appear.

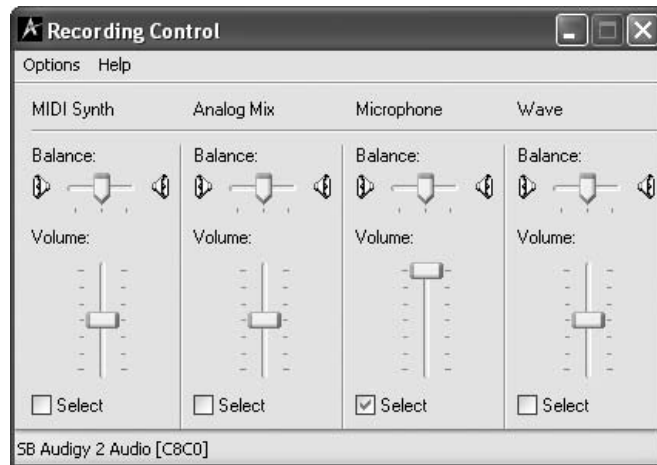


FIGURE 4.4 Microphone with the correct settings.

Using Windows Sound Recorder

A simple way to record sounds for your games is to use the built-in Windows sound recorder software and a microphone. The quality of the sound will depend on the quality of your microphone, and whether you can filter out any other noise in your environment. This is a good way to create simple sounds to give you an idea of how your final game might be before you record better or more professional sound. In computer game creating, developers usually put "markers" in the game. A marker can be a simple sound or a graphic, and in a big game development represents voice recordings or graphics that have yet to be created.

So you can use Windows' sound recorder as an easy and inexpensive (as the only additional equipment required is a microphone) way to get sound into your game, or as a temporary measure if you plan to create or buy better sounds later.

1. Click Start | All programs | Accessories | Entertainment | Sound Recorder, or if you are using Windows Vista, Start | Programs | Accessories | Sound Recorder.
2. You will now see the Sound Recorder dialog box as shown in Figure 4.5. If you are using Windows Vista, it will look slightly different but works generally in the same way.



FIGURE 4.5 Windows XP Sound Recorder program.

3. Click on the red button (or the Start recording button in Vista) and it will begin to record the sound. Speak into the microphone or make a noise.
4. To stop recording, click on the black square or the "Stop recording" button (Vista).
5. In Windows XP you will need to do File | Save As to save it in a folder, whereas Vista will automatically ask you for a location to save the file once you stop recording.

CREATING MUSIC

When you start creating music for a game, you usually begin with a basic understanding of the type of music you need. For instance, if you are creating music for a wrestling game, classical music is probably not going to be part of the piece. You may need to do some research. Discuss the requirements with someone, or find a way to listen to existing music that fits your needs. For a wrestling game, you might watch wrestling on television or attend a wrestling match to get a good understanding of the kind of music users would expect.

If you are writing music for a game that reenacts the Civil War, you might watch movies about the Civil War, or talk with music historians about the types of instruments or music popular during that time period.

It's important to understand that you're not looking to simply copy the music, but to discover what makes music appropriate for the time or era. Keep an open mind. You might base your music on what you've seen and heard, or may come up with unique ideas.

There are a number of programs on the market you can consider using. ACID and eJay, for example, use sound loops, which you can place onto the application and arrange them any way you choose. Both programs include samples you can use in your musical creations, and you can download clips from several Web sites. The companion DVD contains three packages you can use to try some of the looping programs. In the Demos folder is ACID XPress, the free version of ACID from Sony, and includes a 10-track version. Also included is a demo of the Pro version of ACID, for those who want to check out the features of a professional and powerful looping product. The full version costs around \$400 and includes over 1000 professional loops, so if you're serious about making music, check out the demo first to see how comfortable you are with it. Sony also has a cheaper product, ACID Music Studio, which is priced at \$70 and allows for live recording and effects processing. Check out www.sonycreativesoftware.com/products/acidfamily.asp for more information about the product range and which product is right for you. Also included on the companion DVD is a demo of Dance eJay 7, used primarily for making dance music and has a cost of around \$40. Other programs in the eJay range provide loops for hip-hop and techno-based music if you are looking for that type of sound for your games.



Using Loops

ACID and eJay afford you the capability to create music from loops, much like mainstream music is produced today. In the past 20 years, the majority of the music industry has used loops or samples in one way or another. This has drastically altered the music landscape, changing the way both amateur and professional producers create their music. A quick glance at many modern albums makes it clear they use loops.

The use of samples in many forms of music has brought about an entire industry that produces music especially for this purpose. Thousands of CDs are available that contain samples you can use for almost any purpose, in standard CD Audio format and in the file formats used by many leading music programs, including ACID and eJay.

Many of these CDs require that you pay for using their samples. There are two basic methods. The first is a royalty-based system, in which the CDs themselves may be free. However, you pay a royalty for every time the sample is used. In the second method, you pay an up-front fee, which gives you a royalty-free license that allows you to do almost anything with the loops from that point on. However, with either method, you usually cannot distribute the materials as a new collection of loops.

The Internet offers a third way to obtain samples. Many Web sites offer fee-based downloads, while others allow you to download their loops for free. Do a search on Google, or go to the relevant product Web sites you might want to use—for example, *Ejay.com* or *Acidplanet.com*.

ACID XPRESS

In this part of the chapter, you are going to install the XPress version of ACID, walk through the registration process, and then take a quick tour of the program. You will also go through a simple creation to give you an idea of how to begin making your songs.



ACID or eJay are not discussed in great depth, as they are considered secondary products within this book. The key aim of the book is to get you to make your own awesome games, so that is where you will spend most of your time learning. Both ACID and eJay come with extensive help material, and the walkthroughs and the examples here should be enough to get you in a position where you can begin making your own songs.

Before you begin, make sure your PC meets the requirements for the software.



These minimum requirements are for the full version of ACID, so some requirements may be less than specified. Additionally, your PC must meet and where possible exceed the minimum operating system requirements to ensure a reasonable user experience of the software.

- Microsoft Windows 2000 or XP 800 MHz processor (1 GHz if using video)
- 200 MB hard-disk space for program installation
- 600 MB hard-disk space for optional Sony Sound Series Loops & Samples reference library installation
- 256 MB RAM
- Windows-compatible sound card CD-ROM drive (for installation from a CD only)
- Supported CD-recordable drive (for CD burning only)
- Microsoft DirectX 8.1 or later
- Internet Explorer 5.1 or later

Installing ACID XPress

First, you need to install the ACID XPress software.



1. Locate the file `acidxpress50a.exe` in the Demos file on the companion DVD, and double left-click on it to launch.
2. You will then see a dialog box advising you where it will install the software as shown in Figure 4.6. Click Next to accept this default path, or click Change.
3. The software will begin to extract files in a temporary location and then begin the installation dialogs.
4. A Welcome dialog will appear, as shown in Figure 4.7. To continue, click Next; to cancel the current process, click Cancel.
5. Now you will see the License Agreement dialog as shown in Figure 4.8. Read the end-user agreement, and then select the "I have read the End User License agreement and the..." radio button. This will allow you to click Next.

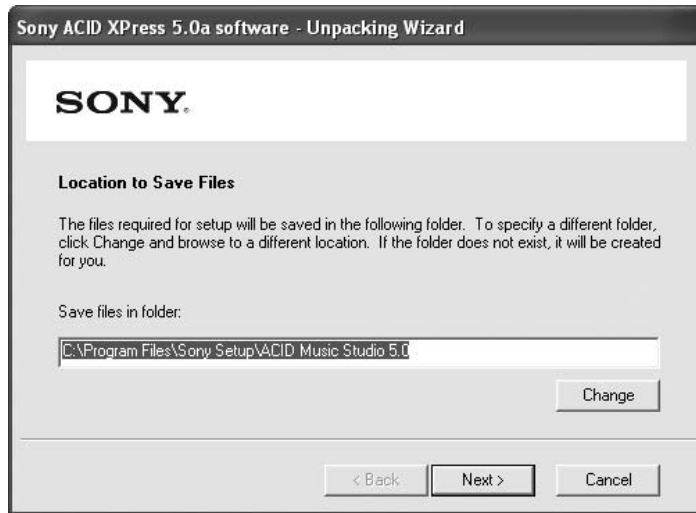


FIGURE 4.6 The default path dialog box.



FIGURE 4.7 The Welcome dialog for ACID XPress.

6. You will verify the installation path and where the plug-ins will be installed. If you are happy with those locations, click Next.
7. You will now see a Ready to Install message. Tick the “Install shortcut on the desktop” checkmark button, and then click Install.
8. The program will begin to install.
9. Once it has completed, you will see a message advising you that the installation was successful. Click Finish to complete the installation.



FIGURE 4.8 The end-user license agreement.

Running ACID XPress for the First Time

When you run the ACID XPress software for the first time, you will be asked to register the software before starting to use it (Figure 4.9).



FIGURE 4.9 The Registration dialog that first appears when you run Xpress.

1. Ensure Register Online is selected, and then click Next.
2. Another box appears as shown in Figure 4.10. You will need to enter your details.

FIGURE 4.10 Complete the dialog box with your details.

3. You will also need to tick the “By providing this registration information...” box before you can click Finish.
4. The registration program will now access the Internet, and after it successfully sends your information it will reply with a success message as shown in Figure 4.11. Click OK to continue.



FIGURE 4.11 The registration was successful.

5. You will now see the ACID XPress and a Show Me How dialog box in the middle of the screen as shown in Figure 4.12. This how-to box is very useful for a quick tour of the program and accessing information to help you get started as a new user.
6. You can either click on one of the options, or click Close.

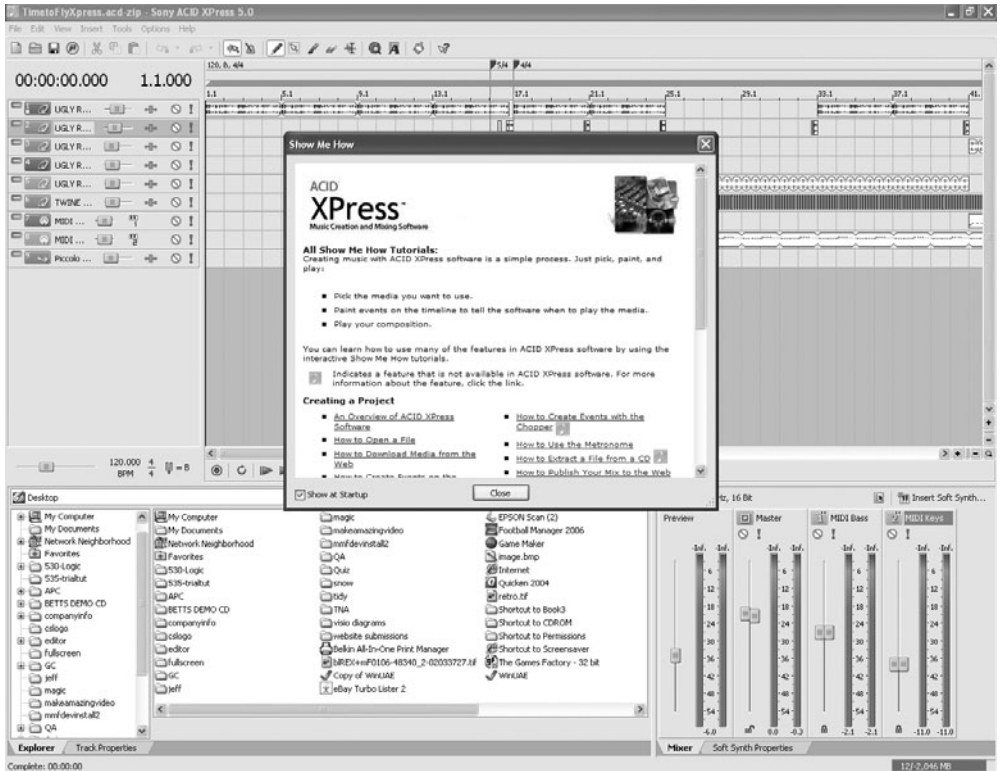


FIGURE 4.12 The ACID XPress window and the Show Me How dialog.

Tour of ACID XPress

The ACID XPress software can be broken down into a number of windows, identified in Figure 4.13.

- 1: **Track Header:** Where you place all your media files you will put into your song. Once it is here, you are able to place it onto the timeline. You can import a number of different formats, but some are not available in the Xpress version even though they are listed. For this simple example, you only need to import WAV and MIDI formatted files.

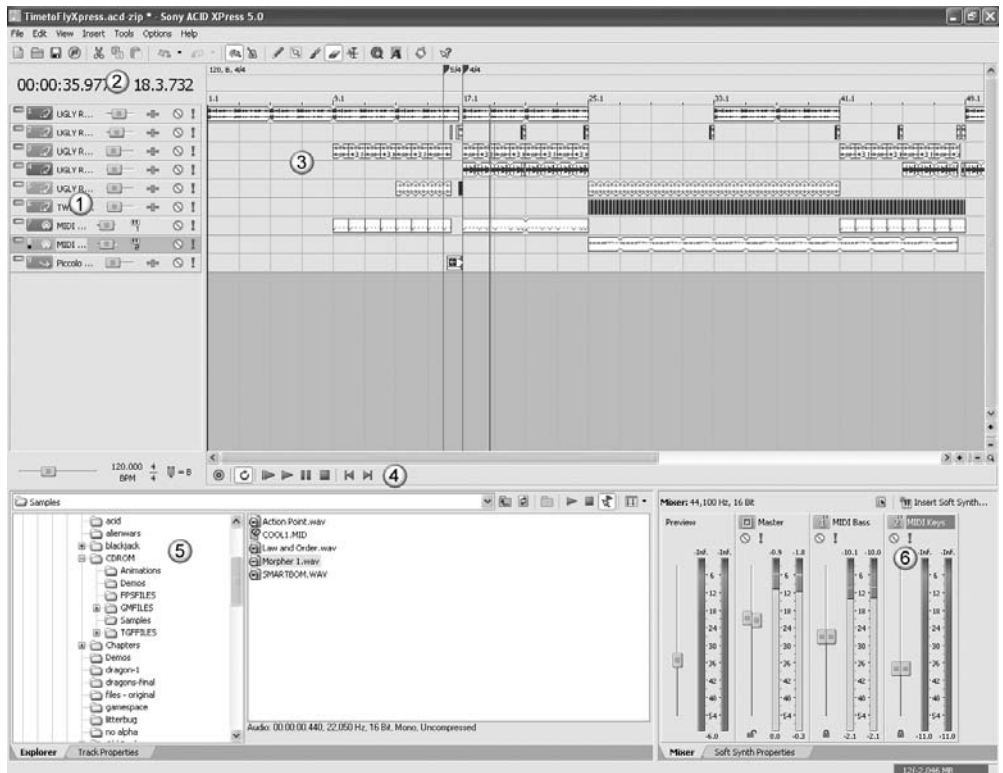


FIGURE 4.13 The different areas of the screen.

- 2: **Time Display:** Tells you exactly where the cursor is within your song on the timeline. The time on the left is the actual song play time, and on the right is measured in measures.beats.ticks.
- 3: **Time Line:** Where you'll do most of the work to create your music. The timeline has a number of rows where the different imported tracks will be painted or drawn onto the timeline to tell ACID to play that loop. Each square on the timeline is equal to a number of musical beats, as shown by the number at various intervals across the top of the timeline. How much you zoom in or out on the timeline will depend on what level of beats you will see. Also, the size of the loop you import will determine if it will fill the whole square or just part of it. If you import a sample but don't paint enough of it onto the timeline, the entire track will not play.
- 4: **Transport toolbar:** The standard play and stop controls of any digital music player. You can listen to your songs at any point by clicking play. The song will not by default loop the music you have created.
- 5: **The Explorer:** A Windows Explorer tab that allows you to search for media files to use in your creations. You are able to drag any supported items from

this window to the timeline. You can also access that particular track property by clicking on the tab next to Explorer.

- 6: Mixer: Allows you to keep an eye on the sound levels and finely tune them if required. For example, if it is too loud or too quiet, you can quickly and easily apply it to the whole song.

A Simple ACID Creation

You will now run through a simple creation in ACID XPress to get an idea of how to begin to put your songs together. There is extensive help documentation with the product, so look there when you need more information on making more complex songs.

1. The ACID program is open and has a pre-loaded song already displayed. Press the play button on the transport toolbar to listen to it and get a basic understanding of how it is put together.
2. You'll now remove this song by creating a new song. Click the File | New option. A New Project dialog box will appear, as shown in Figure 4.14.

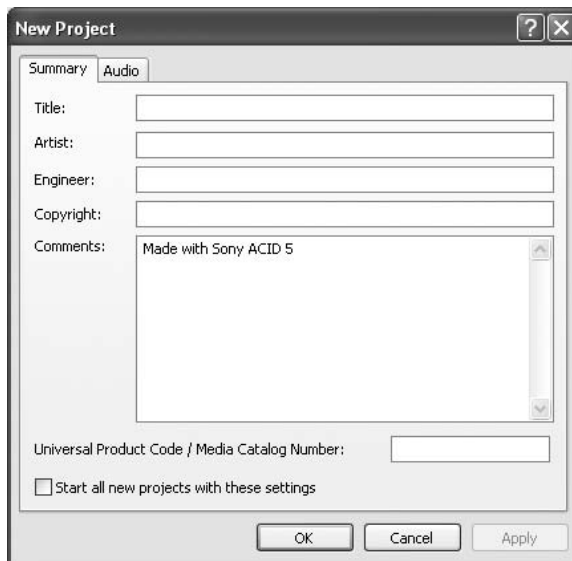


FIGURE 4.14 The New Project dialog.

3. Enter the information that will identify this project; type anything you wish in the Title, Artist, and Comments boxes, and then click OK.
4. You will notice that you have a blank track listing, so you can either add your tracks by right-clicking on the track header or by using the Explorer toolbar. For this sample, use the Explorer toolbar to add two sound files.



Browse the companion DVD and locate the Samples folder.

5. You will see the sound files displayed in the right-hand pane of the Explorer window. Double left-click on the items SMARTBOM.WAV and COOL1.MID to add them to the track header as shown in Figure 4.15.

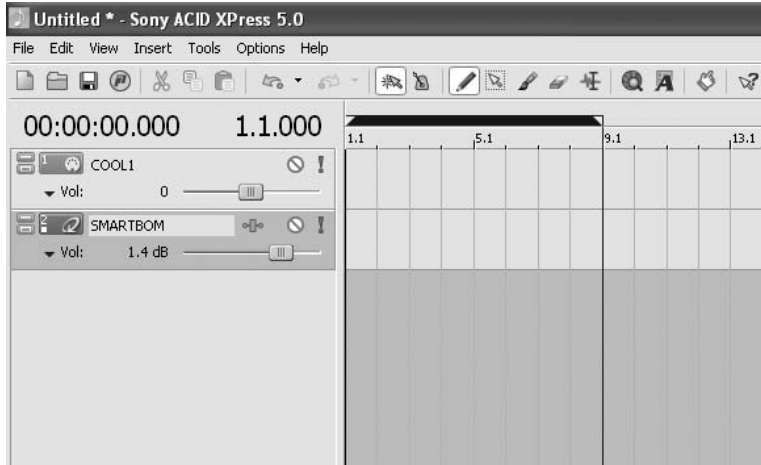


FIGURE 4.15 The two selected sound items.

6. Ensure the COOL1 item is selected (it will be highlighted). On the timeline, notice that the cursor has changed to a pencil icon. Starting from the very first box on that row, hold down the left mouse button, and drag it across about 34 boxes. You will now see the frequency of that item in the timeline in Figure 4.16.

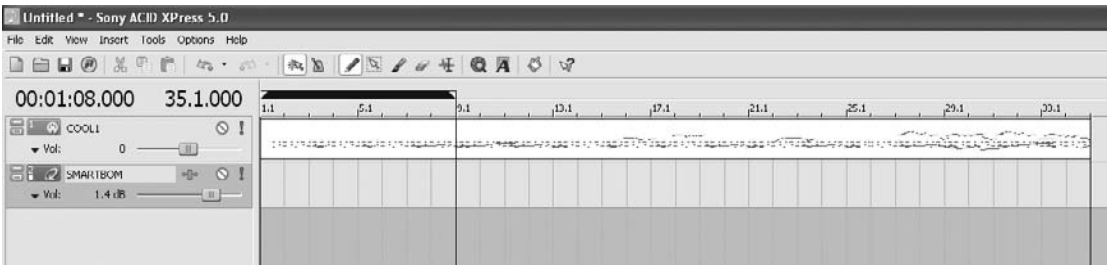


FIGURE 4.16 The file shown on the timeline.

7. Press the play button to listen to the song so far. You will notice that it plays a simple tune. In reality, you might only have a short file, and you might place these at various points in the timeline to create a song, unlike in this example where we have one sample with different levels and sounds in it.

8. Notice the blue loop bar above the timeline. When the loop button is selected in the transport toolbar, it will continuously play between the blue area. You will amend this very soon, once you have added your next loop into the timeline.
9. Click on the SMARTBOM track to highlight it. Click on various boxes within the time line. In Figure 4.17, there are two gaps between each. If you make a mistake, you can drag and drop any of the media files on the timeline by holding down the left mouse button and moving it to the required location.

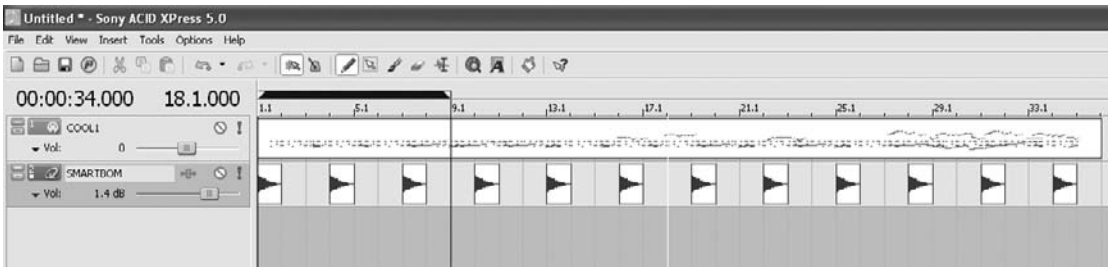


FIGURE 4.17 The second loop placed at a two-gap interval.

10. When you play the song, notice that the SMARTBOM sound is too loud. You can adjust the volume for this item only by moving the sound slider; in this case, change it to -16.4 dB as shown in Figure 4.18.

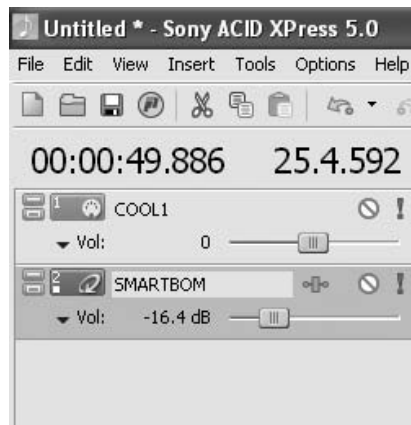


FIGURE 4.18 The sound slider.

11. Currently, it is looping a small part of the whole tune, which isn't very useful, as you want to listen to the whole song before deciding what to do next with it. To change the area, move the mouse cursor to the end of the blue

bar, where a left and right arrow will appear. Hold down the left mouse button and drag it; this will allow you to extend the loop area.

Now that this simple tune is complete, it is time to export it to a file. This is generally how these types of programs work; you import different loops and then create a whole new loop or tune from it.

12. Click File | Render As. Notice that a number of formats are available to select from; in this case, we are going to use the default of WMA. Type in a file-name. Click on the Template drop-down box, select 48 Kbps, and then save in a folder of your choice.



Not all formats are available in the Xpress version; for example, the WAV format.

Another limitation of the Xpress version is that you can only save at 48 Kbps; the lower the Kbps, the lower the quality. This also means that it is more compact for placing on the Internet for users to download and play.

The program will begin to render the file into a single WMA formatted file. After doing so, it will display the completed Export dialog as shown in Figure 4.19.

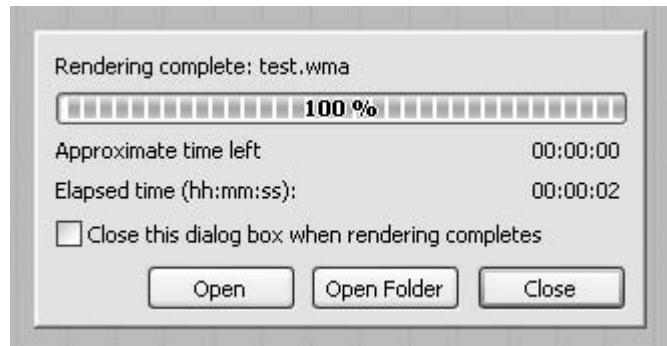


FIGURE 4.19 The Export dialog.

You will now be able to click Open to play it, or click on the Open Folder to open the location in which it was saved. We saved it in a Windows Media Player format, so to listen to it you will need the player from Microsoft. You can also find the example exported file, test.wma, in the Samples folder on the companion DVD.



Do not save in MP3 format if you intend to distribute music with your games, as there may be an additional cost in doing so. If you make commercial software and want to use MP3 files, you will need in most cases to purchase a distributable license. It is better in most cases to save in a license-free format, as you won't notice any difference in the quality. You can find out more about MP3 license costs at <http://mp3licensing.com/>.

DANCE EJAY 7

eJay is similar in concept to ACID XPress and Pro, whereby you place sound loops onto a timeline and create a song or another loop from it. The minimum and recommended system requirements are:

Minimum specs:

- Pentium 3, 800 MHz or higher
- 256 MB RAM
- Windows 98, ME, 2000, XP
- 1.4 GB free hard disk space
- 4x CD-ROM
- CD-WRITER (for Audio CD Burning feature)
- DirectX 9.0-compatible graphics card with 32 MB of video memory (16-bit color, 1024 × 768, 32 MB)
- DirectX 9.0 compatible sound card (16 bit)
- DirectX 9.0c
- Web browser

Recommended specs:

- Pentium 4, 1.8 GHz
- 512 MB RAM
- Windows 98, ME, 2000, XP
- 1.4 GB free hard disk space (for Install)
- 2.0 GB free hard disk space (for OS Cache)
- 4x CD-ROM
- DirectX 9.0-compatible graphics card with 64 MB of video memory (16-bit color, 1024 × 768, 64 MB)
- DirectX 9.0-compatible sound card (16 bit)
- DirectX 9.0c
- Web browser

The graphics card must be compatible with Direct X 9.0c.

Installing eJay 7



The companion DVD includes a demo version of eJay, *DanceeJay7Demo.exe*, in the Demos folder.

1. Browse to the Demos folder and double left-click on *DanceeJay7Demo.exe* to begin the installation.
2. Select the language to use for the installation; the default selection is English. Click Next.
3. The Welcome dialog will now appear as shown in Figure 4.20, Click Next to continue.

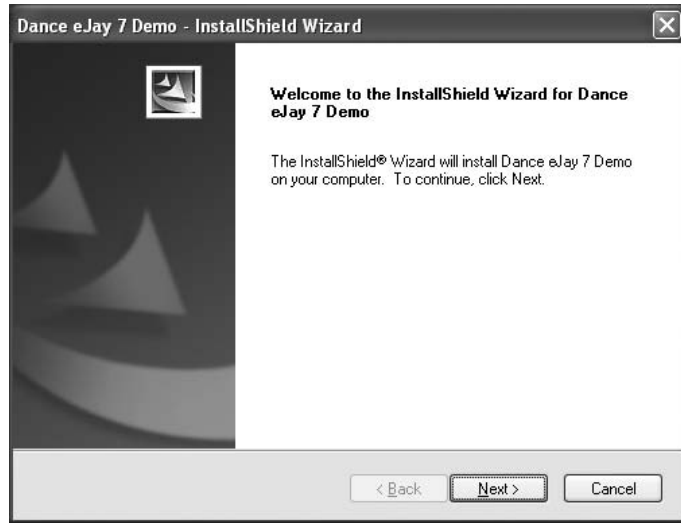


FIGURE 4.20 The eJay Welcome dialog box.

- 4. A license agreement dialog appears (Figure 4.21). Read the license terms and select the “I accept the terms of the license agreement” (if you agree with them). You will then be able to click Next.

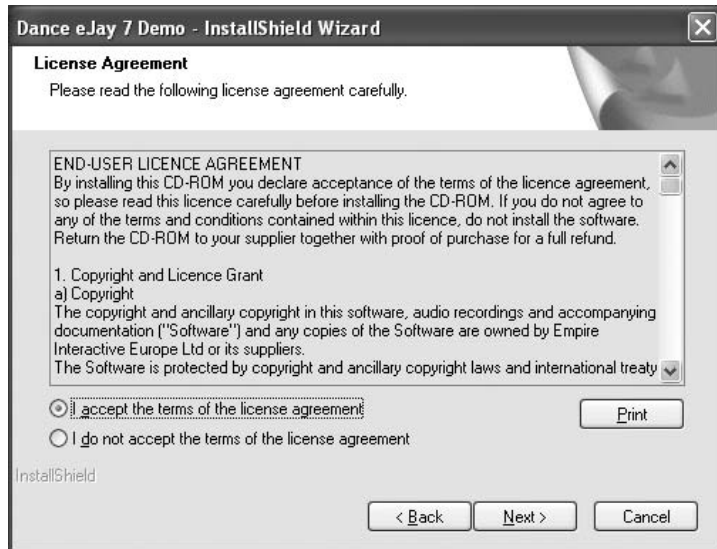


FIGURE 4.21 The License Agreement dialog box.

5. You will now see the default installation path as shown in Figure 4.22. If you are happy with this path, click Next; otherwise, click Change and select a new path.

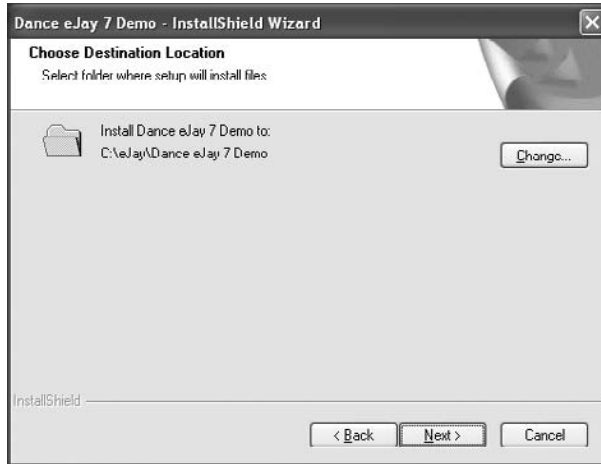


FIGURE 4.22 The default installation path.

6. You will then be asked if you wish to create shortcuts for the program; the default is to Install Shortcuts. Click Next to continue with the installation.
7. The program is now ready to install, so click Install to begin the installation. If you wish to quit the installation, you still have the option of selecting Cancel.
8. Files will begin to be copied to the local hard disk.
9. A dialog box will appear asking if you wish to register with Empire Interactive, the makers of eJay. Select either of the options to proceed.
10. The Installation complete dialog box will appear. Click Finish, and a Readme file will appear giving you the latest information about the installation file.

Running eJay for the First Time

You can run the eJay program from the desktop icon, or from Start | All programs | Dance eJay 7 Demo | Dance eJay 7 Demo. When the program starts, a dialog box will appear as shown in Figure 4.23, telling you how many days you have before the trial software expires. You can click on the link to purchase the software, or if you have days remaining, click Finish.

1. Click Finish.
2. The eJay main menu will start and display various options as shown in Figure 4.24.

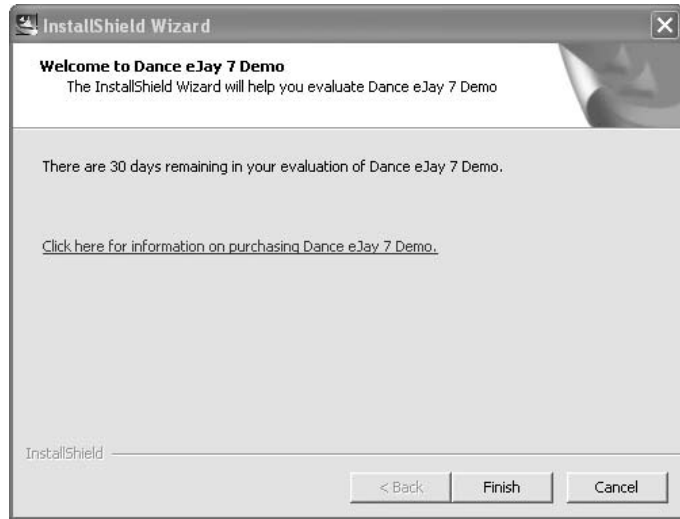


FIGURE 4.23 Dialog box showing how many days remain of the trial version.

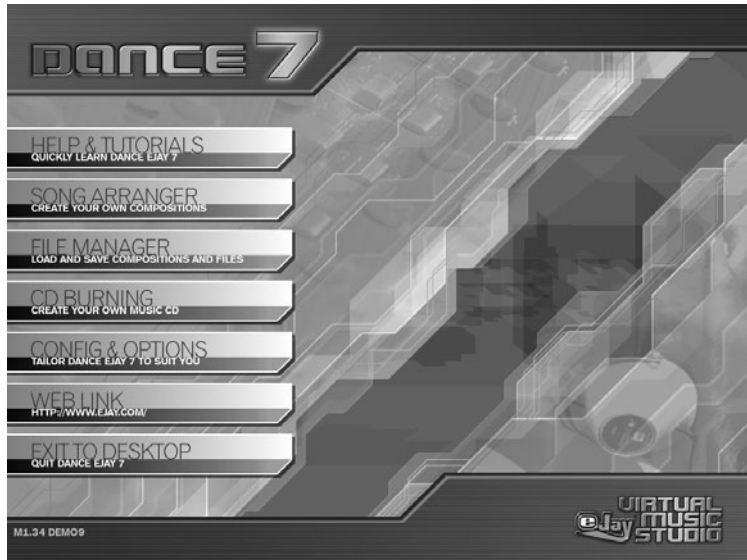


FIGURE 4.24 The eJay main menu.

You will now have a number of different options available from the main menu shown in Figure 4.24:

Help & Tutorials: Loads an HTML help page, which provides information on how to use the product.

Song Arranger: Where you will create your music from the loop files.

File Manager: Allows the user to load and save files, and search folders for relevant files.

CD Burning: Only available in the full version of the software, but allows you to burn any created songs to a CD.

Config & Options: Setup and configure various options for the eJay program.

Web Link: Takes you to the *www.ejay.com* Web site, and will confirm which language site you wish to view.

Exit to Desktop: Closes the eJay program and returns you to the desktop.

Tour of eJay

Clicking the menu option Song Arranger from Figure 4.24 will bring you to the main music creation screen as shown in Figure 4.25.

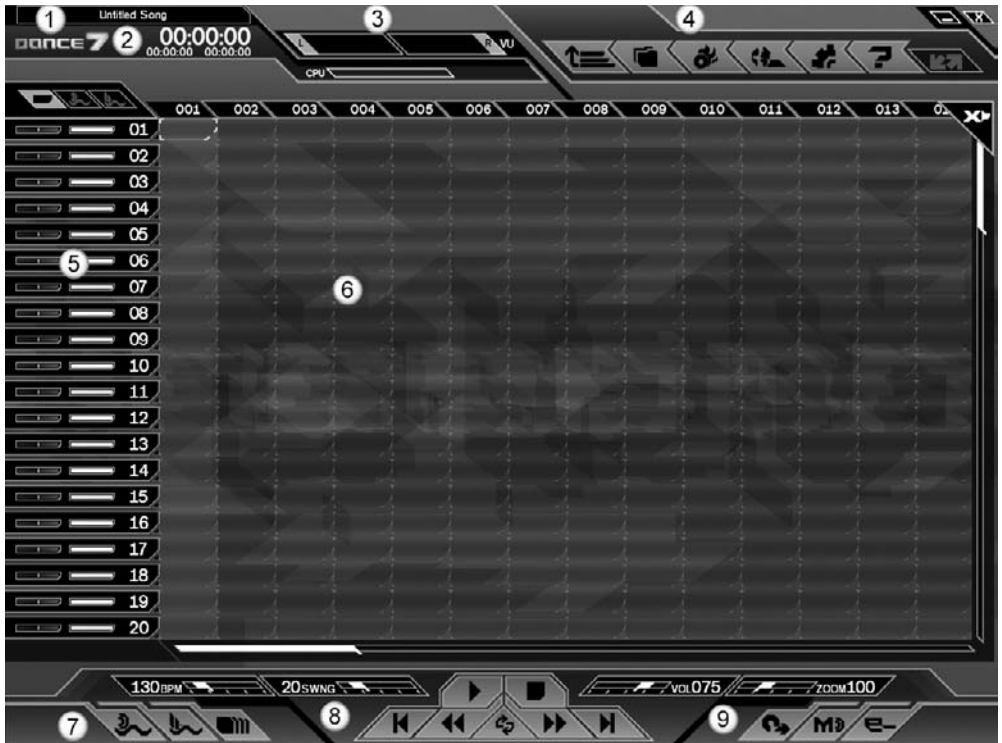


FIGURE 4.25 The main screen in eJay.

You will need to click OK to close the message about the demo version. Figure 4.25 shows the most important aspects of the Song Arranger:

- 1: **Song Name:** Identifies the name of the song, which will also appear in the File Manager, so you must name the song appropriately so you can locate it later.
- 2: **Song Timer:** Shows the length of time the song has been playing for since you clicked Play—the entire length of the song etc., in minutes, seconds, and hundredths of seconds.
- 3: **Sound Levels:** Shows a visual cue to the overall sound in the left and right speakers.
- 4: **Toolbar:** Menu options in the main menu are also accessible here; these are main menu, file manager, CD burner, Web link, config/options, and help/tutorials. The icon attached to the end of the toolbar is a redo/undo button for applying or undoing anything you did on the track placement area.
- 5: **Track Controls:** Displays the different track numbers you can apply loops to, and allows you to control the volume of each track. You can also change the volume through the left and right speakers for each track.
- 6: **Track Area:** Where you place your loops. Each block is equal to four beats; this is the timing of eJay.
- 7: **Three buttons:** Pan performance curve, volume performance curve, and sound clip archive. Pan allows you to draw a line across a track to match the left or right side of the speaker. Volume performance changes the volume of the track for its lifetime. The sound clip archive allows you to access the large archive of loops that are provided with the full product, and in this case, the loops provided with the demo to place them on the track area.
- 8: **Music Controls:** The standard controls for playing, stopping, and rewinding the song.
- 9: **Effect bar:** Consists of the Sample studio, master effects, and the sound clip editor. The sample studio allows you to create or edit samples, which are accessed via the file manager. Master effects allows you to apply a sound effect to a sample, which can change a sample's sound and create a completely different sound. You can apply up to four sound filters to a sample. The sound clip editor allows you to change the properties of the sound currently on the track area.

A Simple eJay Creation

You are now going to create a simple song using the included loops within the program. It will show how the basic creation aspect of the program is handled, and will allow you to experiment with the program to explore it further.

1. In front of you will be the sound arranger screen ready for you to create your first music masterpiece. If there are tracks on it already, you can listen to it by pressing the play button. Right-click on the track area and select Clear Everything to remove the tracks so you can start from a blank window.
2. Click on the sound clip archive button to bring up the sound archive as shown in Figure 4.26.



FIGURE 4.26 The sound clip archive.

3. Select Hard Lead Lines, and then select Garfueled with a single left-click of the mouse. Drag the song while holding the left mouse button, and place it at row Track 1 in position 001.
4. Left-click on the sound track you added at position 001, hold down the mouse button, and drag the copied item at position 005 on track 1.
5. Now navigate to “Hi hat drum loops,” select “Actatak hats,” and place on Track 2 from positions 001 to 023. This will now look like Figure 4.27.
6. Now, add another sample. Click the “Sound clip archive” button, and then select the Pads folder (you will need to scroll down to find this folder). Then, select Palendra Low, place it on track 3 in position 009, and drag and drop it again on track 3 position 013.

Now you are going to add some vocals to the song. You should do this with care in any music you are making for games, as it may not work right within a game context. Look at similar games in the same genre to see how they handle music.

7. Find the folder female vocals, select “in the distance,” and place it at line 04 and position 009.
8. Still within the female vocals folder, select and drop “ha aaa 1,” and place it at line 05, position 011.

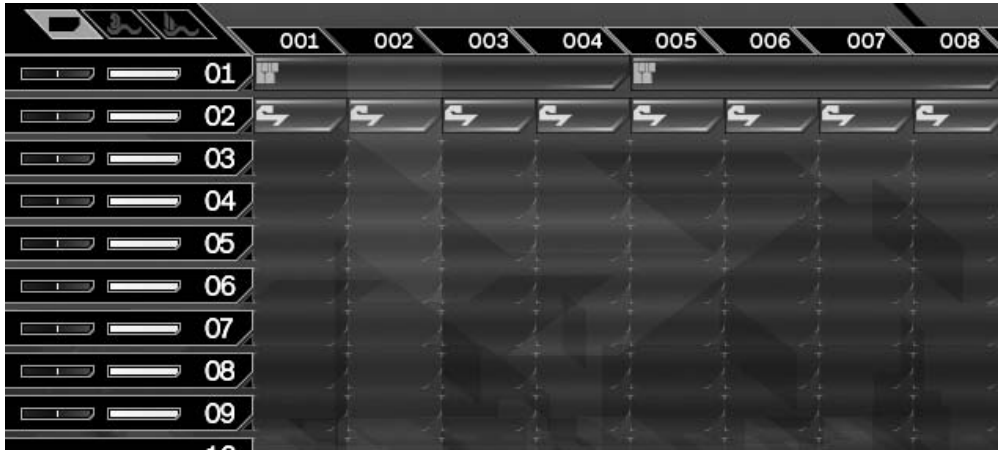


FIGURE 4.27 The two loops you have placed so far.

9. Using the white bar, scroll across to the right until you can see positions 017–027.
10. Under the hard bass lines folder, drag and drop “nerquit” and place it on track 04 and position 017.
11. Find the Pads folder, drag and drop “Palladium,” and place it on track 05 and position 017.
12. Navigate to the female vocals folder, and drop “on the rooftops” onto track 04 and position 021.
13. Finally, go to the female minor folder, and drop “under it all lolo 2” to track 04 and position 023.

You just completed your simple creation, so press the play button to listen to it. This simple process is the same one you use to begin to create your game music. There are many more features to eJay you can use to apply different sounds and effects, so take the time to look through them all.



Make sure you save your creations on a regular basis to ensure you have backups in case of a computer crash.

Exporting Your Songs

If you want to export your song into a format you can import into your game creation software, you need to access the File Manager editor.

1. Click on the File Manager option in toolbar. This is the second icon on the top right, which looks like two folders stacked onto each other.
2. You will now see the File Manager window as shown in Figure 4.28.
3. Click on the Export option. You will then be asked if you want to export; click Yes to export.

- The song will be exported and, depending on the size of the song, may take a while. A dialog box will tell you the song has exported successfully. Click OK.



FIGURE 4.28 The File Manager window.



You can find the example song and its exported WAV file in the EJAYFILES folder on the DVD provided with this book.

CHAPTER SUMMARY

It's easy to see why music and sound effects are so important to the development of a game. They can add so much to the experience of a game player by setting a mood or location. Well thought-out music and sound effects go hand in hand with great graphics on which so many developers prefer to focus.

In this chapter, you used two of the best tools available for game developers: ACID, with its easy-to-use interface to create a simple song; and eJay, to record a larger song. In the next chapter, you'll look at the elements required for designing a game.

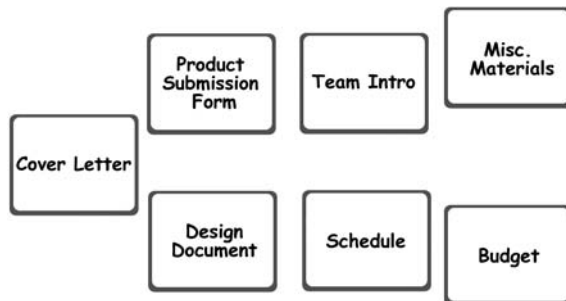
This page intentionally left blank

ELEMENTS OF DESIGNING A GAME

In This Chapter

- Introduction
- Game Elements
- Game Market
- Technical Information and Associated Risks
- Required Resources and Scheduling

Basic Parts of *Game Proposal*



So far, you have looked at the most fundamental parts of game design and development: equipment, the building blocks of a game (sights and sounds), and the genres, or classifications, of games. Now you are ready to look at the stage of game design in which you actually start designing the game—where you start to keep track of your resources, explore your limits, form your ideas, and put it all down on paper. This chapter will help you generate a design document in the early stages of developing your game. You'll also look at the game treatment and game proposal, which can be important parts of getting your game published. If you are a small group of people working together over the Internet or a one-person developer looking to only make games for fun, some of the bigger concepts (getting published and checking sales data) are not necessarily relevant to you at this stage. This information will be of benefit if you ever decide to make programs that will be published.

INTRODUCTION

All too often, people start generating design documents and jump right into development. While recording your ideas and other information, and prototyping and testing as you go, can be invaluable, an unbridled jump into development can be a waste of time and can physically and mentally lock you into a tight spot. It is far harder to change the course of something that has momentum than to set it rolling on the right path to begin with. Planning is important and cannot be stressed enough.

Not planning can be disastrous if you are putting a lot of time, talent, and money on the line; dashed hopes can throw a small team or business under. Don't jump right into this stage of breaking out the specific elements of your game, or you may develop yourself into a corner. Building the proper groundwork is essential. Otherwise, you may design and develop a game no one wants, you can't legally use, or represents wasted time and money.

The actual elements of a computer game are no secret; they are just mysterious to most people because they think of the design document as something you just dash off. They think that having enough pages to impress any reader is sufficient. They have trouble filling these pages as they struggle through each item, trying to fill in the blanks. The truth is, only after you have decided on your game idea, genre, and the overall feasibility of the game idea will you be ready to tackle this phase. The elements of your game should flow on paper, once you know what your game is.



Design documents are not fill-in-the-blank forms. If you approach them this way, you will be frustrated, and your game will not be as good as it could have been had it been planned up front. Design documents are the result of your game idea. They are not game ideas waiting to happen; they are guidelines for the areas you should develop in your game. But as you'll see, they are not the end-all.

There are three phases of game design: predevelopment, development, and post-development. We are most concerned with predevelopment at this point. During predevelopment, you should be defining your limits and strengths, researching the feasibility of your game idea, and, of course, defining and refining it.

Before you develop a game, you must determine if your audience will like that game. As mentioned previously, you have to design for an audience, whether it is for one person or a million. And you have to know who these people are, which computer systems they like or dislike, and other factors explored here.

Along with what type of computer system you design for, your audience will determine how complex your game will be, how long it will take to play, and its content. You should design with the best technology in mind, but obviously, you are limited by the technology you are familiar with and have access to. The latter part of this book discusses just that.

In designing a game, you should include *only what you need*. This is important, because many designers throw in all the elements they can think of, in a predetermined order. What the *proper parts of a game are* is not at issue here; what will *make a game successful* is. Remember, any bold and conclusive statements you've heard about what a game *has to be* are wrong. The truth is that a game must be fun for the intended audience. What a game should be, or contain, continues to evolve and is never set in stone.

GAME ELEMENTS

Simply listing the elements of a game is easy, but where's the fun in that? Anyone can just list game elements and attempt to fill in the blanks, but it takes more than that to design a game worth developing. Remember, with the tools on the market today, anyone can make a game, but few can make a game worth playing.

Element One: Game Type

You looked at game genres previously in detail; now is the time to fill in the blanks. Write down your game type from the following list. You need to at least know this bit of information.

The book *Game Architecture and Design* defines genres in their truest sense, and breaks them down into the following seven types:

Action: Lots of frantic button pushing

Adventure: The story matters

Strategy: Nontrivial choices

Simulation: Optimization exercises

Puzzle: Hard analytical thinking

Toys: Software you just have fun with

Educational: Learning by doing

This is a good start for your design decisions and will greatly simplify other decisions. The basic genre you develop will determine the focus on technology, art, content, and research. It will even determine the approximate size of your team, budget, and other resources.

As you decide about your game's main genre, you should also make notes about the other genres or game types you hope to incorporate into it. Keep in mind that adding, or layering, genres on your game increases everything—time, money and resources needed, and the complexity of the project. After you have a good idea of the type of game you want to develop, depending on your strengths and weaknesses, you are ready to move on to the next step.

Element Two: Game Idea and Game Treatment

You are now ready to write your game idea down, but not the treatment. The idea and the treatment are two very different things, and people often confuse the two.

Game idea: A game idea is just that, an idea. You should write it first, to convey your game idea to others.

Game treatment: A game treatment is written after substantial research, design, and development has been done. It serves primarily as a selling document to pique the interest of publishers, investors, and department heads in larger companies. This is a concise document for an already well-formed game project. In other words, while the game idea represents the sum total of what you plan for the game, the game treatment is a distillation of a much larger body of work and only touches on the highlights of the project.

Initially, you should write a rough draft of your game idea that presents as much information as possible about the game, clearly and concisely. It should spell out the general resources (time, talent, and cash flow) the game will require, and why you think it is such a great idea. You can then use this document to discuss and research the feasibility of the game.

As the game project gears up, you'll need to line up resources and (possibly) team members, determine needs, develop budgets and schedules, and define the game to a great degree. The original idea will change, evolve, and grow more solid. This will generate a mountain of useful information.

At the end of this stage, you'll write the game treatment, which will explain the exciting game development effort you have underway.

The treatment *generally* contains the (proposed) title, the genre, the feel of the game play, the overall look of the game, features you plan for the game, and any marketing information that will back up the feasibility of the title. Money, budgets, and dollar amounts should wait until after the publisher is interested in your game.

As a selling document, the treatment should open with the most marketable feature of you or your game development effort. If you are a top-selling developer, or if you developed a technological wonder or an artistic masterpiece, present those facts first.



Be careful. This document is deceptive to many because of its brevity, but writing it concisely and effectively requires a great deal of industry knowledge and writing skill. You are attempting, in as few words as possible, to get a publisher or investor to invest in your idea. This document is the equivalent of the query letters writers send out to get book and article publishing deals, and the cover letters that accompany business proposals. These are all selling documents and contain the same basic elements.

Even if you are making a game on a small scale, you should still get in the habit of writing down your ideas and documenting the development of your title. This focus will benefit your title, and the practice will clarify your thoughts and clear the way for new ideas to bubble up.

Element Three: Technology

This element consists of the game platform and the technology needed to play the game.

You should know the system requirements for your final game. Will it require a CD-ROM drive, a special video card or peripheral, a certain amount of RAM, or other special resources? What operating systems, drivers, or special software will the user need? These are all considerations the publisher will want to hear about.

Element Four: Audience

Several questions are important here.

- Who did you develop the game for, and why?
- Did you get audience input?
- What were their suggestions?

Keep in mind the previous sections of this book, dealing with game design and the audience.

Element Five: Team

You need a team, even if you are the only one working on your game. Here are some questions to ask.

- What team members will you need, and what jobs need to be done?
- Who are your team members, and where will you get them?
- What are their strengths and weaknesses?
- How will you manage them, and do you have any experience managing people?

Element Six: The Design Document

So, what should a design document contain? The design document comes from having a good game idea, along with the breakdown of the elements needed to develop

that idea into a game. Upcoming developers always want to see a design document, and with good reason, because it represents a complete game, along with the elements it requires. It is what the developer aspires to create.

Looking at an existing design document can be very useful as a guide. However, remember that it is for someone else's title and most likely will not be a perfect fit for your game. Like the game treatment, a design document is a product of your game; it should come after you flesh the game out, not at the start. If you are doing a 3D Shooter that is action-oriented, you don't need a huge backstory. In fact, that may be a detriment to the document from a development and selling point of view.

Appendix A, "Design Document: First-Person Shooter," includes a sample design document, but is not the "fill-in-the-blank" form most of you were hoping to find. However, reading and adapting it should serve as a guide for defining your game. Take note of the elements of this design document, but realize that your own game may have none, more, or all of the elements listed in that document.

Sample Design Document Outline

If you'd like to follow along with an actual design document, Appendix A contains the complete version of a design document for a 3D shooter. You can also use this as a basic template for creating your own design documents.



ON THE DVD

You can find the design document template, in Microsoft Word format, on the root of the companion DVD; the file is called "Design.doc."

What Is a Design Document?

A design document is often overlooked in the rush and excitement of a game idea. After all, if you have a unique idea that could conceivably be a great game, why would you want to waste your time working on something that doesn't really get you any closer to the end product?

Many times, even relatively large development teams don't spend the time to create a fully functional design document. Most game developers will try to stay away from unnecessary work, but the long hours spent creating a thorough design document will actually save countless hours later down the development road. You might be lucky enough to create a very good quality game without a design document, but the key word in this is "lucky." Most often, a game that begins without a properly developed design document will be delayed for months, or may not even be finished.

Creating a design document is similar to creating a movie script. In it, you will write details of an exact story (if you have one—for example, racing games would probably not have a story), an overview of the characters or opponents you intend to create, detailed descriptions of every level, and so on.

If this is the first time you've ever considered creating a design document, you should be aware of a few things.

First, the design document is not chiseled in stone. It can and should evolve as the game does, but shouldn't be drastically altered. The design document will serve

as a sort of road map to how the project will develop and should be as complete as possible. That being said, you should change it when necessary; for example, to include a new character or plot change. Design documents are team oriented, and therefore should include as many contributions as possible from the individuals who make up the team.

Sometimes, one person is the main author of a design document. If this is you, be careful not to be offended if someone suggests that you change something in the document. Input from others is important to the process, and can give you invaluable information.

Appendix A should serve as a good guide to creating a design document. Feel free to change what you see there for any projects you are working on. Remember that creating a design document is not an exact science; not all games are alike, nor are all design documents.

Also, proofread carefully! You would be surprised at the number of simple spelling errors that appear in most design documents. While everyone misses a word now and then, you should try your best to keep grammar and spelling mistakes to a minimum. This may not seem important to you, but again, you don't know who might end up reading your document.

Importance to Team Members

A design document is important to all team members for a number of reasons. For a potential publisher, it details the game and provides a vision of what you hope to accomplish. For a development team, its purpose is rather simple; it sets out the responsibilities of everyone involved.

Depending on the team member, a design document will mean different things. Producers will use it to make their estimates, while programmers may look at it as a series of instructions for carrying out their part of the project. Artists will use the design document to help them visualize the characters they need to create. Designers often use it to scope out important elements, such as the mood for a level. Audio personnel need to have a basis for developing sound effects and music. The design document may be the only place they can truly acquire the appropriate knowledge.



While the design document is very important, it doesn't take the place of meetings among the members of the development team. Having team members share thoughts at regular intervals is very important. These meetings don't have to be formal. They can be in person or over some electronic medium, such as a discussion board. Like most things, it's not important how they occur, just that they actually do.

Things to Include

Now that you have a basic understanding of design documents, let's look at the components or ideas that make them up. Many teams will include information such as legalities, target audience, and market analysis for a game in their design document. While this can work, it would make more sense to include those types of

“business-related” materials in a game proposal to a publisher, something you’ll look at in more detail later in this chapter. It’s counterproductive to have team members scrolling through pages of information they really don’t need to review.

Game Overview (Storyline)

This may be the most important piece of the entire design document. Without a solid story or game overview, the later steps will be much more difficult to create. Be thorough with the game overview. If you leave something out, go back and fix it immediately. Sometimes the smallest details can make a big difference in a large project.

Because you don’t know exactly who will read your design document, make sure to include as many details as possible, just as you would if you were creating a good storybook. Many teams place background information (information that tells how the situation shown in the game came about) in its own category, but because it relates to the story, you can put it in the game overview section. However, some genres, such as a sports simulation, wouldn’t need a background section.

Levels

The next item to address is the levels that make up a game. If you do a thorough job in the preceding step, this one is very easy. You compile a list of levels, in the order in which a player will encounter them in your game, adding any details you feel are necessary. Some optional materials to include about the levels include ideas such as the layout, and a general description of the placement of enemies. Try to create a mood for a level at this time. If you do, a designer or artist can simply browse this information to get a feel for what he or she needs to create.

Creating a set of maps for the levels can be helpful to the members of the team, especially the programmers and level designers. These maps can be very detailed pictures, but more likely will be a set of simple lines, circles, and squares that form a rough layout of the levels. You can see an example of this in Figure 5.1.

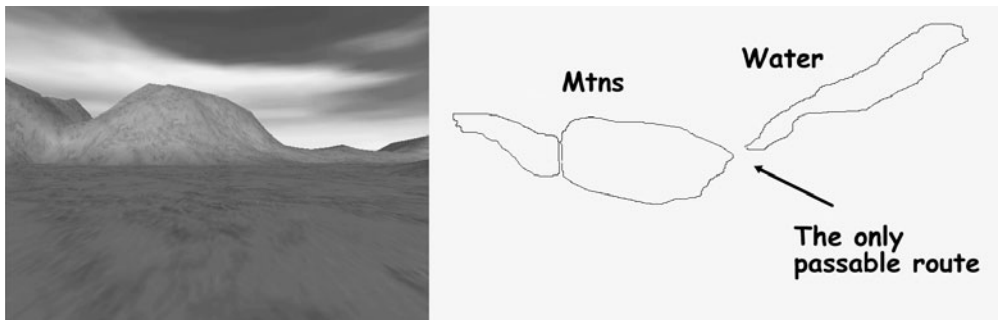


FIGURE 5.1 A level with a map.

Heroes and Enemies

The next section of the design document deals with the characters in your game. Like the level section, the character section should almost fall into place if you created a detailed game overview.

There are two basic types of characters in most games: a hero, and enemies. You can include details of the hero, such as background information or rough sketches. These will help team members understand what they're trying to accomplish. For every hero, you should also include a list and description of animations that apply to that character. Depending on their role in the story, you can also include descriptive ideas of their intelligence level and strength, and basic information about how they react to the rest of the characters. Again, this information will be beneficial to the team when they are working on those characters.

After you finish with the heroes, create a section for enemies. This could include anything (human or not) that will attack a player. For instance, in an FPS, you might include a dinosaur; in a space combat game, you could include an asteroid. Follow the same basic procedures you did in fleshing out the heroes, making sure to include similar details and sketches where appropriate.

Finally, you need to include information about the types of weapons the characters will have access to. Include detailed descriptions of every weapon either type of character can access. Sketches can be valuable for everyone on the team. Also, create a list that details the damage each weapon will cause, along with the type and quantity of ammunition for each weapon (see Figure 5.2).



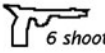

<i>Weapons</i>	<i>Damage</i>
 <i>Knife</i>	<i>Light damage to guards</i>
 <i>Flame Thrower</i>	<i>Strongest Damage Possible</i>
 <i>6 shooter</i>	<i>Limited damage.</i>
 <i>Automatic Rifle</i>	<i>Only auto repeat weapon.</i>

FIGURE 5.2 List of weapons and damages.

Notice that the sketches in Figure 5.2 are simplistic. You can make the sketches as detailed or as simple as you wish. Often, it's more important to get them drawn than to worry about how great they look. You can always go back and clean them up later.

Menu Navigation

Another very important element is creating a list that details the game's menu navigation. It helps you keep track of how parts of the game link to other parts, and is particularly important to the programmers. You should create the main menu and a simple illustration of how the screens will link together. You don't need anything fancy, but all the menus should be included. For example, you could use something like Figure 5.3 to display information about the opening screen of a racing game.

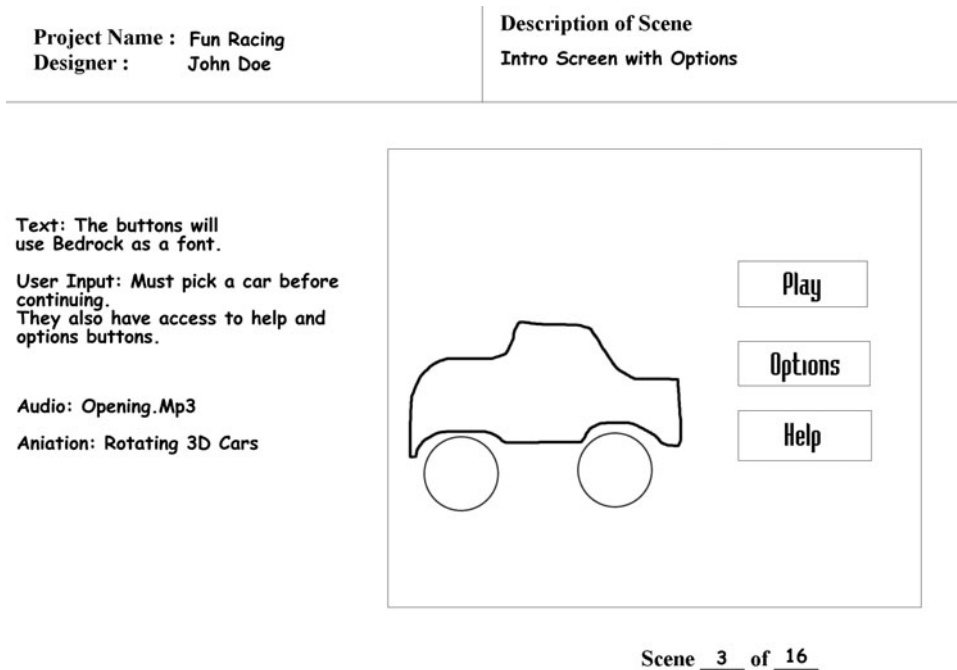


FIGURE 5.3 A fictitious racing game opening screen.

User Interface

The user interface goes hand in hand with the menu navigation system. For convenience, you could place them under the same category, because they deal with many of the same ideas. The information for this category can be text-based information about what you plan to do, but ideally, some type of sketch works best. Like

most of the design document, this section doesn't have to be fancy, but details are important (see Figure 5.4).

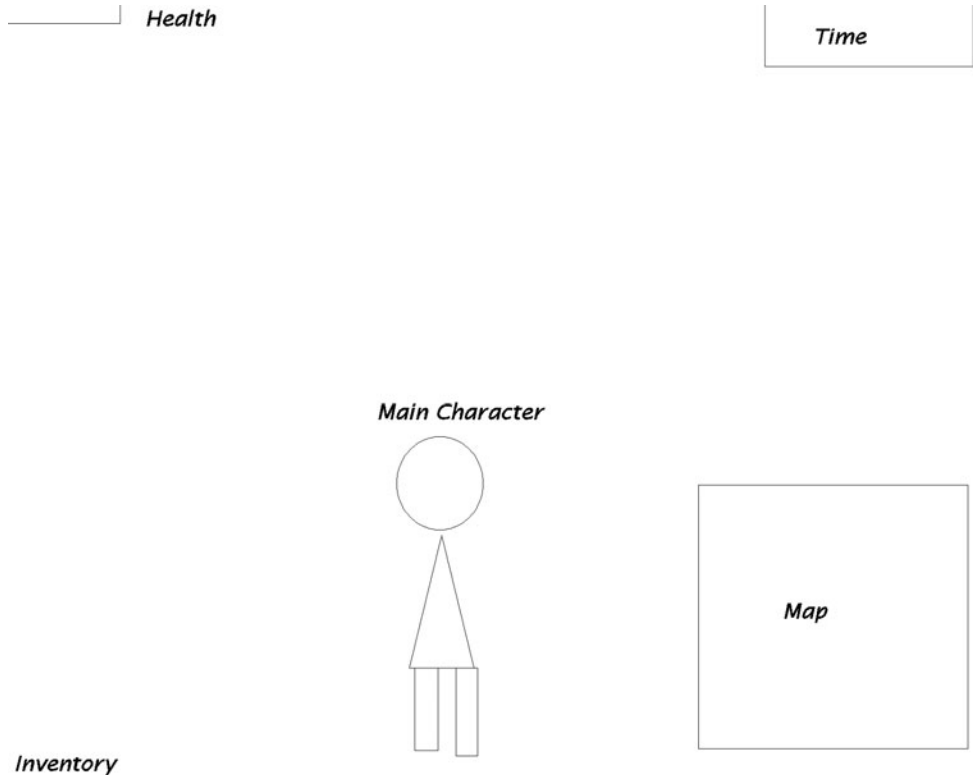


FIGURE 5.4 A user interface example.

Music and Sound Effects

This section is important to the audio personnel on the team, and to the programmers who will use their sounds in the game. You can discuss the tools you plan to employ, the types of sound effects, and possibly detail the music you have in mind for the levels you listed earlier in the process.

In the first draft of the design document, the most important details to include here are the audio formats and sound API (application programming interface) you plan to use, along with what types of music and sound effects. For instance, you should decide if you are going to use MIDI, WAV, or MP3 files for the music, and if you'll need things like explosions or footsteps for sound effects. You should also list the genres of music you're planning, such as rock or pop. This keeps the programmers and audio personnel informed, so they are not surprised two months into the project.

Single or Multiplayer

The next step focuses on the game play itself. If you worked hard on the game overview, you may have already covered this, and this section will be much easier. If not, start by determining if the game will be single player, multiplayer, or both. For example, if you are planning an FPS clone like Quake, you might decide that it is single player only. If you are doing a sports game like basketball, you'll probably want to have multiple player support. Sometimes, the information in this area of the document appears in other areas, but you shouldn't worry about duplication of ideas. This is especially true on a first draft, as you can always change the document later.

If you're designing a single player game, you can probably describe the game experience in a few sentences and perhaps break down some of its key elements. For example, if it's a Quake clone, you could begin by setting up the location of the game. Next, you could detail the types of enemies players will face, and the route to complete the game, such as players have to finish 10 levels before the game is over. You could also list how the game ends, and what happens if a player doesn't finish a given level on time. You might also include a projection of how long players will take to finish the game, and how a player wins the game.

Obviously, a multiplayer game is more difficult to design, and it's harder to create the design document.

A multiplayer game description starts the same way as a description of a single player game. Take a few sentences to describe the basics of the game play. The design document for the basketball game mentioned earlier could start by mentioning the type of game it is—for example, a street ball game, a college game, or an NBA or international rules game. You might also decide what types of options the game will include, such as franchise mode for a professional game, or what types of parks you'll include for a street ball game.

Now is also a good time to decide how many players will be able to play simultaneously, and how you plan to implement the client-server or peer-to-peer system. For instance, do you plan to use something like DirectPlay or another API, and how many players do you plan to allow to play against one another? In the basketball example, you need to decide how many people can play on a team. You don't need to have complete technical details, but at the least, you should discuss the client-server vs. peer-to-peer system issue. Optimally, in this section you will also discuss potential pitfalls that are common in multiplayer games.

Miscellaneous and Appendix

The final area of the design document should discuss miscellaneous information that may be specific to a certain type of genre, or doesn't fit neatly into another category. You can name this category anything that works well for you. For example, suppose you decide to do a basketball street game and you want to include information about the way the basketball players will dress, so you can keep track of players from both teams. For example, you could have one team play in white shirts and another in red shirts. If you have comments to make about several different topics,

split the discussion up, to keep everything easy to read and follow. The appendices are a good place to put items such as sketches or concept drawings. This way, you can refer the reader to an appendix instead of cluttering up your text.

Wrapping It Up

After the design document is finished and everyone on the team has had a chance to read it and suggest changes, you should print a copy for everyone. Keep the original in a safe place, where it will not be altered unless the necessary parties agree. For example, if you leave the document on a server where everyone can access it, team members may decide to alter it on their own, which would ultimately defeat the document's entire purpose.

Element Seven: The Game Proposal

The game proposal is a much more formal document. Its general purpose is to be used to approach a publisher for possible funding for your project. If you plan to develop the project with your own money, a game proposal is probably unnecessary. A game proposal takes the design document to a higher level and involves several issues that should not appear in a design document. For example, it should include information such as technical specifications, and marketing, financial, and legal issues. After the design document has been thoroughly digested by the lead programmers or the senior members of the team, it should be included with the game proposal. A game proposal will not be broken down in detail here, but for a quick overview, refer to Figure 5.5.

Basic Parts of Game Proposal

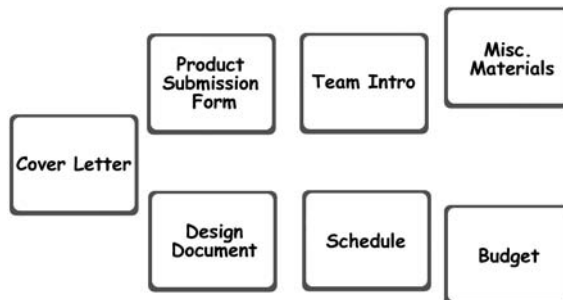


FIGURE 5.5 Basic parts of a game proposal.

GAME MARKET

The game market is a good place to begin. You can determine a market simply by looking at titles that are similar to the one you are developing. By looking at sales figures, you can figure out how large a market a particular style or genre has, and

your potential for sales. An excellent source of game market data is popular online Web stores or game Web sites. You can also search certain organizations that keep track of software sales in various countries, which are more accurate, including ELSPA (www.elspa.com) or NPD Group (www.npd.com). You may need to be a registered organization and pay a subscription of thousands of dollars to get actual sales figures, so if you are an indi developer or one person making smaller games, this isn't a viable option. Table 5.1 shows an example of the data easily available on the Internet. If you are planning a game to run on multiple platforms, you should try to break this information down among them.

Table 5.1 Example of a PC Chart Available on the Internet for One Month in 2007

GAME NAME

Command & Conquer 3: Tiberium Wars
 The Sims 2
 The Sims 2 Seasons Expansion Pack
 Football Manager 2007
 World of Warcraft: The Burning Crusade
 The Sims 2 University
 The Elder Scrolls IV: Shivering Isles Expansion Pack
 S.T.A.L.K.E.R Shadow of Chernobyl
 The Sims 2 Pets Expansion Pack
 The Complete Collection of the Sims
 Medieval II: Total War
 Age of Empires III
 Battlefield 2142
 ArmA: Armed Assault
 Neverwinter Nights 2

TECHNICAL INFORMATION AND ASSOCIATED RISKS

The most important things to list in this area are the team members' development experiences in developing a game similar to the one you are working on. For instance, if the lead programmer developed 3D engines in the past, mention this experience. If this is your team's first 3D engine, you should convey this as well. If you are using third-party software, such as a code library or sound effects, list them as well.

There are technical risks in any project, so try to provide a workaround if you encounter one. For example, you could purchase a 3D engine from company XYZ if the engine you are working on does not pan out.

REQUIRED RESOURCES AND SCHEDULING

The final area to include is required resources and scheduling information. The schedule should include an estimate of the project's length, along with specific milestones that occur along the way, such as an alpha or beta product. The required resources should include all financial estimates, such as the cost of employees, hardware, and software.

CHAPTER SUMMARY

In this chapter, you looked at the main parts of good game design, and how to document that design. Nailing down your game type and the elements that go into creating your game is just the start. You then saw how to bring these decisions into your design document, and looked at the importance of the game treatment and game proposal.

There are no set rules for game design and development. Still, with the material from this chapter, you shouldn't have a problem creating a functional game treatment, design document, and game proposal for yourself or for a team.

Two of the greatest things about game development are that the genres are flexible and the technology is powerful. There are no barriers to entry. In fact, it is incredibly easy to get started in game development. It is all about knowing what tools and resources you have at your disposal and where you want to go with your ideas. In the next chapter, you'll use some of those ideas and begin to make a 2D game.

This page intentionally left blank

INTRODUCTION TO GAME MAKER

In This Chapter

- Installation
- System Requirements
- Game Maker Interface
- Resource Explorer
- Menus and Toolbar



This chapter introduces you to Game Maker. The program's name is indicative of its ease of use; with the Game Maker, you can make computer games without writing a single line of code. Using easy-to-learn drag-and-drop actions, you can quickly make professional-looking 2D games. The games can contain any number of elements, including backgrounds, animated graphics, and music and sound effects. After mastering the drag-and-drop actions, you can move on to the simple built-in programming language that lets you add advanced functionality to any game. To top it all off, Game Maker is free of charge for the lite version and allows you to create standalone games you can distribute freely. For a small fee, you can upgrade to the professional version, which will remove some of the restrictions on the lite version.



You'll find Game Maker in the Demos folder on the companion DVD. You can also get it from the Web site www.yoyogames.com.

INSTALLATION

Game Maker is easy to install.



1. Put the companion DVD in your DVD drive.
2. In Windows Explorer or My Computer, open the Demos folder.
3. In the Demos folder, run gmaker.exe. This starts the installation process. You will see a screen similar to Figure 6.1.



FIGURE 6.1 The Game Maker Welcome dialog.

4. Click Next to display the Information screen. This provides useful information about the product, what type of computer it should be installed on, and where to get more information if required.

5. Click Next to display the license agreement (Figure 6.2). Be sure to read it before proceeding.

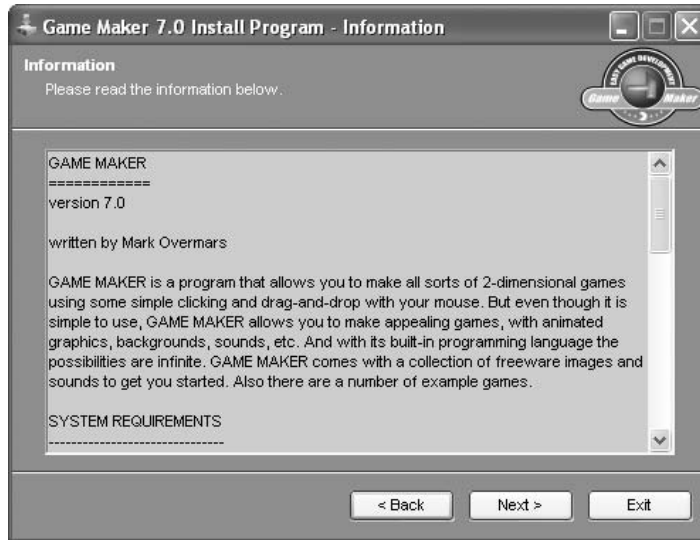


FIGURE 6.2 The Game Maker License Agreement screen.

6. Ensure the "I agree to the above terms and conditions" radio button is selected, and then click Next.
7. You will now see the Directory dialog box. This shows where the default installation of the program occurs (Figure 6.3).

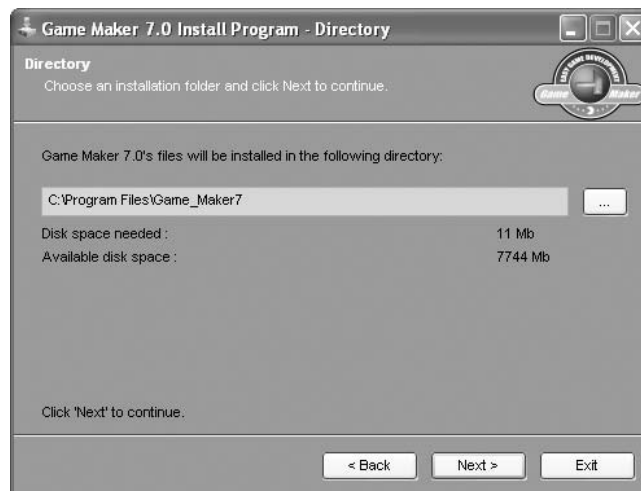


FIGURE 6.3 Default installation path of Game Maker.

8. Click Next to accept the default installation path, or click the ... button to change the path. In this example, the default path is fine, so click Next to continue.
9. If the folder does not exist on your computer, a message box will appear. Click Yes.
10. You will now see a confirmation dialog box, which is ready to install the product onto your computer, as shown in Figure 6.4. If you are happy with the details, click Start; if you want to change the details, click back. In this example, click Start.

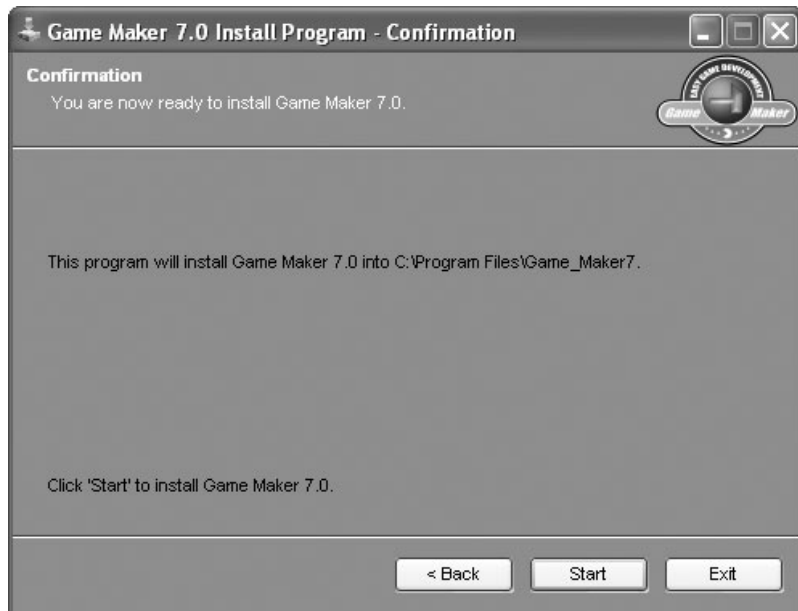


FIGURE 6.4 The Confirmation dialog.

11. Files will begin to be copied across, and once complete, the Installation – End dialog box will appear.
12. In this final dialog, you can click on the View Readme dialog to display an information page with details about the product and where to obtain help. If you click Exit, it will launch the Game Maker program automatically because the Launch Game Maker 7.0 box is ticked. For now, you don't want to launch it, so untick the box, and click Exit.

Notice that there is a new desktop icon for the Game Maker program, and it also appears in the Start All Programs | Game Maker 7 menu. You can see the desktop icon in Figure 6.5.



FIGURE 6.5 Game Maker desktop icon.

SYSTEM REQUIREMENTS

Before you begin learning the Game Maker program, it is important to know the system requirements for the program, to ensure your kit setup is optimized for running the program.

- Pentium PC or higher
- Windows ME, 2000, XP or Vista (or higher)
- 10 MB of hard disk space
- 65000 colors (16-bit)
- 800 × 600 screen resolution
- 32 MB 3D graphics card (DX compatible 8.0)
- Sound card

GAME MAKER INTERFACE

Start Game Maker by double left-clicking on the icon on the desktop, as shown in Figure 6.5. A dialog box appears, telling you the current version and the advantages of using the professional version (Figure 6.6).

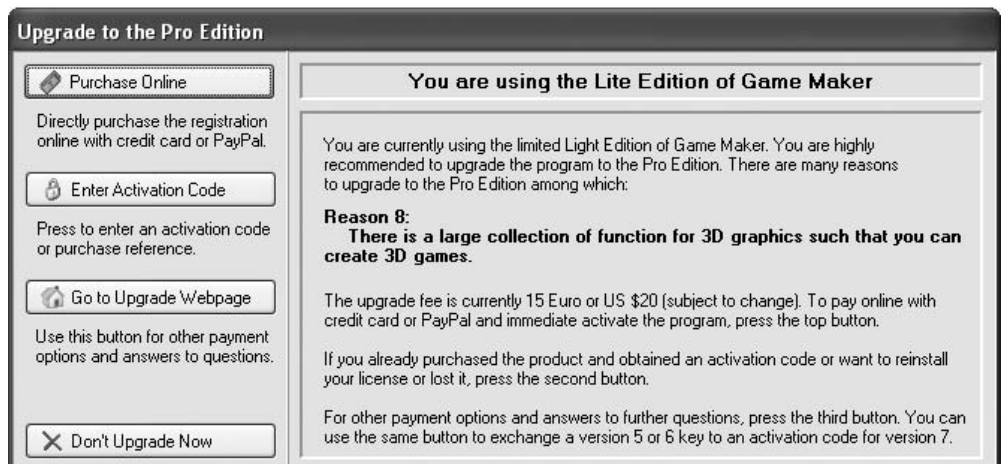


FIGURE 6.6 The starting dialog for the Game Maker program.

On the left side of the dialog box are a number of options:

Purchase Online: This will take you to a page to purchase the upgrade serial code.

Enter Activation Code: After purchasing your license, you will receive an activation code. You will need to select this item and fill in the relevant details before you can start to use all the features in the program.

Go to Upgrade Webpage: This takes you to a Web page, which will provide you with further purchasing details and help. You will need to register on the YoYo Games site before you are able to see this information.

Don't Upgrade Now: This allows you to continue to use the lite version.

Click Don't Upgrade Now to start the program.

Game Maker is very easy to work with because everything has been designed with simplicity in mind. In Figure 6.7, you can see the various elements that make up the Game Maker Integrated Development Environment (IDE).

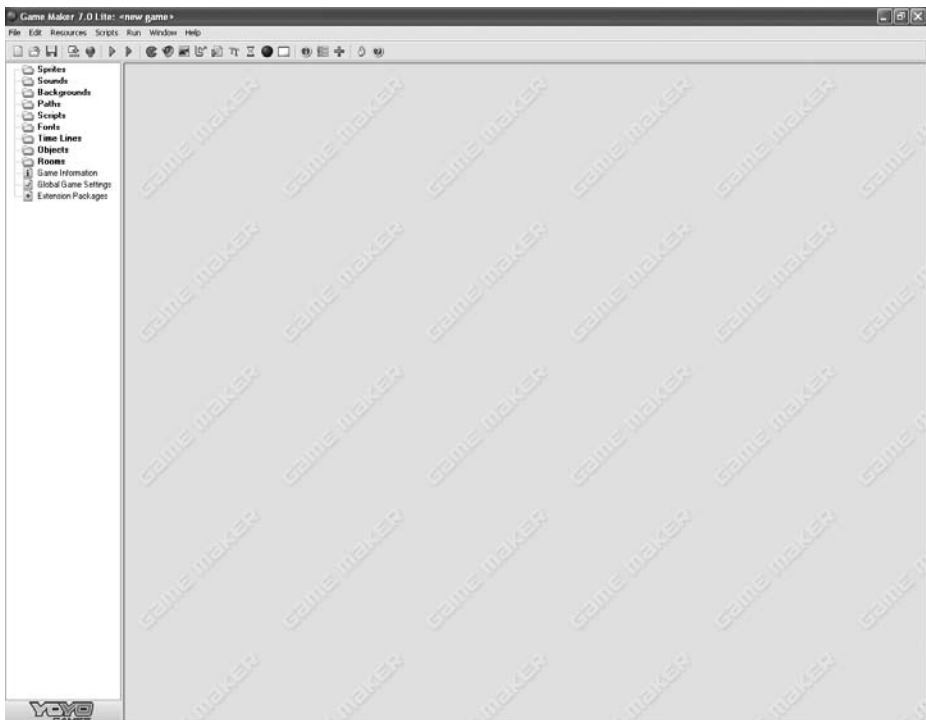


FIGURE 6.7 The Game Maker interface.

RESOURCE EXPLORER

In the upper left is the Resource Explorer. This displays the various resources that can make up a game project: Sprites, Sounds, Backgrounds, Scripts, Objects, and

Rooms. Along with the resources, you'll see Game Information and Game Options. You'll also see a standard type of menu and toolbar at the top of the screen.

The Resource Explorer gives a tree-like view of all the resources in your game. Here's how it works:

1. To open a resource, right-click it.
2. If a resource has a + next to it, you can click the + to expand the tree view and see the resources inside it.
3. If a resource has a – next to it, you can click the – to collapse the tree view and hide the resources inside it.

Figure 6.8 shows resources with the + sign. You can click the + to expand the view of these resources. Figure 6.9 shows the Resource Explorer after the user clicks on the + sign for Sprites.



You will only be able to expand the folders when there is content in them. This is either content you created, or if you have just loaded a file that contains sprites, for example.

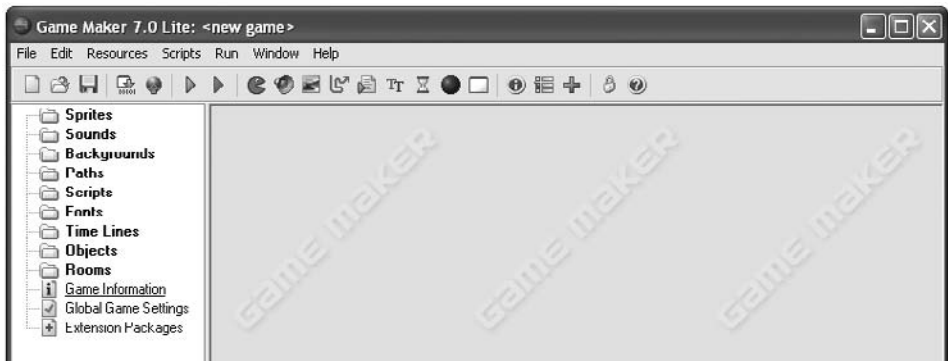


FIGURE 6.8 The unexpanded Resource Explorer.

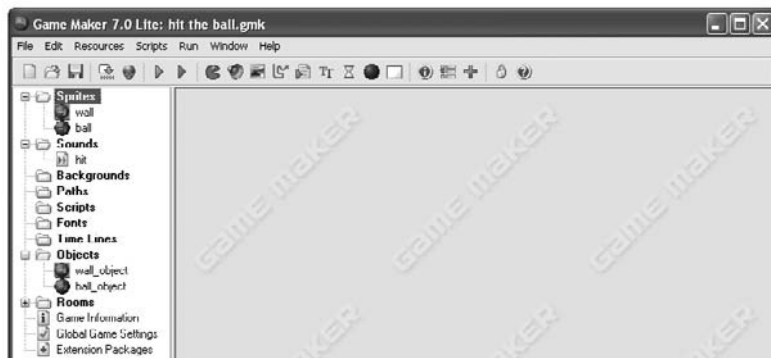


FIGURE 6.9 The Resource Explorer with items expanded.

MENUS AND TOOLBAR

The menus and toolbar appear at the top of the IDE. The following sections explain what you'll find there.

File Menu

The File menu includes the usual options.

New: Creates a new game project.

Open: Opens an existing game file.

Save: Saves an existing project with its current name. The first time you save a project, it will prompt you for a name.

Save As: Saves a project, but prompts you for a name first.

Create Executable: Creates a standalone game that can be run like any standard Windows program.

Publish Your Game: Connects to the Yoyo Games Web site so you can upload your finished version to the Web.

Merge Game: Allows you to combine a number of games, so you can get access to all its resources.

Advanced Mode: Shows all available resource and menu options. This is useful if you want to make the interface less cluttered and easier to use.

Preferences: Lets you set various preferences for Game Maker.

Exit: Exits the program.

Figure 6.10 shows the File menu options.



FIGURE 6.10 File menu options.

Edit Menu

The Edit menu contains commands that affect a currently selected resource, such as a sprite, object, or room. The commands available depend on the currently selected item.

Create resource: Lets you create a selected resource in the selected resource group.

Duplicate: Makes a copy of the current resource and adds it after the currently selected item.

Create group: If resources are combined, they are called a *group*. Adds a group into the project.

Delete: Deletes the current resource.

Rename: Lets you rename the current resource.

Properties: Lets you edit the properties of the current resource.

Find Resource: You can search for a particular resource. This is useful if your game contains many resources.

Expand Resource Tree: You can do this by clicking on the + sign, or using this menu option. It will open the folder and display its contents.

Collapse Resource Tree: You can do this by clicking on the – sign, or using this menu option. It will collapse the folder and hide its contents.

Show Object Information: This will display information for the selected resource. This will include its movement, if it's a sprite, etc.

Figure 6.11 shows the Edit menu options.

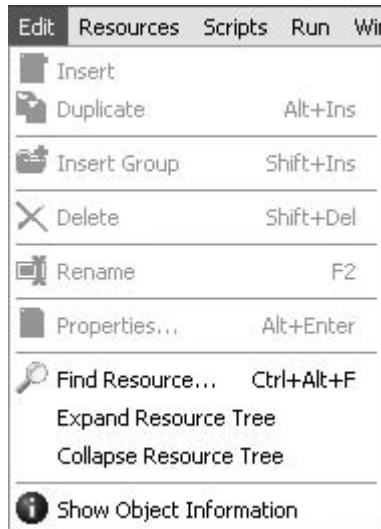


FIGURE 6.11 Edit menu options.

Resources Menu

The Resource menu lets you add resources to your game. You can add a new resource of any type from this menu.

Create Sprite: Create a sprite object.

Create Sound: Add a sound to your resources.

Create Background: Add or create a background image to your games.

Create Path: You might want to create a path for your enemies to follow.

Create Script: Explanation in sentence case with no end punctuation.

Create Font: Add a font to your creation. You will need to specify the font's size and type.

Create Time Line: If you want to create some actions at a particular moment, you can use the Time Line.

Create Object: Create an object with specific events and actions.

Create Room: A room can be thought of as a level in your game.

Change Game Information: Contains information about your game, which the user will be able to see.

Change Global Game Settings: This will change settings that apply to the whole game, including screen resolution and author information. Some options are only available in the Pro version.

Select Extension Packages: Only available in the Pro version. Allows the user to add more functionality to Game Maker.

Figure 6.12 shows the Resources menu options.



FIGURE 6.12 Resources menu options.

Scripts Menu

The Scripts menu option allows you to import code from an outside file. Scripts are small pieces of program code used for advanced features. They are not covered in this book.

Figure 6.13 shows the Scripts menu options.

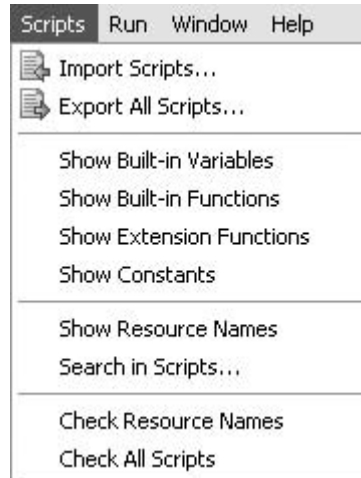


FIGURE 6.13 Scripts menu options.

Run Menu

You will use the Run menu option to test your creations.

Run Normally: This will run the game normally as if a user was playing it.

Run in Debug Mode: This is a special mode, and allows you to access information about the game, pause it, etc. This is good for testing your game and finding any bugs.

Figure 6.14 shows the Run menu options.

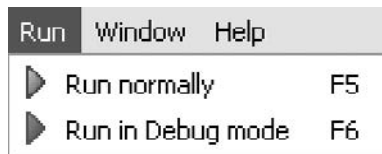


FIGURE 6.14 Run menu options.

Window Menu

The Window menu contains the usual commands to manage the different windows.

Cascade: Displays all windows so they are partially visible.

Arrange Icons: Arranges all the windows when they are minimized as icons.

Close All: Closes all open windows.

Figure 6.15 shows the Window menu options.



FIGURE 6.15 Window menu options.

Help Menu

The Help menu lets you access the help information for Game Maker. The options are self-explanatory (Figure 6.16).

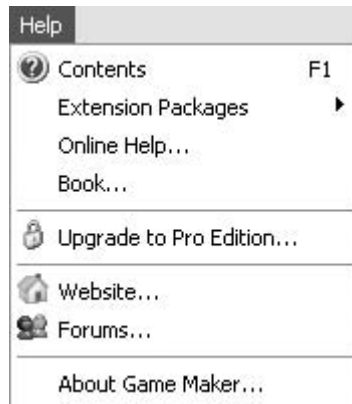


FIGURE 6.16 Help menu options.

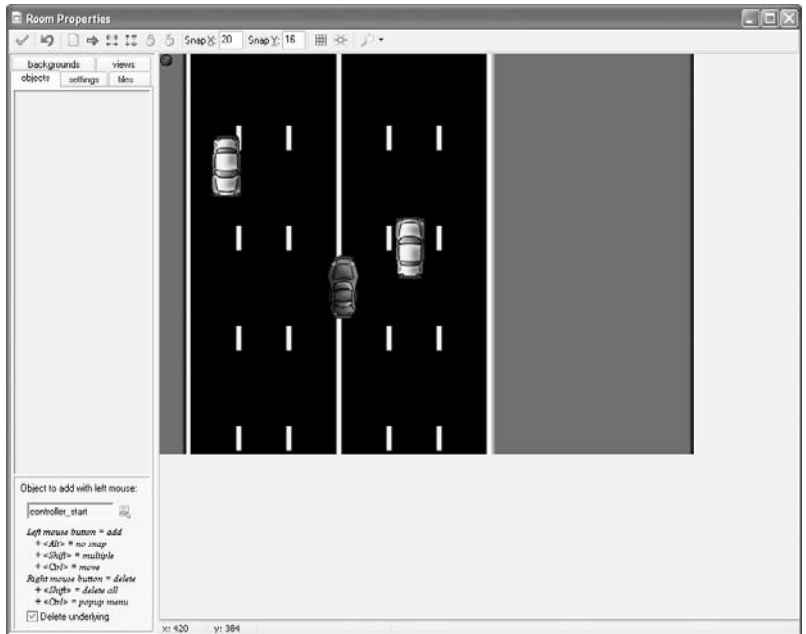
CHAPTER SUMMARY

This chapter walked you through the installation of Game Maker and looked at some of the basics of the IDE. In the next chapter, you'll look at Game Maker in more detail, and create your first game.

YOUR FIRST GAME MAKER PROJECT

In This Chapter

- Game Maker Basics
- Creating a Simple Program
- Save and Run



Before you start using Game Maker, you need to learn about a few of the ideas behind the program, which will help tremendously when you create a game.

GAME MAKER BASICS

Games created with Game Maker take place in one or more rooms, which correspond to the levels you see in a game. For instance, a Racing Car type of game would have a room with a road, player-controlled car, and the computer-controlled cars (see Figure 7.1 for an example). Game Maker is only 2D, so the rooms are flat. However, you can give a 3D appearance to a room by designing the graphics appropriately.

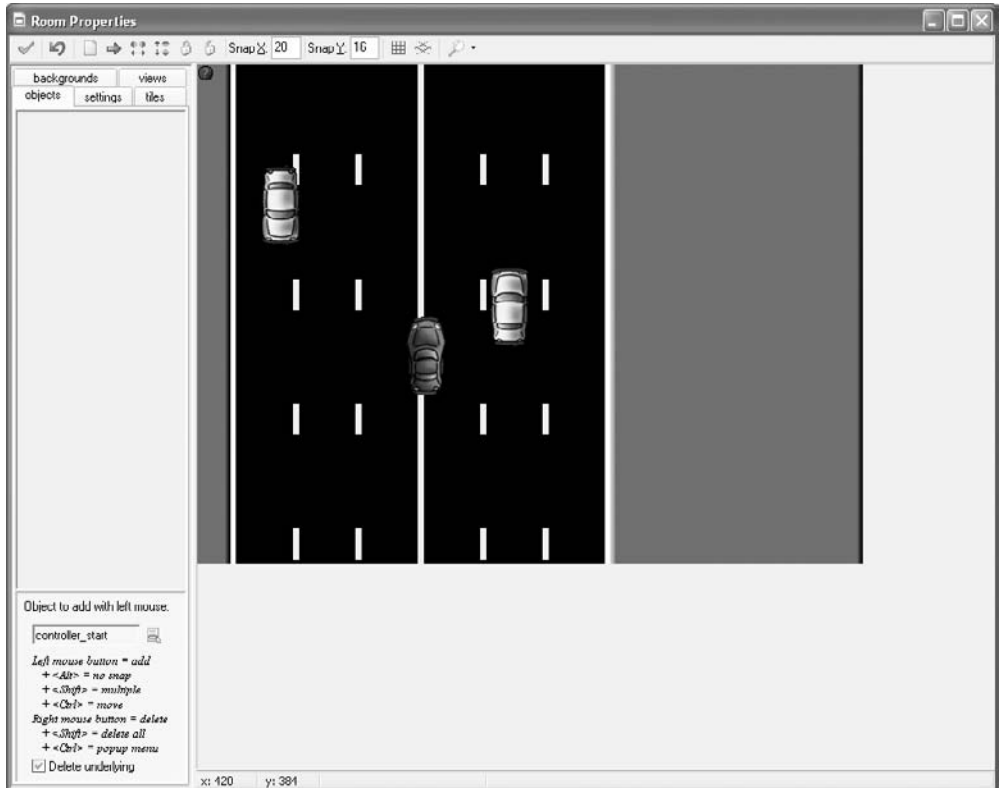


FIGURE 7.1 A room for the racing car game included with Game Maker 7.0.

Objects

All rooms contain objects, which include anything used in the room. For instance, the objects in the previously mentioned racing car game would include items play-

ers control, such as the car; stationery objects, such as a health meter; and computer-controlled objects, such as the other cars.

Sprites

For objects to appear on the screen, they must have an associated image. In Game Maker, you'll use sprites for this purpose. Sprites generally are composed of many separate images. Figure 7.2 is an example of a sprite made up of several sprites.

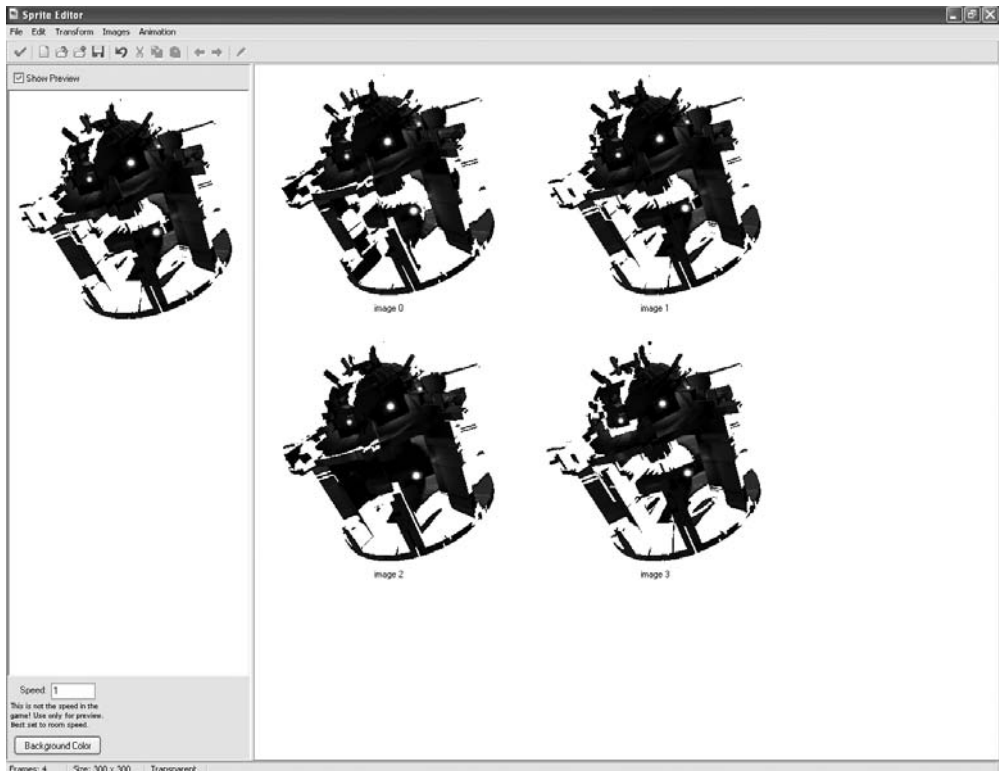


FIGURE 7.2 Sprites made up of several images.

Events

As sprites move around the room, things happen to them. For example, they can collide with walls and other objects. These encounters are called *events*. Events also include user actions, such as when a player clicks on an object. These events allow us to create games in Game Maker.

When an event occurs, you as the developer can establish what happens. For instance, if two objects collide, you can make them rebound. You can also create an object for which you can set its speed and direction, or even have it play music.

CREATING A SIMPLE PROGRAM

You'll now create a simple program that will show you some of the elements involved in creating a game for Game Maker. Start Game Maker. You'll see a screen like the one in Figure 7.3.



The program you'll create in this chapter is available on the companion DVD. Find the first.gmk file in the GMFILES\Game 1 folder. You can open it and look through it, or follow along by creating the project yourself.



