

Making Everything Easier!™

GameMaker: Studio

FOR

DUMMIES®

A Wiley Brand

Learn to:

- Quickly design, develop, test, and publish your first game
- Use GameMaker Language Code to create more advanced games
- Publish your game on multiple platforms for all major devices

IN FULL COLOR!

Michael Rohde

Web development professional and writer



Get More and Do More at Dummies.com®



Start with **FREE** Cheat Sheets

Cheat Sheets include

- Checklists
- Charts
- Common Instructions
- And Other Good Stuff!

To access the Cheat Sheet created specifically for this book, go to
www.dummies.com/cheatsheet/gamemakerstudio

Get Smart at Dummies.com

Dummies.com makes your life easier with 1,000s of answers on everything from removing wallpaper to using the latest version of Windows.

Check out our

- Videos
- Illustrated Articles
- Step-by-Step Instructions

Plus, each month you can win valuable prizes by entering our Dummies.com sweepstakes. *

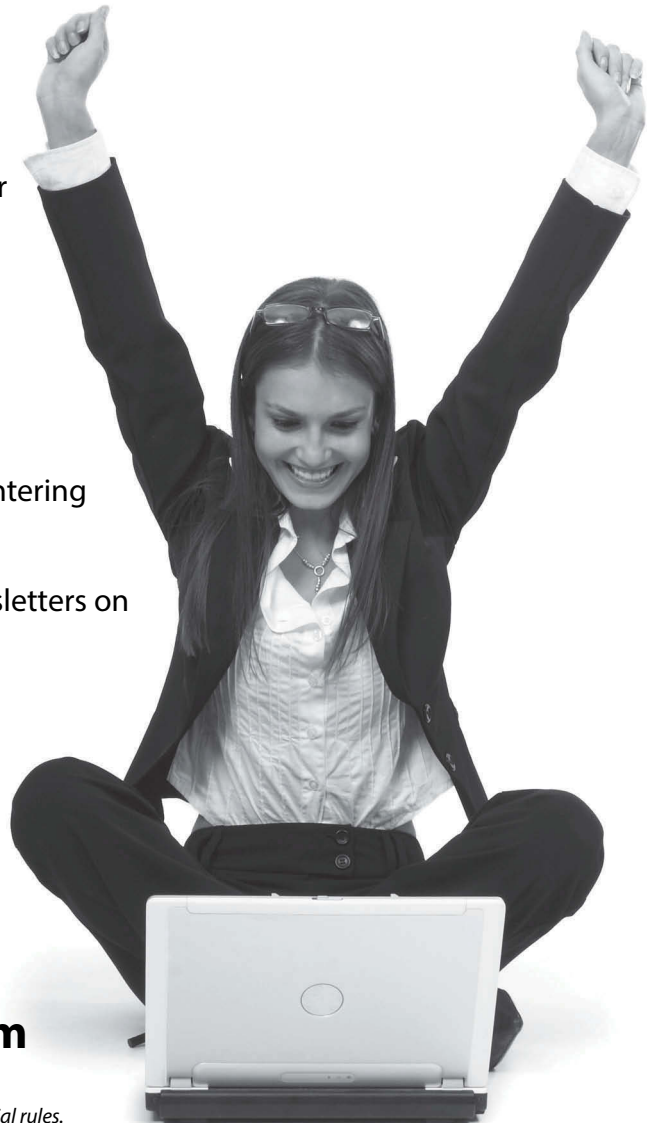
Want a weekly dose of Dummies? Sign up for Newsletters on

- Digital Photography
- Microsoft Windows & Office
- Personal Finance & Investing
- Health & Wellness
- Computing, iPods & Cell Phones
- eBay
- Internet
- Food, Home & Garden

Find out “HOW” at Dummies.com

*Sweepstakes not currently available in all countries; visit Dummies.com for official rules.

www.it-ebooks.info



GameMaker: Studio

FOR
DUMMIES®
A Wiley Brand

by Michael Rohde

FOR
DUMMIES®
A Wiley Brand

GameMaker: Studio For Dummies®

Published by: **John Wiley & Sons, Inc.**, 111 River Street, Hoboken, NJ 07030-5774, www.wiley.com

Copyright © 2014 by John Wiley & Sons, Inc., Hoboken, New Jersey

Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the Publisher. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Trademarks: Wiley, For Dummies, the Dummies Man logo, Dummies.com, Making Everything Easier, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and may not be used without written permission. GameMaker: Studio is a trademark of YoYo Games, Ltd. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services, please contact our Customer Care Department within the U.S. at 877-762-2974, outside the U.S. at 317-572-3993, or fax 317-572-4002. For technical support, please visit www.wiley.com/techsupport.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2014935513

ISBN 978-1-118-85177-7 (pbk); ISBN 978-1-118-85210-1 (ebk); ISBN 978-1-118-85215-6 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Contents at a Glance

<i>Introduction</i>	<i>1</i>
<i>Part I: Getting Started with GameMaker: Studio</i>	<i>5</i>
Chapter 1: Introducing GameMaker: Studio	7
Chapter 2: Discovering GameMaker: Studio Features and Tools	13
<i>Part II: Basic Tools and Techniques</i>	<i>31</i>
Chapter 3: Creating Events	33
Chapter 4: Adding Actions to Events	47
Chapter 5: Working with Views	77
Chapter 6: Working with Time Lines	99
<i>Part III: Creating Special Effects</i>	<i>125</i>
Chapter 7: Using Actions to Create Special Effects with Particles	127
Chapter 8: Coding Special Effects with Particles	151
Chapter 9: Using Surfaces to Create Effects	175
Chapter 10: Applying Scaling	201
<i>Part IV: Coding Physics, Mobile Controls, and More....</i>	<i>219</i>
Chapter 11: Coding with GameMaker Language	221
Chapter 12: Using Physics	247
Chapter 13: Creating Controls for Mobile Devices	265
Chapter 14: Saving and Loading	293
<i>Part V: The Part of Tens</i>	<i>311</i>
Chapter 15: Ten Tips for Game Developers	313
Chapter 16: Ten (Or So) Frequently Asked Questions about GameMaker: Studio	319
<i>Index</i>	<i>323</i>

Table of Contents

Introduction 1

About This Book	1
Foolish Assumptions.....	2
Icons Used in This Book	2
Beyond This Book	3
Where to Go from Here.....	3

Part I: Getting Started with GameMaker: Studio 5

Chapter 1: Introducing GameMaker: Studio 7

Dragging and Dropping Your Way to Making a Game.....	7
Understanding the GameMaker Language	9
Publishing Games to Different Platforms	9
Highlighting the Key Features of GameMaker: Studio	11
Creating a Game, Step by Step.....	11

Chapter 2: Discovering GameMaker: Studio Features and Tools . . . 13

Navigating the GameMaker Interface.....	13
Pump Up the Volume: Adding Sounds to Your Game.....	14
Making Things Sparkle with Sprites	17
Loading and creating Sprites.....	18
Editing Sprites	20
Configuring Sprites	21
Populating Your Game with Objects.....	26
Placing Objects in the Game Room	29
Creating a Room.....	29
Putting your Objects in the Room	30

Part II: Basic Tools and Techniques..... 31

Chapter 3: Creating Events 33

How to Add an Event to an Object	33
The Create Event	35
The Destroy Event	36
The Alarm Event	36
The Step Event	37

The Collision Event	38
The Mouse Event	39
The Other Event.....	39
The Draw Event.....	41
The Key Events	44
The Asynchronous Event	45

Chapter 4: Adding Actions to Events. 47

How to Assign an Action to an Object	47
Move Actions	48
Move Directional Actions	48
Jump Actions.....	51
Path Actions	52
Step Actions.....	54
Main1 Actions	55
Objects Actions.....	56
Sprite Actions	57
Sound Actions	57
Room Actions	58
Main2 Actions	59
Timing Actions	60
Info Actions.....	61
Game Actions	62
Resources Actions	63
Control Actions	64
Questions Actions.....	64
Other Actions	66
Code Actions	67
Variables Actions	68
Score Actions	69
Score Actions.....	69
Lives Actions	71
Health Actions	71
Extra Actions	72
Draw Actions.....	73
Drawing Actions.....	74
Settings Actions	75
Other Actions	75

Chapter 5: Working with Views. 77

Using Speed and Movement with Views	78
Creating a Room with a View	79
Using a Controller Object to Move the View.....	83
Reaching the End of the Room.....	84
Positioning the Score Display in a Room with a View	86

Setting Object Boundaries for Rooms with Views.....	87
Destroying Instances When They Leave a Room	89
Shooting Guided Bullets in a Room with Views.....	90
Creating Enemy Planes in a Room with Views.....	91
Creating Patterns of Enemy Planes in a Room with Views.....	95
Transitioning to the Next Room	96

Chapter 6: Working with Time Lines. 99

Creating a Time Line	100
Creating a Time Line Object.....	104
Creating the Time Line Room	106
Transitioning to the Next Room with Time Lines	111
Adding a Boss Battle to a Time Line	112
Creating the boss Sprite.....	113
Creating the boss Object	114
Adding the Boss to the Time Line.....	123

Part III: Creating Special Effects 125

Chapter 7: Using Actions to Create Special Effects with Particles. . . 127

Coding a Button to Trigger the Particles.....	128
Using the Create Effect Action	131
Using the Particle Actions and Emitters.....	138
Creating puffs of smoke	139
Creating the Catherine wheel firework	142
Creating an emitter for the firework.....	144

Chapter 8: Coding Special Effects with Particles 151

Coding the Particle System and the Smoke Trail	152
Coding the Simple Firework Effect	155
Coding Emitters	160
Coding burst emitters	160
Coding stream emitters.....	165
Coding Complex Particle Effects	169

Chapter 9: Using Surfaces to Create Effects 175

Understanding the Surface Function	175
The application Surface	176
Creating Shadows with Surfaces	176
Using Decals with Surfaces.....	182
Lighting Effects with Surfaces	185
Creating a flashlight effect	188
Adding darkness to the Surface	190
Using Surfaces as a View	194
Setting up the Room and an Object for Surface Views	194
Creating the Surface View effect	197

Chapter 10: Applying Scaling 201

Configuring the Global Game Settings	202
Scaling to the Window	204
Scaling to best fit.....	205
Scaling the View.....	207
Scaling on mobile devices.....	208
Scaling with Draw GUI Events	210
Maximizing the GUI.....	210
Hiding the letterbox.....	211
Placing GUI elements to create a heads-up display.....	214
HTML5 Scaling	215

Part IV: Coding Physics, Mobile Controls, and More 219**Chapter 11: Coding with GameMaker Language 221**

Displaying Sub-Images with Code.....	222
Randomly displaying sub-images	222
Sorting sub-images by index.....	226
Adding Code to Objects.....	227
Coding keyboard controls	228
Coding an Object with collisions and Step Events	230
Coding a Score Display and Lives.....	235
Coding to Start and End Games	240
Introducing the game	240
Ending the game.....	243

Chapter 12: Using Physics. 247

Understanding Fixtures	248
Defining the physics properties.....	249
Creating a fixture.....	250
Adding Movement to an Object with Physics.....	253
Example code for an Up Keyboard Event to apply local force	254
Example code for Left and Right Keyboard Events	255
Adding Physics to Rooms.....	257
Physics and Shooting Bullets.....	259
Defining bullets to work with Physics.....	259
Coding the ship to shoot bullets.....	262

Chapter 13: Creating Controls for Mobile Devices. 265

Putting Your Game on a Mobile Device	265
Using an emulator	265
Putting your game online	266
Adding Simple Touch Controls	267
Understanding virtual keys	267
Creating virtual keys	267
Adding Sprites to virtual keys	269
Adding Swipe Controls	272
Single-direction swipes	273
Directional swipes	276
Pinching and Zooming	278
Creating Analog Sticks for Mobile Devices	281
Drawing the sticks onto the GUI script	282
Making the player move	283
Shooting with the analog stick	289

Chapter 14: Saving and Loading. 293

Understanding the File System	293
Understanding INI Files	295
Using Real Values to Save Game Options	297
Making a button for the background music	297
Making a button for the sound effects	301
Using String Values to Save Game Options	302
Mapping the Gameplay Keys	305
Resetting the Default Keys	307

Part V: The Part of Tens 311**Chapter 15: Ten Tips for Game Developers 313**

Developing Cross-Platform Games	313
Knowing Your Audience	314
Monetizing Your Games	314
Choosing Your Game Genre	314
Making Your Game User-Friendly	315
Testing Your Game	315
Considering Limitations for Mobile Devices	316
Thinking Big	316
Expanding Your Opportunities	316
Going Indie	317



**Chapter 16: Ten (Or So) Frequently Asked Questions
about GameMaker: Studio 319**

What Types of Games Can I Make with GameMaker: Studio?	319
Can I Run GameMaker: Studio on a Mac?	319
How Do I Recover My License Key?	320
How Do I Include Advertising in My Game?	320
How Do I Activate In-App Purchases?	320
What's Involved with Publishing My Game?	320
When Is a New Feature Coming Out?	321
Where Can I Get More Game-Making Tips and Tricks?.....	321

Index 323

Introduction



My interest in GameMaker: Studio started in the summer of 2013. I was scrolling through Twitter when a buddy of mine mentioned using GameMaker to make a game. So, of course, I had to check it out. I was immediately hooked on the idea of creating games without needing to know how to code. I read through some of the tutorials and got a couple of basic games to work, and soon enough I found myself coding. From there, it was only a matter of time before I realized that I wanted to write a book on GameMaker: Studio.

I hope you're as excited as I am about creating video games. This book provides a ton of procedures on how to do everything from creating basic movements to creating some very special effects. I also provide overview concepts on how different parts of the game fit together. My hope is that you take the concepts you learn here and apply them to your own game. Who knows? Maybe one day we'll be playing each other's games on our favorite consoles and devices.

About This Book

This book shows you how to make video games, quickly and easily. You make games geared toward PC and mobile devices with the possibility of expanding to consoles. You create different effects that you can apply to your own games. You find out all about Events and Actions and Sprites and Objects and Instances and Scripts and Rooms and Targets. . . . I could go on and on. There is an absolute ton of information packed into these pages.

If you enjoy reading through tutorials, written in a clear manner, full of details, you'll appreciate *GameMaker: Studio For Dummies*. Much of the material I present here is stuff I learned from the GameMaker tutorials. But it was slow and hard going and I made a ton of mistakes. That's where this book comes in to play: I'm sharing with you the secrets and details necessary to make the tutorials work, so that you can use them on your own games.

GameMaker: Studio For Dummies is modular — you don't have to read straight through the book from Chapter 1 to the end. You can skip around and read the sections you're most interested in. I include plenty of cross-references, so you won't feel lost. I explain concepts a couple times in a couple different ways. That way, you can apply lessons learned in different situations. That should provide a solid foundation from which you can use GameMaker: Studio.

Foolish Assumptions

In writing this book, I made a few assumptions about you as the reader:

- ✔ You've probably been playing video games your entire life and now you want to make games. You want to make shooters, roguelike, and scrolling platform games.
- ✔ You want to play games on your iOS and Android devices.
- ✔ You may be envisioning getting your own Sony Developers License so that you can make games for the PS4 and Vita using GameMaker: Studio. Before you hit the big time, you need to start off slow, and that's why you most likely already downloaded the free version of GameMaker, if not the paid edition.
- ✔ You may or may not know the first thing about code, but if you do, you're one step ahead of the game. If you don't know how to code, you want to learn.
- ✔ You've already gone through some of the in-software tutorials. The first couple of tutorials were easy enough, but when you tried some of the more complex tutorials, well, that's when things got messy fast. At least, that's what happened to me. And that's why I wrote this book. Much of this book has you opening the in-software tutorial files, but I provide you the missing information, the step-by-step instructions that you want and deserve.

Basically, I'm envisioning you to be a go-getter who's ready to make and play some games.

Icons Used in This Book

I use several icons throughout this book to help you understand GameMaker: Studio. The following is an explanation of each of the icons.



Whenever I provide information that'll help you do something easier, faster, or better, I mark it with the Tip icon.



I still want to write a *For Dummies* book in which I can write a warning as to when pirates are about to attack, but that hasn't happened yet. Instead, anything marked with the Warning icon has to do with your game crashing — as in, make sure to do or not to do something, so that your game doesn't crash.



Whenever I remind you of something that I've already covered elsewhere, or tell you about something you should commit to memory, I mark it with the Remember icon.



The Technical Stuff icon mainly refers to details on code. You can skip anything marked with this icon, but I hope you read this stuff because it can help you grow as a developer.

Beyond This Book

In addition to the material in the print or e-book you're reading right now, this product also comes with some access-anywhere goodies on the web. Check out the free Cheat Sheet at www.dummies.com/cheatsheet/gamemakerstudio for helpful information, all provided in a concise, quick-access format. You can also find articles online that build on the material you find in this book. Go to www.dummies.com/extras/gamemakerstudio for these informative articles.

I could have written a 700-page book to cover everything that you can do with GameMaker: Studio. In fact, I've written several extra tutorials, which you can find at www.gamemaker-tutorials.com.

On the YoYo Games website (<http://www.yoyogames.com>), you can find developer resources, support, and forums where you can ask and answer questions. You can also find a showcase of games that people have made using GameMaker. There's a lot of cool stuff there — I recommend that you check it out if you haven't already.

Where to Go from Here

Like all authors, I would love it if you read this book from cover to cover, writing notes in the margins, highlighting passages, and dog-earing pages for quick reference later. Alas, you probably just want to know how to add virtual keys for mobile devices (see Chapter 13), or how to make an explosion look really cool (see Chapters 8 and 9). In that case, you can jump to those chapters. But if you get lost, you can start from the beginning to get the core concepts of what it takes to make a GameMaker game.

You may be tempted to skip Chapter 1, thinking it's all fluff, but it's not. In Chapter 1, I provide a procedure on how to choose a Target for a game. That's very important information, especially if you want to play your game in a browser and not on Windows.

See how I did that? I included a reference in the Intro, to get you reading Chapter 1, and now you've taken the next step to create, publish, and play your video games.

Part I

Getting Started with GameMaker: Studio



For Dummies can help you get started with lots of subjects. Visit www.dummies.com to learn more and do more with *For Dummies*.

In this part . . .

- ✓ Get to know GameMaker: Studio.
- ✓ See how Sounds, Sprites, Objects, and Rooms work together.

Introducing GameMaker: Studio

In This Chapter

- ▶ Coding without coding using Actions
- ▶ Coding by coding with GameMaker Language
- ▶ Getting your game into players' hands
- ▶ Identifying the major features of GameMaker: Studio
- ▶ Looking at the main steps of creating a game

GameMaker: Studio software, from YoYo Games, is a game development tool that enables you to make video games. At the Game Developers Conference (GDC), I had an opportunity to meet with key players from YoYo Games. I saw firsthand the level of excitement this team from Dundee, Scotland has. It's as if each of them embodied the spirit of a magical dragon, circling high overhead a misty mountaintop, ready to breathe down green flames of magical energy to level up all game developers with enhanced game-making attributes.

All game developers can benefit from GameMaker: Studio. Whether you're a professional developer looking to enter the videogame community, or a university student interested in breaking into the videogame field, or a 10-year-old with no coding experience at all, GameMaker: Studio has the tools you need to make your vision a reality.

Dragging and Dropping Your Way to Making a Game

You can make games with GameMaker: Studio without knowing code! That's because GameMaker uses the drag-and-drop Actions (see Chapter 4). You use Actions to tell GameMaker how, what, when, and where you want something to happen during your game. When you create an Action, GameMaker creates the code for you in the background. For example, say you create an Action

so that when the player presses the spacebar, she shoots bullets from her plane. When you make that Action, GameMaker creates the code that causes the plane to shoot a bullet when the player presses the spacebar. In that way, using the drag-and-drop Actions enables the non-developer to develop code.

The drag-and-drop Actions are also good for developers who *do* know how to code but want a quick and efficient way to write large blocks of code. For example, a developer could sit down and write out the code by hand, which can be timely and can lead to typos and errors. Or, she can use the drag-and-drop Actions to create those same blocks of code, which can take seconds instead of hours.

Looking to the future

When I met with members of the YoYo team during GDC, I walked away with the feeling that this is a team with a dream. While I spoke to them, in the background, they had a PS4 setup running games made with GameMaker. They were telling me that it's only a matter of time before everyone with a PS4 will be able to purchase and download those games. These are exciting times when a person with a Sony developers license can make her own games, whether it be from the office or the kitchen table, and then play it on a major console.

YoYo Games sees a day when any self-starter, of any age, can make a game with GameMaker: Studio, play it on his Nintendo 3DS, show it to his friends, and then possibly put it up for sale on the eShop. This is the direction we, as a gaming community, are headed: Independent developers have a spotlight shining on them, and it's time to show the world what you can do.

As you start to think about your own possibilities for making games, keep these words of inspiration fresh in your mind:

- ✓ Start with the free version of GameMaker: Studio, and then keep growing and adding on as you make your way to becoming a professional.

- ✓ Hard work is what it's all about — you'll get out of it what you put into it.
- ✓ Include your own special touch in each of your games to make them unique and special.
- ✓ You can work your way up in the game development industry. Start out as a tester and then climb the ladder to becoming a developer.
- ✓ You don't have to join a big company anymore to make a living making games. You just need the passion and desire to make the best game you can. The exploding indie scene opens game development up to everyone.

YoYo Games is looking forward to seeing the games you make so it can feature them on www.yoyogames.com. The YoYo Games website has a Showcase of games, along with YoYo Labs, where you can find open-source games. It also has a lively community in the forums and Help pages. Who knows? One day you might even see your game available for sale on your favorite console or handheld device!

Understanding the GameMaker Language

The drag-and-drop Actions provide a quick and easy way to create code. But you should start your game-making adventure with an open mind for learning how to code because, ultimately, it will benefit you and your games. Plus, by learning code, you gain a better appreciation for how the Actions work.

I'm not a developer, but I'm familiar enough with code that I can tweak existing code. Having this level of familiarity with how code works give you some background knowledge on what you're doing when you drag and drop an Action.



If you're a professional developer, and you want to write specific code for your own specific means, then GameMaker: Studio features a code-editing tool that most professional game developers should appreciate. Part IV of this book dives into the GameMaker language — it's filled with example code, including working with physics and controls for mobile devices. Chapter 14 includes sample code for saving game configurations such as muting the sound or customizing the game keys.

Publishing Games to Different Platforms

As of this writing, you can install GameMaker: Studio on a PC or on a Mac with a Windows emulator, but that doesn't mean you're stuck making only games for Windows. Oh no, my friends, that's far from the truth. When you're ready to invest in the Professional or Master Collection version of GameMaker: Studio, you open yourself up to a world of platforms on which you can publish your game.

Figure 1-1 shows the drop-down list showing the different platforms to which you can target your game.