ON THE DVD

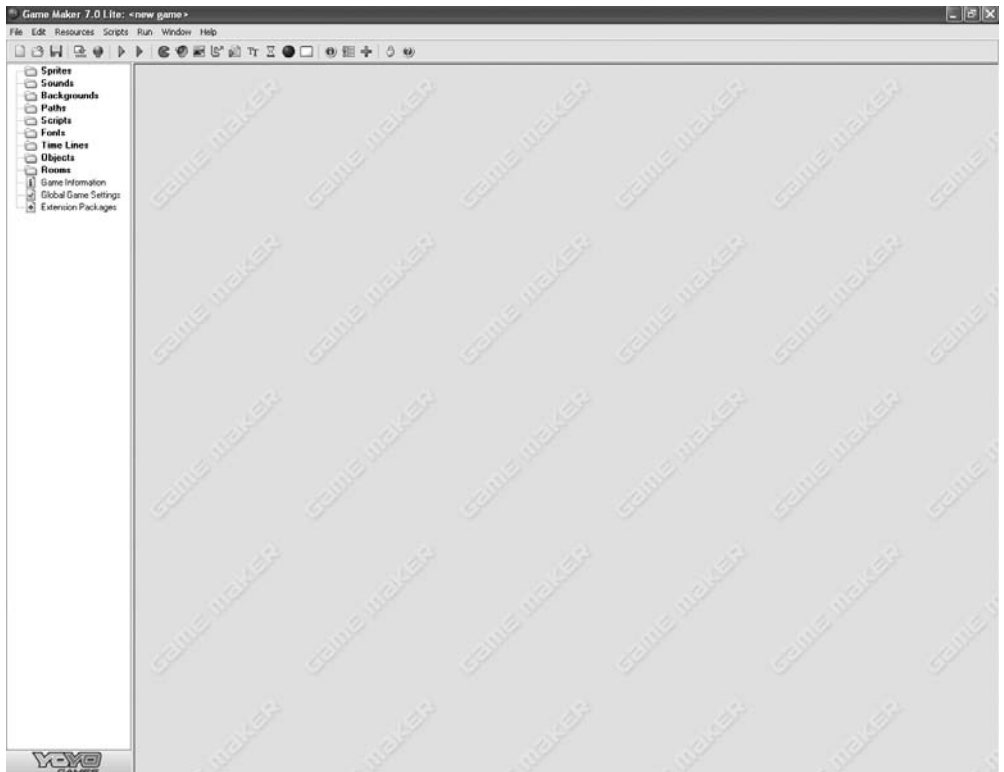


FIGURE 7.3 Game Maker displays an empty project when it first opens.

Next, you'll create the objects that will make up your project. Here's how.

1. Right-click on Sprites in the Resource Explorer. You'll see a menu like the one in Figure 7.4.

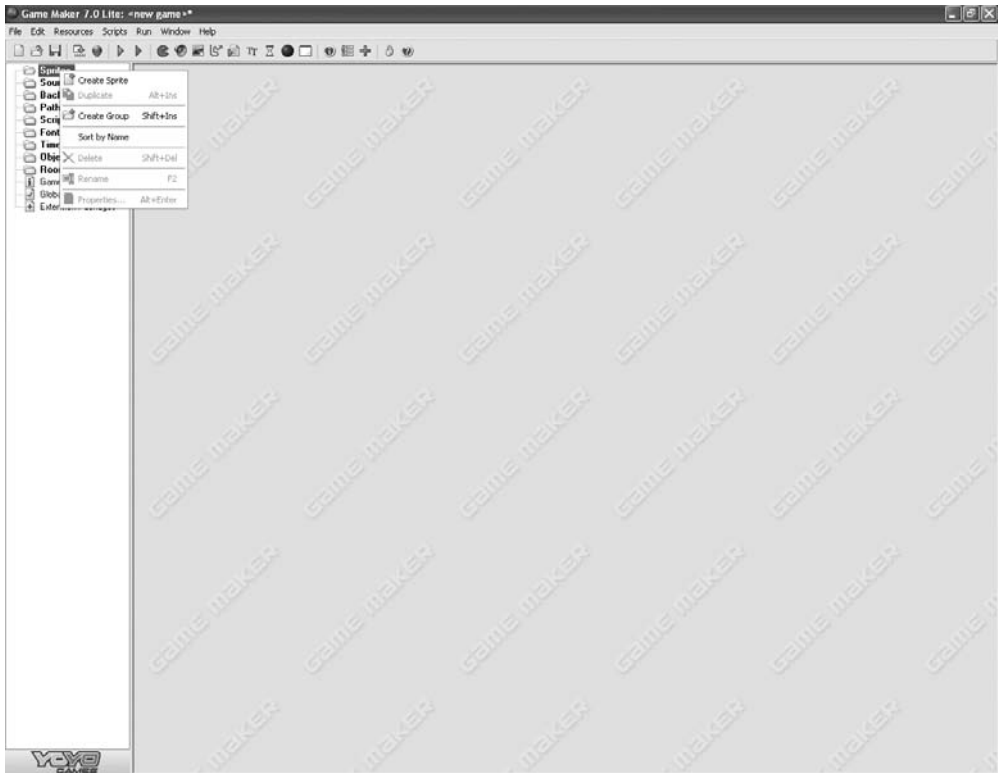


FIGURE 7.4 Right-clicking a resource allows you to add it to the project.

2. Select Create Sprite from the menu. You'll see the Sprite Properties window, which looks like Figure 7.5.
3. Click the Load Sprite button. You'll see an Open dialog box, as shown in Figure 7.6.
4. From this window, choose the GMFILES\Game 1\Player ship folder. Select `playership_10.png` from the list of files. This will display the file in the Sprite Properties window. Click OK to close the window.
5. Right-click the Objects resource, and then choose Create Object from the pop-up menu. This displays the Object Properties window (Figure 7.7).
6. Click the <no sprite> button beneath the Object Properties window label. This lets you select a sprite for the object, so you can use it in the project.
7. Select <sprite0> in the pop-up menu that appears.



If you delete a sprite and then create a new one, the number system will increment by one. For example, if you open Game Maker, create a sprite, and then delete it, the next sprite you create will be sprite1.

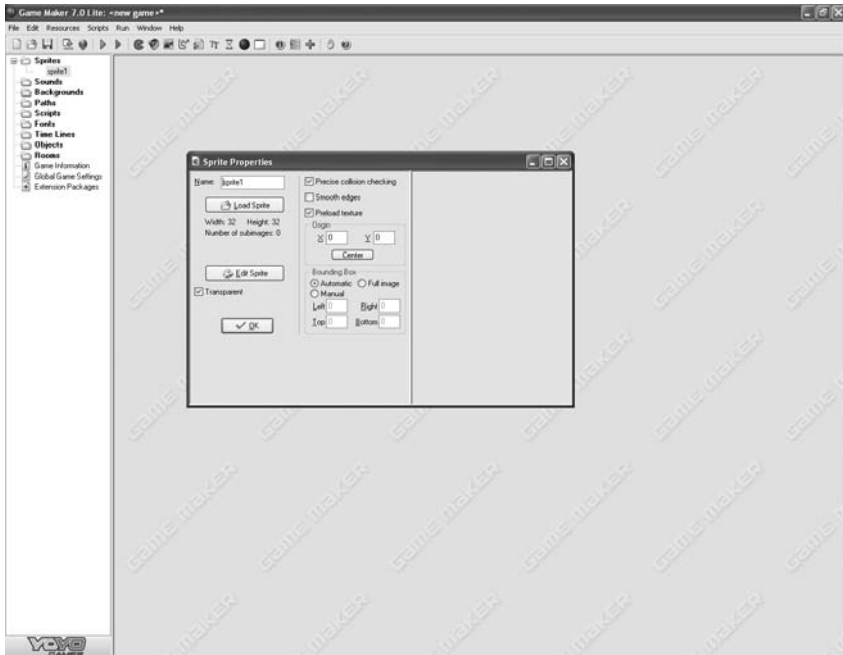


FIGURE 7.5 The Sprite Properties window allows you to create or alter a sprite.

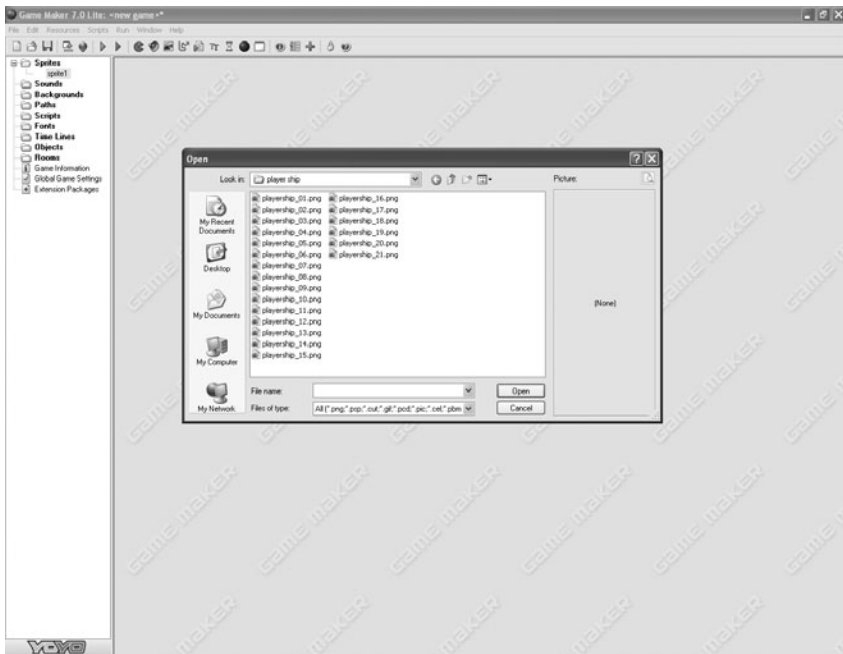


FIGURE 7.6 This box allows you to select the sprite you need for your project.

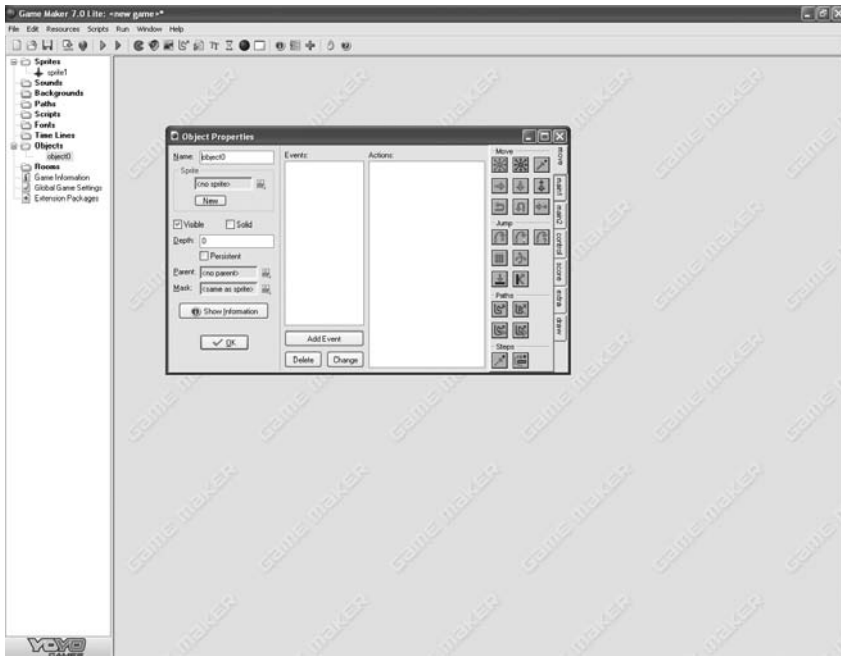


FIGURE 7.7 The properties for the object.

8. In the Object Properties window, click Add Event. This will display the Event Selector dialog box as shown in Figure 7.8. This allows you to program an *event*. An event is when “something happens”; in this case, we want to check when the player is pressing the left or right arrow keys.



FIGURE 7.8 The Event Selector.

- Click Keyboard, and a selection of key presses will appear, as shown in Figure 7.9.

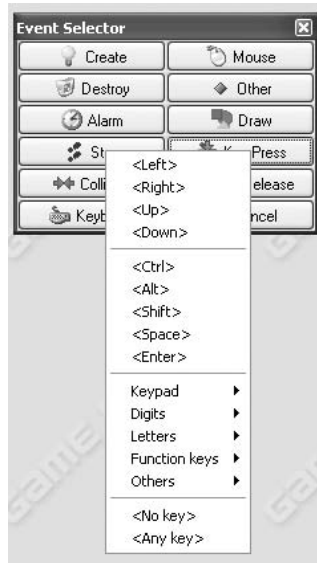


FIGURE 7.9 The different key presses you can program.

- Select the <Left> option.
- Locate the toolbar at the far right. In the first row of the toolbar is an icon with red arrows pointing in every direction. Drag this icon onto the window immediately to its left. When you let go of the button, a dialog box will appear. Click on the left pointing arrow and type the speed as 8 (Figure 7.10).
- Click OK, which will return you to the Object Properties window.
- Do the same process again, but this time select the event of the <Right> key press, and configure the movement to the right at a speed of 8.
- We'll now set the bounce action that will occur if the object attempts to move outside the game area. Click Add Event, and then select Other | Intersect Boundary.
- Drag the object with red arrows pointing in all directions into the Actions box to the left of it.
- Click Square in the middle of the Move Fixed dialog and ensure the speed is set to 0. Click OK.
- Click OK again to close the Object Properties dialog box.

Now that you've created an object and given it properties, you'll create a room and put the spaceship object in it. Here's how.



FIGURE 7.10 This window lets you give an object a direction when the game first opens.

18. From the Resource Explorer (remember, it's at the upper left), right-click on Rooms and then select Create Room from the pop-up menu. You'll see the Room Properties window, as in Figure 7.11.

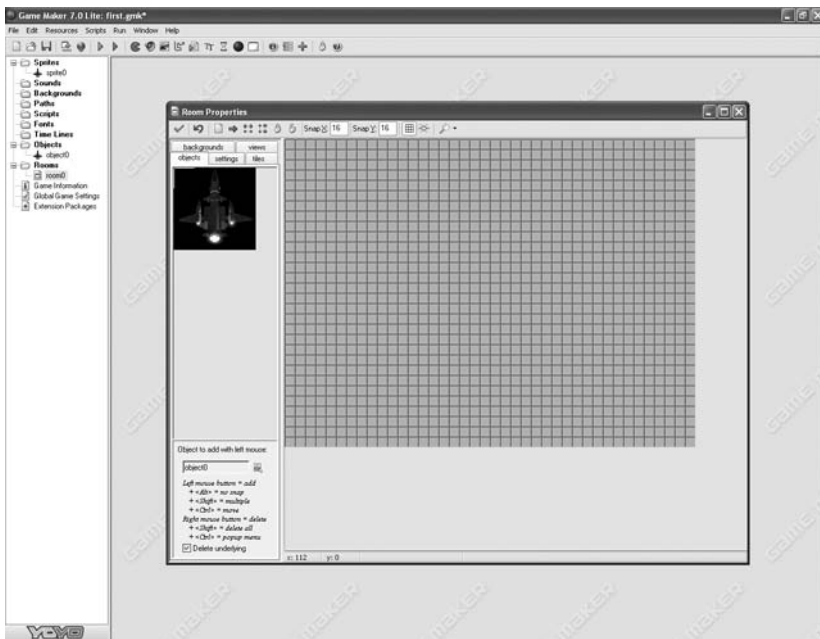


FIGURE 7.11 The Room Properties window lets you set up a room for your project.

19. Now, you'll set up the background for the room. Click Backgrounds. In the new window, click Background Color and then choose a light-blue color. Click Objects to return to the previous window.
20. At the bottom of the window, is an object selection box. Notice that the box currently has Object0 selected, which is the object you created earlier. In games with more objects, you will have to select the one you want as the active object.
21. Click once at the bottom area of the grid to place the spaceship. If the spaceship isn't fully on the screen, right-click on the object you placed to delete it. Keep trying until your spaceship is displayed similar to Figure 7.12.
22. Click the Green tick graphic to save your graphic to the room.

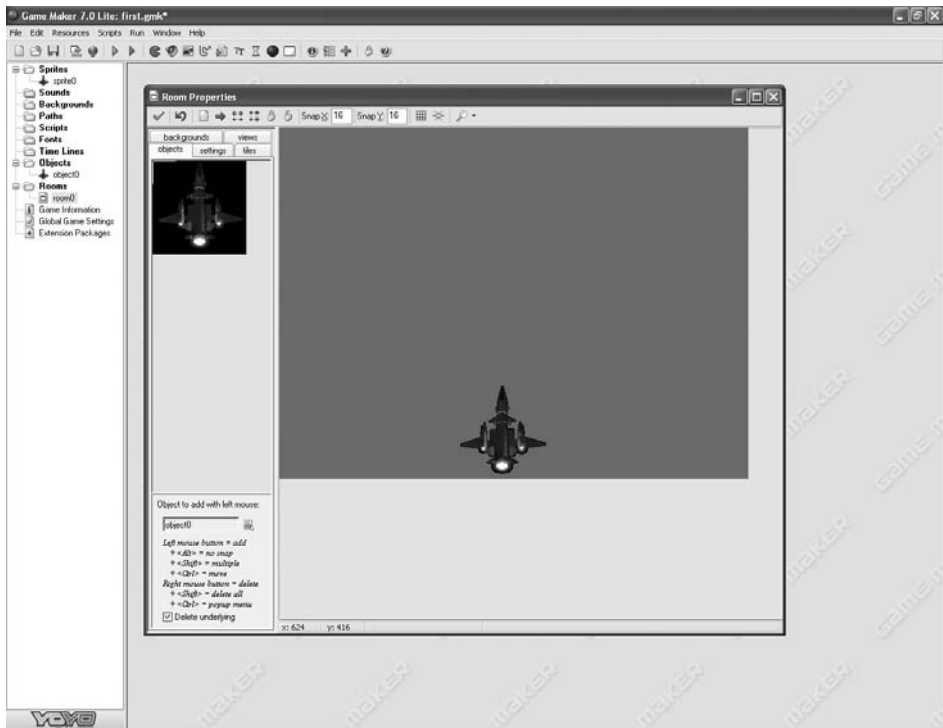


FIGURE 7.12 Spaceship on the room grid.

SAVE AND RUN

Congratulations! You've finished your first Game Maker program! It's a very simple spaceship you can move left and right of the screen. You also coded it so it does not disappear off screen. Although it's a simple game, creating it taught you much more about Game Maker—which will make future game development much easier.

It's time to save and run the game. Here's how.

1. On the File menu, choose Save. Give the file a name, and click Save. Game Maker will automatically add the extension gmd to the name you choose.
2. On the toolbar, click the green triangle, below the Run menu item. Your game will start running. You'll see a screen like Figure 7.13.

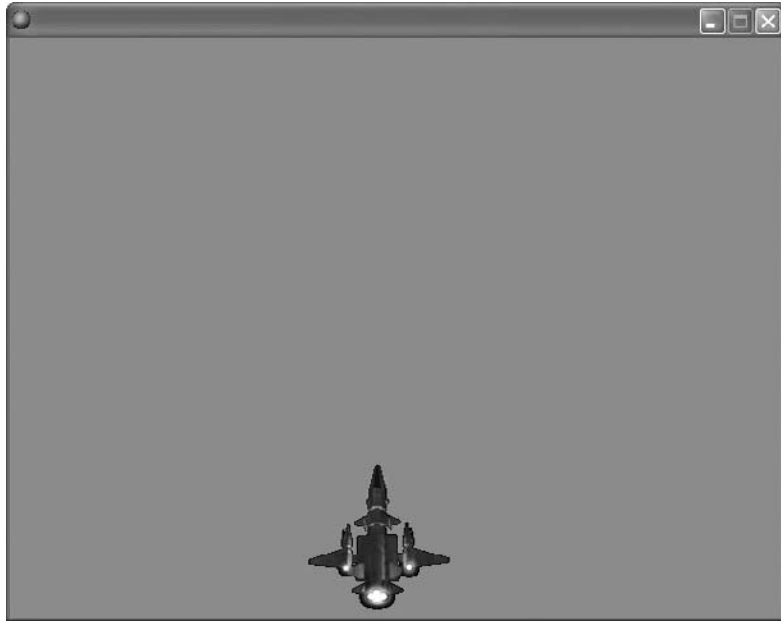


FIGURE 7.13 Running the game.

CHAPTER SUMMARY

Now that you have successfully developed a simple program, you have a good basis for future work. In this chapter, you learned about a variety of very important items, such as the resources that make up a project (rooms and objects), and how to put together a basic project. In the next chapter, you'll create a space shooter using Game Maker.

This page intentionally left blank

2D SPACE SHOOTER— END OF THE EARTH

In This Chapter

- Setting Up the Game
- Programming Objects
- Adding Sound Using a Script
- Adding a Help File
- Creating an Executable File



This chapter expands on what you learned when creating your first Game Maker project in the previous chapter. Here, you'll add quite a few additional steps and create an exciting game:

- A menu screen
- A game screen
- A ship that fires bullets at a pre-determined time
- A space backdrop
- Two buttons that will allow navigation through the game
- The playing of music
- The creation of asteroids
- A scoring system
- A health and lives system

You can see an example of the final game in Figure 8.1.



FIGURE 8.1 The game you'll create in this chapter.

The story behind the game:

The year is 2171 and Earth is on a collision course with a number of asteroids. You have been dispatched as Earth's last hope to destroy the asteroids and save it from total destruction.

Can you save the planet in time?

The game is a simple asteroids type game, where you will fly a ship left and right on the screen, trying to navigate past floating asteroids. You will be able to shoot, but you have to time each shot perfectly, as you cannot shoot every second. The

aim of the game is to destroy as many of the asteroids as you can before losing three lives and ending the game.

The game is separated into two screens: the menu screen where you can select to play the game or quit, and the screen where you play the game.

SETTING UP THE GAME

To begin, start the Game Maker software. This will create a new project automatically, and should look something like Figure 8.2.

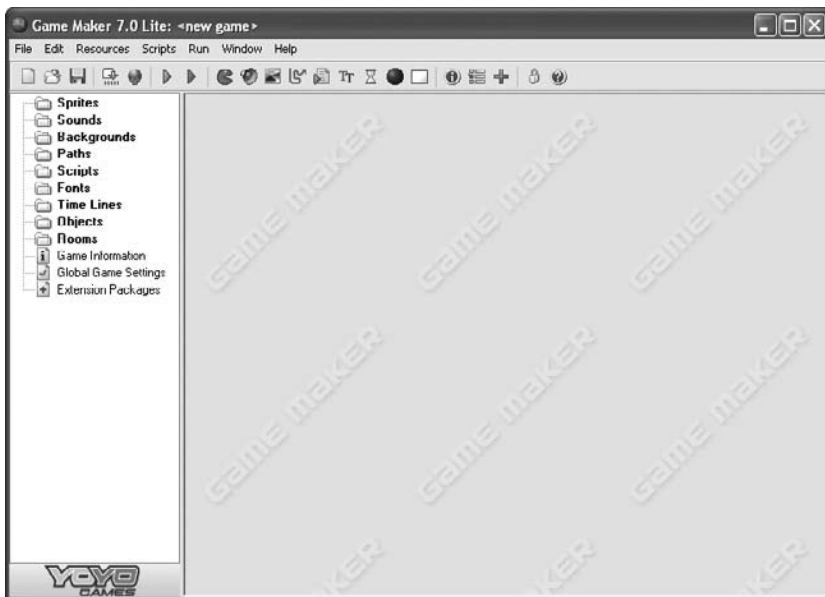


FIGURE 8.2 Upon opening, Game Maker automatically creates a new project.

Sprites

You won't need to design any of the graphics used in this chapter, as they have been made by a graphic artist for this particular game. This saves time and allows you to concentrate on learning the product and how to make games. You will use a number of sprites in this game, including the ice asteroids, the player's ship, a bullet, an interface bar, and other items.

Making the First Sprite

1. To add a sprite, click Create Sprite from the Resources menu as shown in Figure 8.3. This opens the Sprite Properties menu, as shown in Figure 8.4.

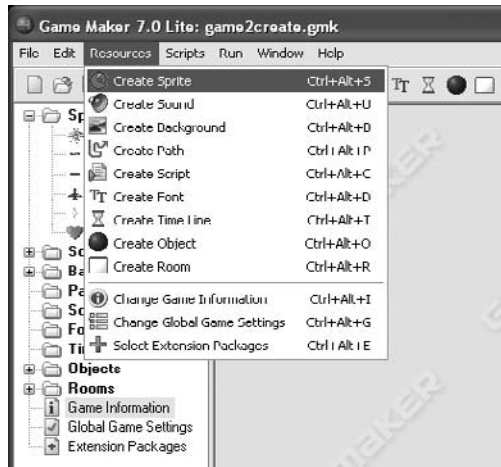


FIGURE 8.3 The Create Sprite option from the Resources menu.

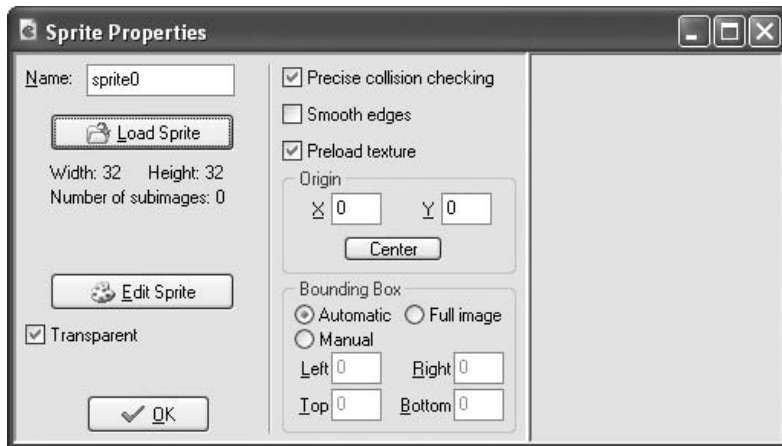


FIGURE 8.4 You'll see the Sprite Properties window when you click Create Sprite.

2. Click Load Sprite. An Open File dialog box will appear, as shown in Figure 8.5.
3. Navigate GMFILES\Game 2\items\icerock folder on the companion DVD. This folder contains all the images for your ice asteroid. There are 48 images, which allow for the object to be animated, and in the game, the ice asteroid will rotate as it's moving through space.
4. Click the icerock_01.png file, and then click Open. This will import the graphic image into the Game Maker Sprite Properties window. You will now see the ice asteroid as shown in Figure 8.6.



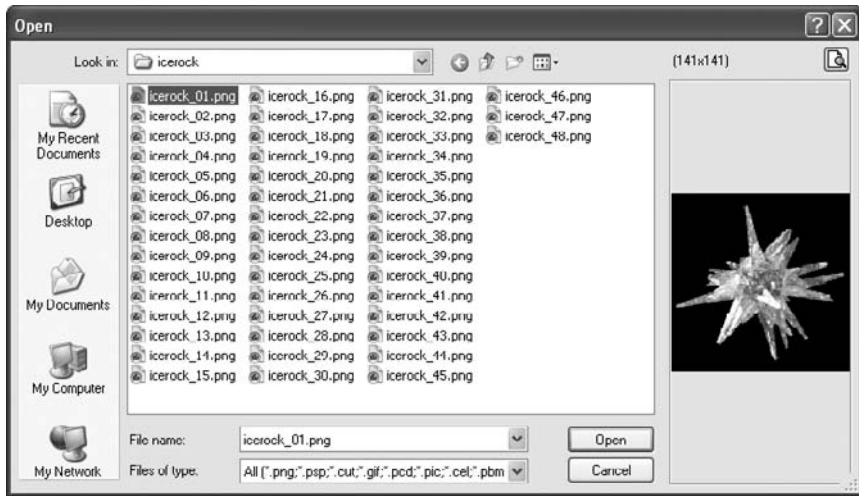


FIGURE 8.5 An Open File dialog box allows you to add graphic images to your project.

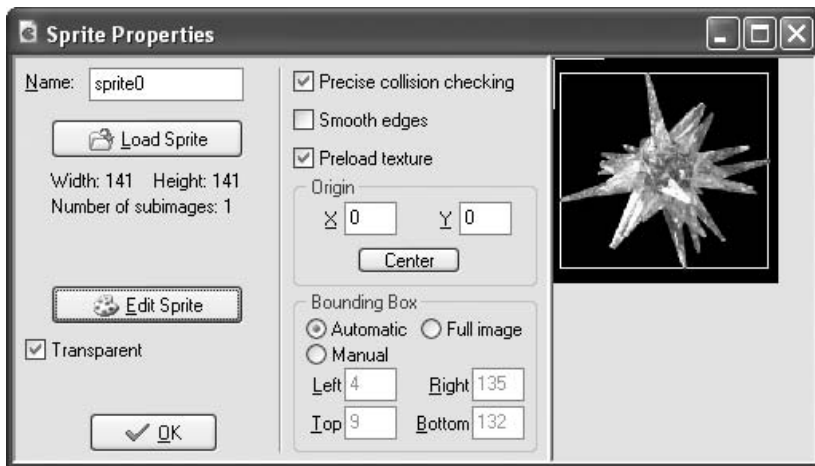


FIGURE 8.6 The image in Game Maker.

5. The ice asteroid contains 48 images to represent its spinning animation, and currently you have only imported a single image. Click Edit Sprite to enter the Sprite Editor as shown in Figure 8.7.
6. Click Add Sprite From File, which will display the Open dialog again. This time, you will be able to select multiple images, so pick from icerock_02.png to icerock_48.png. Single left-click on icerock_02.png, hold down the Shift key, and click icerock_48.png.

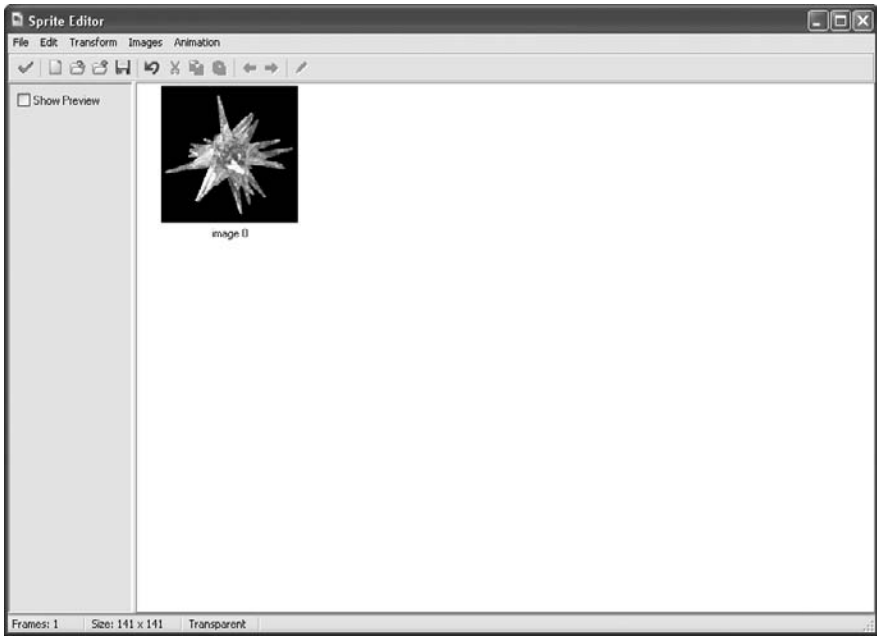


FIGURE 8.7 The Sprite Editor with your single sprite displayed in the dialog window.

7. Click Open, and after a couple of seconds, you will then see all the animations inserted into the Sprite Editor as shown in Figure 8.8.
8. To save these animations within the sprite, click the green tick icon.

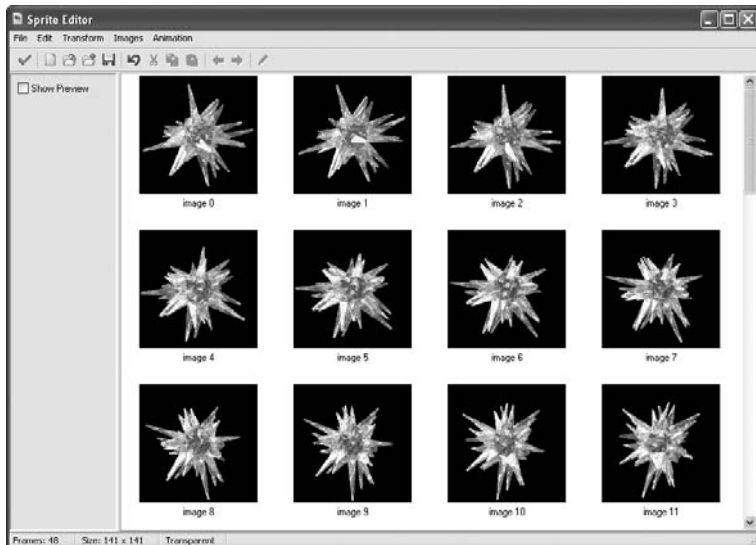


FIGURE 8.8 All the animations for your ice asteroid now imported.

- You will now be back at the Sprite Properties window, and now it's time to change the name of the sprite (currently called `sprite0`) to something more appropriate. Highlight the `sprite0` text and type `Ice_asteroid`. Click OK to close the Sprite dialog.

Making the Rest of the Sprites

You now need to insert a few more sprites. Figures 8.9 through 8.13 show the properties for each sprite so you can compare them when you import them. See Table 8.1 for more information on what you need to import.

Table 8.1 Sprites You Need to Import for Your Game

SPRITE NAME	GRAPHIC FILE NAME AND LOCATION
Play_btn	DVD\GMFILES\Game 2\buttons\buttonplay_1.png
Quit_btn	DVD\GMFILES\Game 2\buttons\buttonquit.png
Player_ship	DVD\GMFILES\Game 2\items\ship\shipcentral.png
Bullet	DVD\GMFILES\Game 2\bullet\shot1_01.png
Lives	DVD\GMFILES\Game 2\Lives\livesbutton.gif



ON THE DVD

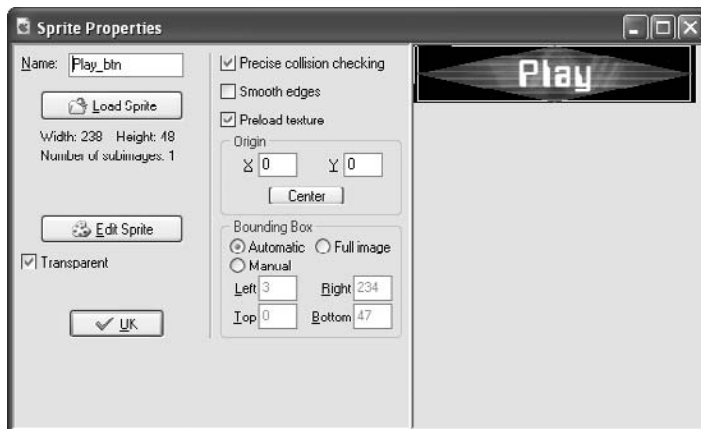


FIGURE 8.9 The Play button properties and image.

To create the other sprites, use Table 8.1 as a reference. Follow the steps you used earlier to create the ice asteroid and make the following changes where required:

- Create the sprites using the file location in Table 8.1.
- Rename each sprite entry to that in Table 8.1.
- For the `Player_ship`, you will need to import all three images; image 0 will be the ship flying straight ahead, image 1 will be it flying to the left, and image 2 will be it flying to the right.

- For the bullet, you only need to import one image, shot1_01. Notice that there are many images for this object; you can if you want get an animated bullet, but for this game it is not required.



There are more images for the game in the GMFILES\Game 2 folder for you to extend the game if you feel like it.

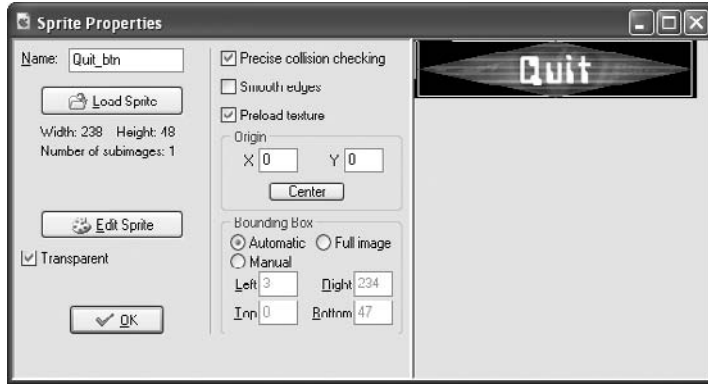


FIGURE 8.10 The Quit button properties and image.

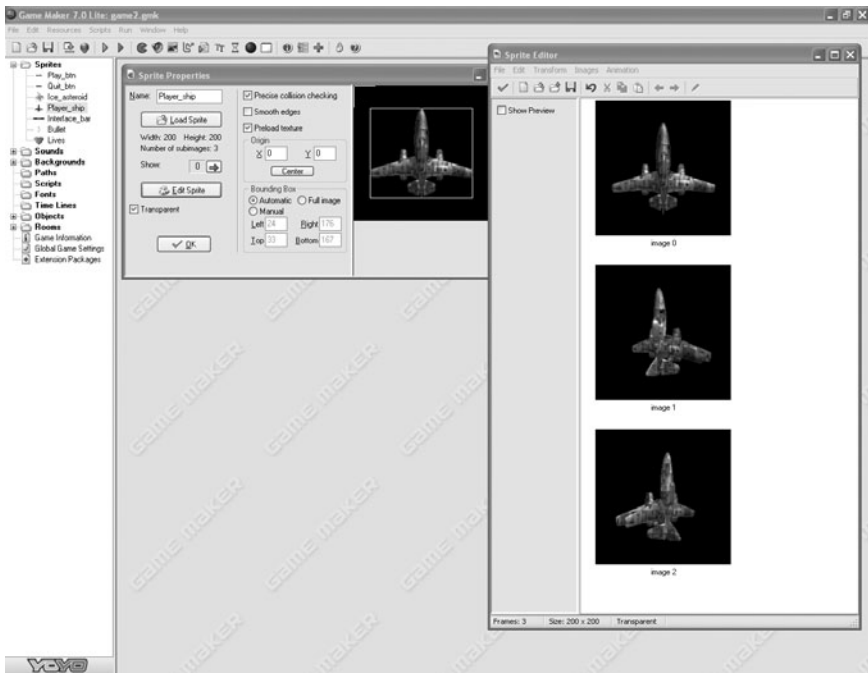


FIGURE 8.11 The player's ship properties and its animation frames.

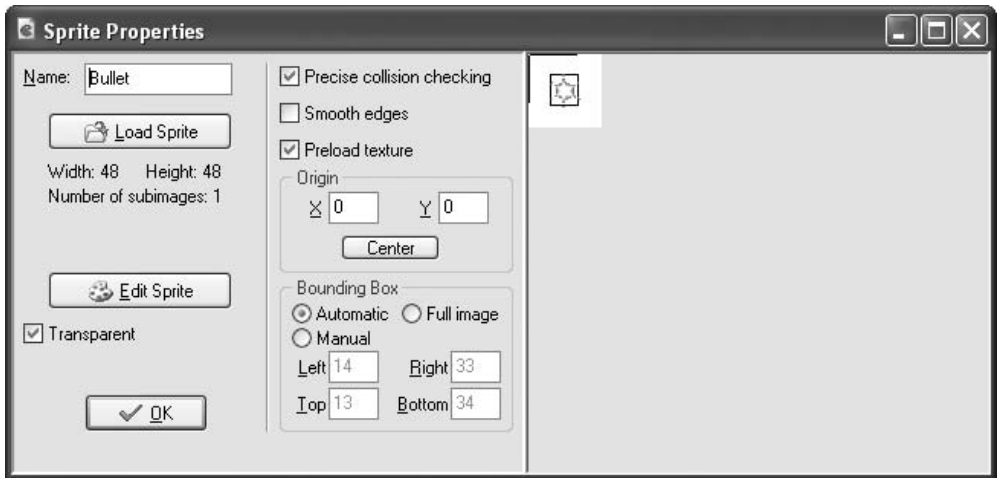


FIGURE 8.12 The bullet properties and image.

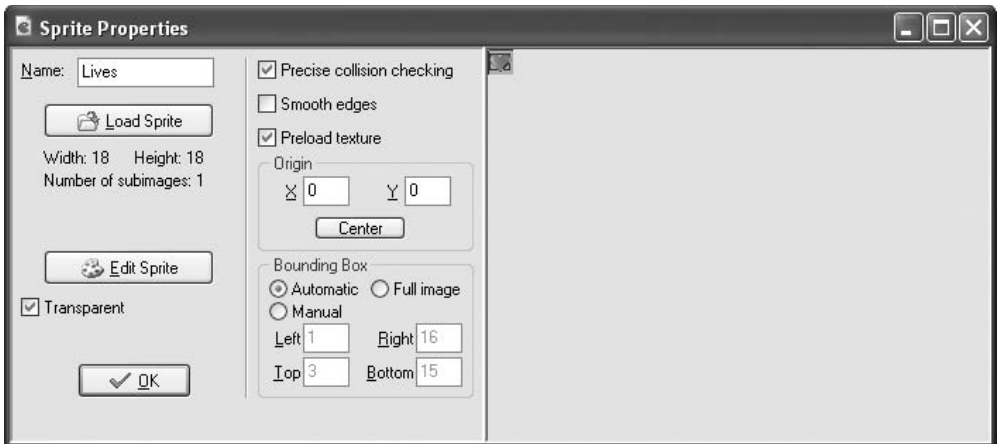


FIGURE 8.13 The lives properties and image.

After completing all the imports, you will be able to see the list of items in the left-hand windowpane of Game Maker. This list of items should match up with Figure 8.14; if it does not, go back, and make sure you imported all the images.

Sounds

It's now time to add the music that will play in the game. The game consists of two screens (called rooms in Game Maker); the first where the player clicks a button to start the game, and the second is the game.

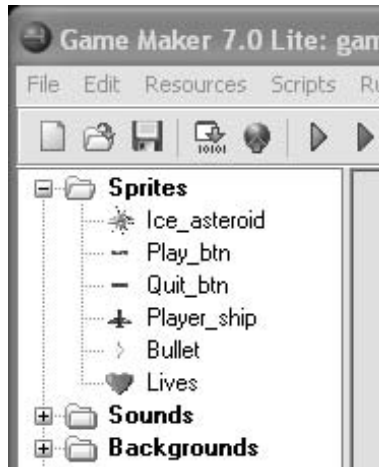


FIGURE 8.14 All the sprites imported into Game Maker.

Music will play in both rooms.



The music file in this game is from The Games Factory 2 library of files. Many thanks to Click-team for this resource.

You could add more sounds to the game if you wish to expand it further.

1. Choose Create Sound from the Resources menu, as shown in Figure 8.15.

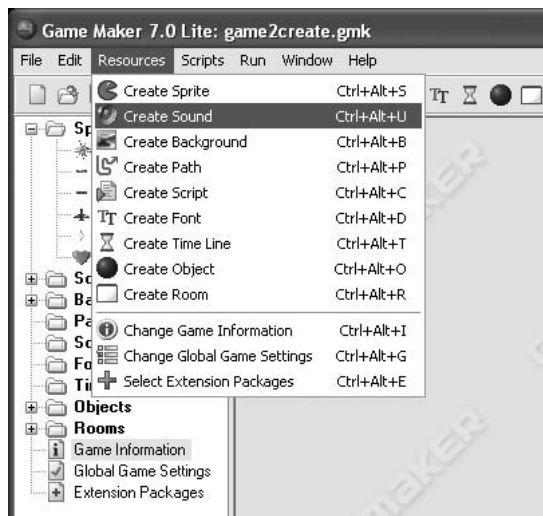


FIGURE 8.15 The Create Sound option from the Resources menu.



2. In the window that appears, click the “Load sound” button. This displays an Open dialog box that lets you choose a sound file for the music you will play.
3. Select the path DVD\GMFILES\Game 2\Music, select the Lastday.wav file, and click Open.
4. The music file will now load as shown in Figure 8.16. You can click on the green arrow button to play the music and listen to it to check the file.



FIGURE 8.16 The Sound Properties dialog with the wav file loaded.

5. Rename the sound element to “Music.”
6. Click OK to close the dialog.

Backgrounds

Now you need to add two backgrounds, the first for the main menu and the second for the background image for the game.

1. Choose Create Background from the Resources menu.
2. In the Background Properties dialog as shown in Figure 8.17, click the “Load background” button.
3. The Open dialog box will appear, browse to the location DVD\GMFILES\Game 2\Backgrounds, select Background1.png, and click Open. Change the name of the object to “Main_Menu,” and then click OK to close the Properties dialog.



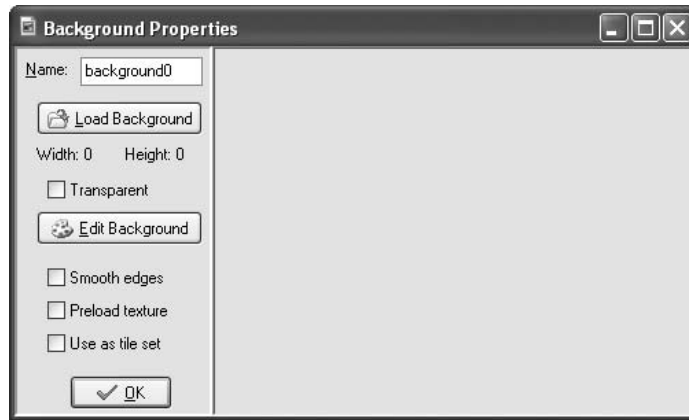


FIGURE 8.17 The Background Properties dialog.

4. Follow the same process for the second background image you need to import. You'll find it in the same folder; the name is "Background2.png" and should appear as "Game_Screen" in the Properties window.

Creating Objects

Now that you have your sprites and background, you need to turn your sprites into objects, with which Game Maker can then interact. This creates many possibilities, including making objects move (either on their own or using the mouse or keyboard), or you can test for collisions or many other different options.

1. Choose the Create Object option from the Resources menu option. This brings up an Object Properties window that looks like Figure 8.18.

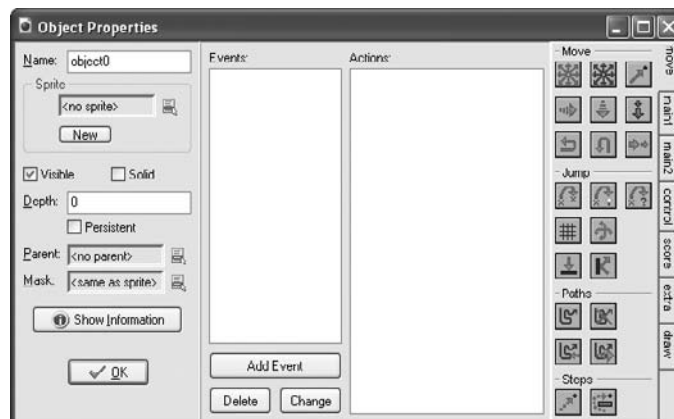


FIGURE 8.18 The Object Properties window.

- Notice that the object's current name is object0, and under sprite it is <no sprite>. You will add the play button as an object, so enter the name Press_Play; under sprite, click on the drop-down box and select the Play_btn sprite as shown in Figure 8.19.

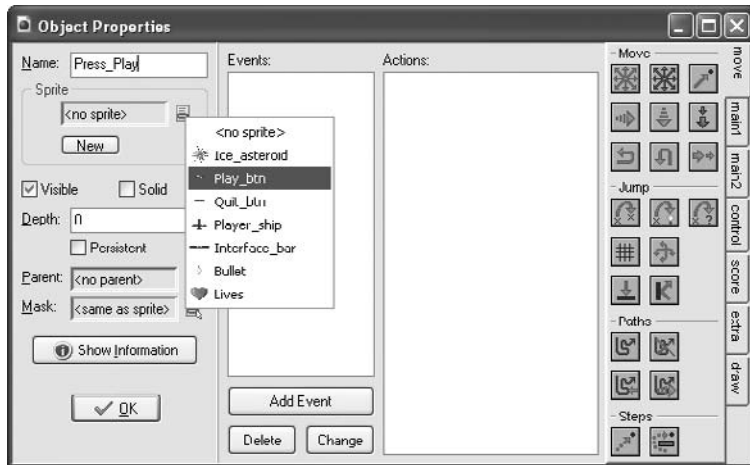


FIGURE 8.19 The Object Properties window with sprite selection enabled.

- You'll come back into the Object Properties dialog to program your objects, but for now, click OK.
- What you just did is assign an object called Press_Play to the sprite Play_btn, so you will be able to program this object and it will directly affect the selected sprite within the game.

You now need to do the same process for the rest of the sprites, and add two objects that won't have sprites assigned to them. See Table 8.2 for details of the additional items to add in the object section.

Table 8.2 Additional Objects to Add to Game Maker

NAME	SPRITE
Press_Quit	Quit_btn
Ice	Ice_asteroid
Player	Player_shup
Bullets	Bullets
Health	<no sprite>
Lives	<no sprite>

Your Game Maker windowpane will now look like Figure 8.20.

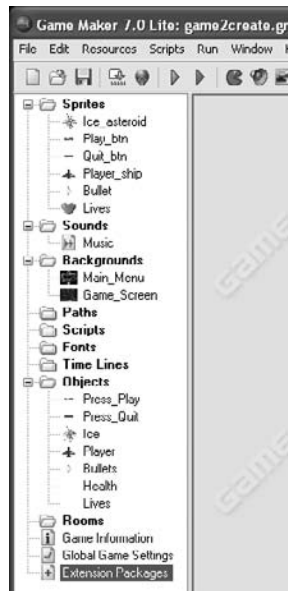


FIGURE 8.20 The current state of the left-hand pane in Game Maker.

Rooms

You will place your sprites/objects in rooms, where you will set out your game graphically. For this game, you need two rooms—Main and Game. This gives you a description of what each of the two rooms is going to be doing. In a bigger game, you could call them by level names or number them.

1. Select Create Room from the Resources menu option.
2. A blank room will now appear on screen, in a grid, as shown in Figure 8.21.
3. By default, each room is named Room, and a number will be put on the end. The first room you will create will be Room0, then Room1, and so on. Create the two rooms, and ensure you are on Room0.
4. Now, change the names of both rooms to the more appropriate names, Main and Game. Click on the Settings tab for Room0 and change the name to Main.
5. While on this tab, you also need to change the screen size to 800 × 600. Type 800 in the width, and 600 in the height section. The grid will resize automatically.
6. You can see these changes in Figure 8.22.

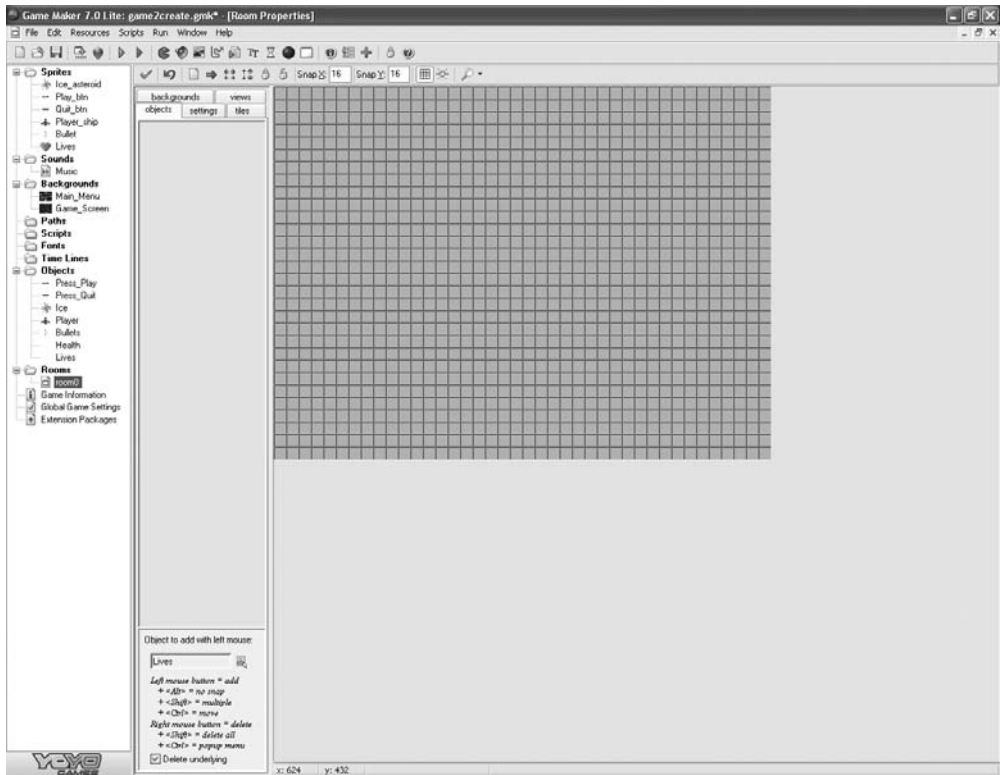


FIGURE 8.21 Blank room in Game Maker.

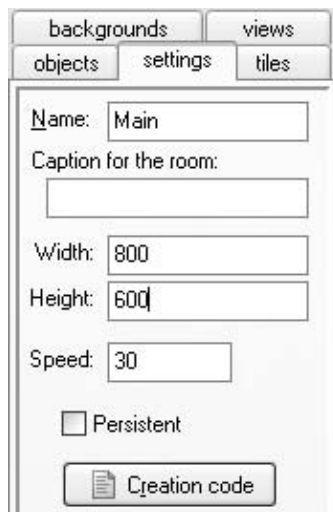


FIGURE 8.22 The Settings tab for Room0.

7. Click on the green tick to save the room details and then make the name change to “Room1” to Game and change its screen size to 800 × 600.
8. You now have your two rooms, but you have no items on the screen. So, it’s time to place the images that will make up the scene, and the other objects that handle health and lives.
9. Make sure the Main room is currently displayed, and then click on the Backgrounds tab.
10. You will notice there is a box that reads <no background>; this is where you can select the objects that are stored in the program for the backgrounds you added earlier.
11. On the Main room, you need to drop the background first and then the two buttons, so click on the drop-down box, and select Main_Menu. This will automatically place this object on the screen as shown in Figure 8.23.

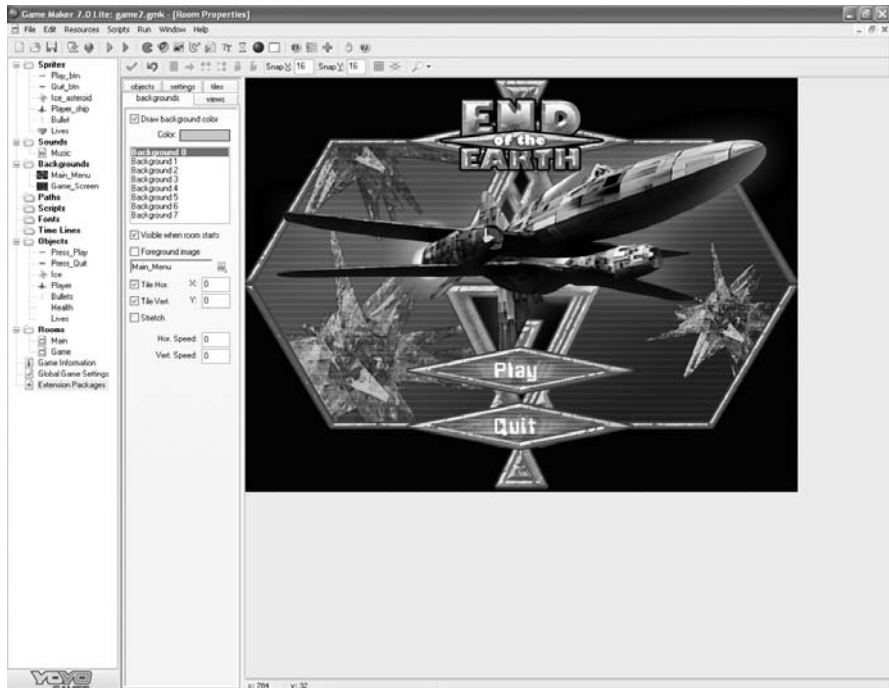


FIGURE 8.23 The Main_Menu background selected and displayed in the Room Main.



In Figure 8.23, the grid option for the actual picture for the book is turned off so it looks good in print. Although you don’t need the grid for the background image that covers the whole room area, you would use the grid for placing the other objects, as it allows for more accurate placement of those objects. The grid is a large number of small boxes across the whole room. If you wish to turn it off and on, you can use the “Toggle the showing of the grid” option in the menu bar.

12. You now need to add the two buttons in a specific location, so click on the Objects tab. At the bottom left of the Objects tab you will see a drop-down menu and some instructions on how to place an object onscreen. Click on the drop-down and select the Press_Play object.
13. This object will appear on the top of the objects frame ready for you to place on the room. Left-click anywhere on the room to add the object.
14. You can now carefully place the object. Hold down the Ctrl key and left-click on the room; this will allow precise movement of the object. Place it over the “Play” text.
15. Now select the drop-down box again and choose Press_Quit. Again, left-click on the Room, and then use the Ctrl key to place it over the “Quit” text.
16. Left-click on the green arrow to save the information.
17. You now need to place the background and the objects for the Game room.
18. Double left-click on the Game room in the left-hand windowpane.
19. The blank room will now appear.
20. Click on the Backgrounds tab, and then from the drop-down menu (where it says “<no background>”), select Game_Screen.
21. Click on the Objects tab.
22. Now you’ll place five ice asteroids on the screen, so click on the drop-down box and select Ice. Then, place the objects at random positions on the screen (see Figure 8.24 for an idea of where you should place them). Remember, you may need to hold down the Ctrl key to move the items into a better position.

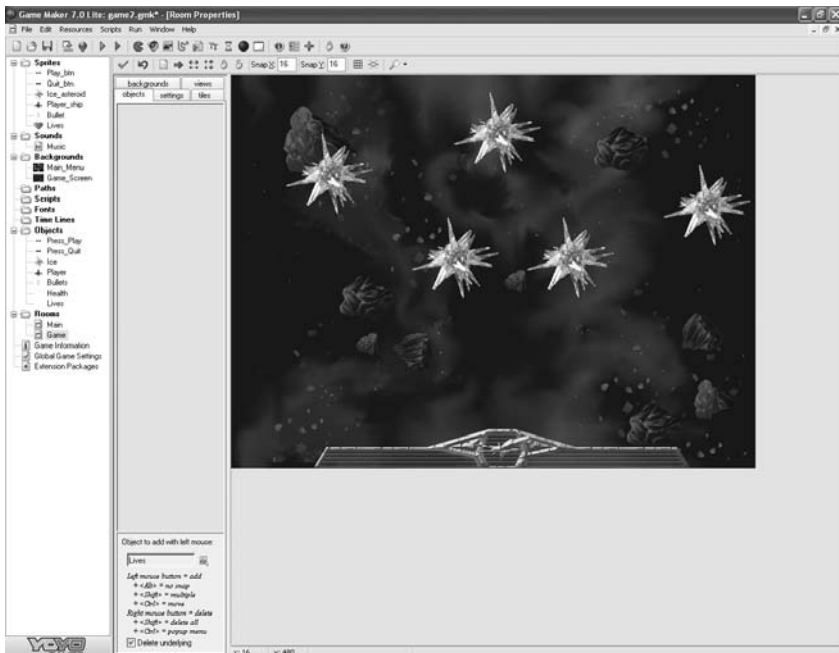


FIGURE 8.24 The Ice objects spread around the Game room.

23. Now you need to place the space ship, so click on the drop-down box, select Player, and then single left-click on the room. Use Ctrl and the mouse to position it at the bottom center of the screen, just above the interface bar as shown in Figure 8.25.



FIGURE 8.25 The placement of the player's ship in the room.

24. There are two items left to add to the room, and these are placeholder items: the Health and Lives items. You will draw these within the eventing, as they require special treatment; for example, reducing the health bar and removing lives is covered within the event system.
25. From the drop-down box, select Health, and then place this on the left-hand side of the control panel, in the location X160 and Y576. This will appear as a blue circle and a question mark.
26. From the drop-down box, select Lives, and then place this on the right-hand side of the control panel, in the location X544, Y576. Your interface bar will now look like Figure 8.26.



FIGURE 8.26 The interface bar with the health and lives markers.

You have now placed all the room objects and can move on to creating the events to your game so it is playable.

PROGRAMMING OBJECTS

Now that all the resources are in place, you need to tell Game Maker what to do with it all; otherwise, nothing will happen. You do this by going back into the objects and applying events and actions that will detail what is “happening,” and then “what to do about it.” For example, an event might be “User presses a key” and the action would be “Move left.”

When creating your events, it is easier if you start on a particular room and then work through them. If you become stuck on the events and actions for a certain room, you can move on to the next and come back to it later.

Navigation Buttons

The first task is to program the events for the Main room, which has two buttons: play and quit. When the player clicks on one of the buttons it will either jump to the Game room or quit the application. You also need to consider a couple of other events; for example, you want the button to animate when the player moves the mouse over it, as this shows the player that it is clickable.

The Play Button in the Menu Room

The first event you’ll create is when the player presses the left mouse button on the play graphic; the action will be to go to the next room

1. Double left-click on the Press_Play object to access the Properties sheet.
2. Click the Add Event button; this will display the Event Options box.
3. In the Event Selector dialog, select Mouse | Left button. You will now see the single event. The action will be to move to another room, so you need to access the Room actions, which are stored under the Main1 tab. Ensure you are in the Main1 tab, and then you will see a set of six buttons under the Rooms heading. You need the icon second from the left, which shows an arrow pointing to the right—this represents moving forward a frame. If you hold your mouse over the icon, a small text tip help will appear telling you what it is.
4. Left-click, hold down the left mouse button, drag the Next Room icon, and drop it onto the blank actions area.
5. A Next Room dialog box appears, which has a drop-down box for a transition. This means you can add a special effect that will help the movement from one room to another. Click on the drop-down box and select “Fade out and in” as shown in Figure 8.27.

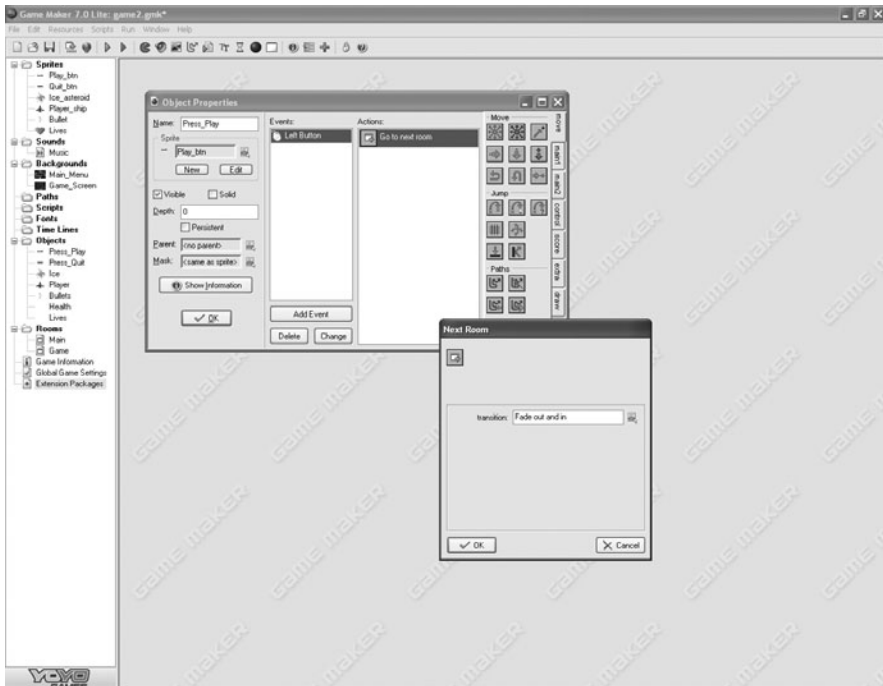


FIGURE 8.27 The Next Room Properties box with a transition applied.

6. Click OK to save the action to the event.
7. If you run the program, you will find that you can click on the play button and it will move to the next screen. Now, create two events that handle the animation of the button. To do this, you need an event that will check if the mouse cursor is over the play button, and when it is not. You need two checks because there are two states to the button—on and off—and by checking when the mouse isn't over the button, you can turn the effect off. First, you'll add the event for when the mouse moves over the play button.
8. Click on the Add Event button, and select Mouse from the Event Selector. Then, select Mouse Enter from the pop-up menu.
9. As you are working on a new event, the actions box will now be blank, ready for you to program it.
10. Within Game Maker are a number of key variables you can set on and off, and they will directly affect what happens in your game. One of those variables is "visible"; when it is true, an object will appear, and when it is set to false, it will be invisible. For the two events you are adding, you will use both to hide and show the button. Setting the variable in the action is a powerful way to handle certain actions that are not covered by the drag-and-drop event system.



There are a number of pre-defined variables available in Game Maker; consult the help documentation for more information.

11. The Variables options are available in the Control tab; click on it and you will see three possible options. You need to select the square with VAR in it; this is the Set Variable option.
12. The Set Variable dialog box will appear.
13. Type in the variable name “visible” and the value as false as shown in Figure 8.28. Click OK. Setting it to false will make the button visible when it is over the button area.

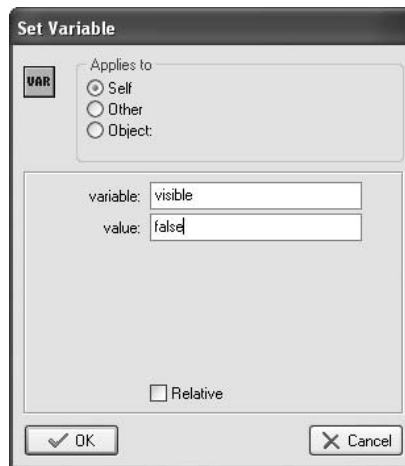


FIGURE 8.28 The Set Variable dialog box.

14. Using the same process, you need to add an event for the mouse leaving the play button and set its visible variable to true.
15. Click on the Add Event button, and select Mouse | Mouse Leave. Then, ensure you select the Control tab, and drag and drop the Set Variable into the Action windowpane.
16. Type the variable name “visible” and set its value to “true.” Click OK to save.
17. Test it now and you will see how it appears and disappears as you move the mouse over it, and away from it.

The Quit Button in the Menu Room

You now need to do a similar process for the Quit button, but rather than move to the next room, we need to quit the application.

1. If the Object Properties for Press_Play is still open, click OK to close it.
2. Double left-click the Press_Quit button in the Objects folder.

3. Click the Add Event button, and select “Mouse | Left button” in the Event selector and pop-up menu.
4. Click on the Main2 tab, and then drag and drop the End Game button (the second object under the game heading).
5. Click on Add Event, and select Mouse | Mouse Enter. Go to the Control tab, drag the Set Variable item, type in the variable name “visible,” and then set its value to “false.” Click OK.
6. Click on Add Event, and then select Mouse | Mouse Leave. Go to the Control tab, drag the Set Variable item, type in the variable name “visible,” and set its value to “true.” Click OK.
7. Click OK to close the Object Properties window.

Spaceship Events

Now you have the game moving from the main menu to the game level, so you can start to program the elements that make up your game. If you run the game now, you will notice you cannot move the spaceship, and both the ship and the ice asteroids are acting very strange (they are moving very quickly on the spot). This is because the program is automatically looping the animations at a default speed; once you begin to program the events to handle the movement, it will play correctly. As you are going to program the movement of the spaceship, you can also create the events and actions for the other things the spaceship will be involved in:

- Setting up the correct animations
- Setting up a variable to handle the weapon
- Collision with the ice asteroids
- Controlling its movement using the left and right arrow keys
- Pressing the space bar to fire the weapon
- Checking the ship’s health
- Checking if the ship is trying to leave the screen
- What to do at the start of the room

On Ship Creation

First, you need to create an event that will run some actions at the very start of the ship’s creation. This means that before the game starts, you can set up certain aspects of the object upon loading. The first is to set the sprite to a certain animation frame; in this case, we want it of the ship facing forward.

1. Double left-click on the Player object; the Object Properties window will appear.
2. Click on the Add Event button and then select the Create button.
3. You have your event, so now you need to add the action. You need to access the sprite object, so click on the Main1 tab, and then drag the first object under the sprite heading (looks like a red PacMan type character). The object is called “change sprite.”

4. A Change Sprite dialog box will appear. The first item is the sprite this is associated to; at the moment, this is “No sprite,” so you need to change this to the Player_ship sprite. The subimage is the image you want to display. If you remember, there were three animation frames for the ship: 0 for the forward facing image, 1 for moving left, and 2 for moving right. So, you can leave this as 0, and change the speed of the object to 0, as it won’t be moving at the start of its creation. The dialog should now look like Figure 8.29.

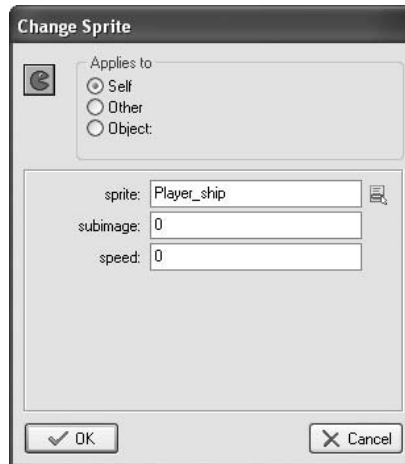


FIGURE 8.29 The Change Sprite dialog configuration.

You need to create another action as part of the “Create” event. This next action may not make much sense on its own, but when combined with some actions and events from other objects it should all become clear. You need to set up a variable called “gunshot” that will handle the frequency of the bullets fired from the ship’s guns. You could just create a simple event that when the player presses the space bar, the gun is fired. However, doing this makes the ship fire many bullets very quickly, which makes the game too easy to play. Therefore, to make it harder and give the player more of a challenge you will create an interval, so the gun fires slower. For this, you are going to need a variable to keep track of a number; when this number is set to 1, the gun will be able to fire, and when it is set to 0, it won’t. For now, you just need to say that at the very start of its creation the gun will be able to be fired; later in the player events, you will handle what to do with the variable.

5. Still under the Create event for the Player object, click the Control tab, and then drag and drop the Set Variable object onto the action area (the Set Variable object is the first button under the Variables heading).
6. When the Set Variable dialog box appears, type “gunshot” in the variable box, enter 1 in the value box, and then click OK. Your event and action should appear as shown in Figure 8.30.

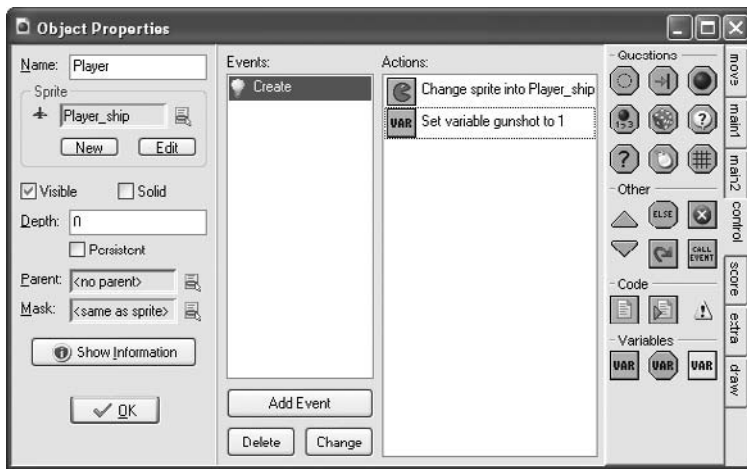


FIGURE 8.30 The Create event and its actions for the player object.

Room Start

Very similar to the Create event, the room start will configure the selected object when the room is first encountered. You would use the room start to configure particular variables and settings; in this case, you will be setting the health and the lives to the correct level. The lives will be set to 3 and the health to 100.

Setting the Health

1. Click on Add Event, and then select “Other | Room start.”
2. Click on the Score tab, and then drag and drop the Set Health object to the action box (the Set Health object is the first icon under the Health heading).
3. A dialog box will appear; enter the number 100, and click OK.

Setting the Lives

1. Still on the “Room start” event and the Score tab, drag and drop the Set Lives object to the action box. The Set Lives object is the first object under the Lives heading.
2. A dialog box will appear; type in the number of lives, which in this case is 3, and then click OK.
3. Your event and its actions will look like Figure 8.31.

Spaceship Movement

Now it’s time to program the events so you can control the movement of the spaceship to the left and right. You also need to add a movement event that checks for no movement. When making your own game, you can spot possible problems if you

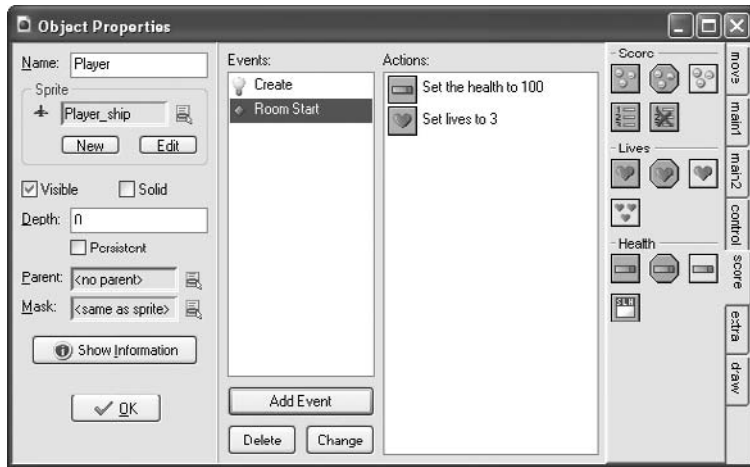


FIGURE 8.31 Setting up the lives and score.

think logically about what you are doing. In the next set of events you will check for the pressing of the left mouse button and then change the left animation. You will also check for the pressing of the right mouse button and play the right animation. That will work fine, but because the program does as it is told, it won't set the animation back to forward pointing once you press the left or right arrow keys. Therefore, when you run the game you will notice that after you have pressed left or right, if you remove your fingers from the keyboard, the ship will be pointing in one of those directions, which will not appear correct. Notice that when you take your fingers off the arrow keys, the ship continues to move in the direction in which you directed it. This means we need to add a third event to take into account when there is no button being pressed, where we will set the animation to forward facing and stop the ship from moving. These kinds of playability issues become much clearer when you begin to test your game and are not a big problem on the whole. You program your basic engine and the movements, test it, and tweak it where necessary.

Left Movement of the Spaceship

Start with moving the space ship to the left.

1. Click on the Add Event button and then choose Key Press | <Left>.
2. The first action is to move the object to the left at a particular speed, so ensure that the Move tab is selected and then drag and drop the Move Fixed object onto the blank action area. The Move Fixed object is the first icon under the Move heading.
3. The Move Fixed dialog box will then appear. Click on the left arrow in the directions section and change the speed to 8. It will now be configured as shown in Figure 8.32.

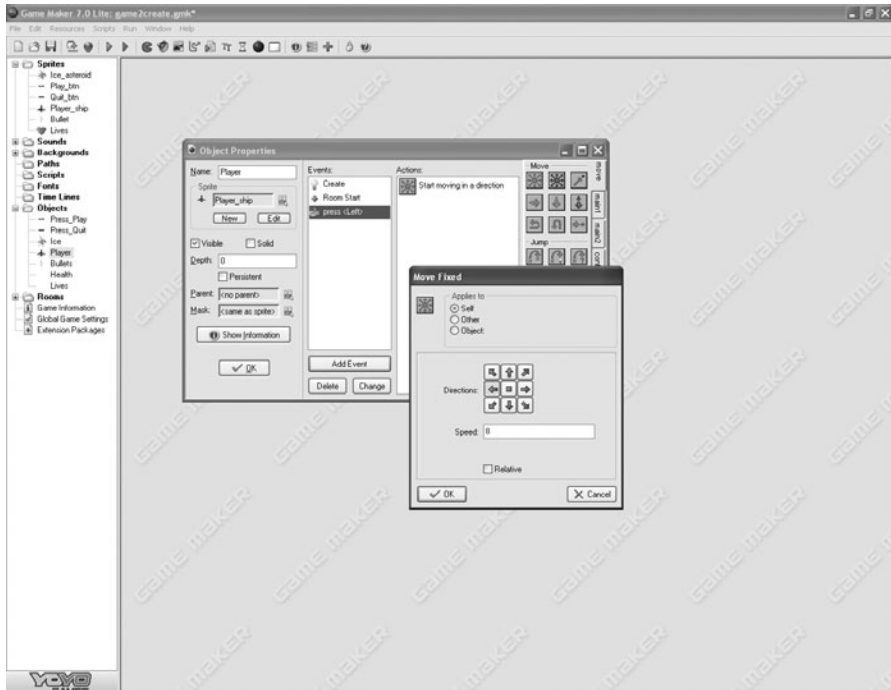


FIGURE 8.32 The configured Move Fixed dialog.



In your own games you may have to enter different speeds to see which number best suits your game. It may involve some trial and error, but it is very easy to test and see if it is appropriate to the type of game you are making.

4. Click on the OK button in the Move Fixed dialog.

You now need to tell the program to play the left animation of the ship so it tilts to the left when the player presses the left arrow. This is very similar to the code that you used when you set the animation to forward when you created the Create event.

1. Click on the Main1 tab and then drag and drop the Change Sprite object (remember, this is the first icon in the sprite section).
2. In the dialog box, click on the drop-down box and pick Player_ship. Then type in the number 1 for the subimage (0 is forward facing, 1 is left, and 2 is right) and replace the speed option with 0. Click OK to save the action. Your event and action will look like Figure 8.33.

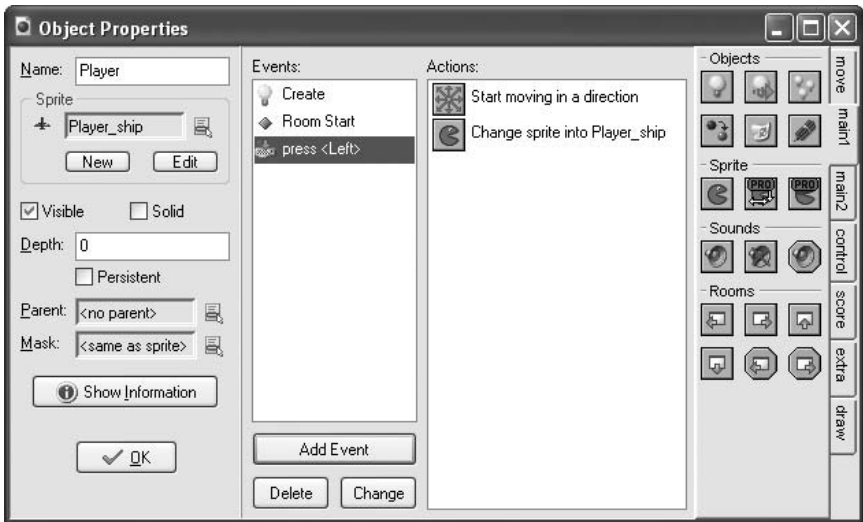


FIGURE 8.33 The left button actions.

If you run the program now and then press the left arrow key, the ship will go flying off to the left and leave the screen. This is working, but obviously you still have work to do to make it a perfect game.

Right Movement of the Spaceship

Now you need to do a very similar process, but for the movement to the right:

1. Click on the Add Event button and then choose Key Press | <Right>.
2. Ensure that the Move tab is selected and drag and drop the Move Fixed object onto the action area (the Move Fixed object is the first icon under the Move heading).
3. The Move Fixed dialog box will appear. Click on the right arrow in the directions section and change the speed to 8. It will now be configured as shown in Figure 8.34.
4. Click on the Main1 tab and then drag and drop the Change Sprite object.
5. In the dialog box, click on the drop-down box and select Player_ship. Then type in the number 2 for the subimage option and then set speed to 0. Click OK to save the action. Your event and action will look like Figure 8.35.

If you run the game now, you can make the ship move to the left and to the right.

No Movement of the Spaceship

To make the game work correctly, you need to stop the movement of the ship and place its animation state into looking forward when the player takes his fingers off the left or right arrow key. For this we can use the No Key event.

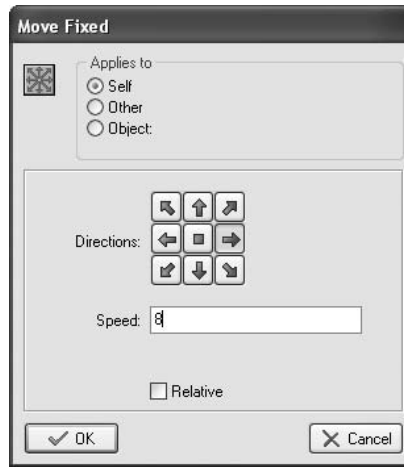


FIGURE 8.34 The movement setting to the right.

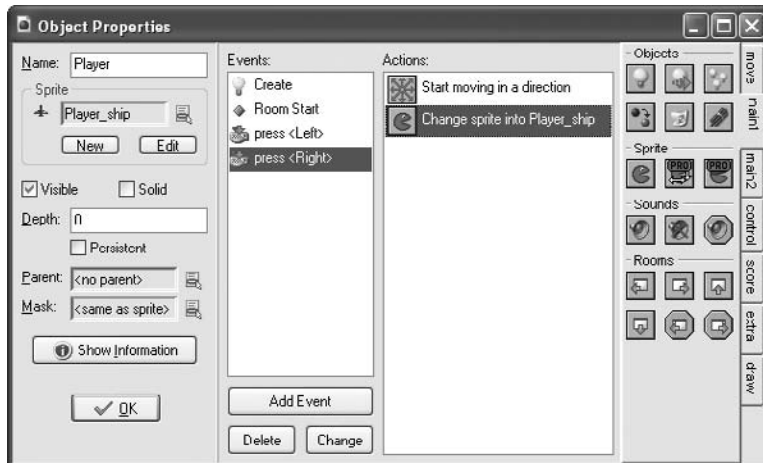


FIGURE 8.35 The events for the right action.

1. Click on the Add Event button, and select Keyboard | <no key>.
2. Select the Move tab and drag the Move Fixed icon to the actions box.
3. Ensure that the square box in the middle of the directions is selected. This represents no movement, and if you don't change this, the ship will continue to move even when no key is being pressed. Leave the speed as 0 and click OK.
4. Now, to change the sprite to forward facing, click on the Main1 tab and drag the Change Sprite option. When the dialog box appears, change the sprite to Player_ship. The subimage should stay at 0, and the speed should be set to 0.

5. Click OK.
6. Run the game now and test that you can move the ship to the left and the right and that when you are not pressing any of the arrow keys, the ship moves back to the forward position and does not move on the screen.

Stopping the Ship from Leaving the Screen

In many games you will want to stop a character or spaceship from leaving the screen. At the moment you can move the spaceship to the left- or right-hand side of the screen and make it disappear out of the window. Though you can still bring it back by using the opposite key, you should, where possible, stop the player from doing things that are not supposed to happen. Even though you could let the player move the ship off the screen, it detracts from the overall polish of the game and is very quick to prevent.

To stop the player's ship from leaving the screen, you can call upon an event that will check if the ship is about to leave the edge of the screen and then apply a stop movement to it.

1. Still within the Player_ship Properties dialog, click Add Event. Then choose Other | Intersect boundary.
2. Drag and drop the Move Fixed object from the Move tab onto the actions box.
3. Click the center square box, leave the speed at 0, and click OK. You can see the event and action in Figure 8.36.

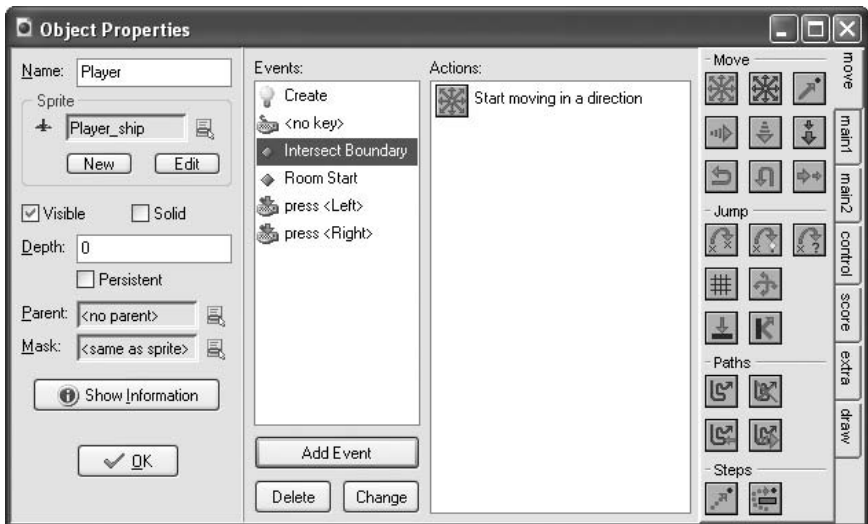


FIGURE 8.36 The intersect boundary and its action.

Ship Collision with Ice Asteroids

You need to check for the spaceship being hit by the asteroids so that you can reduce the amount of health that is left. To do this you can use the collision option. This will check if the object has collided with a specified object, which in this case is the asteroids. When an asteroid hits the ship, subtract 20 points off the health score.

1. Still on the Player_ship Object Properties, click Add Event and then select Collision | Ice so that event will run the action whenever the player's ship hits (collides with) an ice asteroid.
2. Click on the Score tab. Drag and drop the Set health option to the action box. Type "-20" into the box (without the quotation marks) and then click the relative button before clicking OK.



If you do not click the relative checkbox, when the program collides with an asteroid, it will set the health to -20 rather than 20 off the total.

No More Health

You have told the program to reduce the health by 20 every time the ship is hit by an asteroid, and though you haven't programmed the asteroid's movement yet, you can see that you need to create an event and actions to reset the lives and health. When there is no health left, you would need to reduce the lives by 1 and reset the health back to 100.

First, create the event: On the Player_ship Object Properties window click Add Event. Then click Other | no more health.

Removing a Life

The first action to create under this event is to remove a single life. Later on, you will program what happens when there are no lives left, but for now you need to create the conditions that will continue to reduce the lives every time there is no health left.

1. Ensure that the Score tab is selected and then drag and drop the Set lives icon to the action box (first icon under the lives heading).
2. In the dialog box that appears, type in "-1." Very much like the configuration with the setting of the health, you need to click the relative box. Ensure it is selected and then click OK.

Resetting the Health Back to 100

Once you have removed a life, you need to set the health back to 100 so that the process can start all over again. If you do not set it back to 100, then when the ship is hit, nothing else will happen.

1. Ensure that the Score tab is selected and then drag and drop the Set health icon to the action box.
2. In the dialog box that appears, type in “100” and then click OK.

You can see the event and its actions in Figure 8.37.

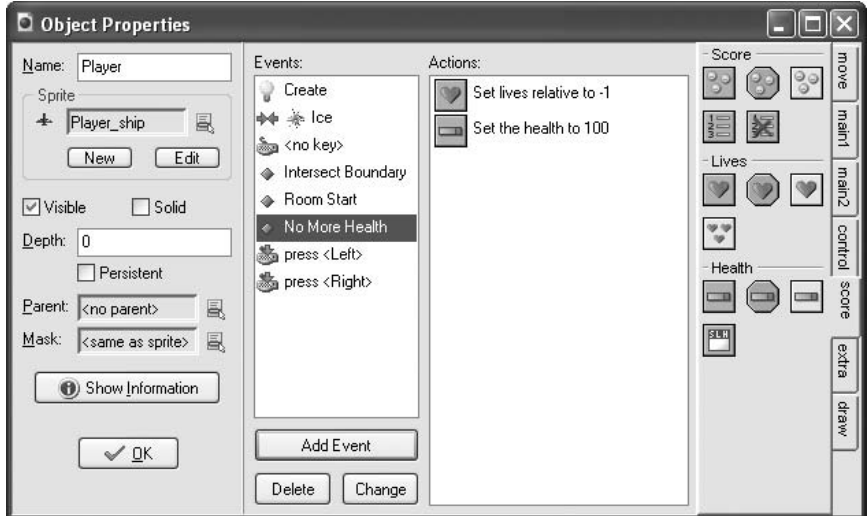


FIGURE 8.37 The lives and health actions for the collision event.

Firing the Gun

You now need to set up the event and actions for firing the gun, which is controlled by pressing the spacebar. You can create a single action event for this process, but the gun will fire many bullets quickly. You want to restrict the firing of the gun to make the game a little harder for the player so that all asteroids are not destroyed before they even get close to the spaceship. You may need to consider fine-tuning your game in these ways so that the player doesn't have an unfair advantage; otherwise, the game might be too easy and the player will get bored of it very quickly.

The actions for this process are slightly more complex than some of the previous code we have completed, as we are going to create a code block. A code block allows you to create an additional check within the actions and only run the code if that action is true. You have already set up a variable called `gunshot`. You may remember creating this variable in the Create event earlier on. Check to see if this equals 1, run the code that fires a bullet, and then create a timer to prevent the gun from firing until the timer has completed.

Start by creating the event:

1. First you need to check for the spacebar being pressed. Click Add Event and select Keyboard | <Space>.

2. Now, create the code that will check to see if the gunshot variable is set to 1. In the Create event at the very start of the game it will be set to 1, which means the player will be able to press the spacebar once and fire the gun. Once the player has fired the gun, the variable is set to 0 until a predetermined time has passed and the counter is reset.
3. The first action to create is to check if the gunshot variable you set up earlier is equal to 1.
4. Click on the Control tab and drag and drop the Test Variable item onto the action area (the middle object under the Variables heading).
5. In the dialog box that appears, type in the variable name as “gunshot” and the value as “1” and then click OK to close the dialog.
6. You now have your test. If it is true, any actions under it will run, but for this to work you need to create a code block that will indent the code.
7. Still on the Control tab, select the Start Block object (the first object under the Other heading; it resembles an up-pointing arrow).
8. Next, you need to place the actions that will fire a bullet. To do this, create another version (copy) of the bullet, called an instance.
9. Click on the Main1 tab and drag the Create instance object onto the action box.
10. A dialog box will appear. In the drop-down box, select the Bullets object. For the X coordinate box type in “76” and for the Y coordinate box type in “16.” Click the Relative box. This will place the bullet at the front of the spaceship. You can see the options for the dialog box in Figure 8.38. Click OK to close the dialog box.



If you do not click the relative box it will place the bullet at X76, Y16, using the top-left corner of the screen as the starting point.

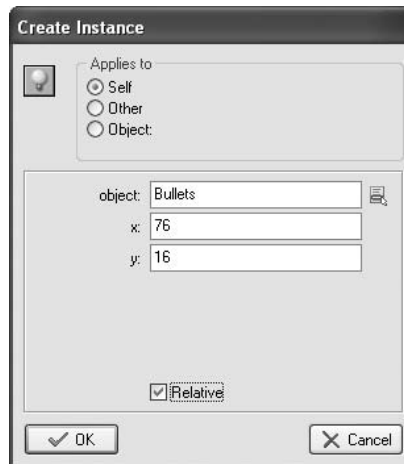


FIGURE 8.38 The Create Instance settings.

11. The next action is to set an alarm, which is effectively a timer. By assigning a timer you can wait for a particular passage of time and then tell the program that the bullet can be fired again. For this event, only set the alarm. In a later event you will create what happens when the time has elapsed.
12. Click on the tab Main2 and select the Set Alarm object (the first object in the Timing group that looks like a clock).
13. A dialog box will appear asking for the number of steps required. 30 steps equals 1 second, so you want to fire slightly less than a second, so type in “25” and leave the alarm number as “Alarm 0.” Click OK to close the dialog box.



The drop-down box in Alarm No allows you to create multiple alarms if you require them.

Now that you have fired the bullet and set a timer, you want to change the variable gunshot to 0 so that this group won’t run again until you change the variable back to 1.

14. Ensure that the Control tab is selected and drag the Set Variable option to the action box.
15. Type in the variable name “gunshot” and set the value to 0. Click OK to close the dialog box.
16. Now that you have finished your code group, you need to close it, so drag the End Block object (the down-pointing arrow) from the Control tab to the action box.

You can see all the actions in the Figure 8.39.

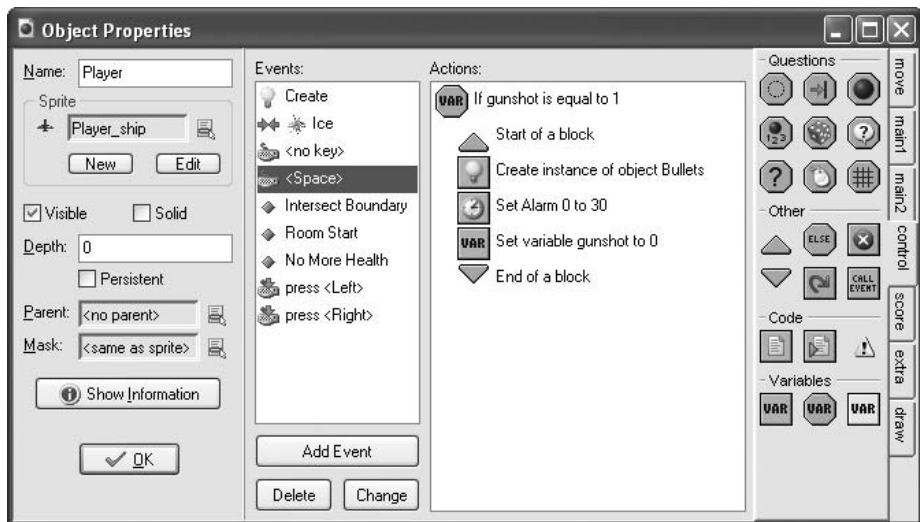


FIGURE 8.39 The actions for pressing the spacebar.

Creating an Alarm

The last event for the `player_ship` is to create a timer (alarm) and tell it to set the variable `gunshot` to 1. The program will automatically handle when this event is run based on the setting of the alarm in the `<space>` event, which is configured to 25 steps.



A good way to get the right timing for your game is to play it and make slight amendments to the property in the relevant dialog. In this example you could try increasing the number of steps and seeing if you prefer that setting. Alternatively, you could reduce it and see how this affects the game play.

1. Click Add Event and then select Alarm and Alarm 0 from the pop-up dialog.
2. Now you need to set the variable `gunshot` to “1” to allow the spaceship to shoot a bullet when the player presses the keyboard. By specifying Alarm 0, you are effectively telling the timer to do something once this alarm has been reached. You have configured the timer to 25 steps, and this event will run when the 25 steps have been reached.
3. Ensure that the Control tab is selected and then drag Set variable to the action box.
4. When the Set Variable box appears, type the name of the variable to be “`gunshot`” and the value to “1.”
5. Click OK. You will now see the relevant event and actions shown in Figure 8.40.

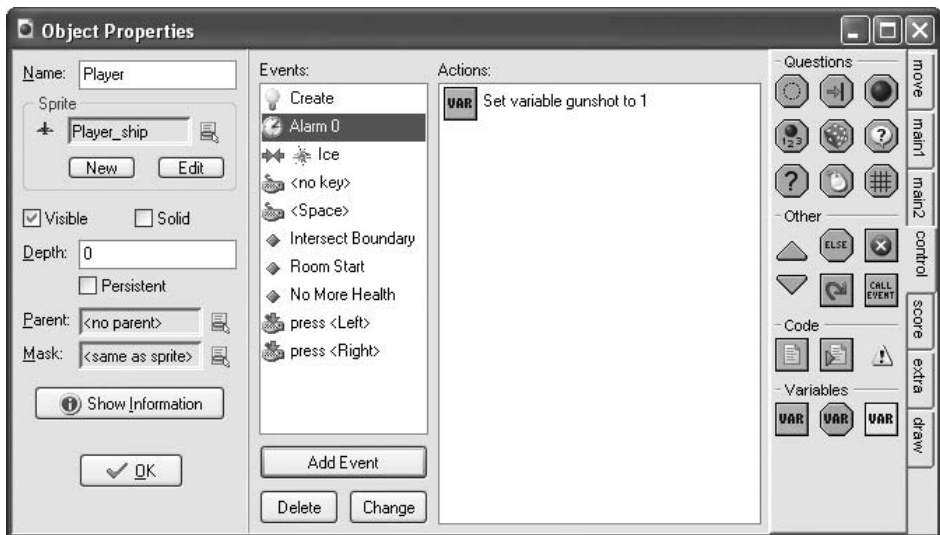


FIGURE 8.40 Actions for the alarm event.

Bullet Events

If you run the game so far, when you press the spacebar it will fire a bullet, but it will paste it directly at the top of the spaceship, and it will not move. You now need to program the bullets so that as soon as they are created they move in an upward direction and when they collide with the ice, they are destroyed.

You need two events to complete the programming under the bullet object.

Creating Bullet Movement

You need to create a Create event that will identify any bullets that have been created, and the action will give the bullets direction and speed.

1. Double left-click on the Bullets object.
2. Click Add Event and select Create.
3. Drag and drop the Move Fixed object from the Move tab onto the action box.
4. When the dialog box appears, click on the up-pointing arrow, type the speed as 8, and then click OK to close the dialog.

Bullet Collision with Ice

Now you need to destroy the bullet once it hits the Ice object. It is very important to destroy items you don't need anymore. If you didn't destroy the bullet, it would continue up the screen and hit other Ice objects, and the Ice would be destroyed. In some games that may be preferable, but it would make this game too easy, and there would be very little challenge for the player.

1. Still on the Bullet Object Properties, click Add Event. Select Collision and then from the drop-down menu click on the Ice object.
2. Click on the Main1 tab and drag and drop the Destroy object onto the action box (the destroy object looks like a recycle bin).
3. Leave the dialog box that appears set to the default of Self and click OK.
4. Click OK to close the Object Properties dialog box.

If you run the game now, you will be able to move the ship and fire the bullets, which will move upward. The bullets will be destroyed when they hit the Ice object, but the Ice object will remain intact, so it's now time to work on this object's events and actions.

Ice Events

The next step is to create the events and actions for the Ice object. The Ice object will be moving on screen and trying to hit our spaceship. It can be destroyed when it hits the spaceship, when it is hit by a bullet, or if it goes too far off screen.

Create Ice

First, you need to place a Create event that will run the actions when the item has been created. Then set a movement and speed for the Ice object to make it fly across the screen.

1. Double left-click on the Ice object in the left-hand window of Game Maker.
2. Click Add Event and select Create.
3. Drag and drop Move Fixed onto the blank action area.
4. When the Move Fixed dialog appears, select the bottom three arrows that are pointing in a downward direction and set the speed to 4.
5. Click OK to place the action.

Collision with Player

You have already programmed what happens to the player's ship when the ice hits it, but now you need to tell Game Maker what happens to the Ice object. In this game you want to destroy the ice, but if you only did that, it wouldn't be long before there was no ice left on the screen, because at the moment you are not creating any more Ice objects. The create instance action is very useful, as you can use it to place a new Ice object on screen when one is destroyed. Therefore, in this game it is a never-ending process, and the ice will never run out.

1. Still on the Ice Object Properties, click Add Event.

Destroy the Ice

2. Select Collision and then Player.
3. Click on the Main1 tab and then drag the Destroy Instance object onto the action box.
4. Do not change the default setting for the Destroy Instance dialog. Click OK.

Create a New Ice Asteroid

Now you need to create a new Ice object, which will be off the top of the game screen so it won't just appear in the middle of the screen. Once it is created above the screen, the movement actions in the Create event you created will handle the movement of the object.

1. Still on the Player event, ensure that the Main1 tab is selected.
2. Drag and drop the Create Instance object onto the action box.
3. In the dialog box that appears select the Ice object and set its X coordinate to 400 and its Y coordinate to -20.
4. Click OK to close the dialog box.