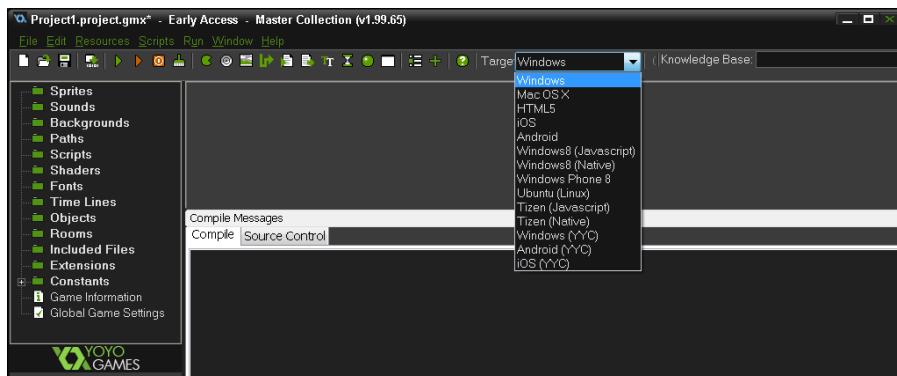


Figure 1-1: You have lots of options for where to publish your game.



If you don't see your favorite platform on this list, it's probably just a matter of time before it appears.

If you're wondering why you don't see PS4 or Vita in Figure 1-1, that's because you need to be a licensed Sony developer for that option to appear.

To publish a game to a specific platform, follow these steps:

1. **From the icon menu, click the Target drop-down list (refer to Figure 1-1).**

The Target options appear.

2. **Choose the platform you want to play your game on (for example, HTML5, if you want to be able to play it in a browser, such as Chrome or Firefox).**

The chosen platform appears in the Target field.

3. **Choose File → Create Application.**

A navigation window opens, allowing you to choose where to save the files. GameMaker names the file for you.

4. **Navigate to where you want to save the files, and then click Save.**

GameMaker saves the appropriate files for the platform you chose, as shown in Figure 1-2.

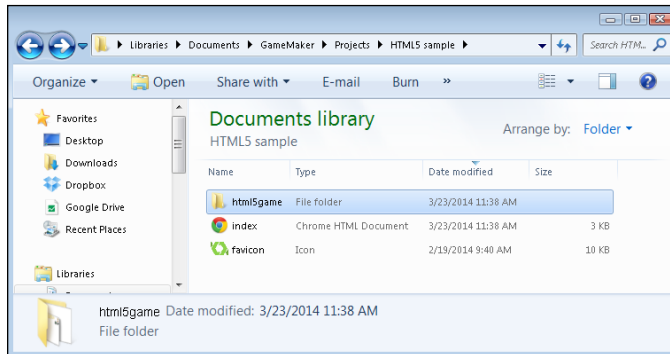


Figure 1-2: The saved game folder structure and files for HTML5 games.

Now you can take these saved games files and folders and use them to upload to a website, which is great for HTML5 games. I use FTP software such as FileZilla (<https://filezilla-project.org>) to do that.



You can update the favicon file to your own icon. The favicon icon appears in the tabs of most web browsers.

Highlighting the Key Features of GameMaker: Studio

GameMaker: Studio is so feature-rich, with such a multitude of tools, that I could write ten books on the subject! This book is excellent for those who want to learn what an Object is, and how Objects relate to Instances, and how you can assign Events and Actions to Objects to make them do things, like create a flower blossoming into a beautiful display of color, and then have the player blow it all up. You know, fun stuff like that. Objects and Actions are covered in detail in Chapters 3 and 4, respectively.

The interface for GameMaker: Studio is based on a series of windows. When you first open the software, you see a New Project window, from which you can do several things, such as open a tutorial, a demo, or a saved project, or start an entirely new project. When you start working on a project, the main window appears with a Resource tree, main menu, icon menu, and compile window. (I go into detail on each of these in Chapter 2.)

Another critical feature of GameMaker: Studio is the robust coding window, complete with color coding to highlight text. I discuss this in detail in Chapter 11. YoYo Games is planning an upgrade to this code window so that it supports tabs. This feature is not available as of this writing, but it should be out by the time you're reading this book.

Creating a Game, Step by Step

The steps I take to create a game are the same approach I use for most any creative project. First, you have to plan. Then you have to plan some more.

Here's a quick list of action items I take before creating a game:

- 1. Create an outline of the game, complete with objectives, characters, and game type.**
- 2. Think about how each level of the game will play.**

Too often, developers create an awesome first and second level, only to run out of ideas for consecutive levels.
- 3. Gather your assets, including images for Sprites, text, and any other resources you need for your game, such as music and sound effects.**
- 4. As you create your game based on your outline, start testing each new aspect.**
- 5. Test the game some more.**
- 6. Finally, test your game.**

Seriously. You can't test enough.

Discovering GameMaker: Studio Features and Tools

In This Chapter

- ▶ Getting acquainted with the interface
- ▶ Adding Sounds to your game
- ▶ Making things pretty with Sprites
- ▶ Bringing your game to life with Objects
- ▶ Putting your Objects in Rooms

In this chapter, I provide a broad overview of some of the most used features and tools of GameMaker. That includes the interface, Sounds, Sprites, Objects, Actions, and Events. I go into much more detail on Actions and Events in the following chapters, but here I show you how to get started with creating and loading Sprites, adding music as a resource, and creating an Object that you can place in a Room. Buckle up, folks — this ride is about to start!

Navigating the GameMaker Interface

The basic work area within GameMaker consists of several sections. These sections are laid out in an organized manner to form the interface from which you work. You see a main menu along the top and an icon menu directly below that; these two menus provide separate ways of, say, creating a Sprite, loading a Sound, creating an Object, and so on. The icon menu is where you can find the Target drop-down list, which is where you tell GameMaker which platform you want to publish to, such as HTML5, Android, iOS, and so on (see Figure 2-1).

Along the left pane, you can find the Resource tree, which contains a quick list of resources, including Sprites, Sounds, Backgrounds, Objects, Rooms, and more. The bottom pane consists of a Compile window, which blasts into



life when you run the game (meaning, when you run the game, you can watch GameMaker compile the game through a series of lines of text).

I tend to keep the Compile window closed while I'm working just to save screen space; however, if you have a large screen, you may find it useful when debugging and setting up extra Target modules, such as Android.

If you're working from a tutorial, in the right pane you see the tutorial window. Figure 2-1 highlights the various parts of the interface.

When you create a new Object, or a Sprite, or a Room, or what have you, a new floating window appears. This floating window could contain several tabs of configurations, or it could have radio buttons, text fields, drop-down icons, drag-and-drop icons, and more. It's within these floating windows where you do the majority of your work.

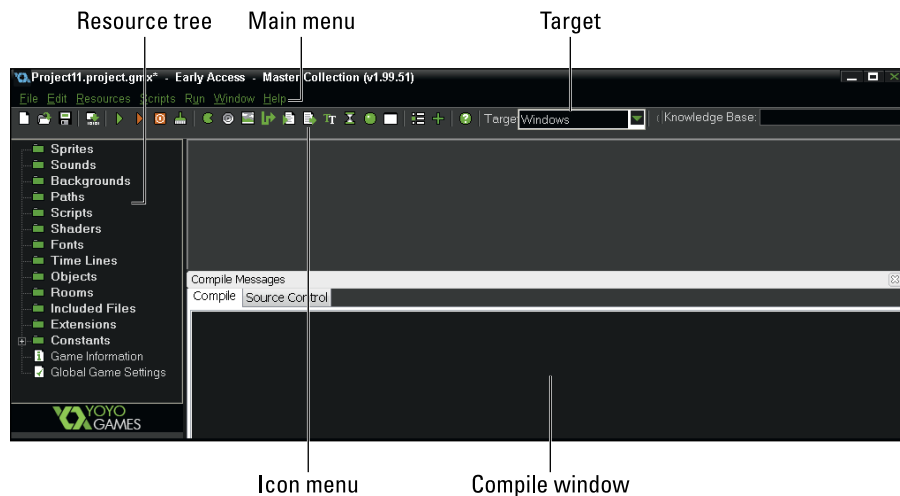


Figure 2-1: The GameMaker interface.

Pump Up the Volume: Adding Sounds to Your Game

Sounds make up a large part of games these days. Everything from moody background music that builds as the player anticipates a challenging boss battle, to explosions, bullets firing, a character jumping (sproing!), and more.

To add a Sound resource to GameMaker, follow these steps:

1. **Choose the icon that looks like an old-fashioned stereo speaker (see Figure 2-2).**

The Sound Properties window opens.

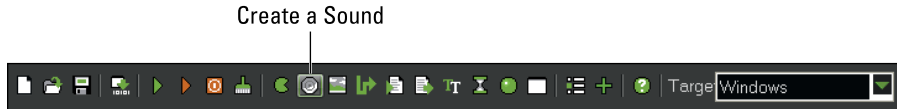


Figure 2-2: Click this icon to create a new Sound.

2. **To load a sound, click the Load Sound button (see Figure 2-3).**

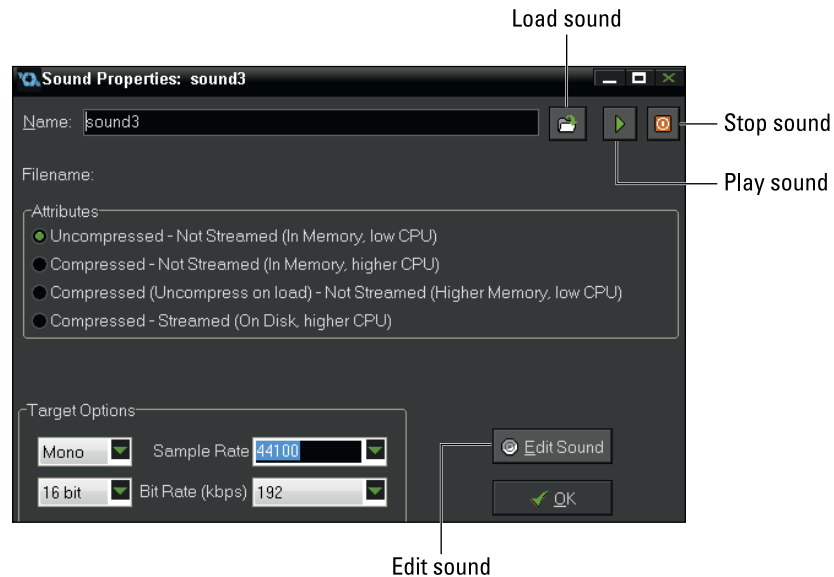


Figure 2-3: The Sound Properties window.

3. **Navigate to a sound file that you previously saved on your PC.**
4. **To play and stop the Sound, click the Play and Stop buttons (refer to Figure 2-3).**

You hear the Sound when you click Play and it continues to play until you click the Stop button.

5. You can edit the Sound by clicking the Edit Sound button (refer to Figure 2-3).

You need to configure an external sound editor to work with GameMaker in order to edit a Sound. Read on for instructions on how to do this.

6. Click OK to save and close the Sound Properties window.

You can configure external editors to work within GameMaker. This includes sound editors, image editors, and code editors. To configure an external editor to work in GameMaker, follow these steps:

1. From the main menu, choose File→Preferences.

The Preferences window appears.

2. Click the Editors tab (shown in Figure 2-4).

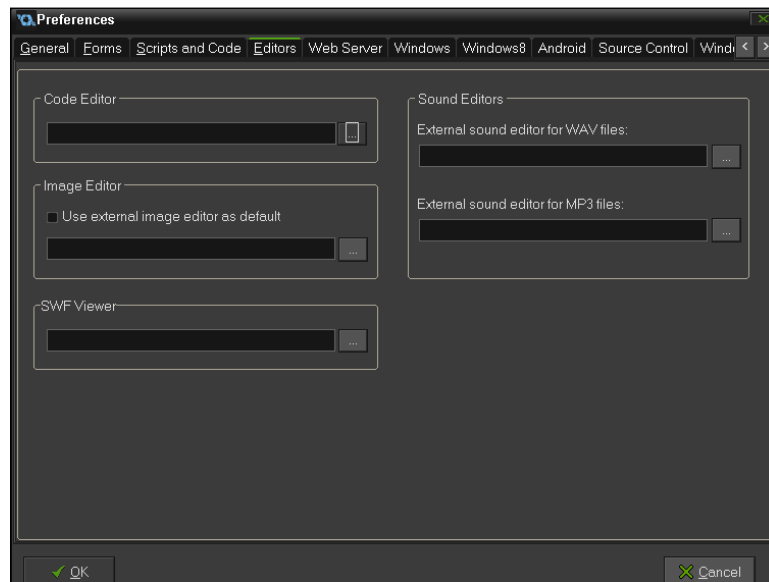


Figure 2-4: The Editors tab of the Preferences dialog box.

3. Click the Open (...) button for the type of editor that you want to configure to work with GameMaker.

Your options are Code Editor, Image Editor, SWF Viewer, and Sound Editors (for WAV files or MP3 files).

A navigation window opens from which you can navigate to and select the editor you want on your PC.

4. Click OK to save and close the Preferences window.

Making Things Sparkle with Sprites

A Sprite is an image, or a series of sub-images, that you assign to Objects. You can use Sprites to represent Objects in the game, such as the player (a ship or a character), explosions, walls, score displays, or anything you want in your game.

When a Sprite consists of a series of images, those images are known as *sub-images*, and they play continuously, as if they were mini-movies. That can give the appearance of a plane's propellers rotating, an animated explosion, and so forth. Each sub-image within a Sprite is named consecutively (image 0, image 1, image 2, and so on), as shown in Figure 2-5.

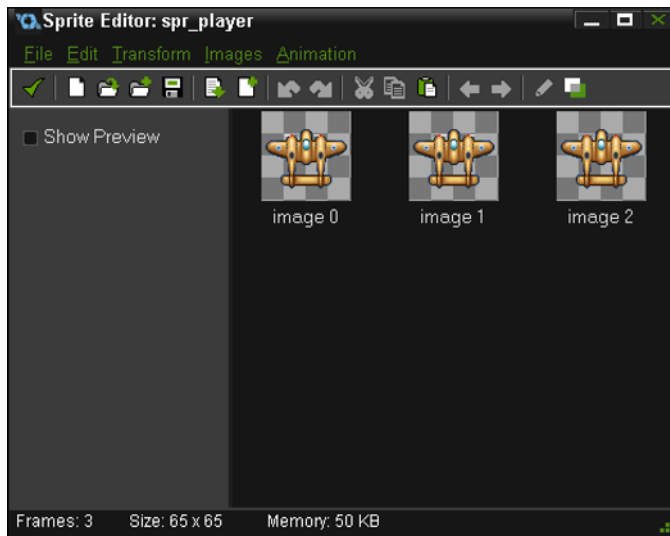


Figure 2-5: The Sprite Editor showing sub-images.



Just as in programming, which starts counting at 0, the sub-image numbering starts at 0 as well.



If you don't want the Sprite to play continuously, you can tell GameMaker to display a specific sub-image of your choice. You can even tell GameMaker to choose a sub-image at random (see Chapter 11).

The remainder of this section covers

- ✓ **Loading and creating Sprites:** You can do this a few ways, including using the main menu, the icon menu, or the Resource tree.
- ✓ **Editing Sprites:** You can do everything from adding the final detail to an already loaded Sprite to creating a new Sprite from scratch.

- ✓ **Configuring Sprites:** Sprites have their own properties, which you can set to affect their behavior during a game.

Loading and creating Sprites

To load a Sprite that already exists, follow these steps:

1. Click the icon that looks like Pac-Man from the icon menu (see Figure 2-6).

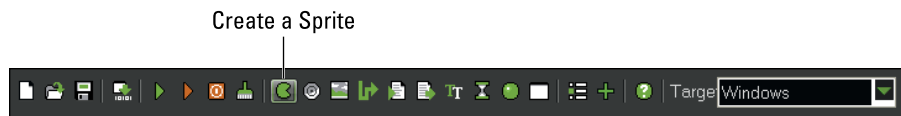


Figure 2-6: Click this icon to create a new Sprite.

The Sprite Properties window appears (see Figure 2-7).

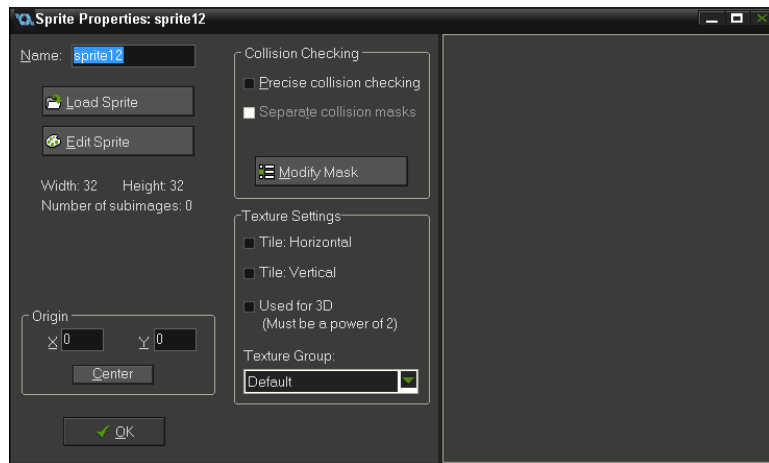


Figure 2-7: The Sprite Properties window.

2. Click the Load Sprite button (refer to Figure 2-7).

A navigation menu opens from which you can find an image on your computer.

To create a new Sprite from scratch, follow these steps:

1. Click the icon that looks like Pac-Man from the icon menu (refer to Figure 2-6).

The Sprite Properties window appears (refer to Figure 2-7).

2. Click the Edit Sprite button in the Sprite Properties window (refer to Figure 2-7).

The Sprite Editor window appears (see Figure 2-8).

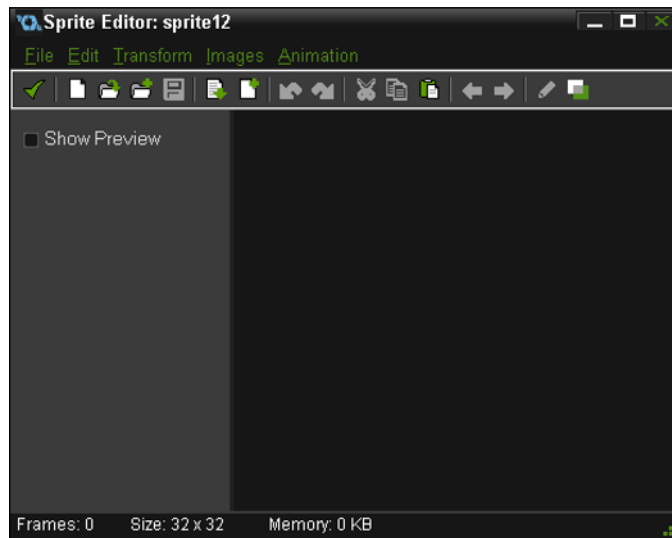


Figure 2-8: The Sprite Editor window.

3. Choose File→New.

The Create a New Sprite dialog box appears (see Figure 2-9).



Figure 2-9: The Create a New Sprite dialog box.

4. In the Width and Height fields, type in the dimensions you want the Sprite to be.
5. Click OK.

A new Sprite, which consists solely of a transparent background at this point, appears in the Sprite Editor. You can now double-click this Sprite to open the image editor from which you can actually draw the Sprite.

To create a sub-image, you can choose one of the Create an Empty Image icons in the Sprite Editor (see Figure 2-10) to add the sub-image either before the current one or at the end. A new, blank sub-image appears in the Sprite Editor.

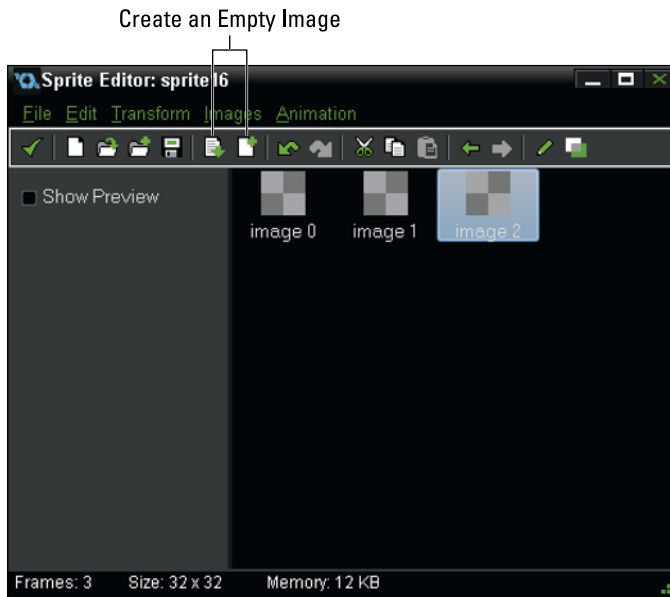


Figure 2-10: The Sprite Editor with three sub-images.

Editing Sprites

GameMaker has a built-in Image Editor (see Figure 2-11) that you can use to edit Sprites. To access the Image Editor, click Edit Sprite from the Sprite Properties window. Then double-click the image that you want to edit.

I don't cover the Image Editor in this book, but the in-software manual goes into detail on it. Just press F1 in GameMaker to access the manual. Then choose the Index, search for "image editor," and then choose Display at the bottom of the window.

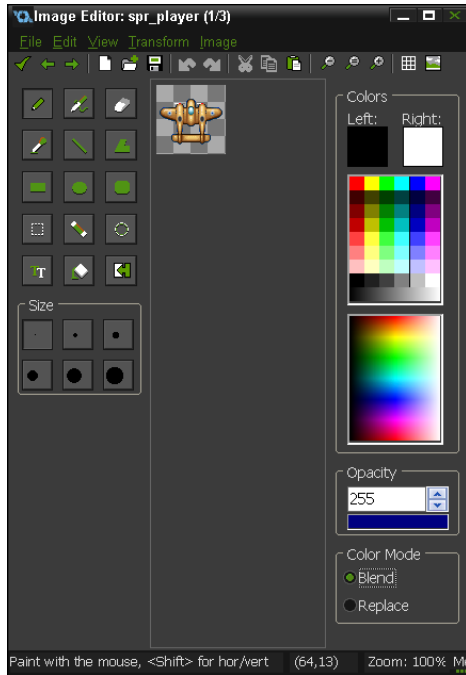


Figure 2-11: GameMaker's Image Editor for Sprites.

Configuring Sprites

When you have your Sprite looking the way you want it, you can configure the Sprite with different properties. This includes placing the Origin, setting the Collision Checking (which includes modifying the Mask), and using the Texture Settings. I go into detail on each of these configurations in the following sections.

Using the Origin

The default *anchor point*, which is considered the Origin, is the top-left corner of a Sprite. GameMaker uses the Origin as a point of reference for rotating, scaling, and so on (for example, turning the image around when it bounces off a wall).

While in the Sprite Properties window, when you click Center (see Figure 2-12), you're telling GameMaker to use the center of the Sprite as the anchor point (that is, the Origin). Chapter 6 provides a good example of working with the anchor point, in the section on creating the boss Sprite.

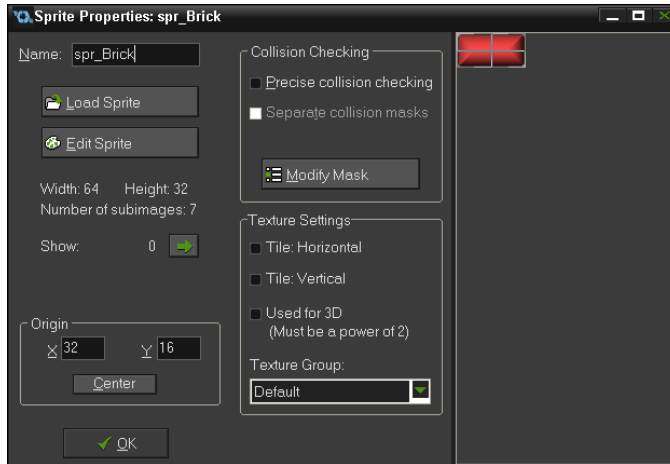


Figure 2-12: Click Center to set the anchor point of the Sprite in the center.

Using Collision Checking

The *Collision Checking* settings are found in the middle of the Sprite Properties window (see Figure 2-13). Collision Checking refers to how the Object (an Object with the Sprite loaded) reacts when colliding with other Objects. You can select the Precise Collision Checking box (refer to Figure 2-13) for more accurate collision detection, but this could potentially slow down the game. You can also modify the Mask from the Sprite Properties window, as discussed next.

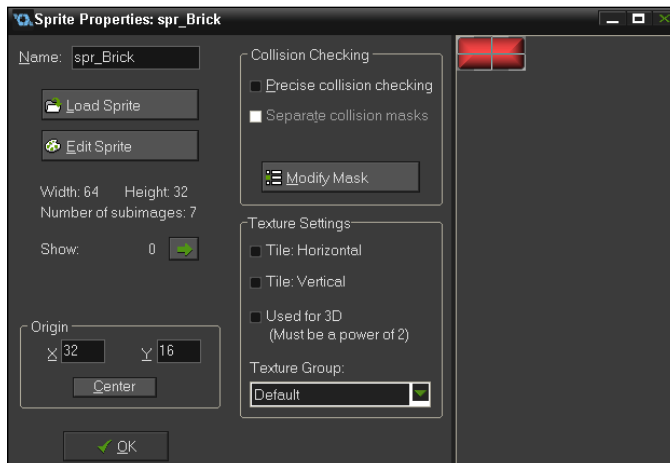


Figure 2-13: The Collision Checking settings.



If your Sprite has sub-images, you can assign a separate Mask to each sub-image. To toggle this option, select the Separate Collision Masks in the Sprite Properties window (refer to Figure 2-13).

To set the Mask, follow these steps:

1. In the Sprite Properties window, click the Modify Mask button (refer to Figure 2-13).

The Mask Properties window appears (see Figure 2-14).

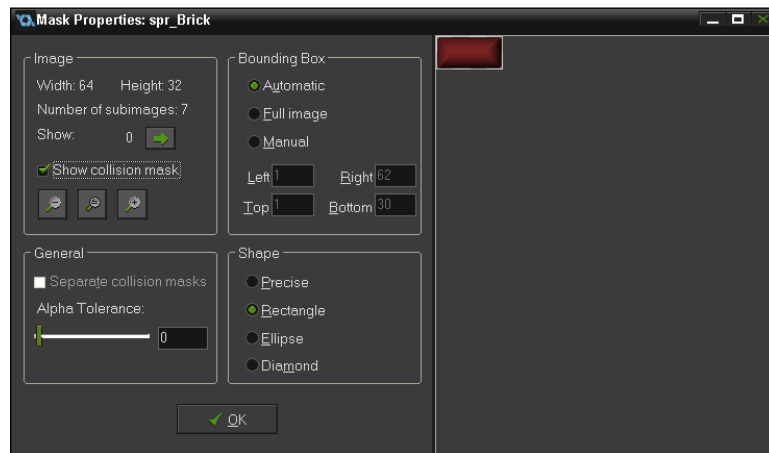


Figure 2-14: The Mask Properties window.

2. In the Image section, select the Show Collision Mask check box (refer to Figure 2-14) if you want the collision Mask to show in the editor; deselect the Show Collision Mask check box if you don't want it to show.
3. In the General section, select the Separate Collision Masks check box to use a separate collision mask for each sub-image.

If you select the Separate Collision Masks check box, you can set the Alpha Tolerance. The higher you set the tolerance, the more the partially transparent pixels are not affected by the Mask. Pixels become transparent as the Sprite transitions between sub-images. It's during these transitions that the Alpha Tolerance comes into play.



By turning on Separate Collision Masks, you significantly increase the workload for GameMaker to run your game, which could cause performance issues.

4. In the Bounding Box section, select Automatic, Full Image, or Manual.

If you select Automatic, GameMaker does its best job of forming a Mask around the Sprite; if you select Full Image, GameMaker selects the entire image, including the transparent background; and if you select Manual, you can configure the settings yourself by typing in values in the Left, Right, Top, and Bottom fields.

5. In the Shape section, set a more precise Mask shape by choosing Precise, Rectangle, Ellipse, or Diamond.

If you select Precise, GameMaker does its best to set the Mask exactly against the pixels. If you choose Rectangle, Ellipse, or Diamond, GameMaker creates a Mask based on the chosen shape.

When you select Precise, you significantly increase the workload for GameMaker to run your game (even more so than using a separate collision mask), which could cause performance issues.



6. Click OK to save and close the Mask Properties window.

In Chapter 8, you can find information on how to use fixtures instead of collision Masks, for when you want to make a physics-based game.

Using Texture Settings

The Texture Settings, and in particular the Texture Group, can be useful when you start making very large games and you need to keep track of many different resources. Essentially, GameMaker enables you to save Sprites, Backgrounds, and Fonts to different groups for your reference later.

To create a Texture Group, follow these steps:

1. From the main menu, choose Resources ⇨ Change Global Game Settings.

The Global Game Settings window opens.

2. Click the Texture Groups tab, as shown in Figure 2-15.

3. Click the Add button near the bottom of the window (refer to Figure 2-15).

A new Texture Group appears in the Texture Groups section.

4. To rename the new Texture Group, select it, and then click the Rename button (near the Add button).

The Rename Texture Page dialog box appears (see Figure 2-16).

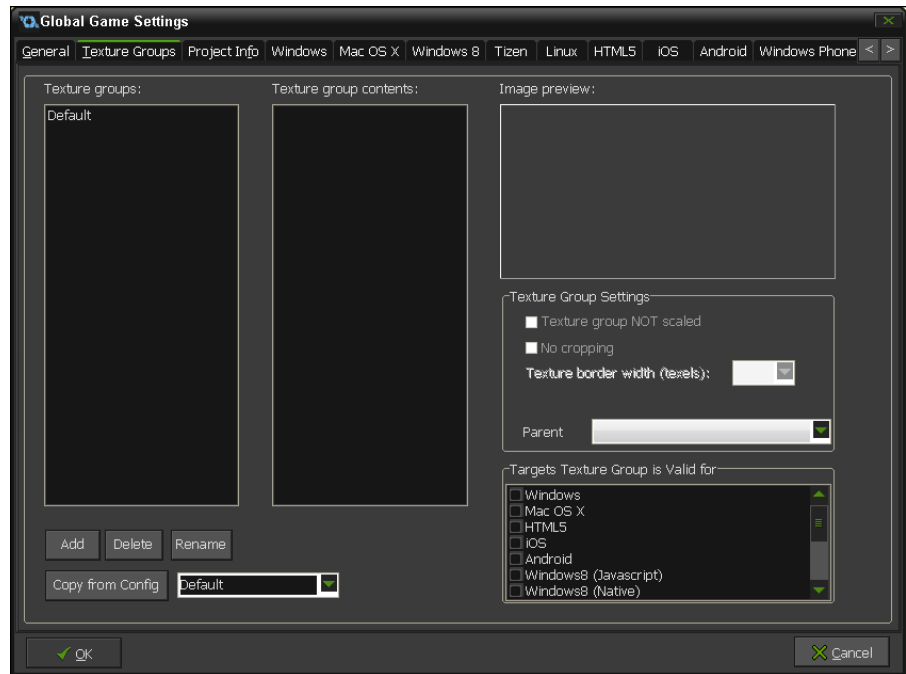


Figure 2-15: The Texture Groups tab of the Global Game Settings window.

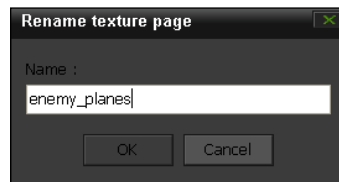


Figure 2-16: The Rename Texture Page dialog box.

5. Enter a new name, and click OK.

The renamed Texture Page appears in the Texture Groups section.

6. Click OK to save and close the Global Game Settings window.

Now that you have more than one Texture Group, when you're in the Sprites Properties, you can assign that Sprite to the new Texture Group. From the Texture Group drop-down list, select the group you want (see Figure 2-17). Texture Groups do help optimize game performance.

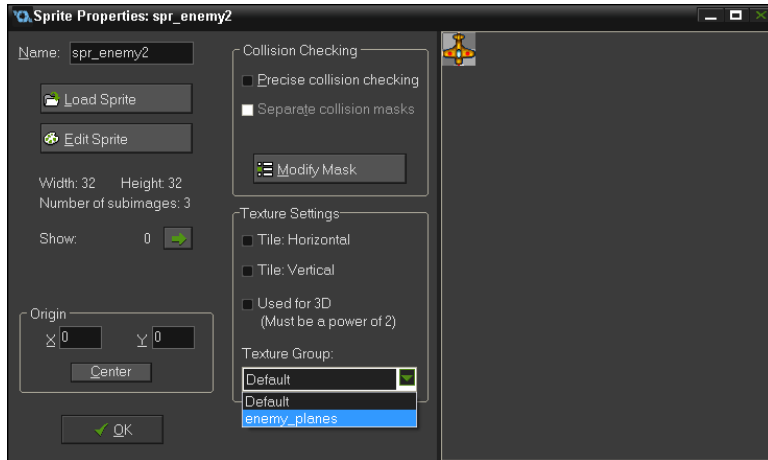


Figure 2-17: Choose the Texture Group you want from the drop-down list.

Populating Your Game with Objects

In this section, I provide a quick overview of *Objects*, which are what make up the different parts of the game, such as the player, the enemy, the score display, and so on. Objects can also control the game through the different Events and Actions that you add to them.

To create an Object, follow these steps:

1. Choose the green ball icon from the icon menu (see Figure 2-18).



Figure 2-18: Choose the green ball to create a new Object.

The Object Properties window appears (see Figure 2-19).

2. In the Name field, type a name for the Object, such as `obj_plane`.
3. To assign a Sprite, click the drop-down icon from the Sprite section (see Figure 2-19).

A list of available Sprites appears for you to choose from. Sprites provide a visual representation of the Object — you know, so that the player has a cool-looking ship to shoot the bad guys (and the bad guys have fearsome-looking ships).

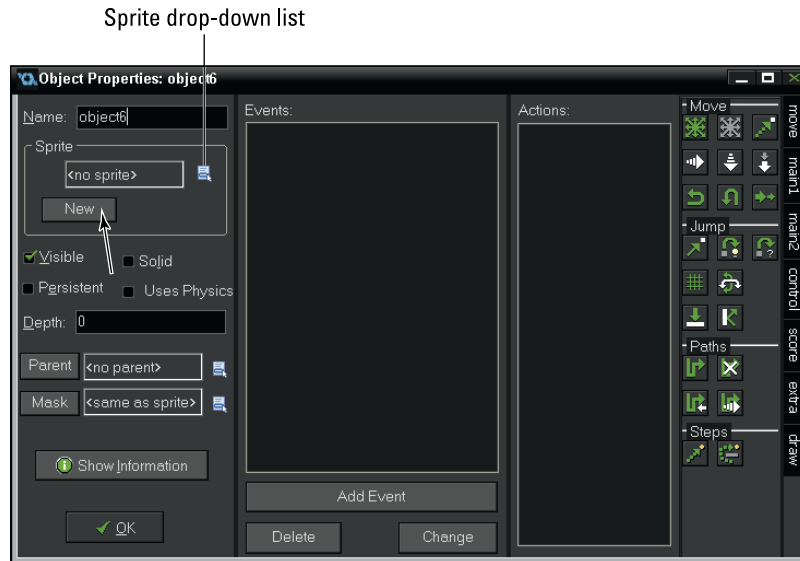


Figure 2-19: Click the drop-down icon to open a list of available Sprites.

4. Click the New button from the Sprite section to open the Sprite Properties window.

You can load a Sprite or create a new one through the Editor (see “Making Things Sparkle with Sprites,” earlier in this chapter, for more information).

5. Select the Visible check box if you want the Object to be visible in the game.

6. Select the Solid check box so that GameMaker knows this is an Object that will likely collide with other Objects.

For example, say you have two balls in a Room and they collide. You would expect them to bounce off each other as if they were balls in real life. But if one of the balls is not Solid, then the one ball might appear to roll right through the other as if it were made of air.

7. Select the Persistent check box for the Object to carry over from Room to Room.

The Object is not destroyed even if the player leaves one Room and enters another. The only way to destroy a Persistent Object is to do so explicitly through Events and Actions.

8. Select the Uses Physics check box to create a physics world.

See Chapter 12 for more on using physics.

9. In the **Depth** field, enter a numerical value (such as 10,000) to determine which layer of the Room you want the Object to reside at.

The Depth field enables you to place Objects over other Objects.

10. To assign a parent to the Object, click the drop-down icon (see Figure 2-20), and select an Object.

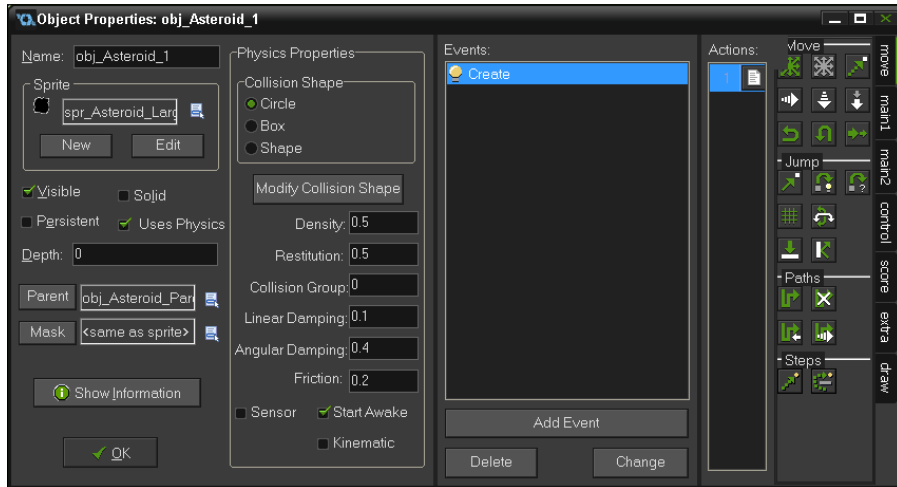


Figure 2-20: Assign a parent Object to inherit the Events and Actions.

When you make a child Object, that Object inherits the properties of the parent Object.

11. To assign a Mask to the Object, click the drop-down icon and select the Sprite with the Mask you want to use.

If you don't do this, GameMaker uses the Mask of the Sprite you assigned to the Object.

12. Click the **Add Event** button to select an Event to add to the Events section.

See Chapter 3 for details on Events.

13. Drag and drop Actions from the tab on the right to the Actions section.

See Chapter 4 for details on Actions.

When you finish creating your Objects, you're ready to place the Objects in the Room, so that you can create Instances of those Objects within the game. You may have a single Object that represents a bad guy, but you can create several Instances of that one Object within the game; in that way, you can create many of the same bad guy to attack the player. I go into more detail on Rooms at the end of this chapter.



If you forgot to load a Sprite before you started making your Object, you can click New from the Sprite section of the Object Properties window to open the Sprite Properties window. Then you can proceed to load a Sprite or create a new one through the Editor (refer to Figure 2-11).

Placing Objects in the Game Room

If you think the magic only happens in the, um, kitchen, then does GameMaker have a Room for you! GameMaker has a feature called *Rooms*. A Room is where you place all the different parts of your game. After you finish creating and refining your resources — Sounds, Sprites, Objects (configured with Actions and Events) — you're ready to place Instances of your Objects in the Room. But first you have to create one.

Creating a Room

To create a new Room, click the Create a Room icon (which looks like a white rectangle; see Figure 2-21). The Room Properties window opens (see Figure 2-22).

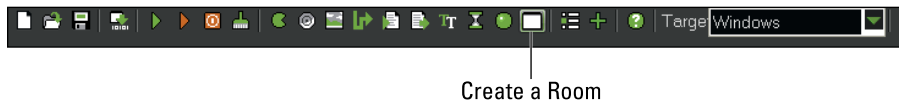


Figure 2-21: The Create a Room icon.

The Room Properties window has several tabs of features:

- ✓ **Backgrounds:** Here, you can set the background color, or choose a background image that you can tile to fill the Room.
- ✓ **Views:** Chapter 5 is all about Views. There, you can find an example on how to use Views in the Scrolling Shooter game.
- ✓ **Physics:** Chapter 12 is a great place to get your feet wet using Physics.
- ✓ **Objects:** Here, you can select Objects in order to place Instances of those Objects in the Room (see the following section).
- ✓ **Settings:** Here, you can name the Room, set the size of the Room, and set the speed of the Room. The default speed is 30 steps, which means there are 30 steps per second. Knowing the speed of the Room is useful when it comes to setting Alarms and so forth.
- ✓ **Tiles:** Here, you can select and manually place different images to fill the background. This is different from the Backgrounds tab in that the Backgrounds tab can automatically fill the Room for you, while the Tiles tab is manual.

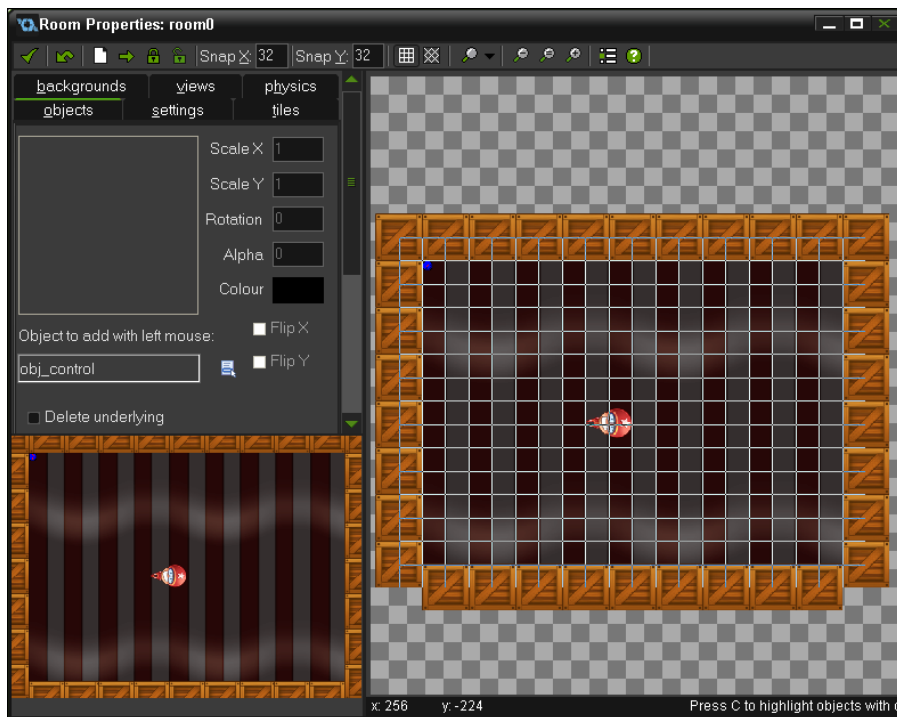


Figure 2-22: The Room Properties window.



Putting your Objects in the Room

When you place an Object in the Room, it becomes an Instance of that Object. You can have several Instances of a single Object in your game.

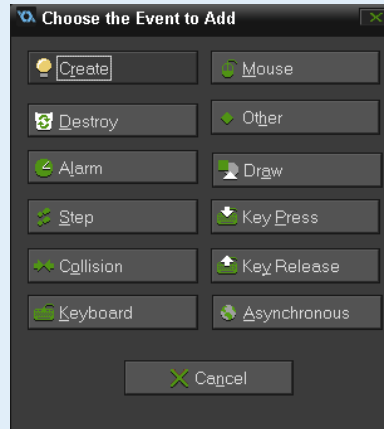
To place an Object in a Room, follow these steps:

1. **From the Resource tree, double-click a Room.**
The Room Properties window appears (refer to Figure 2-22).
2. **Click the Objects tab.**
3. **Click the drop-down icon indicated in Figure 2-22 to select the Object you want to place an Instance of in the Room (for example, obj_wall).**
The name of the Object appears in the field, and you see that Object's assigned Sprite in the top corner of the window.
4. **Click in the Room where you want to place an Instance of the Object.**

The Sprite for the Object appears where you click. If you're placing an Object without an assigned Sprite, you see a little blue circle with a question mark in it.

Part II

Basic Tools and Techniques



Find out more about GameMaker: Studio features for developers and non-developers alike in a free article at www.dummies.com/extras/gamemakerstudio.

In this part . . .

- ✓ Create some action with Events.
- ✓ Define those Events with Actions.
- ✓ Make games with Views.
- ✓ Use a Time Line to create your game.

Creating Events

In This Chapter

- ▶ Creating and destroying Objects
- ▶ Setting Alarms
- ▶ Using Steps
- ▶ Making Instances collide
- ▶ Using keyboard and mouse controls
- ▶ Drawing to the screen
- ▶ Enhancing your game with other Events

In this chapter, I cover each of the Events you can use within GameMaker: Studio. If you think of Objects, Events, and Actions in terms of English grammar, then the Objects are the subjects, the Events are the object (so to speak, just work with me here), and the Actions are the verbs. In that sense, the Objects are the things in the game (such as the player's plane), the Events are what is acted upon (such as creating the plane), and the Actions make the magic happen (such as making the plane fly).

How to Add an Event to an Object

There are 12 different Events that you can use in GameMaker, as shown in Figure 3-1.

To add an Event to a new Object, follow these steps:

- 1. From the main menu, choose Resources → Create Object.**

The Object Properties window appears.

- 2. Click the Add Event button, shown in Figure 3-2.**

The Events menu appears.

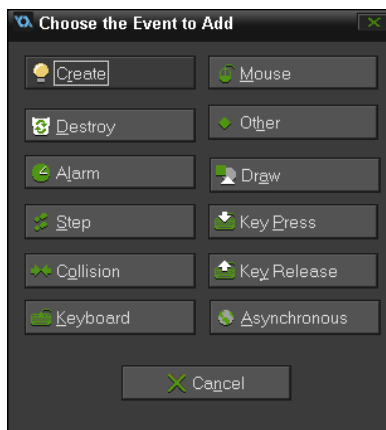


Figure 3-1: GameMaker Events.

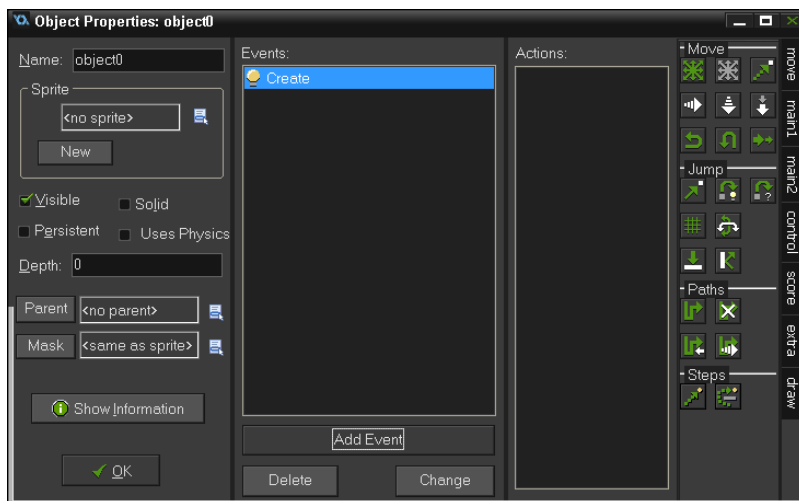


Figure 3-2: To open the Events menu, click Add Event.