Collision with Bullets

If the player hits the ice with a bullet, you need to do three corresponding actions. First destroy the ice. You don't need to worry about destroying the bullet, as you did that in the Bullet Object Properties. You need to add 20 to the score and, finally, create a new Ice object to replace the one you destroyed.

First, create the Bullet Collision event.

1. Still in the Ice Object Properties, click Add Event and select Collision. From the pop-up menu choose Bullets.

Destroy the Ice

Now you need to destroy the Ice object.

2. Click on the Main1 tab, select the Destroy Instance object, and place it on the actions box.
3. Leave at "Applies to Self and click OK.

Set the Score

Now it's time to add 20 to the score, and you will need to select the relative box to ensure that it adds 20 rather than sets it to 20.

1. Make sure the Bullets event is still selected.
2. Click on the Score tab.
3. Drag Set Score onto the action box.
4. In the Set Score dialog box, type in "20," click the Relative box, and then click OK.

Create Another Ice Object

Now that we have destroyed the Ice object and added to the score, we should create the new instance ready to fly toward the player's ship.

1. Ensure that the Bullets event is selected and then click on the Main1 tab.
2. Select Create Instance and drop it onto the actions box.
3. In the dialog box, click on the object drop-down box and choose Ice. Then type "400" for the X coordinate and "-20" for the Y coordinate.
4. Click OK to close the Create Instance dialog box.

You can see the three actions for the Bullet event in Figure 8.41.

Moving Outside the Screen

When the ice asteroid does not get hit by a bullet or hit by the spaceship, it will fly off the bottom of the screen. If this was to continue, all the objects that are off the

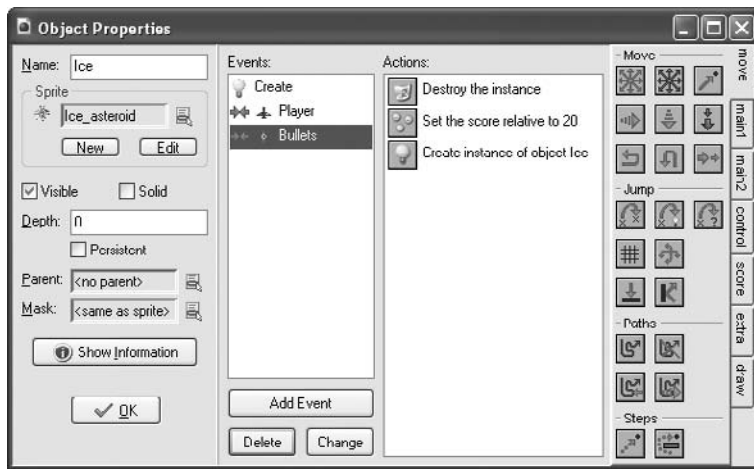


FIGURE 8.41 The three actions for the bullet event.

screen would still exist in the game, and the game would take up more and more memory as it continued. You need to destroy the ice asteroids that are no longer on the screen. First, you need to create an event that checks to see if the items are outside the room.

1. Ensure that the Ice Object Properties are displayed and then click Add Event. Select Other | Outside room.

Destroy the Ice

We need to destroy the Ice object.

2. Click on the Main1 tab and select the Destroy Instance object and place it on the actions box.
3. Leave at Applies to Self and click OK.

Create Another Ice object

Now that we have destroyed the Ice object, we should create the new instance ready for it to fly toward the player's ship.

1. Ensure that the Outside Room event is selected and click on the Main1 tab.
2. Select Create Instance and drop it onto the actions box.
3. In the dialog box, click on the object drop-down box and choose Ice. Then type "400" for the X coordinate and "-20" for the Y coordinate.
4. Click OK to close the Create Instance dialog box.
5. Click OK on the Ice Object Properties sheet, as you have completed all of the events for this object.

Health Events

There is only one health event and action, and this is to draw the health bar on screen. Once it is drawn, reducing the player's health will automatically update the health bar.

1. Double left-click on the Health object in the object's folder to display its Property dialog box.
2. Click Add Event and select Draw.
3. Click the Score tab and drag and drop the Draw Health bar.
4. A dialog box will appear that at first sight may seem a little complicated. Type in the settings in Table 8.3.

Table 8.3 The Draw Health Settings

TYPE	AMOUNT
x1	160
y1	576
x2	260
y2	586
Back color	Black
Bar color	Green to red

X1 and Y1 represent the starting coordinates of the bar's location, while the X2 and Y2 coordinates represent the bottom-right position of the bar.

5. Click OK to close the dialog box.

Life Events

The final set of events to create are for the Lives object. First you need to create an event that will keep track of how many lives the player has, and when it reaches zero it will restart the game. Then you need to draw the Lives object. This will take the Lives sprite and place it in the location we specified in the room.

1. Double left-click on the Lives object in the objects folder to open up its properties.

No More Lives

2. Click Add Event and select Other | No more lives.
3. Select the Main2 tab and drag and drop Restart Game to the action box.

Draw

1. Click Add Event and select Draw.
2. Select the Score tab and then drag and drop Draw Life Images onto the action box.
3. Leave the X and Y coordinates at 0 and then from the Image drop-down box select Lives.
4. Click the Relative box and then click OK button to close the Properties dialog.

You have now completed all of the events and actions for the game. If you run the game, you will notice that you nearly have a fully working game, but you need to add some music to make the game stand out. This is done via a small bit of code, which is discussed next.

ADDING SOUND USING A SCRIPT

You can use more traditional programming with Game Maker, and though it's not the scope of this book to go into this, we will do it for playing music, as it will make it very quick and straightforward. Learning the coding aspects takes longer than using the event system, but because some aspects of the event system are not as powerful as scripting, you will probably find yourself trying it once you have become more proficient at the events. It is more complex but it adds a new level of power to your game.



The scripting language is called GML, which stands for Game Maker Language.

There are a number of different places you can create a script, for example, in the Scripts folder, and then you can call it by an event. We are going to add a script to the Main room, so that as soon as the game starts, it will play our music.

1. Expand the Rooms folder and then double left-click on Main to open its Properties box.
2. Click on the Settings tab in the Main Room Properties window.
3. Click the Creation Code button.
4. A new window will open called Room Creation Code. This is where you can type any code that you want to execute in this room.
5. Type in "sound_loop(Music)." This will play the sound file Music that we added at the start and then loop it.



There is a particular command to play sounds using scripts within Game Maker, and you can apply a number of settings to it. If you are interested, you can find more information about GML in the product's help files.

6. Click on the green arrow in the Room creation code to close the dialog and save the script.
7. Click on the green arrow in the Room Properties dialog box to close it.

If you now run your game, you will have sound that starts in the first screen and continues into the game level. It will loop continuously.

ADDING A HELP FILE

When you create a game, or any computer program, you should always provide a help file. This file can be part of the game or a separate text file, Adobe Acrobat file, or Windows Help file. Game Maker provides a built-in help system the player can access at any time by pressing the F1 key.

You can access the blank help file by double left-clicking the Game Information option in the left-hand window of the Game Maker application. You can see the blank help window in Figure 8.42.

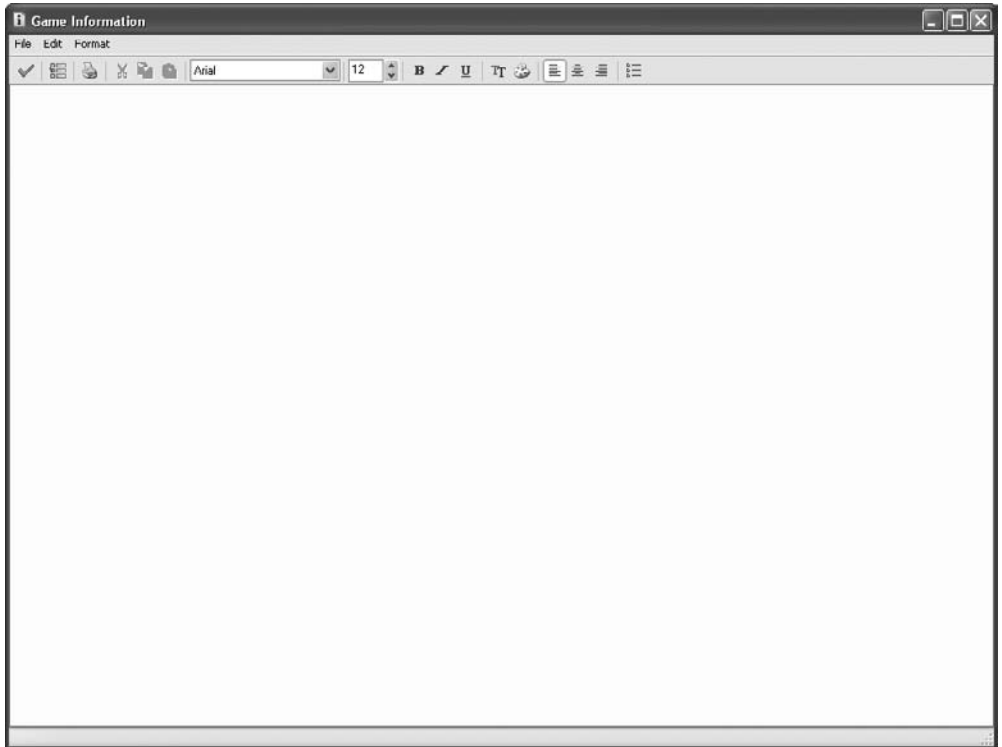


FIGURE 8.42 The Game Maker help system.

Type in the information about your game. This should include:

- The basic premise of the game
- The game controls
- Any copyright information
- The location of any additional information about the product
- Support and patch information, including Web sites

Type in your game information and then click on the green arrow to save it. You can see an example of a completed help file in Figure 8.43.

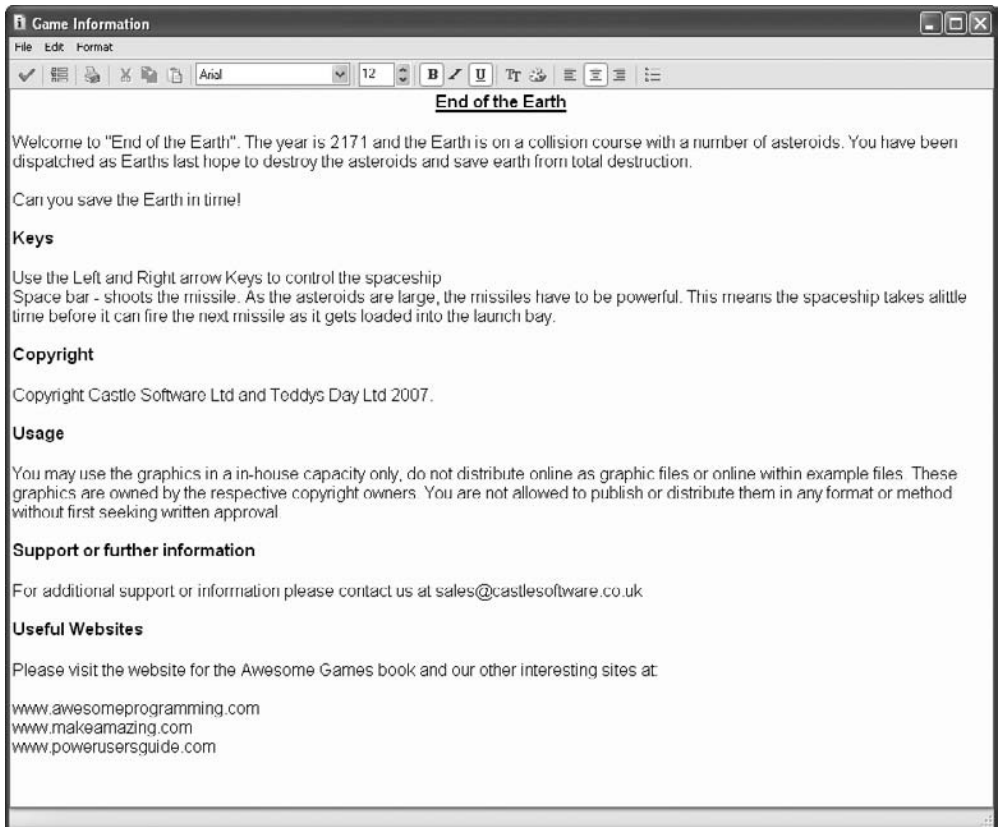


FIGURE 8.43 An example of a completed help file.

CREATING AN EXECUTABLE FILE

When you have completed your game, the next step is to create a Windows executable file. This step allows you to distribute your game so that anyone can play it without the need for the Game Maker software.

1. Click File | Create Executable.
2. It will ask for the name of the executable and where you want to save it. Select a location and type in a filename. Then click Save.

As you are using the lite version, when the executable is launched, a short banner will be displayed that advertises the Game Maker software. To remove that from your executables, you will need to upgrade to the Pro version.

CHAPTER SUMMARY

In this chapter you made an exciting space shoot 'em-up game called End of the Earth, and you learned a lot about how to use the Game Maker software. You learned how to add different types of events and actions to your game, and how to add sound using a simple script. You should now be confident enough to begin making your own games using the software. You have learned as much about the product as possible without going into its scripting capabilities. We recommend that you continue to make small changes to this game and add a few more features. The next chapter will look at another event-based system called The Games Factory 2, which takes the eventing to a very high level, allowing you to make exciting and powerful 2D games.

This page intentionally left blank

INTRODUCTION TO THE GAMES FACTORY 2

In This Chapter

- About TGF2
- TGF2 Requirements
- Installation of TGF2
- Starting TGF2 for the First Time
- A Quick Introduction to TGF2



In the following chapters you will be making a number of games step-by-step using one of the most powerful and simple to use 2D game-making programs, called The Games Factory 2.



The Games Factory 2 software is also called TGF2 for short and is called this throughout this book.

Many of the ideas and techniques you will learn in the games you create for TGF2 in this book will help you make more complex programs using the same process. Once you are comfortable with TGF2, you will be able to use it to produce games and interactive applications with ease. TGF2 also contains state-of-the-art animation tools, movement functions, and game bases routines that make it easy to produce your own games with no programming.



You can also make slide shows, interactive tests, presentations, and screensavers with TGF2.



ON THE DVD

Start by installing TGF2 and getting familiar with its major functions. The trial version of the product (TGF2Demo.exe) is available on the DVD provided with this book. You will find it in the folder called Demos.

ABOUT TGF2

The introduction to this chapter stated that TGF2 is a tool used to create games without the need for programming. It achieves this by using an event-based system whereby the “programmer” uses the mouse to select a number of conditions and actions by using the mouse. This is all done using a graphical interface and does not require the need for the programmer to learn key words or programming terms. This means the basic concept of making programs in TGF2 is the same regardless of what you are trying to make. This allows you to stop programming in it for a few months and not have any problems picking it back up again and starting again where you left off. The reason for this is that the program is based on the concept of editors, and once you understand how to use them, it’s very easy to remember how to begin putting your game together. This is unique in the game and programming world, as most programs require either the user to type in text or a combination of text and event-based programming. This can cause the programmer headaches if he doesn’t have a great memory for remembering the text that has to be typed in to get something to work.

You may be thinking that the exclusion of typing in lots of text (traditional programming) would mean TGF2 is not very powerful. TGF2 is a program with a long heritage, and previous versions of it (under other names) have been in existence for over a decade. This means the program has become very powerful and very logical to use, as over the years it has been refined and developed. This makes the development of many 2D programs easy without any programming knowledge.

You can find more information, downloads, and tutorials for TGF2 at the maker's Web site: www.clickteam.com.

TGF2 REQUIREMENTS

Tables 9.1 and 9.2 list the basic minimum requirements for installing and running TGF2 as well as the recommended requirements. Where possible you should ensure you meet or exceed the recommended requirements, as this will lead to a better development experience when working on more complex and resource-hungry games. TGF2 runs on most PC-based configurations and even works on older operating systems as well as the latest from Microsoft, including Windows Vista™.

Table 9.1 Minimum System Requirements for Installing and Running TGF2

MINIMUM REQUIREMENTS

Operating system: Windows 95 with IE 4.0, Windows 98, Windows NT 4.0 with Service pack 3 or above, Windows 2000, Windows XP, Windows Vista

Pentium Processor

32 MB RAM with Windows 9x, 64 MB with Windows NT, 128 MB with 2000 and Windows XP, 512 MB with Vista

CD-ROM drive

Graphics card with 8 MB or more (or minimum OS requirements)

Sound card (optional but recommended)

50-100 MB free hard disk space

Table 9.2 Recommended System Requirements for Installing and Running TGF2

RECOMMENDED REQUIREMENTS

Operating system: Windows 98, Windows 2000, Windows XP, Windows Vista

Pentium 4 Processor

64 MB RAM with Windows 98, 256 MB RAM with Windows 2000 or XP, and 1 GB RAM with Windows Vista

CD-ROM Drive

Graphics card with 32 MB RAM

Sound card

200-500 MB free hard disk space

INSTALLATION OF TGF2



This section will guide you through the installation of the trial version of TGF2 that is provided on the DVD with this book. The TGF2 software is located on the DVD in the folder called Demos. Find this folder and then double left-click on the file TGF2Demo.exe.

1. The first screen you will see is the “Welcome” dialog shown in Figure 9.1. This dialog box gives some details about TGF2 and asks you to ensure that you are not running any other Windows programs before proceeding with the installation.



FIGURE 9.1 TGF2 Welcome dialog box.

2. Click Next button to continue with the installation.
3. The next dialog box, shown in Figure 9.2, provides detailed information about the demo version of the software and what’s possible in this version.
4. Read through the information and then click Next.
5. You will now see the License dialog box shown in Figure 9.3. This provides details on what restrictions are placed on using the software and other legal details. To continue, you need to select the I agree with the above terms and conditions radio button, so select that radio button and click Next.
6. Now you will be asked where you want to install the TGF2 files, as shown in Figure 9.4. The default location is C:\Program Files\The Games Factory 2. You can either use this path or change it by clicking on the button with the ellipsis within it. Once you have a location you are happy with, click the Next button.

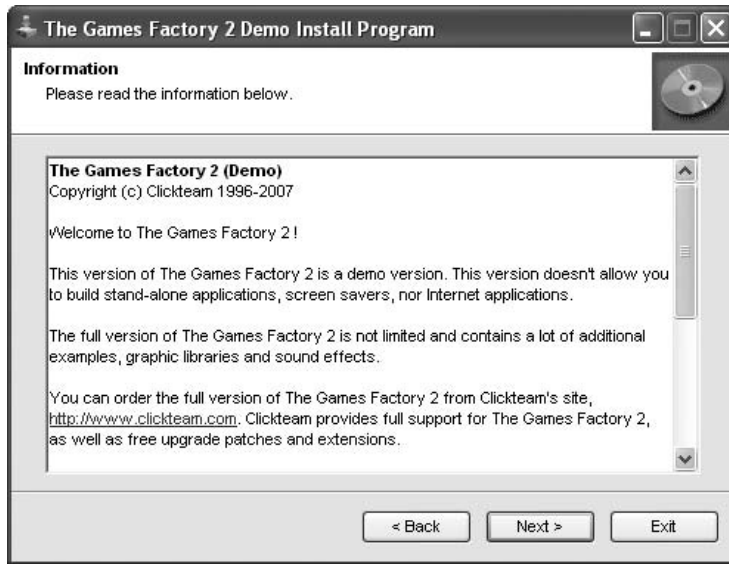


FIGURE 9.2 The Information dialog box.

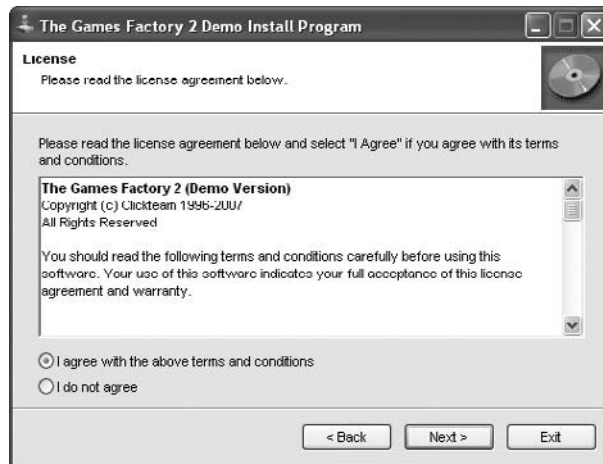


FIGURE 9.3 The license dialog box.

7. You may be advised that the destination folder does not exist and asked if you want to create it, as shown in Figure 9.5. Click Yes to continue.
8. You will now receive a final confirmation message, as shown in Figure 9.6, advising you that the program is ready to copy files to your machine. Click on Start to begin the installation of TGF2.

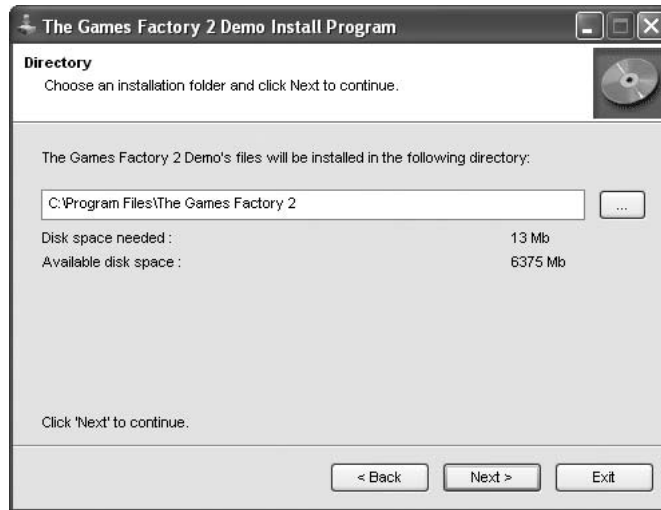


FIGURE 9.4 The default installation path.

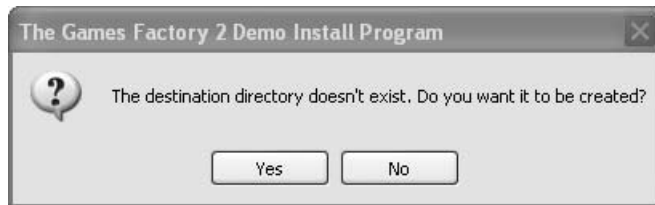


FIGURE 9.5 Destination folder does not exist message.

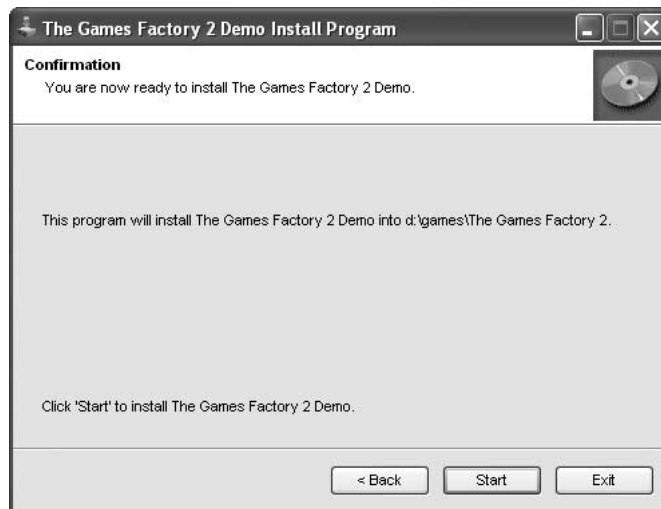


FIGURE 9.6 Confirmation dialog.

9. Once the files have been installed, the final installation dialog box will appear, advising you of the installation success, as shown in Figure 9.7.

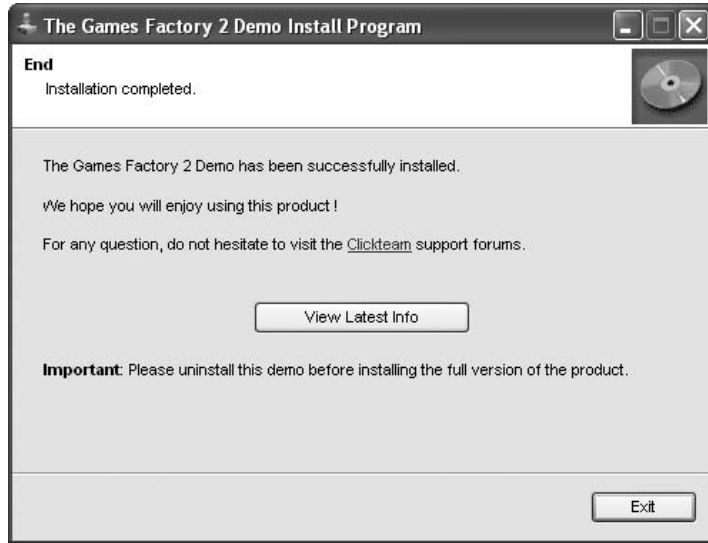


FIGURE 9.7 The final installation dialog box.

10. From this dialog box you can view the latest product information and visit the support forums. Click on the links to access the relevant Web pages.

STARTING TGF2 FOR THE FIRST TIME

When you double left-click on the TGF2 icon on the desktop or access it through the Start button, you will be presented with the Demo Version dialog box shown in Figure 9.8. This details what options are missing from the trial version.

The demo version allows you to create games in the TGF2 native format, which can be opened in the full version if you decide to purchase it. You can click on the link on the bottom left to visit the Clickteam Web site at www.clickteam.com or click Continue to load the program.

Once you have clicked Continue, the TGF2 window appears with a tutorial help file, as shown in Figure 9.9. This tutorial provides an excellent introduction to the product, and you should consider looking at it after you have read through this chapter. You can close it by clicking on the red cross in the right-hand corner of the window. If you need to open it again later, select Help | Tutorial from the text menu.

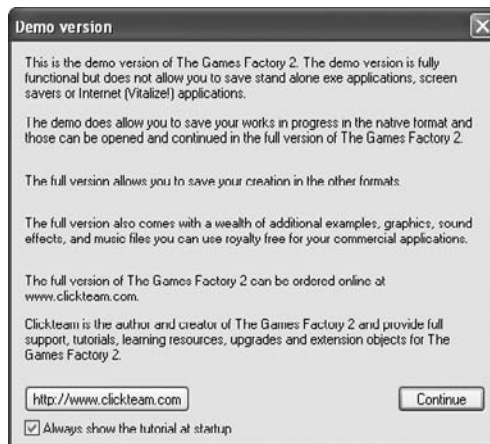


FIGURE 9.8 The Demo Version information box.

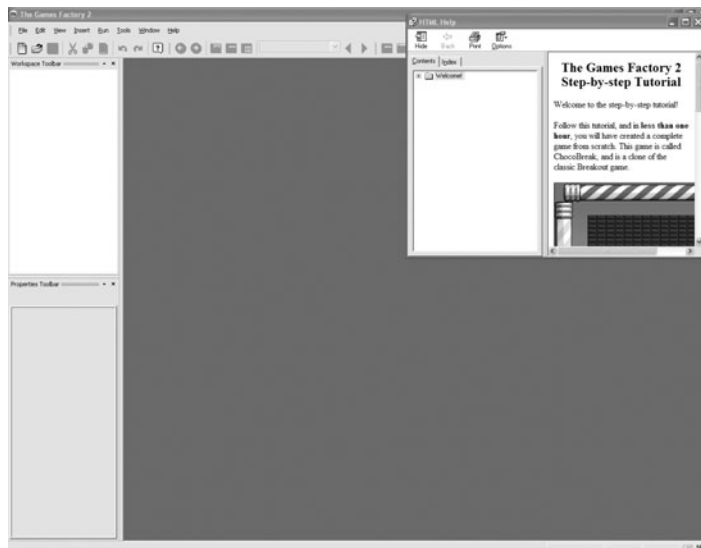


FIGURE 9.9 The TGF2 program with ChocoBreak tutorial.

A QUICK INTRODUCTION TO TGF2

TGF2 centers around three editing screens that allow you to control the main parts of your game:

- The *Storyboard Editor* lets you specify the order of the levels in the game.
- The *Frame Editor* lets you specify which characters, backgrounds, and objects to put in your level.
- The *Event Editor* lets you assign the actions and responses that will make your game come alive.

You can easily move from one editor screen to the next by clicking the editor icons from the toolbar at the top of the screen. If you are unsure which icon allows you to navigate to which editor, leave your mouse over the icon, and a handy tip message will appear. You can see the icons that allow you to move to the main editors in Figure 9.10.



FIGURE 9.10 Storyboard Editor, Frame Editor, and Event Editor.

Storyboard Editor

Most games are composed of several levels. This screen lets you add levels to your game, copy levels, and change the order of the levels. This is also where you decide on the size of your playing area, add and edit professional-looking fades to each level, and assign a password to enter each level. You can see a single-level frame shown in the Storyboard Editor in Figure 9.11.



In TGF2 each separate level or screen is called a frame.

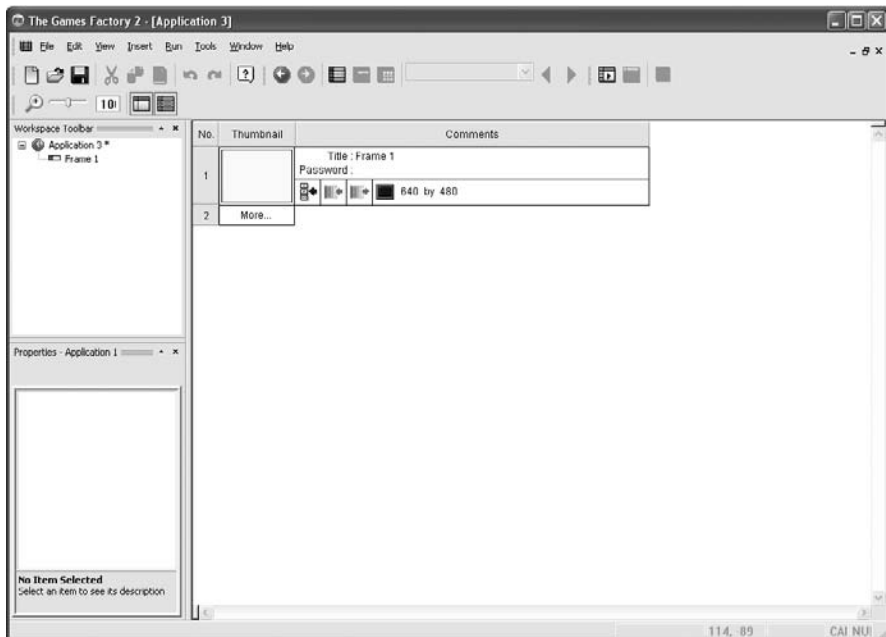


FIGURE 9.11 Storyboard Editor.



When you create a new application in TGF2, it will create a single frame (level) automatically.

The Frame Editor

The Frame Editor shown in Figure 9.12 is the initial “blank page” for each of your levels. The Frame Editor is where you enter the backdrop objects and the main characters of your game. The white area is where any items are automatically displayed within your games window, and the gray area is out of frame, which allows you to position items that can come into play at a particular moment.

This screen also allows you to access the various libraries that come bundled with the trial and the full version and drop them onto your game. It lets you create your own animated objects, text, and other object types. You could consider the frame editor to be the place you set your scene for your game and prepare and configure any items you have added ready for programming in the Event Editor.

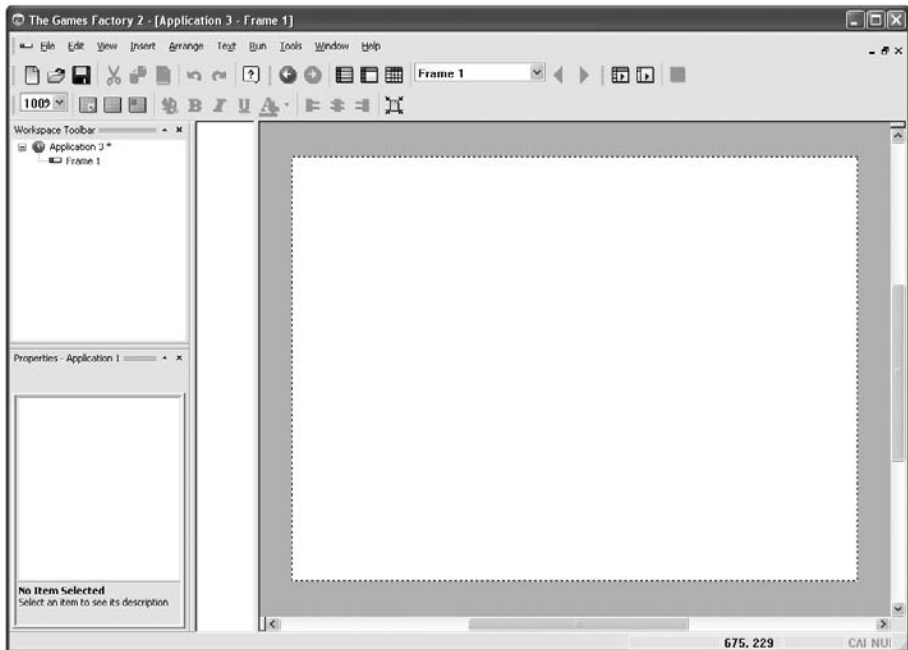


FIGURE 9.12 The Frame Editor, where you place all your items.

Event Editor

This is where you begin to build the logic of your game and make it come to life. You create the interactivity here by assigning conditions and actions. When you are experienced with TGF2, you’ll spend a lot of your time here. This is the editor where you will program your game.

As shown in Figure 9.13, the Event Editor is set up like a spreadsheet (you can only see the top “spreadsheet row” in the figure example). By filling in the rows and columns, you can assign relationships to each object in your game. This setup makes game building easy, since you can see what happens in your game. Examples of the game play elements you can build here include aliens colliding with a spaceship, the main character collecting a power-up or getting hit by a missile, setting a time limit, and assigning a sound event. You can create an explosion, destroy an object, add to the score, subtract a life, or specify complicated events such as changing the direction of a character or a randomly moving object.

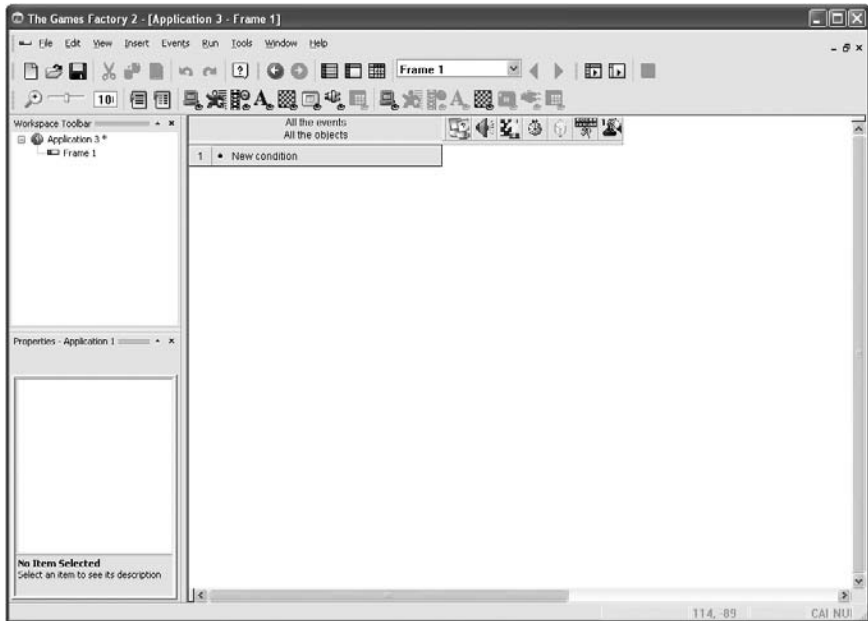


FIGURE 9.13 The blank Event Editor.

That was a quick tour of TGF2. You saw that a game is built in TGF2 in three stages. First you lay out the flow of your game in the Storyboard Editor. Then you lay out each level and their objects in the Frame Editor. Finally, you use the Event Editor to assign relationships and behaviors to your objects.

For the next chapter, you will need to have TGF2 installed and running, if possible, as you will be digging deeper into it.

CHAPTER SUMMARY

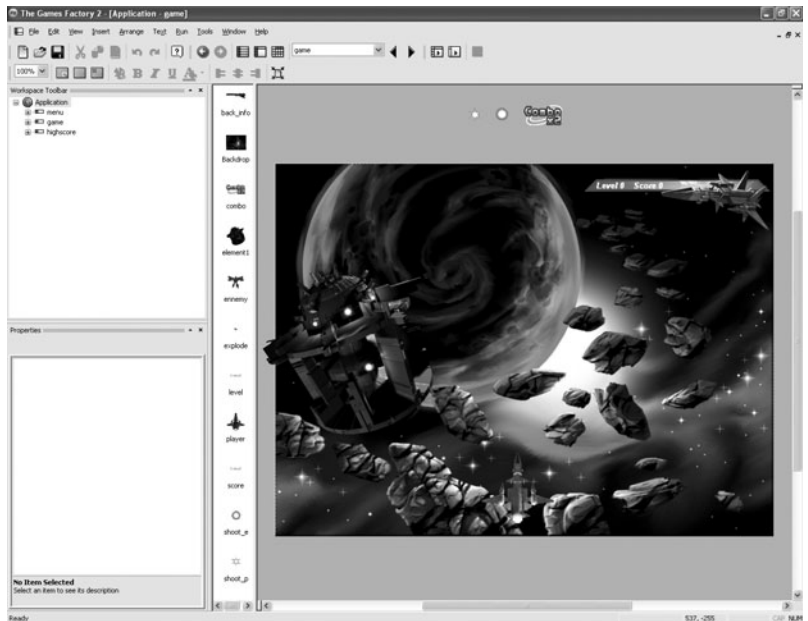
This chapter covered how to install TGF2 and introduced the three main editing screens in which you'll spend most of your time. In the next chapter, you will get to look at a game within the different editors to see how it is made.

This page intentionally left blank

BEHIND THE SCENES OF THE GAMES FACTORY 2

In This Chapter

- About Alien Wars
- Loading Alien Wars
- Alien Wars: The Storyboard Editor
- Alien Wars: The Frame Editor
- Alien Wars: The Event Editor



The next two chapters will take you through the step-by-step process used to construct a very basic shoot-'em-up game with TGF2. This retro-creation called Alien Wars, shown in Figure 10.1, is our own version of *Space Invaders*[™]. You will see that with TGF2, you can create a game that does a lot more than what the original *Space Invaders* could, and it's very easy to create.



FIGURE 10.1 The Alien Wars game.



Retro gaming is very popular. People still love to play Pac Man, Asteroids, Space Invaders, and other older games. Many can be found online in the form of Java applets or Flash that can be played in Web browsers.

ABOUT ALIEN WARS

Before going through the various editors and options available to your game in TGF2, you need to give a little background. The story behind the game goes as follows:

The Earth forces have been fighting the robot invaders for over 10 years. No side is winning the war, but losses are high on both sides. The Earth commanders have given you their latest spaceship hardware, hoping to sway the battle and turn the tide against the invading robots.

Can you handle the new Falcon 29 spaceship? Will you survive the robot attackers? It's time to find out.

Alien Wars will show you the basics of creating games with TGF2. It will show you many of the features and procedures needed to make any game with TGF2. There are a small number of screens and editors you can work in, these will be the same for any game you make with the program.

Even though Alien Wars is a simple game, it will introduce you to many interesting and useful techniques, including:

- Moving between screens
- Creating levels
- Keeping the score
- Creating fade-in effects
- Playing animations
- Creating and using a high score table

The game is split into three frames: the games menu loader, the game itself where the player will fight the enemy spaceships, and the frame that displays any high scores that the player obtains.

LOADING ALIEN WARS



The DVD for this book includes the finished Alien Wars game. The game is quite large, and it is recommended that you copy any files from the DVD onto your PC if you want to open up the code to take a look at it. You need to copy the files to a local hard disk if you want to make any changes, because the DVD is a read-only format, and you cannot save to it. This chapter only reviews the code, but it will still be faster loading the file if it is read from your hard disk, so copy the file `alienwars.mfa` from the `\TGFFILES\Alien Wars` folder on the DVD to a location on your PC.

1. Start TGF2 and click File | Open from the menu.
2. Navigate to the location where you have placed the `alienwars.mfa` file and single left-click on it.
3. In the right-hand corner is a small picture. This is the first screen that is displayed in the game file, as shown in Figure 10.2.
4. Click Open.



FIGURE 10.2 Opening the `alienwars.mfa` file. Notice the thumbnail of this game in the lower right-hand corner.

ALIEN WARS: THE STORYBOARD EDITOR

Now that Alien Wars has loaded, click on the Storyboard icon in the toolbar and you will see the Storyboard Editor screen, as shown in Figure 10.3.

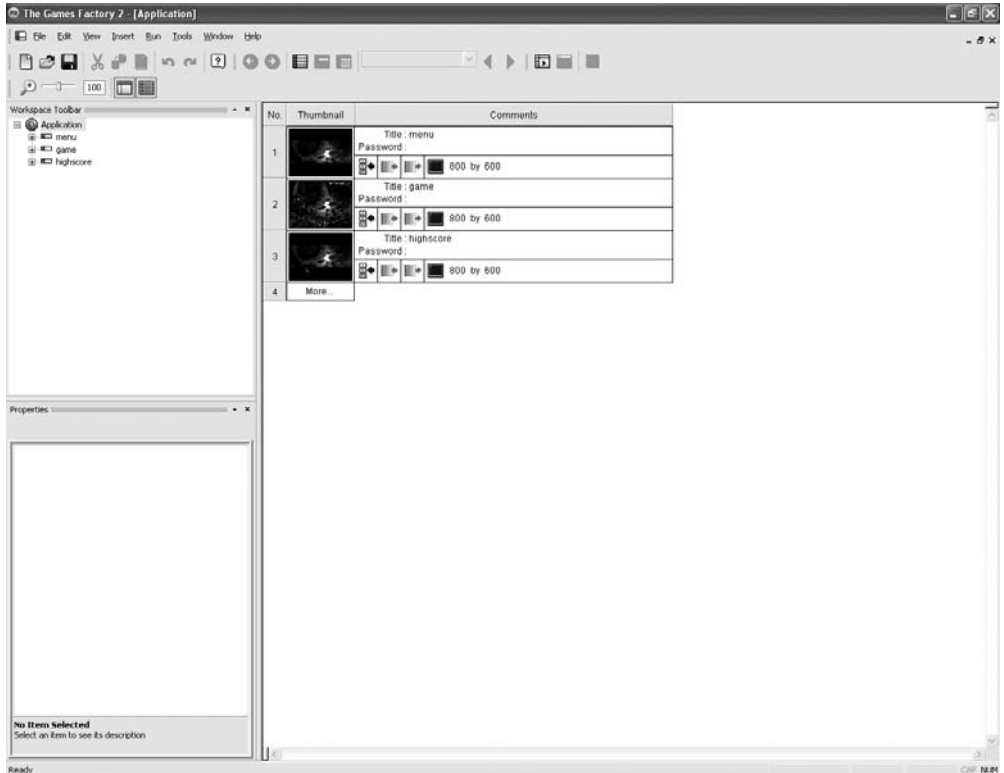


FIGURE 10.3 The Alien Wars game as shown in the Storyboard Editor.

Starting from the top of the Editor screen we will now look at the features of the Storyboard Editor, shown in Figure 10.4. First, look at the number column, which in this example has three boxes with information contained on that row. These are frame numbers, and frames are the levels or separate screens in our game. Clicking on the number will take you directly to that frame and display it in the Frame Editor.

Next to each of these numbers is a small thumbnail picture of the frame of the game. In a large game this can be useful to help you remember which screen does what, so you don't need to click on different frames to find the right one.

Next to the thumbnail images are the comments for that frame, the title of the frame, and the password. To change these, simply left-click the text you want to change. You can edit or add a title or password.







No.	Thumbnail	Comments
1		Title : menu Password :  800 by 600
2		Title : game Password :  800 by 600
3		Title : highscore Password :  800 by 600
4	More...	

FIGURE 10.4 Close-up of the Storyboard Editor window.

Underneath the comments are several buttons shown in Figures 10.5–10.7. These buttons denote a multimedia frame, which is what all your frames are by default. You can ignore these buttons, as they have no current function in TGF2.

The next button allows you to add a fade-in transition to your frame by using the icon in Figure 10.5. You can add a fade-out transition to your level using the icon in Figure 10.6. If you create a fade-in or fade-out transition, it will appear between the number rows and is then selectable and can be changed or removed if needed.







No.	Thumbnail	Comments
1		Title : menu Password :  800 by 600
2		Title : game Password :  800 by 600
3		Title : highscore Password :  800 by 600
4	More...	

FIGURE 10.5 The fade-in transition button.

The play area can be much larger than the screen size, allowing you to create large scrolling games. You can click on the monitor to access a drop-down box or click on the screen size numbers and type in an exact size. The drop-down screen sizes are shown in Figure 10.7.




No.	Thumbnail	Comments	
1		Title : menu Password :	800 by 600
2		Title : game Password :	800 by 600
3		Title : highscore Password :	800 by 600
4	More...		

FIGURE 10.6 The fade-out transition button.




No.	Thumbnail	Comments	
1		Title : menu Password :	800 by 600
2		Title : game Password :	800 by 600
3		Title : highscore Password :	800 by 600
4	More...		

FIGURE 10.7 The screen size options.

ALIEN WARS: THE FRAME EDITOR

Most of the work needed is on the second frame, so click on the number 2 in the Storyboard Editor or double left-click the text “game” in the Workspace toolbar in the left-hand window pane to view the Frame Editor for frame 2.

When you load frame 2 of the game you will see various items displayed on the screen as shown in Figure 10.8.

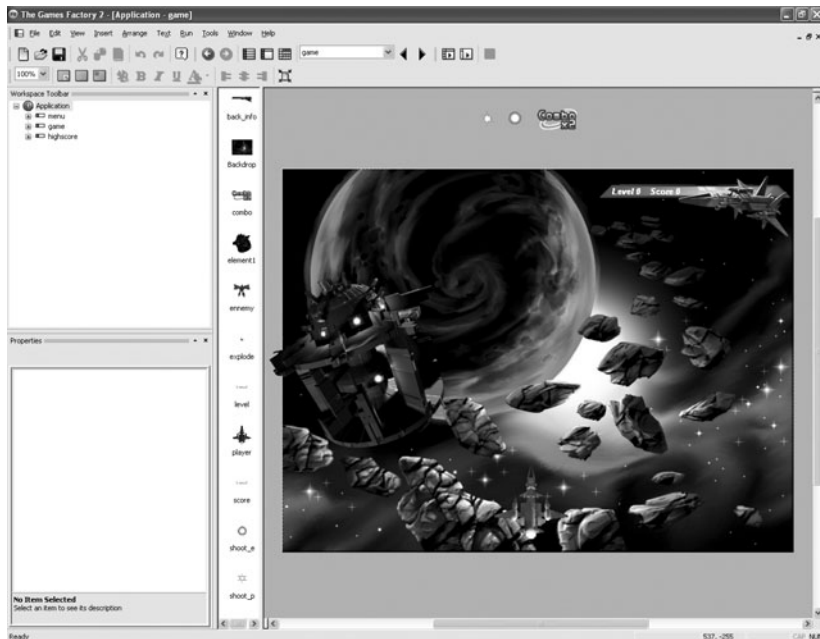


FIGURE 10.8 The second frame for the Alien Wars game.



If you were creating a new frame with no content, it would be displayed as a blank white box surrounded by a gray background.

On the left-hand side of the Frame Editor is a group of items displayed top-down. These are the game items that are used within this frame. The next chapter will show you how these items appear in the game. You can drag items from this toolbar onto the screen, but you should do this with care, as it will create an exact duplicate of any items that may be on the frame already. If you programmed for one type of item to move in a certain direction or act in a certain way, all other objects of the same type will mirror these behaviors.

In the middle of the frame is a space scene. This is what will be displayed when the user plays the game. The gray around it is outside the play area and is used to place items that won't initially appear in the game or that you will make move into the play area while the game is running.

When there are many objects in a level, not all will fit inside the window, so there is a scroll bar on the bottom-right corner to allow you to move around the Frame Editor area.

When you move your mouse over any of the objects in the frame, a handy hint appears telling you the name of the item. This is very useful for identifying the name of the object. You may need to use this function when switching between the Frame Editor and the Event Editor.

Try dragging a few objects from the level panel and placing them on the screen. As you do so, notice that the properties window on the left-hand side is then filled with information. You would use the Object properties window to configure certain aspects of your objects, for example, their movement, visibility, size, and location on the screen. Make sure not to save the program, as you are just getting used to the various options. If you left-click on an object on the frame, you will be able to move it pixel by pixel for perfect placement using the arrow keys. You can also right-click on the object and access a pop-up menu that provides additional features and properties. You can see this pop-up menu in Figure 10.9.

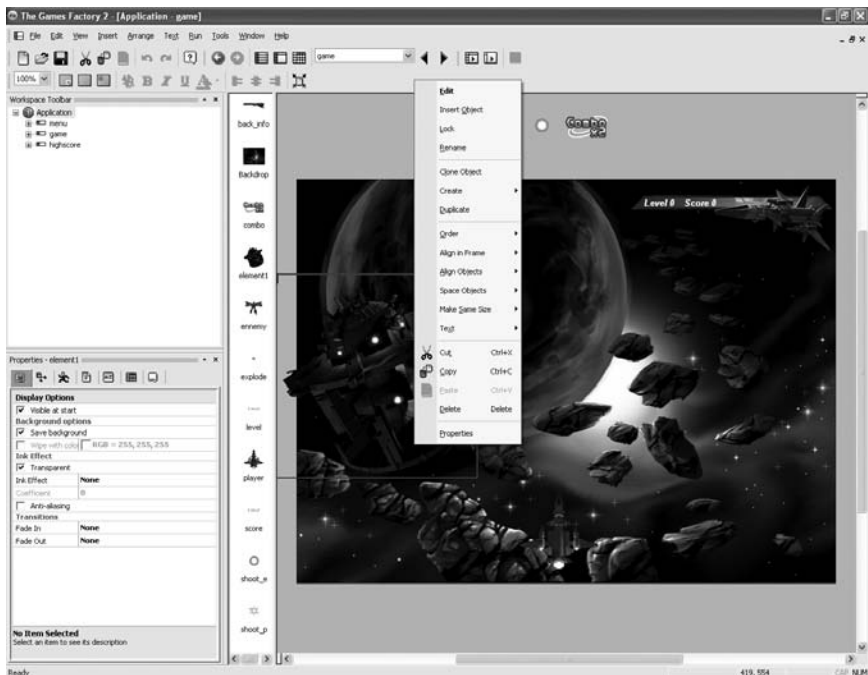


FIGURE 10.9 The pop-up menu displayed when you right-click on an object.

Now that you have had a look at some of the items that make up your game, it is a good idea to familiarize yourself with how it plays, as this will ensure that you understand how it all fits together. There are two ways of playing the game. You can either tell TGF2 to play the current frame (frame 2) or play the whole game from the

start. So that you experience the whole game, run the whole program. You can see the Run Application and Run Frame icons in the toolbar in Figure 10.10.

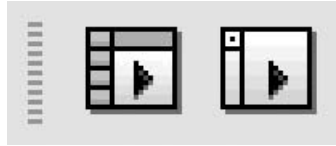


FIGURE 10.10 The Run Application and Run Frame buttons.

Click on the Run Application button and play the game. The controls for the game are the left and right arrow keys (cursor keys) and the space bar to fire the spaceship's weapon. When you have completed playing, you can click on the red cross in the upper-right corner of the game window.

ALIEN WARS: THE EVENT EDITOR

To get to the Event Editor, click on the icon in the toolbar as shown in the previous chapter. You will see a screen that looks like Figure 10.11. The Event Editor is where you will specify what happens in your game. Some of the things you will be doing for this game include:

- Checking for the mouse clicking over a button
- Telling the program to move between frames
- Creating the code to shoot the spaceship bullet when the spacebar is pressed
- Adding sound

The first time you call up the Event Editor it will consist of one horizontal line. It looks like a spreadsheet before you have entered any information. Figure 10.11 shows the Alien Wars game with a number of events that have already been entered. You will have different sets of events for each of the three frames of the game, as each set of events corresponds to that particular frame. Figure 10.11 shows the code for the first frame.

At the top of the Event Editor is a row of icons that represent the possible actions that can happen in your game. A blank game has seven icons that always appear by default. Any icons displayed after this are the objects (graphics, etc.) that have been added to the game. Figure 10.12 shows the objects in this game.

To the right of these event lines are a number of boxes, some of which are blank and some of which contain a tick graphic. Each box lines up with an object icon at the top of the screen, and when the event is true, it runs that action for that particular object.

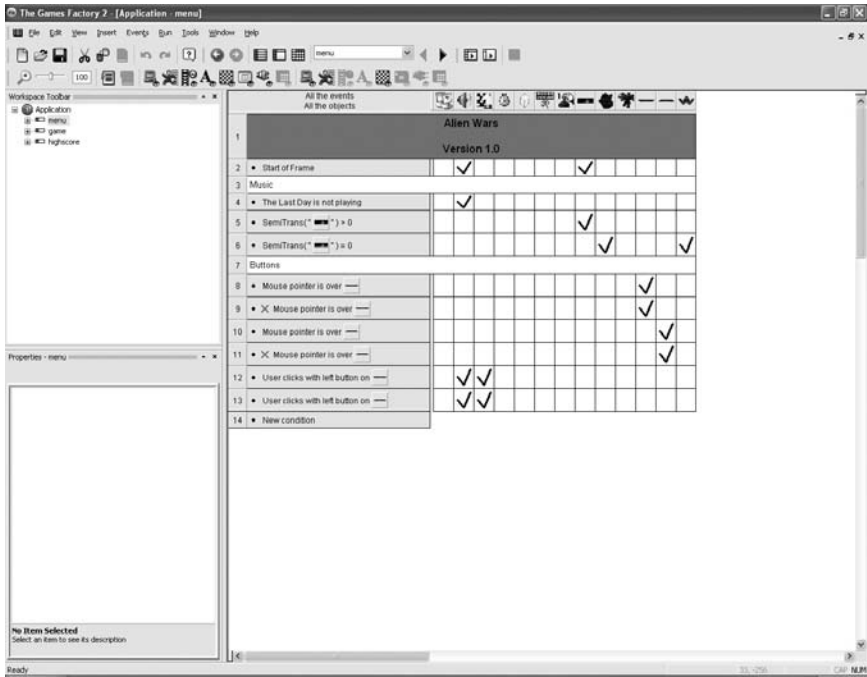


FIGURE 10.11 The Alien Wars Event Editor for the first frame.



FIGURE 10.12 Objects in the Event Editor.

The Object Icons

The first seven icons denote system objects and will appear in every game you create, regardless of if there are any events or objects within it. The following list shows them in order as shown in Figure 10.12

- Special Conditions.** Performs special functions such as enabling and disabling groups, accessing the clipboard, or accessing text or number variables.
- Sound.** Plays music or sample files and allows you to pause, play, stop or select a specific channel on which to play the sound.
- Storyboard Controls.** Allows you to handle the restarting of the game, ending of the game, and moving between the frames in your game.
- Timer.** Sets up a timer.
- Create New Objects.** Allows you to place or create a new object on the screen at certain times or as the result of certain events.

Mouse Pointer and Keyboard. Lets you control how the player interacts with the mouse and keyboard and read certain key presses or mouse movements.

Player 1. Allows you to change the score and lives of the player.

As previously mentioned, the icons shown after the initial seven are objects that have been added to the actual game. The options for these objects vary depending on what the object does. Objects are covered in detail in Chapter 14.

The Events

In the events in this frame, each line is given a number. The first line in this game is a comment line, which is used to document certain aspects of your game. In this case it is a simple version control note. You could also put a copyright notice or a helpful note to explain a difficult bit of code.

Most of the events, which are shown in gray, are readable, and you can get an idea of what they do. Event line two shows the event Start of Frame. This line and its actions run when the frame is first loaded. Once it has loaded, it will never run this line again until the frame is restarted. Line eight checks for when the mouse pointer is over a specific object, in this case when it is over the Play button.

You can see what actions will run when an event is true by moving your mouse to an action box, which contains a tick graphic, and it will appear, as shown in Figure 10.13.

All the events All the objects											
Alien Wars											
Version 1.0											
1											
2	• Start of Frame	✓						✓			
3	Music										
4	• The Last Day is not playing	✓									
5	• SemiTrans(" ") > 0							✓			
6	• SemiTrans(" ") = 0								✓		✓
7	Buttons										
8	• Mouse pointer is over									✓	
9	• ✕ Mouse pointer is over									✓	
10	• Mouse pointer is over										✓
11	• ✕ Mouse pointer is over										✓
12	• User clicks with left button on	✓	✓								

FIGURE 10.13 The actions that are contained in the event.

Each line is called an event, but within each event you can place multiple conditions. A condition is what you want to check for within your game. For example:

- Object is moving
- Object is not moving
- Sound is playing
- Mouse enters a certain area on the screen
- Player has lost all his lives
- Score reaches 100

The conditions can get quite complex, but it is important to remember that if one of the conditions is not true, the event will not run, and the program will continue to the next. Once it has finished reading all the events, it will start back from the top and begin the whole process again. Once the condition is true, the program will run the actions. These actions are run in the order in which they were placed, not in the order in which the objects appear in the Event Editor. Actions are what you want to happen in your game:

- Add to score
- Play a song
- End the game
- Place a message on the screen

Adding to the Event Editor

When you enter events into the Event Editor for the first time, you have the single blank event line. You might want to create a number of possible things. Ensure that you have TGF2 open and click on the New button or click on File | New to create a new application. You need to be in the blank Event Editor for the first frame that has been created by default. Double left-click on the text “Frame 1” in the Workspace toolbar and then click on the Event Editor icon. You are now ready to follow the examples, which give you a quick overview of how to add events, conditions, and actions to your code.

A Comment Line

Comment lines are a great way of putting small bits of information into your game. This allows you to put in your copyright messages or put in notes about a particular bit of code. The second option is very useful if you are working on a difficult bit of code and want to understand why you did something a particular way when you come back to the code after a break.

To add a comment line you will need to:

1. Right-click on the event line number (in a new application or frame that has no events it will be 1).
2. Select Insert | A comment. The comment box will appear as shown in Figure 10.14.

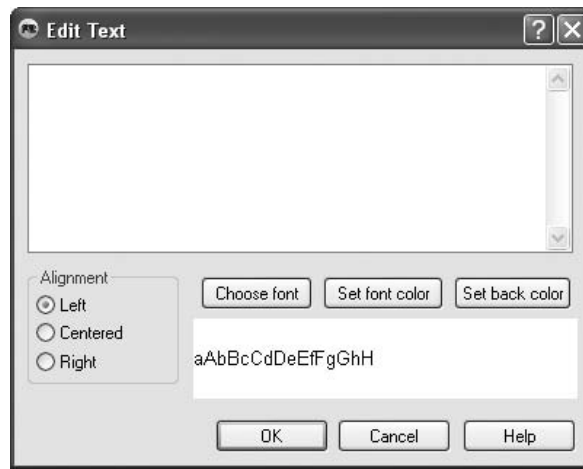


FIGURE 10.14 The enter a comment box.

3. Type in the comment, and if you want, you can change the font, color, and background color.
4. Click OK to close the dialog box and save its contents to the Event Editor.

A Single Condition

If you are adding a single condition to the Event Editor, you can click on the “New condition” text. This is very useful when you are adding an event to the last line of the events. In other words, the New condition option exists at the end of the code. You may want to insert an event in the middle of some already created code, if so you would use the “Add a new Event” option

Using the New Condition Option

If you want to add an event to the very last line of your code (or if there are no events yet it will be the first line of the program):

1. Left-click on the New Condition.

A New Condition dialog box will appear and show a number of icons, as shown in Figure 10.15. These icons represent the seven default objects and any additional objects that you have added to your game. Each of these objects has a set of conditions from which you can select. Remember, a condition is a “check” that the computer will make to see if something has happened. Now create a Start of Frame condition that will run once when the program runs.

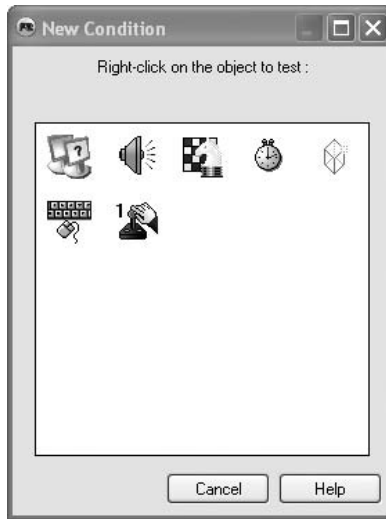


FIGURE 10.15 The New Condition dialog box.

The Start of Frame condition is a Storyboard condition, so right-click on the Storyboard icon (it looks like a horse and a chessboard), and you will see a pop-up menu appear. These are the options that this object has to create conditions on. Figure 10.16 shows this pop-up menu.

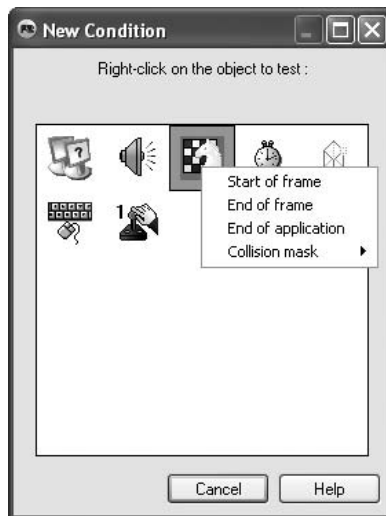


FIGURE 10.16 The Storyboard icon's conditions.



Each object has a set of conditions. Some are similar. Some objects have a lot and others may only have one condition. If an object doesn't have the condition you are looking for, think about another way of achieving what you are trying to do, because you might be selecting the wrong object.

2. Select Start of Frame.

You now have your first condition in a event.

Add Another Condition

If you want to create an event with a number of conditions within it, you cannot click on New Condition, as this will create a separate event.

If you still have a single event and a single condition on your screen, right-click on the Start of Frame text, which should be in event line number 2 if you added a comment line. (If you do not still have an event and a condition on your screen, follow the details in the last section to do this.)

1. From the pop-up menu, select Insert.
2. The New Condition dialog box will appear, allowing you to pick another condition.
3. Select any object and add any condition.

Create a Code Group

Code groups are very useful for putting a selection of code that does a particular job. This makes your code easier to read, but you can also enable and disable code groups at any time.

To add a group:

1. Right-click on any event number and select Insert | A group of events.
 2. A group dialog box will appear as shown in Figure 10.17.
- Type in the title of the group.
 - You can type in a password if you want to protect the group and prevent someone from opening the group if you distribute your code.
 - By default, the group is active when the program or frame is running, but if you want it only to run at a specific time, you can unselect this box and enable the group through an action.
 - Once you are done, click OK. Your group is now be created and looks something like Figure 10.18.



FIGURE 10.17 A blank code group ready to be created.

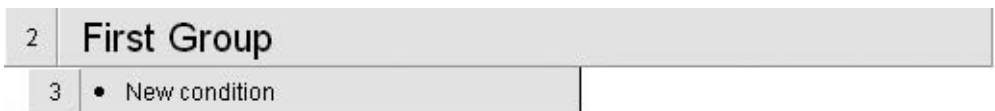


FIGURE 10.18 An example group.

Adding an Action

To add an action you need to move to the right of the event line to which you want to add the action. Consider what action you want to implement, and in all cases it will be specific to a particular object or contained within the seven system objects. For example,

Consider for a moment that you have created a bat-and-ball game in which the ball hits a number of bricks and the player has a bat and tries to keep the ball in play. You have just created a condition that checks for when the ball hits the bat. When you run the game, nothing happens when the ball hits the bat, as you haven't created the action. Therefore, the action you want to apply is to make the ball bounce. As you are going to tell the ball to bounce, you move directly under the ball object and apply a bounce to it. This is how you will apply all of your actions in TGF2 using the same logic.

To add an action, move to the correct event line and then move under the object to which you want to apply the action. Right-click on the white box to reveal a pop-up menu. An example of the actions under the Storyboard Control object is shown in Figure 10.19.

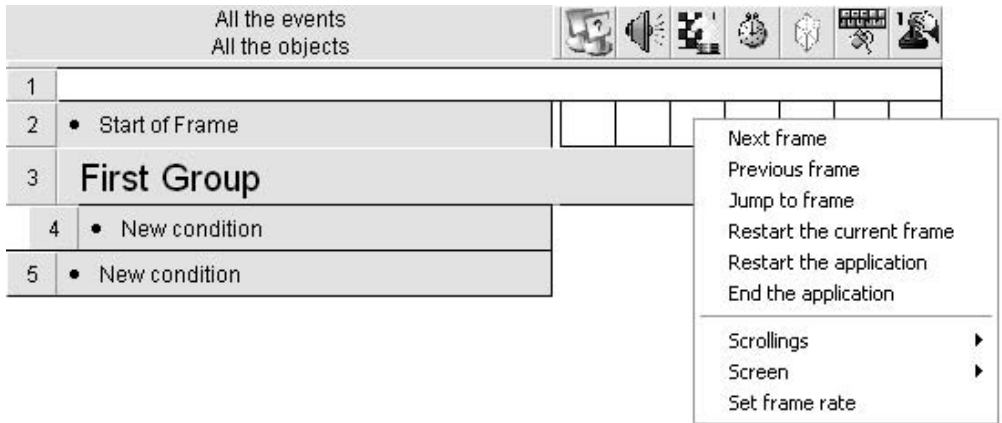


FIGURE 10.19 The Actions menu available to the Storyboard Control's system object.

Select any action from the pop-up menu, and this will place a tick graphic in the box. This tells you there is an action within this location. If you want to add a second action to the same event line and the same object, you can right-click again and select the action.

CHAPTER SUMMARY

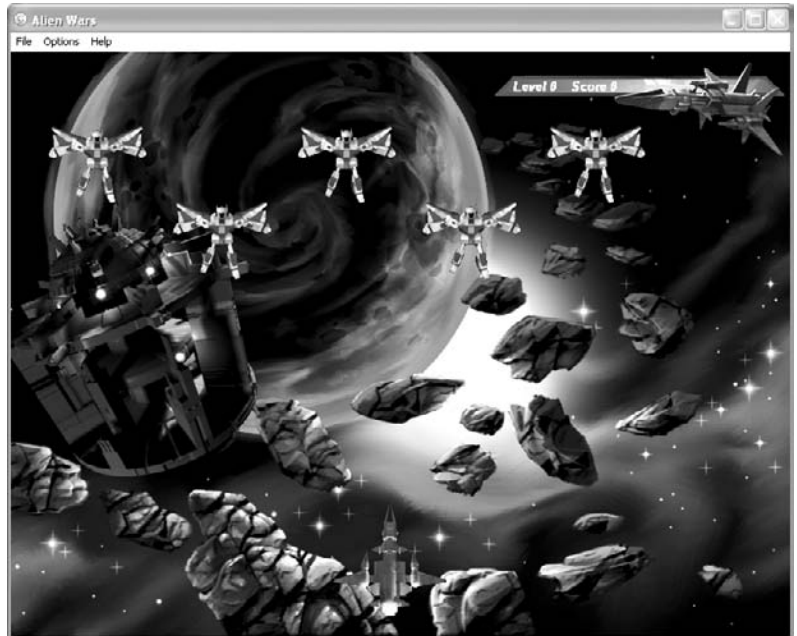
This chapter looked at the game you will be making in the next chapter. You will build your game frame by frame, and by the end, you should have a pretty good idea of how to use TGF2 and how to program in the Event Editor.

This page intentionally left blank

ALIEN WARS

In This Chapter

- Library
- Initial Setup
- Event Programming



In this chapter you will be creating the space shoot-'em-up game discussed in Chapter 10. You will need to create your game file, set up your frames and the objects on screen, and then program the game to react to the player's key presses.

LIBRARY

The Library toolbar is a useful way of adding objects and items already created onto your blank frames. So you don't have to spend a lot of time drawing the spaceships and backgrounds, this has already been done for you. Before you start, you need to connect to a library file that contains all of your objects. The Library toolbar is in the bottom part of the TGF2 screen as shown in Figure 11.1.



FIGURE 11.1 The blank Library toolbar.

If you do not see the toolbar, you need to display it by selecting View | Toolbars | Library Window.

You now need to connect this Library window to the library file that has already been created for you. To do this:

1. Right-click on the left windowpane, and a pop-up menu will appear. Select New.
2. A Browse for Folder dialog box allows you to search for the folder that contains the library. Navigate to the DVD provided with this book.
3. Navigate to the TGFFILES\Alien Wars\Lib folder and then click OK.
4. You can now type the name of your library folder, so in the left windowpane where you see the words New Library type "Alien Wars."
5. Clicking on the words Alien Wars in the left-hand pane will reveal the library file in the right-hand window, as shown in Figure 11.2.

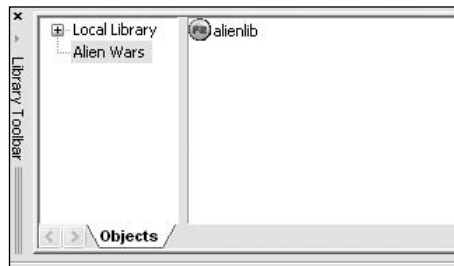


FIGURE 11.2 The Alien Wars library file.

6. Double left-click on alienlib to reveal the frame folders you will be using to set up your frames.

INITIAL SETUP

Begin with creating your game file, creating the frames that will represent your screens within the program.

First, you need to create a TGF2 file:

1. Click on New or select File | New.
2. Your TGF2 game file will have been created as shown in Figure 11.3.

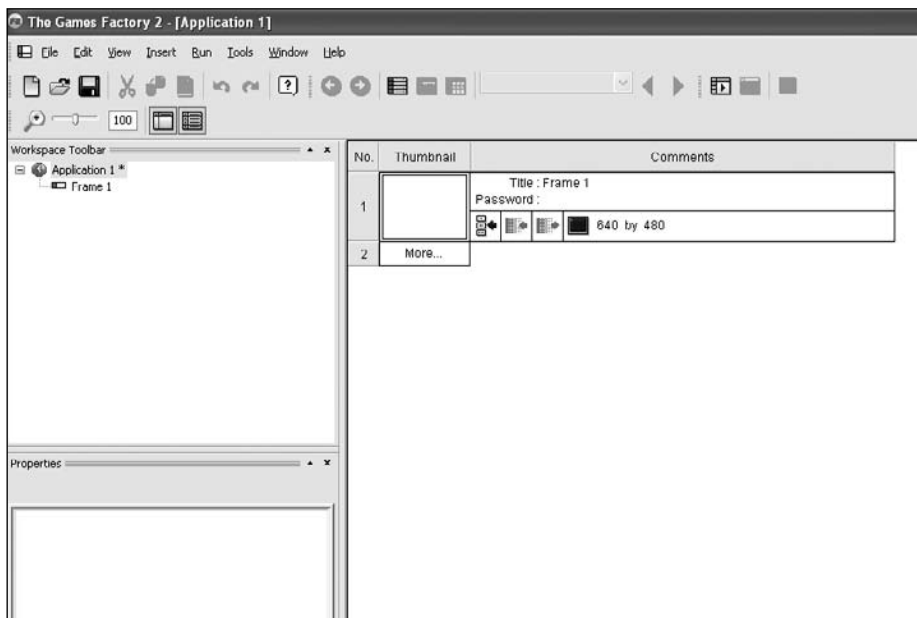


FIGURE 11.3 Your blank game file ready to be configured.

Notice that the Storyboard Editor shows that the only frame is set to a size of 640×480 . The game you are about to create works on a 800×600 screen resolution. TGF2 has a frame resolution and an application resolution, and changing the application resolution changes the size of the current frame and any additional ones you create.

3. Single left-click on Application 1 in the Workspace toolbar.
4. This reveals the application properties information in the Properties toolbar, as shown in Figure 11.4.

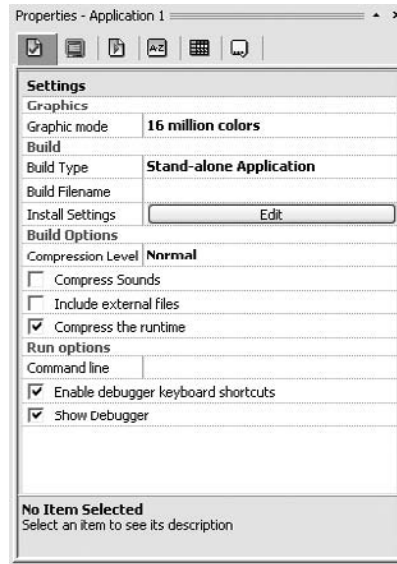


FIGURE 11.4 The Properties window displaying application information.

5. At the top of the Properties window in Figure 11.4 are a number of tabs. Click on the Windows tab. This is the one that looks like a monitor.
6. The first item in the Properties window is now the Size item, and it is set to 640 × 480. Click on the box, and an arrow will appear. Click on this arrow and select 800 × 600 from the drop-down menu.
7. A dialog box asks if you want to modify the frames to the same as the newly set application size, as shown in Figure 11.5. Click Yes to agree to this.
8. You can see the changed settings in the Properties Application window in Figure 11.6.



FIGURE 11.5 Changing all frames to the application size.



In some cases when you click on a dialog box the Properties window will become blank. To display the correct information click on the object you are interested in. For example, if you were viewing the Application Properties, click on the application name in the Workspace toolbar.

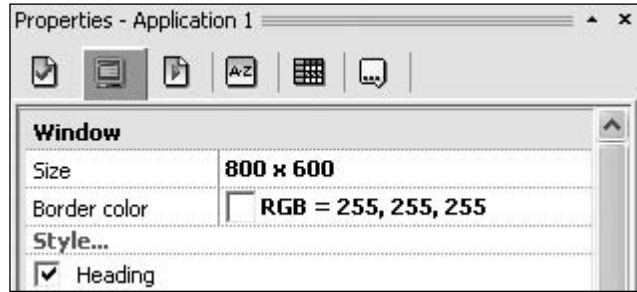


FIGURE 11.6 The changed Application window size.

Creating and Renaming Frames

You have three frames in your game, and by default the program only has one, so you need to create two more.

1. Right-click on the application name and select New Frame as shown in Figure 11.7.

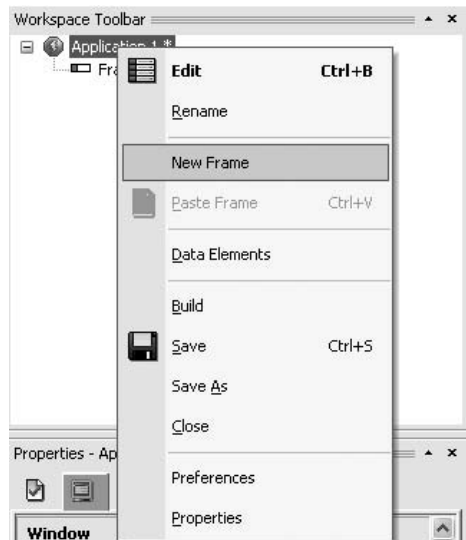


FIGURE 11.7 The Rename option using the right mouse click.

2. This creates a second frame called Frame 2.
3. Right-click on the application name again and select New Frame. This creates a third frame called Frame 3.

You now have all three frames in place and are ready to rename them so they are easier to identify. This identification is not such a problem in a small game like this, but some games could run to a couple of hundred frames, and at this point it can get very confusing as to what each frame does. It is good to get into the habit of giving your frames proper names, as this will be very helpful in larger projects.

4. Right-click on Frame 1 text and select Rename from the pop-up menu. Type “menu” into the selected text box.
5. Right-click on Frame 2, select Rename, and then type “game.”
6. Once more, right-click on Frame 3, and select Rename, and then type “highscores.”

You have renamed your frames, but it is also good practice to rename your application. This name is particularly important because when you run your game in a window, the application name is the name that will appear in the top bar of the window.

7. Right-click on Application 1 in the Workspace toolbar.
8. Select Rename and type “Alien Wars.”

Your Workspace toolbar should now look like Figure 11.8.

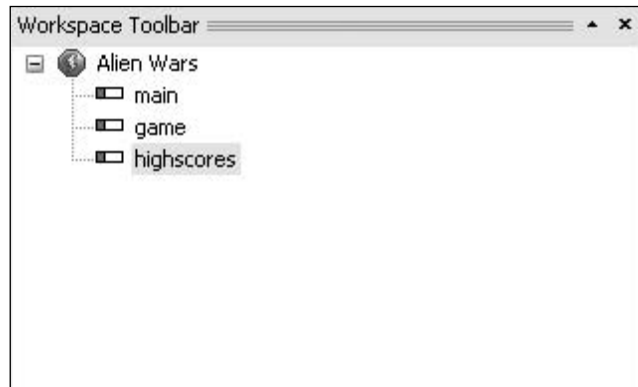


FIGURE 11.8 The current state of the Workspace toolbar.

Main Menu Frame Setup

You now need to place all the objects you are going to use in your game for the main menu frame. To do this you need to ensure that you are on the correct frame and that you have the correct objects for the frame ready to drop into place.

1. In the Library toolbar, double left-click on the library file alienlib.
2. You will now see the names of the frames you are going to use in our game, as shown in Figure 11.9. This library file is an exact replica of the frames you have and contains the objects for each frame that you need to place.

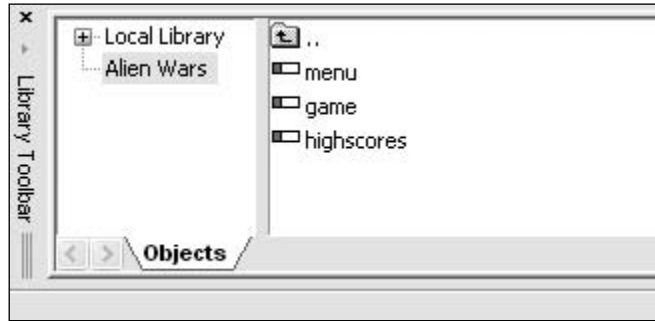


FIGURE 11.9 Each frame in the library contains all the objects you need to create the game.

3. Double left-click on Menu in the Library toolbar to display all the available objects as shown in Figure 11.10.



FIGURE 11.10 The library objects for the main menu.

You are now ready to drag and drop items from the library onto the frame, but you need to ensure that you are on the frame editor for the correct frame, which in this case is the recently renamed frame Main.

4. Click on the number 1 in the Storyboard Editor to access the blank main frame.

You now have the blank frame in front of you, ready for you to begin placing items onto the screen as shown in Figure 11.11. The box that has been highlighted tells you which frame you are on so you can make sure you are changing the correct blank frame.

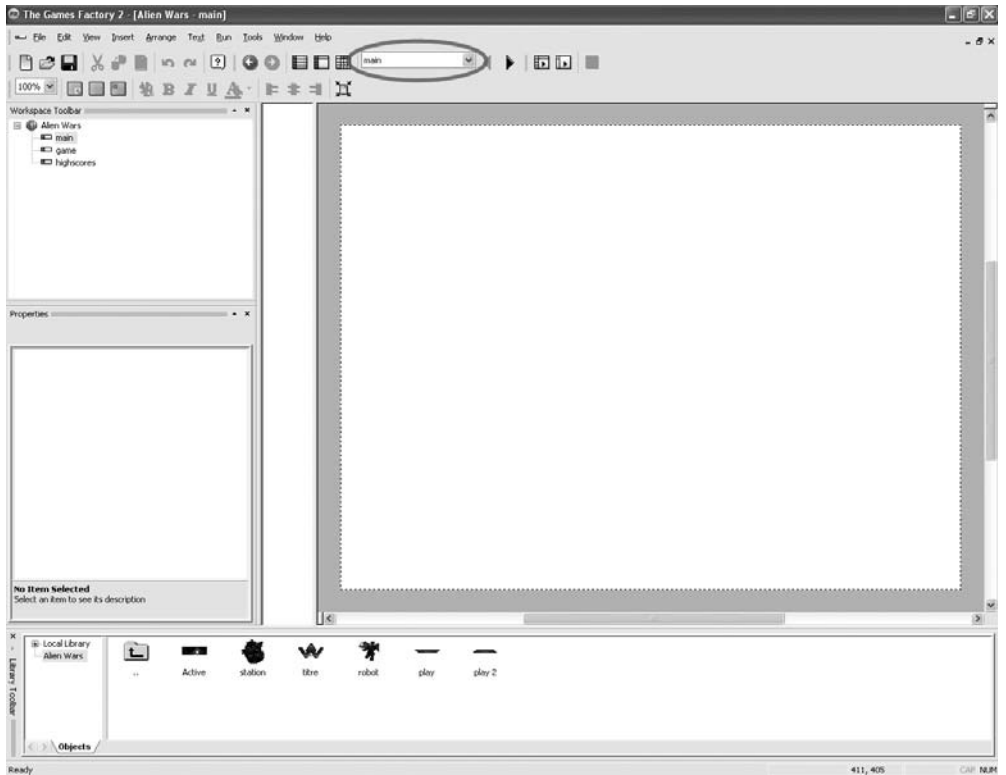


FIGURE 11.11 The blank Main frame ready for dropping the objects onto it.

Setting Up the Scene

It's time to create the scene for the main frame by dragging objects from the Library toolbar and placing them in particular positions.

1. Left-click and then hold the left mouse button on the object called Space_ background and drag it onto the blank frame.
2. It doesn't matter where you dropped it because you are going to precisely place it using the object's properties. Once you have dropped it onto the frame, single left-click on it to display its properties in the Properties window.
3. Click on the Size/Position tab to display the object's current position.
4. Change the X position to 0 and the Y position to 301. This places the item in the middle of the screen.

The graphic is a dark color, and the background of the frame is currently white. You need to change this so that it fits in with the rest of the game, which, as it's a space game, is based on black.

5. To change the frame background color, click on Main in the Workspace toolbar to access the frame's properties.
6. On the first tab displayed the Background color option is currently configured as white. Click on the box to display the color picker as shown in Figure 11.12.

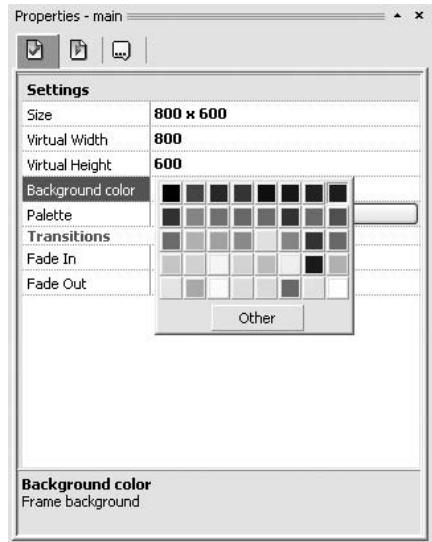


FIGURE 11.12 The color picker, where you can select a background color.

7. Select black from the color picker.

You now need to place all the other items on the screen, using the same process. Use the details in Table 11.1 to position them.

Table 11.1 Objects for the Main Frame

OBJECT NAME	X	Y
Station	-225	42
Btn_Play	-260	475
Btn_Quit	-260	534
Title	824	47
Robot	823	121

Your completed screen will look something like Figure 11.13, which is zoomed out so that you can see where the items are located.

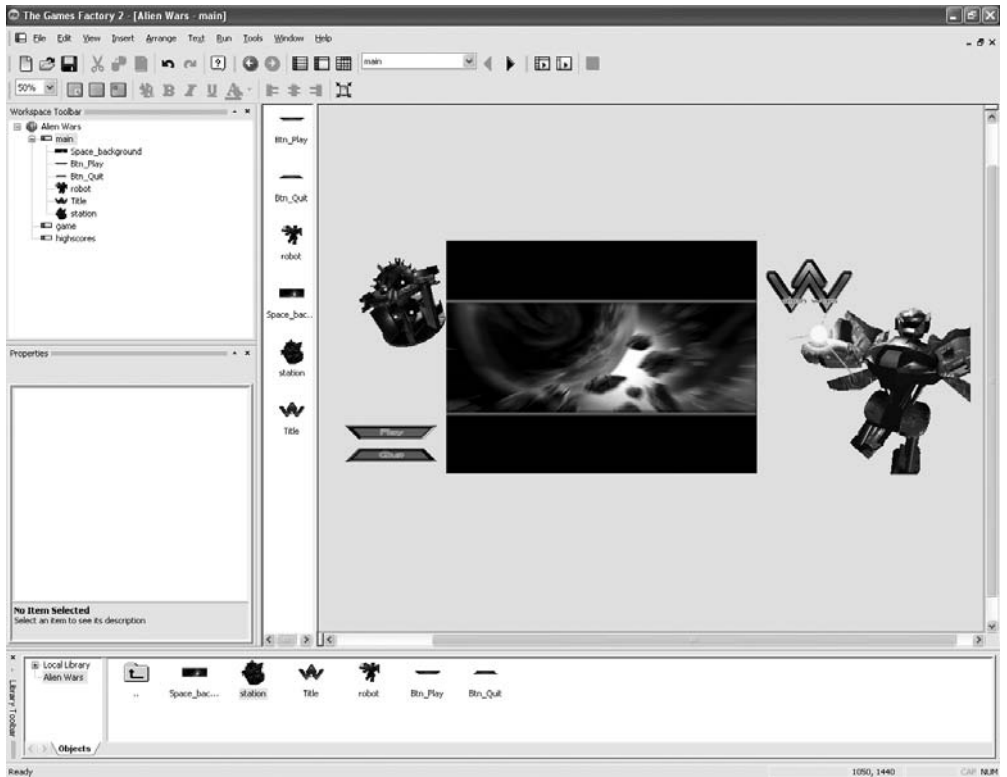


FIGURE 11.13 The setup of the Main frame.

If you click on the Run Frame button or press the F7 button, the frame's objects will snap into place automatically. This is because all items have had a movement applied to them already. You will learn more about movements in Chapter 14.

The Game's Frame Setup

You now need to place all of the objects you are going to use in your game for the game frame. Make sure you are on the correct frame:

1. Double left-click on Game in the Workspace toolbar to display a blank frame.
2. You can confirm you are on the correct frame by checking the text in the Frame Name box on the Button toolbar.

Changing the Library Folder

You need to change the library file location, as currently it is pointing to objects in the first frame.

1. In the Library toolbar are objects that you dragged onto the frame for the Main frame and a yellow folder graphic with an up-pointing arrow. Double left-click on the yellow folder.
2. You can now see the three frame folders. Double left-click on the game folder in the Library toolbar to display the contents of the game folder as shown in Figure 11.14.

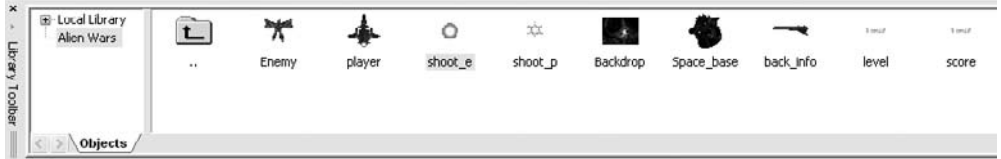


FIGURE 11.14 The game library items.

Setting Up the Scene

It's time to create the scene for the game frame. Again, you need to drag objects from the Library toolbar and place them in particular positions.

1. Left-click and hold the left mouse button on the object called Backdrop and drag it onto the blank frame.
2. Once you have dropped it onto the frame, left-click on it to display its properties in the Properties window.
3. Click on the Size/Position tab to display the object's current position.
4. Change the X position to 0 and the Y position to 0. This places the item in the precise position on the screen.

Now place all of the other items onto the frame following the coordinates given in Table 11.2.

Table 11.2 Game Frame Object Positions

OBJECT NAME	X	Y
Space_base	-33	143
Player	325	462
Enemy	51	-261
Back_Info	496	4
Level	516	27
Score	575	27
Shoot_p	320	-79
Shoot_e	363	-80

Your completed screen will look something like Figure 11.15, which is zoomed out so you can see where the items are located.

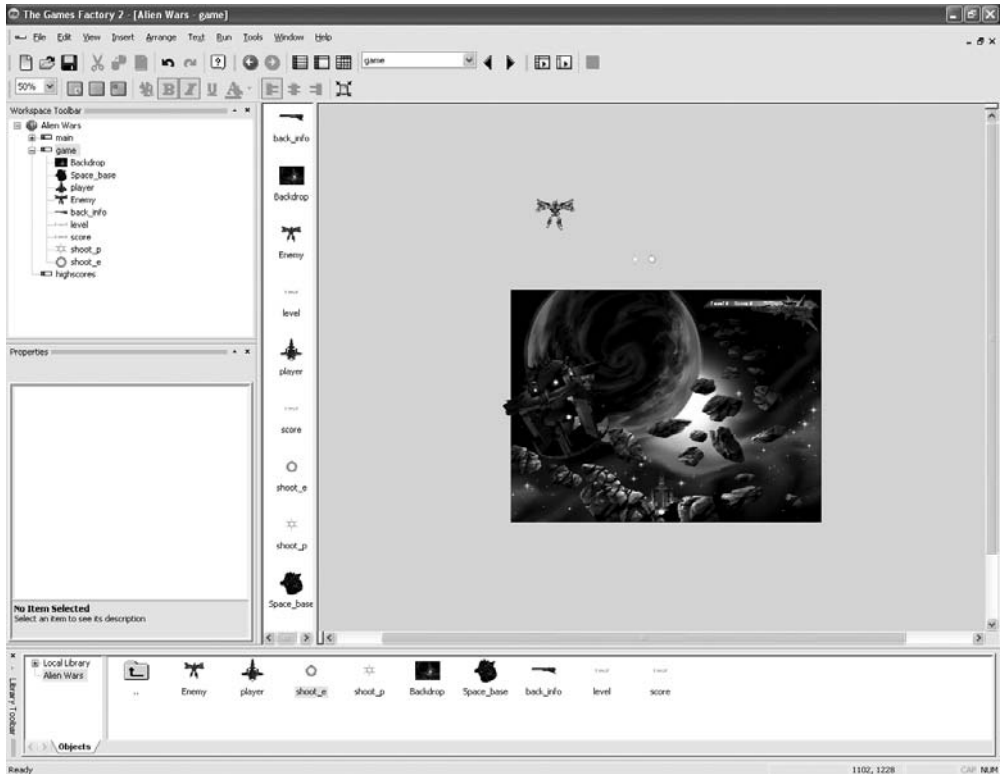


FIGURE 11.15 The game frame with objects all placed in their correct positions.

If you click on the Run Frame button or press F7, you will notice that a space station is spinning on the screen, and you can control the spaceship using the arrow keys (cursor keys).

Highscores Frame Setup

You now need to set up the scene for the final frame, in which the player will see the highscores table.

Let's ensure that you are on the correct frame:

1. Double left-click on Highscores in the Workspace toolbar to display a blank frame.
2. You can confirm you are on the correct frame by checking the text in the Frame Name box on the Button toolbar.

Changing the Library Folder

You need to change the library file location, as currently it is pointing to objects in the second frame.

1. Double left-click on the yellow folder with the up-pointing arrow in the Library toolbar.
2. You can now see the three frame folders. Double left-click on the highscores folder in the Library toolbar to display the contents of the game folder, as shown in Figure 11.16.



FIGURE 11.16 The highscores library items.

Setting Up the Scene

It's time to create the scene for the highscores frame. Again, you need to drag objects from the Library toolbar and place them in particular positions.

1. Left-click and hold the left mouse button on the object called Space_Background and drag it onto the blank frame.
2. Once you have dropped it on to the frame, left-click on it to display its properties in the Properties window.
3. Click on the Size/Position tab to display the object's current position.
4. Change the X position to 0 and the Y position to 301. This will place the item in the precise position on the screen.
5. Change the frame background by clicking on Highscores in the Workspace toolbar to access the frame's properties.
6. On the first tab click on the white box next to Background color.
7. Select black from the color picker.

Now place all the other items onto the frame according to the coordinates given in Table 11.3.

Table 11.3 The Highscores Frame Object Positions

OBJECT NAME	X	Y
Station	-225	42
Robot	823	121
Title	824	47
Hi-Score	211	457

Your completed screen will look something like Figure 11.17, which is zoomed out so you can see where the items are located.

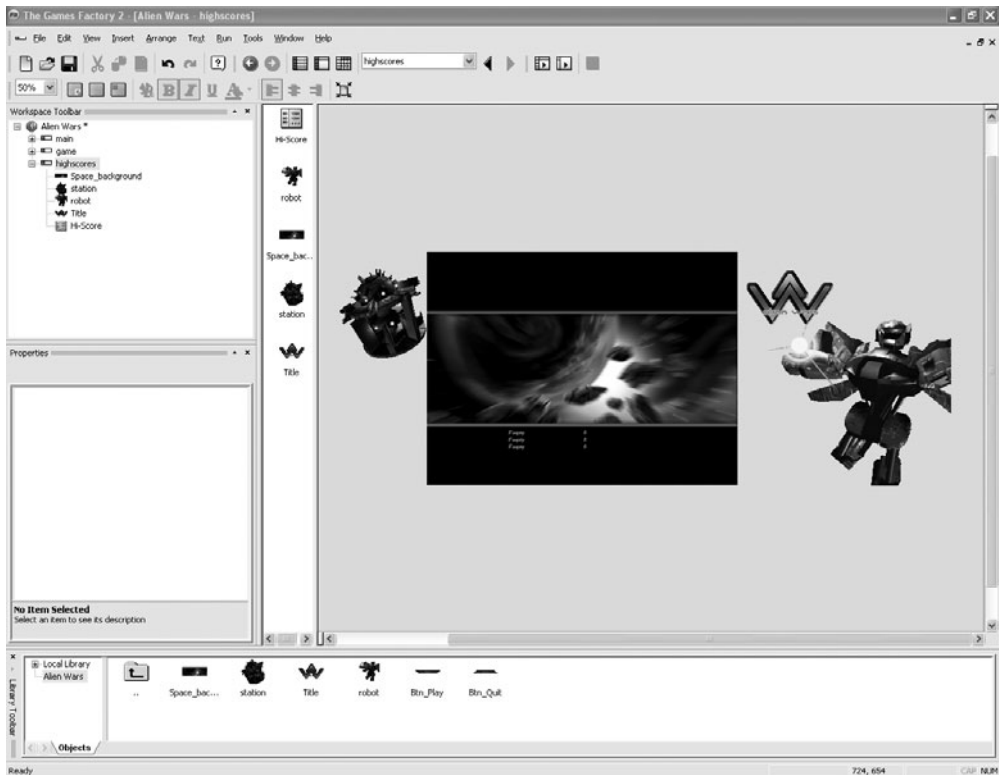


FIGURE 11.17 The highscores frame with objects all placed in their correct positions.

If you click on the Run Frame button or press F7, you will see a space station spinning on screen, and you can control the spaceship using the arrow keys (cursor keys).

EVENT PROGRAMMING

In this section you will begin to program the events that bring together all three frames and the objects within them.

You will work on each frame separately, which makes it a lot easier to get something working quickly. The major benefit of the frame system is that you can pick and choose the sections you want to work on and leave the difficult bits till later. As this is a simple game, you will work on the frames in order.

Programming the Main Frame

The main frame is the first screen the player is introduced to, and its main goal is to direct the player to the areas he wants to go to in the game. In Alien Wars the player can either click on a button to play the game or quit. More complicated games may have a number of different paths the player can choose from. Placing a menu screen in your game makes this process a lot easier.

Before you start, you need to make sure you are on the correct frame:

1. Double left-click on Main in the Workspace toolbar.
2. You will see the main frame and its objects all in place.

To begin coding you need to be in the Event Editor, so click on the Event Editor button on the toolbar.

3. You will now see a blank Event Editor, ready for you to begin coding.

Main Frame Components

In the main frame you want to achieve several things:

- Create a comment to explain the game's name and version number.
- Start playing some music.
- Make some objects fade into the screen.
- Program the buttons to react when the player moves the mouse over them.
- Program the game to either quit or move to the game frame when these buttons are clicked.

Creating the Note Event

The first bit of event program to do actually isn't coding at all. You need to add a message to the start of the game, detailing the game's name and the version number. In your own games you could enter more information if required.

1. Right-click on event number 1 and select Insert | A comment.
2. The comment dialog box appears.
3. Type "Alien Wars" in the text box and while still in the text box, press the return key twice to create a little more space.
4. Type "Version 1.0."

5. You need to amend the font to make it stand out a little more, so click on Choose Font, select the font style Bold, and change the text Size to 12. The font dialog box will be configured as shown in Figure 11.18.

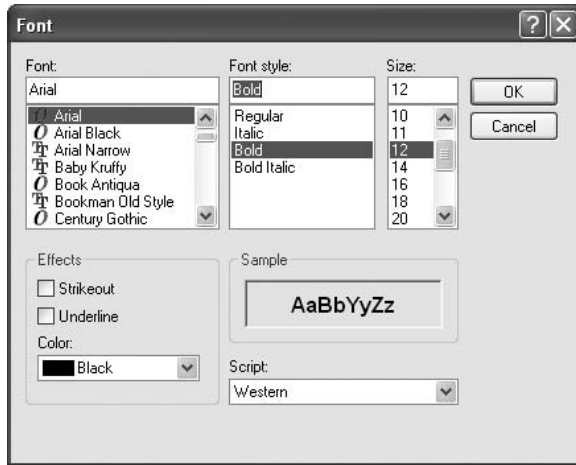


FIGURE 11.18 The Font dialog box.

6. Click OK to save the information to the Comment dialog box.

You can make your comments stand out even more by changing the background color. Some developers create colored comments to highlight certain aspects of their game so that if they come back to it after a long time they will be able to understand it better. In this example you are just going to change the background color so that it looks nice on the Event Editor.

7. Click on the Set Back Color button.
8. In the Color dialog box that appears, select any color you like. In the version of the game on the DVD the second row, second to last color was selected, as shown in Figure 11.19.
9. Click OK button to save this information to the Comment dialog.



Finally, this text will look better if it is centered on the screen. Currently it is set to the left.

10. Click on the Centered radio button.
11. Click OK to save the comment information to the Event Editor. You should now have the results shown in Figure 11.20.

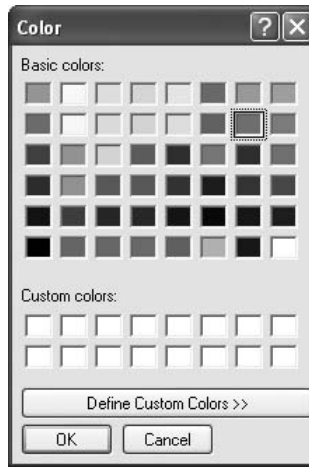


FIGURE 11.19 The Color picker with the color selected.



FIGURE 11.20 The comment displayed in the Event Editor.

Start of Frame Event

At the very start of the frame you want to play some music and set the transparency of an object to make it invisible. These actions are already in a Start of Frame event, so as the frame loads this is the first event it will run. You could consider the start of frame event as an initializing event, where you configure everything before the program appears.

You want to play some music at the very beginning of the game to create atmosphere and make the whole game experience more positive for the player.

Start with adding the following event:

1. Click on New Condition.
2. In the New Condition dialog box right-click on the Storyboard Controls icon (the horse and chessboard icon).
3. From the pop-up menu select Start of Frame.
4. You will now have a new event on event line 2, and it will read “Start of Frame.”

This event will occur only at the start of the frame. After the program has run this event, it won't run it again, so it's a good event for doing one-off initializations or configuring of your game. The first action is to play the game song at the very start of the frame, and it will continue to play after the Start of Frame event has finished.

5. Move to the right of the Start of Frame event, until you are directly under the Sound object (the icon looks like a speaker).
6. Right-click on the blank action box to display the actions available for this object and then select the Sound option to reveal all options that relate to this menu, as shown in Figure 11.21.