3. Select the Event that you want to use (that's what the remainder of this chapter is about).

The name of the Event you chose appears in the Events section of the Object Properties window.

By the way, after you add an Event to the Object, you need to assign Actions to that Event. Chapter 4 is all about Actions.

The Create Event

The Create Event triggers an assigned Action that you want to happen at the start of the game or when the player enters a new Room. The Create Event is best used for initializing variables, starting Time Lines, setting paths, and so on. These are the types of Actions that you want to happen at the start of the game, or when an Instance first appears in the game, and you only need them done once.

You can also use the Create Event to provide an Object with an initial speed for when the Instance appears in the Room. This is useful if you make a game in which a plane flies continuously upward.

Another use for the Create Event is for when a bullet is fired. In that case, when the player presses the spacebar to shoot, that triggers a Create Instance Action (this is an Action, not an Event), which triggers the creation of the bullet Object, then within that bullet Object, is a Create Event. That Create Event is triggered when an Instance of that Object appears in the Room (when the player shoots a bullet) and the speed, sound, and movement Actions are triggered. Figure 3-3 shows the Create Event with the Actions for the bullet Object. Remember, all of these Actions happen when the Instance first appears in the Room.

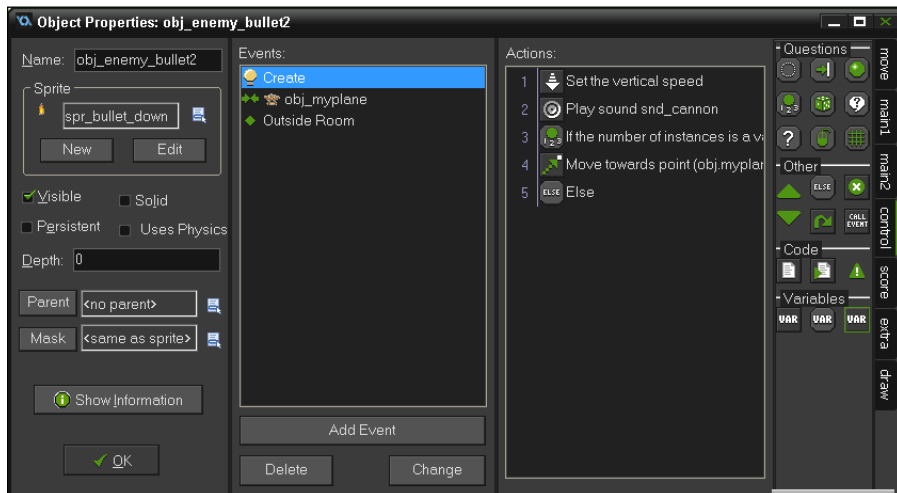


Figure 3-3: Use the Create Event to cause Actions to happen when the Object appears in the Room.

The Destroy Event

Use the Destroy Event when you no longer want an Instance in the Room — say, when an enemy gets blown up. You can use the Destroy Event to trigger an explosion, increase the player's score, or re-create a new Instance of the Object (or all three). Figure 3-4 shows an Object with a Destroy Event with a Set Score Action. This obviously isn't a fully configured enemy Object — you would want the Destroy Event to be the last, and certainly not the only, Event.

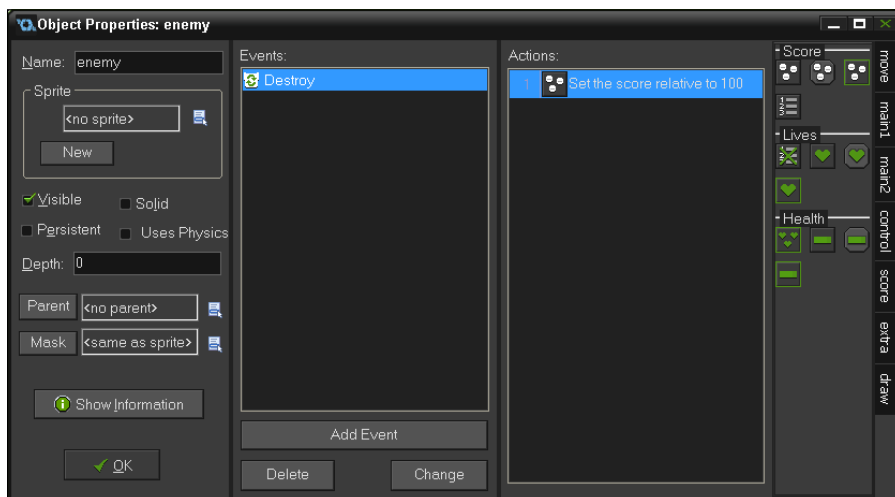


Figure 3-4: Use the Destroy Event to increase the player's score.

The Alarm Event

You can add up to 12 Alarms for each Instance in the Room. Alarms are excellent for when you want to time Actions. For example, if you want the player to only shoot bullets once every two seconds, you can set an Alarm for that. If you want to time when new enemy planes appear, you can set an Alarm for that as well.

To get an Alarm to work, you must first set an Alarm Action to trigger the Alarm Event after a countdown that you set. In that way, Alarms enable you to time when events happen in your game.



Timelines are another great way to time events in your game. To see how to use Timelines, see Chapter 6.

When you choose the Alarm Event from the Event menu, a new submenu opens. From that submenu, you can choose the Alarm you want to configure and add Actions to (see Figure 3-5).

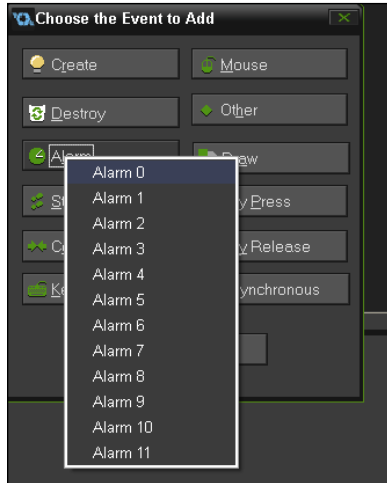


Figure 3-5: You can have up to 12 Alarms for each Instance.



An Instance is an Object that you've placed in the Room. You can place more than one Instance of an Object in a Room, so you can have multiple Instances of the same Object in your game.

The Step Event

Step Events are great for when you want something to continuously happen, or you want something continuously checked for each *step* of the game. GameMaker tells time by steps, which is defined by the Room Speed (see Chapter 2), which sets how many steps are performed each second.

An example of using a Step Event is to determine if certain Instances have left the Room, and if so, to reappear at the top of the Room. As GameMaker needs to continuously check to see if the Instance left the Room, a Step Event comes in very handy to use.

When you click Step from the Event menu, a submenu appears giving you three options, as shown in Figure 3-6:



- ✓ **Step:** This will probably work best for you 90 percent of the time.
- ✓ **Begin Step:** This option is useful when you absolutely need the Step Event to be the first Event checked, no matter what, every time.

GameMaker updates all built-in variables between the Step and End Step. So, if you want something to happen after the variables have been updated, then choose End Step. If you want something to happen before the variables are updated, then choose Begin Step.

- ✓ **End Step:** This option is just like Begin Step, except it's for when you absolutely need this Step Event to be checked last, every single time.

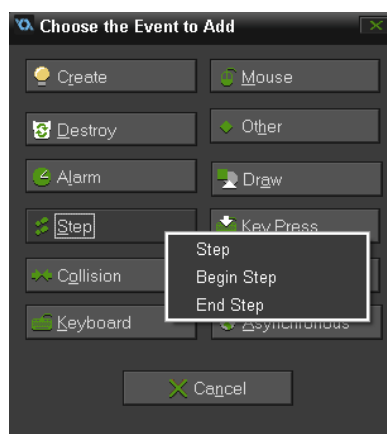


Figure 3-6: Your Step Event options.

The Collision Event

If your game has Instances flying every which way and collisions are inevitable, then you may be using a lot of Collision Events. These Events are great for when you want something to happen when two Instances collide — such as when a bullet Instance hits a plane Instance or, more simply, when a player Instance hits a wall Instance and you want the player to bounce off the wall (see Chapter 2).

Masks are important to Collision Events because this is how GameMaker determines where the actual collision takes place. (See Chapter 2.)

The Mouse Event

Mouse Events can be as simple as assigning the left button to fire, but there is a large submenu of Events that you can choose from (see Figure 3-7). Figure 3-7 also shows the tertiary menu for Global mouse. In this way, GameMaker provides you great control over how you want the mouse to behave in your game. After you choose which Mouse Event you want to use, you can add Actions to that event or even your own code.

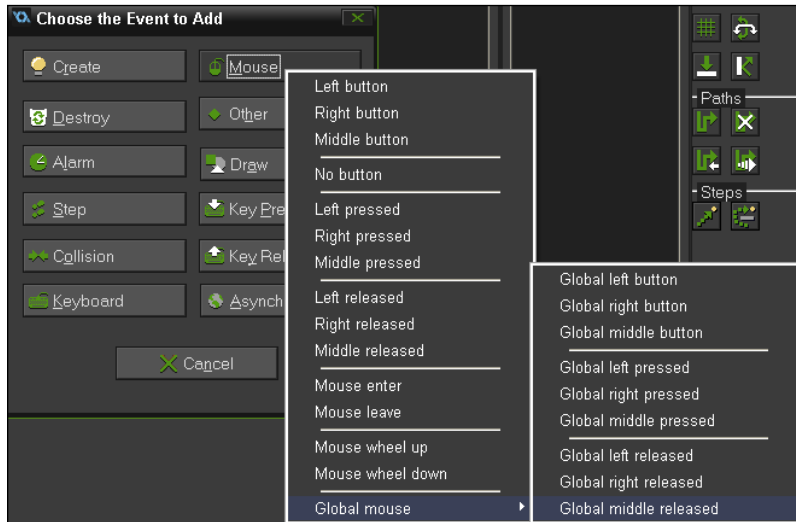


Figure 3-7: The many options for the Mouse Event.

The Other Event

Ah, the Others. Those not deemed worthy of their own place on the Event menu. I kid. These Other Events can be kinda awesome. The menu items (shown in Figure 3-8) are pretty much self-explanatory, so I won't go into detail on them. For full descriptions of each of these items, you can always press F1 within the GameMaker and search for Events.

The Other Events are

- ✓ **Outside Room:** The Event is triggered when the Instance leaves the Room.
- ✓ **Intersect Boundary:** The Event is triggered when the Instance touches the inside edge of the Room.

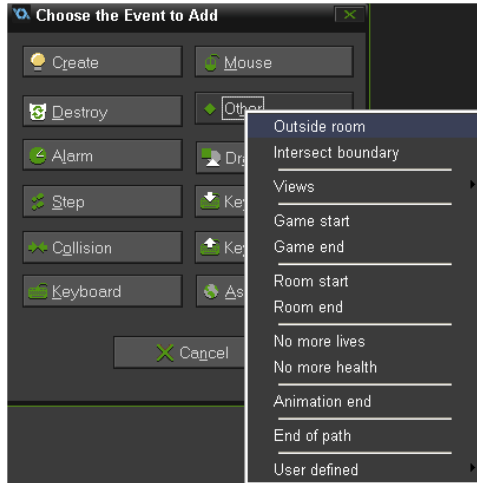


Figure 3-8: The Other Events you can use.

- ✓ **Views:** This option has a submenu from which you can choose Outside View or Intersect Boundary. The Views here correlate with the Views in the Room. (For more on Views, see Chapter 5.)
- ✓ **Game Start:** This Event is triggered only once during the game. The Game Start Event is triggered after the Create Event of all other Instances. You can use this Event in controller Objects to start music and initialize global variables and other Actions that you want to happen at the start of the game.
- ✓ **Game End:** This Event is also triggered only once during the game. It really works only with Windows, Ubuntu (Linux), and Mac OS — it doesn't work well on all mobile devices.
- ✓ **Room Start and Room End:** These Events are triggered when a Room starts or ends.
- ✓ **No More Lives:** GameMaker uses global variables, and Lives is one of them. This Event is triggered when the player runs out of lives and is typically set up to restart a Room or end the game.
- ✓ **No More Health:** Health is another global variable. You can use this Event to decrease the number of player's lives when the player's health is depleted.
- ✓ **Animation End:** This Event works with Sprites and sub-images (see Chapter 2). Basically, this Event is triggered when all the sub-images of a Sprite are finished playing. A good example for an Action for an Animation End Event would be for an explosion Sprite to destroy itself when it's done playing.

- ✓ **End of Path:** If you use Paths in your game, this Event could be useful for when an Instance reaches the end of the Path.
- ✓ **User Defined Event:** This Event is triggered by code that you write through an Execute Code Action. This Event also works with all the Actions.

The Draw Event



Use the Draw Event when you want to display something during the game. The Draw Event does have its own submenu, as shown in Figure 3-9.

If you use a Draw Event, GameMaker no longer defaults to the assigned Sprite, and instead you have to tell GameMaker what to draw.

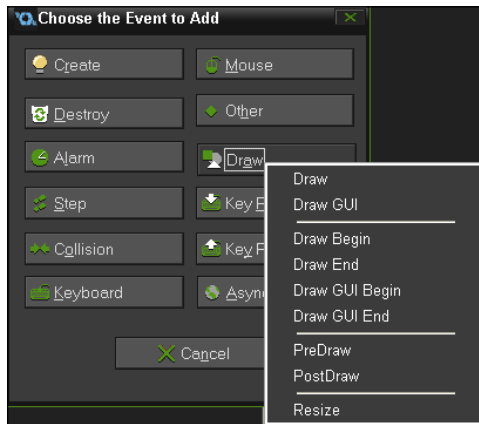


Figure 3-9: The Draw Event submenu.

The different options from the submenu include

- ✓ **Draw:** The standard Draw Event should be sufficient for most of your needs. The Draw Event enables you to place code or Actions for an Object, and those Actions and code override the default draw, which would be the assigned Sprite. This is useful, for example, if you want to replace the assigned Sprite with text to display a message. This Event is called once per View, while the other Draw Events are drawn each step of the game.

All Objects with an assigned Sprite with Visible toggled on (from the Object Properties window, as shown in Figure 3-10) automatically trigger a default Draw Event every step of the game. GameMaker does this for you, which is nice, because you don't have to assign a Draw Event for each Object with an assigned Sprite to have it appear in the game.



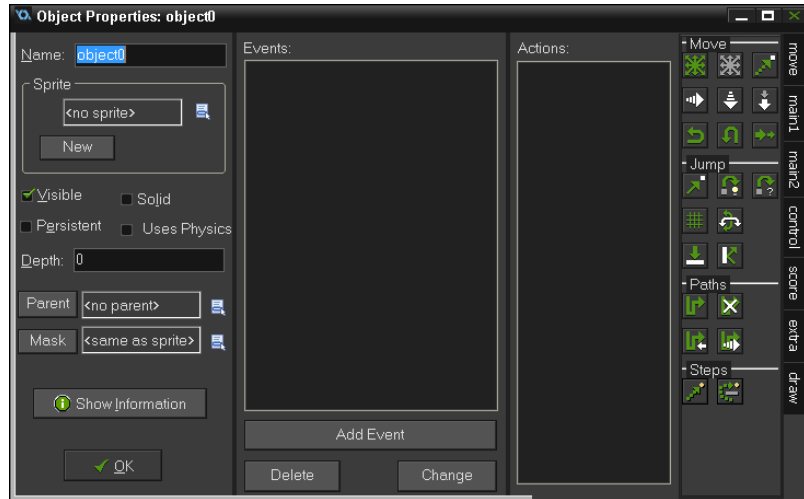


Figure 3-10: Select the Visible check box to automatically trigger a default Draw Event.

- ✓ **Draw GUI, Draw GUI Begin, and Draw GUI End:** GUI is an acronym for *graphical user interface*. The GUI is not affected by the View scale or rotation. Instead, the GUI is drawn on a separate layer than, say, the background. A Draw GUI Event is great for creating a heads-up display (HUD), which can show a player's health, ammo, weapon type, a mini map, and so on. As with the other Begin and End Events, the Draw GUI Begin and Draw GUI End Events determine the order of what's drawn.

- ✓ **Draw Begin and Draw End:** These two Events do exactly as they say: They make sure that something is drawn in the game either before or after the standard Draw Event. Using these Events is a great way to make sure items are drawn in the order you want them to appear.

The Draw Begin, Draw, and Draw End Events all run before the Draw GUI Events and between the PreDraw and PostDraw Events. This has significant meaning in that all these Events are drawn beneath the Draw GUI Event, no matter what the Depth (see Chapter 2) is set at.

- ✓ **PreDraw and PostDraw:** The PreDraw and PostDraw Events are drawn directly to the *screen buffer*, which is the combined screen space for all Views currently visible, or the window size if you don't use Views. The PreDraw Event is triggered before any other Draw Events, which means you can use it to set values, set draw properties, and, of course, draw things. The PostDraw Event runs after the other Draw Events but before the Draw GUI Event.





You need to deselect the Clear Background with Window Colour check box in the Room editor for PreDraw and PostDraw Events to work. Figure 3-11 shows where to deselect this feature (the default setting has it selected). You also need to make sure that a background color is not drawn; you do this on the Background tab (see Chapter 2 for additional information).

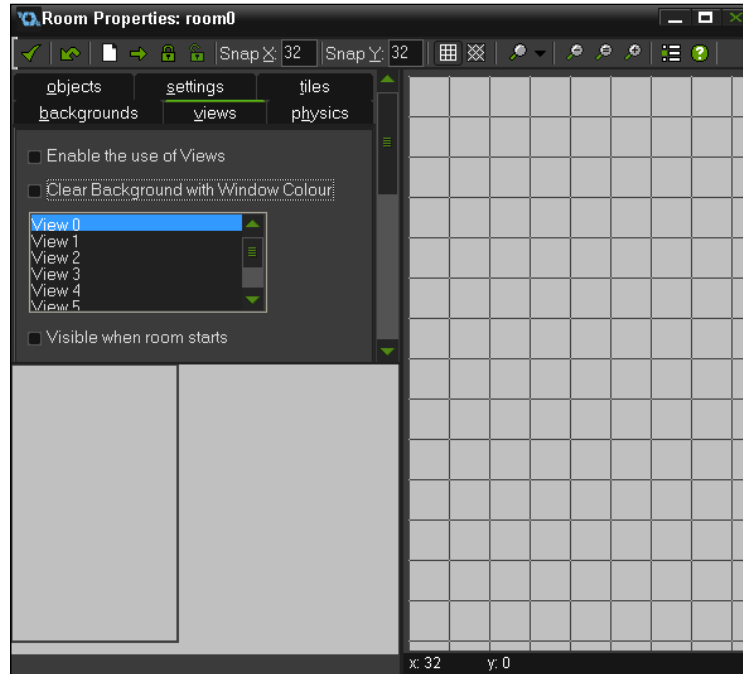


Figure 3-11: Do not select Clear Background with Window Colour for PreDraw and PostDraw Events.



If you turn off the View Clearing and disable drawing a background color, you might experience *trails* (leftover traces of images) during gameplay. These trails happen because now you're drawing directly over the previous frame without it being cleared. You can use code to help solve this potential problem, which is `draw_clear_alpha`. You can place this code in an Execute Code Action within the PreDraw or PostDraw Event.

- ✓ **Resize:** The Resize Event is used specifically for Windows 8 games and doesn't actually draw anything in the game. Instead, the Resize Event is to help snap the game window when the player moves that window to the side of the screen. If you're familiar with working in a Microsoft Windows environment, you're aware that if you're working in a window and you drag the top of the window to the top of your screen, Windows will snap the bottom of the window to the bottom of the screen. This Event is triggered every time the player resizes or moves the game window.

The Key Events

When you click Add Event, you may notice there are three different options for the keyboard: Keyboard, Key Press, and Key Release. Use the Key Press and Key Release Events to trigger an Action once each time an assigned key is pressed or released. The Keyboard Event is good for when the player needs to press the key continuously, such as for moving an Instance across the Room.

Each of the Key Events has a submenu, as shown in Figure 3-12, from which you can assign a key for the Action. Use <Left>, <Right>, and so on for the arrow keys on the keyboard.

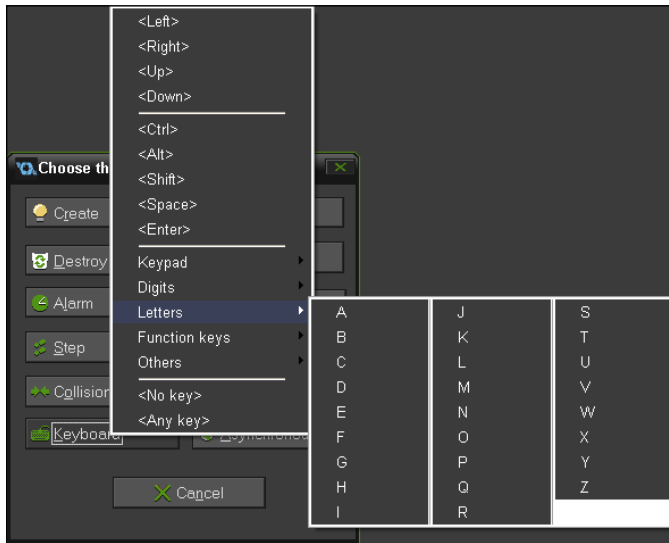


Figure 3-12: The Keyboard has many options to fit your many needs.

The Keypad, Digits, Letters, Function keys, and Others have a tertiary menu. Figure 3-12 shows the third-level submenu for Letters — say, if you want to assign W for moving up, A for moving left, D for moving right, and so on (as opposed to using the arrow keys).



Most of the keys are pretty obvious, except for No Key and Any Key. These Events check for when the player doesn't press a key at all or presses any key on the keyboard.

Also note that the keys for the numeric keypad only work if the player has NumLock toggled.

The Asynchronous Event

The Asynchronous Event is triggered at the end of another Action, such as the loading of a file or a reply from a web server. This Event also has a sub-menu, as shown in Figure 3-13.

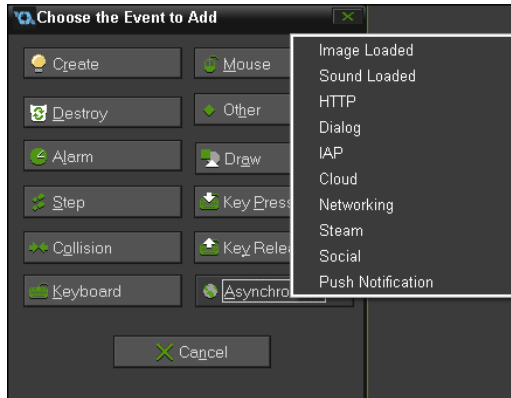


Figure 3-13: The Asynchronous Event triggers after another Action finishes.

Asynchronous Events are triggered by functions that need to receive data from a source at an unknown time in the future. This Event is best used for seasoned game developers who are creating complex online games.

In the following chapter, I go into detail on all the Actions that you can assign to Events.

Adding Actions to Events

In This Chapter

- ▶ Assigning an Action to an Object
- ▶ Giving an Object something to do
- ▶ Making your Object move
- ▶ Keeping score and more

In this chapter, I discuss each of the Actions that you can assign to Objects. You can think of Actions as the verb in a sentence: They tell the Objects what to do, when to do it, and how to do it. Actions are what causes a clown to bounce off the wall, or a plane to shoot a bullet, or an asteroid to split into pieces. By using Actions, you save yourself from having to write the code yourself. I love Actions because I'm not great with writing my own code, and Actions provide an easier and quicker way for me (and you) to create a game.



If you want to write the code yourself, you can do that, but you still need to assign an Execute Code Action to an Event (see Chapter 3) within the Object. Then you add your code to the Execute Code Action.

How to Assign an Action to an Object

Here's a quick, top-level procedure on how to assign an Action to an Object:

- 1. From the main menu, choose Resources→Create Object.**

The Object Properties window appears.

- 2. Choose Add an Event from near the bottom-middle of the Object Properties window (see Chapter 3).**

The Events menu appears.

- 3. Choose an Event, such as Create.**

The Create Event appears in the Create section of the Object Properties window.

4. From the right side of the Object Properties window, drag and drop an Action icon — say, the Move Fixed Action — from the Move tab into the Actions section (see Figure 4-1).

The properties for the Action that you dragged and dropped appear.

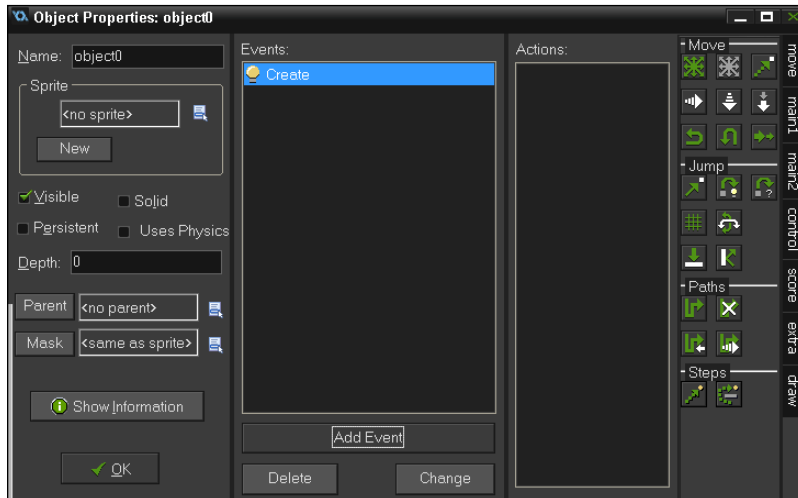


Figure 4-1: Drag and drop Actions from the Move tab to the Actions section.

The remainder of this chapter focuses on the different tabs that contain the Actions and tells you what you can do with each Action.

Move Actions

The Move tab is the first tab in the Object Properties window. It consists of four different sections: Move, Jump, Paths, and Steps (as shown in Figure 4-2).

Move Directional Actions

In the following definitions, I use the term *Instance* to refer to what the Action is affecting. However, you configure the Action in the Object Properties window. Remember that an Instance is simply an Object that you've placed in the Room.





Figure 4-2: The Move tab.

The Move directional arrows apply to the movement, direction, and position of an Instance. They include the following Actions (see Figure 4-3):

- ✓ **Move Fixed:** This action starts an Instance moving in a specified or random direction. Also, it includes the speed you want the Instance to move, which is based on pixels per step.
- ✓ **Move Free:** Use this Action to have the Instance move at a specific angle between 0 and 360 degrees. If you use 0, the Instance moves to the right, a 90-degree angle moves the Instance up, and so on. You can also use random (360) to specify a random movement.

If you select Relative for this Action, any new motion is added to the current motion. Say the Instance is moving up, and then something happens during the game to cause the Instance to move left. If you select Relative, the Instance will start moving up and to the left.

- ✓ **Move Towards:** This action enables you to send an Instance toward another Instance. You can set the speed of the Instance and then use the coordinates of another Instance as the direction for it to move in. An example could be `player.x, player.y` (where `player` is the Instance you want your Instance moving toward). I have a great example of this in Chapter 5, in the section on shooting guided bullets in a Room with a View (but this will work even if you're not using Views).

- ✓ **Speed Horizontal and Speed Vertical:** Use this Action to set the Speed of an Instance.
- ✓ **Set Gravity:** For gravity, you can set the direction you want the gravity to pull the Instance by using the degree of an angle. When you start to experiment, begin with a slow speed of say, 0.01. Gravity is cumulative, in that an Object will pick up speed, such as a boulder would do as it rolled down a mountainside.
- ✓ **Reverse Horizontal and Vertical:** This Action is great for when an Instance collides with another Instance and you want the Instance to reverse direction.
- ✓ **Set Friction:** This Action slows down the Instance. For each step of the game, friction slows the Instance by the amount that you set. You may want to start with a small amount of friction, such as 0.01.

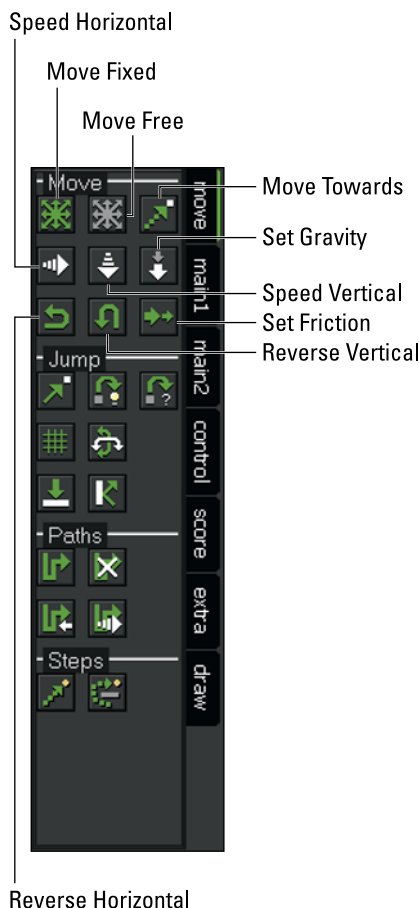


Figure 4-3: The Move Actions from the Move tab.

Jump Actions

The Jump Actions, shown in Figure 4-4, enable you to move Instances around the Room — at one moment, the Instance may be at the bottom of the Room, but in the next moment it's at the top of the Room. Here are your options:

- ✓ **Jump to Position:** GameMaker checks the position of the Instances, such as checking to see if the Object left the Room. If the Object does leave the Room, you could have it reappear at the top of the Room by jumping. Another example is if a player dies, and you want the player to restart in a specific position. You specify the X and Y coordinates to determine where the Instance jumps to.
- ✓ **Jump to Start:** Use this Action to move an Instance back to where it was created.

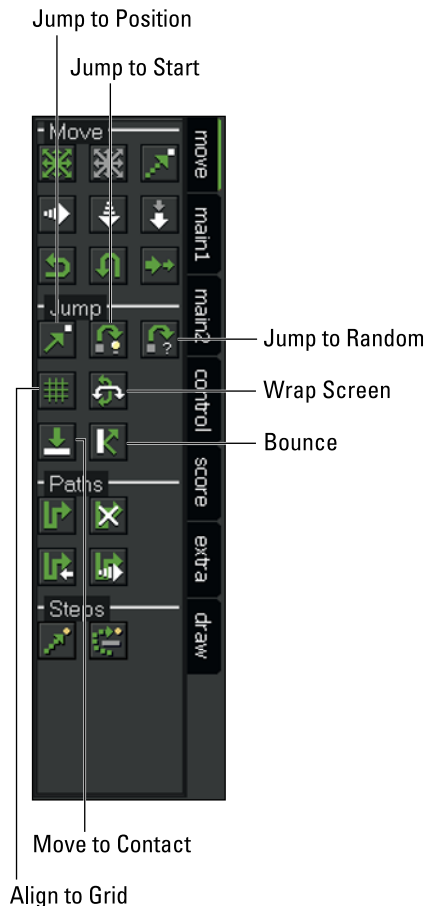


Figure 4-4: The Jump Actions from the Move tab.

- ✓ **Jump to Random:** Use this Action to randomly move the Instance about the Room. Instances won't appear where any other solid Instances are located.
- ✓ **Align to Grid:** This Action enables you to snap an Instance to a position for accurate placement within the game.
- ✓ **Wrap Screen:** Wrapping is when an Instance can leave one side of the screen and reappear on the other. There is an example on how to achieve this with code in Chapter 12, where the player's ship can leave one side of the screen during the game and reappear on the other. This Action is typically used together with the Outside Room Event; when the Instance is outside the Room, wrap it around to the other side. You can choose only horizontal, only vertical, or both.
- ✓ **Move to Contact:** Use this Action when you want an Instance to stop moving when it encounters another Instance or when it moves the maximum amount of distance that you set. You can enter a value of -1 for the Instance to travel for an infinite distance (no maximum). This Action also enables you to set the direction of the Instance. This Action is typically used with a Collision Event.
- ✓ **Bounce:** Use this Action when you want an Instance to naturally bounce off other Instances. If you set Precise to false, you get the natural bounce effect only on straight walls; if you set Precise to true, you get the natural bounce even on slanted walls.

Path Actions

Paths are great when you want to send an Instance to travel in a specific way about the Room. Figure 4-5 shows the Path Properties window and what a Path looks like.

You can start a new Path by clicking the Path icon from the menu (see Figure 4-5) or by right-clicking on Paths from the Resource tree. Paths are represented as a yellow line. The way points along the path are represented by colored squares and dots:

- ✓ The green square represents the starting and end point of the path.
- ✓ The red dot represents the selected way point.
- ✓ The blue dots represent way points.

You can create a Path by clicking and dragging in the grid section of the Path Properties.

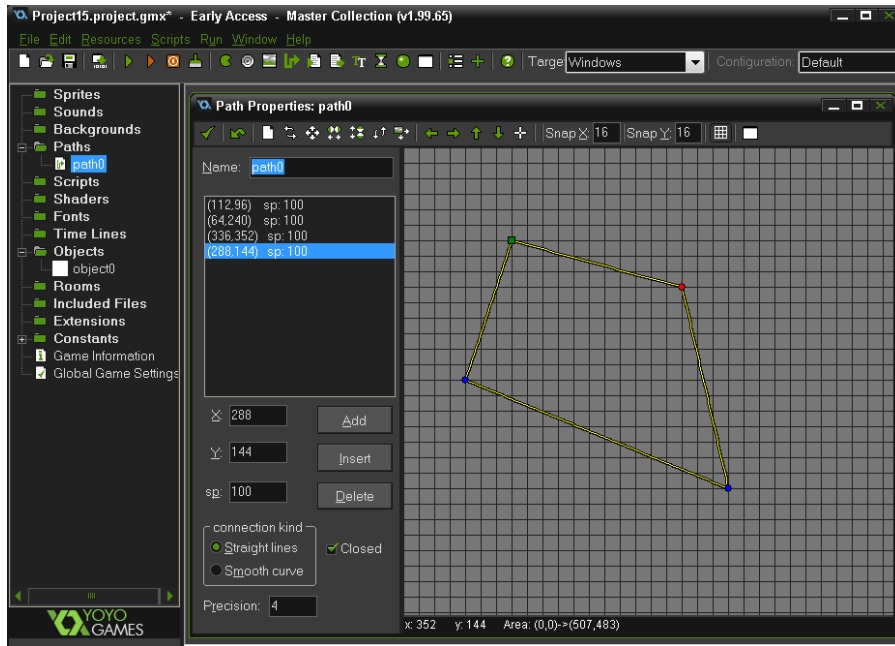


Figure 4-5: Use the Path Properties window to send an Instance on a specific Path.

The Path Actions, as shown in Figure 4-6, are configured directly to a Path that you create and then assign to an Action and, thus, an Object:

- ✓ **Set Path:** This Action enables you to choose a Path, set the speed, determine what to do at the end of the Path, and choose whether the Path is relative or absolute. Choose Relative when you want the start point to begin where the Instance is in the Room; choose Absolute when you've designed the Path location in relation to the Room.
- ✓ **End Path:** Use this Action to stop the Path for the Instance.
- ✓ **Path Position:** Use this Action to change the current position of the Instance in the Path. You must use a value between 0 and 1, with 0 being the beginning and 1 being the end.
- ✓ **Path Speed:** Use this Action to change the speed of the Instance on the Path. Negative speeds move the Instance backward. If you use 0, that will temporarily stop the motion along the path.

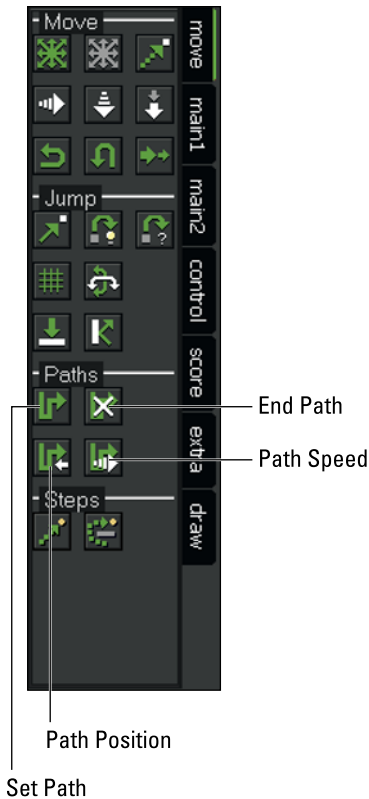


Figure 4-6: The Path Actions from the Move tab.

Step Actions

There are two Step Actions — Step Towards and Step Avoid — as shown in Figure 4-7:

- ✓ **Step Towards:** Use this Action to have an Instance move in a specific direction or toward another Instance. The difference between Step Towards and Move Towards is that in Step Towards, you can set a parameter for stopping the Instance when it encounters either solid Instances or all Instances.
- ✓ **Step Avoid:** This Action is similar to Step Towards, except you can tell the Instance to avoid solid Instances or all Instances.

These Step Actions are a great way to introduce artificial intelligence (AI) to your games, such as having a zombie horde moving toward the player.

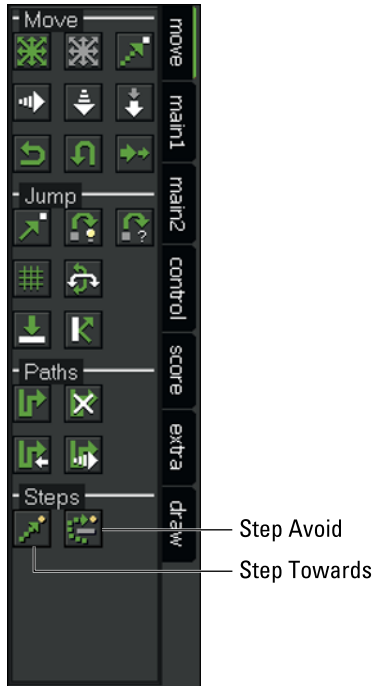


Figure 4-7: The Step Actions from the Move tab.

Main 1 Actions

The Main1 Actions include the Objects, Sprite, Sounds, and Room Actions. Figure 4-8 shows all the main1 Actions, which I cover in detail in this section.



Figure 4-8: The Main1 tab.

Objects Actions

The Objects Actions, as shown in Figure 4-9, are all about creating, changing, and destroying Objects:

- ✓ **Create Instance:** This Action is used to create Instances during the game when they aren't already placed in the Room.
- ✓ **Create Moving:** This Action enables you to set the speed and direction of a newly created Instance.
- ✓ **Create Random:** GameMaker will randomly create one of four different Objects. If you have fewer than four Objects, you can use No Object.
- ✓ **Change Instance:** Use this Action to change one Instance, such as a plane, into another Instance, such as an explosion.
- ✓ **Destroy Instance:** Boom goes the Instance and — poof! — there it's gone. The name of the Action says it all: Use this Action to destroy an Instance and remove it from the Room.
- ✓ **Destroy at Position:** This Action is used in conjunction with a bounding box. When you select the Relative check box, the position is determined in relation to the position of the current Instance.



Figure 4-9: The Objects Actions from the Main1 tab.

Sprite Actions

The Sprite Actions, as shown in Figure 4-10, are all about — yep, you guessed it — Sprites. There are three Sprite Actions on the Main1 tab:

- ✓ **Change Sprite:** Use this Action to change the Sprite of an Instance. You can even configure which sub-image to use. (I discuss sub-images in Chapter 2.) This Action is useful if you want to change the appearance of a character when it changes direction.
- ✓ **Transform Sprite:** This Action changes the size and orientation of the Sprite for the Instance. You can make it smaller, larger, or even rotate it.
- ✓ **Colour Sprite:** Yes, the good folks at YoYo Games are from Scotland, and they spell color with a *u*. This Action blends in color of the current Sprite, so, if you want your Sprite to change colors, a good bit of advice is to create the Sprite in black and white, and then apply the color through this Action. You can also use transparency through this Action.

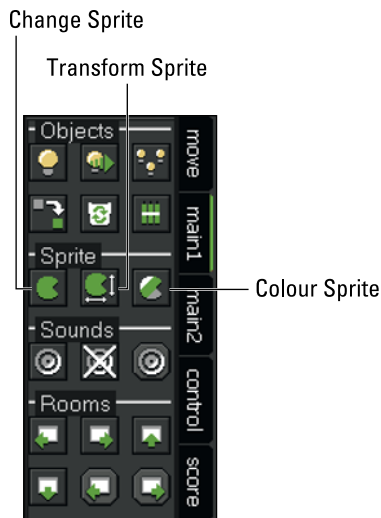


Figure 4-10: The Sprite Actions from the Main1 tab.

Sound Actions

Here are the three Sound Actions, shown in Figure 4-11:

- ✓ **Play Sound:** Use this Action to play a selected audio file.
- ✓ **Stop Sound:** Use this Action to stop the audio file from playing.
- ✓ **Check Sound:** Use this Action when you want GameMaker to test if a particular sound is playing and, if it is, to trigger the next Action.

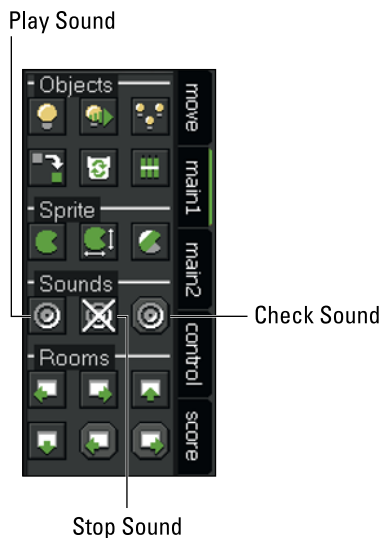


Figure 4-11: The Sound Actions from the Main1 tab.

Room Actions

The Room Actions are highlighted in Figure 4-12 and are best used when you want the player to move from one Room to the next. There are six different Room Actions:

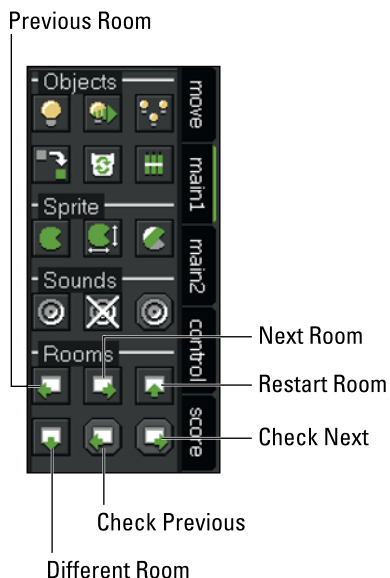


Figure 4-12: The Room Actions from the Main1 tab.

- ✓ **Previous Room:** This Action moves the player to the previous Room.
- ✓ **Next Room:** This Action moves the player to the next Room.
- ✓ **Restart Room:** This Action restarts the Room.
- ✓ **Different Room:** With this Action, you can select a different Room that you want the player to move to from a drop-down list.
- ✓ **Check Previous:** This Action checks to see if the previous Room exists; if it does, the next Action in the Action section is triggered.
- ✓ **Check Next:** This Action checks to see if the next Room exists; if it does, the next Action in the Action section is triggered.



You can change the order of Rooms through the Resource tree by clicking and dragging them up and down.

Main2 Actions

The Main2 tab has four different sections: Timing, Info, Game, and Resources (see Figure 4-13).



Figure 4-13: The Main2 tab.

Timing Actions

Timing Actions, as shown in Figure 4-14, are an important part of the game:

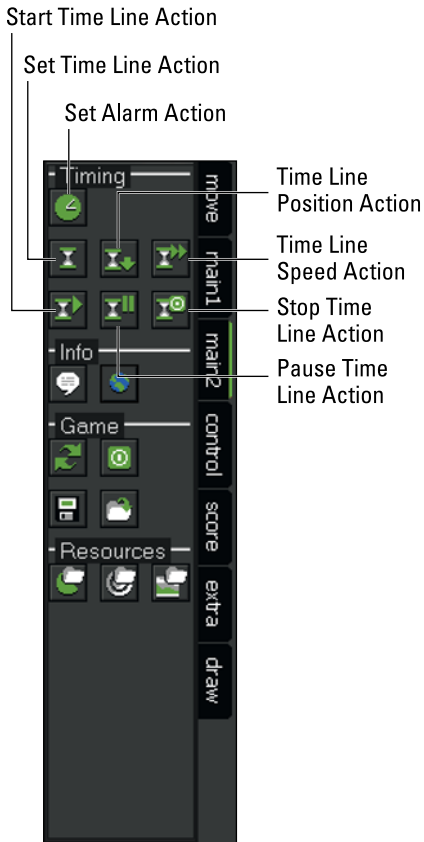


Figure 4-14: Timing Actions from the Main2 tab.

- ✓ **Set Alarm Action:** This Action enables you to set 1 of 12 different alarm clocks for any single Instance. You simply select the number of steps you want to occur before the alarm is triggered; then you set which Alarm to trigger.
- ✓ **Set Time Line Action:** This Action enables you to set which Time Line should run for that Object. You can also set the starting position from within the Time Line, with 0 marking the beginning.
- ✓ **Time Line Position Action:** You can change the position of the current Time Line. You can use this Action to skip or repeat sections of the Time Line.

- ✓ **Time Line Speed Action:** This Action enables you to change the speed of the Time Line.
- ✓ **Start, Pause, and Stop Time Line Actions:** These Actions enable you to set the chosen Action, as well as start, pause, or stop the Time Line.

Info Actions

There are two Info Actions, as shown in Figure 4-15:

- ✓ **Display Message:** Use this Action to make text appear on the screen.
- ✓ **URL Open:** Use this Action to have the game open a web page of your liking.



Figure 4-15: Info Actions from the Main2 tab.



To start a new line within a Display Message Action, use the # symbol. To get the # symbol to appear, use \#.