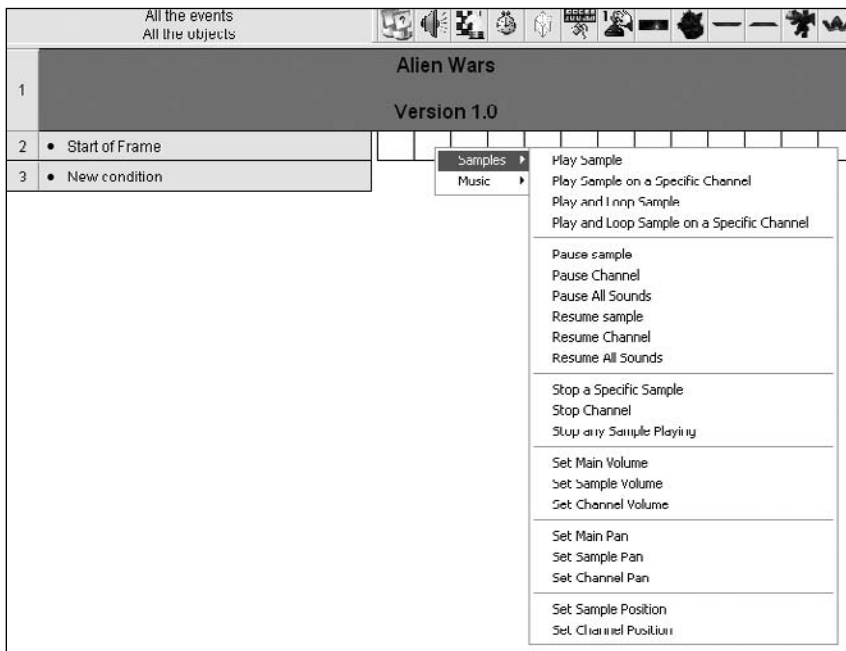


FIGURE 11.21 Actions for the Sound object samples menu.

7. As you can see in Figure 11.21, many options are available. Select Stop any sample playing.

You might be wondering why you should stop the music before you have even started playing any. This is to prevent any future issues with sounds overlapping. This is not so much of a problem in this game, but you can configure sounds to play over frames and to play when the user clicks on something. If you do not stop any samples first, you could end up with the same sound repeating itself over and over again. This can make the sound appear very messy, so from a point of view of safe programming it is always sensible to use common sense event programming. These safeguards are

not necessary for every game you create, but it is good practice and will save you some time fixing programming bugs and problems in your game later on.

Now you can program the music to play:

8. Still on event line 2 and still under the Sound icon, right-click on the box, which now contains a tick graphic.
9. From the options that appear, select Samples | Play sample.
10. A Play Sample dialog box will now appear.
11. Click on the Browse button opposite From a file option. This brings up the Open dialog window. Navigate to the DVD provided with this book to the folder TGFFILES\Alien Wars\Sounds, as shown in Figure 11.22.

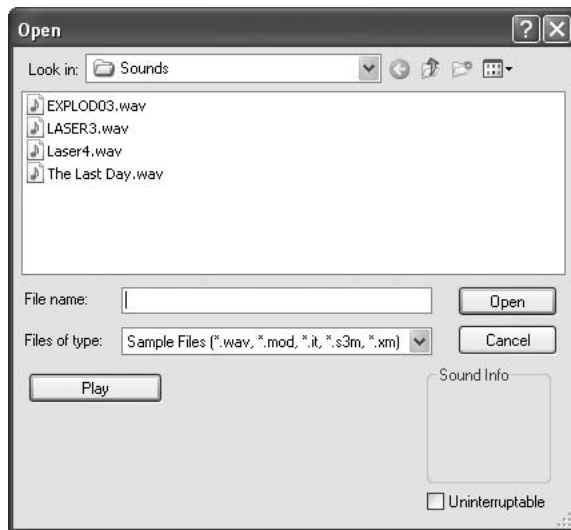


FIGURE 11.22 The Play Sample dialog box.

12. Select the file `The Last Day.wav`. You can click on the Play button if you want to hear the sound before you insert it into the TGF2 file. This is very useful if you have lots of sound files and want to confirm what it sounds like before adding it.
13. When `The Last Day.wav` file is selected, click the Open button. This saves the information to the Event Editor.

Before going any further, run the frame by pressing the F7 key. You will hear the music playing, and all of the objects will appear from off the screen and snap into position. Now it's time to add a little fade-in effect for the background, as it will make the screen a little more animated and give it more of a professional feel.

Still using the Start of Frame event, you will be changing the transparency of the `Space_background` object.

14. Move to the right of the Start of Frame event until you are directly under the Space_background object.
15. Right-click on the blank action box and from the large list of items on the pop-up menu choose Visibility | Set semi-transparency.
16. The Expression Evaluator will ask you to enter a number from 0 to 128 as shown in Figure 11.23. Click OK to save the information to the Event Editor.

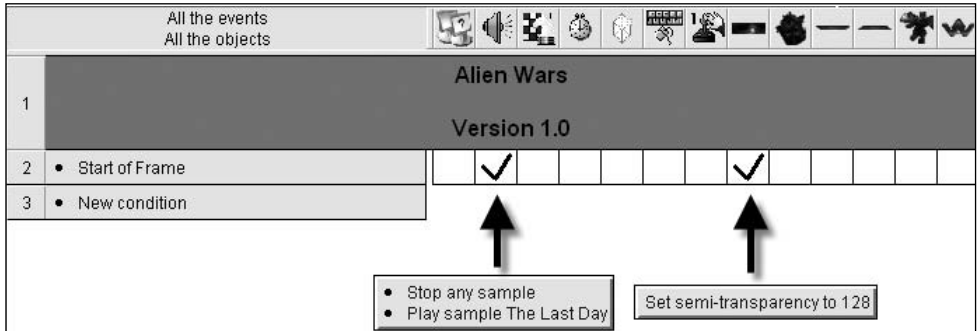


FIGURE 11.23 The dialog box to enter the transparency setting.

In TGF2 transparency for an object works on a scale between 0 and 128, 128 means the object is totally invisible, and 0 means the object is fully visible. In between these two numbers are varying states of transparency.

You have now completed the Start of Frame event, and you can see its corresponding actions in Figure 11.24.

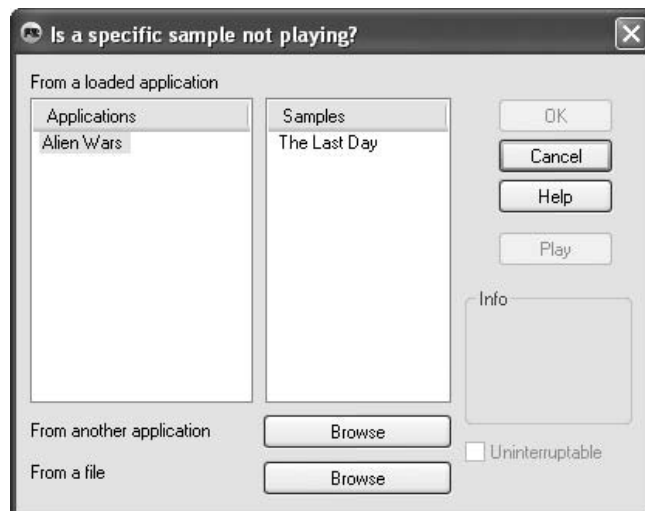


FIGURE 11.24 The actions for the Start of Frame event.

If you run the game now, you will notice that the background has disappeared. This is only a temporary problem, as you will be programming it to appear over a period of time.

Comment Line

Before starting more events, create another comment line to separate the code and make it easier to read and to describe what the code below it is doing.

1. Right-click on event line number 3.
2. Select Insert | A Comment and in the comment box type “Music.” Then click OK.

Music Is Not Playing

The next event to create will determine if the music has stopped playing. If it has, you will want to start the music again.

1. Click on New Condition on event line 4.
2. Right-click on the Speaker object and then from the menu choose Samples | Is a specific sample not playing.
3. A dialog box will appear as shown in Figure 11.25.

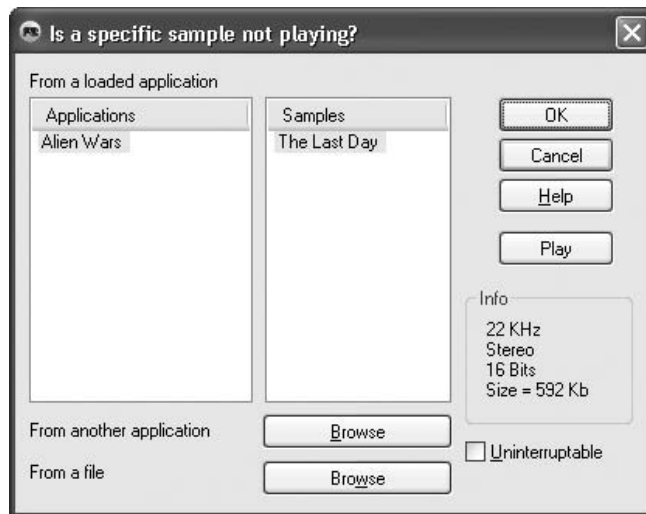


FIGURE 11.25 A browse dialog box.

4. The great thing about any sound files already added is that they will appear in the dialog box automatically, as shown in Figure 11.25, so you don't need to browse the DVD for them. Left-click on The Last Day and then click OK to save the information to the Event Editor.

Now that you have added the event, you need to create an action where the sound plays. This is the same process you followed for the Start of Frame event.

5. On the event line 4 move across until you are under the Sound object and right-click on Samples | Play Sample.
6. In the dialog box that appears, left-click on The Last Day and then click OK.

Comment Line

You need to add a comment line for the next bit of coding, which will set the transparency back to 0.

1. Right-click on event line 5 and select Insert | A comment.
2. In the Edit box type “Transparency” and then click OK.

Setting the Transparency to 0

You will create an event that will check to see if the transparency of your Space_background object is at 0. If it isn't, the action will remove 8 from the object's transparency level. Remember, you set the Space_background object's transparency to 128. An object is invisible at 128 and visible at 0. Once the object reaches 0, you do not need to run this event any more, as it will be fully visible to the player.

The way to check the current value of the object's transparency against a number (in this case 0) is to use the compare two general values option. This allows you to do math comparisons on two different numbers.

1. Click on New Condition on event line 6.
2. Select the Special object, which looks like a computer monitor. Then choose Compare two general values.
3. The Expression Evaluator will appear, requiring you to enter two numbers and select a comparison option.
4. Select the first box, which contains the number 0, and as you want to get the current value of the Space_background object, do this by clicking on the Retrieve data from object button.
5. From the New Expression dialog box that appears, you need to select the object you want to get the information from. Right-click on the Space_background object. Then select Animation | Get semi transparency ratio.
6. It will now read :

```
SemiTrans( "Space_background" )
```

7. In the drop-down box select Greater, as you only want this event to work when the transparency is above 0.
8. You can leave the final box as 0. The Expression Evaluator is now configured as shown in Figure 11.26. Click OK to save the event.

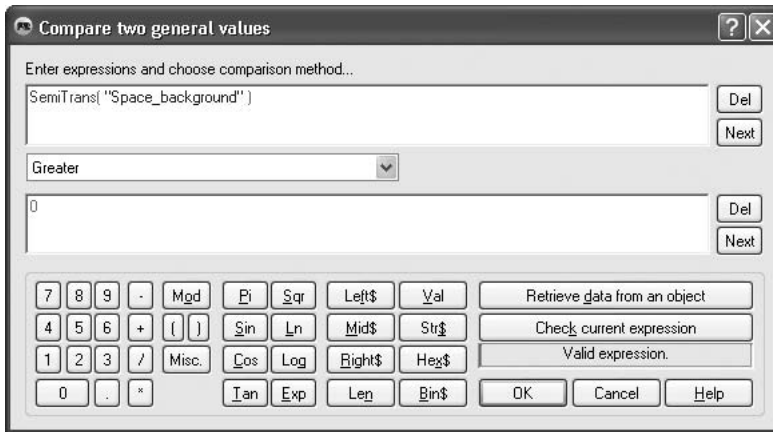


FIGURE 11.26 Comparing the transparency of an object.

For the action, you need to reset the transparency of Space_background to a new number. The aim is to get the current transparency, remove eight from it, and then place the result back into the object so that it is the right amount. To do this you use a standard technique, which is getting the current value, removing a number, and then resetting the result to the value. The following example shows how it works.

- Transparency of Space_background = Starting value 128
- Current Transparency value (128) – 8 = Current Transparency value
- Current Transparency value (120) – 8 = Current Transparency value
- Current Transparency value (112) – 8 = Current Transparency value

Using this method makes the transparency value decrease in value over time, that time being defined by the event frequency.

Now you can add the action to decrease the transparency value of the Space_background.

1. Moving across from event line 6, make sure you are directly under the Space_background object.
2. Right-click on the action box and select Visibility | Set semi-transparency.
3. In the Expression Evaluator dialog box, click the Retrieve data from an object button.
4. Right-click the Space_background object from the New Expression dialog box and then from the pop-up choose Animation | Get semi-transparency ratio.
5. The line will be in the Expression Evaluator, but this is just the current value, so you need to type in “-8” at the end of the expression and then click OK. You can see the expression in Figure 11.27.

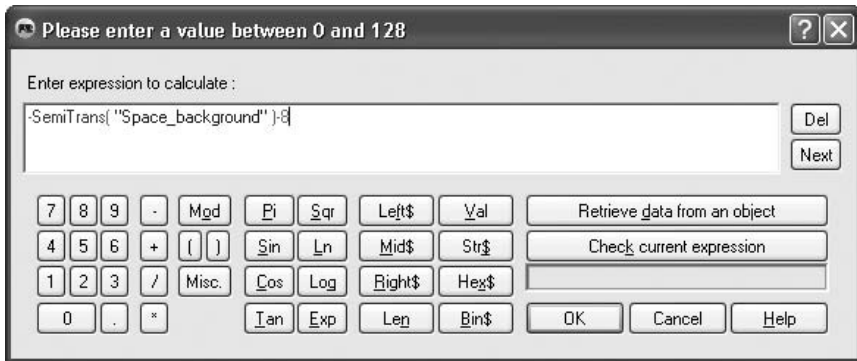


FIGURE 11.27 The Expression Evaluator with the current value minus 8.

The speed at which the object appears is controlled by the speed of the game. By default, the game is set to run at 50 frames per second, so this is 50 times a second. This means the transparency effect will run and be over in less than half a second. If you want to change the time it takes to run the fade-in effect on the background, you can change the number 8. If you change it to a higher number, it will complete the fade quicker, and if you lower the number it will take longer.

You can see the current events and the newly added condition and action in Figure 11.28.

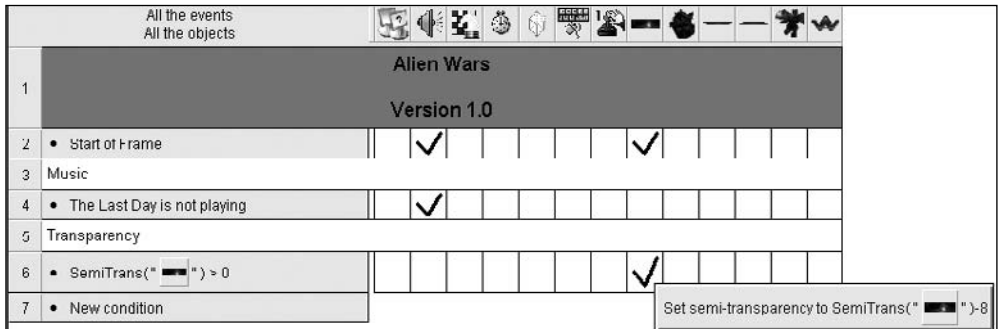


FIGURE 11.28 Current progress in the game.

Add Comment

You need to add a single comment line to show where your next bit of code is going to be placed, and this comment shows where your button events will be.

1. Right-click on event line number 7 and select Insert | A comment.
2. In the Edit box type “Buttons” and then click OK.

Button Effects

If you run the game, you will notice that the animations appear and the background fades in. You can move the mouse around the screen, but moving over the buttons has no effect. When possible, you should design your game interface so that it is easy for the players to navigate around, but also provide them with graphical feedback to show them what they can and can't click on. This game has two buttons on the main frame that will take the user to the game frame or exit the program. These buttons need to be animated so that they light up when the mouse is over them. This is a very simple yet effective way of making your program a better experience for the player.

Each image has two animation frames. The first frame is of a normal button, and second frame is the button in a highlighted state. When the user moves the mouse over the object, the relevant animation frame will change. Don't worry too much about animations yet, as they will be covered in more detail in Chapter 15.

First, you need to add the event condition for when the mouse is over the Btn_Play object:

1. Click on New Condition on event line 8.
2. When the dialog box appears, right-click on The mouse pointer and keyboard object, as this controls all aspects of input from those devices.
3. From the pop-up menu choose The mouse, as you want to check the mouse over this object. Then select Check for mouse pointer over an object.
4. You will then be asked which object to check that the mouse pointer is over. Choose the Btn_Play object.

The event is now added, so now you need to add the action, which is changing the animation frame to the frame that is highlighted.

5. Move to the right of this event until you are directly under the Btn_Play object.
6. Right-click on the blank action box and then select Animation | Change | Animation Frame.
7. The Expression Evaluator advises you that the first frame is equal to 0, so you need frame 1. Type in the number 1 and click OK.

If you run the frame now and move your mouse over the Play button, it will become highlighted. TGF2's Event Editor now knows how to turn the animation on, but when you move the mouse off the object, it stays on, so you need to turn the animation off when the mouse isn't over the object.



As TGF2 is drag and drop, you can quickly drag and drop items from one event to other events and actions of the same object type to another object. This can really speed up your development time as you copy the original and make a small change to the copied events and actions. For now, while you are getting used to make conditions and events, continue to add them in the standard way.

The next event to check for is when the mouse isn't over the Btn_Play object. You can do this in TGF2 by using the Negate option. This allows you to create your event condition and then negate it, which means it will be the opposite of what the condition says.

- 8. Click on New Condition on event line 9.
- 9. When the dialog box appears, right-click on The mouse pointer and keyboard object and then choose The mouse. Then select Check for mouse pointer over an object.
- 10. When you are asked which object to check that the mouse pointer is over, choose the Btn_Play object.

You have your event, but it will be true when the mouse is over the object, which is what the previous event already does. You need to change this so that it will be true when the mouse isn't over this object.

- 11. Right-click on the condition text Mouse pointer is over.. on event line 9, and from the pop-up menu select Negate.

A red cross now appears at the start of the condition to show that the condition has been negated.

Next, you need to add an action that will change the animation frame to 0.

- 12. Move to the right of this event until you are directly under the Btn_Play object.
- 13. Right-click on the blank action box and then select Animation | Change | Animation Frame.
- 14. The Expression Evaluator appears and advises you that the first frame is equal to 0. This default is exactly what you need, so click OK.

The events and actions for the Btn_play object are shown in Figure 11.29.

7	Buttons															
8	• Mouse pointer is over													✓		Force animation frame to 1
9	• X Mouse pointer is over													✓		Force animation frame to 0
10	• New condition															

FIGURE 11.29 The mouse over event and the negated option.

If you run the program now, when you move the mouse over the Play button it will become highlighted, and when you move away from it, it will become dehighlighted. You now need to follow the same procedure for the Quit button.

- 15. Click on New Condition on event line 10.
- 16. When the dialog box appears, right-click on The mouse pointer and keyboard object and choose The mouse. Then select Check for mouse pointer over an object.

17. When you are asked which object to check that the mouse pointer is over, choose the Btn_Quit object.
18. Move to the right of this event until you are directly under the Btn_Quit object.
19. Right-click on the blank action box and select Animation | Change | Animation Frame.
20. In the Expression Evaluator, change this to “1” and then click OK.

Now for the second event to negate the quit button.

21. Click on New Condition on event line 11.
22. When the dialog box appears, right-click on The mouse pointer and keyboard object, choose The mouse, and select Check for mouse pointer over an object.
23. When you are asked which object to check that the mouse pointer is over, choose the Btn_Quit object.
24. Right-click on Mouse pointer is over.. on event line 11 and from the pop-up menu select Negate.
25. Move to the right of this event until you are directly under the Btn_Quit object.
26. Right-click on the blank action box and select Animation | Change | Animation Frame.
27. The Expression Evaluator appears and the default setting that appears is already set to “0” so click OK.

Run the program, and now both buttons should work perfectly. If not, review what you have done and make sure the actions are under the correct object.

Add Comment

Add a comment for the final two events for this frame to identify when a user has clicked on either of the two buttons.

1. Right-click on event line number 12 and select Insert | A comment.
2. In the Edit box type “User Clicks” and then OK.

User Clicks

You need to program what happens when the user clicks on the Play or Quit button. For the conditions, you will be using the mouse pointer and keyboard object, and for the actions, you will play a sound and either move to the game frame or quit the program. The handling of the frame navigation is done by the Storyboard Controls object.

Create the first event, which will be the player clicking on the Play button.

1. Click on New Condition on event line 13. Select The mouse pointer and keyboard object from the dialog box. Then select The Mouse from the pop-up menu and then User clicks on an object. A dialog box will appear as shown in Figure 11.30.



FIGURE 11.30 The condition settings available for clicking on an object.

2. Leave the defaults as left mouse button and single click and then click OK.
3. Another dialog box asks which object you want to check for the mouse being clicked on. In this example it is the Btn_Play object, so select that and click OK to save the condition.

You now need to add the two actions for this event. The first is to play a sound, and the second is to change to a different frame.

4. Move to the right of event line 13 until you are directly under the Sound object.
5. Right-click on the blank action box and select Samples | Play sample. In the Play Sample dialog box, click the Browse button opposite From a file.
6. Navigate to your DVD and select the TGFFILES\Alien Wars\Sounds folder. Open the Laser3.wav file. The sound has been added to the TGF2 program and you should be back at the Event Editor.



Next, you need to add the action that will take the player to the game frame.

7. Still on event line 13, move across to the Storyboard Controls action box, right-click on it, and select Next Frame.

You have completed the event for clicking the Play button, but you now need to do something similar for the quit button.

8. Click on New Condition on event line 14. Select The mouse pointer and keyboard object from the dialog box. Select The Mouse from the pop-up menu and then User clicks on an object. In the dialog box that appears, click OK to keep the default settings.
9. In the next dialog box, which asks for the object that you want to check for the mouse being clicked on, select the Btn_Quit object and click OK to save the condition.

The two actions that are required are to play a sound and to quit the application.

10. Move to the right of event line 14 until you are directly under the Sound object. Right-click on the blank action box and select Samples | Play sample. The sound Laser3 is already listed in the Play Sample dialog. Left-click on it and click OK.
11. Move to the Storyboard Controls object on event line 14, right-click on the action box, and select End the application.

You can see all of the events and conditions in Figure 11.31.

All the events All the objects		Alien Wars Version 1.0									
1											
2	• Start of Frame		✓							✓	
3	Music										
4	• The Last Day is not playing		✓								
5	Transparency										
6	• SemiTrans("■") > 0									✓	
7	Buttons										
8	• Mouse pointer is over									✓	
9	• X Mouse pointer is over									✓	
10	• Mouse pointer is over										✓
11	• X Mouse pointer is over										✓
12	User Clicks										
13	• User clicks with left button on		✓	✓							
14	• User clicks with left button on		✓	✓							
15	• New condition										

FIGURE 11.31 All of the conditions for the menu frame.

You have completed all of the events required for the menu frame. Run the program and make sure it works correctly. In the next part in this chapter you will begin to make the game.

Programming the Game Frame

The game frame is the most complex of the three frames. Even with the features included within the game, the actual amount of programming required is quite small and requires fewer than 30 events. This is one of the great benefits of TGF2, and it won't be long before you are making your own games. Best of all, there are no long amounts of text to type out like in traditional games development.

Game Frame Components

This is the busiest screen in the three frames, and this means there is a lot more to program:

- Initializing the game, and setting the score value
- Checking the music and seeing if it is playing
- Stopping the ship from leaving the screen
- Making sure the player's score and level is always on the topmost layer of the screen
- Creating the enemy ships on screen
- Testing for the player pressing the space bar and checking if there is already a bullet on screen
- Checking for the position of the bullets on screen and destroying them if they are off screen
- Checking for collisions
- Creating more enemies and adding a level when no enemies are left
- Getting the enemy to shoot

Start of Frame

The Start of Frame condition is used to configure objects and your game before it begins. For the game frame you will be using this event to set the correct animation of the player's spaceship so that it is pointing in the correct direction and to set up a global value to store the score.

1. Click on New Condition on event line 1.
2. Select the Storyboard object and then from the pop-up menu choose Start of Frame.

It's time to create your first action, which will be to set up a global value and change its starting value to 0. A global value is a number that can be accessed through the whole game and is very useful when you want to take information from one screen to another, for example, a score or lives number, which would need to be carried over various frames.

Many slots can be used to store information. By default, there are 26 global value slots, which are labeled A to Z. You can store information in any slot, so long as you remember which slot you are working with. You can create more slots if they are needed, but for this part of the game use slot "S" to store the score. The score can be stored several ways. You could also use the Score object, which is a much simpler process, but because the Score object uses images to display the score, these become too large to fit in the game. You will be saving the score in the global value and then displaying it in a text box so you can specify a font size and make it fit nicely in the game. It is always sensible to set your global values to 0 at the start of the game to ensure that no other number is currently stored in that value.

Even though the score is displayed in a text box, you still need to update the player's score on the Player object. This will be needed to place the score into the highscore table, as that object reads the player's score action. This may seem a little confusing, but as you work through it, the process will become much clearer.



If you want to see how the standard Score object works in TGF2, try the ChocoBreak tutorial that is displayed automatically when TGF2 starts.

3. Right-click on the action box directly below the Special Conditions object on the start of frame event line. From the pop-up menu, choose Change a Global Value | Set.
4. In the first drop-down box in the Global value expression editor box that appears are the slots to select from, as shown in Figure 11.32.

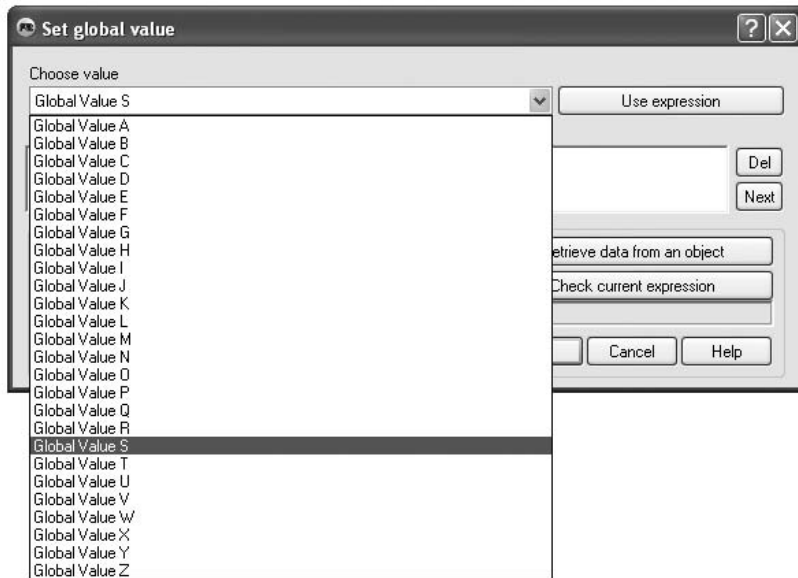


FIGURE 11.32 The available global values.

5. Select Global Value S. The number in the box below needs to be 0, which it is, so click on OK to save.

Now you need to set the animation frame to the correct frame, which in this case is Frame 1. You can view the animations for the spaceship by double left-clicking on the object in the Frame Editor. You will learn more about the picture and animation editor later on in this book. The player's spaceship has a set of animations for moving left, a set for moving right, and, very importantly, a single animation

where it is pointing forward. This final animation is necessary because otherwise, when the player takes his fingers off the keys, the spaceship would be pointing left or right, which wouldn't look right for this type of game.

6. Move to the right of the first event line until you are directly under the Player object and then right-click on the action box and select Animation | Change | Animation Frame. When the Expression Evaluator appears, leave it at 0 (as this represents the first frame) and click OK.

If you run the game now, you can move the spaceship left and right, and the correct movements will play. This is because the program already has the correct animations placed within it and it knows when to play the left and right animations depending on the direction the ship is moving.

Music Not Playing

As you did for the main frame, you need to check if the music is playing, and, if not, you need to tell it to play The Last Day.

1. Click on New Condition on event line 2.
2. Select the Sound object and then Samples | Is a specific sample not playing?
3. In the dialog box that appears, select the The Last Day sample and click OK.

Now you need to add the action, which will be to play the sample.

4. Move across to the right until you are under the Sound object and then right-click the action box. Select Samples | Play Sample. In the dialog box that appears, select The Last Day sound item and click OK. You can see the current events in Figure 11.33.

All the events All the objects														
1	• Start of Frame	<input checked="" type="checkbox"/>												
2	• The Last Day is not playing		<input checked="" type="checkbox"/>											
3	• New condition													

FIGURE 11.33 The two events created so far in our game frame.

Stopping the Ship from Leaving the Screen

The spaceship already has a movement applied to it. This is why when you play the game you can already move the ship left and right. If you keep pressing the left or right arrow keys you will notice that the spaceship leaves the screen. You need to prevent this from happening, as the ship is supposed to be contained within the game window.

In TGF2 you can check an object's location on the screen, if it is moving out of the frame, or even coming into the frame. You can tell TGF2 to stop the movement of the object, which will create the desired effect of preventing it from leaving the left or right side of the screen.

First, add a comment line to identify this part of the code:

1. Right-click on event line 3 and select Insert | A comment.
2. In the dialog box enter "Stop ship from leaving screen" and press OK.

Next you need to create an event that will test the position of the player's ship and test to see if it is leaving the screen.

3. Click on New Condition on event line 4.
4. In the New Condition dialog box, right-click on the Player object, and in the pop-up select Position | Test position of player.
5. The Test Position of Player dialog box makes it easy to check the location of an object on the screen. By selecting the arrows you can automatically check for the location. The large arrow in the center checks to see if the object is located in the frame, and the large arrow in the bottom left checks for objects outside the frame. The four arrows pointing outward check for any objects moving out of the frame from that side of the screen (e.g., left, right, top, or bottom). The four arrows pointing inward check if the object is outside of the frame but moving into the frame.
6. As the player's spaceship can only move left or right, you only need to test its location moving left or right out of the frame. Select the left and right arrows pointing outward as shown in Figure 11.34 and then click OK to save the event.

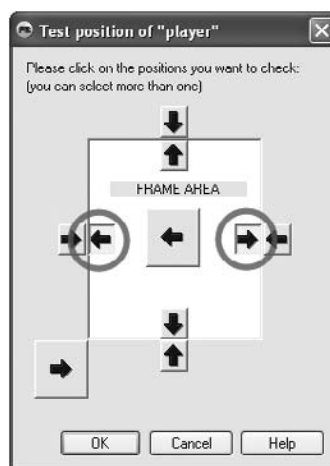


FIGURE 11.34 Testing the location of the player's ship on the left and right sides of the frame.

Now you need the action that changes the movement of the player's ship to stop. Once the player presses the opposite direction, the program will no longer run this event or action, and then ship will move again.

7. Move across from event line 4 until you are directly under the Player object, right-click, and select Movement | Stop. You can see the created comment and event in Figure 11.35.

The screenshot shows the event editor interface. At the top, there are two tabs: 'All the events' and 'All the objects'. Below the tabs is a toolbar with various icons for actions like 'New Event', 'New Action', 'Copy', 'Paste', 'Delete', 'Undo', 'Redo', 'Zoom In', 'Zoom Out', 'Reset', and 'Help'. The main area is a table with 4 rows and 15 columns. The first three rows are event lines, and the fourth row is an action line. The first three rows have a checkmark in the first column. The fourth row has a checkmark in the eighth column.

	All the events	All the objects													
1	• Start of Frame		✓										✓		
2	• The Last Day is not playing			✓											
3	Stop ship from leaving screen														
4	• [Ship icon] leaves the play area on the left or right												✓		

FIGURE 11.35 Current events and event lines.

Run the game now, and you will not be able to move the ship out of the game's frame.

Display Always on Top

The score and level panel on the top-right corner displays important information about how well the player is doing. Later on in the game when you start to create enemy robots, these robots will appear over this display. In some games you might be quite happy with this, but for this game the user should be able to see the score and level at all times. You can do this by telling TGF2 to always place these objects on the very top of the screen. When you add an object to the screen, it picks up an order number. This is how it will appear in front or behind another object. By using this action, you don't need to worry about when you added the object or the order number that it has obtained. You will know that it will always appear on top. The easiest event condition for this is the Always event, which will work for every loop of the program. It is important not to use too many Always events, especially to have too many actions, as this could slow down your game.

First, create the comment line:

1. Right-click on event 5 and then select Insert | A comment. Type the word "Display" into the text box and click OK.

Now to add the Always event:

2. Click on New Condition on event line 6.
3. Select the Special object and then click on Always.

You need to create three actions for this event line: display the back_info, the Score, and the Level objects to the top of the frame.

4. Move to the right of the Always event line, which is event line 6, until you are directly under the `back_info` object, right-click, and select `Order | Bring to front` and then do the same for the `Score` and `Level` objects. This creates the events and conditions shown in Figure 11.36. If your conditions do not match up or you do not have actions (ticks) under the correct objects, you need to recheck your code.

All the events All the objects		[Icons: Ship, Cannon, Explosion, etc.]																						
5	Display																							
6	• Always																					✓	✓	✓
7	• New condition																							

FIGURE 11.36 Events and conditions for event lines 5 and 6.



Events one to four are not shown in Figure 11.36, as we have scrolled up the events to show you the two you worked on with the relevant object images.

Creating Enemy Ships

The next few events involve placing enemy ships on the screen. When the scene was set for the game a single ship was placed off the top of the frame. You will duplicate this ship and place them in positions on the screen. Once all ships are destroyed, you will need to create them all again, so that the player is playing in a never-ending loop. In a game with many levels you would increase the difficulty of each of the waves. For this game, just recreate the ships each time.

To do this you need to create a group. This group of code will be run once at the very start of the game to create our initial wave and then will be enabled only when all ships have been destroyed. A group is a great way of creating code and only accessing it when you need it.

First, create a comment line to separate the code:

1. Right-click on event number 7 and select `Insert | A comment`. Type in “Create Enemy Ships” and click OK to save the event.

Now you need to add a group that will store all the events to create the enemy ships.

2. Right-click on event number 8 and select `Insert | A group of events`. When the `Group Events` dialog box appears, type “Level Placements” and then click OK.

Within the group, you will create eight events. Some of the actions could go in a single event, but as this is your first game, to make it easier to read you can separate them.

The first event in this group adds to the level number. As the frame starts, the level number is zero, and as this group is enabled to begin with, it will add 1 to make level 1. As soon as this group has run, it will disable itself, so until all enemy ships are destroyed it will not add any more level numbers. To store the level number, use a global value called L.

3. Under the group Level Placements is a blank event line New Condition. Click on New Condition in this group. This will be event line number 9.
4. Select the Special object and then Always.

Now that you have your event, set your global value. To do this use a simple calculation, putting the current value of L (the global level value) and adding 1.

5. Select the Special Conditions object for event line 9 and then pick Change a global value | Set. Click on the drop-down box in the Global Value dialog box and select Global Value L.
6. In the Enter Expression box you need to get the current value and add one to it. Click on the Retrieve data from an object button and then right-click on the Special object. Find L in the Retrieve a global value dialog box that appears (shown in Figure 11.37). Click OK to save the information into the Expression Evaluator.

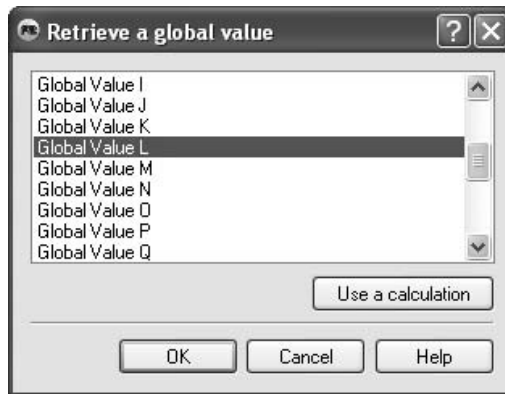


FIGURE 11.37 Retrieving the L value.

7. The text “Global Value L” will be displayed in the Expression Evaluator. At the end of it type “+1.” This will now look like Figure 11.38.
8. Click OK in the Expression Evaluator to save this information to the Event Editor.

The next event to make will create an enemy robot at a particular position on the screen. You need to do this five times. The process is as follows:

9. Click on New Condition on event line 10 and select the Special object.

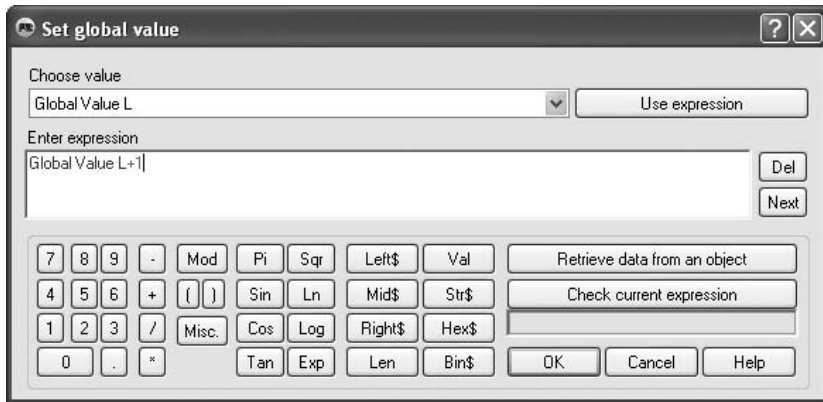


FIGURE 11.38 The Global Value L and the addition of one level in the Expression Evaluator.

You need to create an exact copy of the enemy that is off screen and then place it on a location within the game frame. To do this, use the Create object.

10. Move across from the event line 10 you just created until you are under the Create object. Right-click and select Create object. In the dialog box that appears, you can select an object to create. Find the Enemy object as shown in Figure 11.39 and then click OK.

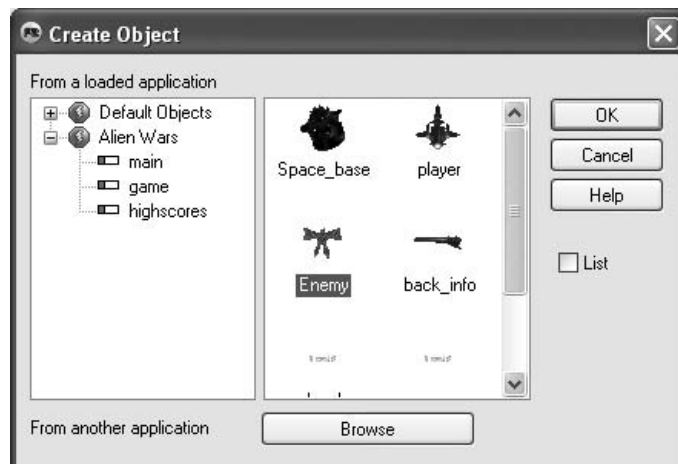


FIGURE 11.39 The Create Object selection box.

You will now be given the option of where you want to place this newly created object. You can either place it in a particular position or at a certain distance from another object. Place the object at a particular position.

11. Type in the X coordinate of 321 and the Y coordinate of 10, as shown in Figure 11.40.

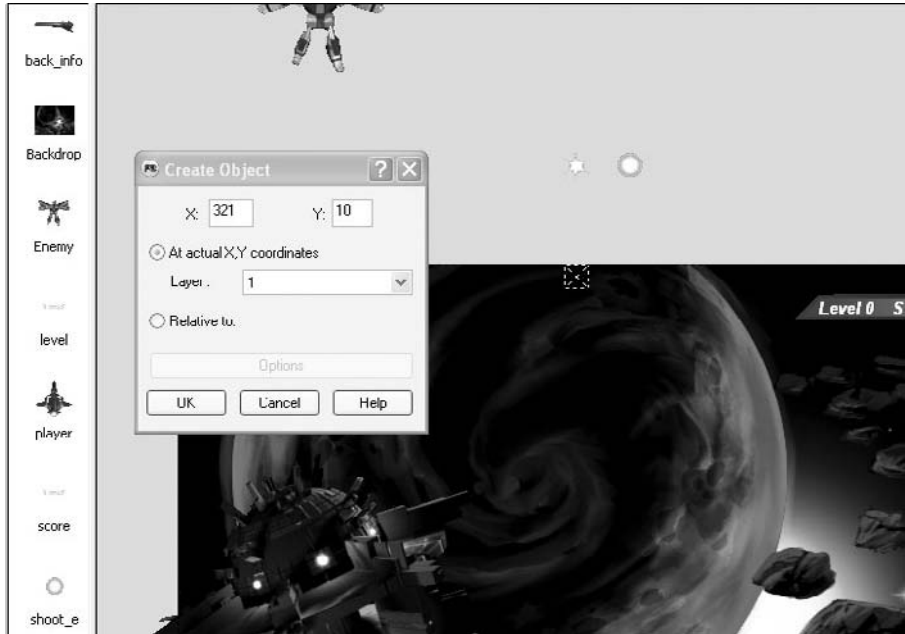


FIGURE 11.40 The coordinates for the Create Object action.

12. As you type in the coordinates, you will see a square with a cross in it, to show you where the object will appear. Click on OK to save the information to the Event Editor.

You need to create four additional events that have the “Only one action when event loops” and then create action for the Enemy object. See Table 11.4 for the positions of the four enemy robots.

Table 11.4 The Four Additional Enemy Robot Positions

EVENT LINE NUMBER	X POSITION	Y POSITION
11	193	88
12	449	88
13	65	10
14	577	10

If you run the game now, it should look like Figure 11.41. If not, review your code and the positions of each object you created.

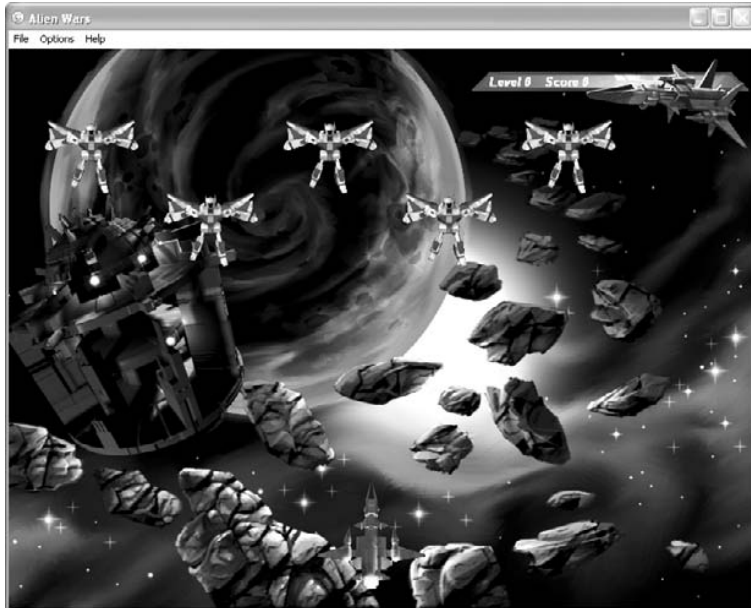


FIGURE 11.41 The enemy robots on screen.

The enemy robots are also moving, because they have already had a path movement applied to them and will keep moving along their path and back again until they are destroyed.

The next event in the Level Placements group is to write the current level to our text object Level in the top-right corner of the screen. Use the Always event again, so it will happen every time the program loops. This would be a problem if this bit of code was only in the Event Editor, but as it's in a group, it will only run when the group is enabled.

13. Click on New Condition on event line 15. Select the Special object and then choose Always.
14. You need to write the contents of the Global Value L to the object Level, so move across until you are under the Level object.
15. Right-click and select Change Alterable String. This option allows you to set the text of the object. In the Expression Evaluator that appears, you need to first put the level text in the box so that it is displayed and then add the current level onto the end of it. Type "Level" (with quotes) and then a plus sign. You now need to add the current value of the global value L, which will be 1. There is a slight problem: The Global Value is a number and the

text box is a string (text). You have to convert it to the correct format and convert the number to a piece of text so that it will be displayed. Click the Str\$ button, which will place the code Str\$(>Enter number here<) into the Expression Evaluator. The Enter number here is selected. Click Retrieve data from an object, select the Special object, and choose Retrieve a global value. From the dialog box that appears, select L and then click OK.

- The text in the Expression Evaluator should now look like Figure 11.42. If it does, click OK. If not, review the process of adding this action.

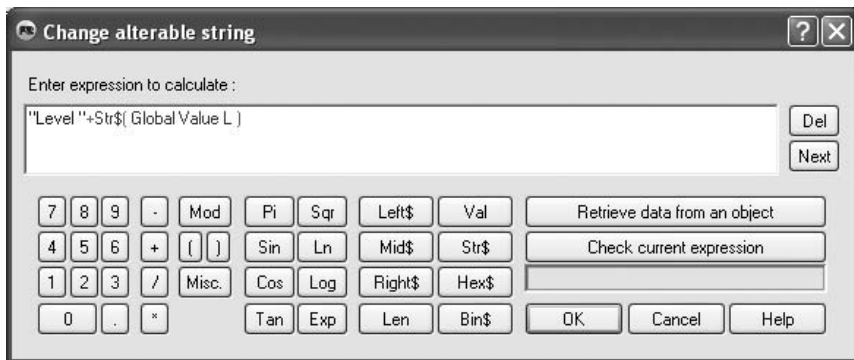


FIGURE 11.42 The code for getting the current level number into a text object.

The final event in this group is to disable the group. This means all of the actions in the group will only be run once. You will enable it again later on

- Click on New Condition on event line number 16.
- Select the Special object and then select Always.
- Move across to the right of the event to the Special Condition object, right-click, and select Group of events | Deactivate. A dialog box will appear, listing all of the groups available. You have only added one, so this is already selected. Click OK to save the action to the Event Editor.

You have now finished the code for the Level Placements group and you can see the events in Figure 11.43.

Player Shooting

It's time to create the event that will control the player's spaceship weapon. You could let the ship shoot many bullets, but to give the player a bit more of a challenge, create bullets one at a time. The player won't be able to fire again until the previous bullet has hit an enemy ship or moved off the frame. For this you need to create a two-condition event. The first condition checks the number of bullets on screen, and if this is less than one, the second condition checks to see if the user is

7	Create Enemy Ships																	
8	Level Placements																	
9	• Always	✓																
10	• Only one action when event loops							✓										
11	• Only one action when event loops							✓										
12	• Only one action when event loops							✓										
13	• Only one action when event loops							✓										
14	• Only one action when event loops							✓										
15	• Always																	✓
16	• Always	✓																
17	• New condition																	
18	• New condition																	

FIGURE 11.43 The events for the Level Placements group.

pressing the space bar. If both of these are true, the event will run the actions, which will play a sound and then shoot a bullet from the spaceship.

First, create the comment line.

1. Ensure that the Level Placements group is expanded. This means you can see all of the events within the group. If you closed the group, the event line numbers would be different, so for the rest of this project leave it expanded.
2. Right-click on event line number 18 and select Insert | A comment. Then type "Player Shoot" in the comment box. Click OK.

Now it's time to add the two-condition event.

3. Click on New Condition text on event line 19.
4. Select the Shoot_p object and choose Pick or Count | Compare the number of Shoot_p objects. This option allows you to compare the number of Shoot_p bullets that are currently created.
5. You want to allow this event to run as long as there are no other bullets on screen, so in the Expression Evaluator that appears, click on the drop-down box and select Lower. Type "1" in the Enter expression to compare with box and then click OK.
6. To add a second condition to the same event line, right-click on the condition you just added and select Insert. A dialog box will appear, allowing you to create another condition. Select The mouse pointer and keyboard object and then The keyboard | Upon pressing a key. Press the spacebar when prompted to do so.

You have your two conditions, so now you need to add the actions. The first action to create plays a sound when the bullet is fired.

7. Move to the right of the event until you are under the Sound object, right-click, and select Samples | Play sample. In the dialog box, click on the Browse button opposite the From a file option. Navigate to the TGFFILES\Alien wars\Sounds folder and open the file Laser4.wav.

Now it's time to add a second action on the same event line. To create a shooting bullet you can use a special action under most objects to fire an object from it.

8. Move across to the Player object, right-click, and select Shoot an object.
9. In the dialog box that appears, asking for the object that you want to shoot from the player's spaceship, select the Shoot_p object and click OK.
10. You now have a properties box asking for additional information about the speed and direction of the bullet. Change the speed dialog to 40.
11. You can also configure which direction the bullet will move from the ship. To specify the direction, select the Shoot in selected directions . . . radio button. This launches the Direction dialog box, which has the arrow pointing upward, as shown in Figure 11.44.

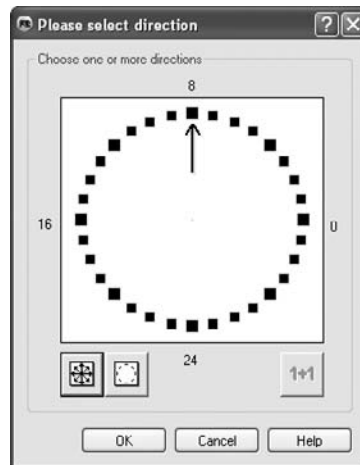


FIGURE 11.44 The Direction dialog box.

12. Click OK on the Direction dialog box.
13. The Shoot an Object dialog box will be configured as in Figure 11.45.

Position of Bullets

You only want the player to shoot one bullet at a time, and you have already created the condition to allow that, but when a bullet is off the screen, it will still be counted as a bullet. This will prevent the player from shooting any more bullets. You need to create an event to check the position of the bullet and, if it is off the top of the screen, will destroy it.



FIGURE 11.45 The Shoot an Object dialog box.

1. Click on New Condition on event line 20.
2. Select the Shoot_p object and from the pop-up menu choose Position | Compare Y position to a value. Y coordinates go up and down, so a higher value is lower on the frame and a lower value is toward the top of the frame. A negative value would be off the frame but above it, so in the Expression Evaluator that appears change the drop-down box to Lower and then enter “-32” and click OK.
3. Move to the right of this event until you are directly under the Shoot_p object and select Destroy.

This destroys the object if it is before -32 on the Y coordinate. This will happen quite quickly once the object has left the screen.

Collision Between the Enemy and the Bullet

If you were to run the game now you would notice the enemy robots moving, and you would be able to fire the ship’s weapons by pressing the spacebar. The bullets currently fly through the enemy and have no effect, so you need to check for a collision between the robot and the bullet and then run a number of actions. These actions include:

- Adding 20 to the global value 20
- Playing an exploding type sound
- Adding 20 to the internal TGF2 score object so that it can be transferred to the highscore table later
- Destroying the robot
- Destroying the bullet
- Setting the score

Begin with the collision event between the robot player and the bullet.

1. Click on New Condition on event line 21.
2. Select the Shoot_p object and then from the pop-up select Collisions | With another object. A dialog box will appear that requires you to select the second object in the collision with the bullet; in this case it’s the Enemy, so select that object and then click OK.

The condition has now been created, so you can create all of the required actions.

The first action is to add 20 to the global value, so for each robot destroyed the player will get 20 points.

3. Right-click on the Special Conditions object and select Change a global value | Add to. In the Expression Evaluator from the drop-down box select Global Value S, type in the number 20 and then OK.

Now you can add the exploding sound.

4. Move to the Sound object, right-click, and select Samples | Play sample. Click on the Browse button opposite From a file and then navigate to the TGGFILES\Alien wars\Sounds folder and open the file EXPLOD03.wav.

You need to add 20 to the player's score. This isn't the score that is displayed but the score that the system requires to put in the highscores table. This score is exactly the same as the score that is displayed on screen.

5. Still on event line 21, move across until you are under the Player 1 object. Right-click and select Score | Add to score. In the Expression Evaluator box type in "20" and then click OK.

You can now destroy both the enemy ship and the bullet.

6. Move to the right of the event until you are under the Enemy object, right-click, and select Destroy.
7. Move to the right until you are under the Shoot_p object, right-click, and select Destroy.

The last thing to do on this event line is set the currently saved score to the Score text object.

8. Move to the right and right-click on the Score object. Select Change alterable string. Type in

```
"Score "+
```

9. Click on the Str\$ button and then click on Retrieve data from an object. Choose the Special object, then Retrieve a global value, and from the dialog box select Global Value S. Click OK in the dialog box. The Expression Evaluator will now read:

```
"Score "+Str$( Global Value S )
```

10. Click OK to save this information to the Event Editor.

If you run the game now, you will be able to shoot and destroy the enemy robots, you will hear a sound when the bullets hit the robots, and finally the score will increase. You can see the events you have added for this section of code in Figure 11.46.

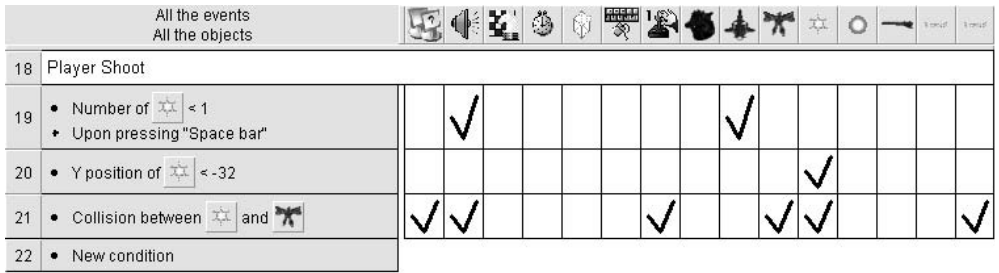


FIGURE 11.46 The Player Shoot code section.

Recreate the Enemy Robots

When no robots are left, you need to create them all again so that the game plays continuously. The main code for this has already been created, and the only action needed is to enable the Level Placements group, which handles placing all of the objects.

For the event you need to compare the number of robots on screen. For this program you need to see if it equals 1, so run the group. You might be wondering why you are comparing the robots against 1 rather than 0. This is because a single robot is placed offscreen that the player never gets to see, but TGF2 knows it is there. Therefore, when the comparison is done you know there will be one left, but all objects on the frame will have been destroyed.

Create a comment line first:

1. Right-click on event number 22 and select Insert | A comment. In the text box type "Regenerate enemy."

Compare the number of enemy robots on screen and see if it equals 1.

2. Click on New Condition on event line 23. Select the Enemy object and then Pick or count | Compare to the number of enemy objects. In the Expression Evaluator leave Equal, type in the number 1 and click OK.

Now you have your event, and when this is true, you want it to create a whole new group of enemies. This can be done by enabling the group code, which also increments the level number by one.

3. Move to the right of the event and right-click on the Special Conditions object select Group of events | Activate. Then select the only group available—(1) – Level Placements—and click OK.

Robot Firing

The next two events handle the firing of bullets from the enemy robots. The first plays the firing animation for the robot every two seconds. This time is set manually, so if you wanted to make the game easier you could increase the time, or you could decrease it to make it animate more often. The second event knows when the firing animation is playing and fires a bullet in the downward direction.

First, create the event that will pick at random one of the enemy robots every two seconds.

1. Click on New Condition on event line 24. Select the Timer object and then Every. The Timer dialog box shows hours, minutes, seconds, and so on. You want something to happen every two seconds, so use the slider so that the seconds number changes to 2 or type in "2" as shown in Figure 11.47.

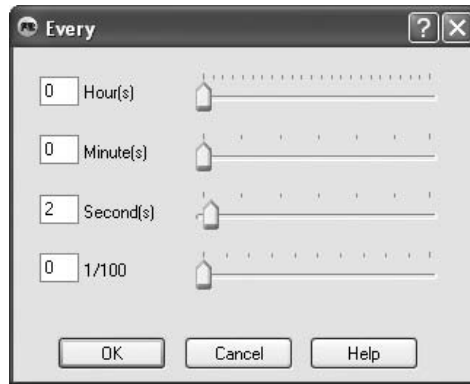


FIGURE 11.47 The timer dialog box.

You need to add another condition to the same event, which will pick one of the enemy robots at random.

2. Right-click on Every 02"-00 text and select Insert. Select the Enemy item and then from the pop-up choose Pick or count | Pick enemy at random.

This picks any of the robots on screen at random every two seconds. You now need to change the animation of the selected robot to a special animation called Shooting Animation.

3. Move to the right of event line 24 until you are below the Enemy object. Right-click the action box and select Animation | Change | Animation sequence. When the animation dialog box appears, select Shooting Animation and click OK.

The next event requires three conditions. These conditions check which animation is playing and when animation frame 15 is playing, and then the action fires a bullet down toward the player's spaceship. This is needed because of the special animation of the enemy robot throwing a bullet. In normal circumstances where you are using just a spaceship, you wouldn't need to create these conditions. You would put the shooting action under the timer event created previously.

4. Click on New Condition on event line 25. Select the Enemy object and then Animation | Which animation of enemy is playing. In the dialog box that appears, select Shooting Animation and then click OK.

The next condition on the same event compares the current animation frame of the object and is true once it reaches 15. Objects are animated by frames and run through each image to create the animation. In this case, animation 15 of the Enemy object is when it is about to throw a bullet.

5. Right-click on the condition you just created and select Insert. Then select the Special object and Compare two general values. In the Expression Evaluator box that appears, ensure that the first box is selected and then click Retrieve data from an object button. Choose the Enemy object and from the pop-up select Animation | Current Frame.
6. In the second box type "15" and then click OK.

You now need to add the final condition to this event, and for this you will just be checking that there are fewer than two bullets on screen (in other words, one bullet). The reading of all of the events is very quick, and it would be possible for two bullets to be fired from the enemy very close together, as the events in this case are read twice before the animation frame has moved onto animation frame 16. To prevent this from happening, place this condition, which will restrict the running of this event while there is one bullet on screen.

7. Right-click on the condition you just created and select Insert.
8. Select the Special object and then from the pop-up menu choose Compare two general values. In the dialog box, ensure that the top entry area is selected and then click on Retrieve data from an object and select the Shoot_e object, followed by Count | Number of Objects.
9. Change the drop-down box to read "Lower" and then select the second entry area and type in the number "2." Your Compare two general values box should now look like Figure 11.48.

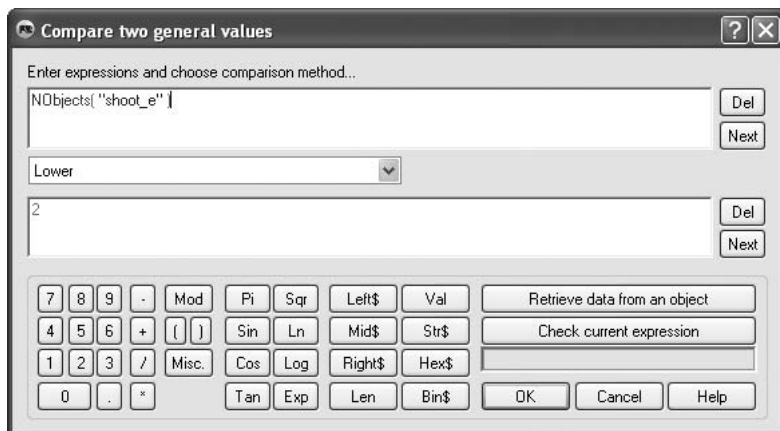


FIGURE 11.48 The Compare two general values entry box.

Now it's time to add the action to shoot the bullet.

10. Move to the right of this three-condition event until you are under the Enemy object. Right-click on the action box and select Shoot an object. From the dialog box, pick the Shoot_e object and click OK.
11. In the speed and direction dialog box type in the speed of 65 and click on the Shoot in selected directions . . . radio button.
12. When the direction dialog appears, replace the up-pointing arrow with a down-pointing one (there should only be a down-pointing arrow left on the dialog). Click OK. Then click OK again to save the information to the Event Editor.

You can see the event conditions in Figure 11.49.

All the events All the objects													
22	Regenerate enemy												
23	• Number of = 1	✓											
24	• Every 02"-00 • Pick one of									✓			
25	• animation Shooting Animation is playing • Image(" ") = 15 • NObjects(" ") < 2									✓			
26	• New condition												

FIGURE 11.49 The events created under the Regenerate enemy comment.

Robot Animations

Once the shooting animation has completed, you need to put the enemy robot back into its normal movement animation.

1. Click on New Condition on event line 26. Select the Enemy object and then Animation | Has an animation finished? From the dialog box select the Shooting Animation entry and click OK.
2. Move to the right of this event until you are directly under the Enemy object, right-click, and select Animation | Change | Animation Sequence. Select Walking from the dialog box and click OK.

You can see the event in Figure 11.50.

All the events All the objects													
26	• animation Shooting Animation is over											✓	
27	• New condition												

FIGURE 11.50 Checking that the animation has completed.

Collision Between Bullet and Spaceship

Our final event in this program is to finish the game when the player's ship has been hit with an enemy bullet. In some games you might create health or additional lives, but for this game only allow the player a single life. When the player is hit, the game automatically moves to the final highscores frame.

1. Click on New Condition on event line 27. Select the Shoot_e object and then from the pop-up select Collisions | Another object. Select the Player object from the dialog and click OK. Move to the right until you are directly under the Storyboard Controls object, right-click, and select Next Frame.

You should have created the final event as shown in Figure 11.51.

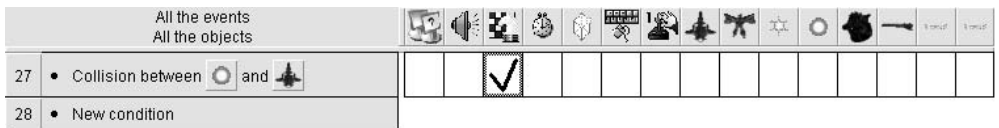


FIGURE 11.51 Collision between enemy bullet and players ship.

You can see all of the events for the game frame in Figure 11.52.

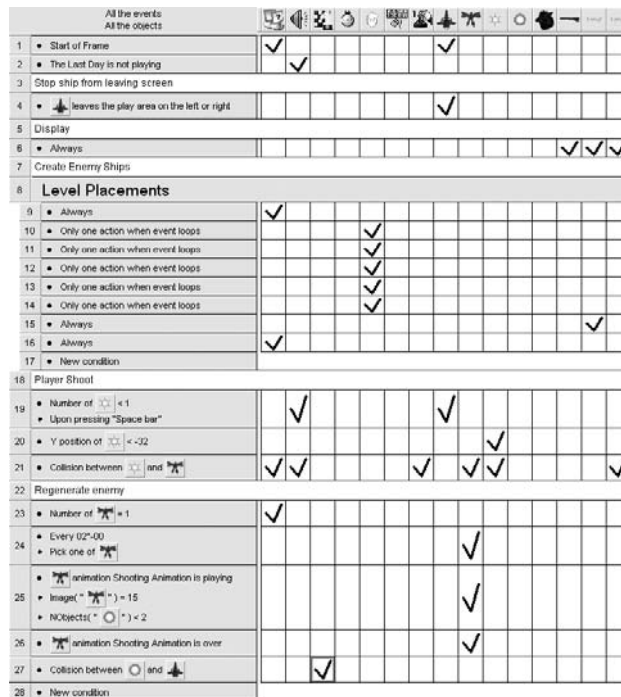


FIGURE 11.52 The completed code for the game frame.

Programming the Highscores Frame

Before you begin work on this frame, ensure that you are looking at the correct frame:

1. Double left-click on the text “highscores” in the Workspace toolbar.
2. In the highscores frame, click on the Event Editor icon to see a blank Event Editor ready for programming.

Highscores Frame Components

The highscores frame only requires us to program two simple events. The first checks if any music is playing, and the second moves the player back to the main menu once he is ready.

Music Is Not Playing

You need to create a condition the same way you did in the first frame to check to see if The Last Day is not playing, and the corresponding action will be to start playing it. This ensures that the music is playing continuously on these frames.

1. Click on New Condition on event line 1.
2. Select the Sound object and from the pop-up menu choose Samples | Is a specific sample not playing.
3. When the sample dialog box appears, select The Last Day and click OK.
4. Move to the right of event line 1 until you are directly under the Sound object, right-click on the empty action box, and select Samples | Play Sample.
5. The Play Sample dialog box will appear, and the sound you want is already listed. Select The Last Day and click OK.

Going Back to the Start

When the player presses a key, you want to take him back to the start of the game. It is very important to allow the user to easily navigate around the screens and ensure that there is a simple way of getting back to the main menu.

1. Click on New Condition, which is in event line 2.
2. Select The mouse pointer and keyboard object and then The keyboard | Upon pressing a key.
3. When you are asked for a key, press the Escape key (shown as ESC on your keyboard).
4. The event has been added.
5. Move across from this event until you are directly under the Storyboard Controls icon, right-click on the action box and select Jump to Frame. This displays a Choose a Storyboard Frame as shown in Figure 11.53.

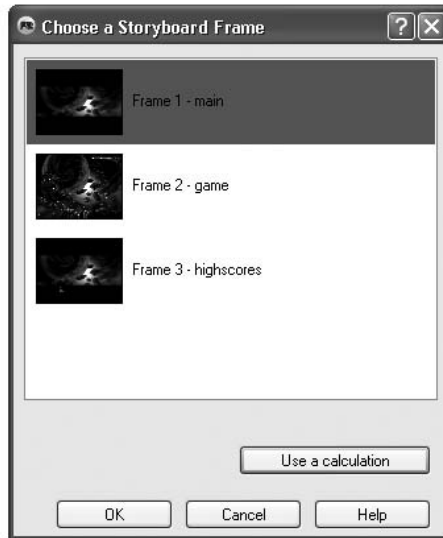


FIGURE 11.53 The storyboard selection dialog box.

6. Select Frame 1—main and click OK to tell the Event Editor to jump to this frame when the user presses any key.

Run the game and see if you can get a highscore.

CHAPTER SUMMARY

Congratulations. You have completed your first game in TGF2. We hope you have learned a lot about actions, conditions, and events and begun to understand how to use these in your games.

It will take a little time for you to remember where certain options are and where the relevant actions are in the objects you are using, but over time this will become second nature and you will find that TGF2 is very powerful.

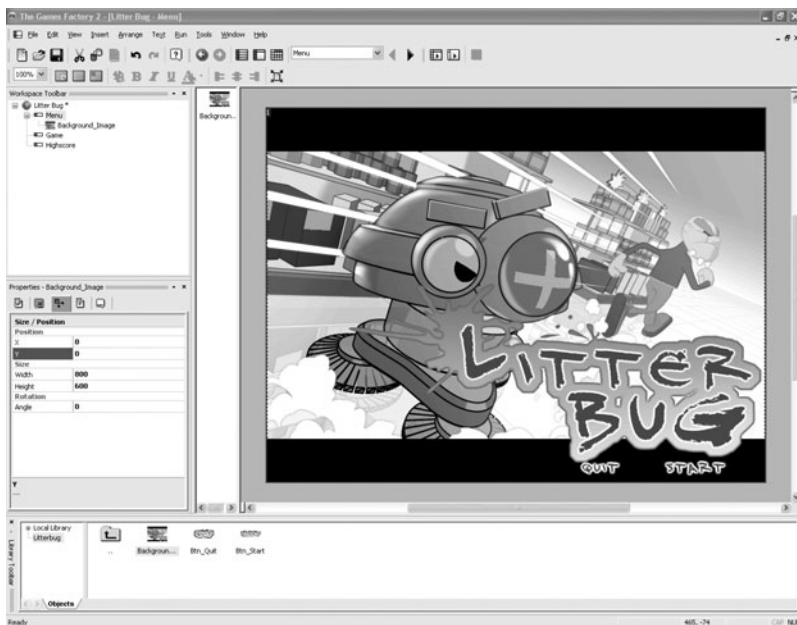
Let's make another game in TGF2 so you can get some more experience with the Event Editor and learn some more skills.

This page intentionally left blank

LITTER BUG

In This Chapter

- Introduction
- Library
- Initial Setup
- Event Programming



In this chapter you will be creating your second TGF2 game, called Litter Bug. Litter Bug is a single-screen game where you control a robot and move him around the game world using the arrow keys. This is a different type of game from Alien Wars and provides insight into what types of games TGF2 can produce easily. You will learn a lot of different game techniques that you can use in your own games when you begin to make them.

INTRODUCTION

You are about to create a 2D single-level game called Litter Bug. The story behind the game goes as follows:

You are Robot X2000, the ultimate cleaning device. You are employed at the local spacemart and your job is to ensure the shop floor is kept spotless. Unfortunately two people from a rival store have come to drop litter and ensure you get fired from your job. Keep the floor clean and ensure your boss is kept happy or you might just get fired!

The player controls the robot using the cursor keys (arrow keys) to move between the two floors of the game.

The game has the following components:

- A menu screen
- A game screen
- A highscores screen
- A robot controlled by events
- A number of shop floor graphics
- Counters to keep track of rubbish
- Two nonplayer characters that move and drop litter

You can see an example of the completed game in Figure 12.1.

LIBRARY

A library file has already been created, which contains all of the graphic resources you need to produce your game. You need to connect to the library so you can begin the process of dragging and dropping these items into place.

1. Right-click on the left-hand pane of the Library toolbar and select New from the pop-up menu.



If no Library toolbar is displayed at the bottom of the screen, select View | Toolbars | Library Window to enable it.



FIGURE 12.1 Litter Bug game.



2. Browse the DVD for the folder TGFFILES\Litter Bug\Lib, and then click OK.
3. Type in the name of the library entry as “Litterbug.”
4. Single left-click on Litterbug in the left-hand pane to reveal the litter_lib file and then double left-click on the litter_bug file to display all three library frames as shown in Figure 12.2.

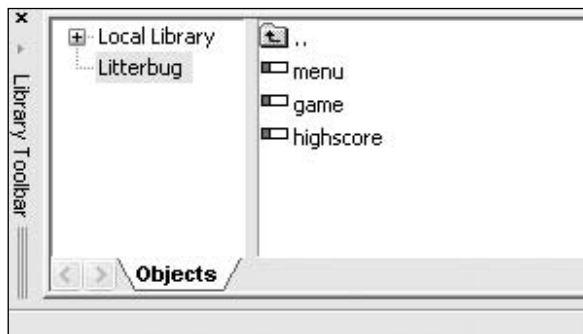


FIGURE 12.2 The Litter Bug library files.

INITIAL SETUP

Now that the library file is in place, you need to start building up the basic scenes of the program. The game has been split into three frames:

Menu. This is the first screen the players will see, and they will have two choices to select from: start the game, which will take them to the Game frame, or quit the game.

Game. This is where most of the work and programming will take place. You will need to code many aspects of the game in this screen including keeping the score, the enemy dropping the litter, and programming your own movement engine rather than using the built-in movement types.

Highscore. Very little coding is required here—only one event. After a set period of time, you will move from the Highscore frame to the Menu frame.

Creating a New File

Click on the New button or select File | New from the text menu. This creates a blank application file with a single frame already created within it. You now need to create the other two frames and rename them all to describe what they will do within the game.

1. Right-click on Application 1 in the in the Workspace toolbar and select Rename. then type in the word “Litter Bug” and press the Enter key to save the information.
2. Right-click on Frame 1 in the Workspace toolbar. Select Rename and type in “Menu.”
3. Right-click on Application 1 and select New Frame. Type in the name of the frame to be “Game.”
4. Right-click on Application 1 and select New Frame for a second time and then type in the name of that frame to be “Hiscore.”

Your Workspace toolbar will now look like Figure 12.3.

Game Window Size

By default, all games created by the TGF2 are set to a screen size of 600 × 800. You need to amend this to be 800 × 600. 800 × 600 is one of the most common screen sizes in use today and is a good window size for any games you might want to make.

1. Click on Litter Bug in the Workspace toolbar.
2. In the Properties window, click on the Window tab. This displays a number of settings. Set the one called Size 640 × 480.
3. Click on the size, and a pop-up window will appear. Select 800 × 600.
4. Once you have clicked away from this property, a dialog box will appear asking if you want to modify the size of the frames to the same size as the application window. Click Yes.
5. All frames are now set to 800 × 600.

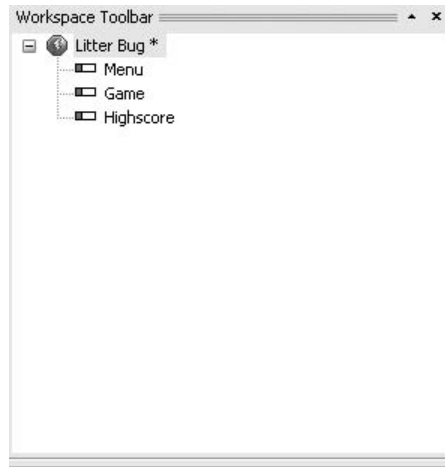


FIGURE 12.3 The Workspace toolbar.

Menu Frame Scene Setup

Now you need to set the scene for the Menu frame. This frame only consists of three items:

Background_Image. This is the main image of the menu screen that will cover the whole frame.

Btn_Quit. A graphical image with the word “Quit” on it, which will become an animated button to allow the user to quit the game.

Btn_Start. A graphical image with the word “Start” on it. This will provide the user with a way to move to the next frame.

Library File for Menu

Before you begin placing these objects, you must ensure that you are looking at the correct library folder that contains the menu items.

1. On the right-hand pane of the Library toolbar you should see three folders, double left click on the “menu” text.
2. This will reveal the three items to be used in the game, as shown in Figure 12.4.

Placing the Items

Now that the screen is the correct size and you have added all of the frames, it’s time to begin placing the graphics and objects in their correct locations.



FIGURE 12.4 The Menu frame graphic items in the Library toolbar.

First, ensure that you are looking at the correct frame:

1. Double left-click on Menu on the Workspace toolbar. This displays a blank frame in the Frame Editor.

Now you need to drag and drop items from the Library toolbar (Menu folder) to the frame and position them in the correct locations.

2. Drag the item Background_Image from the Library toolbar and drop it onto the blank frame.
3. To check its exact location, click on the graphic object on the frame to reveal its properties sheet. Click on the Size/Position tab in the properties sheet. Ensure that the Position entries for X and Y both are set to 0. If they are not, left-click on the entry to edit it.
4. Drag and drop the item Btn_Quit onto the frame and place it at X position 498 and Y position 561.



Remember that to check an object's position you need to single left-click on it once it is on the frame. Click on the Size/Position tab and then amend the X,Y coordinates.

5. Drag and drop the item Btn_Start onto the frame and place it at X position 633 and Y position of 561.

You have now completed setting the scene for the Menu frame, and it will look like Figure 12.5.

Game Frame Scene Setup

Now you need to set the scene for the Game frame. This frame consists of the most items:

Counter_NLvl. This counts the number of objects on the level at any one time so that it can change the Counter object to display how well or poorly the player is doing.

Rubbish_Cleaned. This is the amount of rubbish the player has cleaned. This information is placed into the String_Score text object.

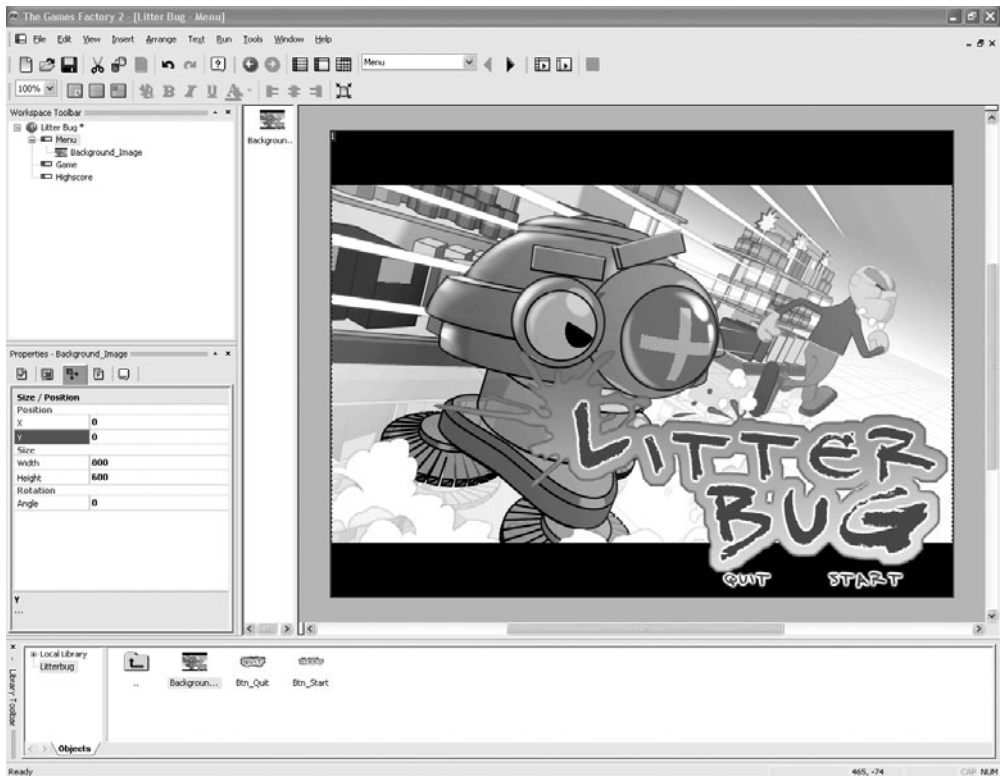


FIGURE 12.5 The Menu frame with all of the objects placed.

Layer object. In a game where objects can move in front of and behind other objects it is sensible to tell the objects where they need to appear so that they do not overlap incorrectly.

Clean_Fluid. This is attached to the back of the robot, and when it touches the dirt, the dirt is removed.

Dirt (n). There are two dirt objects: one for the ground floor of the shop and another for the second floor. These appear to be dropped by the Badguy1 and Badguy2 objects.

String_Score. This is a text item that keeps track of the score. The score increases by one every time a piece of dirt has been cleaned.

Control_Panel. This is the panel at the bottom of the screen that separates the game screen from the score and performance bar.

Counter. This counts the number of pieces of rubbish left on the floor. Once a pre-defined number is met (in this case 40), the game will end.

Guardrail_(a). A number of guardrails are positioned around the screen to give the impression of a ground floor and second floor.

Player. This is the player robot cleaning device. It is controlled with the arrow keys.

Coll_player. This is used to prevent the player character from moving where it shouldn't and to prevent collisions with other objects.

Badguy (n). Two badguys drop litter around the store. These two graphics have already had a movement applied to them. All you will need to do is get them to respond to colliding with the player's object.

Shelves (n). The many shelves in the game make it look more like a real shop. they also provide some cover for the badguys to drop litter where you cannot see it to make the game a little harder.

Background_Image. This is the background image to our game.

Movement_Map. This is a grid that will be used to prevent the player object from moving anywhere except the designated blank spaces on the map.

Shadow. This provides a shadow effect to other objects on the screen. It is just for graphical purposes.

Library File for the Game

Before you begin placing these objects you must ensure that you are looking at the correct library folder that contains the game items.

1. Your Library toolbar is currently in the menu folder, so double left-click on the yellow folder with the up-pointing arrow.
2. You can now see the three folders of objects that make up the game. Double left-click on the Game folder. You will see a large list of objects in the Library window, as shown in Figure 12.6.

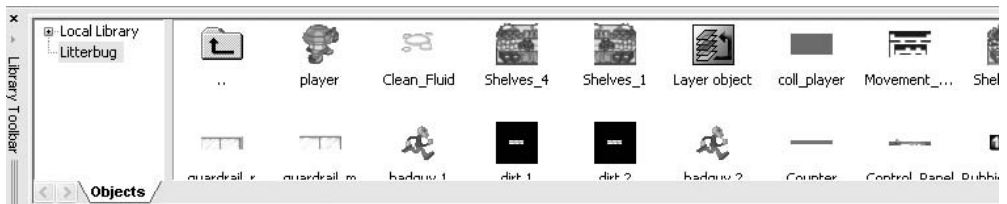


FIGURE 12.6 The objects for use in the game level.

Placing the Items

The Game frame contains the objects that need placing within our game, but this will always be the case when creating your own games. It is common practice in TGF2 to create a separate frame for each level of your game. This makes the designing and setting up of the frame a lot easier than trying to place everything in one place and then coding the Event Editor to move items into place when the player is on the correct level. This game only has a single level, so a single game frame is perfect for what you need to do.

First, be sure you are on the correct frame:

1. Double left-click on Game in the Workspace toolbar.
2. You should now see a blank frame.

Now start with some of the background objects:

3. Drag and drop the object Movement_Map from the Library window to the frame. Click anywhere on the colored part of the Movement_Map on the frame to display its properties.



The white areas on the Movement_Map are transparency. This means TGF2 ignores them, especially when you are trying to click on an object. To select an object with transparency, you need to click anywhere that has a pixel drawn.

4. Click on the Size/Position tab in the Properties window and change the X and Y coordinates to 0 and 0. Your screen will now look like Figure 12.7.

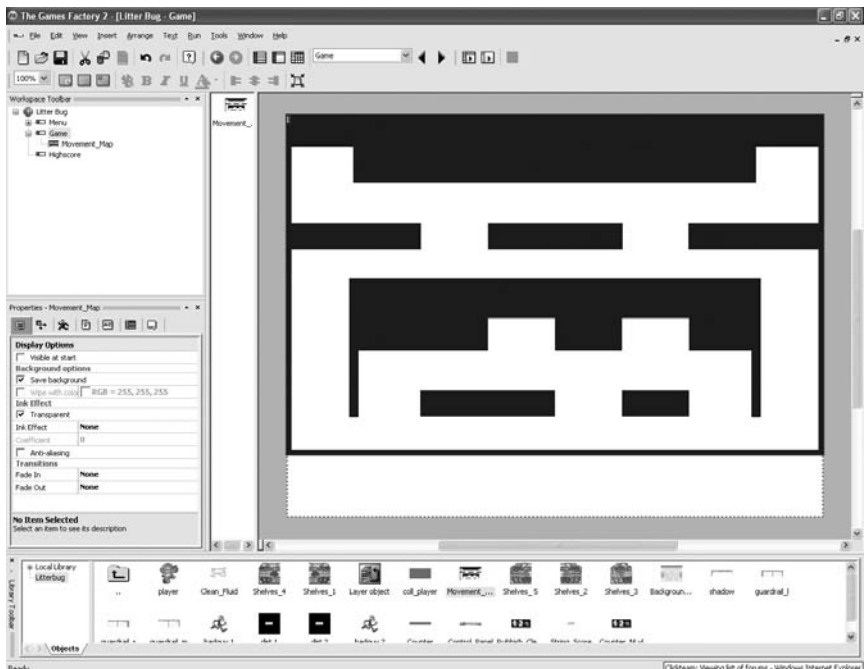


FIGURE 12.7 The movement map.

The movement map shows the places that our cleaner robot can move, and it can only move on those areas that are transparent. You will program the actual directions later on in this chapter, but using a hidden map can be very useful for restricting movement of objects by code.

5. Drag and drop the Background_Image object onto the frame and place it at X coordinate of 0 and Y coordinate of 0 (0,0).
6. Drag and drop the Shadow object and place it at (100,350).



To make the coordinates easier to read, some may be placed in brackets. The first number is the X coordinate, and the second is the Y coordinate.

Placing the Shelves

You have several similar objects to place onto the screen. These shelves provide the graphics that the badguys and the player can move in front of and behind. See Table 12.1 for the name of the object and its position on the frame. Place them using the same process you used for the other objects.



Some of the same objects will be placed more than once.

Table 12.1 Shelf Objects and Locations

OBJECT NAME	X COORDINATE	Y COORDINATE
Shelves_1	300	406
Shelves_1	199	56
Shelves_1	500	56
Shelves_1	0	155
Shelves_1	400	157
Shelves_2	201	306
Shelves_3	400	306
Shelves_3	600	306
Shelves_3	500	406
Shelves_3	300	56
Shelves_3	601	56
Shelves_3	100	156
Shelves_3	599	156
Shelves_4	200	406
Shelves_4	100	56
Shelves_4	401	56
Shelves_4	301	157
Shelves_4	700	156
Shelves_5	101	306

Your frame will now look like Figure 12.8.



FIGURE 12.8 The current layout of the Game frame.

Placing the Players

Now you shall place the player character, the two badguys who will drop the rubbish, and a collision box to be used for checking the position of the player on the collision map that you placed earlier. You can see all the positions in Table 12.2.

Table 12.2 Player and Badguy Positions

OBJECT NAME	X COORDINATE	Y COORDINATE
Coll_Player	426	428
Player	447	438
Badguy 1	221	461
Badguy 2	253	210

Placing the Off-Screen Objects

Some objects will be used in the game, some will appear on screen when required (for example, the dirt), and others will stay off screen and be used in the game to count the current score or be used in the Event Editor to program certain events. You can see all of these objects' positions in Table 12.3.

Table 12.3 Position of Off-Screen Objects

OBJECT NAME	X COORDINATE	Y COORDINATE
Dirt 1	932	179
Dirt 2	932	309
Clean_Fluid	-49	154
Layer Object	324	-76
Rubbish_Cleaned	239	-51
Counter_NLvl	213	-51

Final Scene Objects

You now need to place the final objects on the scene. These items include the scoreboard and the rails that go around the second floor. See Table 12.4 for the object names and their positions.

Table 12.4 Positions of Final Game Frame Objects

OBJECT NAME	X COORDINATE	Y COORDINATE
Counter	200	543
Control_Panel	0	599
String_score	610	551
Guardrail_l	93	266
Guardrail_m	200	266
Guardrail_m	300	266
Guardrail_m	400	266
Guardrail_m	500	266
Guardrail_r	700	266

You have completed placing all of the objects for the game screen, as shown in Figure 12.9.

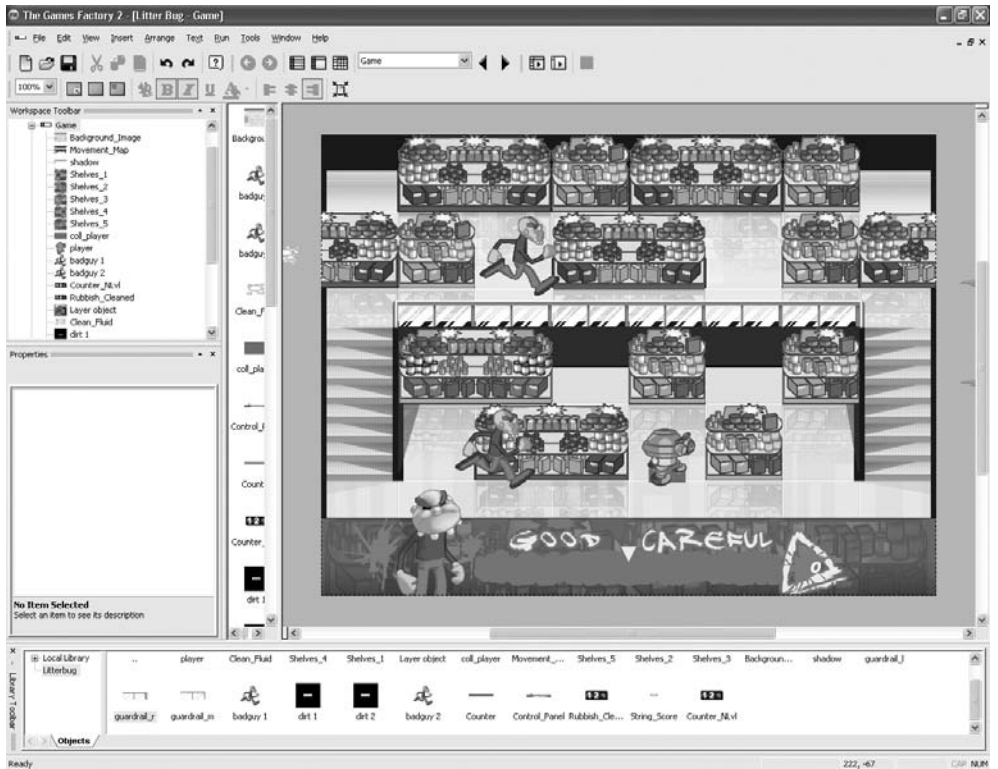


FIGURE 12.9 The completed positions of all of the Game frame objects.

Highscore Frame Scene Setup

Now you need to set the scene for the Highscore frame. This frame consists of only two items:

Background_Image. This is a background image that is used to decorate the frame.

Hi-score. This is an object called highscore, which saves all of the top scores into a table that has already been configured for font size, type, and color.

Library File for the Game

Before you begin placing these objects, you must ensure that you are looking at the correct library folder that contains these two items.

1. Your Library toolbar is currently in the game folder, so double left-click on the yellow folder with the up-pointing arrow.
2. You can now see the three folders of objects that make up the game. Double left-click on the Highscore folder. This displays a large list of objects in the library window as shown in Figure 12.10.



FIGURE 12.10 The objects for use in the Highscore level.

Placing the Items

The Highscore frame contains only two objects that need placing within our frame. Make sure you are on the correct frame:

1. Double left-click on Highscore in the Workspace toolbar.
2. You should now see a blank frame.

Drag both items and place them at their correct coordinates.

3. Drag and drop Background_Image onto the frame and place it at the coordinates of (0,0).
4. Drag and drop Hi-score onto the frame and place it at the coordinates of (503,639).

You have completed placing all of the objects required for the Highscore frame and you can see the results in Figure 12.11.

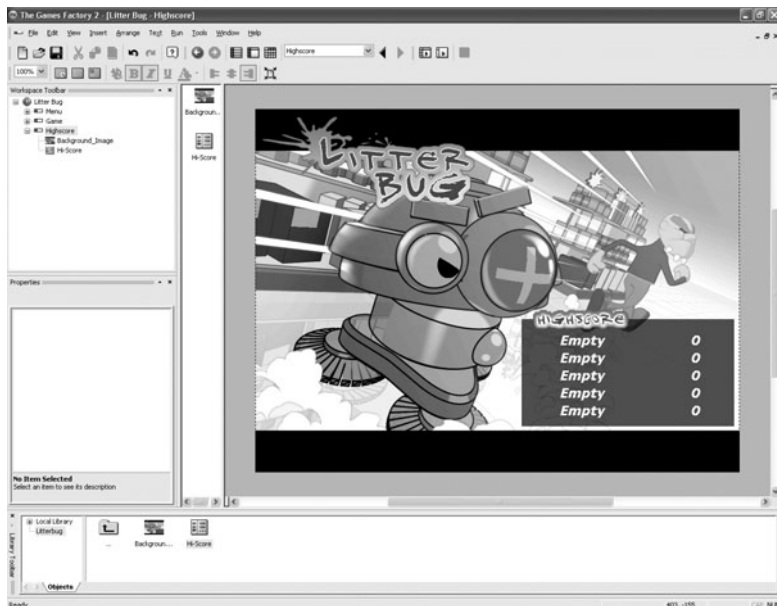


FIGURE 12.11 The Highscore frame with all objects in place.

EVENT PROGRAMMING

The scene is set for the three frames, and you are now ready to begin event programming to make the program functional. Start off with the Menu and then move onto the Game frame, which will take the longest, and, finally, finish with the High-score frame. In your own games you can program your frames in whatever order you decide and can leave the harder frames until last or complete them first.

Menu Programming

For the Menu programming frame you need to do the following:

- At the very start of the frame play and loop some music.
- Check for when the mouse is over either of the two buttons.
- Check for when the mouse isn't over either of the two buttons.
- Check for when the user clicks on either of the two buttons

Game Name and Version Comment

Start off the programming by adding a simple comment line to detail the program name and its current version number.

First, ensure you are on the correct frame.

1. Double left-click on Menu in the Workspace toolbar. You will now see the Litter Bug menu screen.
2. Click on the Event Editor button, and you will be presented with a blank Event Editor ready for programming.

Now, add our comment.

3. Right-click on event line 1 and then select Insert | A Comment. The Edit Text dialog box will appear. Type the text "Litter Bug" and then press the Enter key to move to the next line on the editor. Then type in the text "Version 1.0." Click on the Centered alignment button and then click OK.

You have now added your comment line as shown in Figure 12.12.



FIGURE 12.12 The product name and version comment.

Start of Frame

When the frame begins, you will play a sample song. As the song is short, loop it continuously so that it keeps playing. In your own games it would be better to use longer songs, as a short one looped can get irritating for the player after a short while.

First, add a comment line to separate the code.

1. Right-click on event line 2 and select Insert | A comment. In the dialog box type in “Start” and then click OK.



You now need to add a Start of Frame condition that will run as the frame is loaded and create a single action to play a sample file that is located on the DVD.

2. Click on New Condition and then select the Storyboard Controls object. Then from the pop-up menu choose Start of Frame.
3. Move across from this condition until you are under the Sound object. Right-click the blank action box and select Samples | Play and Loop Samples. When the Sound dialog box appears, click on the Browse button opposite From a file. Browse to the DVD to the \Samples folder, select the file called Action Point.wav, and click Open. You will now get a Expression Evaluator dialog which asks you to enter a number for the number of times you want the tune to play for. Enter “0” as this will make the song play continuously, then click on “OK”.



Your current code and action will look like Figure 12.13.

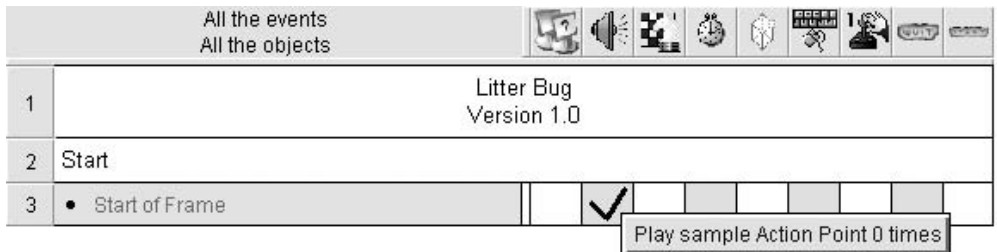


FIGURE 12.13 The start of frame code and action.

When the Mouse Is Over an Object

Now you need to check when the mouse is over either the Btn_Start or the Btn_Quit object. If it is, make the object visible. By default, both of these objects are visible in the game when the frame starts, so the code will not work until you have completed the next section of code.

First, create the comment line.

1. Right-click on event line 4 and select Insert | A Comment. Then type the text “Mouse Pointer Is Over” and OK.

Now you need to create an event that checks when the mouse is over the Btn_Quit object.

2. Click on New Condition on event line 5. Then select The mouse pointer and keyboard object and from the pop-up menu choose The mouse | Check for

mouse pointer over an object. A dialog box appears that is the object you want to check for the mouse being over. Select the Btn_Quit object, which is the object that shows the text “Quit.” Then click OK.

3. Move to the right of this event until you are directly under the Btn_Quit object. Right-click on the action box and select Visibility | Make object reappear.

You now need to create the same condition on a separate line for the Btn_Start object and make the same action for the Btn_Start object.

4. Click on New Condition on event line 6. Select The mouse pointer and keyboard object and then The mouse | Check for mouse pointer over an object. Then choose the Btn_Start object and click OK.
5. Now move to the right of this event line until you are directly under the Btn_Start object. Right-click the action box and select Visibility | Make object Reappear.

Mouse Pointer Is Not Over

You need to create an opposite effect of what you have just created. Otherwise, the two buttons will always appear, as there is nothing telling them to be invisible when the mouse isn’t over them.

First, create a comment line.

1. Right-click on event line 7 and select Insert | A comment. Then type “Mouse Pointer Is Not Over” and click OK.

Follow the same process you did before to create the two events that check for when the mouse is over the two objects. Once they are created, you can negate them. This means that the opposite of the event is true, which in this case will mean when the mouse is not over the objects. You also need to create the actions that will make the object invisible.

2. Click on New Condition on event line 8 and select The mouse pointer and keyboard object. Then select The mouse | Check for mouse pointer over an object, choose the Btn_Quit object, and click OK.
3. Move to the right of this event line until you are directly under the Btn_Quit object, right-click the action box, and select Visibility | Make object Invisible.

Now you need to negate it.

4. Right-click on Condition in event line 8 and from the pop-up menu shown in Figure 12.14, select Negate. This puts a red cross in front of the condition, which tells you it is negated.
5. Click on New Condition on event line 9 and select The mouse pointer and keyboard object. Then select The mouse | Check for mouse pointer over an object, choose the Btn_Start object, and click OK.

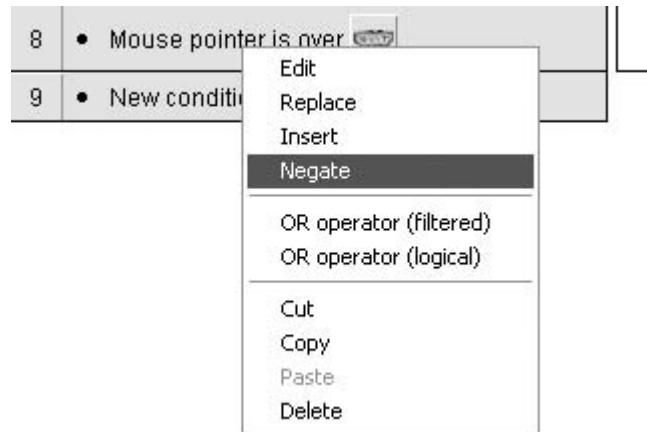


FIGURE 12.14 The Condition pop-up menu.

6. Move to the right of this event line until you are directly under the Btn_Start object, right-click the action box, and select Visibility | Make object Invisible.
7. Right-click on Condition in event line 9 and from the pop-up menu select Negate.

Your conditions should now look like Figure 12.15.

All the events All the objects											
1	Litter Bug Version 1.0										
2	Start										
3	• Start of Frame		<input checked="" type="checkbox"/>								
4	Mouse Pointer is Over										
5	• Mouse pointer is over									<input checked="" type="checkbox"/>	
6	• Mouse pointer is over										<input checked="" type="checkbox"/>
7	Mouse Pointer is Not Over										
8	• <input checked="" type="checkbox"/> Mouse pointer is over									<input checked="" type="checkbox"/>	
9	• <input checked="" type="checkbox"/> Mouse pointer is over										<input checked="" type="checkbox"/>
10	• New condition										

FIGURE 12.15 The conditions created so far.

If you run the code, you will notice that the buttons are now hidden and only appear when you move the mouse over either of the two buttons.

User Clicks

You now need to create two events to check for when the user presses the left mouse button over either of the two buttons. Clicking on the Start button will take the user to the Game frame, and clicking on the Quit button will exit the program.

First, create a comment line.

1. Right-click on event line 10 and select Insert | A Comment. In the comment box type “User Clicks” and click OK.

Now you need to create the two events.

2. Click New Condition on event line 11 and select The mouse pointer and keyboard object. Then choose The mouse | User clicks on an object. When the dialog box appears, keep the defaults and click OK. You now need to select an object, so choose Btn_Quit and click OK.
3. Move to the right of this event until you are under the Storyboard Controls object, right-click the blank action box, and choose End the application. This will quit the program when the user clicks on this button.
4. Click on New Condition on event line 12, select The mouse pointer and keyboard object, and then pick The mouse | User clicks on an object. Click OK to keep the defaults and then select the Btn_Start object. Click OK.
5. Move to the right of this event until you are under the Storyboard Controls object, right-click and choose Next Frame.

If you run the program by clicking the Run Application button, you can test both the frame movement and the quitting of the application buttons.

You have completed the code for this frame. The full code can be seen in Figure 12.16.

All the events All the objects		Litter Bug Version 1.0									
1											
2	Start										
3	• Start of Frame									✓	
4	Mouse Pointer is Over										
5	• Mouse pointer is over										✓
6	• Mouse pointer is over										✓
7	Mouse Pointer is Not Over										
8	• ✕ Mouse pointer is over										✓
9	• ✕ Mouse pointer is over										✓
10	User Clicks										
11	• User clicks with left button on			✓							
12	• User clicks with left button on			✓							
13	• New condition										

FIGURE 12.16 List of all the events in the first frame.

Game Frame Programming

The Game frame is where most of the event programming is done, and it can be split into the following tasks that you need to complete:

- At the start of the frame, configure the counter that displays the amount of litter on the floor.
- Configure the guardrails that will always appear in front of any objects.
- Configure the order of any objects to the front or back of other objects they come into contact with or move past.
- Program the movement of the player character.
- Destroy the cleaning fluid once it gets to a certain animation frame, to prevent too much from appearing on the screen.
- Check for collisions between the cleaning fluid and dirt and then destroy the dirt.
- Create a code group for the badguys so you can control how much dirt they drop and how they react when colliding with the player.
- Sort out the scores and display them on the screen.

Before beginning the programming, you need to ensure that you are on the correct Event Editor screen.

1. Double left-click on Game in the Workspace toolbar.
2. Click on the Event Editor button and you should see a blank Event Editor.

Start of Frame

Begin with a comment line and then create a Start of Frame event, which is used to control anything that needs to happen at the start of the frame once it's loaded. Then you can create an action to set the counter to 0.

1. Right-click on the number 1 on the Event Editor and select Insert | A comment. Type "Start" in the comment box and click OK.
2. Click on New Condition on event line 2. Select Storyboard Controls and from the pop-up menu choose Start of Frame.
3. Move across to the right of the event until you are under the Counter object, right-click the action box, and select Set Counter. The Expression Evaluator appears with a 0 already, which is what you want, so click OK.

Your current code should look like Figure 12.17.

Guard Rails

The guardrails are a graphic item whose only role is to improve the look of the game. You want them to always appear in front of the robot and the badguy, which will appear to walk behind them. This is just for graphic effect and is a good example of small things you can include in your game to make it more professional looking and give an overall higher-quality feel to your creations. You will also create some actions for the cleaning fluid and the location from which it appears.

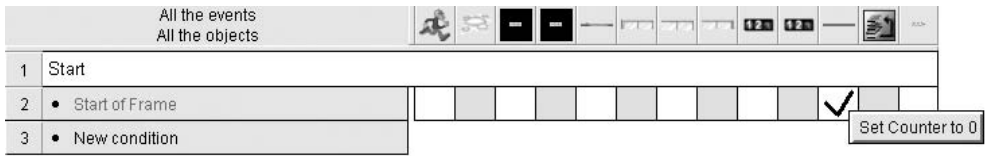


FIGURE 12.17 The start comment and condition.

First, create a comment for the event you are about to create.

1. Right-click on event number 3 and select Insert | A Comment. Type “Guard Rails & Player” and click OK.
2. Click on New Condition on event line 4 and then select Special | Always.

The first two actions are for the Player object and the cleaning fluid. To do this, use the shoot action, which shoots an object from the player. You would normally use this action for shooting bullets, but it can be used in effects as well. You will destroy the objects before they get too far later on in the program; otherwise, they would continue in the direction specified.

3. Move to the right of the Always event until you are directly under the Player object. Right-click and select Shoot an Object. A dialog box will appear. Use the scroll bar to locate the Clean_Fluid object, single left-click on the object to highlight it, and then click OK.
4. In the next dialog box that appears, type in “5” for the speed of the object and then click on the Shoot I selected directions radio button.
5. Ensure that all directions are selected, as shown in Figure 12.18 and then click OK. Click OK again to save the information to the action box.

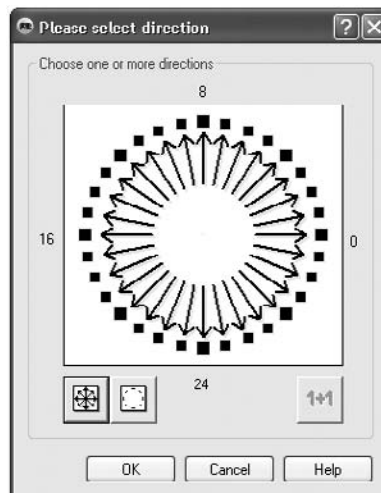


FIGURE 12.18 All directions selected.

Now you need to ensure that at all times (which is why you used the Always event) the guardrails are set to the front. This prevents any objects from being placed over them.

6. Move across from the Always condition until you are under the guardrail_l object, right-click the action box, and select Order | Bring to Front.
7. Move under the guardrail_m object, right-click the action box, and select Order | Bring to Front.
8. Finally, go under the guardrail_r object and in the blank action box select Order | Bring to Front.

The four events should look like Figure 12.19.



FIGURE 12.19 The events that have been created.

Order

Many of the objects in this game move in front of and behind other objects. This can be handled two ways. You can use the Layer object or use the Layer toolbar. The Layer object is very useful when you want to change objects based on their positions on the frame, which is why you will be using it for this game. In TGF2 there is a condition that checks the Y positions of selected objects. Once the event is true, you can change the layer position, which changes if the object is in front of or behind other objects.

You need to add a comment line first.

1. Right-click on event line 5 and select Insert | A comment. In the text box that appears type “Order” and click OK.

Now to add the six events. These will be added first, as all the actions are the same. The event tests if a particular object is overlapping another. These are:

- Player is overlapping the Shadow object.
- Badguy 1 is overlapping coll_player.
- Badguy 2 is overlapping coll_player.
- Coll_player is overlapping Group.Neutral.
- Badguy 1 is overlapping Group.Neutral.
- Badguy 2 is overlapping Group.Neutral.

Group.Neutral is a special feature in TGF2 that allows you to select a number of objects into a single group and then create events and actions for that group. In this

case the group is called Neutral and is identified by an apple in TGF2. The objects in this group are the shelves around the shop.

1. Click on New Condition on event line 6 and select the Player object. From the pop-up menu choose Collisions | Overlapping another object. In the next dialog box that appears, select the Shadow object and click OK.

The event is now created, so now you can create the other five events before you create the actions.

2. Click on New Condition on event line 7, select the Badguy 1 object, and then select Collisions | Overlapping another object. Select the Coll_player object and click OK.

Follow this same process for the rest of the items as shown in Table 12.5.

Table 12.5 Events 7 to 11

OBJECT	OVERLAPPING	OBJECT
Badguy 2	Is overlapping	Coll_Player
Coll_Player	Is overlapping	Group.Neutral
Badguy 1	Is overlapping	Group.Neutral
Badguy 2	Is overlapping	Group.Neutral

Once you have created all of these events, your code should look like Figure 12.20.













5	Order
6	•  is overlapping 
7	•  is overlapping 
8	•  is overlapping 
9	•  is overlapping 
10	•  is overlapping 
11	•  is overlapping 
12	• New condition

FIGURE 12.20 Events 5 to 11.

Now it's time to add the action, and the action is the same for all six events.

3. On event line 6, move across until you are under the Layer object, right-click the blank action box, and select Sort | By Y (Decreasing).

Do this for the other five events. Each Layer box against each event will now contain the words "Decreasing Y Sort," as shown in Figure 12.21.













5	Order							
6	•  is overlapping 						✓	
7	•  is overlapping 						✓	Decreasing Y Sort
8	•  is overlapping 						✓	
9	•  is overlapping 						✓	
10	•  is overlapping 						✓	
11	•  is overlapping 						✓	
12	• New condition							

FIGURE 12.21 All six events containing the same action.

Movement

Do not use the default movement in this game. Instead, you will create your own. You will learn about all the different built-in movement types in Chapter 14. Sometimes it is useful to create your own movements in a game, as this allows you to be a little more unique and create it to fit in with a certain type of movement you are trying to create. The default movement will serve many movement purposes except when you are trying to make your games more unique. This game uses a screen map that is tested with the red box. If movements overlap, the movement will stop. If movements do not overlap, the red box will move in that particular direction by a number of pixels. You need to check if the red box is overlapping or not, as well as checking which direction key (arrow key) the player is pressing.

Create the comment box first.

1. Right-click on event line 12 and then select Insert | A comment. Type "Movement" and then click OK.

For the first four events you need to create two conditions in each event. The first is a negated condition checking if the red box (coll_player) is overlapping the Movement_Map and repeating while the direction arrow is pressed. Add these four events before you add the actions.

2. Click on New Condition on event line 13. Right-click on Coll_player and then select Collisions | Overlapping another object from the pop-up menu. From the next dialog box choose the Movement_Map object and click OK. Right-click on the condition you have just created and select Negate from the pop-up menu. A red X will now appear before the event.
3. Right-click again on the condition you have just added and select Insert. Then from the pop-up menu select The mouse pointer and keyboard object and The keyboard | Repeat while a key is pressed. When the dialog box appears asking you to select a button, click the Right Arrow button. Your event now looks like Figure 12.22.

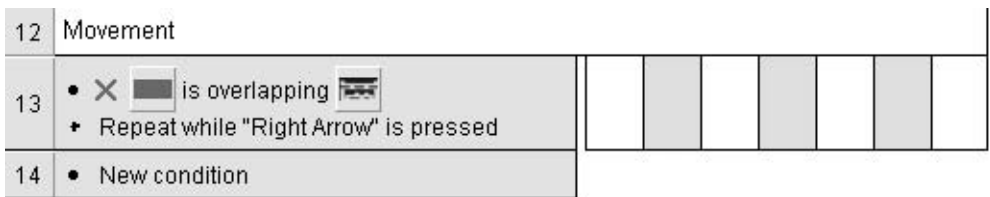


FIGURE 12.22 The two-condition event.

4. You now need to create the next three events using the same process. See Table 12.6 for details of what each event should contain.

Table 12.6 The Negated Events for Movement

OBJECT	IS OVERLAPPING	OBJECT
[Negated] Coll_player Repeat while Left Arrow is pressed	Is Overlapping	Movement_Map
[Negated] Coll_Player Repeat while Up Arrow is pressed	Is Overlapping	Movement_Map
[Negated] Coll_Player Repeat while Down Arrow is pressed	Is Overlapping	Movement_Map

5. Your four events should now look like Figure 12.23.

You now need to add the actions for these four events, and all the actions take place on the Coll_player object. You need to add two actions for each event. The first action in each will get the object's current position and then add or subtract a number from its position. If the player has pressed the up or down keys, this will add or

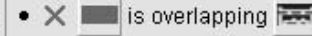

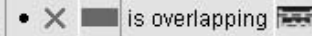

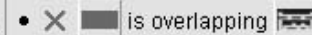

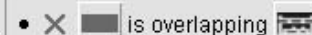

12	Movement		
13	<ul style="list-style-type: none"> •  is overlapping  • Repeat while "Right Arrow" is pressed 		
14	<ul style="list-style-type: none"> •  is overlapping  • Repeat while "Left Arrow" is pressed 		
15	<ul style="list-style-type: none"> •  is overlapping  • Repeat while "Up Arrow" is pressed 		
16	<ul style="list-style-type: none"> •  is overlapping  • Repeat while "Down Arrow" is pressed 		
17	<ul style="list-style-type: none"> • New condition 		

FIGURE 12.23 The four negated two-condition events.

subtract the Y coordinate, and the left and right arrows will take off or add to the X coordinate. The second action for each box sets the `coll_player`'s direction. The direction of the `coll_player` is used to set the direction the Player object will look. Therefore, when the red box is pointing right, it then knows the player has pressed the right key on the keyboard and should display the Player animation pointing to the right.

Now create the actions for event line 13.

- From event line 13 move across to the right until you are directly under the `Coll_player` object and right-click on the action box. From the pop-up menu select Position | Set X Co-ordinate. The Expression Evaluator will then appear. You want to get the current X position of `Coll_player` and then add four to it. This makes it move to the right.
- Click on Retrieve data from an object. When the dialog box appears, right-click on the `Coll_player` object and choose Position | X Co-ordinate. This places some text in the Expression Evaluator. Place the cursor at the end of the expression and then type "+4" (without the quotes). Your code now looks like the following:

```
X("coll_player")+4
```

- Click on OK to save this action. Now you need to create the second action in the same box, so right-click again on the `Coll_player` box for event line 13 and choose Direction | Select Direction. The Direction dialog box will appear, and as you are editing the event, which is about pointing to the right, the arrow is in the correct position, so click on OK. Your actions should look like Figure 12.24.

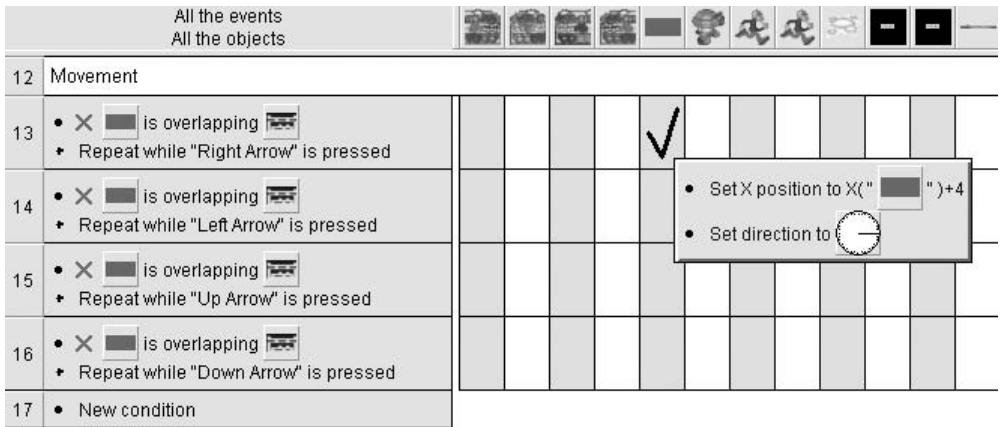


FIGURE 12.24 The two actions required for the first event of the movement section.

You need to do very similar things for the other three events. See Table 12.7 for details on how to configure them.

Table 12.7 The Actions for the Other Three Events

EVENT LINE	WHAT THE ACTION SHOULD SAY
14	Set X position to $X \times 4$ Set direction to Left
15	Set Y position to $Y \times 4$ Set direction to Up
16	Set Y position to $Y + 4$ Set direction to Down



If you get stuck and can't remember which options are required for both actions, see the previous example for the actions for event line 13.

Your events and actions should resemble Figure 12.25.



We have modified the image in Figure 12.25 to show you the actions for each event. Hold your mouse over the action box to reveal the contents and confirm that they are correct.

You need to create another set of events to check for when the Coll_player is overlapping, and if it is, to move the box away from the collision and set the direction. You might be wondering why you would need to move the box; if you didn't, it would always be overlapping the Movement_Map once after it first hit it. Until the

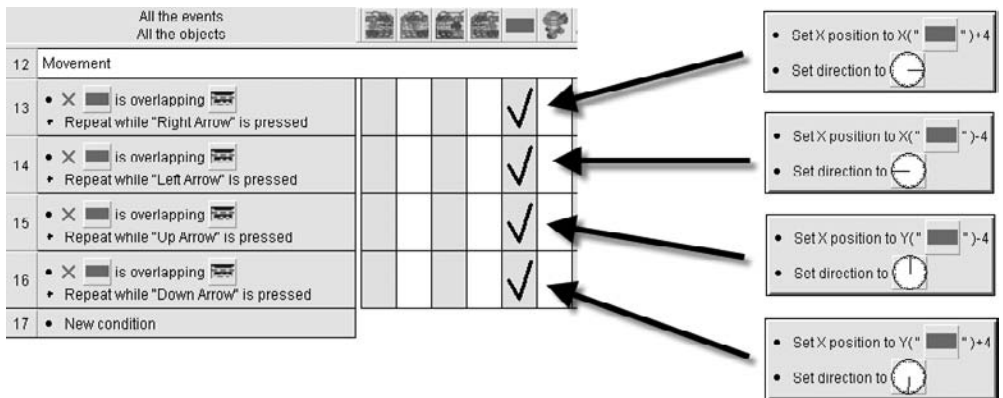


FIGURE 12.25 Actions for all four events.

player changes direction, no more movement would happen. This would provide the ability to stop the player from moving.

The event conditions are exactly the same as the ones you just created but without the Negate option being used. Create the conditions for the first of the new events now.

9. Click on New Condition on event line 17. Right-click on Coll_player and select Collisions | Overlapping another object from the pop-up menu. From the next dialog box choose the Movement_Map object and click OK.
10. Right-click again on the condition you have just added and select Insert. From the pop-up menu select The mouse pointer and keyboard object and then The keyboard | Repeat while a key is pressed. When the dialog box appears asking you to select a button, click the Right Arrow button.

Now create the event conditions on event line 18.

11. Click on New Condition on event line 18. Right-click on Coll_player and select Collisions | Overlapping another object. Choose the Movement_Map object and click OK.
12. Right-click again on the condition you have just added and select Insert. From the pop-up menu select The mouse pointer and keyboard object and then The keyboard | Repeat while a key is pressed. When the dialog box appears asking you to select a button, press the Left Arrow button.

Now create the event conditions on event line 19.

13. Click on New Condition on event line 19. Right-click on Coll_player and select Collisions | Overlapping another object. Choose the Movement_Map object and click OK.

14. Right-click again on the condition you have just added and select Insert. From the pop-up menu select The mouse pointer and keyboard object and then The keyboard | Repeat while a key is pressed. When the dialog box appears asking you to select a button, press the Up Arrow button.

Now create the event conditions on event line 20.

15. Click on New Condition on event line 20. Right-click on Coll_player and select Collisions | Overlapping another object. Choose the Movement_Map object and click OK.
16. Right-click again on the condition you have just added and select Insert. From the pop-up menu select The mouse pointer and keyboard object and then The keyboard | Repeat while a key is pressed. When the dialog box appears asking you to select a button, press the Down Arrow button.

Your four new events should look like Figure 12.26.









17	<ul style="list-style-type: none"> •  is overlapping  • Repeat while "Right Arrow" is pressed
18	<ul style="list-style-type: none"> •  is overlapping  • Repeat while "Left Arrow" is pressed
19	<ul style="list-style-type: none"> •  is overlapping  • Repeat while "Up Arrow" is pressed
20	<ul style="list-style-type: none"> •  is overlapping  • Repeat while "Down Arrow" is pressed
21	<ul style="list-style-type: none"> • New condition

FIGURE 12.26 The four new events on event lines 17–20.

Now you need to create the actions for each of the four events, and again it is similar to the actions you already created for the previous four. There will be two actions to each event, under the Coll_player object. The first of the two actions will add or subtract from the box's position on screen, and the second sets the direction it is looking.

First, create the two actions for event line 17.

17. From event line 17 move across to the right until you are directly under the Coll_player object and right-click on the action box. From the pop-up menu select Position | Set X Co-ordinate. The Expression Evaluator will then appear. Get the current X position of Coll_player and then subtract four to move to the left slightly.

18. Click on Retrieve data from an object. When the dialog box appears, right-click on the Coll_player object and choose Position | X Co-ordinate. This places some text in the Expression Evaluator. Place the cursor at the end of the expression and type "-4" (without the quotes). Your code now looks like the following:

```
X("coll_player") - 4
```

19. Click OK to save this action. Now you need to create the second action in the same box, so right-click again on the Coll_player box for event line 17 and choose Direction | Select Direction. The Direction dialog box will appear, and as you are editing the event, which is about pointing to the right, the arrow is in the correct position, so click OK. You can see the actions for this event in Figure 12.27.

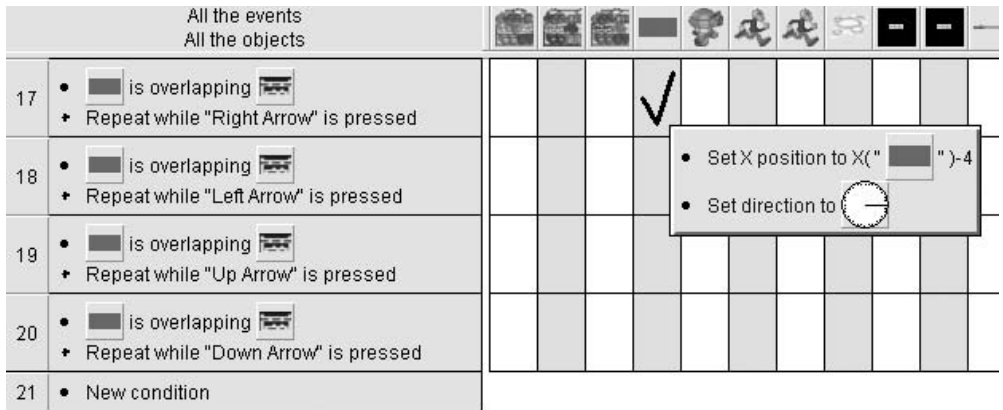


FIGURE 12.27 Actions for event line 17.

You need to do very similar things for the other three events. See Table 12.8 for details on how to configure them.

Table 12.8 The Actions for the Other Three Events

EVENT LINE	WHAT THE ACTION SHOULD SAY
18	Set X position to X + 4 Set direction to Left
19	Set Y position to Y + 4 Set direction to Up
20	Set Y position to Y - 4 Set direction to Down

You can see all of the events in Figure 12.28.

All the events		All the objects			
12	Movement				
13	<ul style="list-style-type: none"> • is overlapping • Repeat while "Right Arrow" is pressed 				✓
14	<ul style="list-style-type: none"> • is overlapping • Repeat while "Left Arrow" is pressed 				✓
15	<ul style="list-style-type: none"> • is overlapping • Repeat while "Up Arrow" is pressed 				✓
16	<ul style="list-style-type: none"> • is overlapping • Repeat while "Down Arrow" is pressed 				✓
17	<ul style="list-style-type: none"> • is overlapping • Repeat while "Right Arrow" is pressed 				✓
18	<ul style="list-style-type: none"> • is overlapping • Repeat while "Left Arrow" is pressed 				✓
19	<ul style="list-style-type: none"> • is overlapping • Repeat while "Up Arrow" is pressed 				✓
20	<ul style="list-style-type: none"> • is overlapping • Repeat while "Down Arrow" is pressed 				✓
21	• New condition				

FIGURE 12.28 The eight movement events.

If you run the game now, you can try to move the robot Player object but it will seem as if nothing is happening. Actually you are moving the coll_player object, but it is transparent so you cannot see it. Later in the game you need to tell the Player object to move with the transparent box to complete the robot’s movement. You’ll have the chance to do this a little later in this chapter.

You can also see that the cleaning fluid is moving outward from the player, because the whole animation is playing. This animation and the speed of the object make it move halfway across the screen. Next, you will destroy it once it gets to a certain animation frame so that it will only appear for a short time.

Cleaning Fluid

You want to destroy the Clean_Fluid object as soon as it gets to a particular animation frame because at this point it gets to animation frame 8 and then just glides across the screen with the graphics that are in that last frame. If you destroy it at frame 7 (you could pick another frame number if you wanted), it will only live for a short period of time, just long enough to appear at the back of the robot.

Create a comment line first.

1. Right-click on event line 21 and select Insert | A Comment. In the edit box type “Cleaning Fluid” and then click OK.

Now create the checking of which frame is playing.

2. Click on New condition on event line 22. Right-click on the Clean_Fluid object and then Animation | Compare current frame of Clean_Fluid to a value. In the Expression Evaluator leave the drop-down box at Equal and type "7" in the expression area. Click OK.
3. Move across from this event until you are directly under the Clean_Fluid object, right-click the action box, and select Destroy.

You have now completed the event to destroy the cleaning fluid before it goes swirling around the screen. You can see the event, comment, and action in Figure 12.29.

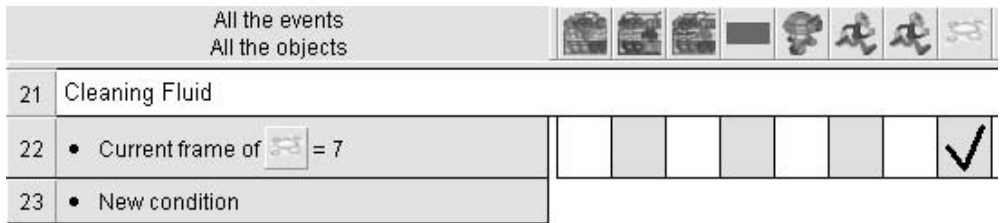


FIGURE 12.29 Cleaning fluid event and action.

Collisions Between Cleaning Fluid and Dirt

The next set of events checks for when the cleaning fluid is overlapping the dirt objects, at which point you want to destroy the dirt and add 1 to the score.

First, create the comment line.

1. Right-click on event line 23, select Insert | A comment, type in "Collisions," and click OK.

Now create the two events before you add the actions.

2. Click on New Condition on line 24 and select the Clean_Fluid object followed by Collisions | Overlapping another object. In the dialog box that appears, select Dirt 1 and click OK.
3. Click on New Condition on line 25 and select the Clean_Fluid object and then Collisions | Overlapping another object. In the dialog box that appears, select Dirt 2 and click OK.

Now add the two actions for each event. The first is to destroy the dirt object that is being overlapped, and the second is to add to the score.

4. Move to the right of event line 24 until you are under the Dirt 1 object, right-click and select Destroy.

5. On the same event line (24), move under the Rubbish_Cleaned object, right-click, and select Add to counter. Type the number “1” in the expression box and click OK.
6. Move to the right of event line 25 until you are under the Dirt 2 object, right-click, and select Destroy.
7. On the same event line (25), move under the Rubbish_Cleaned object, right-click, and select Add to counter. Type the number “1” in the expression box and click OK.

Your events should look like Figure 12.30.

All the events All the objects		[Toolbar icons]																				
23	Collisions																					
24	• is overlapping									✓											✓	
25	• is overlapping										✓											✓
26	• New condition																					

FIGURE 12.30 Collision events.

To separate the next bit of code, add a blank event line.

8. Right-click on event line 26 and select Insert | A comment. Don’t type anything in the edit box. Click OK.

Computer Players Group

The next step is to create a group to store six events, which handle what will happen when the badguys collide with the player object and when they throw rubbish onto the floor. They are in a group because groups keep code tidy and make it easier to understand and change. If you have problems with the computer players, it is easy to locate the problem, as you will have stored all the code in a single group.

First, create the group, which is enabled at the start of the frame.

1. Right-click on event 27 and select Insert | Group of events. When the dialog box appears, type “Computer Players” into the Title of the group and then click OK.

You now have a group that is expanded. You can tell when it’s expanded as you can see the New Condition text below it. You must ensure that the group stays expanded throughout this next part of the code, as this dictates what line numbers exist. When the group is expanded, the line numbers include the events in the group. When it’s collapsed, it does not.

Create the first event in this group, whereby when the badguys collide with the robot (player) and then reverse their movement.

2. Click on New Condition on line 28. This is the event line within the group. Select the Special object and then Limit Conditions | Only one action when event loops. The first condition has been added. You now need to add a second condition to the same event.
3. Right-click on the condition you have just added, select Insert and then select Badguy 1 from the dialog box and Collisions | Another object. Select the Player object and click OK.
4. Move across to the right of this event until you are under the Badguy 1 object, right-click the action box, and select Movement | Reverse.

You need to create a similar set of conditions and the same action for Badguy 2.

5. Click on New Condition on line 29 and then on Limit Conditions | Only one action when event loops.
6. Right-click on the condition you have just added and select Insert. Then select Badguy 2 from the dialog box and select Collisions | Another object. Select the Player object and click OK.
7. Move across to the right of this event until you are under the Badguy 2 object, right-click the action box, and select Movement | Reverse.

You can see the two events in Figure 12.31.

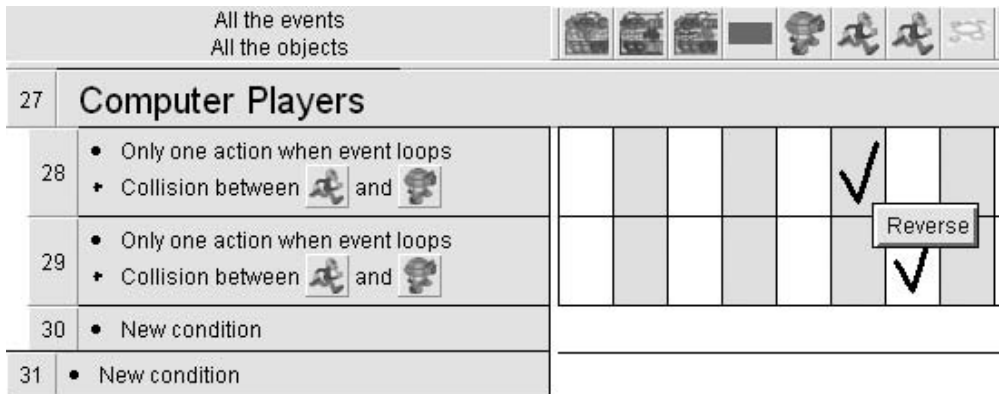


FIGURE 12.31 The first two events in our Computer Players group.

You now need to create four events, which contain four conditions each. Two of the events are for Badguy 1 and the second two are for Badguy 2. The two events are separated into left and right movements. These events handle how often and when the badguys throw rubbish. The badguys only throw rubbish when they are overlapping one of the shelves (using the neutral group), when they are facing left or right, and only when a random number is equal to 0. This random number can be increased to increase the litter drop time or decreased to increase the drop rate.

Create the first event of this group of four.

8. Click on New Condition on event line 30 in the Computer Players group and then select Limit Conditions | Only one action when event loops.
9. Right-click on the condition you have just added and select the Badguy 1 object and Collisions | Overlapping another object. Then select the Group.Neutral object.
10. Right-click on the condition you've just added and select Insert. Select Badguy 1 and Direction | Compare Direction of Badguy 1. The next dialog box is pointing to the right, which is what you require, so click OK.
11. Finally, right-click again on the last condition you have just added and select Insert. Select the Special object and choose Compare two general values. When the Expression Evaluator appears, type in "Random(30)" and leave the second box as "0." Click OK.

In the last condition TGF2 runs a random number from 0 to 30 during each loop of the event. Only when this number is 0 will this event condition be true, and as long as all the other conditions are true, it will run the actions. Your conditions should look like Figure 12.32.

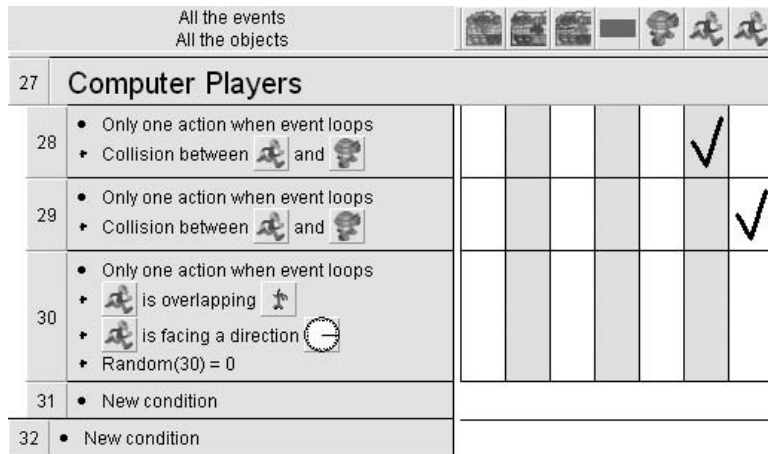


FIGURE 12.32 The four condition events.

Create the next event the same way but change the direction to left.

12. Click on New Condition on event line 31 in the Computer Players group and select Limit Conditions | Only one action when event loops.
13. Right-click on the condition you have just added and select the Badguy 1 object and Collisions | Overlapping another object. Then select the Group.Neutral object.

14. Right-click on the condition you've just added and select Insert. Select Badguy 1 and Direction | Compare Direction of Badguy 1. The next dialog box is pointing to the right. Change this so it is pointing to the left and then click OK.
15. Finally, right-click again on the last condition you have just added and select Insert. Select the Special object and choose Compare two general values. When the Expression Evaluator appears, type in "Random(30)," and in the second box leave this as "0." Then click OK.

Finally, create two more events for the right and left, but replacing Badguy 1 with Badguy 2. Once you have done that, your events should look like Figure 12.33.

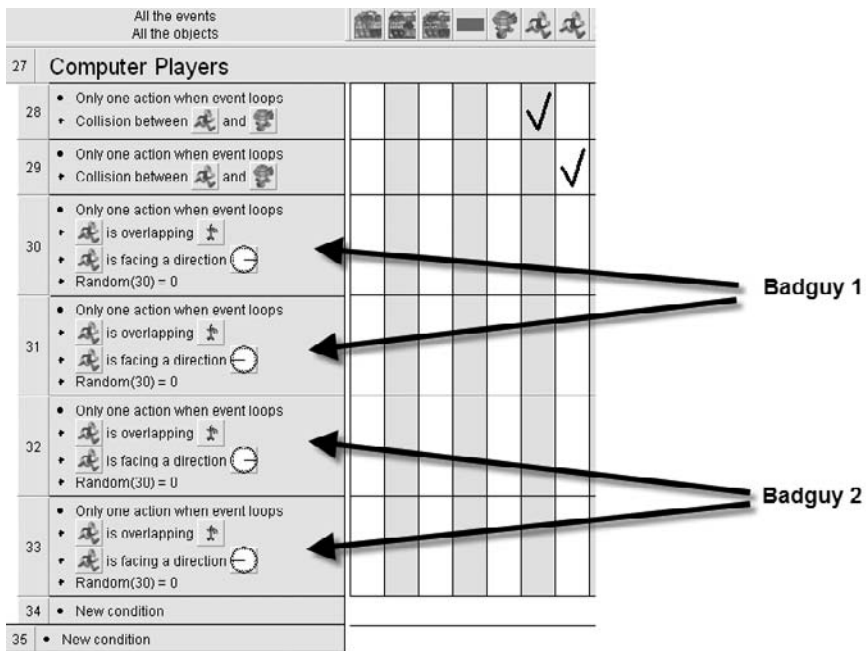


FIGURE 12.33 All four events, two for Badguy 1 and two for Badguy 2.

You now need to create the actions for each of these events, where a sound will be played to show the user that something has happened, and create some dirt (either Dirt 1 or Dirt 2 depending on the badguy). Start with event line 30.

16. Move across from event line 30 until you are directly under the Sound object. Right-click and select Samples | Play Sample.
17. Click on the Browse button opposite the From a file option. Browse to the DVD, navigate to the Samples folder and open the file GLASS 3.wav.



18. Still on event line 30 move across until you are under the Create New Objects object. Right-click and select Create object. In the dialog box that appears, select Dirt 1 and click OK. A Create Object dialog box should appear. Click on the Relative to button, select Badguy 1, and click OK. Ensure that the X and Y boxes are set to 0. You can see the configuration of the dialog box in Figure 12.34.

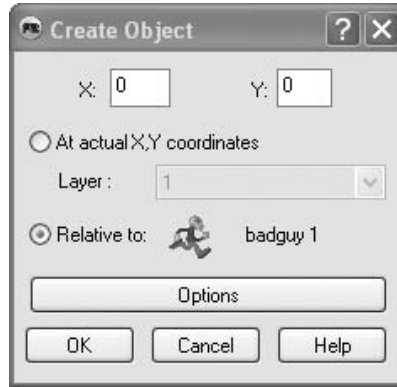


FIGURE 12.34 The Create Object location box.

19. Click OK.

Follow the same process for event line 31.

You need to follow the same procedure for events 32 and 33, but this time for the create action you need to select Badguy 2 and Dirt 2. Here's the process for event line 32:

20. Move across from event line 32 until you are directly under the Sound object. Right-click and select Samples | Play Sample.
 21. Select GLASS 3 from the samples box and click OK.
 22. Still on event line 32 move across until you are under the Create New Objects object. Right-click and select Create object. In the dialog box appears, select Dirt 2 and click OK. A Create Object dialog box should appear. Click on the Relative to button, select Badguy 2, and click OK. Ensure that the X and Y boxes are set to 0.

Follow the same process that you used for line 32 for event line 33.

You have completed this section of code, which handles the computer players. Leave the group expanded until you have completed the rest of the code. Otherwise, it will affect the event line numbering.

End Code

The final code for this frame involves tidying up the loose ends, getting the player to move, setting the objects scores, and checking when the litter dropped is over 40.

First, create the comment line.

1. Right-click on event line number 35 and select Insert | A comment. In the dialog box type “End Code” and click OK.

Now it’s time to create the first event of this last section. This sets the Player object to the same direction and position as the Coll_player object. Then set the String_score object to the current counter value of how many bits of rubbish have been cleaned up.

2. Click on New Condition on event line 36. Select the Special object and then the Always option.
3. Move to the right of this event until you are under the Player object, right-click, and select Direction | Select Direction. In the direction box that appears, click on the Calculate Direction button (this appears as a button with 1+1 on it). This opens up the Expression Evaluator. Click on the Retrieve data from an object button and then select the Coll_player and from the pop-up menu select Animation | Current Direction value. It now says the following in the Expression Evaluator:

```
Dir(“coll_player”)
```

4. Click OK button.
5. You need to add a second action on the same event line (36). Right-click on the Player box where you just created an action and select Position | Select Position. Click on the Relative to button, choose the Coll_player object, and click OK. Type in the X position of 24 and Y of 10 and then click OK.
6. You now need to add another action on the same event line but in a different action box. Move across until you are directly under the String_score object. Right-click and select Order | Bring to front.
7. Finally, for this event you need to add one more action to the String_score object, so right-click on the action box and select Change Alterable String. The Expression Evaluator should appear. As you are placing a number within a string object, you need to convert it, so click on the Str\$ button. This enters part of the information into the box. Click on Retrieve data from an object while the “>Enter number here<” is selected. Right-click on the Rubbish_Cleaned object. From the pop-up menu, select Current Value. This places the following in the Expression Evaluator:

```
Str$(value(“Rubbish_Cleaned”))
```

8. Click OK. The events and actions should look like Figure 12.35. The actions have been placed in the picture also, so you can compare them against your own code.



FIGURE 12.35 The events and actions you just created.

The next event will again be an Always event, with the actions calculating the total number of dirt items that are in play on the frame. Remember, there are two dirt objects: Dirt 1 for Badguy 1 and Dirt 2 for Badguy 2. You can work out the total by just adding up the total number of objects on the frame.

9. Click on New Condition on event line 37. Select Special | Always.

The first action for event line 37 is to set the total number of Dirt objects in the Counter object. This is the bar that counts up in the bottom middle of the game screen. The more dirt there is on screen, the higher the counter goes.

10. Move to the right of event line 37 until you are directly under the Counter object. Select Set counter, and the Expression Evaluator will appear. Click on Retrieve data from an object and select the Dirt 1 object, followed by Count | Number of objects. This enters an expression, but you want to add the objects from Dirt 2, so at the end of the text type a plus sign and again click on Retrieve data from an object. Select Dirt 2 and Count | Number of objects. Your Expression Evaluator should now show the following expression:

```
Objects("dirt 1")+Objects("dirt 2")
```

11. Click OK to save the information to the action box.

Now you need to follow the same process for the Counter_NLvl. This is the counter that is checked to see if it has reached 40 in another event. If it has, the game will end. This counter keeps track of how the player is doing, so the game can be stopped when the player cannot clean the floor quickly enough.

12. Move across event line 37 until you are under the Counter_NLvl object. Right-click and select Set counter, and the Expression Evaluator will appear. Click on Retrieve data from an object and select the Dirt 1 object and then

Count | Number of objects. Add a plus sign at the end of this expression and again click on Retrieve data from an object. Select Dirt 2 and Count | Number of objects. Your Expression Evaluator should now show the following expression:

```
Objects("dirt 1")+Objects("dirt 2")
```

13. Click OK to save the information to the action box.

Now it's time to add the last event of this frame. This event checks the Counter_NLvl object to see if it is greater than 40. If so, it sets the score collected in the Rubbish_Cleaned counter and the game moves to the Next Frame.

14. Click on New Condition on event line 38. Right-click on the Counter_NLvl object and select Compare the counter to a value. The Expression Evaluator now appears. Change the Comparison Method box to Greater and enter 40 in the Edit box. Click OK.

15. Move to the right of this event until you are under the Player 1 object (the icon looks like a joystick). Select Score | Set Score. When the Expression Evaluator appears, click on Retrieve data from an object and then find the Rubbish_cleaned object. From the pop-up menu choose Current value. The Expression Evaluator now has the following in it:

```
value( "Rubbish_Cleaned" )
```

16. Click OK.

17. Move under the Storyboard Controls object, right-click, and select Next Frame.

Congratulations. You have completed the events for the Game frame.

Highscore Programming

You only need one event and one action for the final frame. This checks for when the timer is greater than 10 seconds. When it is, go to the Menu frame.

First, make sure you are working on the Highscore Event Editor.

1. Double left-click on Highscore in the Workspace toolbar. You will see the Frame Editor and the objects placed on screen. Click on the Event Editor button to display the blank Event Editor frame.

Now you can create the event.

2. Click on New Condition on event line 1. Select the Timer object and then Is the timer greater than a certain value. When the Timer dialog box appears, change the seconds slider so that it reads 10 seconds and then click OK.

3. Move to the right of this event until you are under the Storyboard Controls object, right-click, and select Jump to Frame. You will see the dialog box shown in Figure 12.36, which lists all the frames to which you can jump. You want to jump to Frame 1, which is already highlighted, so click OK.

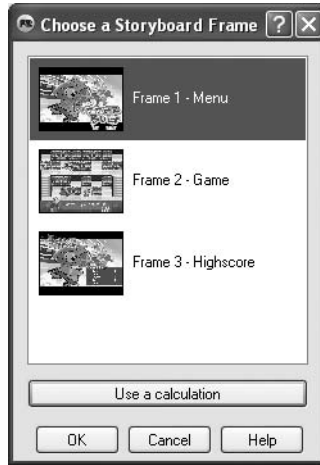


FIGURE 12.36 The dialog box to select a frame to jump to.

You have now completed the game. Run the whole application and see how it works and think about how you could improve on it, by either making changes to the actual game or making harder levels whereby the badguys drop more litter.



You can see the completed game executable (`litterbug.exe`) located on the DVD in the `TGFFILES\Litter bug`. The full code to this game can also be found in the same folder and is called `litterbug.mfa`.

CHAPTER SUMMARY

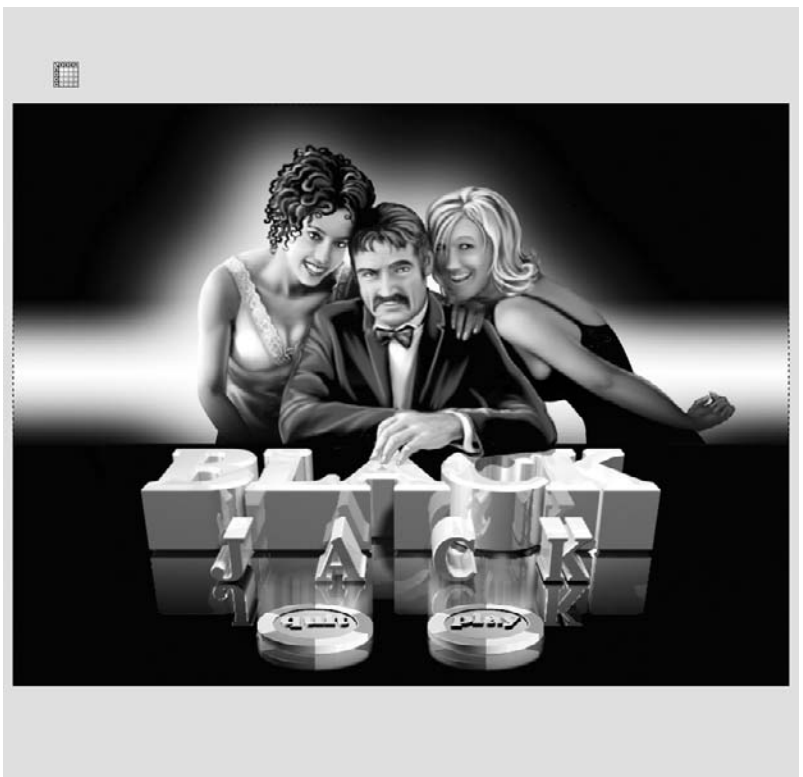
In this chapter you created a whole new game in TGF2, starting with creating the frames, placing the graphics, and then finally creating the game. We hope you have gained a lot of knowledge of how to structure your own games and now feel pretty confident in how to use TGF2.

This page intentionally left blank

ADVANCED GAME OVERVIEW

In This Chapter

- Advanced Games



This chapter looks at two games that are already made and takes a quick tour around them to show you how complex they can get and give you an idea of what you can do with the TGF2 program if you want to.

ADVANCED GAMES

When using various tools, it is a useful exercise to visit the relevant Web sites and download different game examples and games of various genres. This gives you a good overall feel of what the product is capable of. This also gives you an opportunity to think about the types of games you would like to create and the features they might include.

Black Jack

Card games are more complex games to create in TGF2 and Game Maker than other types of games. The reason for this is that the programmer has to think more about the rules and logic of the game, rather than moving and shooting objects using some of the built-in features of the engines. You also have to take into account the computer-controlled player and its ability to play the game.

Blackjack is one of a number of popular card games. In Black Jack you play against a dealer, and whoever gets closest to 21 is the winner. The kings, queens, and jacks are all counted as 10 each, and the ace can be counted as a 1 or 11. If you go over 21, this is considered a bust, and you lose all the money you bet.

Frames

The Black Jack game has been split into two frames, as can be seen in Figure 13.1. The first frame is the standard menu screen, which introduces the player to the game. This is essential for providing an easy to understand start screen before leading the player into the game. The second frame is the game itself. You could also create a third frame to display the highest amounts of money the players have gained. The screen size has been set to 800 × 600 for this game.

Application Properties

By clicking on Black-jack in the Workspace toolbar, you can access the application properties. Here you can set how the application will be displayed. You can see the properties for the application in Figure 13.2.

In the Window tab the No Maximize box and the No Thick frame are selected. This prevents the user from making the game fill the whole screen. First, by removing the option to maximize the screen (the square box in the top-right corner of the game window) and, second, by selecting the No Thick frame, this prevents the user from dragging the window to a different size. Another option that has been removed is the Menu bar. This provides quick access to text menu options at the top of the game window. It is very useful in certain types of games and particularly in applications, but it is not needed for many games.

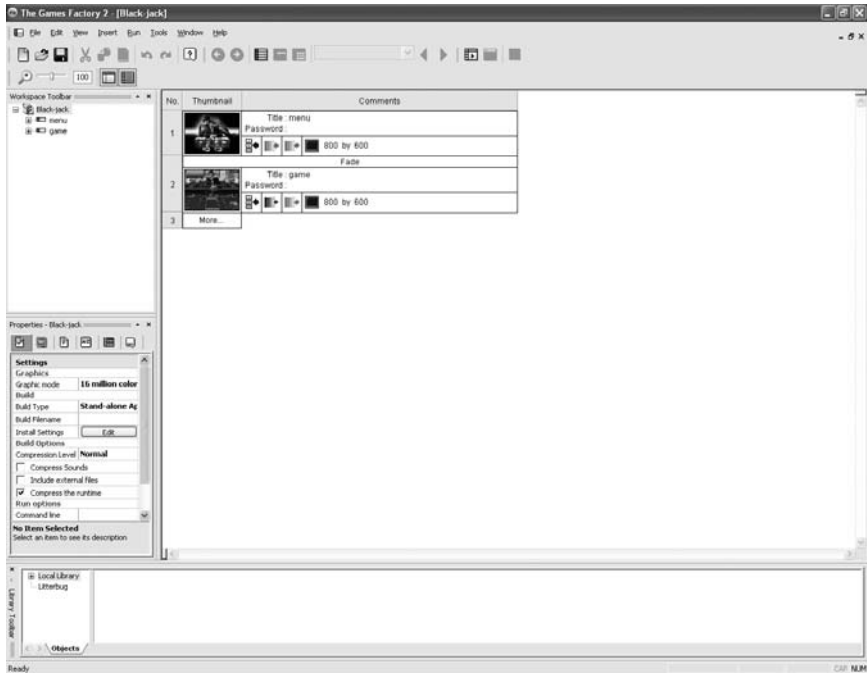


FIGURE 13.1 The two frames needed for this game.

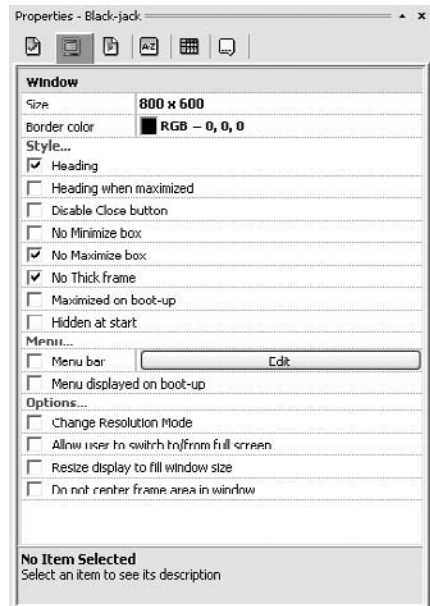


FIGURE 13.2 The application properties for the Black Jack game.

Where possible, you should disable anything you don't want the user to be able to use. This reduces the amount of support emails and problems caused by not turning them off. For example, if you do not switch off the Menu bar, you need to ensure that the contents of the Menu bar all work and are useful to the game.

Files



You can find the completed game on the DVD, located in the TGFFILES\Black Jack folder. The file is called BlackJack.exe. For the game to work correctly, you need to copy the whole Black Jack folder to a writable drive because the game writes information back to the array files to store information about the cards and the files containing various information. Otherwise, as the DVD is read-only, the game will not work as expected.

You can find the source code to this game in the TGFFILES\Black Jack\ folder. The file is called BlackJack.mfa.

You need to open the BlackJack.mfa file in TGF2 to follow the following examples.

Menu Frame

The menu frame allows the user to quit or play the game by clicking on one of two buttons. You can see the screenshot of the Black Jack menu frame in Figure 13.3.

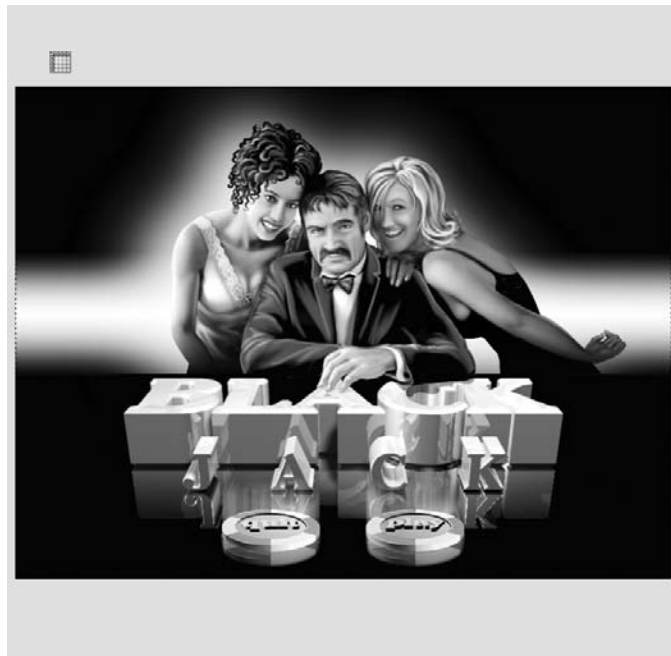


FIGURE 13.3 The contents of the Black Jack menu frame.

The frame consists of a few objects including:

- An image for the frame
- Two buttons for the user to click on
- An array

You have dealt with a number of items before, but this is the first time you have seen an array in a TGF2 game. An array is a special type of object that you can use in TGF2 to load, save, and store text or numbers at runtime. By clicking on the object, you can reveal the object's properties as shown in Figure 13.4.

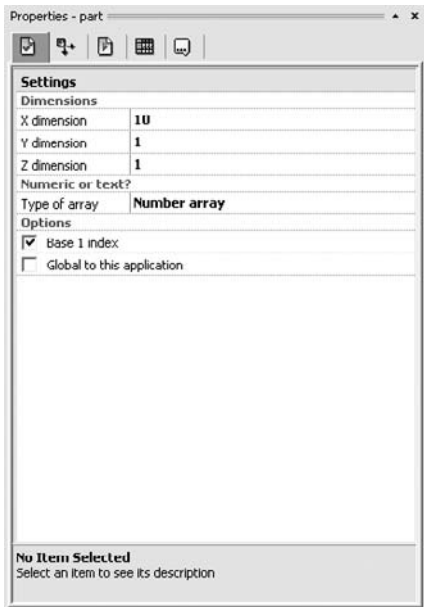


FIGURE 13.4 The array properties.

You can see that the array has dimensions. This is like a grid in which it can store information. When it is set to 10, 1, 1, it means that it will store everything in the X dimension from either 0 to 10 or 1 to 10. Whether it starts from a 0 or 1 depends on another option discussed below called Base 1 Index. In this array file, next to Type of array it states Number array. This means it is going to store numbers. Arrays can only store numbers or letters and cannot do both at one time. If you want to store information in an array and want to store both types you can create two array objects, one for each type of information. Note that the Base 1 Index option is selected. This means the X dimension will be referenced starting from the number 1. You can reference the array through the Event Editor, and when you want to access an entry in the array, you have to specify the X dimension number. If you were using the other dimensions, you would also specify that particular number as well.



For more information on arrays please consult the product help files.

Menu Event Editor

There are seven events for this first frame, mainly because it doesn't have to do much work. Most of the code, much like the games you've already made, is stored in the Game frame. You can see the events in Figure 13.5.

All the events All the objects		[Icons]											
1	• Start of Frame	<input checked="" type="checkbox"/>									<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
2	• SemiTrans(" [Image] ") > 0 • Mouse pointer is over [Image]										<input checked="" type="checkbox"/>		
3	• SemiTrans(" [Image] ") < 128 • <input checked="" type="checkbox"/> Mouse pointer is over [Image]										<input checked="" type="checkbox"/>		
4	• SemiTrans(" [Image] ") > 0 • Mouse pointer is over [Image]											<input checked="" type="checkbox"/>	
5	• SemiTrans(" [Image] ") < 128 • <input checked="" type="checkbox"/> Mouse pointer is over [Image]											<input checked="" type="checkbox"/>	
6	• User clicks with left button on [Image]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>										
7	• User clicks with left button on [Image]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>										
8	• New condition												

FIGURE 13.5 The events for this frame.

You should be comfortable with all of the event conditions in this frame, as they are very similar to the ones in the previous games. Most of the code is used to create a fading and appearing effect when the player moves the mouse over either of the two buttons.

In Figure 13.6 the array has been given the value 10 to (1). This (1) is the dimension (position) in the X array, so in the 10 slots available to the array (from 1 to 10) the number 10 has been placed into slot 1. This number is used for the starting number of chips the player has and saves this amount to an array file. This array file changes the amount throughout the game to hold how many chips the player has, so it is important to reset it at the start of a new game so the player does not get an unfair advantage when starting again. If you want to save information to an array file, you first write the changes to the array and then save it to a file.

Game Frame

The game frame is the busier of the two frames, and can be seen in Figure 13.7. On the game frame you can see many different objects and items. Here is a quick run-down of what some of them do:

- The banker, with a number of animations for talking and looking upset if he loses money.

All the events All the objects													
1	• Start of Frame		✓								✓	✓	✓
2	• SemiTrans(" ") > 0 • Mouse pointer is over										✓		
3	• SemiTrans(" ") < 128 • ✕ Mouse pointer is over										✓		
4	• SemiTrans(" ") > 0 • Mouse pointer is over											✓	
5	• SemiTrans(" ") < 128 • ✕ Mouse pointer is over											✓	
6	• User clicks with left button on	✓	✓										
7	• User clicks with left button on	✓	✓										
8	• New condition												

- Write Value 10 to (1)
- Save array to file Appdir\$+Appdir\$+part.ani

FIGURE 13.6 The writing and saving to an array file.



FIGURE 13.7 The game frame for Black Jack.

- A number of other graphic objects.
- Three active graphic objects for increasing, decreasing, and accepting stakes. These objects have a transparency added.
- A speech bubble with Yes or No buttons that are used to allow the player to answer questions the banker asks. For example "Do you want another card?"
- A File object, which the program uses to save and load files. These files are needed for storing information about the game.
- A text box placed over the speech bubble. This is a blank text box but is loaded with the correct information from a list box as the game continues.

- Two list boxes, which load the questions and card numbers when the frame starts.
- Lots of counters to keep track of the cards in play, how many cards have been dealt, and the money on the table.
- Two arrays: one to keep track of the cards for the bank and one to keep track of the cards for the player.

Game Event Editor

Most of the code for this game section is stored in various groups. Two groups are enabled at the start of the frame, but the rest of the groups are disabled until they are required by the program code to run. You can see all of the groups in Figure 13.8.

The screenshot shows a software interface for editing game events. At the top, there are two tabs: 'All the events' and 'All the objects'. Below the tabs is a toolbar with various icons for editing and a row of ten numeric input fields, all currently set to '0'. The main area is a table with 17 rows, each representing a group. The first three rows have a small dropdown menu on the left and a grid of checkboxes on the right. The first row has a checkmark in the 10th column. The third row has a checkmark in the 16th column. The remaining rows are simple text labels.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Start of Frame									✓								
No music is playing																	
Always																✓	
INI																	
YES / NO Buttons																	
First round																	
Calculate the Players hand																	
Calculate the Bank Hand																	
First round																	
Stake																	
Insurance against a natural black jack of the cpu																	
Natural black jack for cpu																	
Double stake																	
Other cards for player																	
Other cards for cpu																	
End of normal game : who win ?																	
New condition																	

FIGURE 13.8 The groups in use in the program.

Some of these groups handle:

INI. This initializes the game and loads and deletes any of the files required by the game.

YES/NO. These are events to handle the display of the Yes and No buttons that are displayed in the banker's speech bubble.

First. This handles the first round of cards

Calculate the Player's hand. This works out the total value of the player's hand.

Calculate the Bank hand. This group works out the value of the bank's hand.

First round check. This checks the first round of cards and provides options depending on card that has been dealt.

Stake. This handles the money being put into the stake by using the + or – buttons.

Insurance against Black Jack. The player is offered insurance against the banker getting black jack,

Black Jack for CPU. This is the code that runs when the CPU has black jack. It will also check to see if the player took out insurance against the computer getting black jack.

Double Stake. This allows the player to double his stake if he thinks has a good hand.

Other cards for player. This handles the other cards for the player and takes into account doubling the stake.

Other cards for CPU. This handles the other cards for the CPU. If the current score is under 17, the computer continues to get cards. If the computer has over 17, it will stick.

End of normal game: who win? This compares the scores once all the cards have been dealt, works out the winner, and, if needed, adds money to the player's chips.

Alterable Variables and Flags

The game extensively uses alterable variables and flags. Alterable variables are good for storing, retrieving, adding, and comparing information, while flags are useful as checks in your game, as you can turn them on or off and compare their states. Every object can have a flag or alterable values assigned to it. You can see some examples of alterable variables and flags in use in Figure 13.9.

All the events All the objects		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0															
4	INI																
5	YES / NO Buttons																
6	First																
7	<ul style="list-style-type: none"> Internal flag 0 is off Alterable Value B of = 0 	✓											✓				
8	<ul style="list-style-type: none"> Internal flag 0 is off Alterable Value B of = 1 	✓											✓				
9	<ul style="list-style-type: none"> Internal flag 0 is off 																
10	<ul style="list-style-type: none"> Internal flag 0 is off Alterable Value B of > 2 																
11	<ul style="list-style-type: none"> Internal flag 0 is on Alterable Value C of = 0 	✓												✓			
12	<ul style="list-style-type: none"> Internal flag 0 is on Alterable Value C of = 1 	✓												✓			
13	<ul style="list-style-type: none"> Internal flag 0 is on 																
14	<ul style="list-style-type: none"> Internal flag 0 is on Alterable Value C of > 2 	✓															
15	New condition																
16	Calculate the Players hand																
17	Calculate the Bank Hand																
18	First round Check																
19	Stake																
20	Insurance against black jack																
21	Black Jack for cpu																
22	Double stake																

FIGURE 13.9 Flags and alterable variables being used in Black Jack.

Loops

In Figure 13.10 you can see a selection of events that contain the words “on loop.” A loop is something that repeats over and over until it reaches a pre-defined limit. This loop runs a set of actions repeatedly, which can greatly speed up your games or make repetitive processes much easier to handle without needing to repeat the code multiple times.

Event ID	Description	Obj 1	Obj 2	Obj 3	Obj 4	Obj 5	Obj 6	Obj 7	Obj 8	Obj 9	Obj 10	Obj 11	Obj 12	Obj 13	Obj 14	Obj 15	Obj 16	Obj 17
6	First																	
7	Calculate the Players hand																	
8	Internal flag 14 is off			✓														
9	Internal flag 14 is on			✓														
10	On loop "calculation" (ValueAB(13, value(0))) mod 13 = 0				✓													
11	On loop "calculation" (ValueAB(13, value(0))) mod 13 >= 10				✓													
12	On loop "calculation" (ValueAB(13, value(0))) mod 13 <= 0, (ValueAB(13, value(0))) mod 13 <= 1, (ValueAB(13, value(0))) mod 13 < 10				✓													
13	On loop "calculation" 0 > 10, (ValueAB(13, value(0))) mod 13 = 1				✓													
14	On loop "calculation" 0 <= 10, (ValueAB(13, value(0))) mod 13 = 1				✓													
15	to modify the value of an ace when you are up to 21																	
16	On loop "calculation" 0 > 21, Internal flag 14 is on, (ValueAB(1, 1)) mod 13 = 1				✓								✓					
17	On loop "calculation" 0 > 21, Internal flag 14 is on				✓								✓					

FIGURE 13.10 Loops are used extensively in the Black Jack game.

Dragons

Dragons is a side scrolling game where you play the part of a dragon. You must try to survive the traps in the game’s caverns and hallways and locate the treasure. You control the dragon using the arrow keys and use the space bar to shoot fire to try and destroy the spiked balls. You can see a screenshot from the game in Figure 13.11.

Files



You can find the completed game on the DVD, located in the TGFFILES\Dragons folder. The file is called Dragons.exe. You can find the source code to this game in the TGFFILES\Dragons\ folder in a file called Dragons.mfa.

You need to open the Dragons.mfa file in TGF2 to follow the following examples.



FIGURE 13.11 A game screenshot of Dragons in action.

Frames

The Dragons game has been split into three frames, which can be seen in Figure 13.12. The first frame is the menu screen, which introduces the player to the game. The second frame is where all the hard work is done and the user can play the game. The third frame is a highscore frame where the player can see the scores when he has completed the game.

Application Properties

The most important application properties of Dragons can be seen in Figure 13.13. You can access this by clicking on the word “Dragon” in the Workspace properties toolbar and then clicking on the Window tab. Notice that this game doesn’t have a heading or a Menu bar selected. A No Minimize box and No thick frame have not been selected because this game displays across the whole screen. This is known as full screen and is common in professional and independent games that need to hide the desktop and only display the game. To make the game fill the whole screen you need to tick Change resolution mode. You can still display the menu in this case, but it doesn’t look very professional and should be removed unless it is necessary.

Notice that frames one and two are 800×600 , but the second frame is set to $3,000 \times 1,125$. This is because it will scroll to the right and scroll a small amount up and down to allow the dragon to move around the screen.

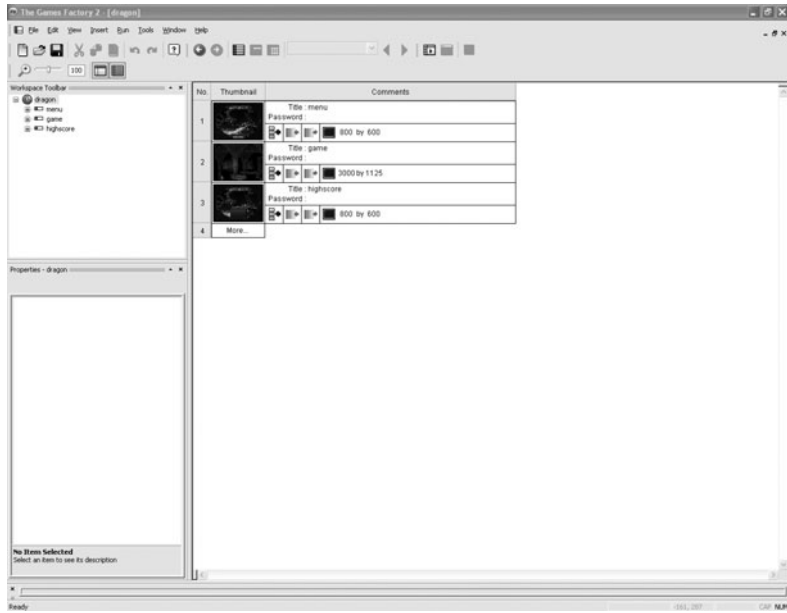


FIGURE 13.12 The three frames needed for this game.



FIGURE 13.13 The application properties for Dragons.

Menu Frame

The menu frame allows the user to either quit or play the game. When the frame starts, there is a dragon noise and then music. You can see a screenshot of the menu screen in Figure 13.14.



FIGURE 13.14 The Dragon menu frame.

The frame consists of three objects:

- The background image
- A button for starting the game
- A button for quitting the game

The menu frame is very straightforward and very similar to the two games you have already created in TGF2. You can see a screenshot of the events and conditions used in this frame in Figure 13.15.

All the events All the objects								
1	• Start of Frame	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	• Mouse pointer is over	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3	• X Mouse pointer is over	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4	• Mouse pointer is over	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
5	• X Mouse pointer is over	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
6	• User clicks with left button on	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7	• User clicks with left button on	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
8	• Always	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
9	• New condition	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

FIGURE 13.15 The Event Editor code for the menu frame.

Game Frame

The game frame is the busiest frame of the three and can be seen in Figure 13.16. The game frame contains many items, including:

- A dragon with flying animations already created.
- An object called `Collide_player`, which is used the same way as the red box object in *Litter Bug* to handle movement and collisions of the object. This object has been given an Eight Direction movement. This means the movement of the box is handled by TGF2's own movement engine, unlike in *Litter Bug*, where you wrote your own. The dragon is told to follow precisely this red box around the screen.
- An object called `GUI`, which contains the health and fire bars.
- Several arch objects that the dragon will fly behind and in front of.
- Several attack zones that are used to check when the player has entered them so that the spikey ball can then go flying into the dragon.
- Several backdrop objects to improve the overall look and feel of the scene.
- Several objects assigned to making fire effects on the screen.
- A collisions map, which is the same concept as that used in *Litter Bug*, where the player cannot move into the colored areas on the grid.
- Two counters for fire and health.
- Several enemy objects that reduce the player's health if they hit the dragon.
- A clock, which is a simple active object with several animation frames counting down.



FIGURE 13.16 Part of the game frame with many items on screen.

Game Event Editor

This game has some code stored in groups, but also has some code in object behaviors. This will be discussed shortly, but when looking at the number of events, you might wonder where code for some of the program is, as there doesn't seem to be much for the amount of things happening in the game. Most of the code in the Event Editor for this frame is not within groups, but there are still four groups:

Display. How to handle displaying the life and backgrounds. This handles the layers in the game and what objects can appear in front of others. It also handles the collision between the dragon's red movement box and the background grid. You can see some code from this group in Figure 13.17.

Enemy Sphere. This handles the sound effects when the red collision box hits the Group.bad qualifier group. It also handles the reduction of the dragon's health.

End. This handles the destruction of certain objects at the end of the game to remove them from the screen. If time has run out, it will display a time-out box on screen. This then calls the final group called After End.

After End. This group will take the player to the high score screen or restart the game.

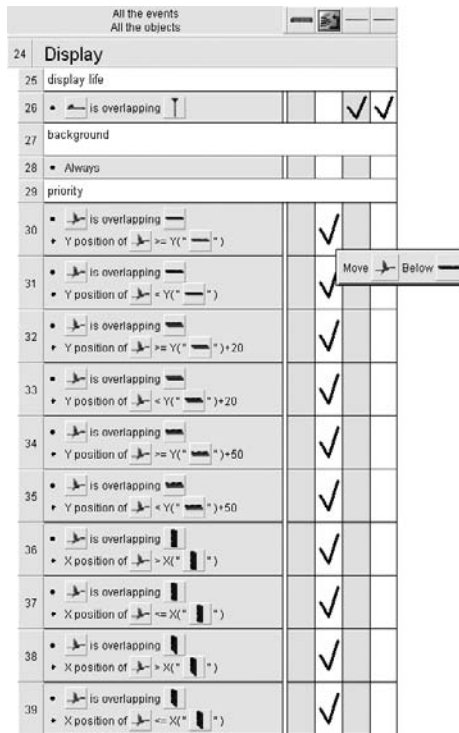


FIGURE 13.17 Some of the code in the Display group.

Alterable Variables and Flags

This game also uses alterable variables and flags extensively in the game. Some of these events can be seen in Figure 13.18. These save a number to the alterable value, which is then read back in an Always event to set the speed of the dragon. This action can be seen in event line 10.

FIGURE 13.18 The alterable values used in Dragons.

Behaviors

If you look at the Event Editor code, there doesn't seem to be enough code in the Game Editor to handle everything that happens. This is because some code is placed within the objects themselves. This is called behaviors and is very useful for programming an object to react a certain way in a game. You could place this code at the event level, but there are a couple of benefits of placing code directly into the object:

- If you copy an object into another game, the behaviors are copied with it to the new program. This means you can create objects that do specific tasks and not worry about reprogramming it every time you want to use that object in another game.
- If you want to change how an object reacts in the game, you can change the code in the object rather than having to look through lots of code in the main game Event Editor. This can be a good way of keeping your code clean.

To ensure that you can recognize an object with behaviors attached, the letter B appears on the bottom-left corner of the object's icon in the Event Editor as shown in Figure 13.19.

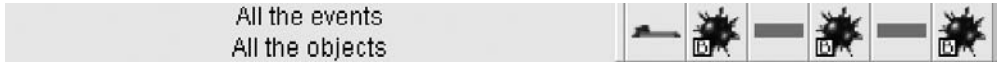


FIGURE 13.19 Behaviors attached to an object.

To access the object behavior, you need to be on the Frame Editor. Select an object that you want to either add or view the current behaviors for. In this example, the Events tab of the Properties of object Enemy 1 has been selected. You can see that this object has a qualifier set and a behavior labeled #1, as shown in Figure 13.20.

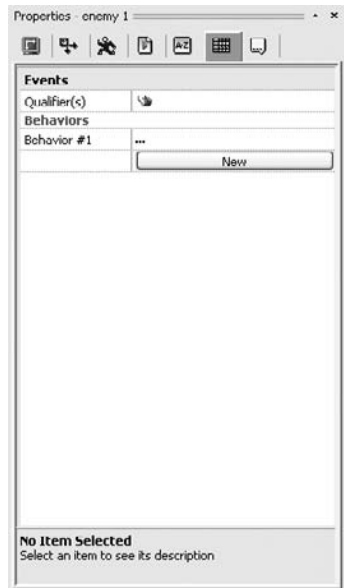


FIGURE 13.20 The Events tab for the Enemy 1 object.

If you click on the dotted lines next to Behavior #1, a little Edit button will appear. If you click on this, you will enter the object Behavior Event List. You can see an example of the code in the Object Behavior settings in Figure 13.21. It looks exactly like code in the normal Event Editor, and it is.

Event	1	2	3	4	5	6	7	8	9	10	11	12
1 • Start of Frame												
• Internal flag 0 is on								✓				
2 • Above 0 ?												
• is overlapping												✓
3 • Internal flag 0 is on								✓				
4 • Internal flag 0 is on								✓				
5 • Alterable Value B of >= 360								✓				
6 • Internal flag 0 is on								✓				
• Collision between and								✓		✓		
7 • Internal flag 0 is on								✓				
• Alterable Value C of <= 0								✓				
8 sphenes highscore												
9 • is overlapping												
• internal flag 0 is off												✓
10 • internal flag 0 is on												
• internal flag 2 is off												
• internal flag 1 is off											✓	
• internal flag 0 is on											✓	
• is overlapping												
11 • internal flag 2 is off												
• internal flag 1 is on												
• X position of > X(" ")												
12 • internal flag 2 is off												
• internal flag 1 is on												
• X position of < X(" ")												
• internal flag 2 is off												

FIGURE 13.21 The behavior code in the object.

Highscore Frame

The highscore frame consists of only three objects:

- A backdrop object that covers the whole frame, which is for decoration only
- An active object that is a picture of the highscore table, which covers the background image
- A highscore object that displays any scores in the active object area.

The frame can be seen in Figure 13.22.

Highscore Event Editor

The Highscore Event Editor only has two event lines. The first is to check when the timer, which starts at the beginning of the frame, equals 1 second. If it does, then play a specific sample and loop it, so it continues playing until the frame is exited. The second event checks to see when the timer equals a time greater than 10 seconds, at which point it restarts the application. This means it automatically starts from the menu frame. You can see the two events in Figure 13.23.

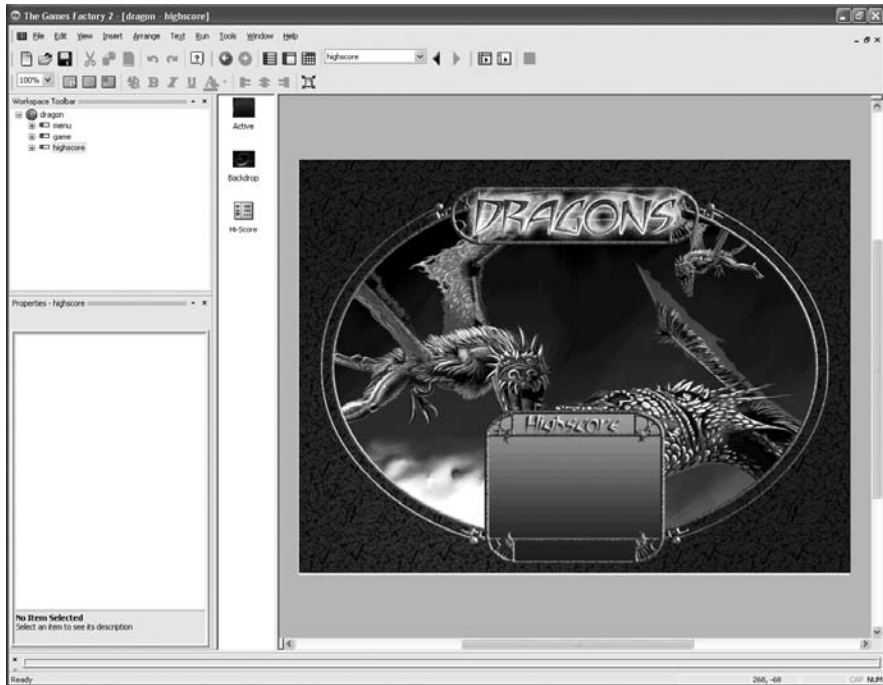


FIGURE 13.22 The highscore frame.

All the events All the objects		
1	• Timer equals 01"-00	<input checked="" type="checkbox"/>
2	• Timer is greater than 10"-00	<input checked="" type="checkbox"/>
3	• New condition	<input type="checkbox"/>

FIGURE 13.23 The two events in the Highscore frame.

CHAPTER SUMMARY

In this chapter we looked at two advanced games that use several features in the Event Editor to make games even more powerful. It may take a little more practice and a few simpler games under your belt before you begin to use some of these more powerful features, but it is important for you to know that they exist, as they add a lot of power to your game making arsenal.

Take a look at the code and see if you understand what it is doing and how it does it. This should give you some ideas of the types of things you could add to your own games.

In the next chapter you will be looking at how to use built-in movements.

This page intentionally left blank

ADVANCED CONTROL OF OBJECTS

In This Chapter

- Using Objects in Your Games
- Active Objects
- Backdrop and Quick Backdrop Objects
- Hi-Score Object
- Text Objects
- Lives Object
- Score Object
- Movement
- Multiple Movements



You may have noticed that in the previous chapters, you were never called upon to assign movement types or create new objects. All we have done is use an already created set of game items that you placed on the frame. The next few chapters will cover ways to create and manage your own game assets. We will start by looking at active objects and their movement, text objects, hi-score objects, and backdrop objects—the mainstays of TGF2 games.

USING OBJECTS IN YOUR GAMES

TGF2 comes with a selection of objects. When creating the games in this book you dragged these objects and placed them onto the frame. You may have been unaware that these were all created from a basic template of objects and then configured before they were placed into the library file from which you took them. TGF2 comes with a set of about 36 built-in objects that you can use in your creations. Common objects that you have already used, possibly without realizing it, are the active, text, hi-score, and backdrop objects.

To use an object in your games, you first have to place it on the frame. To do this you right-click the Frame Editor and select Insert Object or from the menu, “Insert | New Object.” You then see a list of objects to choose from, which are categorized for easier selection as shown in Figure 14.1.

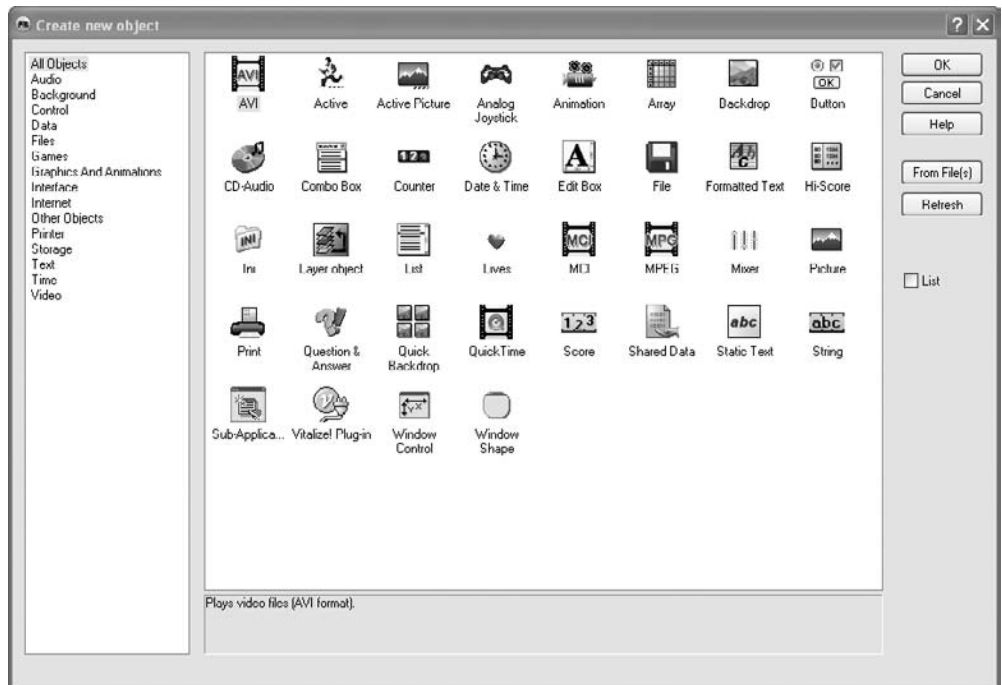


FIGURE 14.1 The selection of objects available in the TGF2 trial version.

Once you have placed an object onto the frame, you can configure its properties and assign it a movement type id.

Various objects exist for various tasks. For example, if you want to create a button, you would use a button object, or if you want video to be played in your creation, you would select the QuickTime object that would allow you to play Apple QuickTime video files.

Let's now delve into the common objects that you will use on a regular basis in your game creating.

ACTIVE OBJECTS

Active objects are used mainly as the main characters of your games and will be the most common object used in your programs. In the games you created in earlier chapters, the objects had behaviors and movement already applied to them, so all you had to do was drag and drop them onto the frame, but when you create your own you can assign and configure the active object properties. For example, you can configure it so the player can use the mouse or keyboard, or you can have the computer control the objects for you. You can also make your active objects animated, making them run, jump, or do whatever you decide.

Active objects are denoted by the icon of a running man in the Create new object dialog box and as a green diamond in the Event Editor. The icons can be seen in Figures 14.2 and 14.3.



FIGURE 14.2 The active object icon in the Create new object dialog.

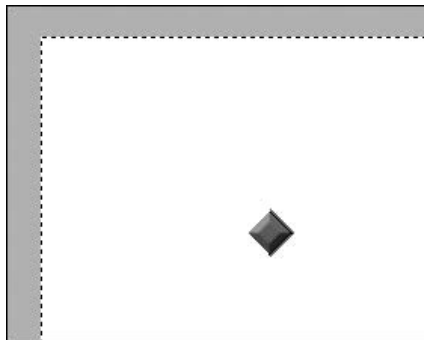


FIGURE 14.3 The active object as shown in the Frame Editor.

To place the active object on the frame, right-click on the frame, select Insert | Object, and then click Active and click OK. Then left-click somewhere on the frame.

You can change the object's properties by single left-clicking on it and accessing the object's Properties workspace on the left-hand side, as shown in Figure 14.4. Don't worry about the image of the object or animation just yet, as those will be covered in the next chapter.

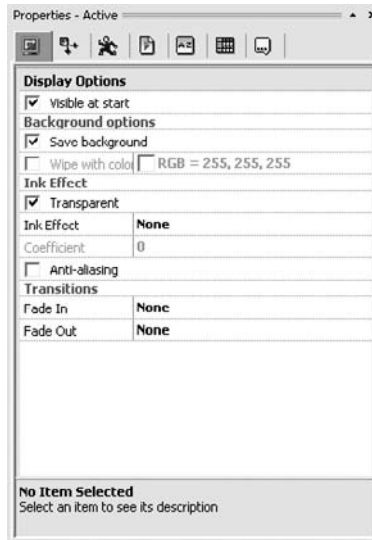


FIGURE 14.4 The active object Properties.

Every object you place on the frame can have a different set of properties. Across the top of the Properties worksheet are a number of graphical tabs that provide access to different properties from size and position to movement. Some objects have more or fewer tabs depending on the type of object.



If you hold your mouse cursor over any of the tabs, a little help text message will appear telling you what each tab is.

The tabs available in this active object are:

Display options. This provides different options for how the object should appear on the frame.

Size/Position. What is the object's size and position on the frame? This tab is used a lot to place an object at a specific location.

Movement. All objects are set to Static by default. You can specify a movement type within this tab.

Runtime Options. How should the object react when the program is running?

Values. You have the ability to store data of both strings (text) and numbers that are accessible to the whole application. This can be done for each object and is very useful when you are accessing the same information over each frame.

Events. You can create specific events, which are stored in the selected object. This means you can create code on how an object reacts in a certain instance and then use these objects in other TGF2 programs and the code will still exist.

About. This is your author information and the help file link. This is very handy if you don't know what a particular option of the object does, as you can click on this Help button and read the document.

You will have used the Size/Position tab when placing objects in a specific location on the frames in previous chapters. The rest of tabs are pretty much self-explanatory when you need to configure an object.

BACKDROP AND QUICK BACKDROP OBJECTS

Backdrop and quick backdrop objects perform a similar function—to provide a background to your games. You can think of it as setting the scene in the games, where you will then place your game characters on top. A backdrop object is usually constructed by using the Picture Editor by importing or drawing an image. You could also use one of the many backdrops provided on the CD-ROM of the full version of the software. The Picture Editor is discussed in the next chapter. A quick backdrop allows you to create a background image using a selection of colors and use a specific shape to apply to it. You can see the icons for the backdrop and quick backdrop objects in Figure 14.5.



FIGURE 14.5 The backdrop object icon.

The backdrop object has very few properties, but under the Settings tab you can access the Picture Editor and then draw or load a picture. The quick backdrop has more properties located in the Settings tab. First, you can amend the shape of the object by selecting it as a rectangle, a line, or an ellipse. When selecting the line, you can only select the color and its width, but using rectangle or ellipse opens up a new set of options and you can create a “fill.” From here can select the following options:

None. This allows you to create a rectangle with a border that is configurable with a color and size. The contents of the rectangle stay with the frame color unless you change the width to a size that completely fills the shape.

Solid Color. This option fills the shape (rectangle or ellipse) with a single solid color. You can change the color by clicking on the Color Settings property.

Gradient. This creates a background that changes from one color to another. This is an easy way of creating a sky background, which changes color as it goes further down the screen. If you have played old computer games, you might recognize gradients, as they were very common in games in the 1980s. To use the gradient you normally set the first color to the darkest color you require and then set the second color to the lightest color you require. The program then creates a shape using all of the colors in between to create a background. You can set it to be a vertical or horizontal gradient by ticking the Vertical Gradient button. You can see an example gradient running from black to a light red in Figure 14.6.

Motif. This is the only option in the quick backdrop object that allows you to access the Picture Editor. The motif type takes a single image and replicates it a number of times in the backdrop area. You can see an example of the motif in use in Figure 14.7, where a single image of a dragon has been selected, and the motif has placed it multiple times over the object.



FIGURE 14.6 The gradient setting used on the full size of the frame.

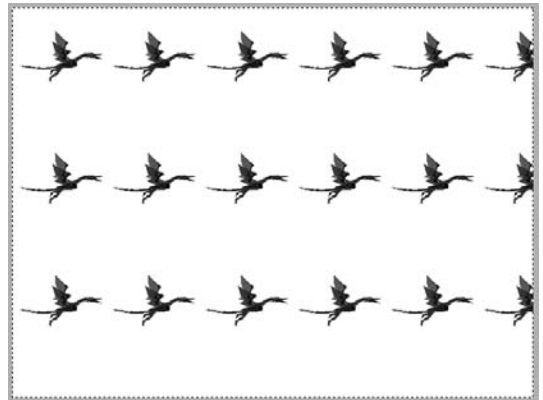


FIGURE 14.7 The motif setting using a dragon image on the full size of the frame.

Obstacle Type

Both the backdrop and the quick backdrop objects have an additional setting under the Runtime tab. You can change how other objects interact with a backdrop object by turning on or off the menu options in this tab under Obstacle Type. The options are shown in Figure 14.8.

There are four options:

None. This option means the backdrop object will not be an obstacle to active objects. You cannot detect a collision between the backdrop object and an active object when this option is turned on.



FIGURE 14.8 The obstacle options.

Obstacle. This option means it is possible to detect a collision with an active object. You must test for a collision with a backdrop object in the Event Editor and insert a stop action.

Platform. This option means the backdrop object acts as a platform for active objects controlled by platform-type movement. This is not the same as the Obstacle option. You cannot detect a collision between a backdrop object and an active object that has been assigned a platform-type movement.

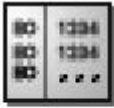
Ladder. This option treats the backdrop object as a ladder when an active object is using a platform-type movement. If the active object has an animation sequence, the animation is automatically changed when the object climbs a ladder. If the active object does not have an animation sequence, you can change the animation via the Event Editor.

HI-SCORE OBJECT

We have used the hi-score object a number of times in our games, where it displays the current top scores. These scores by default can contain a set of fake data for the player to try to beat, and once the player has beaten one of those scores, the hi-score object will also contain their score. The hi-score object icon is shown in Figure 14.9.

When you place the object on the frame, you are presented with the default scoreboard, which contains 10 empty entries as shown in Figure 14.10.

If you left-click on the object you can access the object properties. Click on the Settings tab to access the key properties of this object.



Hi-Score

FIGURE 14.9 The hi-score object icon.

Empty	0
Empty	0
Empty	0
Empty	0
Empty	0
Empty	0
Empty	0
Empty	0
Empty	0
Empty	0
Empty	0

FIGURE 14.10 The default scoreboard.

Some of the options for the hi-score table are:

Number of scores. How many scores do you want to display in the hi-scores table? The default is 10, but this is particularly useful if you only want to display a small hi-score table in your game.

Length of names. What length of name can the player enter when he gets a high score. If you are restricting the space that can be used for the hi-score table, this is useful. Additionally, if you are making a retro game, then you might be trying to replicate the three-character names that were used in many older games.

Show name before score. By default, the name is displayed before the score, but you can change that if required.

Hide at start. Do you want the hi-score table hidden at the beginning of the frame. If so, you can tick this checkbox. You may want to do this if you want to control when the hi-score table is displayed by using code to make it reappear.

Check at start. If you want the hi-score table to check if the player has a top score at the start of the frame (the default), leave this ticked. If, like in the games we have made, you have a separate frame for the game and the hi-score, leave this ticked.

Hide scores. This gives you the option of only displaying the names in the hi-score table.

Edit content. If you want to create a scoreboard with a set of data (recommended), you can click the Edit button to access a dialog box that allows you to put in some fake data, as shown in Figure 14.11. This is a good way to set a range of scores for the player to beat, so don't make it too easy for the player to reach the top score on his first try.

Name (Ini file to use). By default, the high scores are saved on the player's computer in a file called `cncscore.ini` in the Windows system directory. By entering a filename you can save it in a different file.

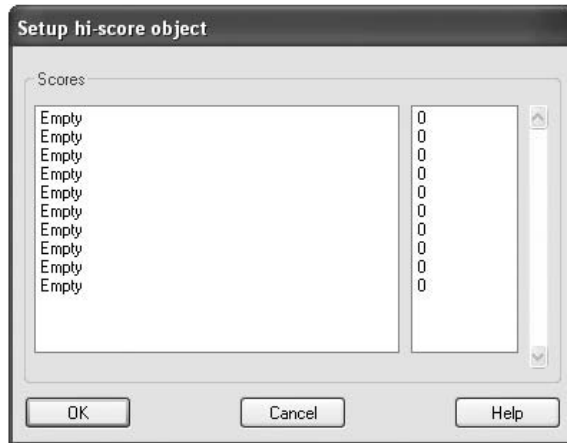


FIGURE 14.11 The edit content dialog box for the hi-score object.

TEXT OBJECTS

Text objects are used to put text on the screen. You can use them for instructions, comments, end of game displays, or just about anything that requires text. It is very easy to make your own text once you have placed a text object on the frame from the add object dialog.

There are three text-based objects in TGF2:

- Formatted text
- Static text
- String

You can see the object icons for all three of these objects in Figure 14.12.



FIGURE 14.12 The three text-based objects available in TGF2.

Available in all three objects is the ability to select fonts, styles, text sizes, and text color, as well as the justification style you want to use. The formatted text object allows you to access these features via the toolbar and the static text and string object via the Object Properties worksheet as shown in Figures 14.13 and 14.14.



FIGURE 14.13 The font options in the toolbar.



FIGURE 14.14 The font options in the Properties sheet.

LIVES OBJECT

The lives object is for keeping track of a player's lives in the game. The icon in the Add Object dialog can be seen in Figure 14.15.

Once placed on the Frame Editor it appears as three hearts, as shown in Figure 14.16. This means the player has three lives, which is the default setting for the lives object.



FIGURE 14.15 The lives object in the Add Object dialog box.

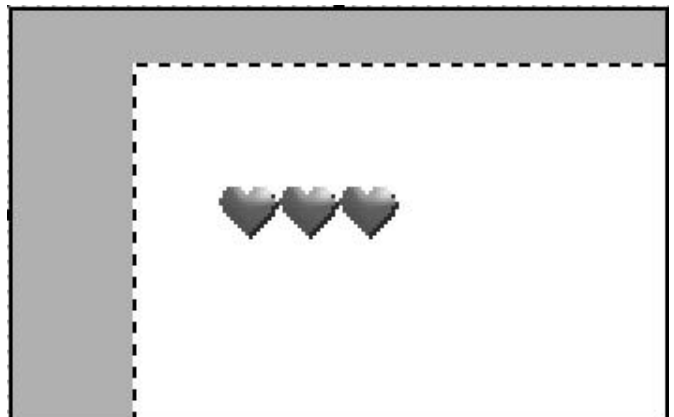


FIGURE 14.16 The lives object as displayed on the Frame Editor.

The Settings tab of the lives object can be seen in Figure 14.17 and has only three key settings:

Player. Which player do these lives images apply to? If you are creating a game with multiple players, for example, a two-player game, you can specify which player these belong to.

Type. You can change the display of the lives object by selecting one of the three options in the Type drop-down box. The default is Images, which displays a heart graphic. You can also select text or numbers. You can also edit the image option and replace it with your own lives images.

Image(s). This button allows you to access the Picture Editor and change the look of the lives object.

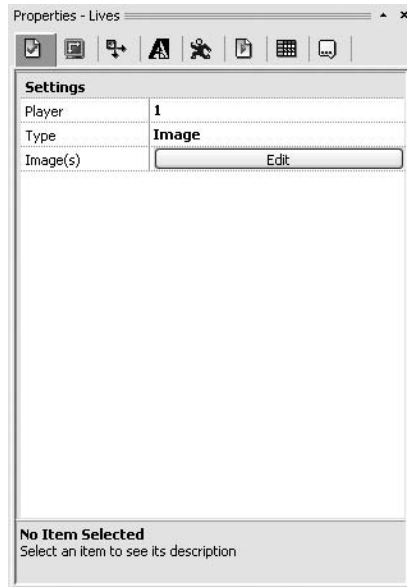


FIGURE 14.17 The Settings tab of the lives object.

To use the lives object within the context of your game, you need to use the Event Editor.



If you want to change the default number of lives from three to another number, you need to click on the application name and then click on the Runtime Options tab for the application properties. Under the Players heading you then have access to the initial number of lives option.

SCORE OBJECT

The score object is used to keep track of the current player's score in the game. You can see the score object icon as shown in the Add New Object dialog box in Figure 14.18.



Score

FIGURE 14.18 The score object icon in the Add New Object dialog.

Once you have added the icon to the desktop, it will automatically be displayed as a graphic number 0. The Settings tab of the object Properties has the same set of options as the lives object.

MOVEMENT

To change an object's movement from the Frame Editor, left-click the object to display the object Properties and then click on the Movement tab in the Properties workspace. You will then see the current movement options for this active object, as shown in Figure 14.19.



FIGURE 14.19 The Movement tab in the object Properties sheet.

Nearly every type of object in TGF2 can be assigned a movement, and though we are discussing it within the context of applying it to an active object, the process and options are the same for any other object.

Notice that the movement type of the object is currently set to Static. This is the default setting of all objects when they are created from the Insert | New Object menu. By clicking on Static, you reveal all of the available movement types, as shown in Figure 14.20.



More movement types are available in the full version after the latest patch has been installed.

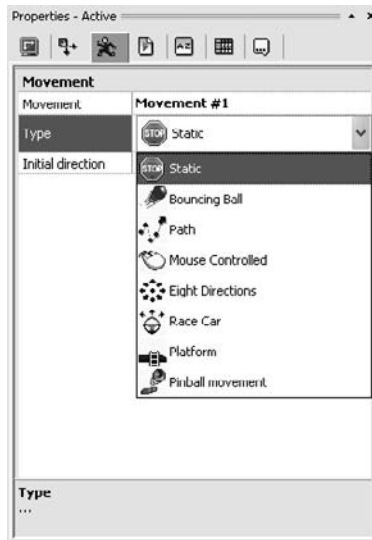


FIGURE 14.20 The available movement types in the trial version.

There are three computer-controlled options: Bouncing Ball, Path Movement, and Pinball Movement. There are four types of movement that can be used by the player of the game when controlling what is happening on the screen: Mouse Controlled, Eight Directions, Race Car, and Platform.

We will now go through each of these movement types so you have a better understanding of what they do and how you apply them to objects.

Bouncing Ball

This movement option, as shown in Figure 14.21, is normally used to produce an object that bounces around the screen like a ball. However, by changing several parameters and using the Event Editor, you can use this movement to control the movement of a host of aliens or other enemies that chase the player around. You can see the other key settings of the Movement tab for the Bouncing Ball in Figure 14.22.



FIGURE 14.21 The Bouncing Ball icon.



FIGURE 14.22 The Movement tab for the Bouncing Ball movement.

Initial Direction

Initial direction defines which direction the object moves when the game first starts. The numbers relate to a direction, for example, 8 is up and 0 is to the right. You can click on the numbers to reveal a direction chart, where you can remove or add ticks to tell TGF2 which directions the object is able to move. You can see the Initial Direction dialog in Figure 14.23.

Speed. Speed controls the speed of all the other types of movement.

Ball Deceleration. When this option is set to zero, a ball keeps bouncing forever. Increasing this value gradually slows your object down until it grinds to a halt.

Moving at start. The object moves automatically when the game starts. If you untick this option, you need to start the object moving via the Event Editor.

of Angles. Number of angles lets you set the bounce angle for the object. It can be 32, 16, or 8. The fewer angles selected, the fewer the directions the ball will bounce.

Bounce Randomizer. This option makes objects bounce in more random directions. As this number increases, so does the randomness.

Bounce Security. This option jiggles objects to keep them from getting stuck in corners, but as a result, the rebound effects are made slightly more random.

Bouncing Ball is covered in a little more detail later.

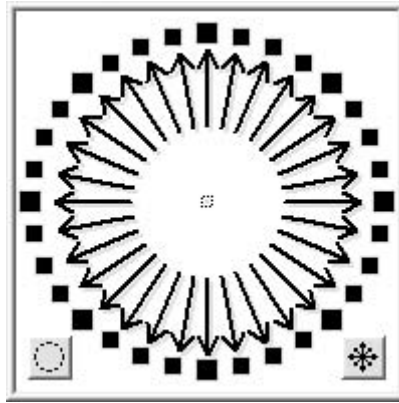


FIGURE 14.23 The Initial Direction dialog.

Path Movement

The Path Movement option shown in Figure 14.24 sets your object moving on a pre-determined path, which you define. For example, you can create a patrolling guard who walks a set distance and then turns around or who walks in a preset path around a corridor. This lets you control many parameters and script some neat effects, such as the looping and speed that an object will move with on different sections of its path.



FIGURE 14.24 The Path Movement icon.

To access the options for the Path Movement, click Edit.

Path Editor

As shown in Figure 14.25, six buttons let you define the movement of an object, plus the speed bar, which changes the speed at which the object moves along its path. A path-type movement is entered using your mouse to define the path.



FIGURE 14.25 The Path Editor option buttons.

New Line. This function adds a single line to the object's movement.



If you already have a movement defined, New Line is added at the end of it by default, unless you insert it somewhere else by choosing the insertion point with the mouse.

Tape Mouse. This function allows you to set a very complex path movement. By holding down the left mouse button and dragging the mouse pointer around the screen, you set the movement you want.

Pause. This function stops your object at its current position for a length of time that you define in seconds.

Loop the Movement. This function repeats a movement that you specify, over and over.



Each time the loop repeats, this function repositions the object back to its original starting position, so make sure the path finishes at the object's starting point, or the object will jump around the screen.

Reverse at End. This function reverses an object's movement and sends it backward along the original path. This function is good for a guard patrolling the grounds.

Reposition Object at End. This function puts your object back at its original starting position when it has completed the movement.

Try Movement. This function lets you try the movement before deciding upon it.

Editing a Path. Once you have added a movement to your object, you can edit it very easily in the Frame Editor. To do this, select the object, choose the Movement tab in the Object Properties toolbar, and then click Edit. This opens the Path Editor again. You can select individual points of the movement, or entire sections, by dragging a box around them. You can manipulate these selected pieces by either deleting them or using the Cut (CTRL-X), Copy (CTRL-C), and Paste (CTRL-V) keys. You can also simply drag one of the selected areas using the left mouse button.



Each box created on the path movement is called a node.

Configuring Node properties. After you've selected a point or area, you can add a condition to it by right-clicking on any node in the path. This lets you insert a condition at that spot, such as Set a Pause, Tape Mouse, and New Line. You can see the popup menu when you right-click a node in Figure 14.26.

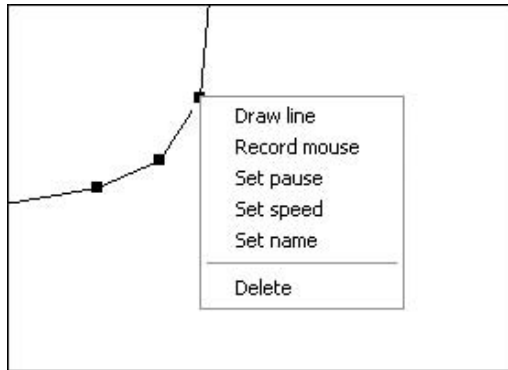


FIGURE 14.26 The node pop-up dialog.