Game Actions

Here are the four different Game Actions, as shown in Figure 4-16:

Restart Game Action



End Game Action

Load Game Action

Save Game Action

Figure 4-16: Game Actions from the Main2 tab.

- ✓ **Restart Game:** Use this Action to restart the game — say, if the player loses all his lives.
- ✓ **End Game:** Use this Action to end the game. The End Game Action does not close out tabs or windows for the HTML5 target. But it does end the game.



If you use the iOS or Windows Phone target, the End Game Action may block the device. It's strongly recommended that you do *not* use this Action for these modules. Your final game will be rejected from the corresponding app store if this Action is used.



- ✓ **Save Game:** Use this Action to enable the player to save her game so that she can load it later. In the Save Game Properties window, there is a field for you to enter the name of the save file. So, if you save the game as AllBasesGone, you'll want to load the file, AllBasesGone.

Not all aspects of the game will be saved. For example, if the game is saved while a Particles event is running, the particles will not be saved (see Chapters 7 and 8 for information on working with particles). Sounds are also not saved. Chapter 14 is all about writing code for saving and loading player-specified configurations.

- ✓ **Load Game:** Use this Action to enable the player to load a previously saved game. In the field within the Load Game Properties window, type in the name of the file you want to load, such as AllBasesGone, which is the same filename you used to save a game.

Resources Actions

The Resource Actions, as shown in Figure 4-17, are all about replacing things like Sprites, Sounds, and Backgrounds:

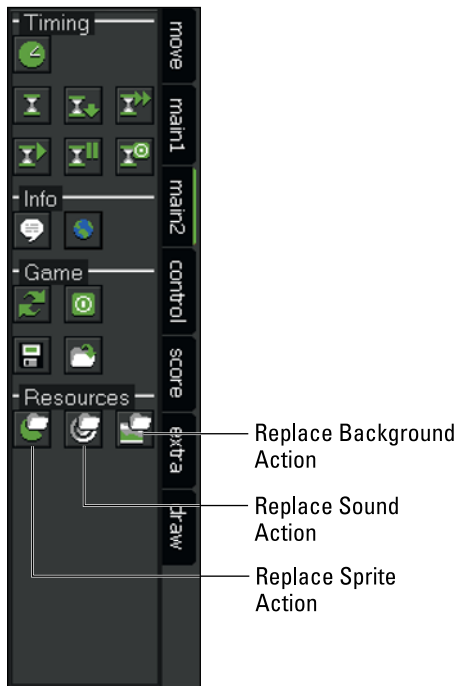


Figure 4-17: Resources Actions from the Main2 tab.

- ✓ **Replace Sprite:** This Action enables you to replace Sprites from the contents of a file. That way, you don't have to load all the Sprites into the game code. Instead, you can call that Sprite during the game. But don't call a Sprite that's currently in view, or you could get some odd effects.
- ✓ **Replace Sound and Replace Background:** These Actions work in a similar way to the Replace Sprite Action, in that you can store the files to be called during the game, so they aren't part of the game program. This helps speed up the loading of your game.



As of this writing, the Resource Actions are still available for use. However, a known bug exists and they should not be used, because they may not work. Please check for updates to see if these Actions are fixed or if they've been removed from the software.

Control Actions

The Control tab (shown in Figure 4-18) includes Actions for Questions, Other, Code, and Variables. These Actions can perform complex tasks.



Figure 4-18: The Control tab.

Questions Actions

There are nine Questions Actions, as shown in Figure 4-19. They ask questions such as, does something exist, is there a collision, are two values the same, and so on. And if so, the next Action in the Actions section is triggered.



Figure 4-19: Questions Actions from the Main2 tab.

- ✓ **Check Empty:** This Action asks to see if the Instance, if placed at an indicated position, would cause a collision. This Action does require that the Instance doing the checking has a Mask index or a Sprite with a valid collision Mask.
- ✓ **Check Collision:** This Action simply checks to see if the Instance collided with another Instance.
- ✓ **Check Object:** This Action checks to see if there is an Instance of a specified Object at the indicated position.
- ✓ **Test Instance Count:** This Action is useful if you want to test if there is a certain number of Instances in the Room, and if so, to perform the next Action.
- ✓ **Test Chance:** This Action takes the result of a one-in-X chance, where you set the X (the input to test). For example, if you use 10, then there will be a 1 in 10 chance (based on a simulated dice roll) that the next Action will be triggered.
- ✓ **Check Question:** This Action enables you to ask the player a yes-or-no question, which triggers a respective Action. The question appears in a pop-up window during the game.

- ✓ **Test Expression:** As described in the in-software manual, this Action is the most general question action whereby you can enter an arbitrary expression and evaluate it. If the expression evaluates to true (that is, a number larger or equal to 0.5) then the next Action (or block of Actions) is performed.
- ✓ **Check Mouse:** This Action returns true if the indicated mouse button is pressed and the next Action is performed; this Action is usually placed within a Step Event.
- ✓ **Check Grid:** This Action returns true if the Instance is within a grid.

Other Actions

The Other Actions section of the Main2 tab, shown in Figure 4-20, include a wide assortment of Actions associated with using code:



Figure 4-20: Other Actions from the Main2 tab.

- ✓ **Start Block and End Block:** Use these Actions to group a block of Actions, as shown in Figure 4-21. In this way, all the Actions within the block are performed if triggered. Normally, only the first Action that follows the question Action is triggered. Place the Start Block Action after the question being asked (in this case, a Test Variable Action). Place the End Block after the last Action you want triggered.

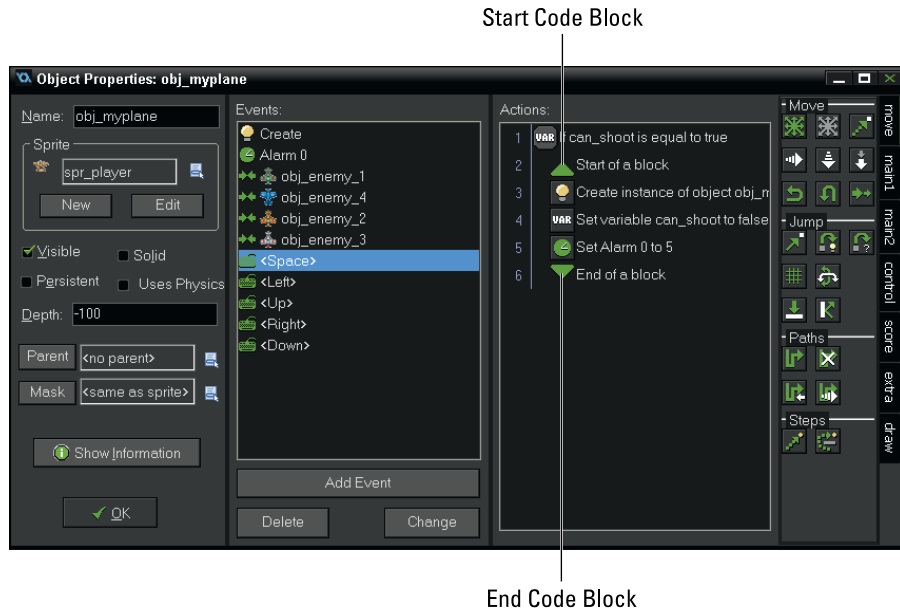


Figure 4-21: Use the Start and End Block Actions to group Actions.

- ✓ **Else:** Use this Action if you want something else to happen if the test question returns false instead of true. Set up your Actions so that you have the question, then a block of code to trigger if true, then an Else Action, and then another block of code to trigger if false.
- ✓ **Exit Event:** Use this Action if you want no further Actions to trigger within this Event.
- ✓ **Repeat:** Use this Action if you want an Action, or a block of Actions, to repeat a set number of times. This feature provides you the luxury of not having to simply paste in the same Actions over and over.
- ✓ **Call Parent Event:** Use this Action with child Objects with a Parent (discussed in Chapter 2). This is useful if you want to trigger the corresponding Event from the parent Object.

Code Actions

There are only three Code Actions, as indicated in Figure 4-22, but don't let the limited number of options fool you — the Code Actions can be very powerful.

- ✓ **Execute Code:** This Action opens up an entire world of custom code that you can write for GameMaker to execute. You can add anything from a simple, single line of code to something as complex as coding artificial intelligence.

- ✓ **Execute Script:** This Action enables you to call a Script. If you intend to reuse the same block of code many times for multiple Objects, then you can write a Script, which you can then call from within an Execute Code Action.
- ✓ **Comment:** This Action enables you to add a comment line to the Action list. These comments can remind you what the surrounding Actions are meant to do.



Figure 4-22: Code Actions from the Control tab.



Even though the Comment Action doesn't necessarily do anything, besides provide you a very helpful reminder, if you place a Comment Action within, say, a Collision Event, GameMaker will still trigger the collision. That is, Collision Events are only triggered when they have an Action, which can include a Comment Action.

Variables Actions

The three Variables Actions, as shown in Figure 4-23, are

- ✓ **Set Variable:** Use this Action to enter a value to a variable. If the variable does not exist, using this Action creates the variable. You can also use the Set Variable Action to configure one of GameMaker's many global variables, as well as the built-in Instance variables.
- ✓ **Test Variable:** This Action checks the value of a given variable, whether that be true, false, greater than, or lesser than, and then performs, or does not perform, the next Action based on the results.

- ✓ **Draw Variable:** This Action enables you to draw the value of a variable at a position that you choose. You can only use the Draw Variable Action within a Draw Event.

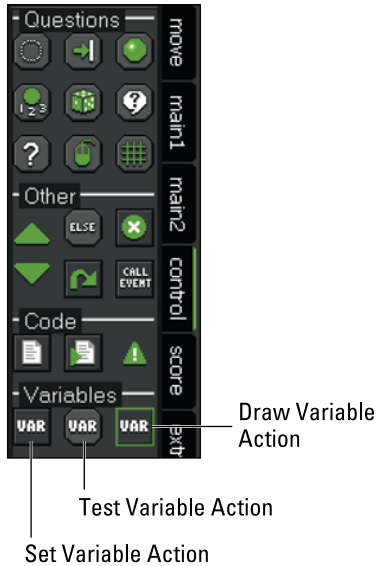


Figure 4-23: Variables Actions from the Control tab.

Score Actions

Scoreboard! Yep, that's right, this section is all about keeping score and seeing who's the winner and who's the loser. Figure 4-24 shows the score tab and its associated Actions, which includes score, lives, and health.

Score Actions

Score is considered a global variable within GameMaker and can be used with any Object. The Score Actions, shown in Figure 4-25, are the following:

- ✓ **Set Score:** This Action is used to determine the player's score. It's important to toggle Relative within the Set Score Action Properties window — you want the player to keep increasing his score, and not to maintain it (as in, you want the player to earn 10 points for every enemy killed and not just have a consistent score of 10).
- ✓ **Test Score:** This Action tests to see what the player's score is, and if it matches a certain value, to trigger the next Action.



Figure 4-24: The Score tab.

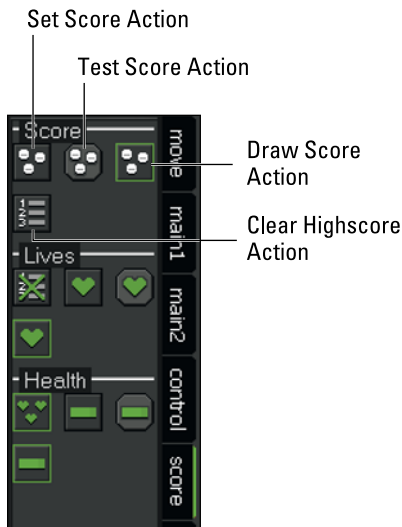


Figure 4-25: Score Actions from the Score tab.

- ✓ **Draw Score:** This Action enables you to draw the player's score in the game as simple text. All you have to do is configure the correct location within the Room.
- ✓ **Clear Highscore:** This Action does exactly as its name implies: clears the high score table, if you have one set up. You should be warned though, that the Clear Highscore Action cannot be undone.

Lives Actions

Here are the Lives Actions (see Figure 4-26):

- ✓ **Set Lives:** This Action determines how many lives the player has before it's game over. Lives is considered a GameMaker global variable.
- ✓ **Test Lives:** This Action enables you to, for example, set a maximum number of lives that the player can have, or a minimum number, if that's what you need for your game.
- ✓ **Draw Lives:** This Action draws the score based on the coordinates that you input. You must place this Action within a Draw Event for it to work.
- ✓ **Draw Life Images:** This Action is awesome for those times when you want to represent the number of player's lives through an image instead of text or a number.

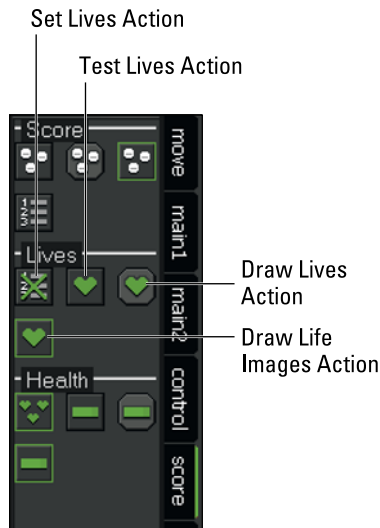


Figure 4-26: Lives Actions from the Score tab.

Health Actions

The Health Actions, shown in Figure 4-27, are similar to the Lives Actions — you have Set Health, Test Health, Draw Health, plus the additional Score Caption Action. Health is a global variable within GameMaker with a range of 0% to 100%. When a player's health reaches 0, an out-of-health Event is triggered.

- ✓ **Set Health:** This Action is used to set the amount of health a player has.
- ✓ **Test Health:** This Action can be fun to use — for example, if you want to warn the player that his health has fallen below a certain threshold, then the following Action is triggered.
- ✓ **Draw Health:** This Action is used to create a bar of health to indicate the strength of the player. You can set the position, size, and color of the health bar.
- ✓ **Score Caption:** This Action enables you to configure whether to display the score, lives, and health, including the caption for each. The Score Caption Action only works with Windows, Ubuntu (Linux), and MacOS targets.

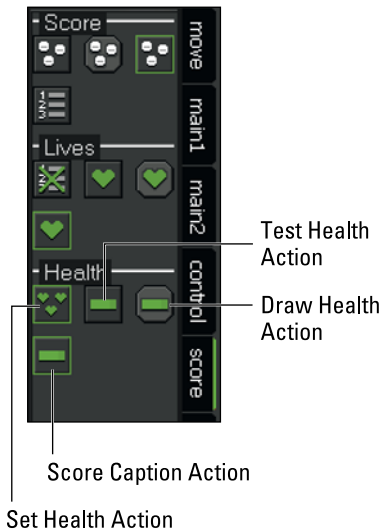


Figure 4-27: Health Actions from the Score tab.

Extra Actions

The Extra tab is all about, well, it's *mostly* about Particles, and then there's an Action for setting the mouse cursor thrown in for good measure. In the interest of page count, I refer you to Chapter 7, where you can see how to use the different Particles Actions to create some very special effects.

You can use the Set Mouse Cursor Action to set a Sprite to represent the cursor (choose the Sprite drop-down menu, shown in Figure 4-28), or show no cursor at all (choose the Cursor drop-down menu, also in Figure 4-28). Easy-peasey, lemon-squeezey, right?

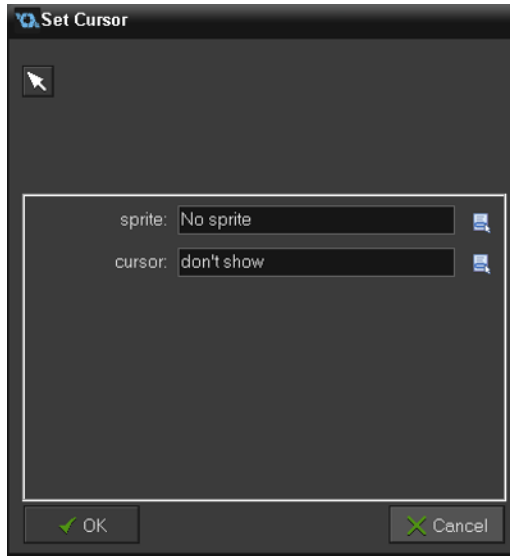


Figure 4-28: Set the Mouse Cursor Action.

Draw Actions

The Draw Actions include Drawing Actions, Settings Actions, and Other Actions, as shown in Figure 4-29.



Figure 4-29: The Draw tab.

Drawing Actions

The Drawing Actions, shown in Figure 4-30, are exactly that: They provide a means for you to draw.

- ✓ **Draw Self:** This is the same Action as the default draw that GameMaker uses. If you were to write the code instead of using a drag-and-drop Action, it would look like this:

```
draw_sprite_ext(sprite_index, image_index, x, y, image_xscale, image_yscale,
               image_angle, image_blend, image_alpha);
```

- ✓ **Draw Sprite, Draw Background, Draw Text, and so on:** Use these Actions to draw scaled text, rectangles, horizontal and vertical gradients, ellipses, and gradient ellipses. You can also draw lines and arrows with these Actions.

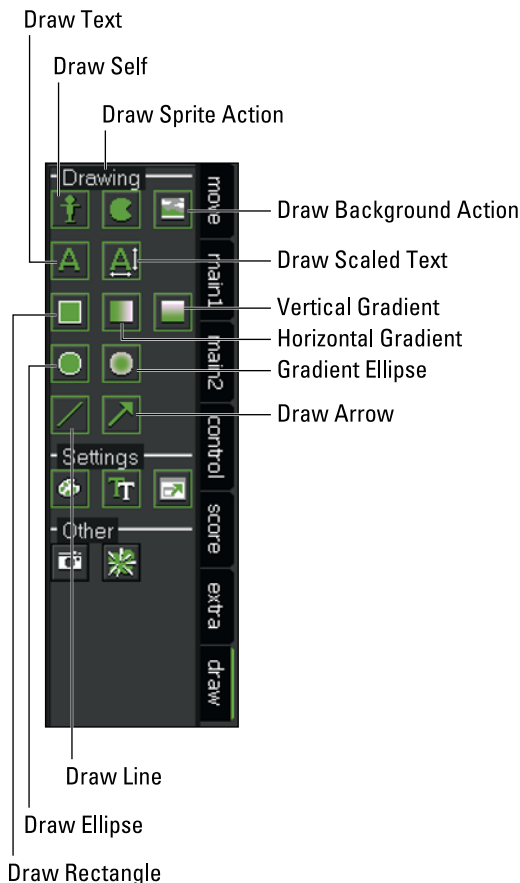


Figure 4-30: Drawing Actions from the Draw tab.

Settings Actions

GameMaker includes Actions for setting color (or, colour, if you prefer), fonts, and going full-screen, as shown in Figure 4-31.

- ✓ **Set Color:** This Action enables you to choose a color for every Instance drawn after this Action is triggered.
- ✓ **Set Font:** The default font for GameMaker games is Arial, 12 point, but if you add a new font to the Resource tree, you can specify for GameMaker to use that font using this Action.
- ✓ **Set Full Screen:** With this Action, you can enable the player to toggle going full-screen and back, which you can use as a player-configured option in a game.

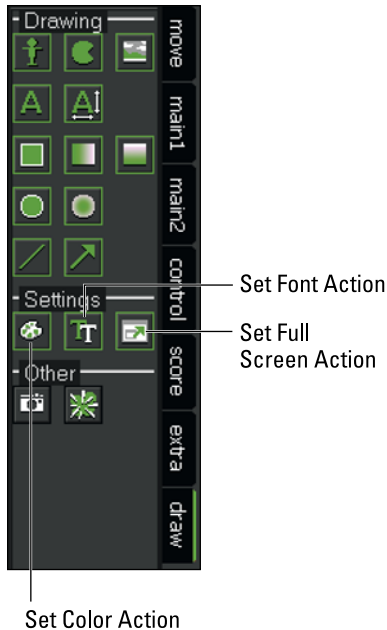


Figure 4-31: Settings Actions from the Draw tab.

Other Actions

The Other Actions on the Draw tab, shown in Figure 4-32, are

- ✓ **Take Snapshot:** This Action takes a screen capture of the game and stores it in a PNG file with a filename of your choosing. By default, GameMaker saves the screen capture in the %localappdata% folder of the game.

✓ **Create Effect:** Use this Action to create many effects in a simple and efficient manner, as I detail in Chapter 7.

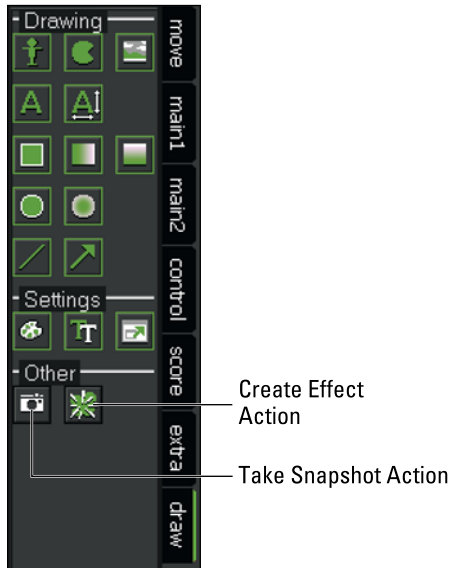


Figure 4-32: Other Actions from the Draw tab.

Working with Views

In This Chapter

- ▶ Adding speed and movement with Views
- ▶ Creating a Room to work with Views
- ▶ Moving the View with a controller Object
- ▶ Checking when the Object reaches the top of the Room
- ▶ Placing a score display in a Room with a View
- ▶ Keeping your player's plane from flying off the boundaries of the View
- ▶ Destroying an Instance when it leaves a Room
- ▶ Guiding bullets toward a target
- ▶ Creating enemy planes in a Room with a View
- ▶ Arranging enemy planes in a Room
- ▶ Moving on to the next Room

In this chapter, I introduce the concept of working with Views in your game. The basic concept of working with Views is that only a portion of the Room is seen at any given time. You set it up so that the Room actually moves down the screen along with Objects that you manually place in the Room.

For an example on how to use Views, I refer to the in-software tutorial Scrolling Shooter. In this tutorial, you begin by creating a background that moves down the screen to give an appearance of a scrolling background. All the enemies, islands, and so forth are randomly generated in the initial version of the game. In contrast, while working with Views, the Room itself actually moves down the screen, and the enemies and islands are manually placed in the Room instead of being randomly generated. This chapter focuses on creating a game with a View.

Using Speed and Movement with Views

When working with Views, keep in mind that it is the View that has speed and moves, which affects what the player sees on his screen. Therefore, some Objects, such as the planes from the Scrolling Shooter tutorial, will have a vertical speed (to stay in view and not be dragged off the screen as the View moves), and other Objects, such as the islands, will not have speed (so that they move along with the View, and are dragged off the screen).



The Room doesn't move; it's the View that moves. In that sense, the Room coordinates are absolute, while the View coordinates for Instances are relative to their position within the Room.



You may want to complete the first part of the in-software Scrolling Shooter tutorial so that you have all the Objects mentioned in the following procedures ready to go.

To add speed to an Object, follow these steps:

1. **Create an Object, such as the plane from the Scrolling Shooter tutorial, by choosing Resources → Create Object (see Chapter 2).**

2. **Choose Add Event → Create.**

The Choose the Event to Add menu appears.

3. **Choose Create from the Choose the Event to Add menu.**

The Create Event appears in the Events section.



4. **Drag and drop the Vertical Speed Action from the Move tab into the Actions section (see Chapter 4).**

The Speed Vertical window, shown in Figure 5-1, appears.



Figure 5-1: The Speed Vertical window.

5. In the Speed Vertical window, enter -2 in the Vert. Speed field.

A negative speed allows the Object to move up the Room; a positive speed allows the Object to move down the Room.

6. Click OK to save and close the Speed Vertical Action.



Rooms, Views, and the screen are separate entities. For example, the View can rotate, so an Instance can still move down the Room, but it will appear to be moving across the screen (because the View was rotated).

Now that the plane has some speed, you need to add speed to the Object you're using for the bullet. You do this in a similar fashion as you did with the Vertical Speed Action. To add speed to the player's bullets, follow these steps:

1. Open obj_mybullet (the Object you want to use for the bullet) from the Resource tree (see Chapter 2).

This Object was created as part of the Scrolling Shooter tutorial.

The Object Properties window opens.

2. Choose Add Event → Create (if the Create Event doesn't already exist).

The Create Event appears in the Events section of the Object Properties window.



3. Drag and drop the Vertical Speed Action from the Move tab into the Actions section.

The Vertical Speed window appears.

4. Enter -16 in the Vert. Speed field.

You want the bullet to move much faster than the plane.

5. Click OK in the Vertical Speed window.

The Vertical Speed window saves and closes.

6. Click OK in the Object Properties window.

The Object Properties window saves and closes.

Creating a Room with a View

When you create a Room to work with Views, you should do some planning ahead of time. Here are some questions to consider:

- ✓ How long and wide do you want the Room to be?
- ✓ How long do you want this level of the game to be?

For example, you can have several Rooms in each game, and each Room can represent a level of your game. Therefore, each Room needs to be sized so that it's big enough (plays long enough) so that you can incorporate each aspect of the level that you want.

Think of it this way: The Room can be a long, skinny column that appears to move down the screen. Only a certain portion of the Room — say, the bottom one-tenth — is seen at any one time. You then place Instances of Objects into the Room. For example, you can place a squadron of enemy planes directly outside the visible portion of the Room, so that they appear shortly after the start of the game.

To create a Room, follow these steps:

1. **Right-click Rooms from the Resource tree.**

A submenu appears.

2. **From the submenu, choose Create Room.**

The Room Properties window appears.

3. **Click the Settings tab, shown in Figure 5-2.**

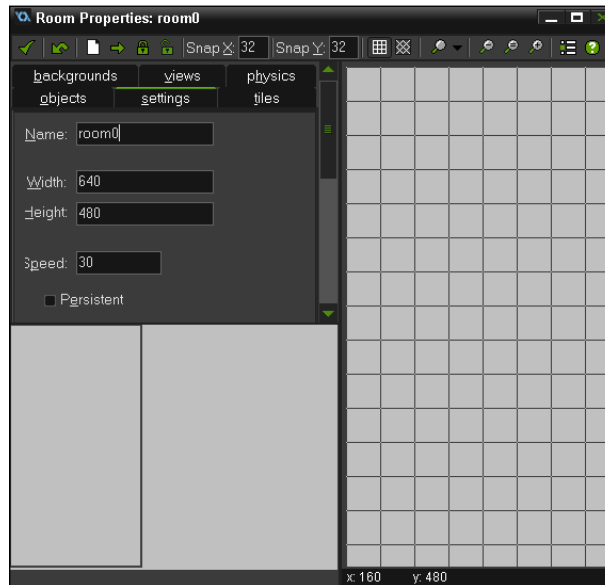


Figure 5-2: The Settings tab.

4. In the Name field, enter the name you want for the Room.
For example, you may enter **Level_1**.
5. In the Width field, enter 640; in the Height field, enter 4800; and in the Speed field, enter 30.
6. Click the Backgrounds tab, shown in Figure 5-3.

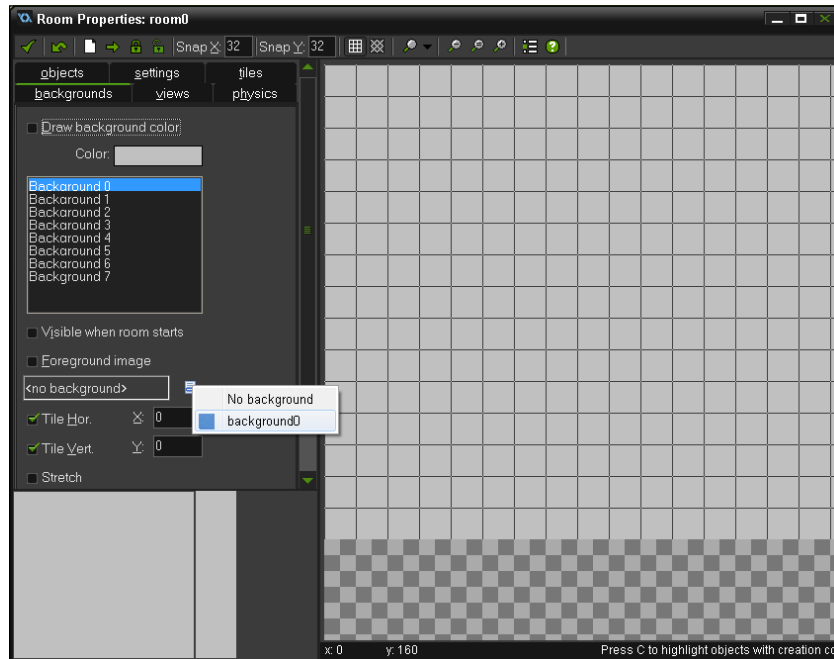


Figure 5-3: The Backgrounds tab.

7. Deselect the **Draw Background Color** check box.
8. Select a background image, such as **Background 0**, from the **Scrolling Shooter** tutorial.
9. Select the **Tile Hor.** and **Tile Vert.** check boxes.
10. Click the **Objects** tab, shown in Figure 5-4.
11. Select an Object, such as the player's plane (refer to Figure 5-4).
12. Click near the bottom of the Room, for example, where you see the plane in Figure 5-4.

An Instance of the Object appears in the Room.

13. Click the Views tab, shown in Figure 5-5.
14. Select the Enable the Use of Views check box.
15. Select the Visible When Room Starts check box.

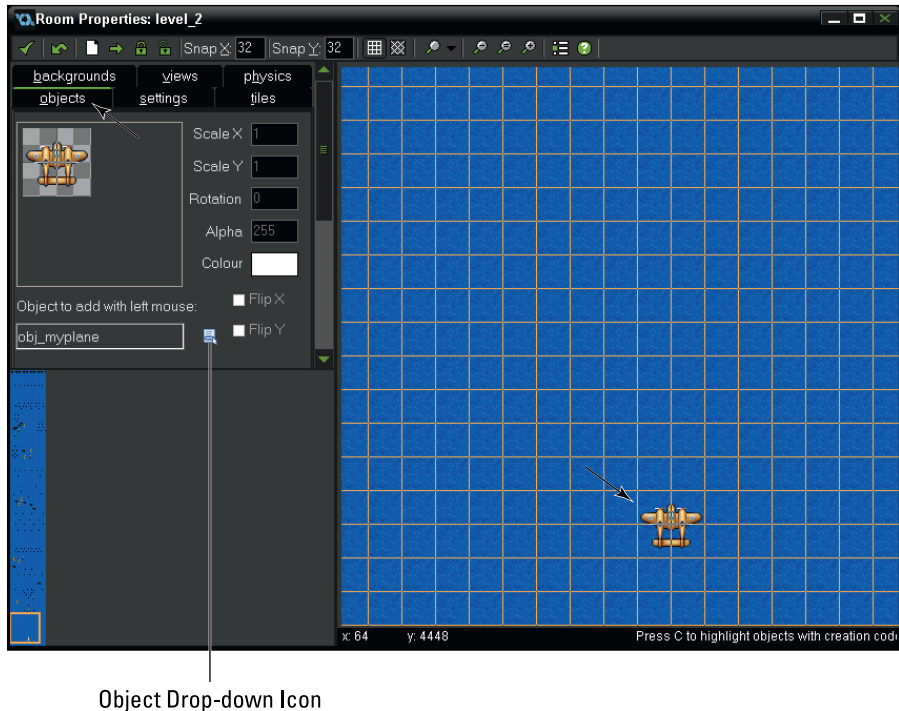


Figure 5-4: The Objects tab.

16. In the View in Room area, enter 4320 in the Y field.

You can leave the default settings for X, W, and H.

These fields indicate the dimensions of the Room that is visible: Y for the top border; X for the left border; W for the right border; and H for the bottom border.



When you have your View configured, you can see the area of the View as indicated by a thin, white border that appears on the Room grid (refer to Figure 5-5).

17. Click the green check mark from the icon menu.

The Room Properties window saves and closes.

If you want to see what you just created, you can run the game now to watch your plane fly up and leave the play area (I show you how to stop the plane from leaving the view later in this chapter).

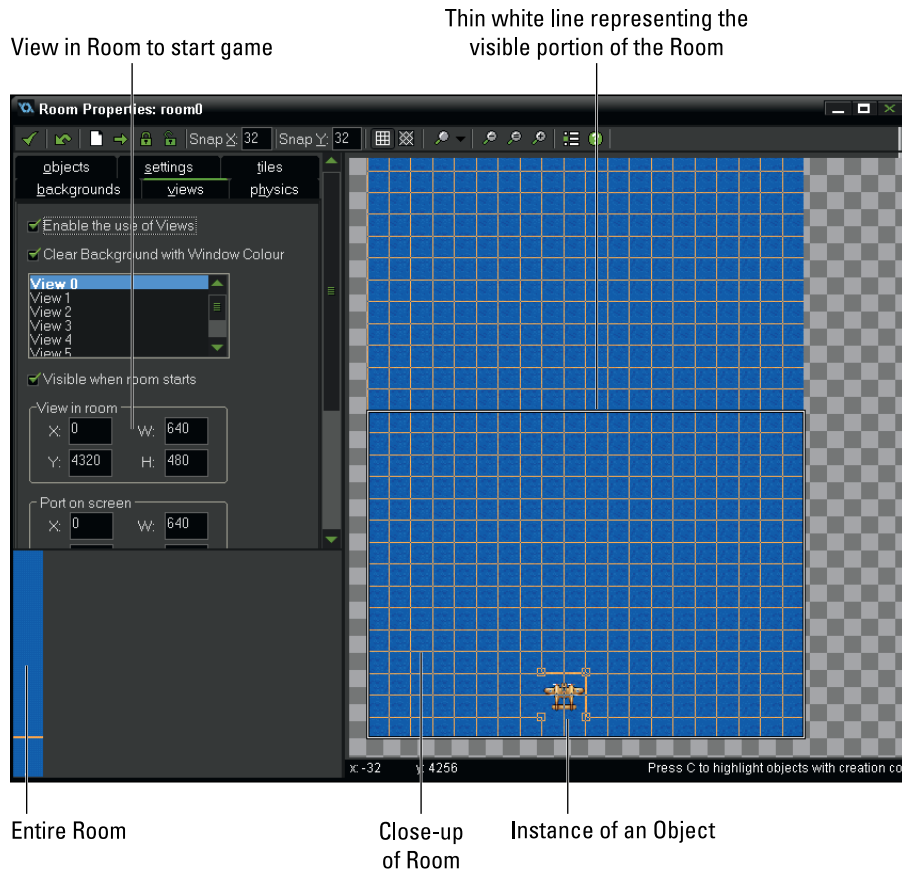


Figure 5-5: The Views tab.

Using a Controller Object to Move the View

You come back later to the Room to add in the enemy planes and the islands. But first, you need to create a controller Object to make the View move with a constant speed.

To create the controller Object, follow these steps:

1. **Create a new Object by choosing Resources⇨Create Object (see Chapter 2).**

The Object Properties window appears.

2. **In the Name field, type controller_Object.**

3. **Choose Add Event⇨Step⇨Step.**

A step Event appears in the Events section.

4. Click the Control tab.



5. Drag and drop the Set Variable Action from the Move tab into the Actions section (see Figure 5-6).

The Set Variable window, shown in Figure 5-6, appears.

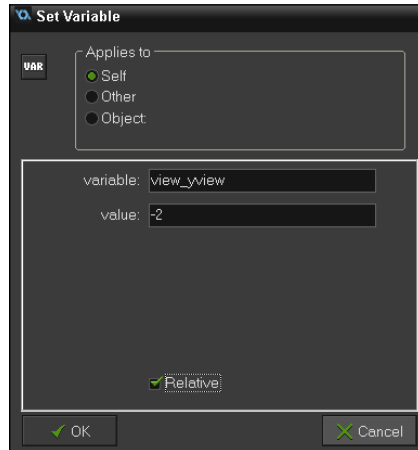


Figure 5-6: The Set Variable window.

6. Enter `view_yview` in the Variable field and `-2` in the Value field, and select the Relative check box.

You just set a variable called `view_yview` and set the value to `-2`. Now, for each step of the game, the position of the View moves 2 pixels upward, which creates a scrolling effect.

Reaching the End of the Room

Now you need to add a check to see when the Object (the player's plane) reaches the top of the Room ($Y < 0$). Then the game should either stop or move on to the next level.

To add the check when an Object reaches the end of the Room, follow these steps:



1. In the Step Event of the controller Object (see the previous procedure), drag and drop the Test Variable Action from the Control tab into the Actions section.

The Test Variable Action properties appear.

2. In the Applies To section, select the Object radio button.

You can now choose, via a drop-down list, which Object you want this Action to apply to.

3. Click the Object drop-down list and choose the Object you want to test, such as the Object for the player's plane.

The Object field populates with the name of the Object you chose.

4. In the Variable field, type y.
5. In the Value field, type 1 (or where in the Room you want to end the game or move on to the next Room).
6. For the Operation field, click to open the Operation drop-down list and choose Less Than.

GameMaker uses floating point values, which means the Y position may never actually hit 0. For example, the Y position may be 0.00000015. Therefore, you want to make sure to configure the Y value to less than the value of 1.



7. Drag and drop the Display Message Action from the Main2 tab into the Actions section.

The Display Message Action properties window appears.

8. In the Message field, type End Room.
9. Click OK to save and close the Action.

The Step Event should now look like Figure 5-7 and should provide a good test to see if everything is working. Later in this chapter, I show you how to update the Step Event so that the player moves on to another Room when he finishes the current Room.

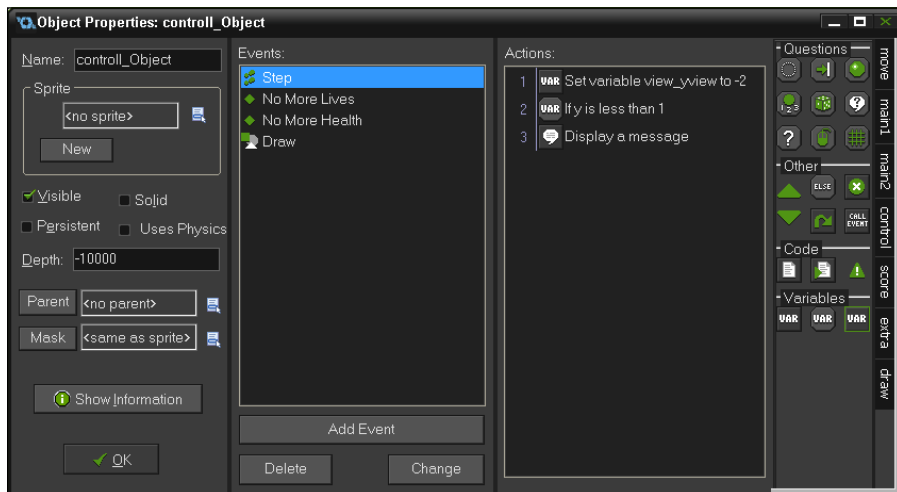


Figure 5-7: The Step Event configured to move the Room and check to see when the player reaches the end of the Room.

Positioning the Score Display in a Room with a View

To display the score in a Room with a View, follow these steps:

1. **Open the controller Object from the previous sections, or create a new Object by choosing Resources→Create Object (see Chapter 2).**
2. **In the Name field, type controller_Object (if not already).**
3. **Choose Add Event→Draw→Draw.**

A Draw Event appears in the Events section.



4. **Drag and drop the Draw Sprite Action from the Draw tab into the Actions section.**

The Draw Sprite Action properties window appears.

5. **In the Applies To section, select the Self radio button.**
6. **For the Sprite field, click the Sprite drop-down list and choose spr_status (choose an appropriate Sprite).**
7. **In the X field, type 0.**
8. **In the Y field, type $\text{view\_yview}[0] + \text{view\_hview}[0] - 76$.**



View variables are arrays, and as such, you should always use the View number when referring to them. However, GameMaker automatically selects `view[0]` when the array is not implicitly shown.

9. **In the subimage field, type -1.**
10. **Click OK to save and close the Action.**

Now you need to add a Draw Score Action to the controller Object, by following these steps:



1. **Drag and drop the Draw Score Action from the Score tab into the Actions section.**

The Draw Score Action properties window appears.

2. **In the X field, type 230.**
3. **In the Y field, type $\text{view\_yview}[0] + 417$.**
4. **Click OK to save and close the Action.**

Now you need to add a Draw Life Images Action to the controller Object, by following these steps:



1. **Drag and drop the Draw Life Images Action from the Score tab into the Actions section.**

The Draw Life Images Action properties window appears.

2. In the X field, type 175.
3. In the Y field, type `view_yview[0] + 439`.
4. For the Image field, click to open the Image drop-down list and choose `spr_life` (or whatever Sprite you want to represent the number of lives the player has left).
5. Click OK to save and close the Action.

Now you need to add Draw Health Action to the controller Object, by following these steps:



1. Drag and drop the Draw Health Action from the Score tab into the Actions section.

The Draw Health Action properties window appears.

2. In the X1 field, type 12.
3. In the Y1 field, type `view_yview[0] + 449`.
4. In the X2 field, type 138.
5. In the Y2 field, type `view_yview[0] + 459`.
6. For the Back Color field, click to open the Back Color drop-down list and choose None.
7. For the Bar Color field, click to open the Bar Color drop-down list and choose Green to Red.
8. Click OK to save and close the Action.



Place an Instance of the controller Object in the Room. In Chapter 2, I provide more detail on how to do this, but here's a quick refresher: Open the Room from the Resource tree. In the Room Properties window, click the Objects tab. Open the submenu with the list of Objects, and choose your controller Object. Then click anywhere in the Room to place an Instance of the controller Object in the Room (it will appear as a blue circle with a question mark inside of it).

Setting Object Boundaries for Rooms with Views

Currently, the player's plane can fly right off the top and bottom of the screen. You probably don't want that to happen in your game, so to prevent that, you'll want to set up a Test Variable Action.

To set the top vertical boundary for an Object, follow these steps:

1. With your Object (for example, `obj_myplane`) open, choose Add Event→Keyboard→<up>.

The Keyboard Event for Up appears in the Events section.



2. Drag and drop the Test Variable Action from the Control tab into the Actions section.

The Test Variable Action properties window appears.

3. In the Applies To section, select the Self radio button.
4. In the Variable field, type `y`.
5. In the Value field, type `view_yview[0] + (sprite_height / 2)`.
6. In the Operation field, open the submenu and choose Greater Than.
7. Click OK to save and close the Action.



8. Drag and drop the Jump to Position Action from the Move tab to the Actions section.

The Jump to Position Action properties window appears.

9. In the Applies To section, select the Self radio button.
10. In the X field, leave the default at 0.
11. In the Y field, type `-4`.
12. Select the Relative check box.
13. Click OK to save and close the Action.

Now, GameMaker will run a test to see if the player moves the plane to a height greater than the relative yview of the Room. If the Instance reaches a height greater than the height of the yview, the Instance performs the next Action after the Test Variable Action, which is the Jump to Position Action. The Jump to Position Action moves the Object down by 4 pixels to keep it within view of the player.



It's a good idea to run Test Variable Actions to see if the Instance of the Object hits the bottom, left, and right boundaries, as well as the top. The following procedure shows you how to test for the bottom boundaries.

To set the bottom vertical boundary for an Object, follow these steps:

1. Choose Add Event⇨Keyboard⇨<down>.

The Keyboard Event for Down appears in the Events section.



2. Drag and drop the Test Variable Action from the Control tab into the Actions section.

The Test Variable Action properties window appears.

3. In the Applies To section, select the Self radio button.
4. In the Variable field, type `y`.
5. In the Value field type `view_yview[0] + view_hview[0] - (sprite_height / 2) - 76`.

6. For the Operation field, click to open the Operation drop-down list and choose **Less Than**.

7. Click **OK** to save and close the Action.



8. Drag and drop the **Jump to Position Action** from the **Move** tab into the **Actions** section.

The Jump to Position Action properties window appears.

9. In the **Applies To** section, choose the **Self** radio button.

10. In the **X** field, leave the default at 0.

11. In the **Y** field, type 4.

12. Select the **Relative** check box.

13. Click **OK** to save and close the Action.

Now, your Object should stop moving when it reaches both the top and bottom boundaries. You can also use this procedure for Left and Right Keyboard Events:

✓ **To set the right boundary:** Create a Right Keyboard Event. In the Test Variable Action, type **x** in the Variable field; type **room_width-40** in the Value field; and in the Operation field, choose **Less Than**. In the Jump to Position Action, type **4** in the X field and leave Y at 0.

✓ **To set the left boundary:** Create a Left Keyboard Event. In the Test Variable Action, type **x** in the Variable field; type **40** in the Value field; and in the Operation field, choose **Greater Than**. In the Jump to Position Action, type **-4** in the X field and leave Y at 0.



In Step 5 of the last two procedures, you used the code, `(sprite_height / 2)`. This code is in relation to the Origin of the Sprite, as discussed in Chapter 2, which is the center of the Sprite. If you didn't include this bit of code, half the Sprite would disappear before the Test Variable was triggered to run the Jump to Position Action.

Destroying Instances When They Leave a Room

If your game involves bullets, or any other Object that you want destroyed when it leaves the Room, the following procedure was made just for you!

To destroy an Instance when it leaves the Room, follow these steps:

1. **Open the Object you want destroyed when it leaves the Room.**

You can do this from the Resource tree.

2. **Choose Add Event→Other→Outside Room.**

The Outside Room Event appears in the Events section.



3. **Drag and drop the Destroy Instance Action from the Main1 tab to the Actions section.**

The Destroy Instance Action properties window appears.

4. **In the Applies To section, you can leave the Self radio button selected.**
5. **Click OK to save and close the Action.**

Now, Instances of the Object that leave the Room will be destroyed.

Shooting Guided Bullets in a Room with Views

If you want the enemy to have bullets that can track other Instances in the Room (say, the player), and you're using Views, you need to use certain values for the targeting to work. I call these types of bullets *guided bullets*. In the following procedure, I show you how to add guided bullets to an enemy plane Object.

To create a guided bullet in a Room with a View, follow these steps:

1. **Create a new Object by choosing Resources⇨Create Object.**

The Object Properties window appears.

2. **In the Name field, type, say, obj_enemy_bullet.**
3. **Choose Add Event⇨Create Event.**

A Create Event appears in the Events section.