Pinball Movement

This allows you to create a movement similar to the bouncing ball movement, but the ball reacts as if it were in a pinball machine. You can see the icon for this movement in Figure 14.27 and the properties in Figure 14.28.



FIGURE 14.27 The icon for the Pinball movement.

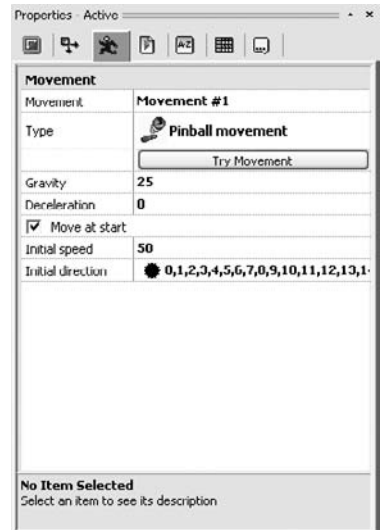


FIGURE 14.28 The Properties sheet for the Pinball movement.

Gravity. This option selects the effect of gravity. A high setting makes your object fall rapidly, allowing only short bounces.

Deceleration. Deceleration sets the rate at which your character object slows down.

Move at Start. When check marked, the Move at Start option causes the object to move in one of the directions you have chosen when the game starts. When this option is unchecked, the object remains stationary until told to move via the Event Editor.

Initial Speed. This is the initial starting speed of the object.

Initial Direction. This option allows you to choose one or more directions for your object to move when the game begins. If you choose more than one direction, TGF2 chooses one of the specified directions at random.

Mouse Controlled

The first type of player-controlled movement is Mouse Controlled. This makes the object exactly follow the movement of the mouse. The icon for this object can be seen in Figure 14.29. To edit the area where the mouse can move, click on the Edit box. The object will be surrounded by a box that represents the object's limits of movement, as shown in Figure 14.30.



FIGURE 14.29 The Mouse Controlled icon.

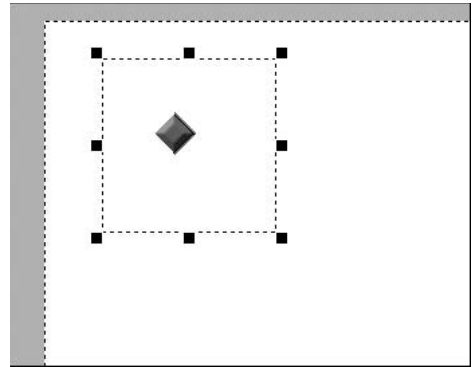


FIGURE 14.30 The Mouse Movement area control box.

You can stretch or shrink the area by grabbing the sizing handles with your mouse and dragging them around.



This box takes its position from the object, not from the screen. This means that if you move the object to a new position on the Frame Editor screen, you may need to edit this box again.

Try Movement

Try Movement tests your object's movement on the screen. To stop the object and return to the Mouse Controlled dialog, press the Escape key.

Eight Direction Movements

This movement control, whose icon is shown in Figure 14.31, provides you with the classic eight directions that are used by a joystick. You can also use the cursor keys to control movement. There are several basic controls. Speed, acceleration, and deceleration have been described previously. The Possible Directions option allows you to select or deselect the number of directions in which your object can move. See Figure 14.32 for the Movement Direction dialog Properties sheet and Figure 14.33 for the Directions and Initial Directions dialog.



FIGURE 14.31 The Eight Directions icon.

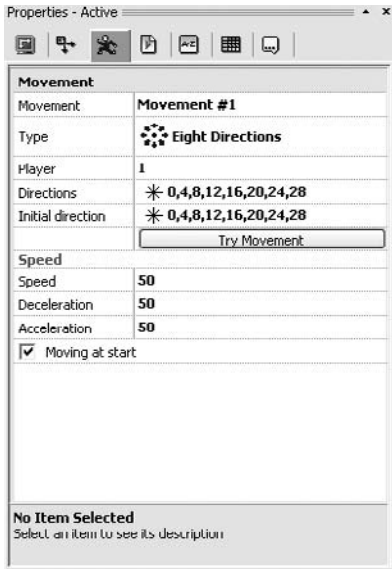


FIGURE 14.32 The Movement Direction Properties sheet.

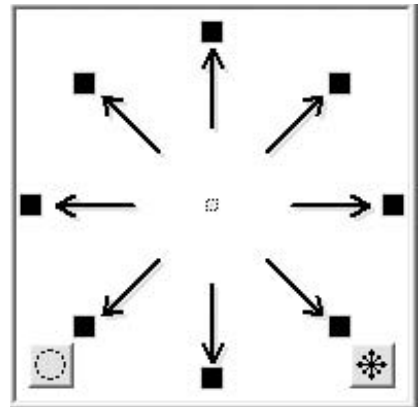


FIGURE 14.33 Direction and Initial Direction have the same dialog.

To select or deselect a direction, click the relevant box (either Direction or Initial Direction). Having an arrow pointing to that box shows its possible directions. In Figure 14.33, the object could move in eight directions. You can click on the button in the bottom-left side of the Direction dialog to remove all directions so you can place one or more directions manually. If you click on the icon on the bottom-right side of the Direction dialog, it selects all directions.

Race Car

Figure 14.34 shows the icon for the Race Car movement, and Figure 14.35 shows its Properties sheet. This movement type simulates a bird's-eye view of a car's movement. There are controls for steering, braking, and accelerating, which users can activate by pressing a key or using a joystick. You can see the keys that can be used for this movement in Table 14.1.

Table 14.1 Keys Used for Race Car Movement

ACTION	KEYBOARD
Accelerate	Up arrow
Brake	Down arrow
Turn left	Left arrow
Turn right	Right arrow



FIGURE 14.34 The Race Car icon.



FIGURE 14.35 The Race Car Properties sheet.

In addition to the speed, acceleration, and deceleration settings are three more options:

Enable Reverse. This option gives your object the ability to go backward. With it turned off, the object can only move forward.

of Angles. This allows you to decide how many different directions the object can move. Selecting four only gives you left, right, up, and down; selecting 32 gives you the smoothest possible direction changes.



You can easily create all the different animation tracks needed for each direction by using the animation tool available in the Picture Editor. We will discuss this in the next chapter.

Rotating Speed. Rotating Speed sets the rate at which the object turns. A high value allows tight corners to be turned, while a low value reduces the cornering ability.

Platform Movement

This movement icon in Figure 14.36 and its Properties sheet in Figure 14.37 are used to define platform-game-type movement. This means characters walk along a set of platforms and climb ladders or jump between floors, viewed from the side, as in games such as *Commander Keen* and *Zeb*. Movement is controlled by the cursor keys or the joystick. In addition to the usual acceleration, deceleration, and speed controls are a number of controls for jumping. You can make platforms and ladders out of backdrop objects.



FIGURE 14.36 The Platform Movement icon.



FIGURE 14.37 The Platform Movement Properties sheet.



You must still test for a collision with a backdrop platform object; otherwise, your active object will fall through the platform as if it weren't there. You can do this through the Event Editor.

Initial Direction. This option allows you to choose one or more directions for your object to move when the game begins. If you choose more than one direction, TGF2 will choose one of the specified directions at random.

Try Movement. You can test out the movement on screen, without the need to leave the movement editing screen.

Speed. This option sets the maximum speed at which your object can move.

Acceleration. Acceleration sets the rate at which your object speeds up to its maximum speed.

Deceleration. Deceleration sets the rate at which your character object slows down.

Moving at Start. When check marked, the Moving at Start option causes the object to move in one of the directions you have chosen when the game starts. When this option is unchecked, the object remains stationary until another object collides with it.

Gravity. This option selects the effect of gravity. A high setting makes your object fall rapidly, allowing only short jumps.

Strength. Jump Strength selects the jumping power of your character. Changing the gravity also affects this parameter.

Jump Controls. Jump Controls are used to change the control system for jumps, as follows.

No Jump. This option turns jumping off for an object.

Up Left/Right Arrow. This option makes the object jump when the up arrow key is pressed at the same time as either the left arrow or right arrow key.

Button 1. Button 1 uses fire button one, or the Shift key, to control the jump.

Button 2. Button 2 uses the second fire button, or the Control key, to activate a jump.

MULTIPLE MOVEMENTS

In all of the movement types you may have noticed that the very first option in the Properties sheets, which we didn't cover, is just displayed as Movement #1. This is the default first movement assigned to that movement type. In TGF2 you can assign multiple movement types to a single object, so you could create three movements for your character, the first being Platform, the second being Race Car, and the third being Mouse Controlled. Of course, this is unlikely in most cases, but when making games it does give you a lot of flexibility to control the objects in your game. An example of this might be that you create a side scrolling game in which the player controls the character using the Platform Movement. Perhaps at a certain stage you switch off the Platform Movement and assign a Path Movement to move the player's character to a specific position to allow for your game to tell a story. By doing this

you can prevent the player from moving the character, and you can tell this part of the story. Perhaps in this case another computer-controlled player appears and says something to the game player's character. You can then switch back to the Platform Movement and allow the player to continue with the game.

To access the multiple movements, click on Movement #1 and you will see a + - button. Click on this to display the movement dialog as shown in Figure 14.38.

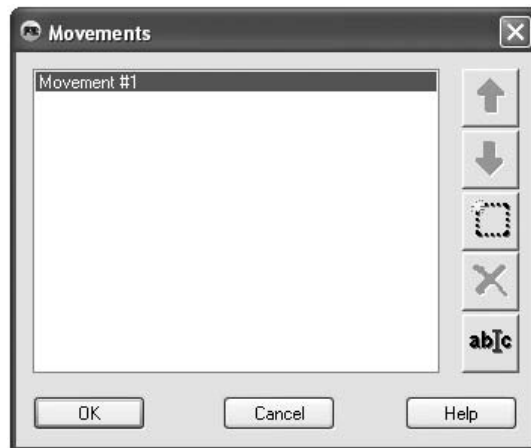


FIGURE 14.38 The multiple movement dialog.

From here you can create new movements and go back into the properties of the object and select the movement number and change the movement type. You can also rename the movement to something more appropriate.

CHAPTER SUMMARY

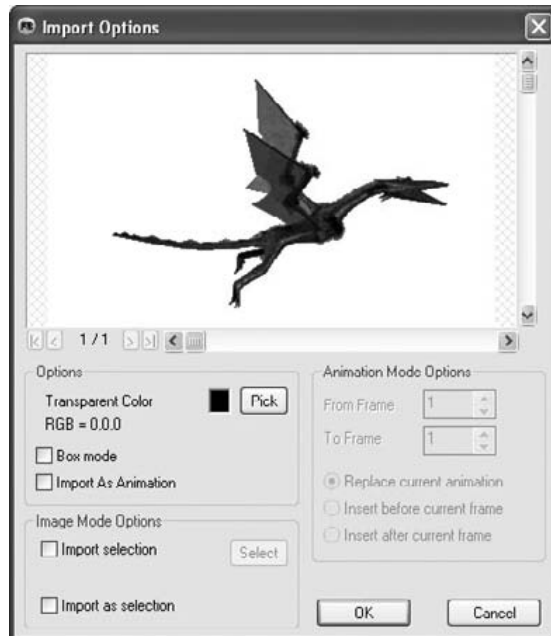
In this chapter, we looked at the basis for all TGF2 games, the active objects, backdrops, text, hi-score, lives, and score objects. These are the most common active objects in TGF2 and the ones you will be working with the most. In the next chapter, we will look at asset creation using the Picture Editor. These tools will round out your ability to make your own games and productions with TGF2.

This page intentionally left blank

WORKING WITH PICTURES AND ANIMATIONS IN TGF2

In This Chapter

- The Picture Editor
- The Animation Tool



In our final dealings with TGF2, we will look at how to create and manipulate assets for your games. Some of the most useful tools for the game developer that come with TGF2 are in the Picture Editor. This editor makes it easy to import and deal with your game assets. They include animation functions that previously required you to work manually in another application such as Photoshop, including copying, rotating, and other tedious operations.

THE PICTURE EDITOR

The Picture Editor lets you create your own animation, background objects, icons, and quick backdrop objects. Because many of the features are identical for all these types of objects, they are summarized in this chapter.

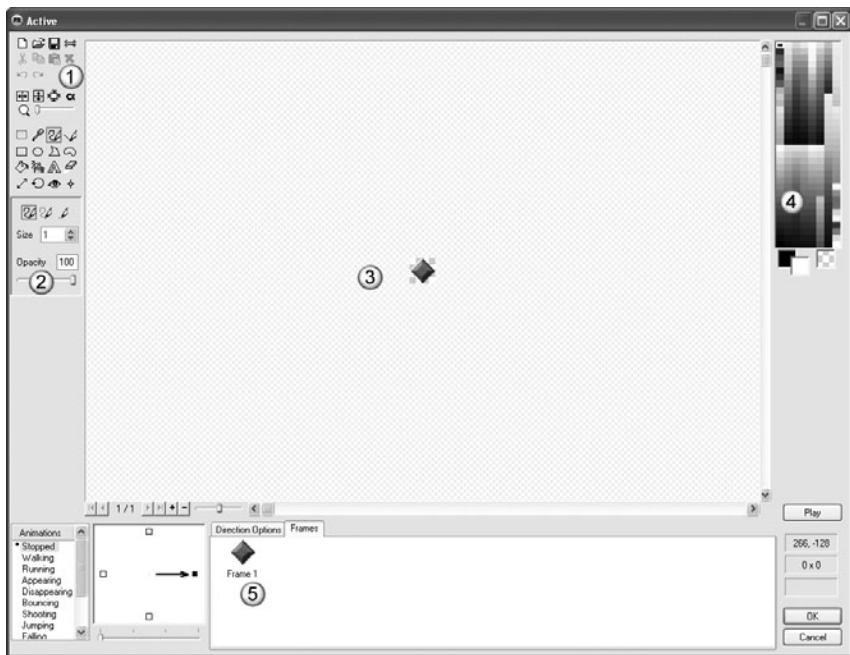


FIGURE 15.1 The TGF2 Picture Editor.

You can see the Picture Editor in Figure 15.1, as well as the other components that make up the editor, which include:

- Tools
- Tool Properties sheet
- Drawing Area
- Color Palette
- Animation Editor

Tools

The drawing tools are located at the upper left of the Picture Editor window. They include the most commonly used features of the digital artist in other paint packages. We will now look at the various tools that make up the Picture Editor's toolbox, starting from the top left and working to the right.

Clear

This option clears the image window so you can start from scratch. If you accidentally clear work that you wanted to save, you can undo the Clear command by clicking on the Cancel button and re-editing the image or using the Hot Key combination Ctrl+Z. You can see the icon for the clear option in Figure 15.2.

Import

This tool allows you to load an image from disk; the associated icon is shown in Figure 15.3. As the picture and animation aspects work hand in hand, you can import multiple pictures at a time to save time. The import supports

- PNG
- JPEG
- GIF
- FLC
- BMP
- PCX

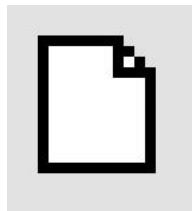


FIGURE 15.2 The Clear Image option.



FIGURE 15.3 The Import option icon.

To use the import option you need to select the file you want to import (if you want to import multiple files, select the first one of its type). An Import Options dialog box appears, providing you with different options to configure your import selection, as shown in Figure 15.4.

Some of the import features are discussed in a bit more detail later in this chapter.

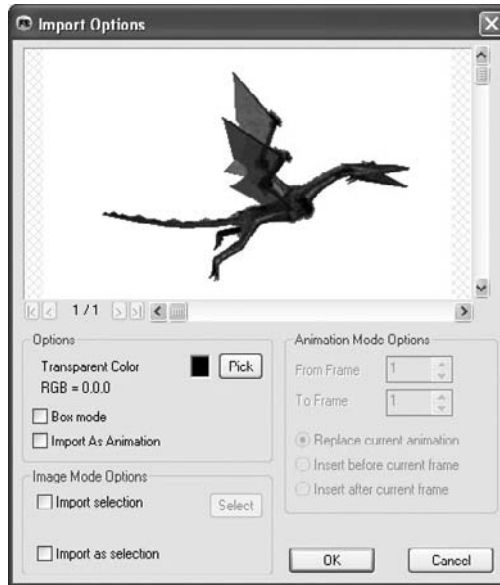


FIGURE 15.4 The Import Option dialog.

Export

Export allows you to save any file you have been working on into a standalone file. When you create a picture from scratch in TGF2's Picture Editor, you can only access it through TGF2. By using the Export option you can save the file and its animations to an external file(s) that can then be accessible by another paint package. The Export option allows you to save the files as PNG, BMP, or JPG files. The Export icon can be seen in Figure 15.5.

When you click on the export button you will be given a simple dialog to select if you want to export a single file or an animation with a number of frames, as shown in Figure 15.6.



FIGURE 15.5 The Export option icon.

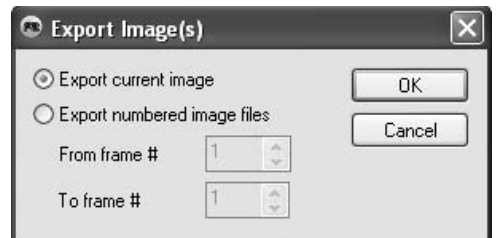


FIGURE 15.6 The Export Image(s) dialog.

Options

Options allow you to configure how the right mouse button is utilized in the Picture Editor. It is common for the right mouse button to be assigned to a second color so you can use two colors at one time without needing to swap between them. This is particularly helpful when you are doing fine art and might be using two similar colors at one time. You can also configure the right mouse button to select the color of the pixel where the mouse cursor is situated. You can also configure the background display where there is no image, which by default is gray and white. The icon for the Options can be seen in Figure 15.7, and the Options dialog box can be seen in Figure 15.8.

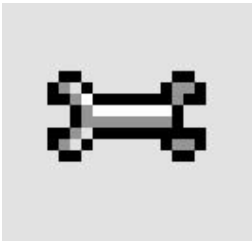


FIGURE 15.7 The Options icon.



FIGURE 15.8 The Options dialog.

Cut, Copy, Paste, and Delete

Once you have selected a block on the canvas, you can cut and copy it using the following commands from various buttons, as shown in Figure 15.9.



FIGURE 15.9 The cut, copy, paste, and delete icons.

Cut. The original area is cut out and replaced with a block of whatever the transparency color currently is. A copy of that area is placed in the Windows clipboard.

Copy. The area is copied to the Windows clipboard. The original area remains unchanged.

Paste. Pastes the contents of the Windows clipboard. After you have cut an area, you can copy it onto the image using this command. The pasted area is in a rectangular block. Drag this block where you want it and then click on it to fix it in place.

Delete. Deletes the contents of the selected area.

Undo and Redo

The Undo tool lets you undo the last step you performed. This is very handy if you make a mistake. If you change your mind when you have undone something, you can click the Redo icon to change it back. You can see the Undo and Redo icons in Figure 15.10.



FIGURE 15.10 The Undo and Redo icons.

Flip Horizontally

This tool reverses your image from left to right, just like a mirror.

Flip Vertically

This tool turns the whole image upside down. The icons for both images can be seen in Figure 15.11.

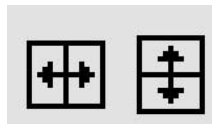


FIGURE 15.11 The Flip Vertically and Flip Horizontally icons.

Crop

Crop removes any available blank space on the canvas. If you have a large canvas, and only a small part of it has any image on it, the Picture Editor tries to remove as much of the blank space as it can using straight lines. The Crop icon can be seen in Figure 15.12.

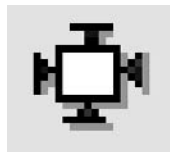


FIGURE 15.12 The Crop icon.

Transparency

Transparency allows you to display the transparent color but also allows you to change it quickly by clicking on the color palette. You can then hide the transparent color again by unticking the box. The Transparency button and its properties box can be seen in Figures 15.13 and 15.14.

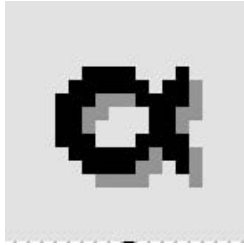


FIGURE 15.13 The Transparency icon.



FIGURE 15.14 The Transparency properties box.

Zoom Control

The Zoom Control, as shown in Figure 15.15, allows you to zoom in closer or zoom out of the current image. If it is to the left, it is zoomed out to the maximum canvas size. The farther it is to the right, the higher the zoom magnification is.



FIGURE 15.15 The Zoom Control.

The Selection Tool

This tool lets you define a rectangular block, which you can then cut or copy. To choose a block, move the mouse to where you want the top-left corner of your block to be and then drag a box down around the area you want. If you make a mistake, click once on another part of your image and try again. You can see the Selection tool in Figure 15.16.

Color Picker

The Color Picker allows you to click anywhere on the canvas and change the current pen color to the selected color. This is very useful when you want to pick a pen color that is already in use on the canvas rather than trying to guess exactly what the color is. You can see the Color Picker icon in Figure 15.17.

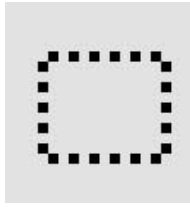


FIGURE 15.16 The Selection tool icon.



FIGURE 15.17 The Color Picker icon.

The Pen Tool

The Pen tool lets you either draw one pixel at a time or draw a freehand line by holding down the left mouse button and dragging over the canvas. You can see the Pen icon in Figure 15.18.

The Line Tool

The Line tool lets you draw perfect straight lines. Click the point where you want the line to start and hold down the mouse button. As you move the mouse, it “drags” a line behind it. When you reach the place where you want the line to end, let go of the mouse button, and you will have drawn a line between the two points. You can see the Line tool icon in Figure 15.19.



FIGURE 15.18 The Pen tool icon.



FIGURE 15.19 The Line tool icon.

The Rectangle and Filled Rectangle Tools

These tools and their properties boxes, let you draw rectangles and squares more easily than by trying to construct them out of four separate lines. After selecting the icon you want, place the pointer where you want the top-left corner, press and hold down the mouse button, and then drag the rectangle to the shape you want. The Rectangle tool produces an unfilled (clear) rectangle. You can select one of the alternative options in the properties box to either produce a solid rectangle or a solid rectangle with a different border color. You can see the Rectangle icon and its properties in Figures 15.20 and 15.21.

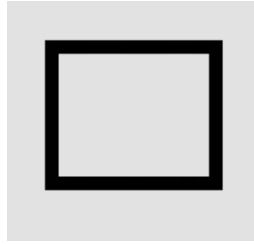


FIGURE 15.20 The Rectangle tool icon.

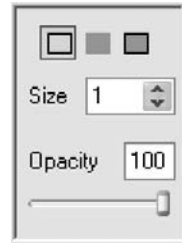


FIGURE 15.21 The Rectangle tool properties.

The Ellipse and Filled Ellipse Tools

These tools and their properties sheets let you create ellipses and circles, both filled and empty. Select the place where you want to start the top-left side of the ellipse. Holding down the mouse button, move the pointer away from place you first began the ellipse. You can see the Ellipse icon and its properties sheet in Figures 15.22 and 15.23.

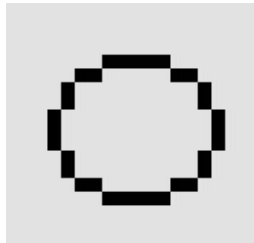


FIGURE 15.22 The Ellipse button icon case.

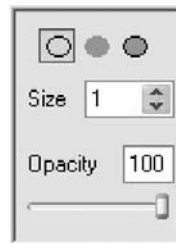


FIGURE 15.23 The Ellipse properties window.

Polygon Tool

It is very simple to create a polygon shape using this tool. You can draw a number of lines that connect to each other. You can see the Polygon tool icon in Figure 15.24 and its properties sheet in Figure 15.25.

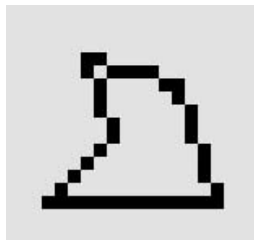


FIGURE 15.24 The Polygon button icon.



FIGURE 15.25 The properties sheet for the Polygon tool.

Shape Tool

The Shape tool allows you to draw a shape, and it will completely enclose it, so if you begin to draw a circle but take your finger off the left mouse button while drawing it, the program completely closes the shape. This allows you to fill the item with a color if required. You can see the Shape tool icon in Figure 15.26 and its properties sheet in Figure 15.27.

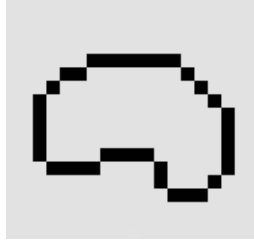


FIGURE 15.26 The Shape tool icon.

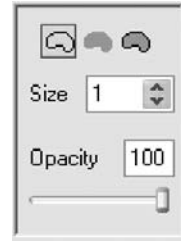


FIGURE 15.27 The Shape tool properties sheet.

The Fill Tool

The Fill tool fills an area on the canvas with a solid block of color. The area to be filled should be completely enclosed. If there is a gap of even one pixel, the color will “leak” out into other areas of your frame. In Figures 15.28 and 15.29 you can see the Fill icon and its properties sheet.

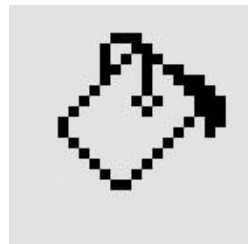


FIGURE 15.28 The Fill tool icon.



FIGURE 15.29 The Fill tool properties sheet.

The Spray Tool

This works the same way a spray can works. It applies a spray of paint onto the canvas. You can change the size of the paint pixel and the pressure that’s applied. You can see the icon for the Spray tool in Figure 15.30 and its properties sheet in Figure 15.31.



FIGURE 15.30 The Spray tool icon.



FIGURE 15.31 The Spray tool properties sheet.

The Text Tool

This allows you to place a piece of text, either a single letter or words, on the canvas. You can apply basic formatting to the text including bold, italic, and underline. It is also possible to select a specific font for the text. You can see the icon for the Text tool in Figure 15.32 and its properties sheet in Figure 15.33.

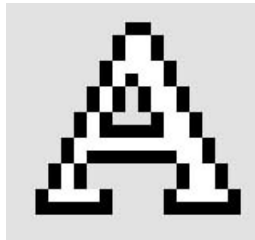


FIGURE 15.32 The Text tool icon.

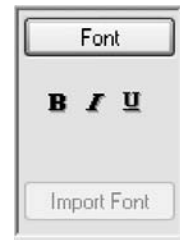


FIGURE 15.33 The Text tool properties sheet.

Eraser Tool

This tool allows you to delete a part of your picture, basically rubbing it out. You can amend the eraser size for when you need to delete a large amount of the image or area on the canvas or you can make the eraser very small for more precise deletions. You can see the Eraser tool icon in Figure 15.34.

Size

This allows you to change the size of the image on the canvas. Its icon can be seen in Figure 15.35. There are also three additional options to stretch the image, to resample it, and make it proportional to the canvas size. This properties sheet can be seen in Figure 15.36.

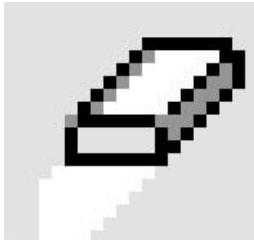


FIGURE 15.34 The Eraser icon.

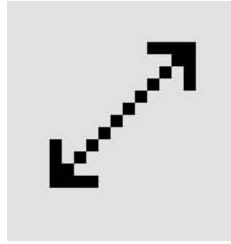


FIGURE 15.35 The Size icon.

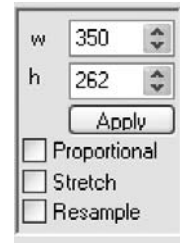


FIGURE 15.36 The Size properties sheet.

Rotate

This tool lets you rotate the whole image with fine control. When you select this function, you can enter a specific angle by which to rotate the image on the property sheet. Once you have clicked OK, the image is turned by the angle you specified. The icon for the Rotate option and its properties can be seen in Figures 15.37 and 15.38.

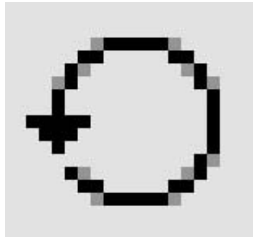


FIGURE 15.37 The Rotate icon.

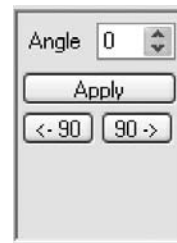


FIGURE 15.38 The Rotate properties sheet.

View Hot Spot

The Hot Spot is an invisible handle, or anchor, that you can use to drag images around on the screen. It is also used as a reference for an object's X, Y coordinates. Each image can have its own separate Hot Spot. As a default, when you create a new active object, the Hot Spot is automatically positioned at the top-left corner of the image, but you can move it anywhere you like.

You can view the Hot Spot by selecting the View Hot Spot icon. Try to position it centrally if your object is going to have several different directions; otherwise, it will "jump" when you change direction. You can see the Hot Spot icon and its properties sheet in Figures 15.39 and 15.40.

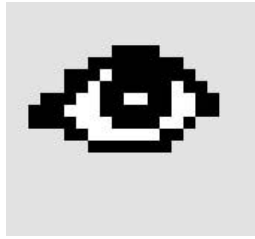


FIGURE 15.39 The Hot Spot icon.

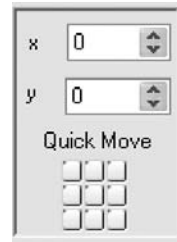


FIGURE 15.40 The Hot Spot properties sheet.

View Action Point

The Action Point is the point where things like bullets are fired from objects. For example, if you had a large spaceship with a gun mounted on it, you would set the Action Point to the end of the gun barrel, where the bullet would first appear. You can show the Action Point by clicking on the View Action Point button. You can see the Action Point icon in Figure 15.41. Its properties sheet looks exactly like the Hot Spot properties sheet in Figure 15.40.

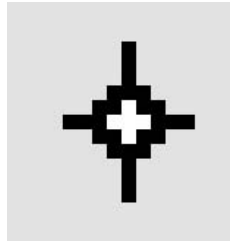


FIGURE 15.41 The Action Point icon.

Drawing Area

There's not much to say about the drawing area. This is the place where you draw your images. If your image is too large for the window, horizontal and vertical scroll bars allow you to move around.

The Color Palette

This is where you select the color you want to draw or fill the canvas with. Figure 15.42 shows a selection of colors to choose from when drawing. The two boxes on the bottom left are the current draw colors for the left and right mouse buttons. The box on the bottom right is the currently selected transparent color.

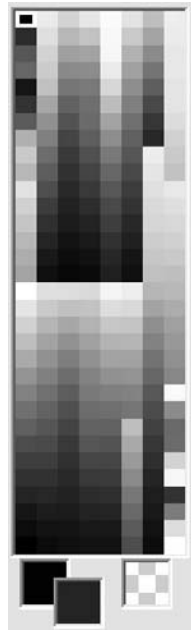


FIGURE 15.42 The color palette.

THE ANIMATION TOOL

Animation is a word that still strikes fear in the hearts of many who want to develop games, but TGF2 makes animation a lot easier with its animation tools built into the Picture Editor. When using the Picture Editor and its animation tools:

- You can create your own active object, draw or import images, and animate it.
- You can use an already created active object, which already contains animations, and then change or update it.

To show you some of the animation features, you will be using a TGF2 file that has an active object that already exists on the frame, and analyzing how the tools have been used to animate it. As you do this, you will see the steps required to create your own animated object.



1. You need to have TGF2 loaded with no file currently loaded.
2. Click File | Open and browse the DVD provided with this book and locate the file called `dragonanimation.mfa`, which is in the `TGFFILES` folder. Then click Open to load it into the program.
3. Double left-click Frame 1 to open the Frame Editor.
4. Notice an active object is already on the playfield of a dragon. Click Run application to see the dragon fly as shown in Figure 15.43.



FIGURE 15.43 Animated active object of a dragon.

5. This active object is moving because it already has its animation applied to it. To see this you need to enter the Picture Editor and view its frames in the animation tool, so double left-click on the dragon picture.
6. You can now see the dragon in the canvas area and all of its animation frames in the animation tool, as shown in Figure 15.44.

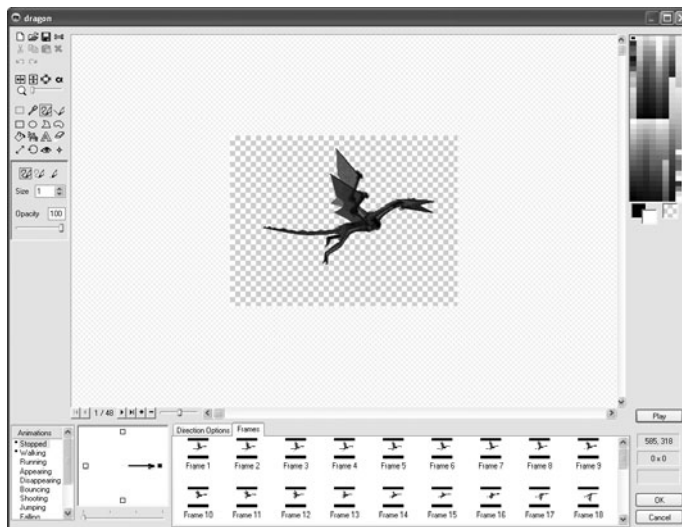


FIGURE 15.44 The animation frames for the dragon.

You can see the different sections of the animation tool as separated in Figure 15.45.

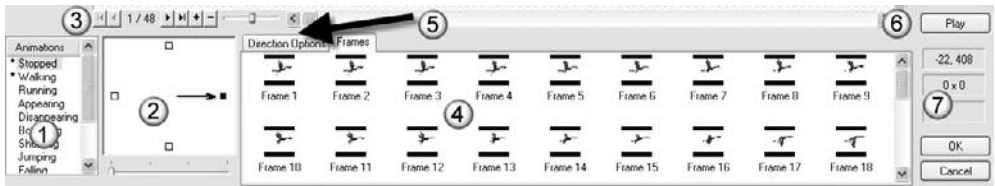


FIGURE 15.45 Various areas of the animation tool.

- 1: These are all the available animations that you can place images against. You can also create your own animation groups.
- 2: Directions is the available directions each animation can have. Initially this is set to four directions, but you can use the slider underneath it to increase it to 32 directions.
- 3: This is the Frame Tools bar, where you can add frames and move through the available frames you have created. This updates the current frame being displayed on the canvas.
- 4: These are all of the frames of your animation for this particular animation group.
- 5: The Direction tab contains specific information relating to the speed of the animation and its loops.
- 6: This plays the current animation to give you an idea of what it looks like.
- 7: Information about the cursor position on the canvas and the colors in use at that particular pixel.



To select all the frames of an animation, press Alt+A once you have single left-clicked in the animation frame window. This lets you move or delete a whole sequence of animations at one time. You can select multiple frames by holding down the Ctrl key while you click on frames, or hold down the Shift key to select all frames between two selected frames.

Directions Tab

Figure 15.46 shows the controls for the speed of the animation (the Lower and Higher speed boxes), as well as the number of times it will repeat itself before it stops (the Repeat box). You can select Looping by ticking the Loop box. This makes the animation sequence repeat over and over.

You can also change which frame number the animation loops back to in the Back To box. This is useful in a long animation if you only want to repeat certain

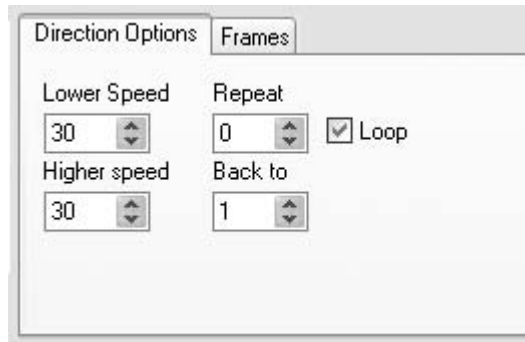


FIGURE 15.46 The Directions tab options.

parts. Say you are working with an animation of a man getting up from a crouched position and then running away. You may only want to play the first couple of frames of him being crouched down, then loop the animation back only to the running sequence as the man continues running.



An animation can have either one or two speeds assigned to it. An object can be animated but be static with regard to its location on screen, or it can be animated and move around the screen. The difference is that the second one also has a movement speed assigned to it as well as its animated speed.

Lower Speed

The Lower Speed box in the Direction Options tab controls the speed when the character is not moving around the screen. Setting this to zero halts the animation. Setting it higher has the animation running all of the time. Of course, it may look unrealistic if your character is running frantically without moving.

Higher Speed

The Higher Speed box in the Direction Options tab controls the maximum rate of the animation when the character is moving around the screen. Note that the rate of animation is proportional to the speed of the character, in between the Lower and Higher settings. To create a realistic running action, you may need to change the Higher setting to a value that matches the character's speed across the screen.

For example, if you were to set a character's movement speed high and the animation speed low, it would appear as though the character was being dragged across the screen. If you had the animation speed high and the movement speed low, it would look as though the character were trying to run fast on an icy floor.

Animation Direction

This is a very useful feature that can transform your single-direction animation into a character that moves to the left, right, up, down, and so on. Take a look at the Direction box, as shown in Figure 15.47, where two images show the left and right direction of our flying dragon. By clicking on the different direction squares in the Direction box, you can create a different animation for each direction the character can move.

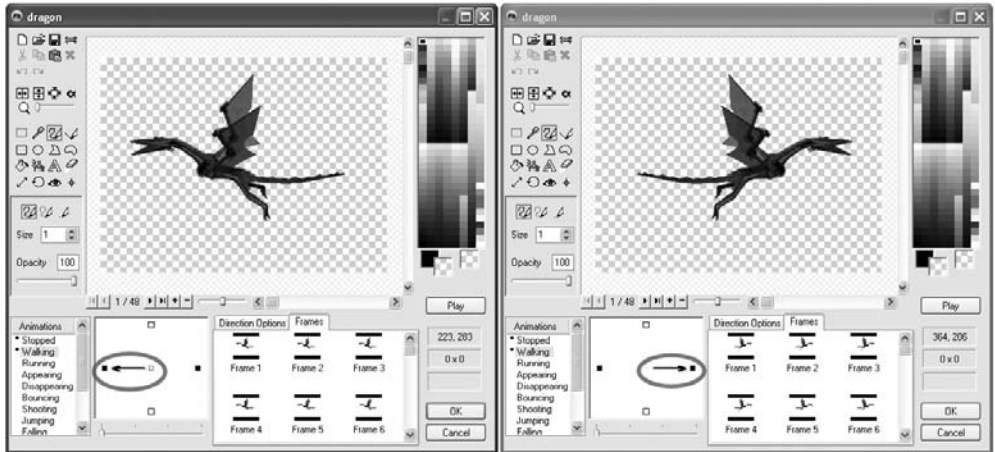


FIGURE 15.47 The difference between the left and right direction arrows.

If you take a look at Figure 15.47 you can see that on the clock face, the 3 o'clock position (rightward motion) and the 9 o'clock position (leftward motion) have small solid black squares with arrows pointing to them. These indicate that an animation is assigned to those positions, so the dragon has a different animation for going left and right.

Counting all possible directions, you can have up to 32 animated sequences for the walk direction of a character. This can help things look smooth, but it is overkill for most purposes. As computers have become more powerful, selecting 32 directions wouldn't cause too much of a resource issue. In a large game with many directions and animations it would start to add up, and, although in most cases this wouldn't slow your machine down, it's good to get into good habits when creating your games. Why waste resources if you don't need to. In the case of the flying dragon you only need it to move left and right, so there is no need to select 32 directions when you are only going to use 2.



An easy way to create several different directions from one animation is to copy the animation to the new direction and then click on the rotate buttons to ensure they are pointing the correct way.

Animation List

The default animation list can be seen in Figure 15.48. This list details the basic set of animations provided with TGF2 that can have animations placed within them. These animations can be referenced from within the Event Editor but also work automatically when using certain movement types on that particular object. The animation sets are:

- Stopped.** When the object is not moving.
- Walking.** Checks the speed of an object and runs this animation if it is moving at a slow pace.
- Running.** Checks the speed of an object and runs this animation if it is moving at a fast pace.
- Appearing.** Runs the animation as soon as the object is created.
- Disappearing.** Runs the animation as soon as the object is destroyed.
- Bouncing.** Plays the animation when the object is bouncing on another object that is defined as an obstacle (something that it will bounce or hit).
- Shooting.** The animation is triggered when a shoot object is triggered.
- Jumping.** The animation is played when the object is jumping. Jumping is used in platform movement games.
- Falling.** When an object is falling, the animation is played. This animation is used primarily in platform games.
- Climbing.** When using the platform movement, this animation plays when the object is climbing a ladder.
- Crouch Down.** If you are using platform movement, this animation set runs when the object is crouching.
- Stand up.** When the object is standing up (not crouching), this animation runs. This is also used in platform movement for platform games.



FIGURE 15.47 Need Caption

You can create your own animation sets for anything that is not covered in this list. These are called user-defined animations. To create your own, right-click on the animation list and select New. You can then enter the name of the animation set.

CHAPTER SUMMARY

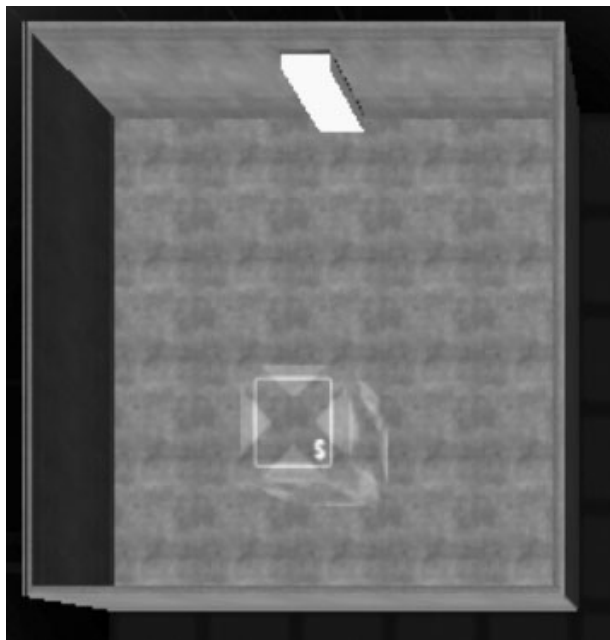
In this chapter we have taken a look at how to use and create your own game graphic assets. To do this you can use another paint package and then import images into TGF2 or you can draw them inside the program's Picture Editor. Either way, you should now have enough knowledge to be able to attempt it in TGF2. You should now be able to begin creating and designing your own 2D games in TGF2. If you need any help you can consult the help file in the Help | Contents menu.

Now that you have had the chance to create some awesome games in a 2D world, let's move on to a package that will allow you to create 3D first-person shooters.

INTRODUCTION TO FPS CREATOR

In This Chapter

- Introduction
- Installation Walkthrough
- FPS Creator Terminology
- FPS Creator Creation Process
- FPS Creator Walkthrough



First Person Shooter (FPS) Creator is a fast and easy way for anyone to create his own FPS game. Without a program like FPS you would need to have access to a 3D game engine, some form of scripting or C++ knowledge, and create or purchase 3D models. FPS Creator takes away many of these headaches by giving you the engine, a world creator, and some 3D models to get you started. Very quickly you will find that you have made a small game and will want to create bigger games in no time. This chapter will look at the installation requirements for the program to ensure you have the right level of equipment to run the program. It will also go through the installation process and a walkthrough of the basic interface to give you a good grounding in the product before you begin to create your first game.

INTRODUCTION

FPS Creator has taken away many of the headaches a game creator might worry about. The developers have created a game engine with all the necessary parts already programmed, and you as the game creator only need to concentrate on putting the components together to make your creation.

Some of the parts that are already present in the program are:

3D World Editor. Using simple building blocks, you can place your rooms and objects within a 3D-built map editor.

One Click Build Export. Using a single mouse click, you can create an executable file that can be distributed to other computers running the Windows operating system.

Direct X9.0 Support. The 3D engine supports Microsoft Direct X9.0. This means the speed and certain features of Direct X9.0 are incorporated into the engine (or can be added at a later stage).

Import. You have the ability to import your own sounds, 3D models, and textures to create your own games how you visualize them.

Included Objects. Hundreds of rooms, guns, characters, and room items are included with the full version. This allows the game creator who does not have any 3D skills to make a game without worrying about the graphic aspect.

Physics Engine. A difficult area for any game creator to include in their games is physics. Physics makes a game more realistic. For example, if you shoot or knock some items off a table, the physics engine handles how they fall and roll in a realistic manner.

Internet and Multiplayer Support. All the programming to connect online and play with other people has already been done, so you can make your game support online play straight out of the box.

Scripting. You can change aspects of your game by using the built-in scripting language. This is used when you want to change or replace the default running of the game, for example, the enemy intelligence (how it reacts when the player is near them or shoots at them).

Many other things have already been done in the program to make it easier for developers to be able to just get on with creating their visions, and you will see some of these aspects as you begin to build games with the FPS Creator program.

System Requirements

The FPS Creator program has a set of minimum system requirements. These can be seen in Table 16.1, and recommended requirements are listed in Table 16.2. It is important to take into account that the bigger the game you create, the longer it might take to create the final output. The number of elements that you include will affect the overall performance and running speed of the game.

Table 16.1 Minimum System Requirements

REQUIREMENTS

OS: Windows 2000 or Windows XP

Processor: Pentium 3 – 1 GHz

Memory: 256 MB RAM

Graphics Card: Direct X9.0c compatible with 64 MB RAM

Hard Disk: 1.4 GB

Other: Printer if you want to print any screens or documentation

Table 16.2 Recommended System Requirements

REQUIREMENTS

OS: Windows XP Home or Pro

Processor: Pentium 4 – 2.66GHz

Memory: 1 GB RAM

Graphics Card: Direct X9.0c compatible with 128 MB RAM

Hard Disk: 1.4 GB

Other: Printer if you want to print any screens or documentation

INSTALLATION WALKTHROUGH



The trial demo of FPS Creator can be found in the Demos folder on the DVD provided with this book. The demo is time limited to 30 days, so only install the product when you are ready to proceed with this part of the book.



The trial version cannot build stand-alone executable files and does not contain all the media and maps. To get access to this additional content you need to purchase the full version from www.fpscreator.com.



1. Double left-click on the file FPSCreatorDemo.exe located in the Demos folder on the DVD.
2. If you get a Warning dialog box, click Run and the files will begin to be extracted.
3. The Welcome dialog box will appear as shown in Figure 16.1.
4. Click Next on the Welcome dialog box.
5. You will now be presented with the License Agreement dialog box as shown in Figure 16.2.

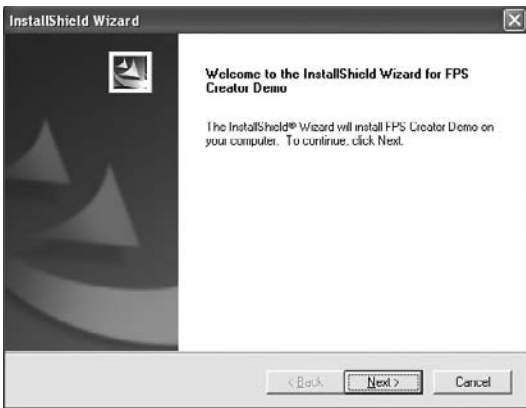


FIGURE 16.1 The Welcome dialog box appears.

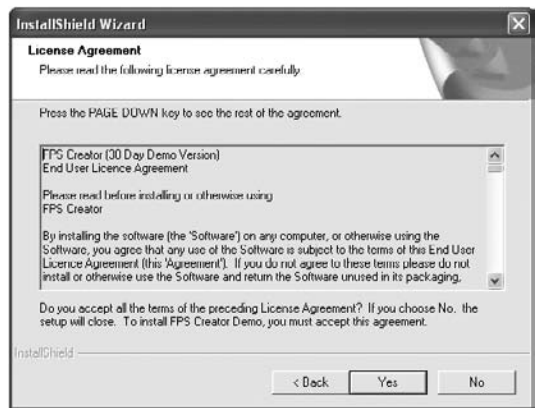


FIGURE 16.2 License Agreement dialog box.

6. Click Yes to accept the license agreement, and then you will be presented with the Choose Destination Location dialog as shown in Figure 16.3.
7. The default destination path is C:\Program Files\The Game Creators\FPS Creator Demo. If you want to use the default path, click Next. To select a location, click Browse.
8. The files will then begin to be copied to your computer, as shown in Figure 16.4.
9. Once all files have been installed on your computer, you will see the final dialog box, shown in Figure 16.5.
10. Click Finish to complete the installation.

An FPS Creator icon will be placed on your desktop, which can be seen in Figure 16.6, and it will also be installed in the Start | All Programs | The Game Creators folder option.

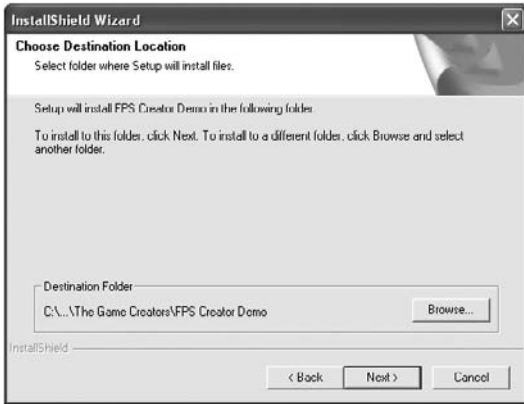


FIGURE 16.3 Choose the destination for installing the files.

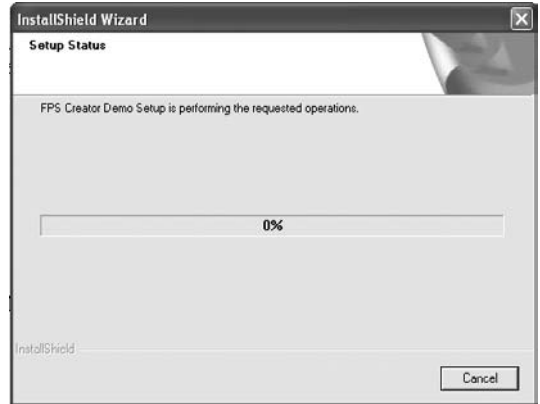


FIGURE 16.4 Files being installed onto the computer.

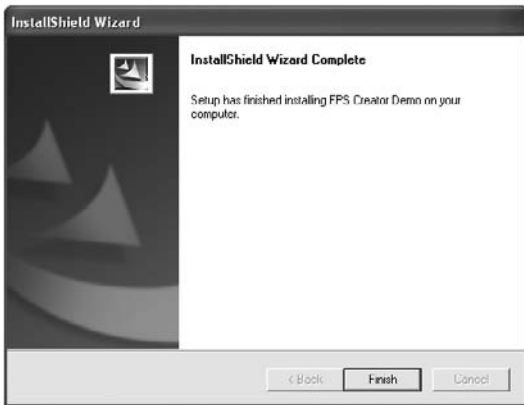


FIGURE 16.5 The final installation dialog box.



FIGURE 16.6 The FPS Creator desktop icon.

FPS CREATOR TERMINOLOGY

Before discussing specific aspects of the FPS Creator program, it is worth introducing some of the terminology used in this part of the book that relates to the program.

Prefabs. Prefab is short for prefabricated—something that has already been created. In this case it is a room that has already been designed, with the entry or exit doors already added.

Segments. Segments are parts of rooms or environments that you can use to create your own areas in your game, for example, corridors, stairs, floors, and wall segments.

Entities. Entities could also be called objects, as they are the physical objects that are in your rooms and environments. These entities can be enemy players, equipment, extra life, extra health, and scenery.

Markers. Markers covers a number of different items including light effects, the starting position of the player, special action zones where something happens when the player moves into them, and checkpoints that can save the player's position or be used to end the level.

FPS CREATOR CREATION PROCESS

The game creation process is quite simple in FPS Creator, and some parts can be skipped or have more time spent on them depending on the type of game you are creating. The demo version of the product comes with media to make either a sci-fi game or a World War 2 game, so you are restricted to these types of stories and content. The game creation process for FPS Creator can be described as follows:

Design. Design your levels or maps. These could be first created on paper and then within the 3D-world editor that FPS Creator provides, or you could skip the paper design and lay out your maps straight in the editor. These rooms and maps require you to create connecting doors, windows, and ceiling tiles. You also need to add any door switches to allow the doors to be opened.

Place additional design aspects. Place the lighting for your room and any specific trigger points. A trigger point is something that happens when the player character walks over it, for example, add health, cause damage, or end the level.

Place your room items. Once you have created your rooms, corridors, and other locations, you need to place the contents of the world to make it more realistic. For example, if you have created a sci-fi laboratory, you need to fill it with tables, chairs, test tubes, and other items.

Place your player items. Place items such as weapons, health, and bullets to aid the player's progress through the game.

Place enemies. You want to have some enemy characters in your game to make it more interesting and more challenging for the player. You can assign where the enemy characters will move.

Test your game. Once everything is in place, it is a good idea to check that it works as you expected it to. You may have to make changes at this stage or add items if you feel it is not playing as you expected it to.

Compile. With the trial version you can test the levels, but you cannot compile them into a file that can be distributed to other computers that are not running the demo. If you own the full version, you can compile the file into an executable that you can give to your friends.

This creation list doesn't have to be completed in the order detailed here, but it provides insight into the easiest order for creating your games. You might go through these steps for each room or for the whole game, though it is easier to design a few rooms at a time if you are doing it directly in the 3D editor. If you are creating on paper, you can design much of the game before putting it into the program.

FPS CREATOR WALKTHROUGH

Double left-click on the FPS Creator icon on the desktop (shown in Figure 16.6) to start the program. At this stage you will either get a loading screen, or if it has been a few days since you installed the product, you may get a Days Left dialog box. This dialog, which is shown in Figure 16.7, tells you how many days you have left until the trial version no longer works on your computer. The dialog box gives you the following information and options:

- How many days you have left of your trial
- The option to continue using the product in the trial mode
- The option to click Purchase FPSC Now to take you to the program creator's Website.

For now, you want to continue using the product, so if you get this dialog box, click Continue Trial. You will then be presented with the FPS Creator window as shown in Figure 16.8.

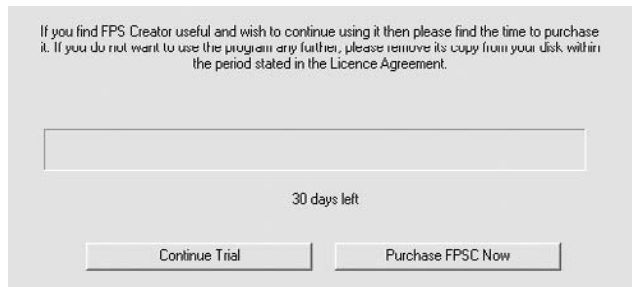


FIGURE 16.7 The Days Left dialog box in the trial version.

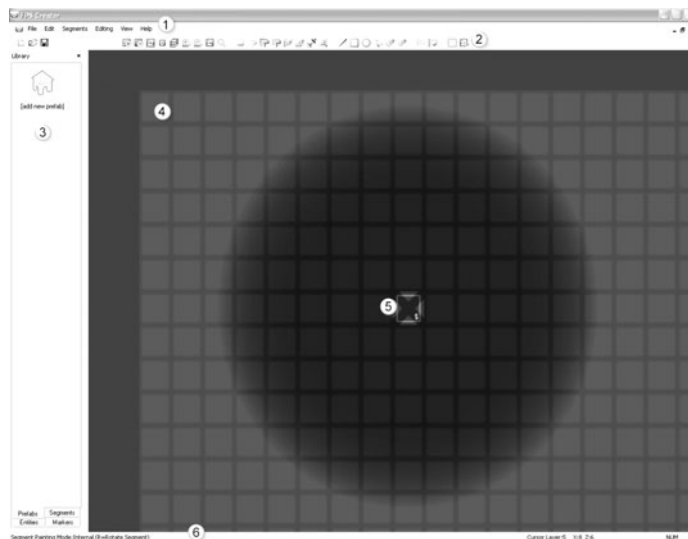


FIGURE 16.8 The FPS Creator program window.

The application has been separated into six distinct areas:

- Drop-down text menus
- Menu buttons
- Library toolbar
- 3D world editor (grid)
- Cursor
- Status bar



The following chapters will go through many of these options in more detail, so do not worry if you do not fully understand them at this point.

Drop-Down Text Menus

You can select various configurations and settings for the application from a number of text menus. You can, for example, save or load a file or apply different options to the 3D world. Only six heading are available from the text menu. These can be seen in Figure 16.9, in which the Segments option has been selected.

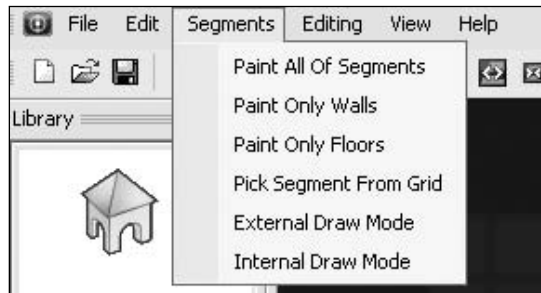


FIGURE 16.9 The Text menu options, with the Segments option selected.

Menu Buttons

When you want to access the main configuration components of the program, use the menu buttons. These menu buttons provide a quick and easy way of getting access to the regular options you will be using the most when making your FPS game. These menu buttons are displayed in groups and can be hidden if required using the Text menu option View | Toolbars.

Standard Toolbar

The Standard toolbar consists of the common saving and loading buttons as well as the options to cut and paste within the 3D editor. The toolbar can be seen in Figure 16.10.



FIGURE 16.10 The Standard toolbar.

From left to right, the options are:

- New Level
- Open Level
- Save Level
- Cut
- Copy
- Erase
- Undo Segment Editing
- Redo Segment Editing

View Toolbar

The View toolbar handles how you see the 3D world and gives you options of zooming in closer, moving to a different layer (see note), or viewing close up. The toolbar can be seen in Figure 16.11.



FIGURE 16.11 The View toolbar.



If you want to make a game in which the player can move upstairs and downstairs, floor levels are called layers in FPS Creator. By default, the program starts on layer 5 to ensure that if you forget about layers and make your game, you still have a number of floors available to you to make your character have the ability to move downstairs (or underground) if needed. If it had started on layer 0, you wouldn't have been able to add any parts of the game lower than this floor, so layer 5 could be considered the ground floor of your game.

From left to right, the buttons are:

- Zoom In
- Zoom Out
- Increase Render Shroud
- Decrease Render Shroud
- Toggle Layers
- Move Up a Layer
- Move Down a Layer
- View Entire Layer
- Close Up View

Segments

The Segments toolbar provides a number of options to paint and color individual segments, walls, and floors. It also has options for entities (characters and items), which will be covered later in this chapter. The Segments toolbar can be seen in Figure 16.12.



FIGURE 16.12 The Segments toolbar.

From left to right, the buttons are:

- Entity Mode
- Segment Mode
- Interior Draw Mode
- Exterior draw mode
- Paint Only Segment Walls
- Paint Only Segment Floor
- Pick Segment
- Select Area

Draw

The Draw toolbar provides access too many options that allow you to draw specific shapes for your segments including lines, rectangles, and freehand. The Draw toolbar can be seen in Figure 16.13.



FIGURE 16.13 The Draw toolbar.

From left to right, the buttons are:

- Draw Segment as Line
- Draw Segment as Rectangle
- Draw Segment as Ellipse
- Spray Segment
- Increase Spray Draw Size
- Decrease Spray Draw Size

Waypoint

In all FPS games you have enemy players who might guard a specific point on the map or move between a number of points. This can be done very easily in FPS Creator using a system called waypoints, whereby you specify the places the player can move on the map. There are only two options to use waypoints from the menu buttons toolbar, these can be seen in Figure 16.14.



FIGURE 16.14 Waypoint toolbar.

From left to right, the buttons are:

- Create New Waypoint
- Waypoint Editing Mode

Test Game/Level

The Test Game/Level menu buttons allow you to test your game and see what the final program will look like. There are two options within the Test Game/Level menu toolbar, which can be seen in Figure 16.15.



FIGURE 16.15 The Test Game menu buttons.

From left to right, the buttons are:

- Test Level
- Quick Level Preview

Library Toolbar

The Library toolbar is where you will access all of the resources, items, and rooms needed to create your 3D world. At the bottom of the toolbar are the four items already discussed: prefabs, segments, entities, and markers. Clicking on any of these items reveals a new blank library screen or a set of objects that can be selected. For example, clicking on Prefab or Entities gives you the option of adding a new entity.

This means adding from the core library to your current game library, which is blank. Markers currently displays all possible options, and segments by default has a ground segment that you can automatically apply. You will be using the Library toolbar later in the next chapter when you create your first simple program.

3D World Editor

The 3D world editor is where you place all of your rooms, room objects, enemy players, and markers to create your game. The editor is made up of an X, Y, and Z grid because it is in three dimensions. The 3D editor looks at first glance like a 2D grid, but you can think of the 3D editor as a big cube, where you have blocks going to the right (X), blocks coming forward (Z) and blocks going down (Y), also known as layers in FPS Creator. The world is made up of a 40x40 grid, with a total of 20 levels (floors) going up and down. You can see the 3D grid in Figure 16.16.

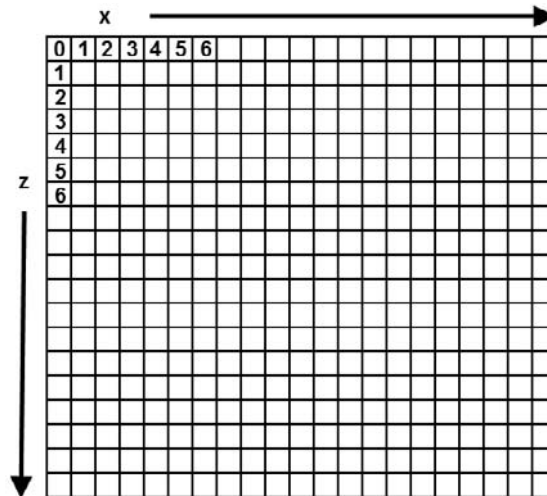


FIGURE 16.16 An example of the editor grid.

When you are viewing the grid, you are actually looking at your world from a bird's eye view, so you are looking down on your rooms from above. If you wanted to place an item at the far left-hand corner of your world, you would specify its coordinates as X being 0 and Z being 0. The Y coordinate would depend on what floor level you are working on, but to begin with it's defaulted at level 5. To work out where you are in the grid, you can see the actual coordinates at the very bottom of the FPS Creator window as shown in Figure 16.17.



FIGURE 16.17 The coordinate location shown in FPS Creator.

Because you are looking down on the world, you must think about where you will place your rooms, corridors, and floors so that you have enough space for them. If you are making a room, you may not want to place it at the very top of the grid, as you will be restricted in the number of directions you can take to any connecting corridors and rooms. An example of a room added to the 3D editor can be seen in Figure 16.18.

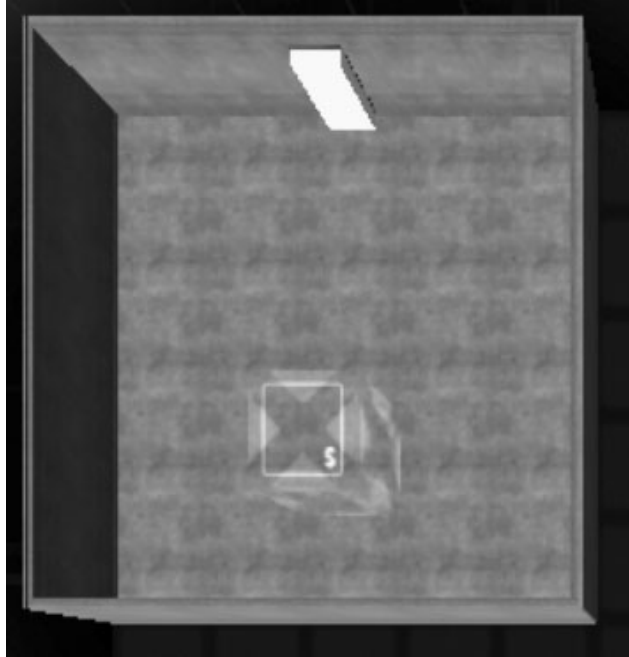


FIGURE 16.18 A 3D room that takes up a number of the grid squares.

Each block on the grid is $100 \times 100 \times 100$ in size in the game world. This is important if you intend to make your own prefabricated rooms using some of the segment building blocks.

Cursor

The cursor is the mouse display when in use in the 3D editor. Depending on whether you are adding a prefab, entity, or painting a floor will depend on what this cursor looks like. This is particularly useful to tell you the current mode the mouse cursor is in, so that you don't accidentally delete or overwrite your rooms. For example, if you are in segment mode, you will have a square box with an X displayed in it and the letter S in the bottom-right corner. The cursor is used to drop many of the game components into your program and paint the various textures onto the walls, floors, and ceilings.

Status Bar

The status bar, shown in Figure 16.17, is located at the bottom of the FPS Creator window. As you move the mouse cursor around the editor, the status bar automatically updates its current position. This is particularly useful when you are placing rooms and items in specific locations.

CHAPTER SUMMARY

In this chapter we took a quick tour of the FPS Creator program and its main toolbars. We have also looked at how you might want to think about creating your game and the order in which you might want to approach it. In the following chapter we will begin to use some of the items identified in this chapter and start to build a simple game. The game won't be too technical but will provide you with all the necessary building blocks to be able to make your own game.

CREATING A BASIC GAME WITH FPS

In This Chapter

- Creating Your First Room
- Testing Your First Level
- Player Starting Position
- Adding a Weapon
- Adding an Enemy Player
- Creating a Corridor



This chapter will go through various options to help you understand how to build your games and add all the content required to make a first-person shooter. While doing so, you will end up with a simple game that will introduce you to the various functionality and mechanics of the product stage by stage. At the end of this chapter you will have learned many of the basic features, which will allow you to begin to put together your own game, structures, and maps.

CREATING YOUR FIRST ROOM

The first thing to do is drop a prefab room onto the 3D editor. A prefab (prefabricated) room is a room that has already been created for you. All you need to do is to select it from the library and place it on the grid. When you load up FPS Creator it creates a blank game file automatically. If you placed a number of items on the grid, you might need to clear these away before starting on your room. You can clear any work you have already done and want to delete by selecting File | New from the Menu option.

1. Ensure that FPS Creator is started.
2. Double left-click on the Add New Prefab icon in the Library toolbar, as shown in Figure 17.1.



FIGURE 17.1 The Add New Prefab icon.

3. You will be shown many prefabricated rooms from the prefab library, as shown in Figure 17.2.



Notice that some of the library items are grayed out. This is because the number of prefabs is restricted in the demo version. If you purchase the full version, these objects will be unlocked for you to use.

4. Select the first prefab, called bunker large, either by double left-clicking on it or single-clicking to highlight and then clicking OK on the bottom-right corner of the screen.
5. You will now see a blue-and-white room move on the grid as you move your mouse across it as shown in Figure 17.3. Notice that there is also a large blue

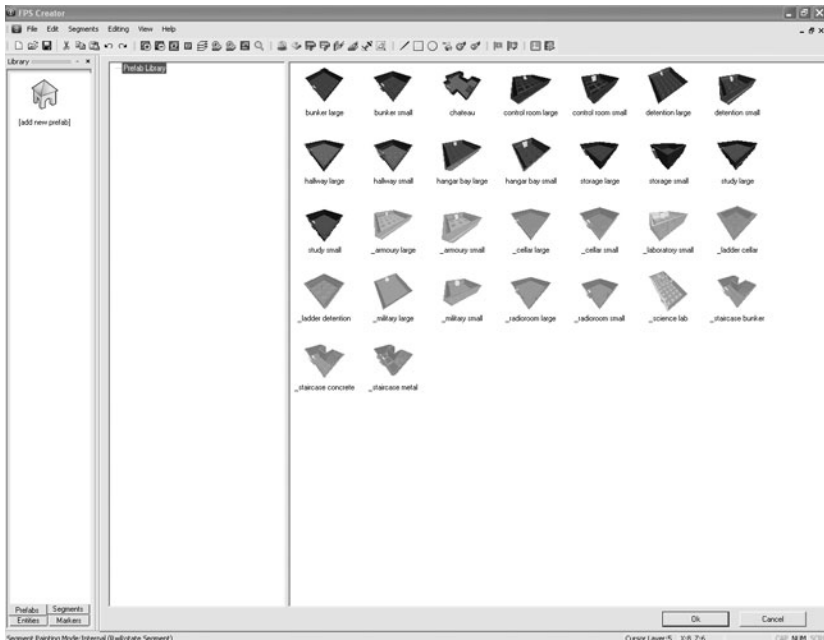


FIGURE 17.2 The prefab library.

arrow pointing upward. This tells you the direction of the room. This is where the exit point of this room is located. In Figure 17.3 it's in the top wall. You cannot place the room at the very top of the grid, as you wouldn't have room to place a corridor or other room for the player to exit into. You either need to rotate the room so that the door is located in a different direction or leave enough space on the grid to allow the placing of other prefabs or sections.

6. You may have trouble seeing the prefab on the grid, as the grid is zoomed in too close to see it. As highlighted in Figure 17.4, you can roll the wheel button on your mouse away from you to zoom in or toward you to zoom out. In this case you want to zoom out, so move the mouse wheel toward you.

Once you have learned how to zoom out, you have two options for placing the prefab and moving around the screen. You can zoom out all the way so that you can see the whole grid area and then left-click on the grid to place the item, or you can use the arrow keys (if you haven't fully zoomed out) to move around the grid. Again, you can click the left mouse button to place the prefab on the screen. Being able to see the whole grid can be very useful, but it can make it difficult to place an item in a specific area of the screen. If you were placing a weapon or a person at a specific location, you would find it easier to zoom in closer and use the arrow keys to navigate. If you are placing a large prefab, you can zoom out a little or you can keep a watchful eye on the status bar at the bottom of the screen to see its current location coordinates.

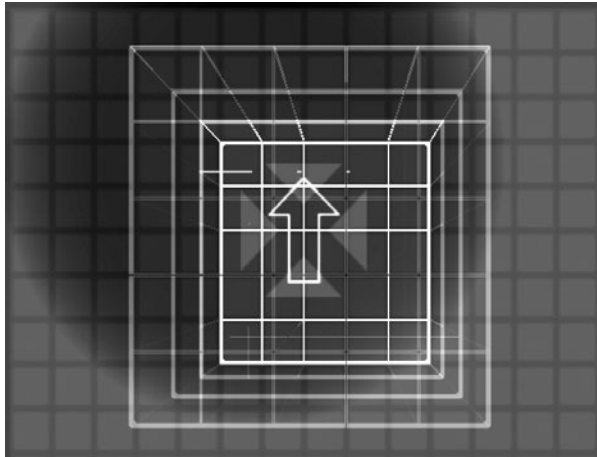


FIGURE 17.3 The cursor has the outline of the prefab attached to it.



FIGURE 17.4 The mouse wheel identified.

Now you can place the prefab onto the screen. For this example, place it near the bottom of the grid.

7. Move your mouse cursor until the coordinates say “X5 Z34.” Then click the left mouse button. This places the prefab room on the grid.



Once you have placed your prefab, if you move the cursor away from the room, you will notice that it changes from a textured gray room with a green door at the top to a blue-and-white shell. This is just how FPS Creator saves resources while creating your maps, as it doesn't then have to display the full graphics and items for every object placed—only the area that you are working on. If it didn't do this, even the more powerful home computers would begin to struggle to display all of the graphics at one time.

Notice that once you have added the first prefab, the mouse still has the prefab room attached to it. This is because you might want to repeat the action a number of times. This saves some time when you are placing common items onto the grid, as you won't need to keep going into the library and selecting the object every time.

You now want to place another room at the top of the prefab that you have just added to the grid. If you want to change the door direction of the prefab, you can do this by pressing the R key, which rotates it. This quick key also rotates other objects.

8. Ensure that the mouse cursor is at the grid position (X5, Z29) and then click on the left mouse button to place the prefab room.



You have finished adding this particular prefab. To prevent any accidental placing of more prefabs or deleting parts of the ones already in place, you can remove the prefab from the cursor by pressing the Delete key.

9. You now have two rooms connected to each other, as shown in Figure 17.5.

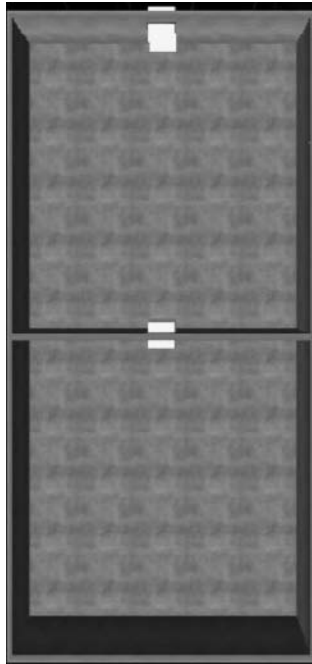


FIGURE 17.5 Two connected prefab rooms.



If you get stuck, you can load an example file that contains the two connected prefabs. The file is called `basic2rooms.fpm` and is located in the `FPSFILES` folder on the DVD included with this book.



ON THE DVD

TESTING YOUR FIRST LEVEL

Now that you have created your very simple level, you need to test it. You can test your creations at any time to check your work and make sure it's running exactly as you intended. You can do this by clicking the Test Level button detailed in the previous chapter and shown in Figure 17.6.



FIGURE 17.6 The Test Level button.

This runs a special program that prepares your game for display on the computer. It displays a dialog box while it is compiling, as shown in Figure 17.7.

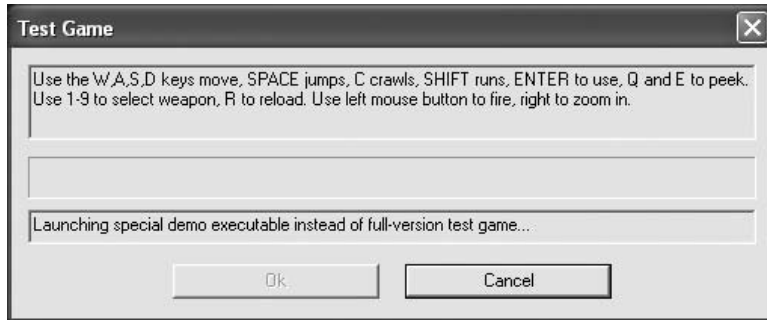


FIGURE 17.7 The Test Compile dialog box.

The Test dialog box also explains how you can navigate around the world once it has been loaded onto your screen. The keys for playing the two-room test are the same as for any game made with FPS Creator and are shown in Table 17.1.

Table 17.1 Keys Used to Play the Test Game

KEY	EFFECT
W	Forward
S	Backward
A	Left

KEY	EFFECT
D	Right
Shift	Run
Spacebar	Jump
C	Crouch
R	Reload weapon
Enter	Use or open door
Q	Peer left
E	Peer right

Once the game has successfully compiled and is ready to play, the OK button will be available. Click OK, and this launches the game in full-screen mode. You can now see your room and a door in the far corner, as shown in Figure 17.8. Move around your game and go through the door at the far end. You can go through one of the doors into another room, and the other door will lead to an outside area, at which point you will fall and lose health and a life. For the final game you want to remove the door leading to nowhere, but as this is a simple test, you can leave it be. When you are ready to exit the test, press the Escape key; this places you back at the editor grid.



FIGURE 17.8 The test game being run.



In Figure 17.8 some yellow text is displayed at the bottom of the screen when you run your test game. This is game performance information, and when you create a finished compiled game in the full version, it does not appear. This information and other available information help you improve the running performance of your game.

Now it's time to add some more items and configure your first creation.

PLAYER STARTING POSITION

Within the world you are creating, you want to specify where the player starts his adventure. This is the player's starting position. This is important, as many FPS games provide challenges and events around the player's position. For example, early on in the level the game may provide the player with a simple introduction to the controls or to the weapons he might use. By placing the starting position in a specific room or area you can guide the player through the challenges, making the end result more interesting.

To add the player's starting position you need to add a marker from the Library toolbar.

1. Click Markers at the bottom of the Library toolbar. You can now see a selection of different items within the library, as shown in Figure 17.9.



FIGURE 17.9 A selection of markers in the Library toolbar.

2. The very top marker is Player Start, so single left-click on this and see that the mouse cursor changes into a green glowing arrow, as shown in Figure 17.10.
3. The cursor coordinates are located on the cursor position, so left-click on the location (X7, Z38). This places a green arrow in the first room you created.
4. You may now want to delete the Player Start icon from the cursor so that you don't accidentally place another start location on the grid, so press the Delete key.

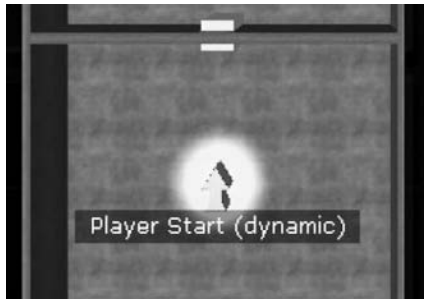


FIGURE 17.10 The cursor changed to the player start marker.

If you run the game now, the player will start in the first room, and you will be able to move between the two rooms. This isn't a big issue when you have created only two rooms, but as the game levels gets larger, it's essential to know where the player's start position is. Therefore, when possible, ensure that you create the player's start position early on in your creation.



You can find an example file of the two rooms and start position on the DVD that comes with this book. It is located in the FPSFILES folder and is called player-start.fpm.

ADDING A WEAPON

An FPS game without a weapon wouldn't be much fun, so the next stage of your game is to add a weapon in the room in which your character starts out. The weapon for this example is placed on the floor, but in your own games you might decide to place a weapon on a table or on top of another object. In a game you are taking more time over you should consider carefully where to place items such as weapons, ammunition, and health, as these can have a major influence on how easy or hard your game is to play. Place too many weapons and too much health, and the game will be a walk in the park for the player. Place very few, and the player might be lucky to get through the first few levels, but would probably give up playing the game. This is a difficult balance to get right, but this fine-tuning can be done later on in your creation.



Using the file you've created so far, which contains two rooms and a player start position, add a single weapon to the first room. If you don't have this file loaded, you can load the playerstart.fpm file from the FPSFILES directory to save you having to create the file again.

1. Click Entities in the Library toolbar and then click Add new entity
2. From the entity library that appears, select Scifi | Items and then select the tavor weapon, as shown in Figure 17.11.
3. Your cursor now contains the tavor gun, and you can rotate the item if required by using the R key. Left-click on the grid location (X5, Z34) to place the gun in the top-left corner of the first room.

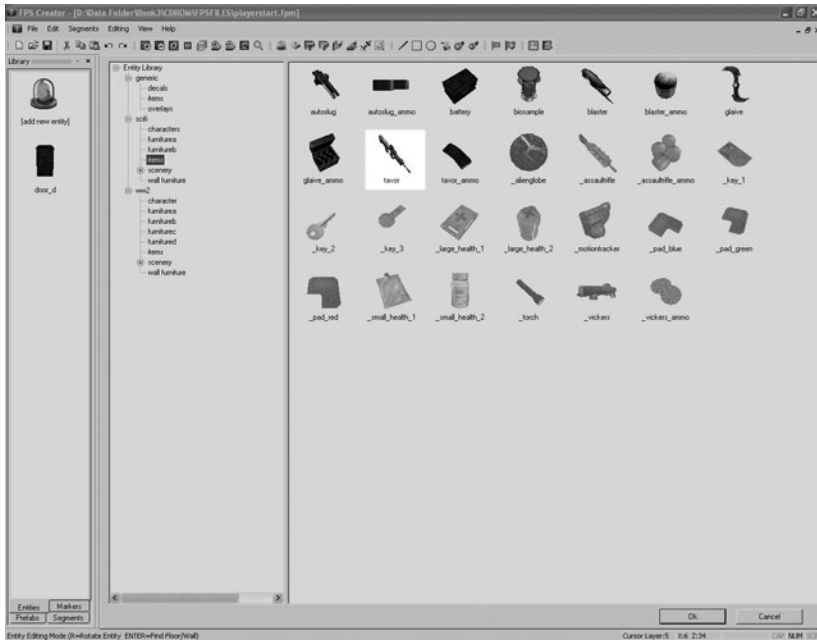


FIGURE 17.11 The entity library with the tavor selected.

4. Press the Delete key to remove the gun entity from the mouse cursor.
5. Run the game by clicking Test Level button. Then pick up the gun by walking over it and try shooting. Figure 17.12 shows how the gun looks in the player character's hands.



FIGURE 17.12 The gun picked up and being carried.

Notice that this gun has limited ammo, so you will need to rectify this later on by placing additional ammunition items around the map, but there are other options to increase the default gun amount. These configurations and how they can effect your games are covered later on.

ADDING AN ENEMY PLAYER

A game wouldn't be very good without enemy computer-controlled players, so you now need to add one to your game.

For this game, add a single enemy player in the second room. This gives the player the chance to pick up the weapon before being introduced to an attacking character. First, ensure that you have the right file loaded. You need the current file you have been working on or the file `firstgun.fpm` loaded. The `firstgun` file is located on the DVD in the `FPSFILES` folder.



1. Ensure that the Entities tab is selected in the Library toolbar.
2. Click Add new entity and select Scifi | Characters from the entity library.
3. Select Conker (Pistol) from the objects that are displayed, as shown in Figure 17.13.



FIGURE 17.13 A selection of enemy characters to choose from.

4. Your cursor changes to a green circle, which contains the Conker character. The character is pointing upward on the grid. This means that when you enter the room he would have his back to you. You need to rotate him so that he is facing you as you come through the door. Press the R key four times to have him pointing downward on the grid. Left-click the mouse button in the second room at coordinates (X7, Z29).
5. Press the Delete key to change the cursor back to its basic type, so that you don't accidentally add any more characters to the grid.

If you run the game now, you can rush toward the gun and pick it up. If you walk through the door, you will be able to get into a gun battle with the enemy character, and you should be able to win against him. An example of what you should see is shown in Figure 17.14.



FIGURE 17.14 The enemy is closing in.

Notice that once you have defeated the enemy character, he drops his weapon, which you can pick up and use. This is another great way of allowing the player to pick up extra ammunition and weapons throughout the game and can be used as an optional way of providing the player with new supplies rather than leaving additional items littering the room.

CREATING A CORRIDOR

Now you want to expand your game some more by creating a corridor to another room. In all of the games you create you need some way for the player to move be-

tween areas. This could be inside between rooms or outside moving between a garden and a building. In both cases you need an interconnecting area. This could be room to room, but that wouldn't make your game levels very exciting. For this example create two corridors. There are a number of ways to create this link between rooms:

- Create a floor area between the rooms, for example, a lawn outside. You need to place a wall around the space to enclose it.
- Create a corridor by placing a many segment components of a corridor. In this option you build the floor, walls, and ceiling independently of each other.
- Create a corridor using segments—a tunnel or ventilation shaft segment. These are easier to place than separate wall, floor, and ceiling segments.

Creating a Corridor Using Single Segments

In the first example you will create a connection between rooms using single segments parts that contain the floor, walls, and ceiling but only cover one block on the grid. First, you need to ensure that you have the right file loaded: the current file you have been working on or the file `firstenemy.fpm`. The `firstenemy` file is located on the DVD in the `FPSFILES` folder.



Before you begin creating your corridor, create another smaller prefab room above the one that contains the enemy computer-controlled player. This allows your hero to navigate to another room and sneak up and attack the enemy.

1. Click on the Prefabs tab in the Library toolbar.
2. Left-click the Add new prefab icon.
3. Double left-click the bunker small prefab.
4. Place the smaller room at the location (X6, Z26).
5. You now have three rooms, and the topmost one on the grid is the smallest. You can see your current room structure in Figure 17.15, where item 1 is our new room, item 2 is where the enemy player has been placed, and item 3 is the player's starting location.
6. Now you want to add the single segments sections by using the Ventilation object, so click Segments in the Library toolbar. Click Add new segment and then select the object called Ventilation Duct Straight.
7. The cursor changes to a line with two arrows, one at each end, as shown in Figure 17.16.
8. These two arrows indicate the direction that the Ventilation Duct Straight is going to be placed onto the grid. This provides you with a good pointer in the direction that it will appear, so first you will need to rotate it to place it in the correct position.
9. Press the R key to rotate the arrows once so that they are pointing left and right.
10. Place the cursor at the coordinates (X4, Z37) and click on the left mouse button to place the first part of the ventilation system.

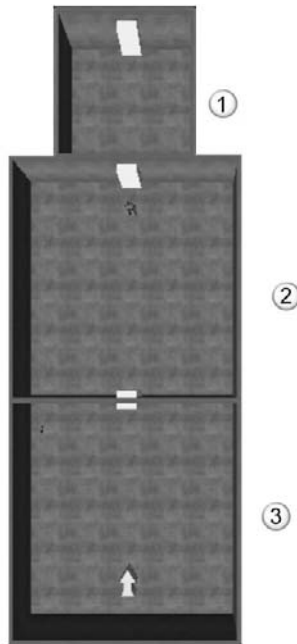


FIGURE 17.15 The three completed rooms in the game.



FIGURE 17.16 The cursor has changed to two arrows.

11. Now place a 90-degree part so that the player can start to move up on the grid. Click Add new segment and choose Ventilation Duct Corner from the Scifi | Corridors menu. Your cursor will change to two arrows, one pointing left and one pointing down. This shows you where the corner of the ventilation segment would be, so press R twice to rotate it so that the player will move in the correct direction.
12. The arrows should be pointing up and to the right. Move your cursor to the coordinates (X3, Z37) and left-click the mouse. This places a right turn in the ventilation system.



Any objects, segments, or prefabs you use in your game now appear in the Library toolbar so that you don't have to select from the large list. This can speed up development if you are using the same items throughout your game.

13. Now that you have the right turn segment, you need to place a number of straight-line pieces to get the ventilation system up to the top room on the grid. Select Ventilation Duct Straight from the Library toolbar, rotate it once, and then place the object at the following coordinates: (X3, Z36), (X3, Z35), (X3, Z34), (X3, Z33), (X3, Z32), (X3, Z31), (X3, Z30), (X3, Z29), and (X3, Z28).
14. It's time for the vent to make a right turn so that it moves toward the top room. Select the Ventilation Duct Corner object from the library. The arrows should be pointing down and to the right. If they are not, press the R key. Once the direction is correct click on the coordinates (X3, Z27).
15. Now it's time to add another straight piece of ventilation system so you can connect it to the room. Select Ventilation Duct Straight from the Library toolbar and ensure that the direction arrows point left and right. Then click on the coordinates (X4, Z27), (X5, Z27). You now have your rooms and connecting corridor looking like Figure 17.17.

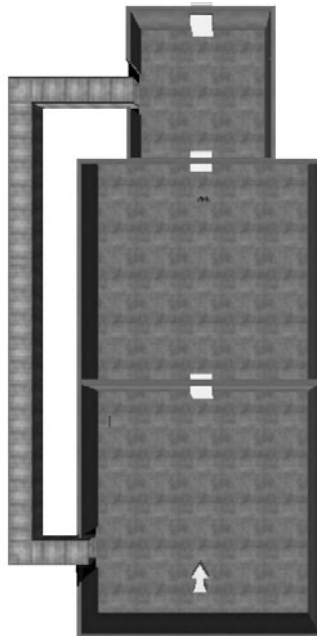


FIGURE 17.17 They current room layout.

When you run the game, you should now see your newly created ventilation system, but it doesn't look quite as shown in Figure 17.18. When you placed the start and end of your ventilation system onto the grid, it deleted the walls that they were connected to.



FIGURE 17.18 Problem with the ventilation system wall.

You can do two things to get this working correctly. First, paint a wall segment to cover the gap that has been created by placing the ventilation system. Then create an access into the ventilation system by placing a small door. If you do not do this in this order, you may find that there is no access into the ventilation system.

To paint a wall segment onto the gaps that were left when the ventilation system was put in place, find the wall texture image used for this prefab and paint it onto the wall gap. When you add a prefab, it also adds the relevant segments that make up the room, so the first thing to do is find the segment that relates to the room. Then you can select the Paint Only Segment walls option.

16. Ensure that Segments is selected in the tabs of the Library toolbar.
17. Click on the Bunker Fighting Area Low item and then in the menu buttons select Paint Only Segment Walls.
18. Press R to rotate the arrow until it is pointing to the left.
19. Single left-click the mouse to paint the wall segments at coordinates (X5, Z37) and (X6, Z27).

If you now run the game, you will notice that the wall segments all look the same, and there is no sign of the ventilation system. Next, you can drop the ventilation door into place, and this automatically knows to cut away the wall section to place its opening. FPS creator does all the major work for you as long as you do it in the correct order as mentioned previously.

Now it's time to drop a ventilation door into place:

20. Click the Segments tab and then Add new segment.

21. Ensure that the SciFi | Scenery menu is selected, and you should see a large selection of door and window options. Double left-click on Door Vent Duct (use).



Notice some of the items have (use), (auto), (remote), or (fake) after them. This is how they can be used or accessed. In the case of a use door, you need to press the Enter key (also called the Use key in the game) to open the door or window. If it is auto, when you are close to the object it will automatically open. Remote means you can create a wall button to open the door, and fake means the door does not open at all and is there for cosmetic reasons.

22. Press R to rotate the arrow until it is pointing to the left and then left-click at the coordinates (X5, Z37) and (X6, Z27).
23. On running the game, you should be able to see the ventilation door as shown in Figure 17.19, and when you walk up close to it, you will be asked to press the Enter key to access it.



FIGURE 17.19 The newly created ventilation system.



If you want to check your work against the completed ventilation system, you can load the example called `vent.fpm` in the `FPSFILES` folder on the DVD.

Creating a Corridor Using the Paint Tools

Rather than use segments as the building blocks, you can paint your walls, floors, and ceilings directly onto the grid. This gives you a lot of power to make the world much more varied than using the basic prefabs or segments that are already provided. Next, take the graphical look and feel of one of the prefabs and create your own corridor from the first room, winding its way up to the last room.



Ensure that you have the file `vent.fpm` loaded. This file can be found in the FPS-FILES folder on the DVD provided with this book.

The file provides you with a simple three-room map, with a ventilation system that you created earlier. You can create this corridor a number of ways. You can paint using the basic floor tiles or paint the whole floor and walls in one go. For speed, and because it automatically applies a good textured wall and floor, use the second option.

1. Click the Segments tab in the Library toolbar.
2. Now you need to select the segment you will use to paint the walls and floor with, so single left-click Bunker Fighting Area Low. Now click on the coordinates shown in Table 17.2 to paint the walls and floor into place:

Table 17.2 Coordinates to Paint the Corridor

COORDINATES (X, Z)

(10, 37)
 (11, 37)
 (12, 37)
 (12, 36)
 (11, 35)
 (11, 34)
 (11, 33)
 (11, 32)
 (12, 32)
 (12, 31)
 (12, 30)
 (12, 29)
 (12, 28)
 (12, 27)
 (11, 27)
 (10, 27)
 (9, 27)



If you make a mistake, click the right mouse button to delete the grid entry you just placed.

You should now have a corridor that connects the bottom room to the top room. Notice that between the two rooms the corridor is connected to an open wall. You will want to have this open area filled in with a wall and a door. Now it's time to paint a wall in both locations.

3. Click Paint Only Segment Walls.
4. Press R to rotate the arrow until it is pointing to the left. Then click on the coordinates (X10, Z37) and (X9, Z27).

You need to add the door, so you can use the one that is already in the Segment tab of the Library toolbar.

5. Click Door Tough (use). The arrow should be pointing to the left. Then click on the coordinates (X10, Z37) and (X9, Z27).

If you now run the game you can access the corridor through the door to the right, walk around the middle room, and attack the enemy player from behind. Notice that there currently isn't any roof to the corridor. You may not necessarily want one in your own game, or you may want to cover part of the corridor to provide small gaps for the player to look out. The easiest way to do this is to use the Bunker Fighting Area Ceiling and apply it to the corridor roof. To do this you have to apply the roof to the level above. Otherwise, you will actually be placing the ceiling on the current level's floor. So first, you need to move up a single level so that you can paste the ceiling.

6. Click the "+" button on the keyboard to move up a level. The cursor layer will now read "6."
7. Ensure that Segments is selected in the Library toolbar tabs. Then single left-click Bunker Fighting Area Ceiling and then click on the coordinates given in Table 17.3:

Table 17.3 Ceiling Coordinates

COORDINATES (X, Z)

(10, 37)
 (11, 37)
 (12, 37)
 (12, 35)
 (11, 35)
 (11, 34)
 (11, 32)
 (12, 32)
 (12, 31)
 (12, 29)
 (12, 28)
 (12, 27)
 (10, 27)
 (9, 27)



You may notice that your rooms are two levels deep, while your corridor is only a single level deep.

Run your game and go through the right-hand door. Your corridor now has a roof though some parts of it are left open for the player to look through, as shown in Figure 17.20.



FIGURE 17.20 Corridor with gaps on the ceiling.



If you want to check your work against the completed corridor that you have just walked through, you can load the example called `corridor.fpm` in the `FPSFILES` folder on the DVD.

CHAPTER SUMMARY

In this chapter you looked at FPS Creator and created your first set of rooms, placed a weapon, and selected the location in which the player would start the game. You also placed an enemy player, created a ventilation system, and painted a corridor system with a roof. You now have the knowledge and the skills to place some of the basic components in the world. In the next chapter you will be expanding our game and placing more complex items and options that you might see in today's FPS games. This includes stairs, elevators, trigger zones, and many other exciting things.

FPS CREATOR: NEXT STEP

In This Chapter

- Adding Windows
- Creating Door Switches
- Lighting Rooms and Corridors
- World Effects: Smoke and Fire
- Making Your World More Visually Exciting



Now that you have your basic game in place, it's time to begin adding some more content and features to the game and increasing its overall complexity. Do this by adding more rooms and more enemies, and while doing so add more features that are available in FPS Creator. This will give you a strong ground- ing to take your games to the next level.

ADDING WINDOWS

Adding windows to your game is a nice touch and can give the world a more realistic feel, as well as allowing the players glimpses into rooms they are approaching. Windows can be destroyed, allowing the player to walk through it, or they can be fake windows that the user can't actually look through but give the feeling of it being a real building. You can also apply smaller windows that can be destroyed but are too small for the player to walk through.

Load the game you have been working on, and then you can add an additional room and add some windows to it. The file is called `corridor.fpm` and is located in the `FPSFILES` folder on the DVD provided with this book.



First, create two new rooms, one of which will have windows that look into the other room and allow you to see what's inside it.

1. Ensure that the file `corridor.fpm` is loaded, and then click Prefabs in the Library toolbar.
2. Click Add new prefab and then select hallway large.
3. Left-click on the grid at the location (X5, Z21) and then with the mouse still displaying the prefab cursor, press the R key three times until the arrow is pointing to the right. Then click the left mouse button at the coordinates (X10, Z21) to place another room to the right of the one you've just added.
4. Run the game to check that you have two rooms, as shown in Figure 18.1.



As you add more and more content to the game, you may notice that it takes longer to load when you want to preview it. This is normal, as your computer has to compile the program into a special version for previewing.

5. Now it's time to add some windows. Click Segments in the Library toolbar. Then click Add new segment.
6. Double left-click Window Large. Press R on the keyboard so that the arrow is pointing to the right.
7. Click on the coordinates (X9, Z21), (X9, Z22), (X9, Z24), and (X9, Z25).

One of the two new rooms added previously now has windows across it. Figure 18.2 shows the grid view, and Figure 18.3 shows the game view.

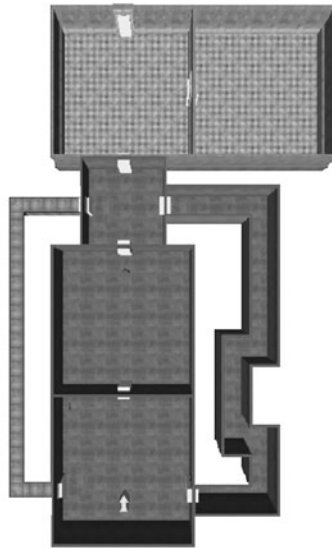


FIGURE 18.1 The room layout with two new rooms.

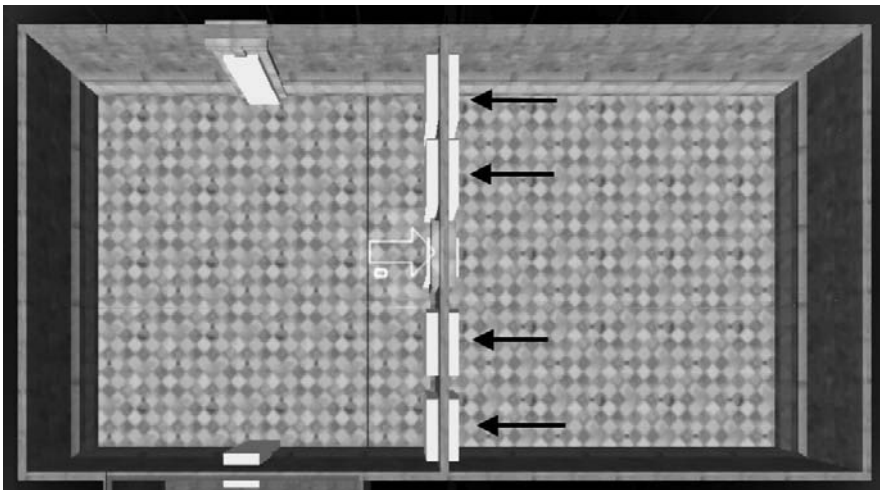


FIGURE 18.2 Grid view of the windows, shown next to the arrows.

8. Run the game and move to the two new rooms. Try shooting out the windows, jumping through them (once they have been destroyed), and generally have a look around and get used to what the window system might do for your games.



FIGURE 18.3 The windows as displayed in the game, with one in the process of being destroyed.

CREATING DOOR SWITCHES

Most FPS games have some doors that open automatically, some that require a key, and others need the player to click on a switch to activate them. When you want to use a wall switch to open a door, you need to place a Remote Door. Until the user clicks the switch, the door remains locked and there is no access into the next room or corridor.

Load the game you have been working on, to which you will add an additional room and a door with a switch. The file you need to load is called `windows.fpm` and is located in the `FPSFILES` folder on the DVD provided with this book.



1. First, add a new room in which you can place a door that will be activated by a switch. Select the Prefab tab in the Library toolbar, click Add new prefab, and choose `bunker small`. Left-click on the grid coordinates (X2, Z22).
2. Now you have a room, but with no door access, so click the Segments tab on the Library toolbar and click Add new segment. As you are using the demo version, not all objects are available, so you cannot pick the best shaped and painted object for your game, but to see how it works, select `Door Prison Cell (remote)`.
3. Press R until the arrow is pointing to the left and then click on the coordinates (X5, Z23).
4. Now you need to place the door switch, so click the Entities tab on the Library toolbar, click Add new entity, navigate to `Scifi | Wall furniture`, and select `switch7`.

5. Place the wall switch away from the door to make it more interesting for the player so he will need to flip the switch in another room to make the door open. In the top-right room (the one with the windows) place the switch at coordinates (X13, Z21). You can see the map and the door and switch objects highlighted in Figure 18.4.

Run the game. You have to go into the room with the windows and flick the switch before the door opens. If you try to access the door before hitting the switch, notice that there is no message on screen advising you to press the Use key, and the door will stay firmly closed. You can see the switch and door in the distance in Figure 18.5.

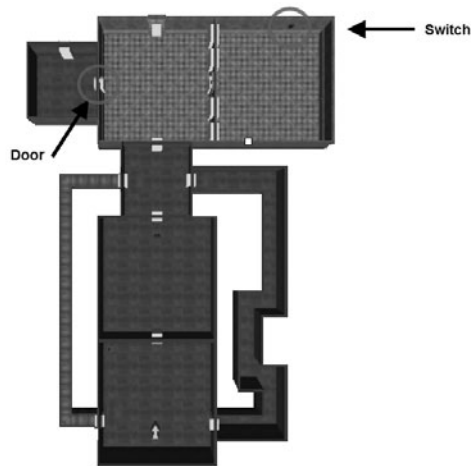


FIGURE 18.4 The door switch and locked door on the grid.



FIGURE 18.5 The door switch to open the door in the distance.

LIGHTING ROOMS AND CORRIDORS

An important aspect of any game is lighting the environment. This can make all the difference in making a more chilling and scary experience for the player, as well as making the world look more realistic. A good example of this is games like *Doom*® 3, where you walk through a space base on Mars that has seen a catastrophic accident, and many of the locations have different levels of damage. Many lights are flickering, and in the corridors very little light exists, which make it very scary for the players to move around the world, as they cannot see far in the distance. This tense situation is made worse where creatures can appear around any corner, taking the player by surprise.

There are two types of lights in games: natural light and artificial light. Artificial light is created easily by placing a lamp or other light object on the screen and then adding a light effect to it. Natural light is the combination of the surroundings. For this type of lighting you can add a light effect on its own without placing it on a light object.

FPS Creator by default lights all of the rooms and corridors you create the same way, so when you play, all the areas can be seen. For a basic game this is perfect and very simple to get up and running. If you want to make your game a little more exciting and professional looking, as soon as you add a light effect, this “light all” is switched off. Therefore, when you play the game with only one light applied, the rest of the rooms are in near darkness. Unfortunately, this means that as soon as you begin to add lighting to your game there is a lot more work to be done to get your game looking right. An example of a room that is lighted and a darker room beyond can be seen in Figure 18.6.



FIGURE 18.6 Room beyond in the dark.

Adding a light is a simple process.

1. Select the Markers tab in the Library toolbar.
2. Select the relevant light object, as shown in Figure 18.7, and drop it onto the grid in the appropriate location.



FIGURE 18.7 The available light markers.

WORLD EFFECTS: SMOKE AND FIRE

In some worlds there may be reason to add a fire or smoke effect to make the game visually pleasing or to restrict the user's movements. For example, you could place a fire effect in a room, blocking a particular route. A number of World War 2 FPS games use fire to add to the overall feeling of danger but also to block a particular route. This is a game trick, as in these cases the route blocked by fire may not have ever been a route out of the map, but it adds an illusion of a bigger world that the player can't get to. In the game you have been making over the past two chapters, you are going to add a smoke and fire effect.

Adding Fire to Your Game

It's time to add a fire effect to the top-left corner room, in which you created a door that could only be opened by a switch. You need to ensure that you have the file



lights.fpm open, which is located in the FPSFILES folder on the DVD provided with this book.

1. Click on the Entities tab.
2. Click the Add new entity graphic.
3. From the menu select Generic | Decals.
4. Select Fire and then you will be taken back to the grid.
5. The cursor has changed to a cone-shaped object, which you can now place on the grid. Left-click on the grid in the locations listed in Table 18.1 to add the fire effect.

Table 18.1 Fire Effect Locations

COORDINATES

X4, Z24

X3, Z24

X2, Z24

X4, Z23

X3, Z23

X2, Z23

X4, Z22

X3, Z22

X2, Z22

You can now see your cones of fire, as shown in Figure 18.8.

If you run the game, go to the door switch and enable it. Then move to the room at the top-left corner of the world and you will see the fire effect. An example of this is shown in Figure 18.9. You can walk into the fire with no ill effects, as you would need to tell the program to reduce health when walking through it.

You will see how to add the reduction of health in the next chapter.

Adding Smoke to Your Game

Using the same process that you used for adding fire, you can add smoke to your game. In this case you are going to add an industrial pipe and make the smoke appear from that. Load the file before smoke.fpm, which is located in the folder FPSFILES on the DVD.



1. Click Entities and select Add new entity.
2. Select SciFi | Furniture | EquipmentD.
3. You should be back at the grid map. Press R twice to rotate the pipe until it is pointing to the right.

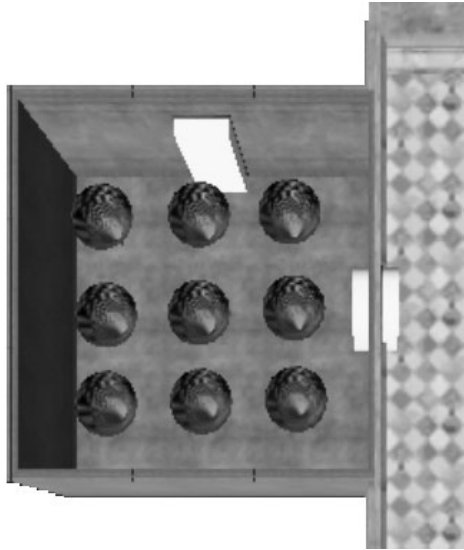


FIGURE 18.8 The cones of fire in place, ready to set the room alight.



FIGURE 18.9 The fire effect in the game.

4. Click on the coordinates (X9, Z35) to place the pipe.
5. Now you can add the smoke. Ensure that the Entities tab is selected in the Library toolbar and then click Add new entity.
6. Choose Generic | Decals and select smoke1 and place it at the coordinates (X9, Z35).

7. As a square isn't as precise as you may want it, once you have placed the smoke, you have the opportunity of moving its location. If you have placed the smoke and you still have smoke as the cursor, press Delete key before you start.
8. Move the mouse cursor over the smoke object and notice that it is highlighted in green. Click the right mouse button to access the object properties and zoom in close to it.
9. Holding down the right mouse button and moving it in a particular direction will move the 3D world, allowing you to get a better look at the smoke's location. An example of this can be seen in Figure 18.10.

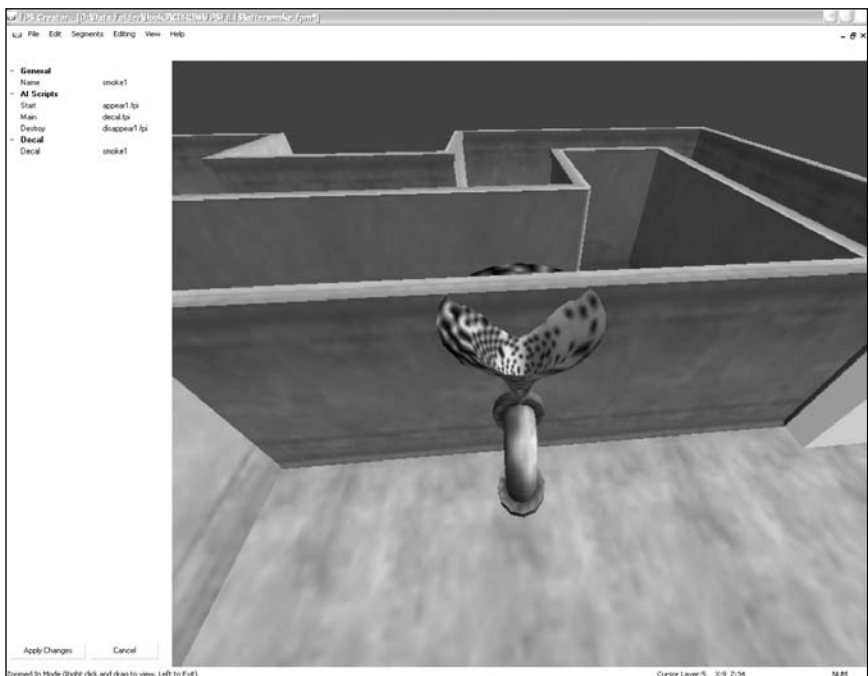


FIGURE 18.10 Zoomed in close to the smoke.

10. Use the arrow keys to precisely position the smoke. Click the left mouse button to return to the normal grid mode.
11. Run the game and see the smoke appearing from the pipe object, as shown in Figure 18.11.



As the smoke object is close to the wall, you may have to move it a small amount to ensure that it looks correct on the screen. If you place it too close to the wall, you may get a strange effect, where part of the smoke is cut away. It is always a good idea to place an object on the grid. Then check how it looks and reposition it if need be.



FIGURE 18.11 The smoke appearing from the pipe.

MAKING YOUR WORLD MORE VISUALLY EXCITING

When you are creating the world for your characters, it is useful to furnish this world with items that relate to the characters' surroundings. For example, if you are creating a game in a hospital, you might want to place beds, medicine cabinets, and hospital-based wall signs.



1. Ensure that the file `aftersmoke.fpm` is loaded. The file is located in the `FPS-FILES` folder on the DVD.
2. Ensure that the Entities tab is selected in the Library toolbar. Select Add new entity.
3. Select the Scifi | Furnitureb menu item and then the `storageC` object.
4. Place two of these items in the room just above the room where the enemy player is located at coordinates (X6, Z26) and (X8, Z26).



You may need to right-click on the shelves to position them exactly on the screen.

5. Now that the shelves have been placed, you can put items onto them. Click Add new entity, then Scifi | Items. Select any of the available items and then place them on the shelves. You can do this a number of times to fill up the shelves.
6. You may find placing items very difficult because all of the items start at floor level. To place an item on a particular shelf level, use the Page up and Page down keys.



Another useful tip for precise positioning of multiple objects of the same item is to place a single item and then right-click on it to edit the properties. You can then precisely position it. Once you have done that, exit out of the properties and then left-click on the object to make it moveable and left-click a number of times to place the object. You can see this in action in Figure 18.12.

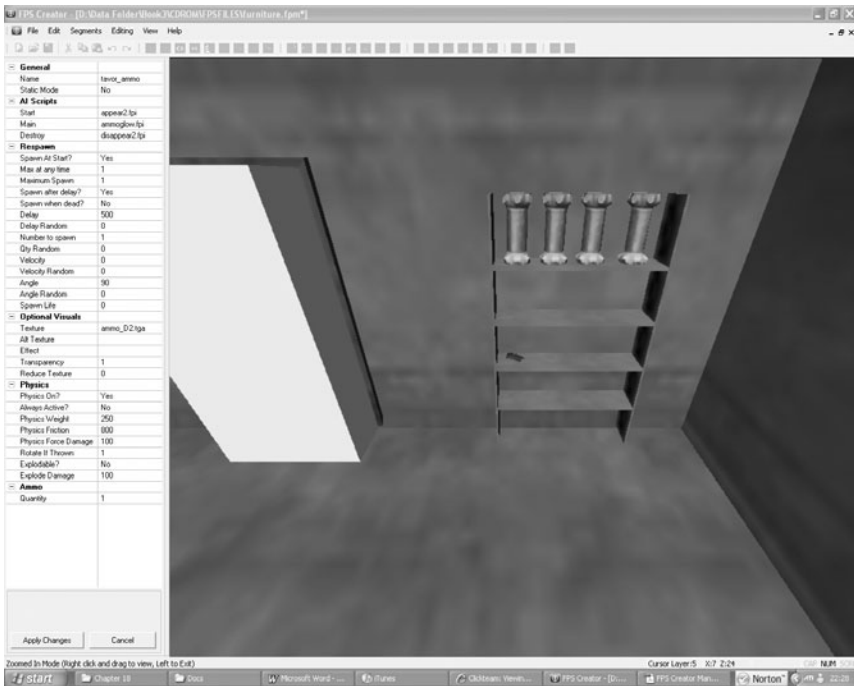


FIGURE 18.12 Close up of one of the shelves to place items precisely.



For an example of some items placed on the two shelves, open furniture.fpm in the FPSFILES directory on the DVD.

7. Try adding more items from the library to make your world more exciting.



You may notice when you run the game that when you look at the shelves, the ammunition is moving up and down. This is because these objects are dynamic objects, which means they may have a movement applied to them. You can change an object's type by going into its properties, which will be covered in the next chapter.

CHAPTER SUMMARY

In this chapter you have built upon your knowledge of FPS Creator and began to make your game more interesting for the players. You did this in two ways, first, by providing more of a challenge, whereby they need to flip a switch to open a remote door, and second, by adding more content to make the game world feel more realistic. When you are creating your own games, think about the surroundings and how they should be presented to the player. In the next chapter you will begin to add more functionality to your game including special game markers, which tell FPS Creator when to do a specific task.

TAKING FPS CREATOR TO THE NEXT LEVEL

In This Chapter

- Stairs, Elevators, and Teleporters
- Creating Enemy Patrols Using Waypoints
- Zones



This chapter covers various aspects of FPS Creator that will start to make your game stand out and make it more professional in terms of what the player can do within the world.

STAIRS, ELEVATORS, AND TELEPORTERS

Many games exist on various height levels, whereby the player moves up a ladder or a set of stairs to reach another room or position on the map. In the game *Half-Life*® 2 you are sometimes walking up building rubble and then jumping back down to the same height level that you started on. Using different types of items to move between levels certainly makes games more interesting than just providing a single level to move around on.



The word level has two different meanings in games. It can mean the number of separate sections in a game that are defined by a loading screen or a increasing level of difficulty. It can also refer to a game that exists on more than one floor. A floor does not necessarily mean it has to be confined to a building, but it is a handy way to visualize how the setup works.

In this chapter you will be adding a number of different ways to move around in your game on the different floors of the rooms. You are going to add a set of stairs, which the user can walk up, an elevator the user can walk into to travel up a floor, and a teleporter that can magically move the player to a new location when he walks into it.

Stairs

This is a common object in games in which the player is moving indoors to allow the player to move between floors. When making a set of stairs in FPS Creator, you will need to think about the placement of the stairs and prepare the floor and rooms above it, so that it all connects correctly. The other issue to think about is that many room prefabs are two levels in the map, so you need to consider adding areas for the player to walk on.



Open the file `beforestairs.fpm`, which is located in the `FPSFILES` folder.

In this example you are going to add a set of stairs and a gantry for the player to walk on and then place a door and a connecting room, which will be on the next level up. First, add a room and a door at layer level 6.

1. Press the “+” key to move to layer level 6.
2. Ensure that the Prefabs tab is selected and then click Add new prefab.
3. Select bunker small and place it at the coordinates (X8, Z33).
4. Now add a floor at the same level as the room (and the same level as the door once you have placed it). Ensure that you are at layer 6, click on the Segments tab, and then click Add new segment. Select Military Gantry Deadend and place it at the coordinates (X7, Z32). Press R twice and click on (X7, Z36).

5. Click Add new segment and choose Military Gantry Cross. Place it at the coordinates (X7, Z33).
6. Click Add new segment, choose Military Gantry Straight, and place it at the coordinates (X7, Z32) and (X7, Z35).
7. Now you can add a staircase to access this floor, so click on the “-” button to move back to layer 5.
8. Click Add new segment, select Staircase Metal, and press R until the arrow is pointing to the right. Left-click to place the object at the coordinates (X6, Z33).
9. Nearly everything is in place to allow the player to walk up the stairs and access another room at a different layer. Click the “+” button to move to layer 6.
10. Ensure that the Segments tab is selected and then click Add new segment. Select Door Control Room (auto).
11. Ensure that the arrow is pointing to the right and click on the coordinates (X7, Z33).
12. Run the game and you should be able to move up the stairs and walk into another room. You can see how it should look in Figure 19.1.



FIGURE 19.1 Stairs with access to another room.



If you want to compare this walkthrough with the final code, open the file after-stairs.fpm, which is located in the FPSFILES folder on the DVD.

Elevators

Using elevators is another great way of adding realism into your games. It is also a good way of breaking your game into levels or sections. Many games have used elevators to move the player between levels, and usually the elevator is one way to

prevent the player from going back where he came from. Half-Life and Half-Life 2 used lifts to great effect as a way of breaking the sections up, to continue telling a story, or creating a situation in a game that becomes hectic, fun, and memorable. In one (of many) memorable sections in Half-life 2 the player has to fight off the enemy while waiting for the lift to descend, so the player is stuck in a room, being closed in on by zombies while trying to survive long enough to get into the elevator. Creating something players fondly remember enjoying in a game is very important to their overall enjoyment and can sway a player's opinion to be positive. Another game that uses elevators well is Max Payne[®], in which toward the end of the game the player has located the enemies' base and the elevator is the level loading between levels, which were the different floors of the building.



In FPS Creator an elevator is called a lift, so we will continue to use both words where required.

In this section you are going to create a simple elevator effect. Creating an elevator requires three Lift parts and a Door to enter and exit the elevator.

Base of Lift. This is the very bottom of the elevator and is the entry point for the player. The door needs to be placed here.

Tube Section. If you create a large elevator over a number of levels, you need to connect the base of the lift and the top using the tube. You can connect the base and the top together in one section if you don't want to create a large elevator.

Top of Lift. This is the very top of the elevator.

When you select the lift segments, you need to consider if the elevator will move up or down. You can do this by selecting the segments that are highlighted with "platform." The top segment with the corresponding platform text in the name begins with the pad that moves the player to the top; the bottom (base) starts with it at the bottom.

In the following example you will create two rooms.

1. Open FPS Creator and begin on a new file.
2. Ensure that the Prefabs tab is selected and then click Add new prefab. Select the bunker large item.
3. Place the bunker large item in the coordinates (X0, Z35).

This places a room, but you need another for the elevator to connect to. Keep in mind the height of the rooms involved and where you need to place the next room to ensure that it is placed on the roof. In this case the building is three levels tall, so you need to move up a few layers before you place the connecting room above the base.

4. Press the "+" key three times. You will know you are at the right height when you see the roof of the room you just placed.
5. As you are still in the Prefab mode, you can left-click on the coordinates (X0, Z35) at layer 8 to place the room.
6. Press the Delete key to remove the item from the mouse cursor.

Now that you have the two rooms one on top of the other, you need to place the elevator segments. The best place to start is to add the bottom segment, followed by the topmost segment, and then fill in the gaps. Finally, place the doors for access to the elevator pad.

7. Click the Segments tab in the Library toolbar and then click Add new segment.
8. Under Scifi | Scenery is Lift Base (platform). This is the bottommost lift item, which allows for it to move up. Double left click on Lift Base (platform).
9. Press the “-” key three times to return to the room at layer 5.
10. Left-click at the coordinates (X4, Z39) at layer 5.

Now add the topmost part of the elevator.

11. Click on Add new segment icon in the Library toolbar.
12. Double left-click Lift Top. Click on the “+” key three times to move to layer 8.
13. Left-click at the coordinates (X4, Z39) at layer 8 to place the item.

Now that you have placed the top and bottom items, you need to connect them.

14. Click Add new segment and then double left-click to select Lift Tube. Press the “-” key twice to move down to layer 6. Left-click on the coordinates (X4, Z39) at layer 6.
15. Press the “+” key once to move to layer 7. As the mouse cursor still has the Lift Tube attached, left-click at the coordinates (X4, Z39) at layer 7.

The elevator parts are now in place. If you run the game you will see the lift in the bottom corner of the room, but there is no way to access it, as shown in Figure 19.2.



FIGURE 19.2 The elevator segments but no access.

You need to add a door to the bottom and top lift segments to allow the player to enter and exit it. The elevator is automatic, so as soon as the player moves into the pad it will move the player up on its own. Therefore, once the doors have been added, you will have completed the elevator.

16. Click Add new segment, and then select Door Telescopic (auto) by double left-clicking on it.
17. Press R twice to rotate the segment and make the arrow point down.
18. Left-click on the grid at coordinates (X4, Z38) at layer 5.
19. Press the “+” key three times, and you should be at layer 8.
20. Left-click on the grid at coordinates (X4, Z38) at layer 8.

You have now added all of the components to create an elevator and allow access to it for the player. Run the game and try the elevator, as shown in Figure 19.3.



FIGURE 19.3 Inside the new elevator.



If you want to compare your version with the one on the DVD, you can find it in the file `liftcomplete.fp` in the `FPSFILES` folder.



At the time of writing, enemies cannot move into the elevator with the player, so this is a good way of getting away from the enemy.

Teleporters

A teleporter is a way to move a player from one location to another. If you have seen the TV show *Star Trek*[™], you know they use a teleporter to move from the ship to other locations, for example, to another ship or to a planet’s surface. A teleporter

is another way of adding something a little more fun and different to allow your player to move around the game world. To use teleporters, you create a start teleporter and an end teleporter. The player is automatically placed at the end teleporter when he walks into the start teleporter.



You will use the file you created for the elevators. If you have already closed it, you can find it on the DVD in the FPSFILES folder. The file is called liftcomplete.fpm.

1. Ensure that you are at layer 5 of the map.
2. First, add a new room at layer 5. Ensure that the Prefabs tab is selected and then click Add new prefab.
3. Select bunker large and place it at the coordinates (X0, Z30).
4. Make sure the Segments tab is selected in the Library toolbar, and then click Add new segment.
5. Double left-click Teleporter IN. This is the starting point teleporter. The arrow on the cursor tells you which way the player can walk into the teleporter, so it is important to rotate the object if it is placed near a wall to ensure that the player can gain access to it.
6. Press R twice so that the arrow is pointing down.
7. Single left-click at the coordinates (X0, Z39) at layer 5. You should see the current layout at layer 5, as shown in Figure 19.4.

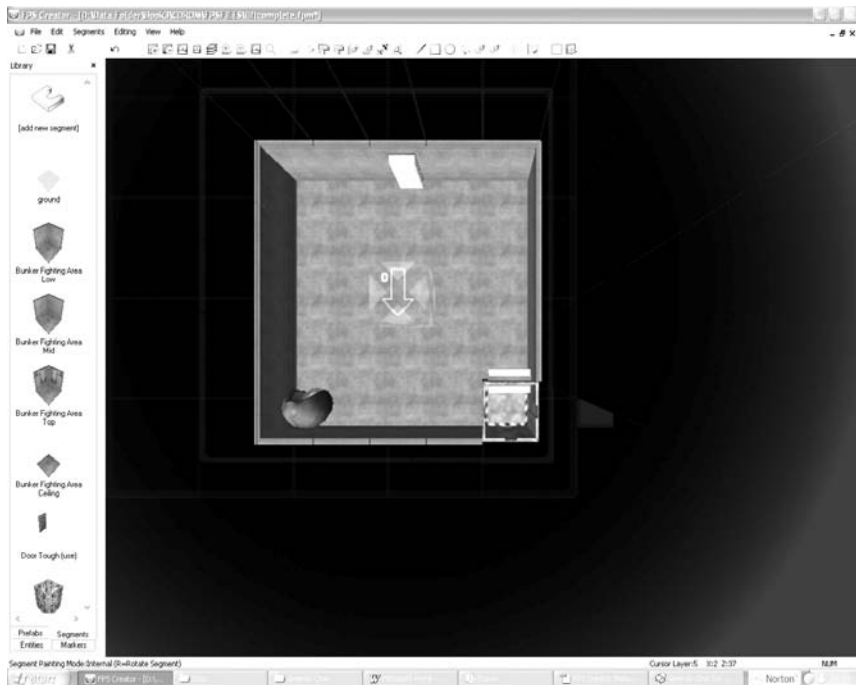


FIGURE 19.4 The current room layout of layer 5 with an elevator and a teleporter.

8. Click on the Add new segment icon and double left-click the Teleporter OUT icon. You may need to press R twice to place the arrow in a up-pointing direction.
9. Place the teleporter at the coordinates (X0, Z30). Your map should now look like Figure 19.5.

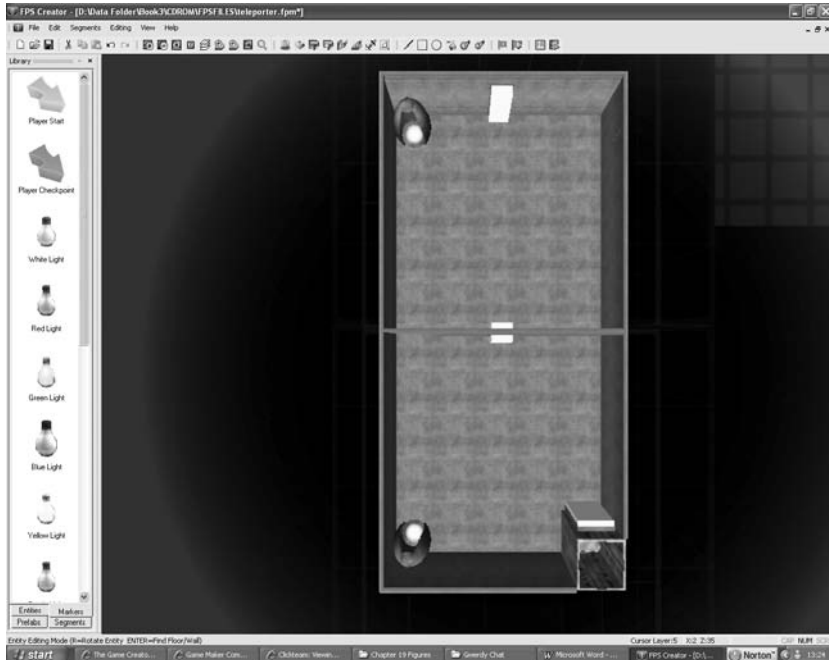


FIGURE 19.5 The two teleporters ready to be used.

If you were to run the game now you would start the game standing just outside the OUT teleporter. This is because FPS Creator places the player where it believes the player should start. You are supposed to tell the program where to start.

10. Click the Markers tab in the Library toolbar and then click Player Start. The mouse cursor changes to a green arrow.
11. Press R three times until the arrow is pointing in a south-east direction (pointing diagonally toward the elevator).
12. Click on the coordinates (X1, Z36).
13. If you run the game now, you will see the teleporter in the right-hand side of the screen and the elevator on the left, as shown in Figure 19.6.



In the demo version there is a bug, which means you cannot place a teleporter on two different layers without some strange effects. You can get around this when you are designing your game and placing your floors of the map on the grid. You can see the problem by opening and running `teleportbug.fpm` in the `FPSFILES` folder.



FIGURE 19.6 The elevator and teleporter in the same room.

CREATING ENEMY PATROLS USING WAYPOINTS

Another way to add more realism is in the form of waypoints for the computer-controlled players. This allows you to set the path a character will walk along, rather than having him stand statically in one place. If all characters in the game, including the enemies, just stood still and didn't interact with the world in which they existed, the game would not be rated as highly as another game that is exactly the same but with this interactivity included. Many games use waypoints or other methods of making their NPC's (nonplayer characters) move around the world. Great examples of this are Grand Theft Auto™ 4 and The Elder Scrolls® Oblivion™. In Grand Theft Auto characters walk along the streets and shout things out if you walk into them. The Elder Scrolls adds even more complexity by giving characters a daily movement path, whereby they move between their homes and work and do tasks. They stop on the way and talk to other characters. Waypoints aren't as powerful as the systems utilized in the two games just mentioned, but they can still make your games appear more realistic.

To create a waypoint, click on the Waypoint icon on the menu bar as shown in Figure 19.7.

You will then have a star icon on the grid, which you can then move to a position in a particular room. When you want to create another point that the player will walk to, hold down the Shift key and click on the star. This creates a new entry point that you can drag and drop into place on the grid.

You are now going to create a waypoint for a computer-controlled character, which will walk around two rooms. The rooms have been created for you and the character placed on screen. All you have to do is create character's path. For this example load the file `waypointbefore.fpm` from the `FPSFILES` folder on the DVD.

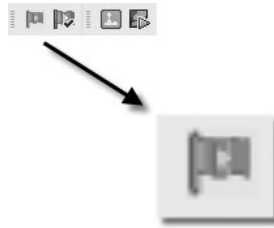


FIGURE 19.7 The Waypoint icon.



1. Open up the file `waypointbefore.fpm` in the `FPSFILES` directory.

In this file you will see four rooms connecting to each other. The player starts in the bottom-left room. An NPC starts in the top-left room. You will make the character move between the top-left and top-right rooms. You can see the layout of the file in Figure 19.8.

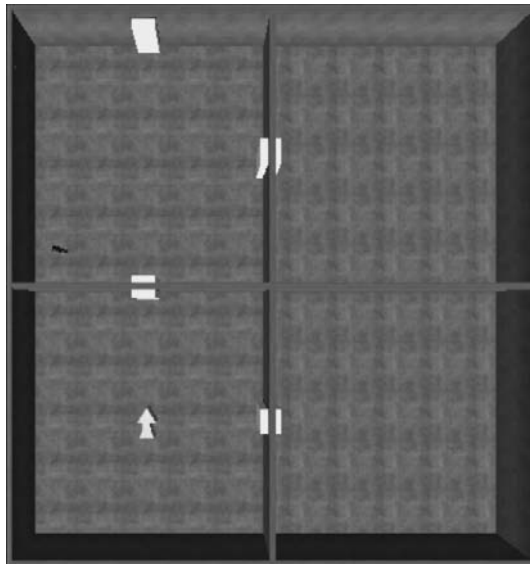


FIGURE 19.8 The room ready to add the waypoint.

2. Click the Waypoint button that was shown in Figure 19.7.
3. You will now have a colored square on the grid. Left-click on it and while holding down the left mouse button drag the star to any location you want. For this example drop it just above the computer-controlled player. You can leave it there by releasing the left mouse button.
4. You need to create another star farther up in the room. To create a second star, hold down the Shift key and left-click on the star you just placed above the computer-controlled player.

- Another star will appear on the grid. You can drag and drop this star farther up in the room. Continue to place these movement nodes on the grid to create a pattern similar to that shown in Figure 19.9.



If you put a node in the wrong place and want to delete it, click the right mouse button while the cursor is over the node you want to remove.

It is important to place a number of nodes (stars) close together around the door, as otherwise the computer-controlled player may not move correctly between the door and the next available node.

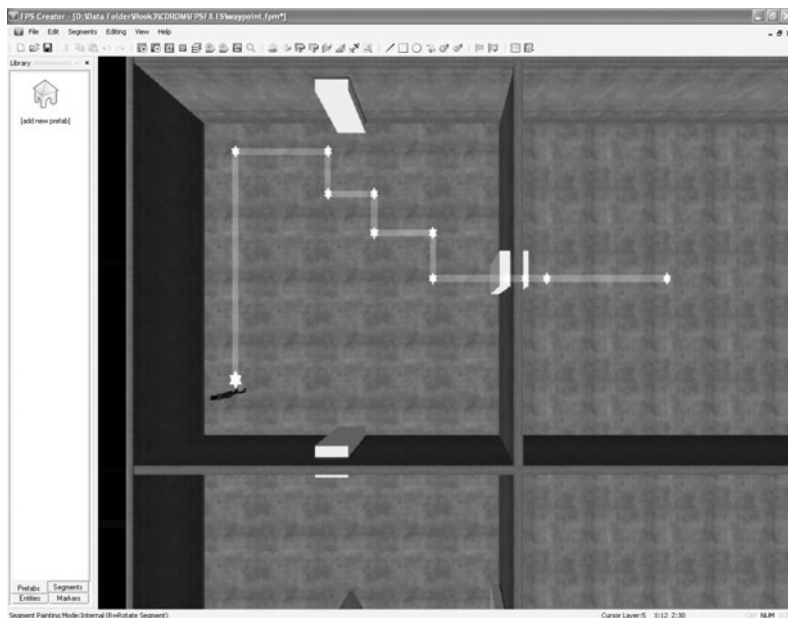


FIGURE 19.9 A path movement for the computer player.

- Run the game. You will need to walk through the door to watch the computer-controlled player walking along the path through the door and then back again. If you want to open an example file, you can load one called `waypoint.fpm` from the `FPSFILES` directory on the DVD.



ZONES

Zones are very important for checking when a player has moved into a specific area on the map. You have probably seen some of these zones in games you have played. For example in *The Elder Scrolls Oblivion* you might be walking through a cave when a rock fall happens and blocks one of the tunnels. This is effectively a trigger zone

whereby the computer continues to track the player until he moves to a specific point on the map, and it will then do something. Other zones can do things like add health, remove health, and create an action on screen. FPS Creator comes with a number of zones for you to place on the map:

Heal Zone. When the player walks into this area he receives an increase in health points. Many games have ways of getting healed other than the traditional health packs or wall health machines seen in *Half-Life* and *Doom 3*. For example, role playing games may have a fountain or pool of health.

Hurt Zone. When the player walks into this area, he will have an amount of health points removed. You wouldn't normally just place this zone and have the player's health removed, as the player would be confused as to why he is losing points. You use a hurt zone in combination with an effect. This could be fire, lava, or spikes, for example. In games like *Doom 3* you might see fire or gas coming intermittently from pipes that you have to traverse. If you time it wrong, you lose points.

Win Zone. When you want the current level to be completed, you can place a win zone. This is the final area or room you want the player to see before he moves on to the next level. You can place multiple win zones within a level to make the game less linear and add more value to the game, as the player can take different routes through the game and still complete it.



Linear is a term that describes a straight line or path. Linear and nonlinear are terms used a lot in games to distinguish between games that only allow a specific route through the game with no alternatives and games that are open ended and have different stories or routes. The more options you can provide in your game, the more interesting it will become and the more replay value it will have, meaning players will come back time and again and replay it to see if they can do things differently.

Trigger Zone. A trigger zone triggers something to happen, for example, the lights being turned on or a door unlocked. A trigger zone is controlled by a script, so you can make it more complex if you know how to write simple code. Trigger zones are very common in today's games and are the trigger to NPC's reacting and animation effects happening. In a *World War 2 FPS* you might be given the task of storming a building or a gun emplacement. When you get to a certain area, the enemy will appear or begin to retreat. This is an aspect of one type of trigger zone.

Sound Zone. The sound zone is a way to play a sound effect or music at a given moment. This is an effective way of building suspense in a game and provides a more realistic experience. If you are creating a haunted house, you could play a distant scream or the creaking of floorboards. This technique was used very successfully in the game *Vampire: The Masquerade* during one of the missions in a haunted house.

Story Zone. Many games use video to reveal the story to the players as they progress. In Call of Duty® when you get to the start or end of a level it might play a story of what's been happening. This not only gives the player an idea of what's going on, but it also helps the player become more immersed in the game. This makes the player care about the characters they interact with and provides a more movie-based emotional roller coaster ride through the game. If the player doesn't care about the characters, he may not care as much for the game. The story zone in FPS Creator allows the game designer to play video at a specific point in the game. When you place a story zone in FPS Creator, it plays a default video until you decide to replace it. It only plays the video once when the player enters the room or area.

Now it's time to create these zones to show you how easy they are to add to your games.

Hurt Zone

First, you are going to create a room that contains fire. The player needs to walk through it to reach the next room. While walking through the fire the player loses health points, so you need to add a hurt zone. There is already a file that contains the rooms and a room that contains fire. The file is called `triggerzones.fpm` and is located on the DVD in the folder `FPSFILES`.



1. Open the file `triggerzones.fpm` in the `FPSFILES` directory.
2. Seven rooms are in an "n" shape, and the second room is already filled with the fire entity. If you were to walk into this area now, nothing would happen except that you would see fire rising from the ground.
3. Click the Markers tab in the Library toolbar.
4. Select Hurt Zone, and the cursor will change to a red cube. Left-click on all possible squares in the second room, which contains the fire entity. Run the game now and walk through the room, a red haze should appear on the screen, and your health points should quickly decrease, as shown in Figure 19.10.
5. Run through the room and open the door on the other side, and the health reduction will stop.

Heal Zone

Now put a heal zone in the room directly after the hurt zone so the player can restore his health. The heal zone is added in exactly the same way as the hurt zone, so use the file you have just added the heal zone.

1. Ensure that the Markers tab is selected and then click on Heal Zone. The cursor will change to a green cube. Fill in a number of boxes in the top-left corner room.
2. Run the game and run through the fire room and into the next room. Watch your health go back up.

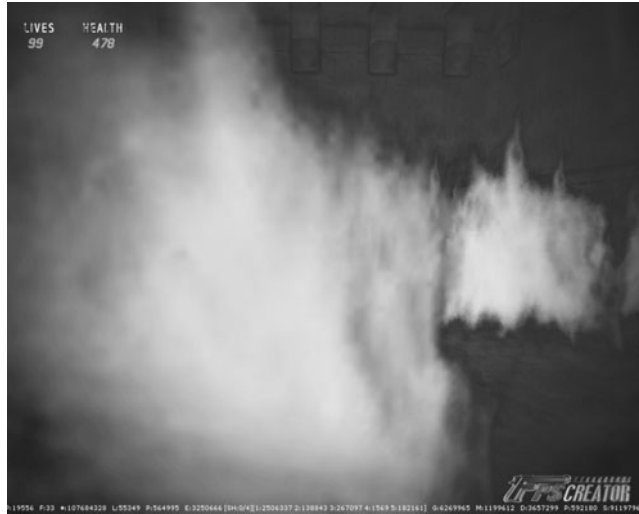


FIGURE 19.10 The health is quickly being reduced.

Sound Zone



Now play some music when the player walks into a room. Use the file you have been working on or load `healzone.fpm` from the `FPSFILES` directory on the DVD.

1. Ensure that the Markers tab is selected in the Library toolbar and then click on Sound Zone. The cursor will change to a blue cube.
2. Left-click just after the door on the top-middle room. The sound zone is currently configured as a default setting, which is to play a sound like a door opening. To change it, right-click on the cube. This accesses the object properties. You can do this process for many of the objects you have added to a game in FPS Creator, which allows you to change the scripts they use or their properties. You can see the properties for this object in Figure 19.11.
3. Under the Sound heading is Sound0 and then a file path. This file path is currently set to its default setting. Left-click on the path, and you can edit it. Clicking on the three dots that appear opens the Select File dialog box, as shown in Figure 19.12.
4. It already defaults to the audiobank folder, so navigate to the `music\generic` folder by double left-clicking on it. Single left-click on the `victory.wav` file and then click Open.
5. The sound file path in the properties is now updated. If you accidentally left-click on the map editor, it will exit out of the properties without saving, and you will need to start again with changing the file path and file it plays.
6. Ensure that the path and file is to the `victory.wav` file and click Apply Changes. This takes you back to the map grid. Play the game, walk through the fire and health zones, and then enter the next room where the music should begin to play.

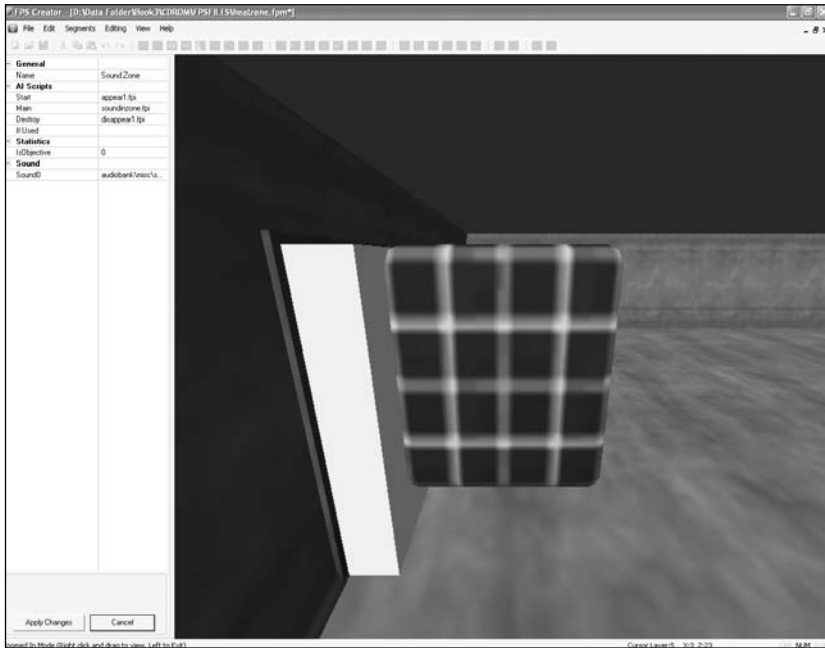


FIGURE 19.11 The properties for the sound zone.

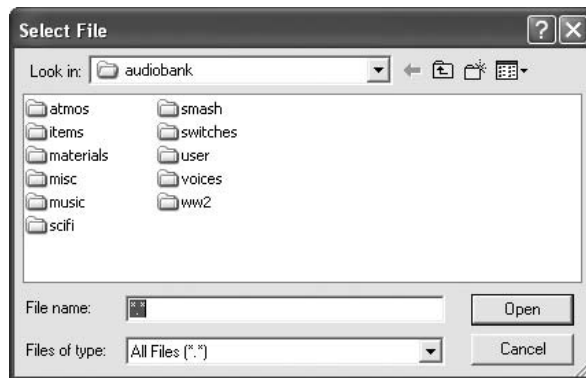


FIGURE 19.12 The Select File dialog box.



The music files provided with the demo are not very long, so try adding your own .wav files if you want to play a song that lasts longer.

Story Zone

Now create a story zone in the top-right corner room of the “n” shaped map. First, you should understand how the default story zone works.



You can either continue to use the file you have been working on or you can load the file `soundzone.fpm`, which contains all the changes you have made using zones up to now. The file is in the `FPSFILES` directory on the DVD.

1. Click on the Markers tab in the Library toolbar and then click on the Story Zone icon. Your mouse will change to a green cube. Single left-click on the top-right corner of the room in a location you expect the user to walk into. In this case place it just next to the door, so it will run as soon as the player walks into the room.



Remember that when you place a story zone or sound zone, you must place it in an area the player is expected to walk through. It would do no good to place a zone in the corner of the room if there is no reason for the player to go there. Good places are near equipment or the entrance or exit to a room or area.

2. You can see where the story zone has been placed in our example in Figure 19.13.

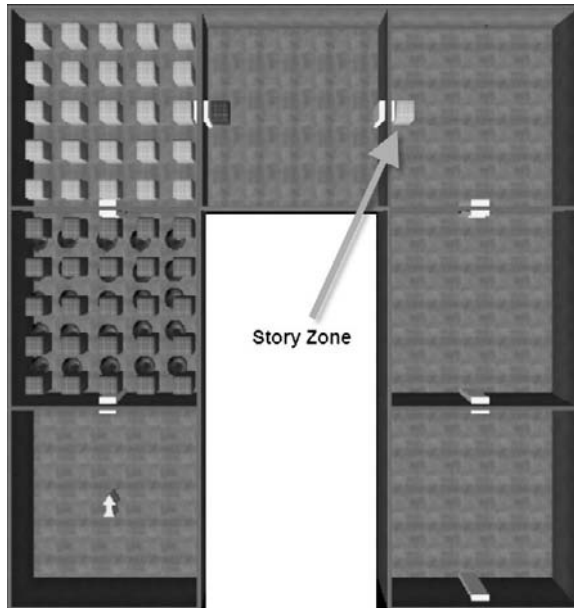


FIGURE 19.13 The story zone placed just inside the room.

If you now run the game and walk through the rooms until you get to the story zone, the game will switch from where you are controlling the player to a short video sequence. The sequence doesn't do a lot; it just plays a video of a character appearing as shown in Figure 19.14.



FIGURE 19.14 The story mode being played.

There are two parts to the story mode sequence: an AVI file, which is a standard video format, and an audio file, which can be used to provide commentary with the video. The storyvideo.avi file provided with the demo has the properties shown in Table 19.1.

Table 19.1 Video Properties of the File Provided with the Demo

VIDEO PROPERTIES

Width = 640

Height = 480

Duration 00:00:04 seconds

Frame rate : 25 frames/second

Data rate : 848 kbps

Video sample size: 24 bit

Video compression : Cinepak Codec

The table should give you an idea of what settings to create the file in. If you try to load a video file that the program does not support, when you go back in to view the properties of the story zone, the visual heading will not contain the path to the file. Additionally, you need to copy the video file you want to use in the story zone into the videobank directory, which is located under the FPS Creator Demo\Files folder. If you do not do this, when you walk into the zone nothing will happen.



If you are using a video with sound, make sure you remove the sound item from the story zone. Otherwise, the sound from the video will conflict with it.

Try adding your own video file to the program. Make sure you copy it into the correct media folder before accessing the object properties and selecting the AVI file.

Win Zone

A win zone is the final zone you would add to your completed level. In an FPS game you would create a number of levels, as the levels help break up the game into manageable chunks for the game creators and allow you to separate your story into sections. The win zone is the final act for a level, and as soon as the player walks over it, the game ends.

1. Ensure that the Markers tab is selected and left-click on the Win Zone icon. Your cursor will now change to a yellow cube.
2. Left-click once on the final room in the set of rooms you have been working on. If you want, you can open the file `storyzone.fpm` in the `FPSFILES` directory or continue using the file that you have been making.
3. Left-click in the final room at the bottom right and place a win zone. You can place one inside the door, as you know the player is likely to walk into that.

When you compile your game as an executable, it will take into account the levels in your game and will go to the next available level when the player walks into a win zone.

4. Run the game now and get your character to walk into the win zone to complete the test level.

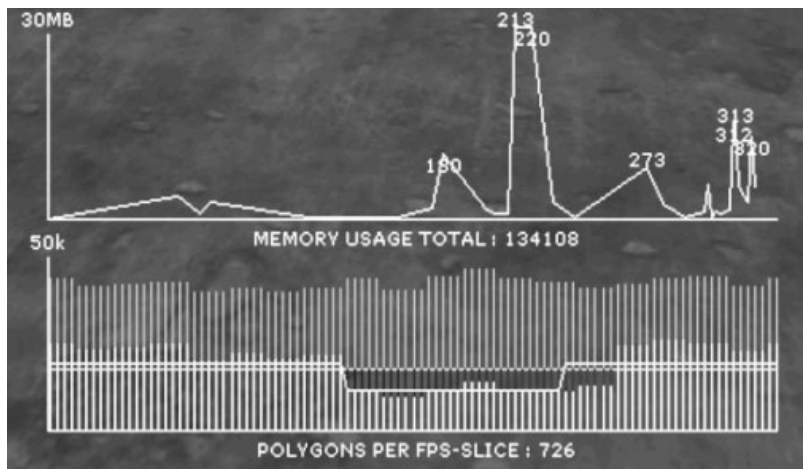
CHAPTER SUMMARY

This chapter took FPS Creator to another level, by showing you many features you can use in your games. It is important to think about how you might implement these items in your game. If they are used correctly, you can really make your game stand out. Adding sound zones, for example, can add atmosphere. Adding a story mode can explain the story and what is happening in the game.

FPS CREATOR ADVANCED OPTIONS

In This Chapter

- Performance Checking
- Building an Executable
- Creating a Multiplayer Online Game



This chapter looks at some of the more advanced features of FPS Creator. Some of these features are only available in the full version, but we will provide a quick look into what they can do and how they work.

PERFORMANCE CHECKING

You may have already noticed that if you press the Tab key while playing a test build game in FPS Creator, a graph will appear with a set of text and numbers, as shown in Figure 20.1. This is information about your game on a particular level, and it is very useful for optimizing your game to ensure that it runs as fast as possible. In FPS Creator this screen is called the Level Profiler.

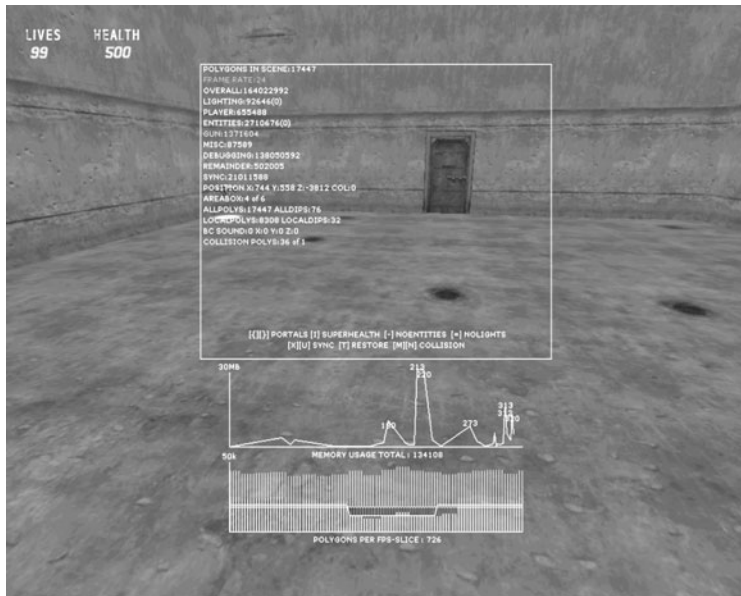


FIGURE 20.1 The Level Profiler.

Adding many objects and moving players and lights all take up valuable memory and space, so it is important to reduce the load on the computer where possible. To do this you need to understand which levels are the most resource hungry and try to minimize the impact, allowing the game to perform better on lower specification machines.

The text in Figure 20.1 is split into a number of items:

Polygons in Scene: Polygons are what make up your 3D world. The more of them you have, the longer it takes to draw and refresh your surroundings. The fewer you have, the less realistic the world looks, but the faster the game will go. Fine-tuning this can make a difference in the speed of the game without compromising the quality.

Frame Rate: This is the current frame rate of the game. If it is too slow, the game will crawl to a stop. You should aim for a frame rate close to 30 or above.

Overall: This calculates the time it takes for an entire cycle of the game to complete. You can compare this amount to other values to work out how much CPU time the game is consuming.

Lighting: This shows how much count in a cycle the lights are using. If this is too high, you can reduce the number of dynamic lights or the overall number of lights you have in a level.

Player: This shows the amount of resources being used to handle the player each cycle.

Entities: This shows the count that is being used to handle the entities you placed in your game, including artificial intelligence, collisions, and any other logic.

Guns: This displays the count in a cycle used to handle the player's gun and directions.

Misc: Any other tasks that the game processes in a cycle are displayed here.

Debugging: This shows how much in a cycle the debugging uses in the game. This is above 0 when the game is in test mode. When you build the game, it no longer needs the debug mode, and it goes to zero.

Remainder: This tells you what count is left between the debug code and the rendering to screen.

Sync: This is the amount of resources in a cycle that the game is using to re-draw the graphics.

Position: This is the player's position in the game world.

Areabox: This displays an area number where the player is standing. The game is split into invisible rectangles by FPS Creator. The first number is the box the player is in. The second number is the total number of boxes in the level.

AllPolys: This displays the number of polygons drawn in this game frame.

LocalPolys: This shows the number of polygons drawn in the invisible area box in which your player is standing.

Collision Polys: When one object collides with another, this takes some processing. This counter shows the number of polygons tested for this frame.

Also notice two graphics below these items. The first shows the amount of memory being used by the resources. There are peaks and low points in the graph, but any constant or regular peaks could signal a problem. Each item on the graph is numbered, as shown in Figure 20.2. These equate to specific tasks, and exact details can be found in the product documentation.

In graph 2 are a number of colored lines:

- The yellow line signifies the number of polygons in use within the game. At one point this dips and then increases again.
- The cyan line details the amount of time required to calculate any collisions in the level.
- The red line is the frame rate that has been achieved.

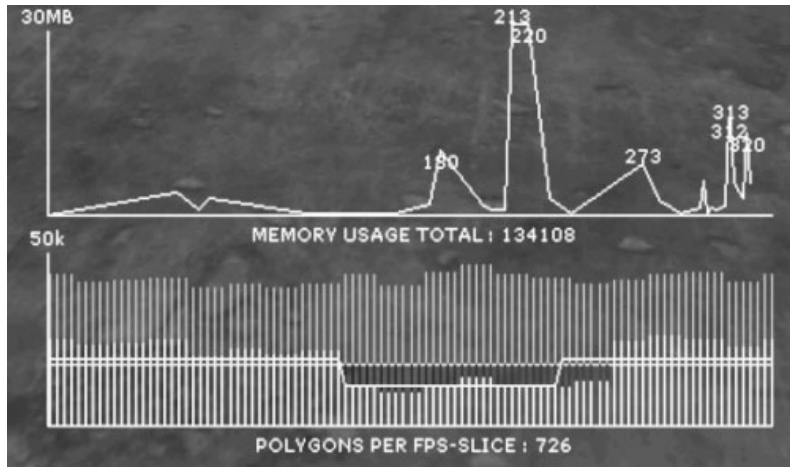


FIGURE 20.2 The Level Profiler graphs.

Using this Level Profiler, you should be able to tell which parts of your game are running slowly and the possible reasons for it. You can then remove lights and entities, reduce enemies, rerun the game, and see the results.

BUILDING AN EXECUTABLE

This section will show you how to create a final executable file in FPS Creator. You cannot do this in the demo version of the product, but it is an important step that is required if you do decide to move to the full version.

In many game creation systems when you have a game loaded into the program and you decide to create an executable of it, it takes the loaded game and produces the final result. This is not the case in FPS Creator. FPS Creator asks you for the levels stored on your disk and then builds them into a playable game.

1. To create an executable file, click on the file option, as shown in Figure 20.3, and select Build.
2. A dialog box appears, as shown in Figure 20.4, and on the Game Project Settings tab you can configure the file name, the controls the player users to move around the screen, and the keys used to change weapons.
3. In the Level settings shown in Figure 20.5 you can specify various aspects of the game properties:

Title Screen. The main menu screen, which is the first screen the player sees when playing the game. The contents of what is displayed in the screen are stored in an FPI file.

Global Script. Allows you to configure items such as the sky map, any fog and ambience levels, the lives and health panels within the game and their positions, and other items.

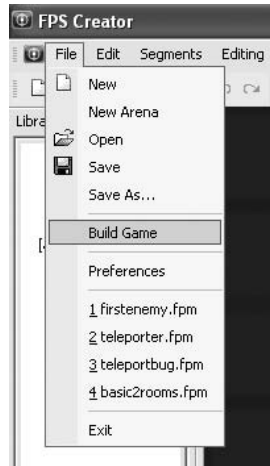


FIGURE 20.3 The Build option in the File menu.

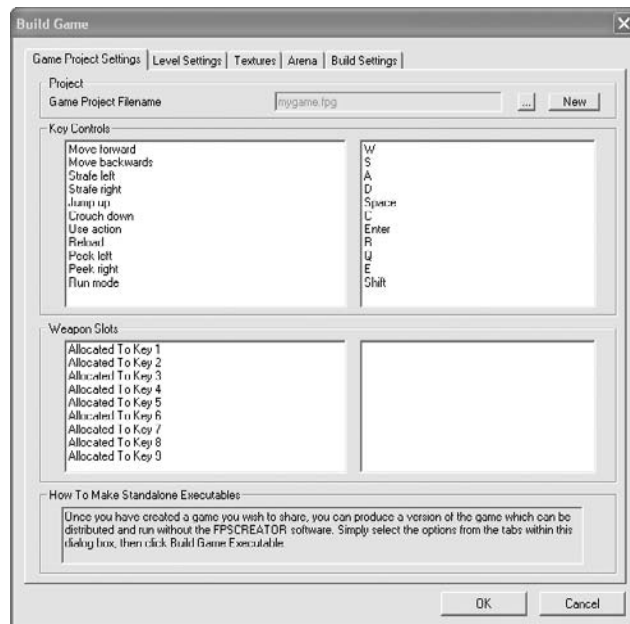


FIGURE 20.4 The Game Project Settings tab.

Game Over. The screen that appears when the game is over and the player did not successfully complete it. You can change the background image, the music, and other items.

Game Complete. The screen that appears if you successfully complete the game. The same as the Game Over screen in terms of what you can change.

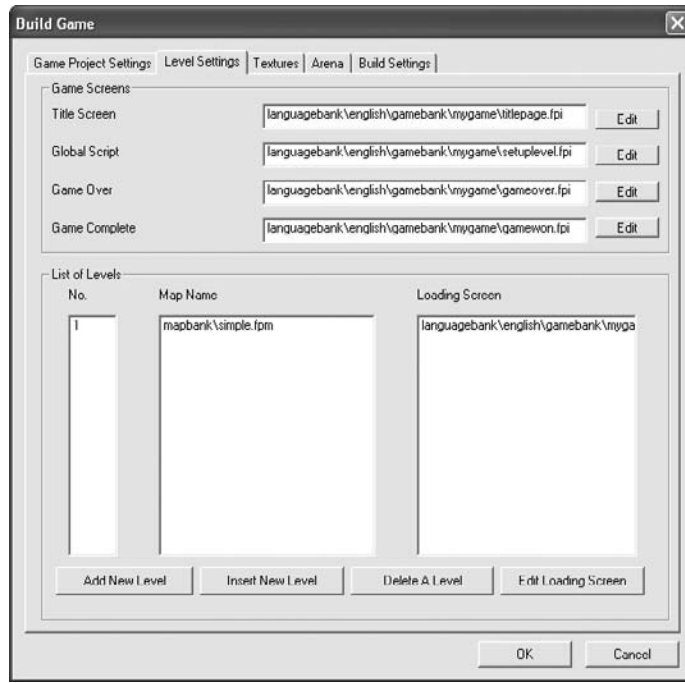


FIGURE 20.5 The Level Settings tab.

Below this you can bring all your levels together into the list of levels by using the add, insert, and delete options. You can also amend the loading screen between the levels.

4. In the Textures tab, shown in Figure 20.6, you can reduce the game file size by reducing the quality of the textures and the lighting in the game. The setting you select here will have a bearing on the overall build time when you build the executable file.
5. The Arena tab is used when you are building an online multiplayer game. You can see it in Figure 20.7. The options are grayed out when you are in the normal single-player mode. You can change into Arena mode from the File menu.
6. In the Build Settings tab, as shown in Figure 20.8, you have the final options for building your executable. In this screen you can amend the location of the file and the filename. You can also make changes to the visual aspect of the game. To build the executable, click the Build Game Executable button.



In very large games with many levels the build may take a long time to generate the executable file.

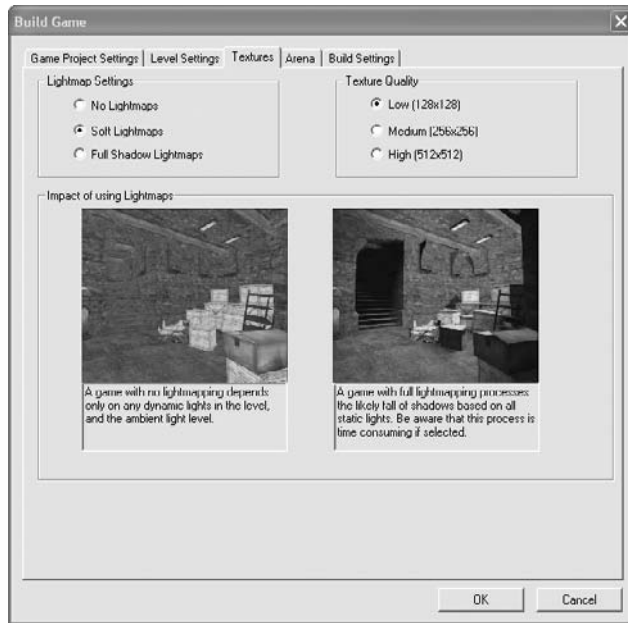


FIGURE 20.6 The Textures tab.

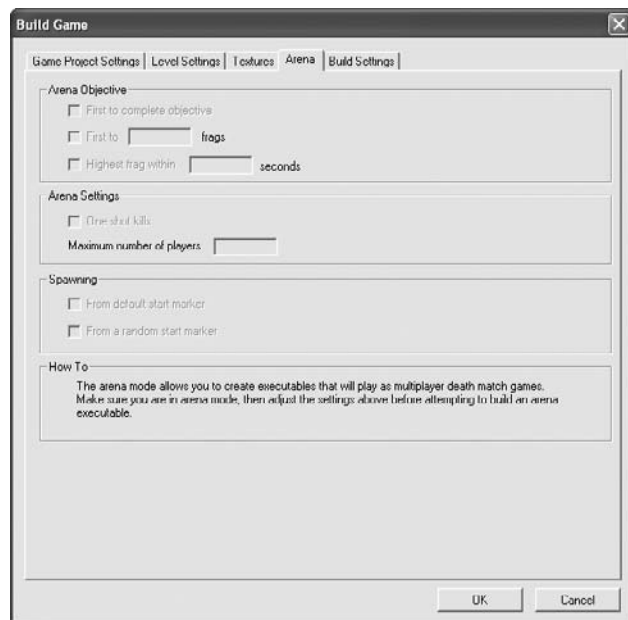


FIGURE 20.7 The Arena tab.

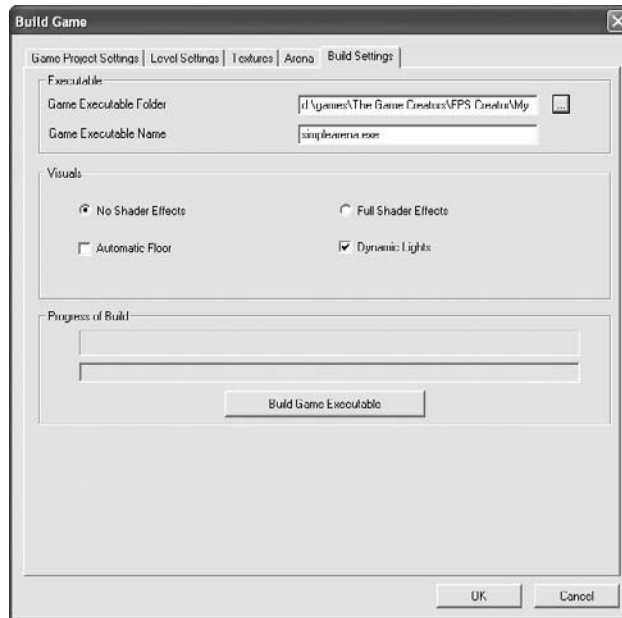


FIGURE 20.8 The Build Settings tab.

CREATING A MULTIPLAYER ONLINE GAME

When you have completed a single-player game and given it to your friends and family to play, you might want to play a game you have created with them playing along. You can do this with FPS Creator by using the multiplayer aspect of the product.

1. The first thing you need to do is change modes from single-player to Arena mode. To do this, select the File New Arena option. You can tell when you are working in Arena mode, as the background color is a dark red rather than the black used for single-player mode.
2. If you now look in the Preferences (accessible from the File menu option), you can see the Single Player or Multiplayer radio button. Ensure that the Multiplayer is selected.
3. You would then create your maps.



Not all features are available in Arena mode, as this is a different type of game than the single-player mode, so some features are disabled. Consult the user guide for further details on which features are available in single-player mode and multiplayer mode.

4. Select the Build Game menu option and follow the same process as you would if you were building a normal executable. This time, when you click on the Arena tab, you can select various options, as shown in Figure 20.9.

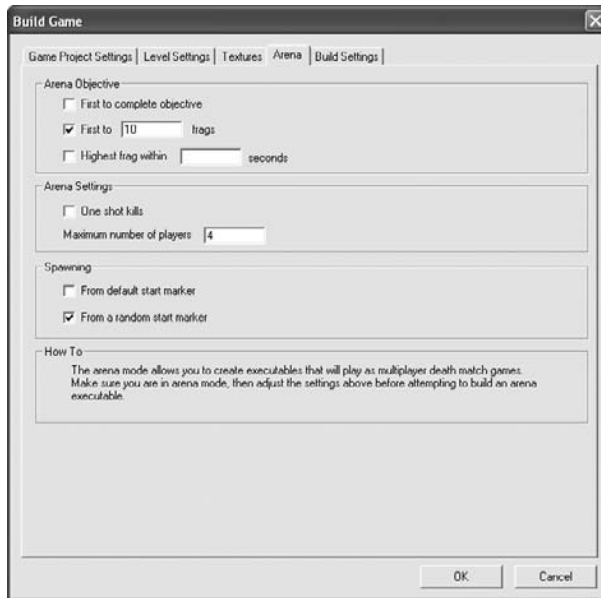


FIGURE 20.9 The Arena tab with selectable options.

You can change the winning objective of the game to either getting to a specific location, getting first to a certain number of frags (kills), or being the one with the highest number after a specified time limit. You can also configure it so that one shot hitting a player will kill him, and you can configure the number of players. Finally, you can spawn (make people appear) at a start marker or a random location on the map.

Once you have built your executable, you can distribute it among your friends, on the Internet, or via file sharing systems. To get involved in a multiplayer game, people who want to play in the same game would start the executable. One of the players would become the host (server), while everyone else would be a client. This means the host is the central hub of all communication between the players, so ensure that this machine is on a fast connection link and preferably a fast PC. All machines must be networked either locally or connectable over the Internet. If you are using the Internet you must ensure that any firewall software you are running allows the software to be the host and client and access and transfer data between them. Consult your firewall software for more information on this if you have problems.

5. Double left-click your executable file, as shown in Figure 20.10.
6. The menu screen will load and will ask for your name, as shown in Figure 20.11. This is the name you will be known as in the game. You can use your real name or a made-up one.

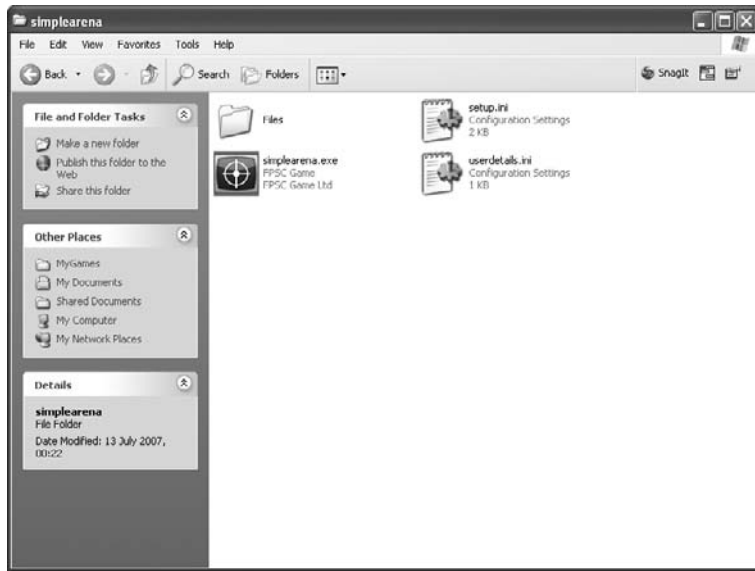


FIGURE 20.10 The arena game files.



FIGURE 20.11 Asking for your game name.

7. You can now see the options screen shown in Figure 20.12. This is where you select if you are a host or if you will join a game. In this example, Host has been selected, and the game will begin to load as shown in Figure 20.13.
8. Next, you are asked to select the character you want to play in the game, as shown in Figure 20.14.



FIGURE 20.12 The options screen to create a game or connect to one.

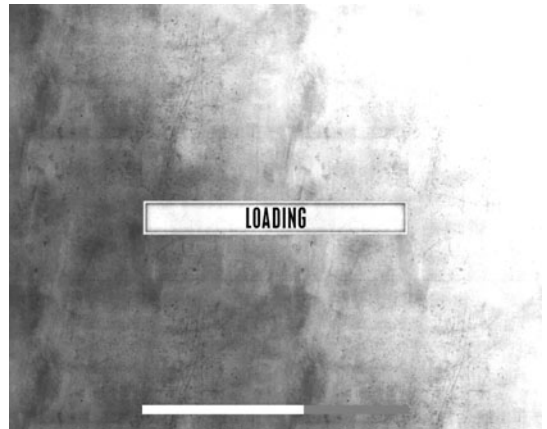


FIGURE 20.13 The game starts to load.



FIGURE 20.14 Select the character you want to play in the game.

9. You should now be in the game, as shown in Figure 20.15, waiting for your friends to join you for a game.



FIGURE 20.15 The game in progress.

CHAPTER SUMMARY

In this chapter you learned how to check the performance of your game, how to create an executable file that you can play on any PC or give to your friends, and how to create a multiplayer FPS game.

When creating your games, you need to make sure they are interesting and fun but also that they perform at a reasonable speed. If they don't run at a reasonable speed, you may find that the fun and interesting points are lost very quickly. Use the Level Profiler to find aspects of your game that are not running as well as they should and then tweak and change your game to make it run as well as it can without impacting on the game's structure.

Though FPS games are a lot of fun, it is even more fun playing them online with your friends. Create different settings and maps and then challenge your friends to a weekly competition to see who is the best FPS player.

This is the last chapter covering FPS Creator. Now you will move on to The 3D Gamemaker for a whirlwind tour of an easy-to-use software tool that might give you some ideas for FPS games.

THE 3D GAMEMAKER

In This Chapter

- System Requirements
- Installation
- Creating a Game with The 3D Gamemaker
- Saving the Game
- Playing the Game



Of all the tools we have looked at, perhaps none is quite as easy to use as The 3D Gamemaker. Though it is the oldest of the tools available, it still provides an easy way for the game maker to create a simple game in minutes. It also provides a good way to see the range of games you might want to create from different genres, so even if you don't use it as a tool to develop programs, it is useful to get the creative ideas flowing.



A demo version of The 3D Gamemaker is on the DVD. This version has much of the functionality of the standard version without the graphic content. You cannot make a distributable game with it. If you would like to purchase the full version, visit www.thegamecreators.com or www.focusmm.co.uk. You can find the demo version of the software in the Demos\3DGM folder on the DVD provided with this book.



ON THE DVD

SYSTEM REQUIREMENTS

The minimum system requirements for The 3D Gamemaker are:

- 400 Mhz Pentium II Processor
- Windows 95, 98, 2000, ME, or XP
- 600 MB of hard disk space
- 64 MB of RAM
- DirectX version 7.0a
- Fully DirectX-compatible graphics card with 3D acceleration and 8MB of memory
- DirectX-compatible sound card
- 4x speed DVD drive

The recommended system requirements are:

- 600 Mhz Pentium III Processor
- Windows 95, 98, 2000, ME, or XP
- 600 MB of hard disk space
- 128 MB of RAM
- DirectX version 8.0a
- Fully DirectX-compatible graphics card with 3D acceleration and 16 MB memory
- DirectX-compatible sound card
- 16x speed CD-ROM drive

INSTALLATION

To install The 3D Gamemaker:



ON THE DVD

1. Put the DVD that accompanies the book into your DVD drive.
2. Double-click My Computer and then the letter of your DVD drive.
3. Double-click the Demos\3DGM directory. Then double-click on the file Setup.exe, which is the installation file for this program. You'll see the screen shown in Figure 21.1.

- From the Welcome screen, click Next. You will then be presented with the License Agreement screen, as shown in Figure 21.2.

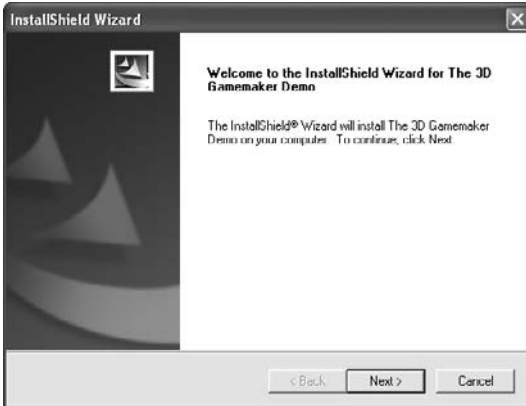


FIGURE 21.1 The opening screen for installation.



FIGURE 21.2 The License Agreement dialog box.

- Read the license information and then click Yes to continue.
- The Setup dialog box appears next, as shown in Figure 21.3. You can only select the typical setup option, so click Next.
- You can then select a destination directory, as shown in Figure 21.4. You can accept the default or click Browse to select a new folder. When you are ready, you can click Next.



FIGURE 21.3 Typical installation option.



FIGURE 21.4 Typical installation dialog box.

- The program folder name dialog box will appear. Unless you want to change the name of the folder, as shown in Figure 21.5, click Next.

9. You should now see the dialog box shown in Figure 21.6. This sets out the installation program information, including the destination directory and how much space is required.



FIGURE 21.5 The program folder's name.



FIGURE 21.6 The installation information.

10. Click Next to continue with the installation if you are happy with the information. If not, click Back and change any of the information.
11. The files get copied to your machine.
12. You are asked if you want to add a shortcut icon to the desktop, as shown in Figure 21.7. Click Yes.
13. Next you see a confirmation dialog box as shown in Figure 21.8 to show you that the installation was successful.

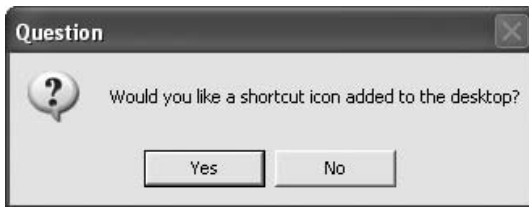


FIGURE 21.7 The Add Icon to the Desktop dialog box.



FIGURE 21.8 The installation completed successfully.

14. Click Finish to complete the installation.
15. You now have a 3D Gamemaker icon on the desktop.

CREATING A GAME WITH THE 3D GAMEMAKER

You have just walked through the steps required to install The 3D Gamemaker. Now that it's installed, you can use it to develop a 3D game. The software comes with a variety of prebuilt environments and 3D models that you can use in your creations. Some have been disabled in the demo version but are available in the full version.

Getting Started

In this section, you'll develop a simple game with The 3D Gamemaker. When you first double-click on the icon on the desktop, you will get the menu screen shown in Figure 21.9.

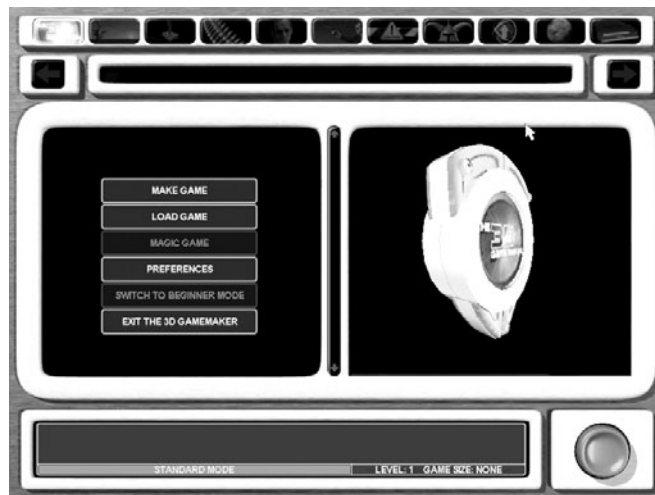


FIGURE 21.9 The main menu screen in The 3D Gamemaker.

From here you can:

Make Game. This is the menu option you use to begin putting your game together.

Load Game. If you have already created a game, you can use the load option to load it back in and continue working on it.

Magic Game. When you click on this button, the software randomly creates a game for you. This is not available in the demo version.

Preferences. Configure the preferences for The 3D Gamemaker software.

Switch to Beginner Mode. There are two modes: the beginner mode and the standard mode (as shown in the demo). This basic mode takes away much

of the complexity and makes the interface easier to use for beginners. The beginner mode does not exist in the demo version.

Exit the 3D Gamemaker. Exit the program.

1. Select the Make Game option.
2. The next screen, as shown in Figure 21.10, shows a number of tabs. The Demo tab is selected by default. These are all of the scenes available for the game creator to pick. As this is the demo version, you can only select those in the Demo tab. Take a look at the other tabs to see what content is available to you if you decide to purchase the full version.
3. Select the scene called `_Cellar03` by left-clicking on it. This draws the scene into the right-hand window, as shown in Figure 21.11.

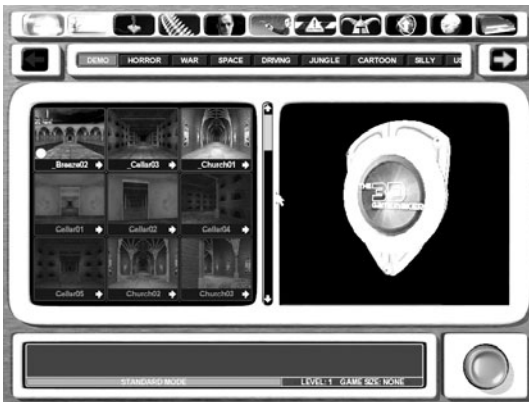


FIGURE 21.10 The current scene selection.

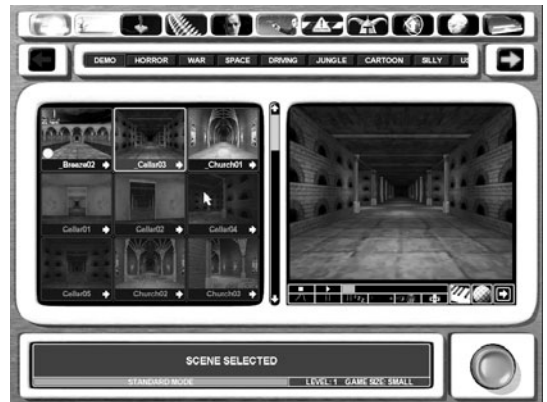


FIGURE 21.11 The scene is displayed in the right-hand window.

Under the larger picture of the scene is a position bar. You can drag this to the right to walk through the scene. You can also click on one of the three boxes to change the sounds, preferences, and textures.

Notice at the top of the screen that the second picture from the left is selected. This is the navigation bar that you use to move through the setting up of your game.

4. The second box in is selected. This is the scene window. Click on the third picture, which looks like a joystick. You can then see the Select a Player window. Choose an object to be the player character. The player character is the player that will be controlled by the person playing the game.
5. Select the `_Crossbow` object and see it being displayed, as shown in Figure 21.12.
6. Click on the next picture in the navigation bar. This is the Player Bullet screen. Here you need to pick a bullet to fire from either a weapon or object, for example, a tank. In this case the player will be using a crossbow, so select the arrow like object called `_Quarrel`, as shown in Figure 21.13.



FIGURE 21.12 The Select a Player window.

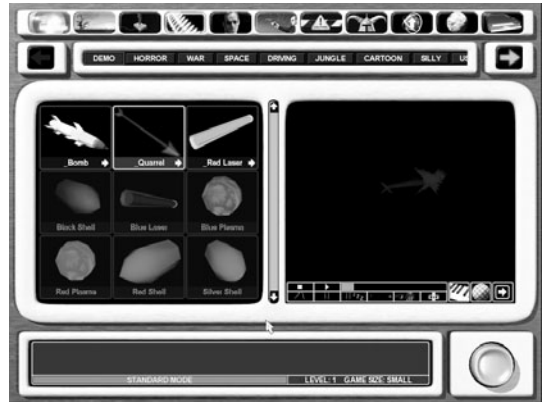


FIGURE 21.13 Picking the arrow object on the Player Bullet screen.

7. Click on the next tab, which is the Game Enemies screen. Here you pick what characters you want to play the enemies. You can select more than one enemy character, as shown in Figure 21.14, where the secret agent is the first, the mutant is the second, and the happy dude is the third.
8. Click on the next picture in the menu bar to select the Enemy Bullets screen. Here you choose the bullets that the enemies will fire at you. In Figure 21.15 the lightning object and the brick are selected. When you are creating a more specific game, you would keep the items in context. For example, if it was a horror game you might select the pumpkin and lightning objects, but not a striped shell. This also applies to the characters and the scene.

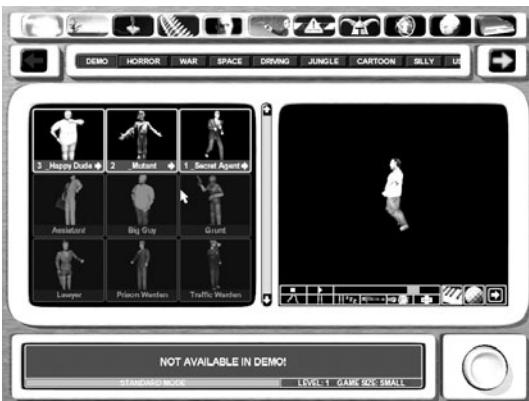


FIGURE 21.14 The Game Enemies selection screen.

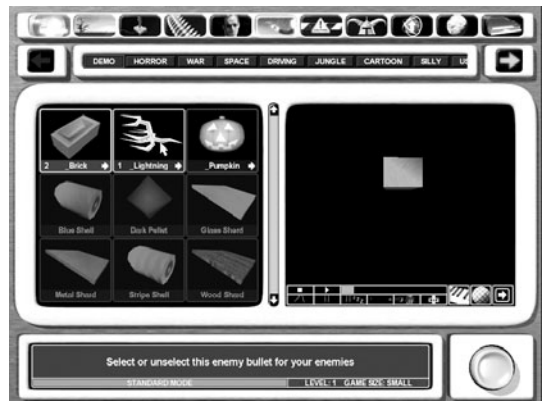


FIGURE 21.15 The Enemy Bullets selection screen.

9. Click on the next picture to access the Game Obstacles selection screen. These are items that will be placed on screen to block your access to rooms or the road. You can destroy them by shooting at them in this game. All three obstacles that are available in this demo are selected in Figure in 21.16.
10. In many games when you get to the end of a level, you have to fight an end of level boss. You can do this in The 3D Gamemaker by clicking on the next picture in the tab, the End of Level Boss selection screen. Select an end of level boss. In Figure 21.17 the Robot is selected.



FIGURE 21.16 The Game Obstacles selection screen.



FIGURE 21.17 The End of Level Boss selection screen.

11. Click the next picture, which is for Game Items. In this screen you can select from different game items. In many games these are called collectibles. The list also contains extra energy, life, and strength items for the player to pick up. In Figure 21.18 all three items that are available in this demo are selected.
12. Now click on the next picture, which is for setting the global game settings, as shown in Figure 21.19. In this screen you can configure the game settings:
 - Objectives, or the aim of the game
 - How difficult it is to play the game
 - Number of players?
 - Volume levels
 - Font styles and type
 - The appearance of the game
 - The title screen layout—the main menu
 - Highscore layout



As you can see from Figure 21.19, the only option available in the demo is to change the menu screen. All other features are disabled in this version.

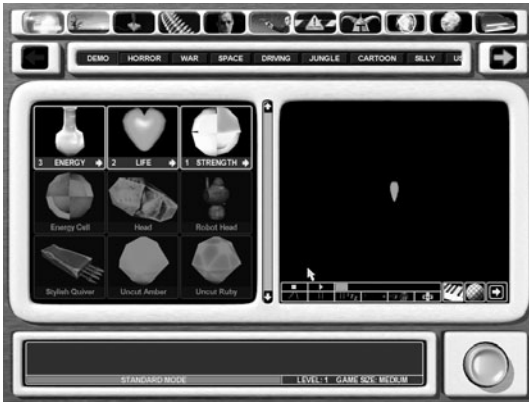


FIGURE 21.18 The Game Items selection screen.



FIGURE 21.19 The Global Settings screen.

SAVING THE GAME

After making all of the selections, it's time to save the game. Click the icon that looks like a disc in the upper-right corner of the screen. A menu will appear with four options: Play, Load, Save, and Standalone (which is disabled in the demo version). You can see this menu in Figure 21.20. Choose Save Game.

The Save a Game menu is displayed next. You can name your game anything you'd like. In the example shown in Figure 21.21, it is called FirstGame.3gm (The 3D Gamemaker adds the .3gm extension).



FIGURE 21.20 Choose Save Game from the menu.



FIGURE 21.21 The Save a Game menu.

PLAYING THE GAME

After you save the game, the original menu is again presented (see Figure 21.21). To play your game, click Play Game.

1. When you execute the game, you see the opening screen. This screen, seen in Figure 21.22, is the screen you could have edited, but didn't, earlier.
2. Press the Space Bar to begin the game. The game will start to load, as shown in Figure 21.23. When it is done loading, you'll see the game level, as shown in Figure 21.24.
3. Go ahead. Test the game. You'll see the game items you selected from the menu system in Figure 21.25 and the end of level boss in Figure 21.26.



FIGURE 21.22 The opening screen of the game.



FIGURE 21.23 The game is being loaded.



FIGURE 21.24 The level is displayed.



FIGURE 21.25 Game items placed on the game screen.

If you complete the game successfully or if you fail your quest, you receive an onscreen message, as shown in Figure 21.27. If you get a high score you are asked to enter your name, shown in Figure 21.28. Finally, you see the results on the high-score table in Figure 21.29.



FIGURE 21.26 The end of level boss.



FIGURE 21.27 The congratulations message if you get to the end of the game.



FIGURE 21.28 Enter your name when you get a high score.



FIGURE 21.29 The highscore table results.

CHAPTER SUMMARY

In this chapter, you built a game using the Standard mode of The 3D Gamemaker. The built-in levels and models make it easy to build a complete project using The 3D Gamemaker. It is recommended that you play with the different features and options that are available. This should give you ideas for your own amazing games.

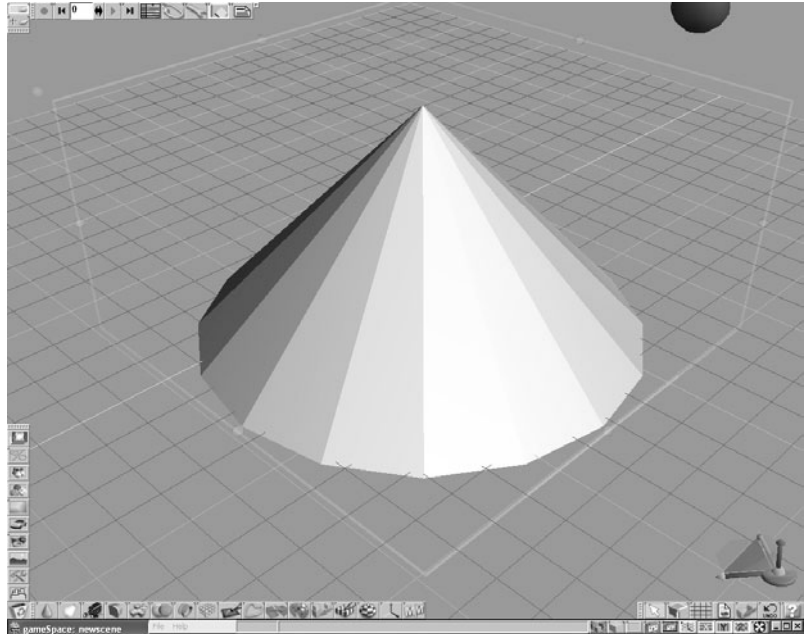
In the next chapter we will be looking at a 3D graphics creation product that allows you to make your own 3D models or export them into animated images for use in 2D game creation packages.

This page intentionally left blank

GAMESPACE LITE

In This Chapter

- System Requirements
- Installation
- gameSpace Lite Interface
- Creating Primitives
- A Simple 3D example
- Exporting the Model



Throughout this book you have been shown how to make your own sounds and sprites for your 2D games. This chapter takes a look at a product called gameSpace (logo shown in Figure 22.1) from Caligari (<http://www.caligari.com/>), which allows game creators and hobbyists to create 3D graphics.



FIGURE 22.1 The gameSpace logo.

For this book you are going to use the Lite version. This is more than just a demo version and is capable of creating good 3D models. It has a few limitations, but if you want to create larger objects and scenes, you can easily upgrade to the full gameSpace product at a very competitive price. If you want to create professional game level objects you might want to purchase Truespace 7.5.

You can export any models created in gameSpace to X format; this can be imported and used in FPS Creator. You may be wondering if gameSpace Lite can be used if you decide to start making games in TGF2 or Game Maker, and the answer is unfortunately “no.” You would need to use the full version of gameSpace 1.6, Truespace, or another 3D package that can export to bitmaps and then use them within your 2D creations. Some people find creating images of characters easier in a 3D package than drawing in a 2D drawing package, so you can still use these in your programs, but remember, it will still be only 2D.

An example of a set of graphics created in a 3D package and exported to BMPs for use in TGF2 can be found on the DVD provided with this book in the folder 3dTO2D. Run the program called MagicMaths.exe. When the animation has completed, you will see an elf drop from the sky onto the game logo. This 3D-looking elf was created in a 3D package.



SYSTEM REQUIREMENTS

The system requirements for gameSpace Lite are

- Windows 98, ME, NT4, 2000, or XP
- AMD Athlon or Pentium 120 (P4 or AMD K7 recommended)
- 64MB RAM (128 or more recommended)
- 50 MB free hard disk space
- 3D video card with at least 16 MB of video memory and D3D or OGL drivers

INSTALLATION



The installation file for gameSpace is located in the \Demos folder on the DVD provided with this book and is called gSLight_PP.exe.

1. Double left-click on the file gSLight_PP.exe to begin the installation.
2. You will be presented with the Welcome dialog box, as shown in Figure 22.2. Click Next to continue with the installation.
3. Now you will see the License Agreement screen, as shown in Figure 22.3. Read the details and usage information for using this product. When you have read this and are ready to continue, click the I accept the agreement radio button. If you do not want to accept the agreement, you can select the other radio button, but you will not be able to continue with the installation.



FIGURE 22.2 The Welcome dialog box.



FIGURE 22.3 The License Agreement dialog box.

4. Ensure that the I accept the agreement radio button is selected and click Next.
5. You now have to select a location to install the product, as shown in Figure 22.4.
6. You can either stay with the default selection or browse for an alternate location. Once you are ready, you can click Next.
7. If you want to reduce the size of the installation, you can unselect some of the options shown in Figure 22.5. The installation size is pretty small, so it is recommended that you leave all boxes selected. If you want to reduce the size of the installation, click on the drop-down box and select the installation type. Click Next to continue.
8. You will be advised of the starting location of the program shortcut files, as shown in Figure 22.6. The default should suffice in most cases, so click Next.

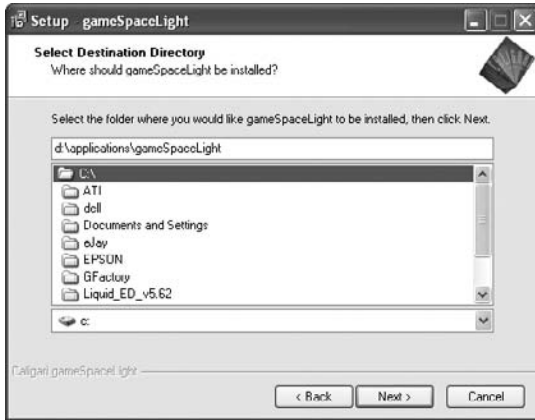


FIGURE 22.4 Selecting where the program files will be placed.

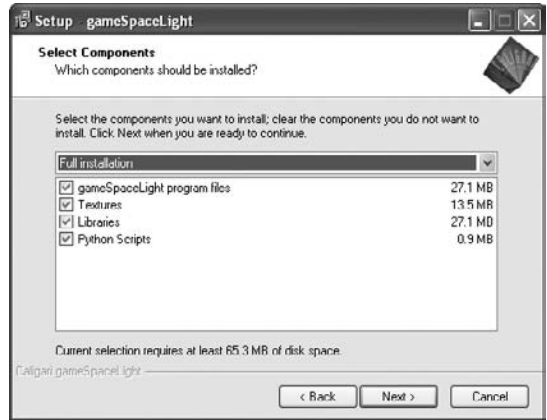


FIGURE 22.5 gameSpace components.

9. You will be asked if you want to create a desktop icon and a quick launch icon and given additional options to choose from. Leave the first two selected, as shown in Figure 22.7, and click Next.

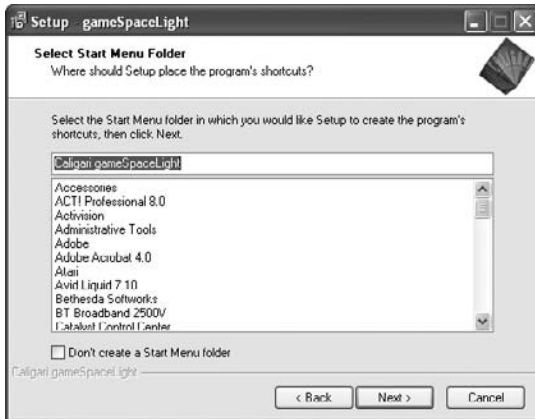


FIGURE 22.6 The shortcut and menu folder dialog box.

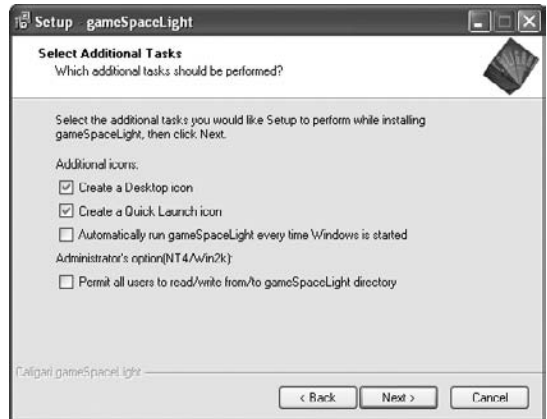


FIGURE 22.7 Additional options.

10. You will be given a final list of installation settings, as shown in Figure 22.8. If everything is correct, click Install. If you want to change anything, click Back and make the relevant changes.
11. These are the settings you want, so click Install. The program files will begin to be installed onto your computer.
12. Once the installation is complete, a final dialog box, as shown in Figure 22.9, will advise you that the install has completed, and a help Web page will open, advising you what to do next (Figure 22.10).



FIGURE 22.8 Installation settings dialog.



FIGURE 22.9 The installation has completed dialog box.

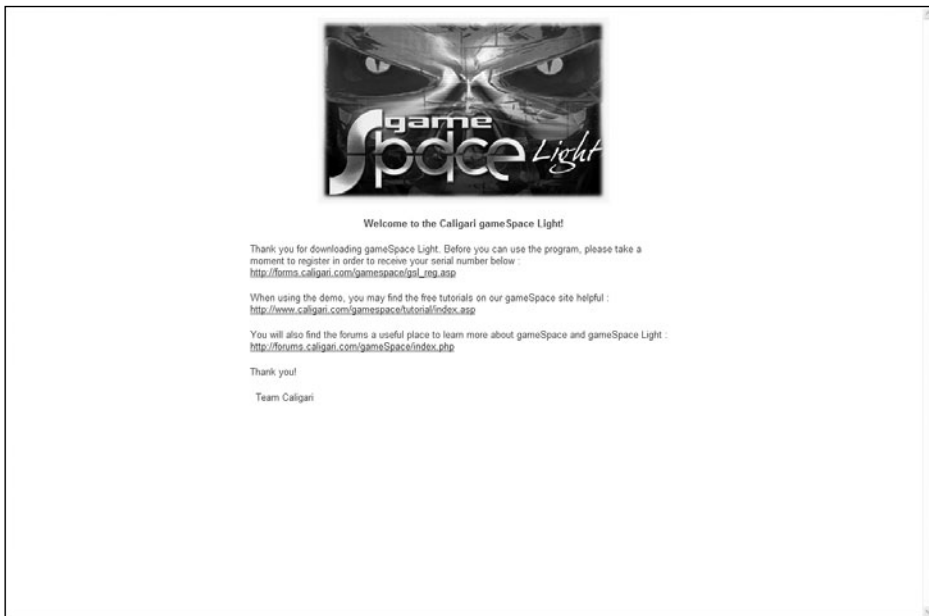


FIGURE 22.10 The welcome help page.

13. Click Finish to close the dialog box, and a text help file will appear. This provides additional information about the requirements of the product and its capabilities. Once you have read this document you can close it.
14. An icon will appear on your desktop. Double-click on it. This opens up the dialog box shown in Figure 22.11, which does not allow you to proceed until you have registered your product and received your serial key.

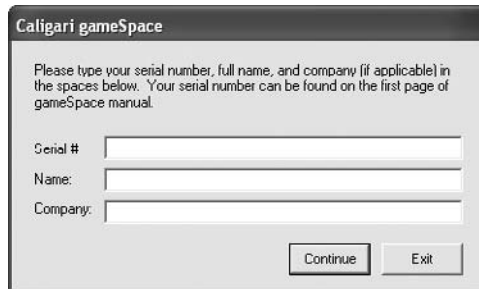


FIGURE 22.11 The serial key dialog box.

15. To register, you can click on the link provided in the HTML help page or visit http://forms.caligari.com/gamespace/gsl_reg.asp. Fill in your details on the Web page and you will be emailed a serial key.
16. Once you receive your email, which contains the serial key, enter it into the dialog box in Figure 22.11 and click Continue. gameSpace now loads, as shown in Figure 22.12.

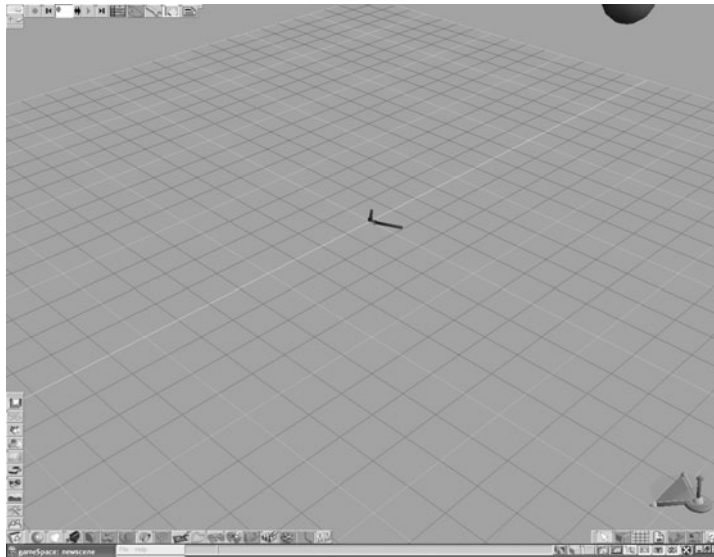


FIGURE 22.12 The gameSpace window.

GAMESPACE LITE INTERFACE

Before we look at the basic modeling features of gameSpace, you need to take a tour of the interface so that you can understand where particular features and options are located. Figure 22.13 shows some of the areas numbered so you can pick out the important options.

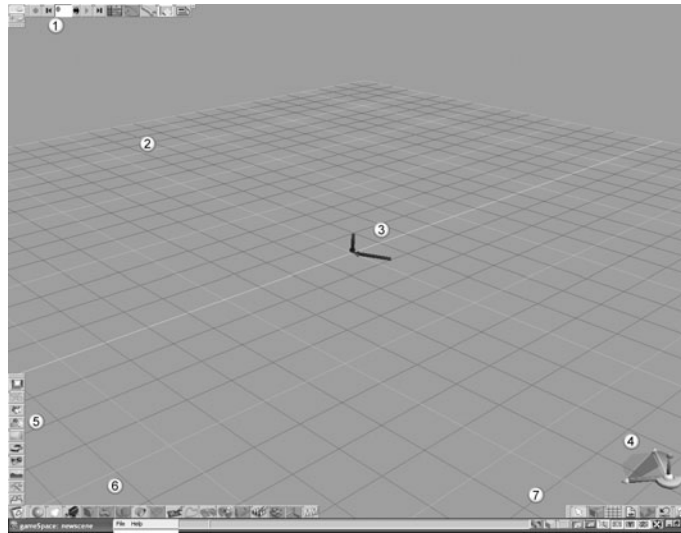


FIGURE 22.13 Some of the options of gameSpace.

- 1:** The top-left corner options handle additional layers, recording of objects moving, and creating of keyframes, which allow you to create animated movies. You also have access to an advanced scene editor so you can set up a scene using a timeline. There are also options to turn on collision detection and create script to handle how the 3D world you create will react.
- 2:** This is your 3D plane, where you place all of your 3D created objects and items to make up your images and models. You can change the view from this single view (called standard) to four views, which creates four windows and displays your models from four different angles. Many people prefer this view, as it gives them a better understanding of the model from multiple directions.
- 3:** These two arrows show the direction of any lights that have been placed on the plane. By default, there are two lights on the plane at the start of the creation, and you can move these or add additional ones as you need. Adding lights increases the luminosity on a particular side or angle of an object, depending on where the light has been placed.
- 4:** This is the move control and allows you to easily move your 3D objects. Hold down the left mouse button over one of the direction controls and move the mouse to move the object in that particular direction.
- 5:** This allows you to view images and 3D models that have already been created, as well as lights, scenes, primitives, and already created projects.



A primitive is a basic shape that you can use to build up your model, for example, a cube or a sphere.

- 6: This allows you to access the various drawing tools for the program, including what primitive shapes to draw and adding text, lights, and cameras.
- 7: This configures options for rendering the output of the object, the display of the plane, and other configuration options.

gameSpace is a very powerful program, and has many options and configurations. To find out more, we recommend you consult the extensive help files or visit the gameSpace product Web pages at www.caligari.com/gamespace/.

CREATING PRIMITIVES

Primitives are the basic building blocks of any 3D model. By placing a primitive on the plane and then pulling it, squashing it, cutting it, and sticking primitives together, you can create a wide range of shapes.

In the bottom-left corner of the screen, next to the recycle bin you can see a shape. If you left-click and hold on this shape, a selection of objects will appear, as shown in Figure 22.14.

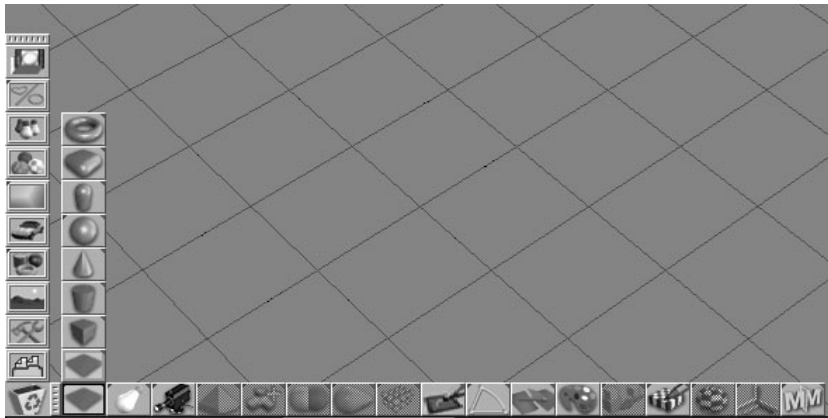


FIGURE 22.14 The various primitives you can use.

Click on one of the primitives to select it. Your mouse cursor allows you to click on the plane and create the primitive. You can make the primitive shape larger or smaller by holding the left mouse button and dragging it on the plane.

There are nine primitives to choose from:

- Sphere/Geosphere
- Plane
- Cube

- Cylinder
- Cone
- Rounded Cylinder
- Rounded Cube
- Torus

You can see what each one looks like in Figures 22.15 to 22.23.

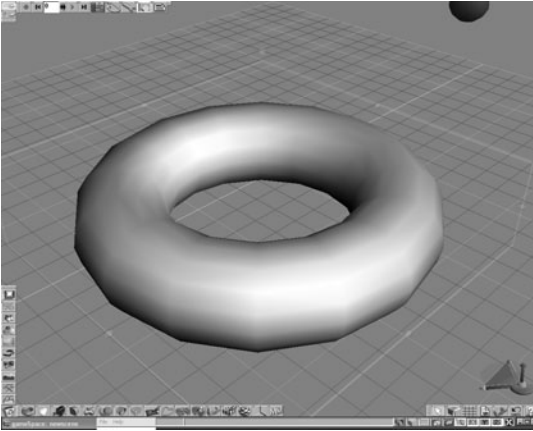


FIGURE 22.15 A Torus Primitive shape.

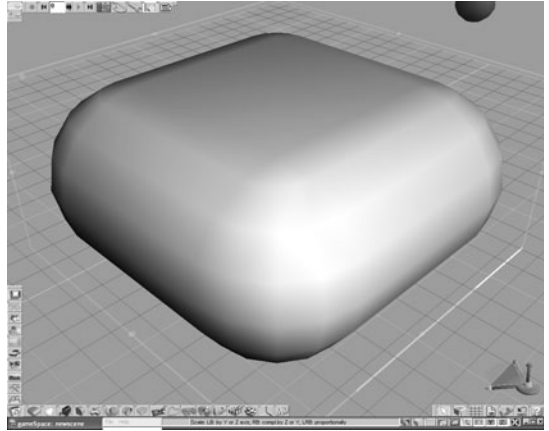


FIGURE 22.16 A Rounded Cube Primitive shape.

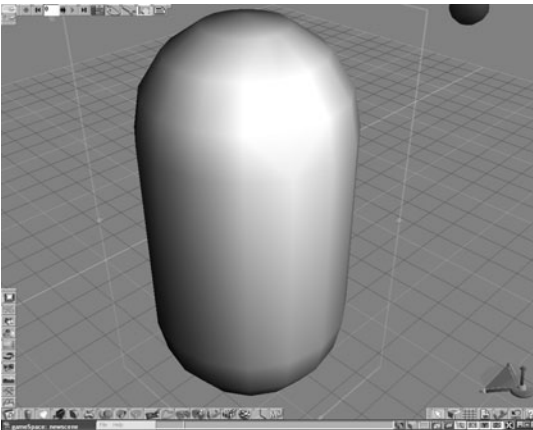


FIGURE 22.17 A Rounded Cylinder Primitive shape.

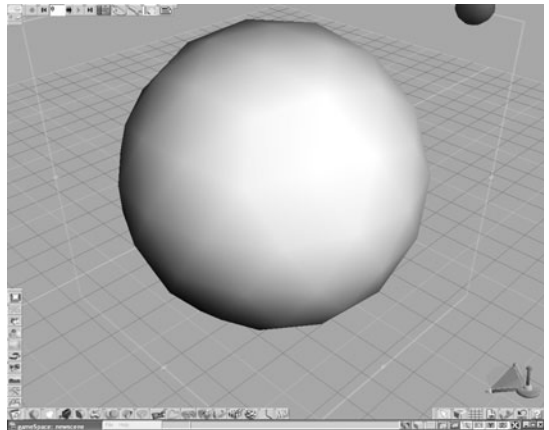


FIGURE 22.18 A Geosphere Primitive shape.

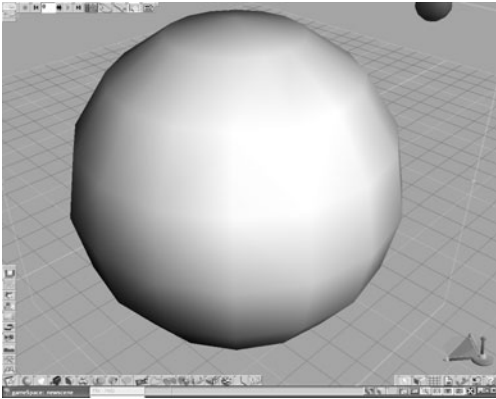


FIGURE 22.19 A Sphere Primitive shape.

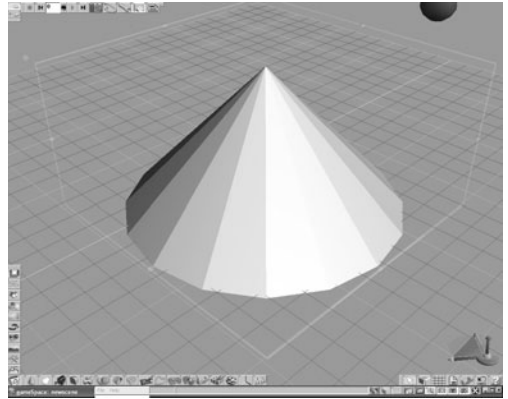


FIGURE 22.20 A Cone Primitive shape.

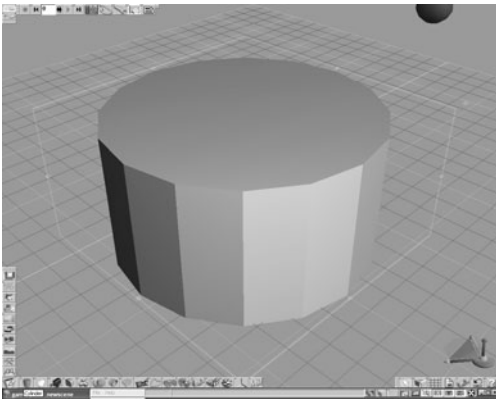


FIGURE 22.21 A Cylinder Primitive shape.

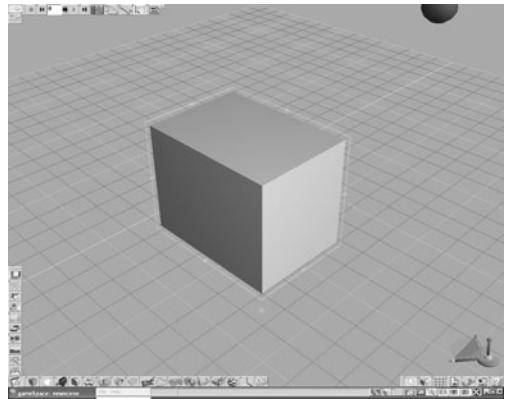


FIGURE 22.22 A Cube Primitive shape.

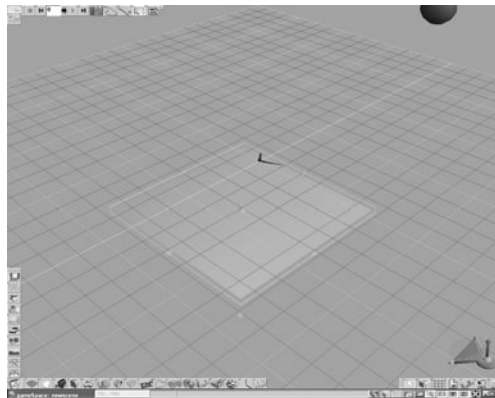


FIGURE 22.23 A Plane Primitive shape.

Once you have placed a primitive, you need to click the right mouse button, or you can continue to press the left mouse button to place more objects. Once you have pressed the right mouse button, if you move your mouse around the object, you will see some colored bars or circles. These allow you to amend the shape of the object or rotate it.

A SIMPLE 3D EXAMPLE

Now you will create a simple 3D model to get a quick idea of what you need to do to make your own models. When making a 3D model, it is good to use reference material or draw a pencil drawing of what you intend to make so you have a good idea of what you need to create and its dimensions.

In this example you will create a simple hut building.

1. Click on the primitives and select Cylinder. Draw a small cylinder shape on the screen. This will be the walls of the building.
2. Now click on the Cone primitive and draw it to make a roof for the building.
3. Your objects now look like Figure 22.24.

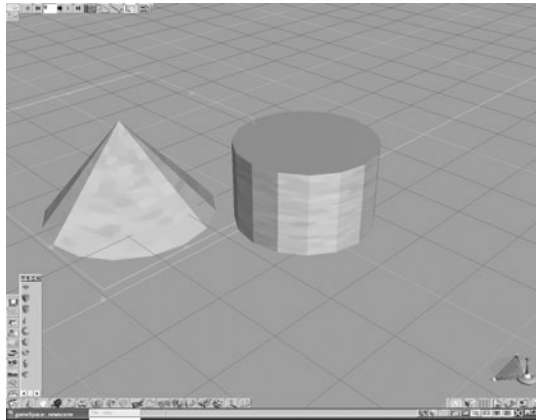


FIGURE 22.24 The two primitives next to each other.

4. Unfortunately, the roof is currently at the same height as the walls, so you need to increase the height of the object. First, click on the object so that it has a box around it. You can then use the right mouse button to adjust the height of the object.
5. By making slight adjustments of height and position, you should be able to get the object on top of the Cylinder, as shown in Figure 22.25.

The main problem with using this single view is that the precise positioning of the object takes a lot of moving around the screen. As mentioned earlier, there is a four-window option to display different views of the object, and this is much better for any precise positioning work that is required.



FIGURE 22.25 The hut taking shape.

6. Left-click and hold on the Standard views button and then from the pop-up, as shown in Figure 22.26, select the four-view button. You can now see four different angles and precisely position the objects.

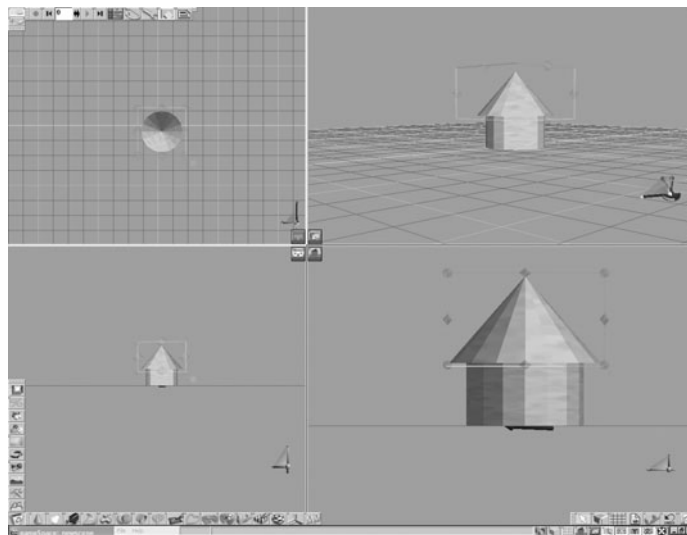


FIGURE 22.26 The two objects precisely positioned.

You can now apply a texture to the objects to make it look better, rather than this dull gray color. gameSpace allows you to place a texture onto a model or paint specific colors. For this, place a texture that is included in gameSpace.

7. Change back to Standard view.
8. Click on the Materials button on the left-hand toolbar, and a Materials toolbar will appear, as shown in Figure 22.27.

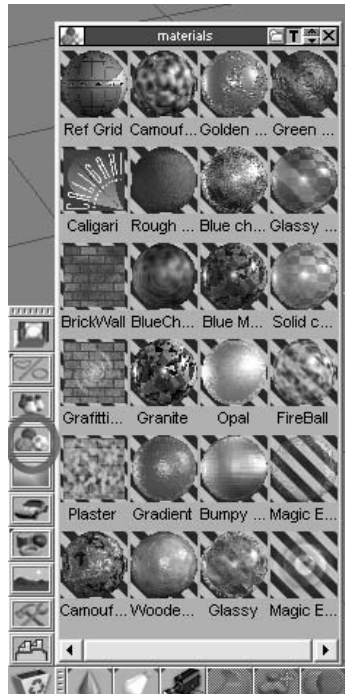


FIGURE 22.27 The Materials library button highlighted and the materials library displayed.

9. Select the Wooden Doll texture in the bottom row, second from the left. Drag and drop it onto the Cylinder object.
10. Select Camouflage from the bottom row on the left and drop it onto the Cone object.
11. You can see how it will look by clicking on the Render button, as shown in Figure 22.28. If you receive an error message about the object being too large, you need to resize the object so that it is a size that this version supports.

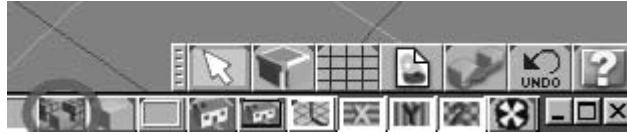


FIGURE 22.28 The Render button, which shows you how the object will look.

EXPORTING THE MODEL

The exporting of the model is very straightforward:

1. Select the File menu option at the bottom of the screen and then select Save As | Object.
2. A Save As dialog box will appear. By changing the Save as type you can convert to a number of different formats, as shown in Figure 22.29.
3. Type in the file name of the object.
4. Click Save when you are ready to save.

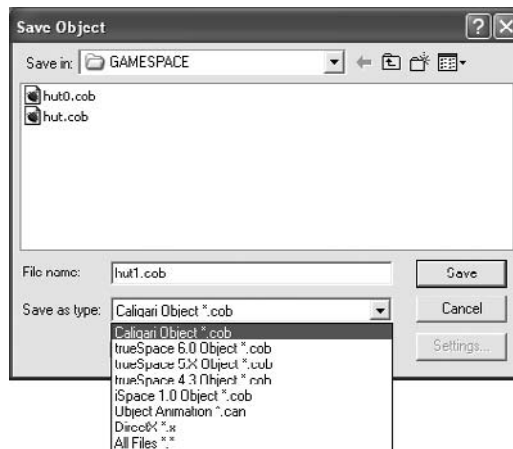


FIGURE 22.29 The Save As dialog box.

CHAPTER SUMMARY

In this chapter you took a quick tour of a free 3D program called gameSpace Lite, which can help you create 3D models for use in FPS Creator or in other game making programs.

3D modeling is a time-consuming process and isn't something that can be learned in one sitting. If you are interested in making your own 3D models for your games, we recommend that you read the accompanying documentation, visit the gameSpace Web site, and get a better feel for the program by spending more time looking over the features and building models. You may need to upgrade to the full version if you want to use it to export bitmap animations for use in TGF2 and Game Maker.

GAME-MAKING WEB SITES

In This Chapter

- Useful Web Sites



If you are looking for additional material to help you improve your game creating skills or are just interested in learning more, the Internet is the best place to start. The Internet has many game-based Web sites and resources that you can access for free and use to hone your skills. This chapter will look at some of the useful Web sites and resource sites you can visit to help you in your quest.

USEFUL WEB SITES

This is a quick round up of useful and interesting sites for you to visit.

Awesome Programming

To accompany this book a Web site has been created, as shown in Figure 23.1. This Web site offers all the latest information on the book as well as links to files and other sites of interest. You can also find contact details if you want to email the author of this book and send him any comments or your own games that you have created with the programs on the DVD.



FIGURE 23.1 Awesome Programming Web site, *www.awesomeprogramming.com*, which accompanies this book.

Make Amazing Games

If you enjoyed using TGF2 and are interested in learning more about it, visit www.makeamazing.com. This Web site accompanies the book *Make Amazing Games In Minutes*, which shows aspiring game creators how to create their own games using TGF2. The whole book is devoted to TGF2 and also has additional material and demos for distributing your own software. This book is also available from Charles River Media. You can see the Web site in Figure 23.2.

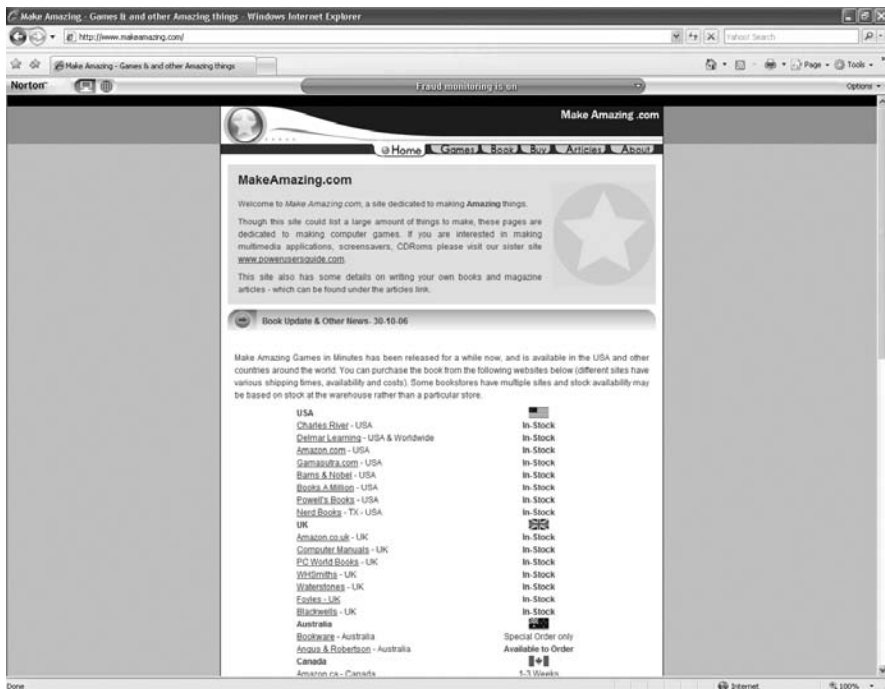


FIGURE 23.2 The Make Amazing Games Web site.

Power Users Guide

If you are thinking about making educational programs, screensavers, or Windows-based applications, you can visit the Web site www.powerusersguide.com. This Web site is based on another book by Charles River Media, called *The Power Users Guide to Windows Development*. The software used in this book is similar to TGF2 and uses the same interface but is called Multimedia Fusion Developer. You can see the Web site in Figure 23.3.

Clickteam

This is the home Web site of TGF2 and its more powerful brothers Multimedia Fusion 2 and Multimedia Fusion 2 Developer. At this site you can access online videos, tutorials,



FIGURE 23.3 *The Power Users Guide to Windows Web site.*

and additional material to load into TGF2. You can also register on the online forums, which provide up-to-date information about the latest versions, new features, and support questions and answers. You can access the Web site at www.clickteam.com; you can see a screenshot of the Web site in Figure 23.4.

EJAY

A product that we covered in this book, eJay is an easy to use music creation software program. You can get more information on the product from the Web site at www.ejay.co.uk. You can also register as an artist at eJay and upload your tunes and music for others to listen to. You can see the Web site in Figure 23.5.

Click Convention

If you want to meet up with like-minded game hobbyists and professionals, you should visit the Click Convention Web site at www.clickconvention.com. This is a meeting of users from the Clickteam forums who get together on a yearly basis to discuss game creation techniques and show off their own creations. You can see the Web site in Figure 23.6.



FIGURE 23.4 The Clickteam Web site at www.clickteam.com.



FIGURE 23.5 The eJay Web site at www.ejay.co.uk.

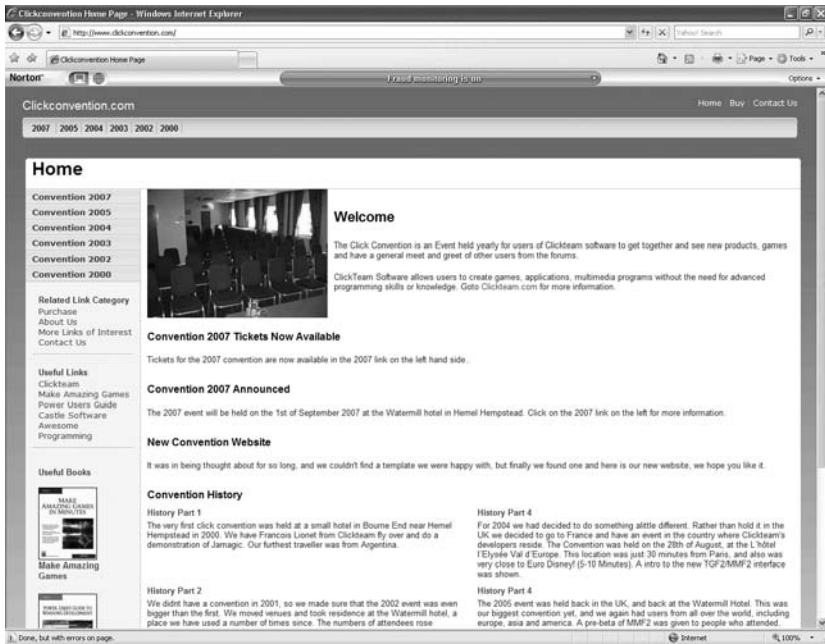


FIGURE 23.6 The Click Convention Web site at www.clickconvention.com.

Caligari

One of the products we included a demo of on the DVD is gameSpace from Caligari. Caligari produces a wide range of 3D-based modeling software to meet the requirements of any hobbyist or professional wanting to make their own models. They offer a wide range of products from a free version to the professional version used to make 3D effects in movies and computer games. You can visit the Web site at www.caligari.com and access gameSpace from the links. You can see the Web site in Figure 23.7.

The Game Creators

We have included products from The Game Creators on the DVD, namely FPS Creator and The 3D Gamemaker. They make a wide range of game creation programs as well as ones that require you to use traditional programming techniques. You can find out more about these products at their Web site at www.thegamecreators.com.

Game Maker

You used Game Maker 7 Lite earlier in this book, and you can find out more, download product examples and help files, as well as upgrade to the full version at www.yoyogames.com/make. You can also upload your own executable game creations to their download service.



FIGURE 23.7 Caligari's gameSpace Web site accessible from www.caligari.com.

ACIDplanet

We used ACID XPress earlier in the book, and this Web site compliments the product very well. It provides a place where artists can upload their own tracks, but also has access to articles and additional help files for the product. The great thing about ACIDplanet is that they release free music loops that you can download and use in your own creations. Visit the site at www.acidplanet.com.

Turbo Squid

Want to purchase 3D models and content for your own games so you don't have to spend time making your own? Turbo Squid is a content provider, so you can search its archives, see the different file formats, and download/purchase content. You should read any license terms carefully to ensure that they meet with your product requirements. You can visit the Web site at www.turbosquid.com.

Gamasutra

If you want to get articles and information about professional game creation and development, Gamasutra is the Web site to visit. There is tons of useful information that will help you understand the game market and game making techniques. You can visit the site and register at www.gamasutra.com.

Retro Remakes

If you are interested in making or playing classic games that appeared on the Spectrum 48/128, Commodore 64, Atari, and Amiga computers, this is a great place to visit. Not only does it have many remakes of classic games, but you can learn a lot about game development, and particularly level development, from older games. You can visit the Remakes site at www.retroremakes.com.

The Daily Click

If you are looking for a community site where you can upload your games and get feedback on them, then the Daily Click site is a good place to visit. You can find game articles, reviews, and games to download. This is a user-based site, so it doesn't have the professional edge some of the other sites have, but it can provide a goldmine of interesting and useful information. You can visit the site at www.create-games.com.

CHAPTER SUMMARY

In this chapter we have taken a quick look at some of the useful Web sites you can visit. Many Web sites are available on the Internet that provide useful and interesting details about game creation. Some are better than others, but using links off some of the ones mentioned here will allow you to access some of the better ones rather than trying to do a general search in your favorite search engine.

A

DESIGN DOCUMENT: FIRST-PERSON SHOOTER

Document written by John Doe.
Version # 1.20
April 23, 2007
Copyright 2007 ABC Gaming. All rights Reserved.

DESIGN HISTORY

This is version 1.20 of the document, which began on April 23, 2007.

Version 1.10

For example, you can use Version 1.10, 1.20, etc.

1. Changed platforms for game. Added Max OS X.
2. Graphics are now 32 bit.

Version 1.20

The story has now been rewritten.

1. Details of story now changed.
2. Enemy character is now a humanoid model.

GAME OVERVIEW

Type of Game

This game is a 3D shooter with. . . .

Game Ideas

This game has been in the creation process for a number of years. The original idea was to place a character in a 2D world, but with the power of today's graphics cards, we can now realize our creation and world within 3D.

Location

The game takes place on an island. You are surrounded by rocky ledges and water as far as the eye can see. You wake up with a cut on your head and you cannot remember how you got here.

Players

You will control the main character in the story from a first-person perspective. You will begin the game with a knife, and additional weapons are available throughout the level.

Main Objective

Your objective is to get off the island and remember what happened.

Game Overview

The game takes a slightly different approach from the development of first-person shooters in that instead of blasting your way through a level, destroying everything in sight, you must also figure out a way to get off the island. There are multiple ways of getting off the island including building a craft or stealing a boat plane. While trying to accomplish the mission of getting off the island, you will also try to piece together how you got on the island in the first place.

FEATURES

General Features

- Large terrains
- 3D graphics
- 32-bit color
- Several types of enemies, including animal and human

Multiplayer Features

- Up to four players using TCP/IP connection over the Internet
- Players can host their own server and allow people to connect

Editor

- No world editor at this time, but planned for Version 2.1
- Free levels available to download off the Internet to registered users

THE GAME WORLD

Overview

An island with no obvious means of escape.

Key Locations

- There is a cave on the south side of the island, which contains a repair kit and supplies.
- The creek that runs throughout the island is the quickest way to travel.
- There is a boat at the bottom of the lagoon. A compass is inside it.

Objects

- Various boat parts to fix the broken boat on the west end of the island
- Food items to restore health
- Additional weapons and ammunition
- Notes and letters from various characters telling the story of the island
- Parts to build a glider on the center mountain ridge

GRAPHICS

Rendering/3D Engine

The 3D engine will be using in-house-written routines that support Direct X9. It will be an FPS view with the ability to view from behind the character if needed. The island will be one continuous area with behind-the-scenes loading, so the player is not interrupted in any way. There will be loading pauses of major story arc chapters, which will describe what is happening once a player achieves a certain goal or objective.

GAME CHARACTERS

Main Character

There is only a single main character in the game. Other characters will appear, but they will only be extras to the story line.

Enemies

There will be several enemies that you'll encounter:

- Wolves
- Rabid dogs
- Sharks
- Guards
- Soldiers
- Elite soldiers

WEAPONS

Types

There will be 12 different weapon types on the island, starting with a knife and including:

- Pistol
- Rifle
- Sub machine gun
- Sniper gun

MUSIC AND SOUND EFFECTS

Music

The in-game music will be made by a third-party musician in a recording studio, which will then be imported into our game.

Sounds

The sound effects include basic weapon loadings, firing, footsteps when walking, a swimming sound, etc.

APPENDIX ABC

Any additional information . . . ideas include:

User Interface

The basic user interface will consist of three menus. . . .

B

THE KEY POSITIONS IN A DEVELOPMENT TEAM

A development project is made up of several key positions. Without any of these, it would not be successful. That being said, depending on the size of your team, a single individual may be forced to wear many hats, or in the case of the lone developer, all of the hats. That is, although all of the positions are required, a single individual may fill one or all of them.

Because the game industry is still in its infancy, it's sometimes difficult to discuss the positions that make up a team. The type of game being produced definitely also has a profound effect on the required personnel. Every development project is arranged differently. As the industry matures, there will certainly be more standard types of arrangements.

DESIGNER

Many development projects have a lead game designer who is responsible for the creation of the game script. This position is often one of the most misunderstood of any of the key positions and is often left completely off the team. This leaves everyone, from the producer to programmers, clamoring for the title.

It is the designer who makes many of the decisions related to the creation of important aspects such as puzzles or the levels in an FPS. Like a screenwriter for a movie, the designer is responsible for the overall feel of the game. Communication is a very important aspect of this job, as designers work with the other team members throughout the duration of a project.

In the beginning stages of a game, designers spend most of their time focusing on writing short scripts and working on the beginning storyboard sketches. A typical storyboard displays the action of a game in a very simple manner. Depending on the basic talents of the designer, the

storyboard may even include stick figures and basic shapes to convey the action. Storyboards are a sort of rough draft that will later be transformed into the game.

After the decisions have been made on the game concepts, the designer begins working on a blueprint for the game, called a *design document*. Simply put, the document details every aspect of a game and will evolve as the game is being developed.

PROGRAMMER

Game programmers are software developers who take the ideas, art, and music and combine them into a software project. Programmers obviously write the code for the game, but they may also have several additional responsibilities. For instance, if an artist is designing graphics for the game, the lead programmer could be responsible for the development of a custom set of tools for creating the graphics. It is also the lead programmer's job to keep everything running smoothly and to somehow figure out a way to satisfy everyone, from the producer to the artists. Unlike the stereotype portrayed on many Web sites, books, and even movies, programmers usually don't stroll into work at noon, work for a few hours, and then leave. The truth is, they often arrive earlier and leave later than anyone else on the development team.

Programmers are responsible for taking the vast number of elements and combining them to form the executable program. They decide how fast characters can run and how high they can jump. They are responsible for accounting for everything inside the virtual world. While doing all of this, they often attempt to create software that can be reusable for other projects and spend a great deal of time optimizing the code to make it as fast as possible.

Sometimes, a project may have several programmers who each specialize in one key area, such as graphics, sound, or artificial intelligence (AI). The following list details the various types of programmers and what they are primarily responsible for:

Engine or graphics programmers. They create the software that controls how graphics and animations are stored and ultimately displayed on the screen.

AI programmers. They create a series of rules that determine how enemies or characters will react to game situations and attempt to make them act as realistically as possible.

Sound programmers. They work with the audio personnel to create a realistic-sounding environment.

Tool programmers. As previously mentioned, programmers often write software for artists, designers, and sound designers to use in the development studio.

AUDIO-RELATED POSITIONS

High-quality music and sound effects are an integral part of any gaming project. This is also an area that many teams simply cannot afford to throw a great deal of money at. Having superb audio components such as music, sound, and voice can greatly enhance the total experience for the consumer. The opposite is also true, however. Music that is done poorly can keep people away from your product, regardless of its other qualities. The positions listed below are usually filled by key audio personnel, although sometimes a programmer or other team member will fill in, as needed.

Musician

When compared with the stress and long hours of the programmers, musicians are often at the other end of the workload. They often have the least amount of work of any of the positions on the team. That's not to imply that they don't work hard; it's just that there isn't as much for them to do. They usually are responsible only for the music for a game. While this is an important job, it doesn't typically take a great deal of time compared with the other team members' jobs. Because of the relatively short production times, musicians often have secondary work outside of the gaming industry.

Sound Effects

Depending on the makeup of a team, a musician could be involved with the creation of the sound effects in a game. This can often make up for the lack of work and help keep the budgets down. Another route many teams choose to follow is the purchase of pre-existing sound effects. Many sound effects companies distribute their work on CD-ROMs or the Internet. Many teams choose to purchase the sounds produced by these companies and alter them to their liking.

ART-RELATED POSITIONS

Artist

The artists are responsible for creating the graphics elements that make up a project. They often specialize in one area within a project, such as 3D graphics or 2D artwork such as textures. The artists usually work from a set of specifications given to them by the programmer. Unfortunately, artists and programmers often have many disagreements on these specifications. For instance, artists might want to increase the polygon counts on a 3D model so that their work will look better, while programmers may want to decrease the counts to make the program run more smoothly.

Game artists have a variety of technical constraints imposed by the limitations of the hardware they are creating for. Although hardware continues to increase in speed and go down in cost, there is never enough power to satisfy a development project. Therefore, it is often the artists who are given the responsibility to create objects that work within the constraints.

Depending on the development team, there are three basic types of artists: character artists (or animators, as some prefer to be called), 3D modelers, and texture artists.

Character Artist

Character artists have one of the most demanding jobs on the team. They create all of the moveable objects in a game, such as the main character and vehicles. It is their job to turn the preliminary sketches that are often discussed by the entire team into a believable object on a computer screen.

Using 3D modeling tools such as 3D Studio Max™, TrueSpace®, Maya®, or Lightwave™, character artists use basic shapes and combine them to form characters. If you have never used a 3D-modeling program, you can think of it as a type of digital clay. Once created, characters are fleshed out with a 2D graphic image that is made in another program.

The character artists are also responsible for the animation of the objects. They may be required to animate a horse, a human being, or a creature that previously existed only in someone's mind. Character artists often look at real-world examples to get their ideas on how a character should move. Depending on the type of game, they may have to create facial expressions or emotions as well.

It's often the responsibility of a character artist to implement cut scenes in a game, as well. Many artists enjoy creating cut screens even more than creating the characters in the game. They have much greater freedom and are not restricted as to the number of polygons a certain object can have or the size of the object.

3D Modeler

The 3D modeler usually works on the settings in which a game takes place, such as a basketball arena or a Wild West wasteland. Background artists work hand in hand with the designer to create believable environments that work within the constraints of a game. Like character artists, they use a wide range of tools for their jobs, including both 2D and 3D graphics tools, although they usually only model static objects.

Texture Artist

Texture artists might be the best friend of the other artists. It is their job to take the work created by the modeler and character artist and add detail to it. For example, they could create a brick texture that when added to a 3D box created by the modeler, creates the illusion of a pile of bricks. On the other hand, they could create a texture that looks like cheese, turning this same box into a block of cheese.

PRODUCER

A producer oversees the entire project and attempts to keep everything moving along as smoothly as possible. A producer often acts as an arbitrator to help patch up any problems between team members. For instance, if an artist wants to increase the color palette and a programmer wants to decrease it, the producer may make the final decision.

SECONDARY POSITIONS

Several secondary positions can be important to the development cycle, as well. Depending on the budget, these positions may or may not exist at all or could be filled by other members of the team.

Beta Tester

Beta testers test the playability of a game and look for bugs that may occur when the game is executed. This is one of the most undervalued of the positions and should never be done by the person responsible for programming the game. In reality, because of tight budgets and deadlines, beta testing is one of the steps that is often cut before it is completed, as due dates unfortunately take precedence over most decisions. If adequate beta testing is performed, a development team can save a tremendous amount of time and resources without having to produce unnecessary patches at a later date.

Play Testers

The play testers are often confused with beta testers. The difference is that play testers only test the *playability* of a game. They often critique areas such as movement or graphic elements. Again, these positions are often filled by people who perform other tasks on the team. Unlike beta testers, play testers do not attempt to find or report bugs. Their purpose is to judge if a game is fun to play.

This page intentionally left blank

C

ABOUT THE DVD

The companion DVD contains everything you need to make all of the programs included in this book.

GENERAL MINIMUM SYSTEM REQUIREMENTS

You need a computer that can run Windows 95 or better with a CD-ROM/DVD drive, a sound card, and a mouse.

ACID XPRESS (WWW.ACIDPLANET.COM) TRIAL

The file name for the program is acidxpress50a.exe, and it is located in the Demos folder.

Requirements:

- Microsoft® Windows® 2000 or XP 800 MHz processor (1 GHz if using video)
- 200 MB hard disk space for program installation
- 600 MB hard disk space for optional Sony Sound Series Loops & Samples reference library installation
- 256 MB RAM
- Windows-compatible sound card CD-ROM drive (for installation from a CD only)
- Supported CD-recordable drive (for CD burning only)
- Microsoft DirectX® 8.1 or later
- Internet Explorer 5.1 or later

DANCE EJAY 7 (WWW.EJAY.CO.UK) TRIAL

The file name for the program is danceejay7Demo.exe, and it is located in the Demos folder.

Requirements:

- Pentium 3, 800 MHz or higher
- 256 MB RAM
- Windows 98, ME, 2000, XP
- 1.4 GB free hard disk space
- 4x CD-ROM
- CD-Writer (for Audio CD Burning feature)
- DirectX 9.0 compatible graphics card with 32 MB of video memory (16-bit color, 1024 × 768, 32 MB)
- DirectX 9.0 compatible sound card (16-bit)
- DirectX 9.0c
- Web browser

Recommended Specs:

- Pentium 4, 1.8 GHz
- 512 MB RAM
- Windows 98, ME, 2000, XP
- 1.4 GB free hard disk space (for Install)
- 2.0 GB free hard disk space (for OS Cache)
- 4x CD-ROM
- DirectX 9.0 compatible graphics card with 64 MB of video memory (16-bit color, 1024 × 768, 64 MB)
- DirectX 9.0 compatible sound card (16-bit)
- DirectX 9.0c
- Web browser

The graphics card must be compatible with Direct X 9.0c.

GAME MAKER 7.0 LITE (WWW.YOYOGAMES.COM) TRIAL

The file name for the program is gmaker.exe, and it is located in the Demos folder.

Requirements:

- Pentium PC or higher
- Windows ME, 2000, XP, or Vista (or higher)
- 10 MB of hard disk space
- 65000 colors (16-bit)
- 800 × 600 screen resolution
- 32 MB 3D graphics card (DX compatible 8.0)
- Sound card

GAMES FACTORY 2.0 (WWW.CLICKTEAM.COM) TRIAL

The file name for the program is TGF2Demo.exe, and it is located in the Demos folder.

Minimum Requirements:

- Operating system: Windows 95 with IE 4.0, 98, NT 4.0 with Service pack 3 or above, 2000, XP, or Vista
- Pentium Processor
- 32 MB RAM with Windows 9x, 64 MB with Windows NT, 128 MB with 2000 and Windows XP, 512 MB with Vista
- CD-ROM drive
- Graphics card with 8 MB or more (or minimum OS requirements)
- Sound card (optional but recommended)
- 50–100 MB free hard disk space

Recommended Requirements:

- Operating system: Windows 98, 2000, XP, or Vista
- Pentium 4 Processor
- 64 MB RAM with Windows 98, 256 MB RAM with Windows 2000 or XP, and 1 GB RAM with Windows Vista
- CD-ROM drive
- Graphics card with 32 MB RAM
- Sound card
- 200–500 free hard disk space

FPS CREATOR (WWW.FPSCREATOR.COM) TRIAL

The file name for the program is FPSCreatorDemo.exe, and it is located in the Demos folder.

Minimum Requirements:

- Operating system: Windows 2000 or XP
- Processor: Pentium 3 – 1 GHz
- Memory: 256 MB RAM
- Graphics card: Direct X9.0c-compatible with 64 MB RAM
- Hard disk: 1.4 GB
- Other: Printer if you want to print any screens or documentation

Recommended Requirements:

- OS: Windows XP Home or Pro
- Processor: Pentium 4 – 2.66 GHz
- Memory: 1 GB RAM
- Graphics card: Direct X9.0c-compatible with 128 MB RAM
- Hard disk: 1.4 GB
- Other: Printer if you want to print any screens or documentation

3D GAME MAKER (WWW.THEGAMECREATORS.COM) TRIAL

The file name for the program is setup.exe, and it is located in the Demos\3DGM folder.

Minimum Requirements:

- 400 Mhz Pentium II Processor
- Windows 95, 98, 2000, ME, or XP
- 600 MB of hard disk space
- 64 MB of RAM
- DirectX Version 7.0a
- Fully DirectX-compatible graphics card with 3D acceleration and 8 MB of memory
- DirectX-compatible sound card
- 4x CD-ROM drive

Recommended Requirements:

- 600 Mhz Pentium III Processor
- Windows 95, 98, 2000, ME, or XP
- 600 MB of hard disk space
- 128 MB of RAM
- DirectX Version 8.0a
- Fully DirectX-compatible graphics card with 3D acceleration and 16 MB memory
- DirectX compatible sound card
- 16x CD-ROM drive

GAMESPACE LITE (WWW.CALIGARI.COM) TRIAL

The file name for the program is gSLight_PP.exe, and it is located in the Demos folder.

Requirements:

- Windows 98, ME, NT4, 2000, or XP
- AMD Athlon or Pentium 120 (P4 or AMD K7 recommended)
- 64 MB RAM (128 or more recommended)
- 50 MB free hard disk space
- 3D video card with at least 16 MB of video memory and D3D or OGL drivers

FOLDERS

A number of folders on the DVD contain important files for use with this book as well as useful information.

3dTO2D. An example of 3D images being used on a 2D program

Animations. A set of images that create an animation effect

Demos. Location of the demo files you can install

EJAYFILES. Files needed for the eJay program

Figures. Color versions of every figure seen in the book

FPSFILES. Example files required for the FPS Creator

GAMESPACE. Example files for the gameSpace chapter

Samples. Sound files for use in some of the example files

TGFFILES. Example files required for the chapters using The Games Factory 2 software

INDEX

NUMBERS

- 1.33:1 aspect ratio, 40
- 1610 microprocessors, 16
- 16:9 aspect ratio, 40–42
- 16-bit color (High Color), 44–45
- 24-bit color (True Color), 44–45
- 256 Color setting, 44–45
- 2D art, 37–38
- 2D games. *see* End of Earth; Litter Bug
- 32-bit color, 44
- 3D cards, 4–5
- 3D Gamemaker, 437–447
 - creating game with, 441–445
 - DVD-ROM for this book, 484
 - installing, 438–441
 - playing game, 445–447
 - saving game, 445
 - system requirements, 438
- 3D games
 - board games, 22
 - fighting games, 26
 - first person shooter games, 30
 - first-person 3D vehicle games, 30–31
 - maze games, 21
 - polygons and, 17
 - rendering real-time, 45
 - third-person games, 31
- 3D graphics, 37–38. *see also* 3D Gamemaker; gameSpace Lite
- 3D world editor, 362, 372–374
- 3dTO2D folder, DVD-ROM for this book, 484
- 4004 microprocessors, 16
- 4:3 aspect ratio, 40, 42
- 6502 microprocessors, 16
- 8-bit color, 44

A

- A key, FPS Creator test game, 381
- ACID Music Studio, Sony, 68
- ACID Planet website, 469
- ACID XPress, 69–78
 - installing, 69–71
 - requirements, 69
 - running for first time, 71–73
 - simple creation in, 75–78
 - sound loops, 68
 - touring, 73–75
 - trial, DVD-ROM for this book, 481
- action games, 91
- Action Point, viewing, 353
- actions
 - Event Editor, adding, 199–201
 - Event Editor icons for, 193
 - Event Editor, in events, 195
 - Game Maker, setting variables, 148
 - Game Maker, spaceship event, 150–152
 - recording, 62–63
 - sound effect created from, 62
 - sound effects conveying, 61
 - TGF2, creating Litter Bug, 278–286
- active objects, 319–321
- Add New Object dialog, 328
- Add New Prefab icon, 376
- Add Object dialog box, 326
- Ad-Lib, history of, 20
- advanced control of objects, 317–339
 - active objects, 319–321
 - backdrop and quick backdrop objects, 321–323
 - hi-score objects, 323–325
 - lives objects, 326–327
 - score objects, 328
 - text objects, 325–326
 - using in your games, 318–319
- advanced control of objects, movement
 - Bouncing Ball, 329–331
 - Eight Directions movement, 335
 - Mouse Controlled, 334
 - multiple, 338–339
 - overview of, 328–329
 - Path Movement, 331–333
 - Pinball Movement, 333–334
 - Platform movement, 337–338
 - Race Car movement, 336–337
- advanced games
 - Black Jack. *see* Black Jack
 - Dragons. *see* Dragons
- adventure games
 - choosing game type, 91
 - overview of, 32
- After End group, Dragons game, 311
- alarm (timer), 161–162, 248
- Alien Wars, 186–254
 - with Event Editor. *see* Event Editor, TGF2
 - with Frame Editor, 191–193
- highscores frame, programming, 252–253
- highscores frame, setup, 214–216
- Library, 204–205
- loading, 187
- overview of, 186
- setup, creating and renaming frames, 207–208
- setup, game frame, 212–214
- setup, main frame, 208–212
- setup, overview, 205–207
- with Storyboard Editor, 188–190
- storyline, 186–187

- Alien Wars, game frame programming, 231–251
 - collision between enemy and bullet, 245–247
 - collision between enemy and spaceship, 251
 - components, 232
 - creating enemy ships, 237–242
 - determining if music has stopped playing, 234
 - display always on top, 236–237
 - player shooting, 242–244
 - position of bullets, 244–245
 - recreate enemy robots, 247
 - robot animations, 250
 - robot firing, 247–250
 - Start of Frame, 232–234
 - stopping ship from leaving scene, 234–236
- Alien Wars, main frame programming, 217–231
 - button effects, 227–229
 - components, 217
 - creating note event, 217–219
 - determining if music has stopped playing, 223–224
 - overview of, 217
 - setting transparency to 0, 224–226
 - Start of Frame event, 219–223
 - user clicks, 229–231
- alterable variables
 - Black Jack, 305
 - Dragons, 312
- AMD (Athlon) processor, 4
- Animation folder, DVD-ROM for this book, 484
- animation tool, 354–359
- animation direction, 358

- default animation list, 359
- Directions tab options, 356–357
- overview of, 354–356
- animations
 - sprite, 50–52
 - TGF2, creating tracks for each direction, 337
 - TGF2, programming buttons in main frame, 227–229
 - TGF2, putting robots back into normal, 250
 - TGF2, setting up in game frame, 233–234
 - TGF2, using Shooting Animation, 248–250
- anti-aliasing technique, 55
- Arena mode, FPS Creator, 432–436
- Arena tab, FPS Creator, 430–433
- arrays, 301–302
- artificial light, 400
- aspect ratio, 40–42
- assembly language, 14–15
- Asteroids game, 13, 24
- Athlon (AMD) processor, 4
- ATI graphic cards, 5
- audience, designing for, 93
- audio files, story zone, 423–424
- auto items, 391
- AVI files, story zone, 423–424
- Awesome Programming website, 464
- B**
- Back Color button, 218–219
- backdrop objects, 321–323
- background
 - Game Maker, creating End of Earth, 126, 139–140, 144
 - TGF2, creating Litter Bug, 259–260, 262, 264, 267–268
 - TGF2, for comments in Alien Wars, 218–219
- backup devices, 6–7
- Badguy, Litter Bug game
 - computer players group, 288–291
 - defined, 262
 - end code, 293
 - ordering, 276
 - positioning, 265
- battle card games, 24
- BD (Blu Ray Disc) DVD format, 6
- behavior code, 312–314
- beta tester, 479
- Black Jack, 298–306
 - alterable variables and flags, 305
 - application properties, 298–300
 - files, 300
 - frames, 298–299
 - game Event Editor, 304–305
 - game frame, 302–304
 - loops, 306
 - menu Event Editor, 302
 - menu frame, 300–302
- Blu Ray Disc (BD) DVD format, 6
- board games, 22
- body collision, recording sound, 62
- Bouncing Ball movement, 329–331
- Build Game menu, FPS Creator, 428–436
- Build Settings tab, FPS Creator, 430, 432
- bullets. *see also* gunshots
 - 3D Gamemaker, enemy screen, 443
 - 3D Gamemaker, player screen, 442
 - Game Maker, firing frequency, 151
 - Game Maker, importing sprites, 135–137
 - Game Maker, programming, 163
 - TGF2, collision between enemy and, 245
 - TGF2, handling firing from enemy robots, 247–250
 - TGF2, player shooting, 242–244
- Bunker Fighting Area Ceiling, 393
- buttons
 - FPS Creator menu, 368–371
 - TGF2, adding comment to, 226
 - TGF2, programming in main frame, 227–229
- buying equipment, 8–9
- C**
- C key, FPS Creator, 381
- C programming language, 18
- C++ programming language, 18–19
- Caligari website, 468–469
- capacitors, 12
- car crash sound, recording, 62
- card games
 - battle card games vs., 24
 - creating Black Jack. *see* Black Jack
 - overview of, 22–23
- Cartesian coordinate system, 37–38
- CD Burning, eJay 7, 83
- CD-R (CD-Recordable) drives, 6
- CD-ROM, accompanying this book
 - ACID XPress, exporting songs with, 78
 - ACID XPress installation, 69–71
 - ACID XPress, simple creation using, 76
 - Alien Wars, 187
 - Black Jack game, 300
 - design document sample, 94
 - Dragons game, 306
 - eJay 7, exporting songs with, 87
 - eJay 7 installation, 79–80
 - End of Earth, adding background to, 139–140
 - End of Earth, creating sprites, 132
 - FPS Creator trial demo, 363
 - Game Maker, creating game, 119
 - Litter Bug library, 257
 - music, programming in main frame, 221
 - number of colors, 44–45
 - sound looping programs, 68
 - TGF2 installation, 176
 - TGF2 trial version, 174
 - user clicks, programming, 230
 - working with animation tool, 354
 - working with Game Maker, 106
- CD-ROM drives
 - backups to, 6, 8
 - game studio setup, 5
- CD-ROMs, creating music from, 6
- ceilings, creating, 393–394
- CGA (PC graphics standard), 20
- chair, for game studio, 8
- Change Sprite dialog, 151
- characters
 - 3D Gamemaker, adding, 443
 - in design document section, 97, 473–474
 - FPS Creator, adding enemy, 385–386
 - Game Maker, stopping from leaving screen, 157, 165–166
 - TGF2, adding to game frame, 265
- ChocoBreak tutorial, 233
- cleaning fluid, 275, 285–287
- Clear Image option, Picture Editor, 343
- Click Convention website, 466, 468
- Clickteam website, 465–467
- clients, in multiplayer online games, 433–435
- CMYK (Cyan, Magenta, Yellow and Black) color value, 43
- code groups. *see* groups, code
- collisions
 - Game Maker, ice events, 163
 - Game Maker, reducing health score for, 158
 - testing with backdrop platform object, 338
- TGF2, dirt and cleaning fluid, 286–287
- TGF2, enemy and bullet, 245–247
- color
 - changing background, 218–219
 - depth, 44–45
 - graphics and, 42–44
 - masking, 52–53
- Color Palette, 353–354
- Color Picker, 347–348
- comments, TGF2
 - in Event Editor, 195–196
 - game name and version, 269
 - preventing object from leaving screen, 235
 - programming in main frame, 217–219
 - programming music, 269–270
 - recreating enemy robots, 245
- Storyboard Editor, 188–189
 - when mouse is over object, 270–271
- Commodore 64, 19
- computer on a chip, 16
- computer players group, TGF2, 287–291
- computers
 - game studio setup, 3–4
 - history of silicon circuits, 12
 - tips for buying, 8–9
- conditions
 - adding to Event Editor, 197–199
 - defined, 196
 - for menu frame, 231
- Cone primitives, 457–458
- Config & Options, eJay 7, 83
- Confirmation dialog, TGF2, 178
- connectors, recording device, 64
- consoles, game, 19–20
- Control Panel
 - creating Litter Bug game, 261, 266
 - using sound card's mixer panel, 65
- Copy, in Picture Editor, 345
- Copy command, 46
- corridors, FPS Creator
 - creating with paint tools, 391–394
 - creating with single segments, 387–391
 - lighting, 400–401
- Counter, 261, 266, 274, 293
- Counter_NLvl, Litter Bug, 260, 266, 293–294
- Create Object action, 238–243
- credit cards, purchasing equipment with, 8
- credit screens, 36
- Crop command, 48–49
- Crop tool, Picture Editor, 346
- Cube primitives, 457–458

- cursor, FPS Creator, 373, 378–379
- Cut, Picture Editor, 345
- Cut command, 46
- Cylinder primitives, 457–458
- D**
- D key, FPS Creator, 381
- The Daily Click website, 470
- debugging, in FPS Creator, 427
- DEC (Digital Equipment Corporation), 12
- delete
 - changing cursor back to basic type, 382, 384
 - Picture Editor tool, 345
 - sprites, 121
- Demos folder, DVD-ROM for this book, 484
- design document
 - components of, 94–95
 - defining, 94
 - first person shooter, 471–480
 - for game idea, 90
 - game proposal vs., 101
 - importance to team, 95
 - miscellaneous and appendix area of, 101
 - overview of, 93–94
 - printing copy for everyone, 101
 - sample outline, 94
 - storyline. *see* storyline
- design history, 471
- designer, 475–476
- desk, game studio setup, 8
- desktop toys, 33
- digital cameras, game studio setup, 5
- Digital Equipment Corporation (DEC), 12
- digital subscriber line (DSL) modems, game studio setup, 5–6
- Direct X9.0, FPS Creator support for, 362
- directions, for animations, 357
- Directions and Initial Directions dialog, 335
- dirt objects, Litter Bug game
 - collisions between cleaning fluid and, 286–287
 - computer players group, 290–291
 - defined, 261
 - end code, 293
 - placing off-screen objects, 266
- Display group, Dragons game, 311
- Doom 2*, 3, 398–399
- door slamming, recording sound, 62
- door switches, 398–399
- dpi (dots per inch), 40
- Dr. Mario, 24
- drag-and-drop games, 15
- Dragons, 306–315
 - alterable variables and flags, 312
 - application properties, 307–308
 - behaviors, 312–314
 - files, 306–307
 - frames, 307
 - game Event Editor, 311
 - game frame, 310
 - highscore Event Editor, 314–315
 - highscore frame, 312–314
 - menu frame, 308–309
- Draw Health settings, 167
- Draw toolbar, FPS Creator, 370
- drawing area, Picture Editor, 353
- drop-down text menus, FPS Creator, 368
- DSL (digital subscriber line) modems, game studio setup, 5–6
- DVD-ROM, accompanying this book, 481–485
 - 3D Gamemaker, 484
 - 3D Gamemaker installation, 438
 - ACID XPress trial, 481
 - eJay Dance 7 trial, 482
 - folders, 484–485
 - FPS Creator, adding door switches, 398
 - FPS Creator, adding enemy player, 385
 - FPS Creator, adding fire effect, 402
 - FPS Creator, adding furnishings, 405
 - FPS Creator, adding smoke effect, 402
 - FPS Creator, adding sound zone, 420–421
 - FPS Creator, adding story zone, 422
 - FPS Creator, adding windows, 396
 - FPS Creator, creating corridor with paint tools, 392
 - FPS Creator, creating corridor with single segments, 387, 391
 - FPS Creator, overview of, 483
 - FPS Creator, player start position, 383
 - Game Maker 7.0 Lite, 482
 - Games Factory 2.0, 483
 - gameSpace Lite, 450, 484
 - gameSpace Lite installation, 451
 - system requirements for, 481
- DVD-ROM drives
 - backup device, 6
 - game studio setup, 5
- E**
- E key, FPS Creator test game, 381
- Edit menu, Game Maker, 113
- editing
 - TGF2, hi-score object, 324–325
 - TGF2 screens for, 181–183
- educational games, 32, 91
- edutainment games, 32
- EGA graphics standard, 20
- Eight Directions movement, 335
- eJay Dance 7, 79–87
 - DVD-ROM for this book, 482
 - exporting songs, 86–87
 - installing, 79–81
 - requirements for, 79
 - running for first time, 81–83
 - simple creation in, 84–86
 - sound loops, 68
 - touring, 83–84
- eJay website, 466–467
- EJAYFILES folder, DVD-ROM for this book, 484
- elements. *see* game design elements
- elevators, adding, 409–412
- Ellipse tool, 349
- End group, Dragons game, 311
- End of Earth, 129–171
 - background, 139–140
 - executable file, 170–171
 - help file, 169–170
 - music, 137–139
 - objects, 140–142
 - overview of, 130–131
 - rooms, 142–147
 - sound using script, 168–169
 - sprites, 131–137
- End of Earth, programming objects, 147–168
 - bullet events, 163
 - health events, 167
 - ice events, 163–166
 - life events, 167–168
 - navigation buttons, 147–150
 - spaceship events, 150–163
- end of level boss, 3D Gamemaker, 444, 446–447
- enemy players
 - 3D Gamemaker, adding, 443
 - design document, 97, 474
 - FPS Creator, adding, 385–386
 - in single or multiplayer games, 100
- enemy players, TGF2
 - bullets from, 247–250
 - collisions between bullet and, 245–247
 - collisions between player's ship and bullet of, 251
 - creating robots, 238–243
 - Dragons game, 311
 - recreating robots, 245
 - robot animations, 250
- Enter key, FPS Creator test game, 381
- entities, FPS Creator
 - defined, 365
 - door switches, 398–399
 - enemy player, 385–386
 - fire effect, 402
 - furnishings, 405–406
 - in Library toolbar, 371–372
 - performance checking, 427
 - smoke effect, 402–405
 - weapon, 383–385
- equipment. *see* game studio setup
- Eraser tool, 351–352
- Ethernet network, game studio setup, 7
- Event Editor
 - Black Jack game frame, 304–305
 - Black Jack menu frame, 302
 - Dragons game frame, 311
 - Dragons highscore frame, 314–315
 - Dragons menu frame, 309
 - lives object requiring, 327
- Event Editor, TGF2, 193–201
 - adding action, 200–201
 - adding to, 196–200
 - defined, 180
 - events, 195–196
 - object icons, 194–195
 - overview of, 182–183
 - programming main frame. *see* Alien Wars, main frame programming
- events, Game Maker
 - overview of, 120
 - programming objects. *see* End of Earth, programming objects
 - selecting, 123–124
- executable files
 - FPS Creator, creating, 428–432
 - FPS Creator, creating multiplayer online game, 432–436
 - Game Maker, creating, 170–171
- Exit to Desktop, eJay 7, 83
- Explorer tab, ACID XPress, 74–75
- Export tool, Picture Editor, 344
- exporting
 - ACID XPress songs, 78
 - eJay Dance 7 songs, 86–87
 - models, in gameSpace Lite, 462
 - one-click build, in FPS Creator, 362
- Expression Evaluator, 222, 224–229
- F**
- F1 key, 169
- fake items, 391
- Family Computer (Famicom) console, Nintendo, 19
- fighting games, 25–26
- Figures folder, DVD-ROM for this book, 484

- File Manager editor, eJay 7, 86–87
- File Manager, eJay 7, 83
- File menu, Build option of FPS Creator, 428–432
- File menu, Game Maker, 112
- files
 - Black Jack, 300
 - Dragons, 306–307
 - executable. *see* executable files
 - help, 169–170
 - new, 258
- Fill tool, 350
- Filled Ellipse tool, 349
- Filled Rectangle tool, 348
- fire, recording sound of, 62
- fire effect, 401–402
- firing gun
 - programming, 159–161
 - programming bullet events, 163
 - recording sound for, 63
- First Person Shooter Creator. *see* FPS (First Person Shooter) Creator
- first person shooter, design document, 471–480
- first-person 3D vehicle games, 30–31
- flags, in Black Jack, 305
- flags, in Dragons, 312
- flight sims, 28
- Flip command, 49–50
- Flip Horizontally tool, 346
- Flip Vertically tool, 346
- FMV (full-motion video) games, 32
- folders, DVD-ROM for this book, 484–485
- fonts, 218, 325–326
- footsteps, recording sound of, 63
- formats
 - backup, 6
 - graphic, 56–57
 - MP3, 78
 - music and sound effects, 99
- formatted text object, 325–326
- FPS (First Person Shooter) Creator, 395–436
 - door switches, 398–399
 - DVD-ROM for this book, 483, 484
 - elevators, 409–412
 - enemy patrols using waypoints, 415–417
 - executable file, 428–432
 - fire effect, 401–403
 - furnishings, 405–406
 - games, 30
 - lighting rooms and corridors, 400–401
 - multiplayer online game, 432–436
 - performance checking, 426–428
 - placing zones, 417–424
 - smoke effect, 402–405
 - stairs, 408–409
 - teleporters, 412–415
 - using models created in gameSpace, 450
 - windows, 396–398
- FPS (First Person Shooter) Creator, creating basic game, 375–394
 - corridor with paint tools, 391–394
 - corridor with single segments, 387–391
 - creating first room, 376–379
 - enemy player, 385–386
 - player starting position, 382–383
 - testing, 380–382
 - weapon, 383–385
- FPS (First Person Shooter) Creator, introduction, 361–374
 - 3D world editor, 372–373
 - cursor, 373
 - defined, 361
 - drop-down text menus, 368
 - game creation process, 366
 - installing, 363–365
 - Library toolbar, 371–372
 - menu buttons, 368–371
 - options in program, 362–363
 - program window, 367
 - status bar, 374
 - system requirements, 363
 - terminology, 365–366
- Frame Editor, TGF2
 - Alien Wars project, 191–193
 - defined, 180
 - overview of, 182
- frame rate, FPS Creator, 427
- frames, TGF2
 - Black Jack game, 298–299
 - Dragons game, 307
 - overview of, 181
 - placing active object on, 320
 - placing objects on, 318–319
- full-motion video (FMV) games, 32
- G**
- Gamasutra website, 469
- Game Complete, FPS Creator, 429
- The Game Creators website, 468
- game design elements, 89–103
 - audience, 93
 - design document, 93–96
 - determining market, 101–102
 - game idea and treatment, 92–93
 - game type, 91–92
 - overview of, 90–91
 - proposal issues, 101
 - required resources and scheduling, 103
 - team, 93
 - technical information and associated risks, 102
 - technology, 93
- game design elements, storyline, 96–102
 - heroes and enemies, 97
 - levels, 96
 - menu navigation, 98
 - miscellaneous and appendix, 100–101
 - music and sound effects, 99
 - user interface, 98–99
- game development, history of, 11–20
 - assembly language, 14–15
 - computer on a chip, 16
 - easier programming languages, 17–19
 - future of, 20–21
 - game consoles, 19–20
 - game genres. *see* genres, game
 - graphics, 16–17
 - polygons, 17
 - silicon circuits, 12
 - Spacewar, 12–14
- game frame, Alien Wars, 231–251
 - application size, 206–207
 - components, 232
 - creating and renaming, 207–208
 - creating enemy ships, 237–242
 - determining if music has stopped playing, 234
 - display on top, 236–237
 - enemy and bullet collisions, 245–247
 - enemy and spaceship collisions, 251
 - initial scene setup for, 212–214
 - player shooting, 242–244
 - position of bullets, 244–245
 - recreate enemy robots, 247
 - robot animations, 250
 - robot firing, 247–250
 - Start of Frame, 232–234
 - stopping ship from leaving scene, 234–236
- game frame, Black Jack, 302–304
- game frame, Dragons, 310
- game frame, Litter Bug, 274–294
 - cleaning fluid, 285–286
 - collisions between cleaning fluid and dirt, 286–287
 - computer players group, 287–291
 - defined, 258
 - end code, 292–294
 - guardrails, 274–276
 - initial scene setup for, 260–267
 - movement, 278–285
 - order, 276–278
 - Start of Frame event, 274
 - game idea, 92–93
 - Game Information, 169–170
 - Game Items selection screen, 3D Gamemaker, 444–445
 - Game Maker 7.0 Lite, 105–116
 - creating shoot-em-up game with. *see* End of Earth
 - desktop icon, 108–109
 - DVD-ROM for this book, 482
 - installing, 106–108
 - interface, 109–110
 - menus, 112–116
 - Resource Explorer, 110–111
 - system requirements, 109
 - website, 468
 - Game Maker 7.0 Lite, first project, 117–127
 - creating simple program, 120–126
 - events, 119
 - objects, 118–119
 - overview of, 118
 - saving and running, 126–127
 - sprites, 119
 - Game Maker Language (GML), 168–169
 - game making websites, 463–470
 - ACID Planet, 469
 - Awesome Programming, 464
 - Caligari, 468–469
 - Click Convention, 466, 468
 - Clickteam, 465–466
 - The Daily Click, 470
 - eJay, 466–467
 - Gamasutra, 469
 - The Game Creators, 468
 - Game Maker, 468
 - Make Amazing Games, 465
 - Power Users Guide, 465
 - Retro Remakes, 470
 - Turbo Squid, 469
 - Game Over, FPS Creator, 429
 - game studio setup, 1–9
 - backup devices, 6–7
 - computer, 3–4
 - digital camera, 5
 - Ethernet network, 7
 - good chair and desk, 8
 - graphic (video) cards and 3D cards, 4–5
 - graphic tablets, 7
 - modem, 5–6
 - network, 7
 - other peripherals, 5
 - overview of, 2–3
 - processor, 4
 - RAM, 4
 - scanner, 5
 - tips for purchasing, 8–9
 - wireless network, 8
 - game treatment, 92–93
 - The Games Factory 2. *see* TGF2 (The Games Factory 2)

- gameSpace Lite, 450–462
 - creating primitives, 456–459
 - DVD-ROM for this book, 484
 - exporting model, 462
 - installing, 451–454
 - interface, 454–456
 - overview of, 450
 - simple 3D example, 459–462
 - system requirements, 450
- GeForce graphic cards, 5
- genres, game, 21–34
 - adventure games, 32
 - battle card games, 24
 - board games, 22
 - card games, 22–23
 - choosing game type from, 91–92
 - crossover and combining of, 34
 - design document, first
 - person shooter, 471
 - desktop toys, 33
 - educational games, 32
 - edutainment games, 32
 - fighting games, 25–26
 - First Person Shooter (FPS)
 - 3D games, 30
 - first-person 3D vehicle games, 30–31
 - flight sims, 28
 - full-motion video (FMV) games, 32
 - game market data on, 101–102
 - maze games, 21
 - overview of, 21
 - puzzle games, 24
 - quiz games, 24
 - racing games, 27
 - real-time strategy games, 28–29
 - role-playing games (RPGs), 31
 - screen savers, 33
 - shoot 'em ups, 25
 - side scrollers, 25–26
 - Sims, 30
 - sports games, 33
 - third-person 3D games, 31
 - turn-based strategy games, 28
- Geosphere primitives, 456–457
- Global Script, FPS Creator, 428
- Global Settings screen, 3D
 - Gamemaker, 444–445
- global values, TGF2
 - collisions between enemy and bullet, 246
 - creating enemy ships, 238–239, 241–242
 - setting up for storing scores, 232
- GML (Game Maker Language), 168–169
- gradient background, 322
- Graetz, J. Martin, 12–13
- Grammy Awards, music and sound of video games, 60
- graphic (video) cards, 4–5, 20
- graphic tablets, 7
- graphics, 35–57
 - adding multiple images, 132–133
 - adding using Open File dialog, 132–133
 - advances in, 16–17
 - anti-aliasing technique, 55
 - aspect ratio and, 40–42
 - color and, 42–45
 - Copy command, 46
 - Crop command, 48–49
 - Cut command, 46
 - design document, first
 - person shooter, 473, 477–478
 - Flip commands for, 49–50
 - formats, 56–57
 - FPS Creator, 362
 - masks, 52–54
 - opacity, 54
 - Paste command, 46
 - pixels, 38–39
 - programming guardrails, 274–276
 - Resize command, 47–48
 - resolution, 40
 - Rotate command, 47
 - sights, 36–38
 - Skew command, 47
 - sound vs., 60–61
 - sprites, 50
 - vector, 13
- graphics accelerator cards, 20
- grid option, backgrounds, 144
- Group Neutral, TGF2, 276–278
- groups, code
 - computer players, 287–291
 - creating, 199–200
 - creating enemy ships using, 237–242
 - Dragons game Event Editor, 311
- guardrails, programming, 274–276
- guns, FPS Creator
 - adding enemy, 385–386
 - adding player, 383–385
 - design document for, 474
 - performance checking, 427
- gunshots. *see also* bullets
 - programming, 159–161
 - programming bullet collision events, 165
 - programming bullet events, 163
 - recording sound for, 63
- H**
- hard disk drives, 7
- Hardware and Sound option, Windows Vista, 65
- HD DVD (High Definition) format, 6
- heal zone, 418–420
- health events
 - adding heal zone, 419–420
 - adding hurt zone, 419
 - adding to room, 146
 - for collision events, 158–159
 - programming, 167
 - setting number of, 152
- Help & Tutorials, eJay 7, 82
- help file
 - creating, 169–170
 - Game Maker, 116
 - gameSpace Lite, 453–454
 - TGF2 tutorial, 179–189
- heroes, in design document, 97–98
- High Color (16-bit color), 44–45
- High Definition (HD DVD) format, 6
- Higher Speed box, Direction Options tab, 357
- high-level programming language, 18
- highscore frame
 - Alien Wars, 214–216, 252–253
 - defined, 258
 - Dragons, 314–315
 - Litter Bug, 267–268, 294–295
- hi-score objects, 323–325
- history, of games. *see* game development, history of
- hosts, multiplayer online games, 433–435
- Hot Spot, 352–353
- HTML (Hyper Text Markup Language), 18
- hubs, Ethernet networks, 7
- human-readable languages, 14–15, 18
- hurt zone, 418, 419
- Hyper Text Markup Language (HTML), 18
- I**
- I, Robot game, 17
- ice events
 - adding to room, 145
 - destroying bullet once it hits, 163
 - events and actions for, 163–166
 - programming, 163–166
 - programming spaceship collisions, 158
- icons
 - 3D Gamemaker desktop, 440
 - FPS Creator, Add New Prefab, 376
 - FPS Creator desktop, 364–365
 - Game Maker desktop, 108–109
 - gameSpace Lite desktop, 452
 - sound control, 65
- icons, movement
 - Bouncing Ball, 330
 - Eight Directions movement, 335
 - Mouse Controlled movement, 334
 - Path Movement, 331
 - Pinball movement, 333
 - Platform Movement, 337
 - Race Car movement, 336
- icons, Picture Editor tools
 - Action Point, 353
 - Clear image option, 343
 - Color Picker, 348
 - Ellipse tool, 349
 - Eraser tool, 352
 - Export option, 344
 - Fill tool, 350
 - Flip Horizontally and Flip Vertically, 346
 - Hot Spot, 353
 - Import option, 343
 - Line tool, 348
 - Options, 345
 - Pen tool, 348
 - Polygon tool, 349
 - Rectangle tool, 349
 - Rotate tool, 352
 - Selection tool, 348
 - Shape tool, 350
 - Size, 352
 - Spray tool, 351
 - Text tool, 351
 - Transparency tool, 347
 - Undo and Redo, 346
 - Zoom Control, 347
- icons, TGF2
 - active object, 319
 - backdrop object, 321
 - Event Editor, 193, 194–195
 - hi-score object, 323–324
 - lives object, 326
 - score object, 328
 - text object, 325
- IDE (Integrated Development Environment), Game Maker, 110
- idea, game, 92–93
- images. *see* graphics
- importing
 - in FPS Creator, 362
 - Picture Editor Import tool, 343–344
- Information dialog, TGF2, 176–177
- in-game assets, screen layout, 36
- integrated circuits, 12
- Integrated Development Environment (IDE), Game Maker, 110
- Intel, 4, 16
- interactive, defined, 45
- interface. *see* user interface
- Internet
 - creating multiplayer online games, 432–436
 - FPS Creator support for, 362
 - obtaining samples from, 68
- interpolation, 39
- intersect boundary, 157
- items, Frame Editor, 191–192
- J**
- Jaws movie, 61

- K**
Kung Fu game, 26
- L**
L global value, 238–239, 241–242
Layer object, 260, 266, 276–278
Layer Toolbar, 276–278
layers, FPS Creator, 369
Legends of the Five Rings battle card game, 24
level boss, 3D Gamemaker, 444, 446–447
level numbers, TGF2, 238–239, 241–242
Level Placements group, TGF2
 creating enemy ships, 237–243
 player shooting, 242–244
 recreating enemy robots, 245
level profiler, FPS Creator, 426–428
levels
 adding win zone to completed, 424
 designing, 96
 as frames in TGF2, 181
 Game Maker rooms corresponding to, 118
 as layers in FPS Creator, 369
 testing in FPS Creator, 380–382
library files
 Alien Wars, game frame, 212–213
 Alien Wars, Highscores frame, 215
 Alien Wars, main menu frame, 208–209
 Alien Wars, working with, 204–205
 Litter Bug, game frame, 262–263
 Litter Bug, Highscores frame, 267–268
 Litter Bug, menu frame, 259–260
 Litter Bug, working with, 212–214
 for sound effects, 61
Library toolbar, FPS Creator, 371–372, 376–379
License Agreement
 3D Gamemaker installation, 438
 ACID XPress installation, 69–71
 eJay 7 installation, 80
 FPS Creator installation, 364
 gameSpace Lite installation, 451
 TGF2 installation, 176
licenses, MP3, 78
life events, Game Maker
 actions for collision events, 158–159
 adding to room, 146
 importing sprites, 135–137
 programming, 167–168
 setting number of lives, 152
lighting, FPS Creator, 400–401, 427
Line In, 64
Line Out, 64
Line tool, Picture Editor, 348
linear routes, 418
Litter Bug, 255–295
 creating new file, 258
 game frame scene setup, 260–267
 game window size setup, 258–259
 Highscores frame scene setup, 267–268
 Library, 256–257
 menu frame programming, 269–273
 menu frame scene setup, 259–260
 storyline, 256
Litter Bug, game frame programming, 274–294
 cleaning fluid, 285–286
 cleaning fluid and dirt collisions, 286–287
 computer players group, 287–291
 end code, 292–294
 guardrails, 274–276
 movement, 278–285
 order, 276–278
 overview of, 274
 Start of Frame event, 274
lives object, TGF2, 326–327
logos, screen layout, 36
loops
 creating music from, 68
 used in Black Jack, 306
- M**
machine language, 14–15
machine noises, recording, 63
Magic The Gathering, 24
main frame. *see also* Alien Wars, main frame programming
 overview of, 217
 setting up, 208–212
Make Amazing Games in Minutes (Charles River Media), 465
Make Amazing Games website, 465
markers, FPS Creator
 adding heal zone, 419–420
 adding hurt zone, 419
 adding player's starting position with, 382–383
 adding sound zone, 420–421
 adding story zone, 422
 adding win zone, 424
 defined, 366
 in Library toolbar, 371–372
 lighting rooms and corridors, 401
market, game, 101–102
masks, 52–54
Materials toolbar, gameSpace Lite, 461
maze games, 21
memory, FPS Creator performance, 427
memory, RAM, 4, 12
menu buttons, FPS Creator, 368–371
menu frame
 Black Jack, 300–302
 defined, 258
 Dragons, 308–309
 initial scene setup for Litter Bug, 259–260
menu navigation, design document, 98
menu screen layout, 36
menus, Game Maker, 112–116
metal oxide semiconductor (MOS) technology, 16
microchips, 12
microphones, 65, 66–67
microprocessors, 16
Microsoft Flight Simulator X, 28
MIDI/Game Port, 64
Minskytron, 13
Mixer, ACID XPress, 75
mixer panel, sound card, 65–66
modems, game studio setup, 5–6
MOS (metal oxide semiconductor) technology, 16
motif, quick backdrop object, 322
mouse, FPS Creator
 adding prefab room, 376–379
 configuring right button on, 345
 cursor in 3D world editor, 373–374
 removing object from cursor, 379
Mouse Controlled movement, 334
mouse pointer, in TGF2
 button effects, 227–229
 creating even conditions, 282–283
 defined, 195
 going back to start of game, 252–253
 not over object, 271–272
 over object, 270–271
 player shooting, 243
 Tape Mouse function, Path Editor, 332
 user clicks, 229–231, 273
Move Fixed dialog, Gamemaker, 153–156, 163
movement, 328–339
 animations within types of, 359
 Bouncing Ball, 329–331
 creating computer players group, 287–291
 creating own, 278–285
 Eight Directions movement, 335
 Mouse Controlled, 334
 multiple movement, 338–339
 overview of, 328–329
 Path movement, 331–333
 Pinball movement, 333–334
 Platform movement, 337–338
 Race Car movement, 336–337
 Movement_Map, Litter Bug game, 262, 263, 282–283
MP3 format, using, 78
Ms. Pac-Man, 21
multiplayer games
 creating FPS, 432–436
 design document, FPS, 472–473
 designing, 100
 multiple movement, 338–339
 multiple-choice quiz games, 24
music
 adding in TGF2, 219–224, 234, 252, 269–270
 adding to rooms, 137–139
 adding using script, 168–169
 creating, 67–68
 in design document, 99, 474, 477
 importance of, 60
 obtaining or creating, 61–62
 recording, 62–67
 types, 60–61
 types of sounds, 60–61
music, with ACID XPress, 69–78
 installing, 69–71
 requirements for, 69
 running for first time, 71–73
 simple creation in, 75–78
 touring, 73–75
music, with eJay Dance 7, 79–87
 exporting songs, 86–87
 installing, 79–81
 requirements for, 79
 running for first time, 81–83
 simple creation in, 84–86
 touring, 83–84
- N**
naming conventions
 adding sprite to project, 135
 game frames, 207–208
 rooms, 142–144
 saving 3D Gamemaker game, 445
NARAS (National Academy of Recording Arts and Sciences), 60
National Academy of Recording Arts and Sciences (NARAS), 60

- natural light, in games, 400
- navigation buttons, programming, 147–150
- Negate option, 228, 279–280
- NES (Nintendo Entertainment System), 19–20
- networks
 - creating multiplayer online game, 432–436
 - game studio setup, 7–8
- New Condition option, Event Editor, 197–198
- Nintendo Entertainment System (NES), 19–20
- No Key event, 155
- No More Lives, 167–168
- nonlinear routes, 418
- note events, 217–219
- number column, Storyboard Editor, 188
- numbering system, creating rooms, 142
- NVIDIA?, 5
- O**
- Object-Oriented Programming (OOP), C++, 18–19
- objects
 - FPS Creator, design document, 473
 - FPS Creator, included in, 362
 - Game Maker, adding, 120–126, 140–142
 - Game Maker, overview of, 118–119
 - TGF2, advanced control of, *see* advanced control of objects
 - TGF2, conditions of, 199
 - TGF2, creating enemy robot, 238–243
 - TGF2, display always on top of screen, 236–237
 - TGF2, game frame setup, 213–214, 264–267
 - TGF2, highscores frame setup, 215–216
 - TGF2, in Event Editor, 193–195
 - using sprites, 119
- Obstacle Type, backdrop and quick backdrop objects, 322–323
- Obstacles selection screen, 3D Gamemaker, 444
- OK button, FPS Creator test game, 381
- online resources. *see also* game making websites
 - FPS Creator full version, 364
 - Game Maker, 106
 - game market data, 102
 - gameSpace Lite, 450, 456
 - MP3 license costs, 78
 - using loops, 68
- OOP (Object-Oriented Programming), C++, 18–19
- opacity, applying, 54
- Open File dialog, 132–133
- Options, Picture Editor, 345
- OS (operating system), purchasing, 4
- overall time, in FPS Creator, 427
- P**
- Pac-Man, 17, 21
- Paint Only Segment walls option, 390, 393
- paint tools, FPS Creator, 391–394
- palette masking, 53–54
- pasting, 46, 345
- Path Editor, 331–333
- Path movement, 331–333
- PDA's (personal digital assistants), 61, 63–64
- PDP (Programmed Data Processor), 12–14
- Pen tool, Picture Editor, 348
- performance checking, FPS Creator, 426–428
- peripherals, game studio setup, 5
- personal digital assistants (PDAs), 61, 63–64
- physics engine, FPS Creator, 362
- Picture Editor, 342–354
 - accessing in quick backdrop object, 322
 - Color Palette, 353–354
 - creating animation tracks for each direction, 337
 - drawing area, 353
 - overview of, 342
- Picture Editor, tools
 - Clear, 343
 - Color Picker, 347–348
 - Crop tool, 346
 - Cut, Copy, Paste and Delete, 345
 - Ellipse tool, 349
 - Eraser tool, 351–352
 - Export, 344
 - Fill tool, 350
 - Filled Ellipse tool, 349
 - Filled Rectangle tool, 348
 - Flip Horizontally, 346
 - Flip Vertically, 346
 - Import, 343–344
 - Line tool, 348
 - Options, 345
 - Pen tool, 348
 - Polygon tool, 349
 - Rectangle tool, 348–349
 - Redo tool, 346
 - Rotate tool, 352
 - Selection tool, 347–348
 - Shape tool, 350
 - Size tool, 351–352
 - Spray tool, 350–351
 - Text tool, 351
 - Transparency tool, 347
 - Undo tool, 346
 - view Action Point, 353
 - view Hot Spot, 352–353
 - Zoom control, 347
- Pinball movement, 333–334
- pixels
 - color depth and, 45
 - computer graphics and, 38
 - interpolation and, 39
 - in resolution, 40
- Plane primitives, 457–458
- Platform movement, 337–338
- Play button
 - Game Maker, adding as object, 141
 - Game Maker, importing sprites, 135–136
 - Game Maker, programming, 149
 - TGF2, programming user clicks, 229–230
 - TGF2, turning animation on and off for, 227–228
- Play Control Balance slider, 66, 135–136
- Play Game, 445–447
- play testers, 479
- playback volumes, sound card mixer panel, 66
- Player Bullet screen, 3D Gamemaker, 442
- player shooting, TGF2
 - cleaning fluid, 275
 - collisions between enemy and bullet, 245–247
 - position of bullets, 244–245
 - programming game frame, 242–244
- player starting position, FPS Creator, 382–383
- Player_ship sprite, Game Maker, 135–136. *see also* spaceship events, programming
- plus (+) sign, 111
- Pokémon battle card game, 24
- Polygon tool, Picture Editor, 349
- polygons
 - graphics accelerator cards rendering, 20
 - history of games, 17
 - performance in FPS Creator, 426–427
 - position, performance in FPS Creator, 427
- positional masking, 53–54
- Possible Directions option, 335
- The Power Users Guide to Windows Development* (Charles River Media), 465
- Power Users Guide website, 465
- prefabs, FPS Creator
 - adding door switches, 398
 - adding windows, 396
 - creating corridor with single segments, 387–391
 - creating room, 376–379
 - defined, 365
 - in Library toolbar, 371–372
- primitives
 - 3D example of, 459–462
 - creating, 456–459
 - defined, 454
 - processors, 4, 19–20
 - producer, design document, 479
- Programmed Data Processor (PDP), 12–14
- programmer, design document, 476
- programming languages
 - assembly language, 14–15
 - development of easier, 17–19
- proofreading, design documents, 95
- properties
 - adding to objects, 140–142
 - backdrop object and quick backdrop object, 321–322
 - background, 140
 - Black Jack, 298–300
 - changing active object, 320
 - Dragons application, 307–308
 - importing sprites, 135–137
 - object, 121–125
 - Pinball movement, 333–334
 - room, 125–126
 - sound zone, 420–421
 - sprite, 121–122, 131–132
- Properties window, TGF2, 205–207
- proposal, basic game, 101
- PSTN (standard telephone) line modems, 5–6
- puzzle games, 24, 91
- Q**
- Q key, FPS Creator test game, 381
- quick backdrop objects, 321–323
- Quit button
 - Game Maker, adding as object, 141
 - Game Maker, importing sprites, 135
 - Game Maker, programming, 149–150
 - Game Maker, properties and image for, 136
 - TGF2, placing, 259–260
 - TGF2, programming user clicks, 230–231, 273
 - TGF2, turning animation on and off, 228–229
 - TGF2, when mouse is over object, 270–271
- quiz games, 24, 32
- R**
- R key, FPS Creator
 - creating corridor, 387, 389–391
 - defined, 379
 - test game, 381
- Race Car movement, 336–337
- racing games, 27
- Radeon line of graphic cards, 5

- rain, recording sound, 63
- RalliSport Challenge, 27
- RAM (random access memory), 4, 12
- random access memory (RAM), 4, 12
- real-time strategy games, 28–29
- recording devices, 64
- recording sounds, 62–67
 - from household items, 62–63
 - overview of, 61–62
 - using PDA, 63–64
 - using recording device, 64–65
 - using sound card's mixer panel, 65–66
 - using Windows sound recorder, 66–67
- Rectangle tool, Picture Editor, 348–349
- Redo tool, Picture Editor, 346
- Regenerate enemy comment, 247–250
- registration, ACID XPress, 71–73
- relative box, 160
- remote items, 391
- Rename option, game frames, 207–208
- render, defined, 45
- Render button, gameSpace Lite, 461–462
- required resources, game proposal, 103
- requirements
 - 3D Gamemaker, 438
 - ACID XPress, 69
 - eJay Dance 7, 79
 - FPS Creator, 363
 - Game Maker, 109
 - gameSpace Lite, 450
 - TGF2, 175
 - using DVD-ROM for this book, 481
- Resize command, 47–48
- resolution
 - computer graphics and, 40
 - setting up game in TGF2, 205–206
- Resource Explorer, Game Maker, 110–111, 125–126
- Resources menu, Game Maker
 - Create Background, 139–140
 - Create Object, 140–142
 - Create Room, 142
 - Create Sound, 138–139
 - Create Sprite, 121, 132
 - defined, 114
- retro gaming, 186
- Retro Remakes website, 470
- RGB (Red, Green, Blue) values, 42–43
- room start event, configuring, 152
- rooms, FPS Creator
 - adding windows, 396–398
 - creating corridor with paint tools, 391–394
 - creating corridor with single segments, 387–394
 - creating for basic game, 376–379
 - lighting, 400–401
- rooms, Game Maker
 - adding music, 137–139
 - creating, 125–126
 - defined, 118
 - objects within, 118–119
 - placing sprites/objects in, 142–147
 - transitions between, 147–148
- Rotate command, 47
- Rotate tool, Picture Editor, 352
- Rounded Cube primitives, 457
- Rounded Cylinder primitives, 457
- RPGs (role-playing games), 31
- Run Application button, Frame Editor, 193
- Run Frame button, TGF2, 193, 216
- Run menu, Game Maker, 115, 127
- Russell, Stephen R. "Slug", 13
- S**
- S key, FPS Creator test game, 380
- samples, obtaining, 68, 484
- scanners, game studio setup, 5
- schedule, game proposal, 103
- score
 - Game Maker, collisions, 165
 - overview of, 328
 - TGF2, collisions, 246, 286–287
 - TGF2, setting global values for, 232–234
- screen
 - displaying objects always on top of, 236–237
 - placing enemy ships on, 237–242
 - preventing object from leaving, 234–236
- screen resolution
 - TGF2, for Alien Wars, 190, 215–216
 - TGF2, for Litter Bug, 258–259
- screen savers, 33
- scripting, FPS Creator, 362
- Scripts menu, Game Maker, 115
- segments, FPS Creator
 - adding windows, 396
 - creating corridor with paint tools, 392
 - creating corridors with single, 387–391
 - defined, 365
 - in Library toolbar, 371–372
 - Segment toolbar, 370
- Selection tool, Picture Editor, 347–348
- serial key, gameSpace Lite, 454
- Set Variable dialog, 151–152
- shadows, adding, 262, 264, 276–277
- Shape tool, Picture Editor, 350
- shelves, in FPS Creator, 405–406
- shelves, in TGF2, 264–265
- Shift key, FPS Creator, 381
- Shoot an Object dialog, 244–245
- shoot-em-ups
 - in Game Maker. *see* End of Earth
 - overview of, 25
 - in TGF2. *see* Alien Wars
- Shooting Animation, 248–250
- Show Me How dialog, ACID XPress, 73
- side scroller genre, 25–26. *see also* Dragons
- sights, computer graphics and, 36–38
- silicon circuits, 12
- sim game genre, 30, 91
- The Sims game, 30
- single player game, designing, 100
- Size tool, Picture Editor, 351–352
- Skew command, 47
- Small Office Home Network (SOHO) network system, 7
- smoke effect, 402–405
- SOHO (Small Office Home Network) network system, 7
- solid-state computers, 12
- Song Arranger, eJay 7, 83–84
- Sony, ACID Music Studio, 68
- sound
 - collisions between enemy and bullet, 246
 - in design document, 99, 474, 477
 - history of game consoles, 20
 - importance of, 60
 - obtaining or creating, 61–62
 - recording, 62–67
 - in Samples folder, DVD-ROM for this book, 484
 - types of, 60–61
 - using script to add, 168–169
- sound, with ACID XPress, 69–78
 - installing, 69–71
 - requirements for, 69
 - running for first time, 71–73
 - simple creation in, 75–78
 - touring, 73–75
- sound, with eJay Dance 7, 79–87
 - exporting songs, 86–87
 - installing, 79–81
 - requirements for, 79
 - running for first time, 81–83
 - simple creation in, 84–86
 - touring, 83–84
- Sound and Audio option, Windows XP, 65
- Sound Blaster, 20
- sound cards, 19, 64–66
- sound clip archive, eJay 7, 85
- sound zone, 418, 420–421
- Space Battle, 24
- Space Invaders, 24, 186
- space scene, Frame Editor, 191–192
- Spacebar key, FPS Creator test game, 381
- spaceship events, programming, 150–163
 - adding to room in End of Earth, 146
 - collisions with ice asteroids, 158
 - configuring room start, 152
 - controlling movement, 152–157
 - creating alarm, 162
 - creating enemy ships, 237–242
 - firing gun, 159–161
 - no more health, 158–159
 - no movement of spaceship, 155–157
 - overview of, 150
 - preventing spaceship from leaving screen, 234–236
 - setting up animation, 233–234
 - setting up correct animations, 150–152
 - stopping from leaving screen, 157
- Spacewar
 - history of, 12–14
 - shoot 'em up genre, 24
 - written in assembly language, 15
- speeds, animation, 357
- Spellfire battle card game, 24
- Sphere primitives, 456–458
- sports games, 33
- Spray tool, Picture Editor, 350–351
- spreadsheet, Event Editor as, 183
- Sprite Editor, 132–133
- Sprite Properties menu, 131–132
- sprites
 - creating first game, 121–122
 - creating objects, 135–136
 - for game graphics, 17
 - importing, 131–137
 - numbering system for, 121
 - overview of, 50, 119
 - setting up correct animations, 150–152
- stairs, 408–409

- standard telephone (PSTN)
 - line modems, 5–6
 - Standard toolbar, FPS Creator, 368–369
 - Start button
 - placing, 259–260
 - programming user clicks, 273
 - when mouse is over object, 270–271
 - Start of Frame condition, 197–198
 - Start of Frame event
 - game frame, 232–234, 274
 - main frame, 219–223
 - menu frame, 269–270
 - Static movement, 329
 - static text object, 325–326
 - status bar, FPS Creator, 374
 - story zone
 - adding, 421–424
 - defined, 419
 - placement of, 422
 - Storyboard condition, 198
 - Storyboard Controls
 - programming to go back to start, 252–253
 - programming user clicks, 229–231, 273
 - Storyboard Editor, 180–182, 188–190
 - storyline
 - design document, 471–472
 - heroes and enemies, 97–98
 - levels of game, 96
 - menu navigation, 98
 - miscellaneous and appendix area, 100–101
 - music and sound effects, 99
 - single or multiplayer, 100
 - user interface, 98–99
 - storyline, for projects in this book
 - Alien Wars, 186–187
 - End of Earth, 130–131
 - Litter Bug, 256
 - strategy games, 28–29, 91
 - string object, 325–326
 - String_Score, 261
 - styles, for text objects, 325–326
 - SuperVGA graphics standard, 20
 - surge protectors, 8–9
 - sync, checking in FPS Creator, 427
- T**
- tabs
 - active object, 320–321
 - backdrop object and quick backdrop object, 321–323
 - hi-score object, 324
 - lives object, 326
 - team members
 - design documents and, 93, 95
 - game proposal about, 102
 - maps of levels for, 96
 - technology
 - choosing game, 93
 - game proposal, 102
 - teleporters, FPS Creator, 412–415
 - terminology, FPS Creator, 365–366
 - Test Compile dialog box, 380
 - test game
 - 3D Gamemaker, 446–447
 - design document, 479
 - FPS Creator, 371
 - Test Level button, FPS Creator, 380–382, 384–385
 - Tetris, 24
 - text, 325–326. *see also* comments, TGF2
 - text menus, FPS Creator, 368
 - Text tool, Picture Editor, 351
 - texture, 430–431, 461
 - TGF2 (The Games Factory 2), 173–184
 - about, 174–175
 - advanced games. *see* Black Jack; Dragons
 - advanced objects. *see* advanced control of objects
 - Alien Wars. *see* Alien Wars
 - animation tool, 354–359
 - DVD-ROM for this book, 483
 - Event Editor, 182–183
 - Frame Editor, 182
 - installing, 176–179
 - Litter Bug. *see* Litter Bug
 - maze games with, 21
 - requirements for, 175
 - starting for first time, 179–180
 - Storyboard Editor, 181–182
 - working with pictures in. *see* Picture Editor
 - TGFFILES folder, DVD-ROM for this book, 484
 - third-person 3D games, 31
 - thumbnails, Storyboard Editor, 188–189
 - thunder sound, recording, 63
 - Time Display window, ACID XPress, 74
 - Time Line window, ACID XPress, 74
 - timer (alarm), 161–162, 248
 - Title Screen, FPS Creator, 428
 - Tixo (TX-O) computer, 12–13
 - Torus primitives, 457
 - Track Header window, ACID XPress, 73
 - transitions
 - between rooms in Game Maker, 147–148
 - using Storyboard Editor, 189–190
 - transparency, in TGF2
 - adding comments, 224
 - setting for objects, 221–223
 - setting to O, 224–226
 - Trespasser, 61
 - trigger zone, 418
 - True Color (24-bit color), 44–45
 - Try Movement, 334
 - Turbo Squid website, 469
 - turn-based strategy games, 28
 - TX-O (Tixo) computer, 12–13
- U**
- Undo tool, 346
 - uninterruptible power supply (UPS), 8–9
 - UPS (uninterruptible power supply), 8–9
 - use items, 391
 - user clicks, programming, 229–231, 273
 - user interface
 - in design document, 98–99
 - Game Maker, 109–110
 - gameSpace Lite, 454–456
 - screen layout, 36
 - user-defined animations, 357
- V**
- variables, in Game Maker
 - predefined, 149
 - programming gunshots, 160–161
 - programming Quit button, 150
 - setting in actions, 148–149
 - setting up correct animations, 150–152
 - vector graphics, 13, 16–17
 - Vectrex, 16–17
 - Ventilation object, 387–391
 - VGA graphics standard, 20
 - video (graphic) cards, 4–5, 20
 - video files, story zone, 423–424
 - View toolbar, FPS Creator, 369
 - views, Action Point, 353
 - views, Hot Spot, 352–353
- W**
- W key, FPS Creator, 380
 - Wave Balance slider, 66
 - waypoints, 371, 415–417
 - weapons. *see also* bullets; gunshots
 - 3D Gamemaker, adding, 442
 - design document, first person shooter, 474
 - FPS Creator, adding, 383–385
 - Web Link, eJay 7, 83
 - websites. *see* game making websites; online resources
 - Welcome dialog, TGF2, 176
 - What You See Is What You Get (WYSIWYG) interfaces, 18
 - widescreen aspect ratio, 40–42
 - win zone, 418, 424
 - Window menu, Game Maker, 116
 - Windows
 - purchasing processors, 4
 - Vista, Hardware and Sound, 65
 - working with sound recorder, 66–67
 - XP, Sound and Audio, 65
 - windows, adding in FPS Creator, 396–398
 - Wing Commander, 28
 - wireless network, game studio setup, 8
 - Witanen, Wayne, 13
 - Workspace toolbar, Black Jack, 298–300
 - Workspace toolbar, Litter Bug
 - adjusting game window size, 258–259
 - creating new file in, 258–259
 - placing objects in game frame, 263
 - placing objects in High-scores frame, 268
 - placing objects in menu frame, 260
 - WYSIWYG (What You See Is What You Get) interfaces, 18
- X**
- X format, 450
 - X-Wing, 28
- Z**
- Zip drives, 6–8
 - zones, FPS Creator, 417–424
 - defined, 418–419
 - heal zone, 419–420
 - hurt zone, 419
 - sound zone, 420–421
 - story zone, 421–424
 - win zone, 424
 - Zoom control, 347

LIMITED WARRANTY AND DISCLAIMER OF LIABILITY

THE DVD THAT ACCOMPANIES THIS BOOK MAY BE USED ON A SINGLE PC ONLY. THE LICENSE DOES NOT PERMIT THE USE ON A NETWORK (OF ANY KIND). YOU FURTHER AGREE THAT THIS LICENSE GRANTS PERMISSION TO USE THE PRODUCTS CONTAINED HEREIN, BUT DOES NOT GIVE YOU RIGHT OF OWNERSHIP TO ANY OF THE CONTENT OR PRODUCT CONTAINED ON THIS DVD. USE OF THIRD-PARTY SOFTWARE CONTAINED ON THIS DVD IS LIMITED TO AND SUBJECT TO LICENSING TERMS FOR THE RESPECTIVE PRODUCTS.

CHARLES RIVER MEDIA, INC. ("CRM") AND/OR ANYONE WHO HAS BEEN INVOLVED IN THE WRITING, CREATION, OR PRODUCTION OF THE ACCOMPANYING CODE ("THE SOFTWARE"), OR THE THIRD-PARTY PRODUCTS CONTAINED ON THIS DVD, CANNOT AND DO NOT WARRANT THE PERFORMANCE OR RESULTS THAT MAY BE OBTAINED BY USING THE SOFTWARE. THE AUTHOR AND PUBLISHER HAVE USED THEIR BEST EFFORTS TO ENSURE THE ACCURACY AND FUNCTIONALITY OF THE TEXTUAL MATERIAL AND PROGRAMS CONTAINED HEREIN; WE, HOWEVER, MAKE NO WARRANTY OF THIS KIND, EXPRESS OR IMPLIED, REGARDING THE PERFORMANCE OF THESE PROGRAMS. THE SOFTWARE IS SOLD "AS IS" WITHOUT WARRANTY (EXCEPT FOR DEFECTIVE MATERIALS USED IN MANUFACTURING THE DISC OR DUE TO FAULTY WORKMANSHIP); THE SOLE REMEDY IN THE EVENT OF A DEFECT IS EXPRESSLY LIMITED TO REPLACEMENT OF THE DISC, AND ONLY AT THE DISCRETION OF CRM.

THE AUTHOR, THE PUBLISHER, DEVELOPERS OF THIRD-PARTY SOFTWARE, AND ANYONE INVOLVED IN THE PRODUCTION AND MANUFACTURING OF THIS WORK SHALL NOT BE LIABLE FOR DAMAGES OF ANY KIND ARISING OUT OF THE USE OF (OR THE INABILITY TO USE) THE PROGRAMS, SOURCE CODE, OR TEXTUAL MATERIAL CONTAINED IN THIS PUBLICATION. THIS INCLUDES, BUT IS NOT LIMITED TO, LOSS OF REVENUE OR PROFIT, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THE PRODUCT.

THE SOLE REMEDY IN THE EVENT OF A CLAIM OF ANY KIND IS EXPRESSLY LIMITED TO REPLACEMENT OF THE BOOK AND/OR DVD, AND ONLY AT THE DISCRETION OF CRM.

THE USE OF "IMPLIED WARRANTY" AND CERTAIN "EXCLUSIONS" VARY FROM STATE TO STATE, AND MAY NOT APPLY TO THE PURCHASER OF THIS PRODUCT.