4. **Drag and drop the Test Instance Count Action from the Control tab into the Actions section.**

The Test Instance Count Action Properties window appears.

5. **For the Object field, click to open the Object drop-down list and choose the Object that you want to test for.**

For example, if you want the enemy to shoot at the player, choose the player Object.

6. **In the Number field, type 1.**

This tests that there is at least one Instance of the assigned Object in the Room. You can't have the enemy shooting at a player if an Instance of the player Object is not in the Room (for example, if it has already been destroyed).

7. **For the Operation field, click to open the Operation drop-down list and choose Equal To.**

There needs to be one Instance of the Object in the Room. If there is, then GameMaker moves on to the next Action.



If you tell GameMaker to reference a variable that doesn't exist (for example, to follow an Object that doesn't exist), an error will occur and the game could crash. That's why it's important to test to make sure at least one Instance of the Object exists.



8. Drag and drop the Move Towards Action from the Move tab to the Actions section.

The Move Towards Action properties window appears.

9. In the Applies To section, select the Self radio button.

10. In the X field, type, say, obj_myplane.x.

Use the name of the Object that you want the bullet to chase after.

11. In the Y field, type, say, obj_myplane.y.

Use the name of the Object that you want the bullet to chase after.

12. In the Speed field, type 8.

13. Click OK to save and close the Action.

You've now created a guided bullet Object that will track the player's plane. To apply this bullet to an Object, such as an enemy plane that will shoot it, check out the next section.

Creating Enemy Planes in a Room with Views

In this section, I show you how to create enemy planes that work in a Room with a View. A couple of the attributes that you create include destroying the planes when they leave the Room and destroying all the bullets, such as the guided bullet described in the previous section, when they leave the Room. After you create the enemy planes, I show you how to add these planes to the Room.

To create enemy planes, follow these steps:

1. Create a new Object by choosing Resources → Create Object.

The Object Properties window appears.

2. In the Name field, type, say, enemy_plane.

3. Choose Add Event → Step Event.

A Step Event appears in the Events section.



4. Drag and drop the Comment Action from the Control tab into the Actions section.

The Comment Action properties window appears.

5. In the comment field, type something such as Check in View.

This is solely for your reference as you're working. Comments are a great reminder of the Actions that follow.

6. Click OK to save and close the Action.

Now you need to add a Test Variable Action to the enemy plane Object.



1. Drag and drop the Test Variable Action from the Control tab into the Actions section.

The Test Variable Action properties window appears.

2. In the Applies To section, select the Self radio button.

3. In the Variable field, type y.

4. In the Value field, type view_yview[0]-32.

5. For the Operation field, click to open the Operation drop-down list and choose Less Than.

6. Click OK to save and close the Action.

You just set a Test Variable to see if an Instance of the Object is not within the View based on the Y coordinate. If it isn't within the View, the next Action is triggered.

Now you need to add an Exit Event Action and a Test Variable Action to the enemy plane Object.



1. Drag and drop the Exit Event Action from the Control tab into the Actions section.

The Exit This Event Action appears in the Actions section. There is nothing to set for this Action. This is the next Action that is performed if the Test Variable Action is triggered. If the Test Variable Action is triggered, the Exit Event Action is triggered, which means no more Actions will be triggered for this Event (until the Test Variable stops being triggered and the Exit Event Action is not triggered as a result).



2. Drag and drop the Test Variable Action from the Control tab into the Actions section.

The Test Variable Action properties window appears.

3. In the Applies to section, select the Self radio button.

4. In the Variable field, type y.

5. In the Value field, type view_yview[0]+480.

You just set a Test Variable to see if an Instance of the Object is not within the View based on the Y coordinate. If it isn't within the View, the next Action is triggered.

6. For the Operation field, click to open the Operation drop-down list and choose Greater Than.

7. Click OK to save and close the Action.



8. Drag and drop the Exit Event Action from the Control tab into the Actions window.

The Exit This Event Action appears in the Actions section and operates the same as in Step 1.



9. Drag and drop the Comment Action from the Control tab into the Actions window.

The Comment Action properties window appears.

10. In the Comment field, type something such as In View, so do the next Action.

11. Click OK to save and close the Action.

You just created a comment that reminds you what the following group of Actions will do when triggered, which is that the Object is currently within view and the following Actions should trigger.

Now you need to add a Speed Vertical Action to the Object.



1. Drag and drop the Speed Vertical Action from the Move tab into the Actions window.

The Speed Vertical Action properties window appears. This is the Action that is triggered when the first two Test Variable Actions are not triggered due to the Instance being within view.

2. In the Vert. Speed field, type 2.

3. Click OK to save and close the Action.

The Instance of the Object will now have speed and movement.

Now add a Test Variable Action to the Object.



1. Drag and drop the Test Variable Action from the Control tab into the Actions section.

The Test Variable Action properties window appears.

2. In the Applies To section, select the Self radio button.

3. In the Variable field, type y.

4. In the Value field, type `view_yview[0]+480`.

5. For the Operation field, click to open the Operation drop-down list and choose Greater Than.

You just set a test to see if the Instance leaves the visible portion of the Room. If it does, then the next Action is performed.

6. Click OK to save and close the Action.

Next, add a Destroy Instance Action to the Object.



1. Drag and drop the Destroy Instance Action from the Main1 tab into the Actions section.

The Destroy Instance Action properties window appears.

2. In the Applies To section, select the Self radio button.

3. Click OK to save and close.

The Instance will now be destroyed if it leaves the Room, based on the Test Variable Action you set earlier.



4. Drag and drop the Test Chance Action from the Control tab into the Actions section.

The Test Chance Action properties window appears.

5. In the Sides field, type 80.

6. Click OK to save and close the Action.

You just created a chance variable that will trigger the next Action with a dice roll's chance of 1 out of every 80. That is, for every step of the game, GameMaker will roll an 80-sided die to determine if the next Action is triggered. If you want the next Action to be triggered more often (you want the enemy to shoot more often), lower this value, from 80 to, say, 20.

Now, add a Create Instance Action to the Object.



1. Drag and drop the Create Instance Action from the Main1 tab into the Actions section.

The Create Instance Action properties window appears.

2. In the Applies To section, select the Self radio button.

3. For the Object field, click to open the Object drop-down list and choose, say, `obj_enemy_bullet` (the name of the Object that you want created, such as the guided bullet described in the previous section).

4. In the X field, type 0.

5. In the Y field, type 0.

6. Select the Relative check box.

7. Click OK to save and close the Action.

You just created an Action that creates a bullet (`obj_enemy_bullet`).

Your Step Event for the enemy plane should look like Figure 5-8.

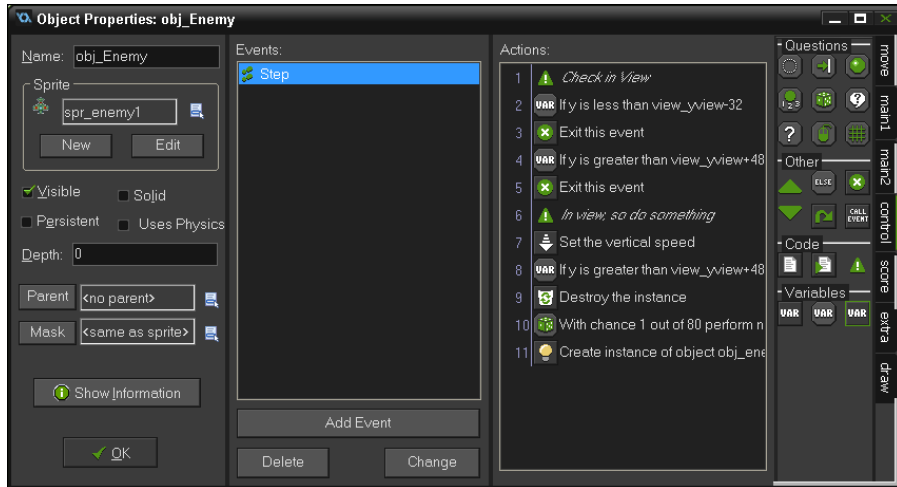


Figure 5-8: The Step Event for the enemy plane.

The enemy plane will now be destroyed if it leaves the View. Also, it won't start to move until it enters the View. Plus, it has a sweet guided bullet tracking system set on full destruction of the player. (Insert evil laugh here.)

Creating Patterns of Enemy Planes in a Room with Views

When using Views, you place the enemy planes in the Room, one by one, to form very cool formations. This is where you can let your imagination run wild in coming up with different patterns to challenge the player. The following procedure steps you through this process.

To add the enemy planes in your Room with a View, follow these steps:

1. **Open (or create) the Room from the Resource tree (see Chapter 2).**

The Room Properties window appears.

2. **Click the Objects tab.**
3. **Select an enemy plane Object.**
4. **Click in the Room near where the player starts.**

When you click in the Room, you place an Instance of the Object that you selected.

5. **Scroll up the Room, outside the visible portion of the view, select a higher-level (tougher) enemy Object, and add a few Instances in the Room.**

6. Continue adding different enemy types in the Room so that there are plenty of targets for the player to shoot at and avoid.
7. Add Instances of the different island types as well.
8. Click the green check mark to save and close the Room Properties window.

You now have a Room filled with enemy planes ready for the player to shoot out of the sky.

Transitioning to the Next Room

For this section, I assume that you have at least two Rooms created. If you have two Rooms ready to go, you're ready to create a transition between Rooms.



Earlier in this chapter, I show you how to set up a message to display End Room when the player reached the end of the Room. Here, I show you how to update those Actions to transition to the next Room after the player clicks the message box to move on to the next Room.

To transition the player to the next Room, follow these steps:

1. **Open the properties for your controller Object (or create a new Object by choosing Resource → Create Object).**

The Object Properties window appears.

2. **In the Name field, type a name for your Object — for example, `controller_object`.**
3. **Choose Add Event → Step → Step (or select the Step Event if it already exists).**

A Step Event appears in the Events section.



4. **Drag and drop the Start Block Action from the Control tab to the Actions section so that it's above the Display Message Action (if you have one).**

The Start of a Block Action appears in the Actions section. There is nothing for you to configure here. For more information on this Action, and all Actions, see Chapter 4.



5. **Drag and drop the Next Room Action from the Main1 tab to the Actions section.**

The Go to Next Room Action appears in the Actions section. There is nothing for you to configure here. For more information on this Action, and all Actions, see Chapter 4.



6. Drag and drop the End Block Action from the Control tab to the Actions section.

The Go to Next Room Action appears in the Actions section. There is nothing for you to configure here. For more information on this Action, and all Actions, see Chapter 4.

You just set up a Step Event, with a series of Actions that will trigger when the Y coordinate of the player's plane reaches the end of the Room. The Step Event should look like Figure 5-9.

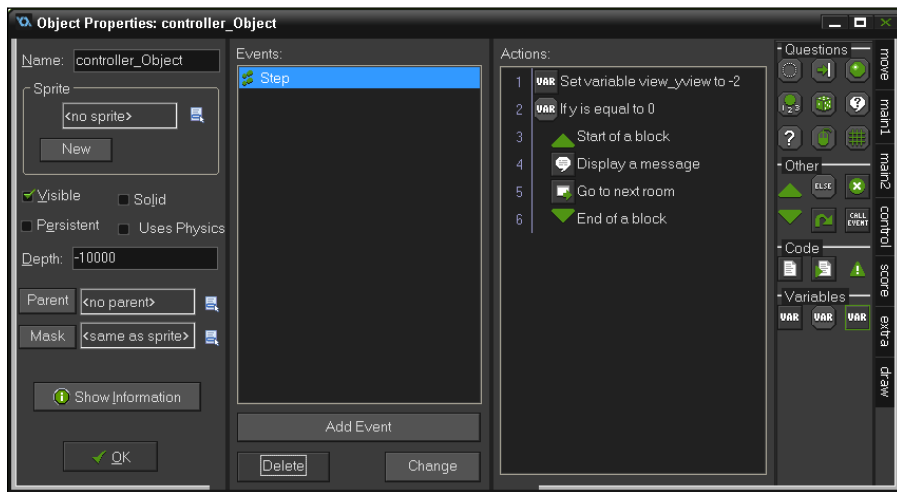


Figure 5-9: A Step Event with a code block for transitioning to the next Room.



If you want to keep the Display a Message Action (you set this up earlier, in the “Reaching the End of the Room” section) to indicate to the player that she reached the end of the Room, you can click and drag that Action, from within the Actions section, into the code block (refer to Figure 5-9).

You now have a game with two different levels that the player can advance through as she completes a Room. The player's score carries over to the next Room because score is a global variable and will be relative no matter which Room the player is in.

Working with Time Lines

In This Chapter

- ▶ Creating a Time Line
- ▶ Using Objects with Time Lines
- ▶ Placing a Time Line Object in a Room
- ▶ Introducing a boss battle

In this chapter, I show you how to create a Room using a Time Line, which you can then use as a level in your game. I conclude the chapter by providing an example of how to create a boss battle.

I'm particularly proud of the boss battle tutorial, because I came up with it on my own after finishing the in-software Scrolling Shooter tutorial (you can find tutorials within GameMaker from the New Project window). As I was teaching myself how to create the boss battle, my research showed me how to perform certain Actions through code instead of using the drag-and-drop Action icons. Later, I learned that I could perform those Actions I coded by hand by using the drag-and-drop Actions. It just goes to show that there are multiple ways to perform the same Actions — by using code or by using the drag-and-drop Action icons. GameMaker is such a flexible tool that you can mix and match handwritten code (through an Execute Code Action; see Chapter 4) and the Action icons within the same Events.

When using Time Lines, you need to become familiar with how X and Y coordinates work, which is how you place Instances of Objects in the Room. At first, using Time Lines may seem like an arduous task of determining X and Y coordinates as opposed to placing the Instances in the Room by hand (as you do when using Views; see Chapter 5). However, after you've set up an enemy fleet of planes, the Time Line provides an extremely flexible way to quickly change when the enemy planes appear by simply updating the Step number. When you manually place the planes using Views, you have to go back and select each plane and then move them up and down the Room to change when they appear. There are trade-offs to each method — you'll learn through experience when it's the best time to use the different techniques.

Creating a Time Line

Using Time Lines to create and control enemy behavior gives you significant control over how your game plays. The in-software Scrolling Shooter tutorial begins by showing how to create a game using a random generator for the enemy planes. This creates a challenging game in that it continuously gets harder over time, but there are problems with that approach, such as planes overlapping each other. Time Lines can help solve that problem.

When you use Time Lines, you present the player with the same pattern of enemy planes each time they play the game. It provides players with an opportunity to learn the game, so they can play it better over time. This gives players a sense of satisfaction — which is important, because they'll want to continue playing your game!

When working with Time Lines, you use Time Line resources; within each Time Line resource, you specify a number of Moments in time (measured in steps of the game); and for each Moment, you create an Action. In this tutorial, you create Time Line resources that control when enemy planes appear. The planes should be destroyed when they leave the Room and not reappear.



Just like with creating a Sprite, Object, or Sound, there are multiple ways to create a Time Line. You can do so from the main menu (Resources⇨Create Time Line), from the icon menu, or by right-clicking Time Lines from the Resource tree and choosing Create Time Line.

When you create a Time Line, you work with the Time Line Properties window, which looks similar to the Object Properties window. You can name the Time Line, you can add Moments (as compared to Events when working with Objects), and to those Moments you drag and drop Actions from the tabs (see Figure 6-1).

To create a Time Line, follow these steps:

- 1. From the main menu, choose Resources⇨Create Time Line.**

The Time Line Properties window appears.

- 2. In the Name field, type a name for this Time Line, such as level_one.**

The name appears in the Name field.

- 3. Choose Add (found directly below the Name field).**

The Adding a Moment dialog box appears (see Figure 6-2).

A Moment is measured in steps, so if you enter 1 in the Adding a Moment dialog box, the Actions you assign to this Moment will trigger at the first step of the game.



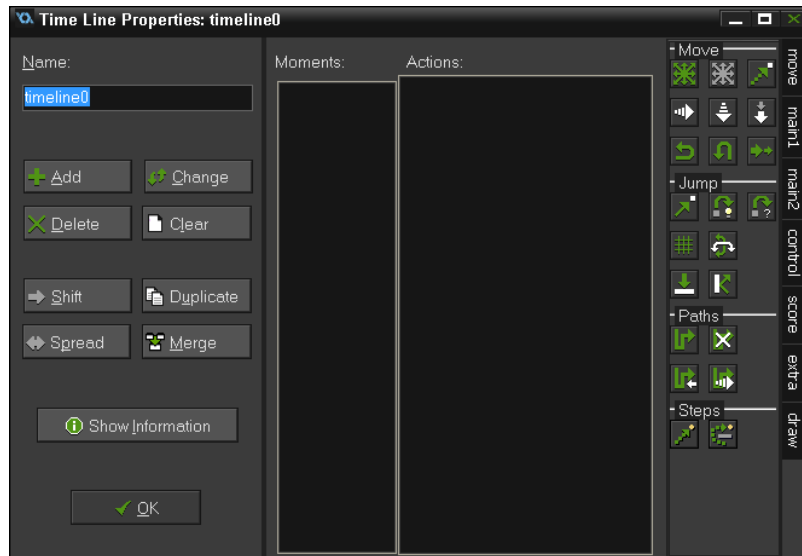


Figure 6-1: The Time Line Properties window.

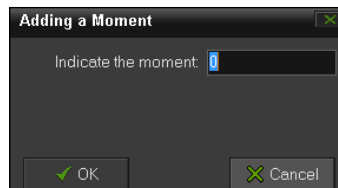


Figure 6-2: The Adding a Moment dialog box.

4. In the Indicate the Moment field, leave the default value of 0.
5. Click OK to save and close the Adding a Moment dialog box.

Step 0 appears in the Moments column of the Time Lines Properties window.



6. From the main1 tab of the Time Lines Properties, drag and drop a Create Instance Action into the Actions section.

The Create Instance Action properties window appears (see Figure 6-3).

7. In the Applies To section, select the Self radio button.
8. From the Object drop-down list, select the Object you want, such as enemy_plane.

The name of the Object appears in the Object field.

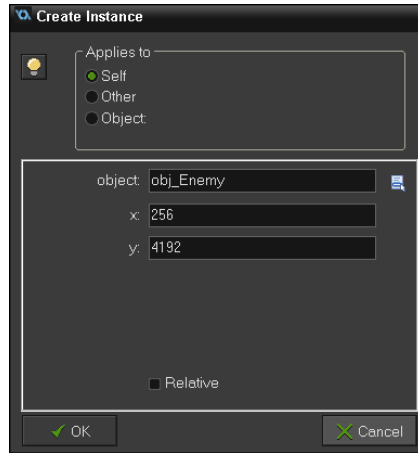


Figure 6-3: The Create Instance Action properties window.

9. In the X field, type 256.

You can get the X and Y coordinates from the Room, which you create next.

10. In the Y field, type 4192.

11. Leave the Relative check box deselected.

12. Click OK to save and close the Create Instance dialog box.

The Action is listed in the Actions section of the Time Line Properties window (see Figure 6-4).

You can have more than one Action per Moment if you want more than one thing to happen at a time during your game. In the next procedure, I show you how to create a second Moment in the Time Line that has more than a single Action.

To add more Moments in the Time Line, to trigger additional enemy planes to appear during the game, follow these steps:

1. In the Time Line Properties window, click Add.

The Adding a Moment dialog box appears.

2. In the Indicate a Moment field, type 50.

It takes approximately 100 steps in the game for an enemy plane to travel down the Room (based on the Scrolling Shooter tutorial).

3. Click OK to save and close the Adding a Moment dialog box.

The new Moment appears in the Moments section of the Time Lines Properties window.



4. From the main1 tab of the Time Lines Properties, drag and drop a Create Instance Action to the Actions section.

The Create Instance Action properties window appears.

5. In the Applies To section, select the Self radio button.

6. From the Object drop-down list, select the Object you want, such as, enemy_plane_2.

Try placing different types of enemies in the game, such as enemies that don't shoot, those that do shoot, and others that can shoot bullets that track the player. You can find examples of this in Chapter 5.

The name of the Object appears in the Object field.

7. In the X field, type 128.

You can get the X and Y coordinates from the Room, which you create next.

8. In the Y field, type 4000.

9. Leave the Relative check box deselected.

10. Click OK to save and close the Create Instance Action properties window.

The Create Instance Action appears in the Actions section of the Time Lines Properties window.



11. From the main1 tab, drag and drop a Create Instance Action into the Actions section of the Time Lines Properties window.

The Create Instance Action properties window appears.

12. In the Applies To section, select the Self radio button.

13. From the Object drop-down list, choose, for example, enemy_plane_1.

The name of the Object appears in the Object field.

14. In the X field, type 544.

You can get the X and Y coordinates from the Room, which you create next.

15. In the Y field, type 4000.

16. Leave the Relative check box deselected.

17. Click OK to save and close the Create Instance Action properties window.

The Create Instance Action appears in the Actions section of the Time Lines Properties window.

Now you should have two Moments defined in your Time Line: one Moment at Step 0, which creates an Instance of a single enemy plane, and a second Moment at Step 50, which creates two enemy planes flying on either side of the Room (see Figure 6-4). Before you can test the Time Line, you need to create a new Room and an Object for the Time Line.

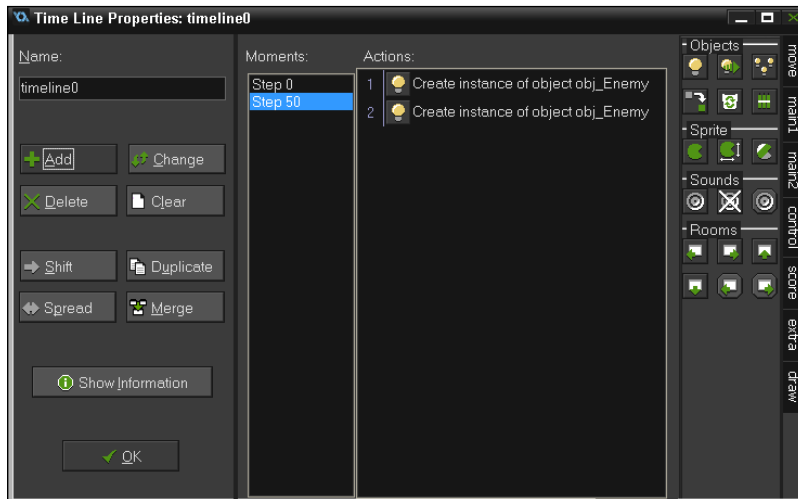


Figure 6-4: The Time Line should have two Moments, each with Actions.

Creating a Time Line Object

In this chapter, I show you how to build a Time Line that you then place in a Room. You can think of the levels in your game as a different Room that contains a different Time Line. In that sense, you can create multiple Time Lines within multiple Rooms that the player can transition to as they complete each level. I show you how to do the transitioning later, but for now, I show you how to create a new Room and an Object for a Time Line.



When making a game with multiple rooms, GameMaker makes it easy for you to duplicate a Room and then remove all Instances from that duplicated Room so that you can start fresh, but with the same Room parameters. To remove all Instances from a Room, choose the icon that looks like a white rectangle from the icon menu while in the Room Properties window, as shown in Figure 6-5.

Clear All Instances from the Room

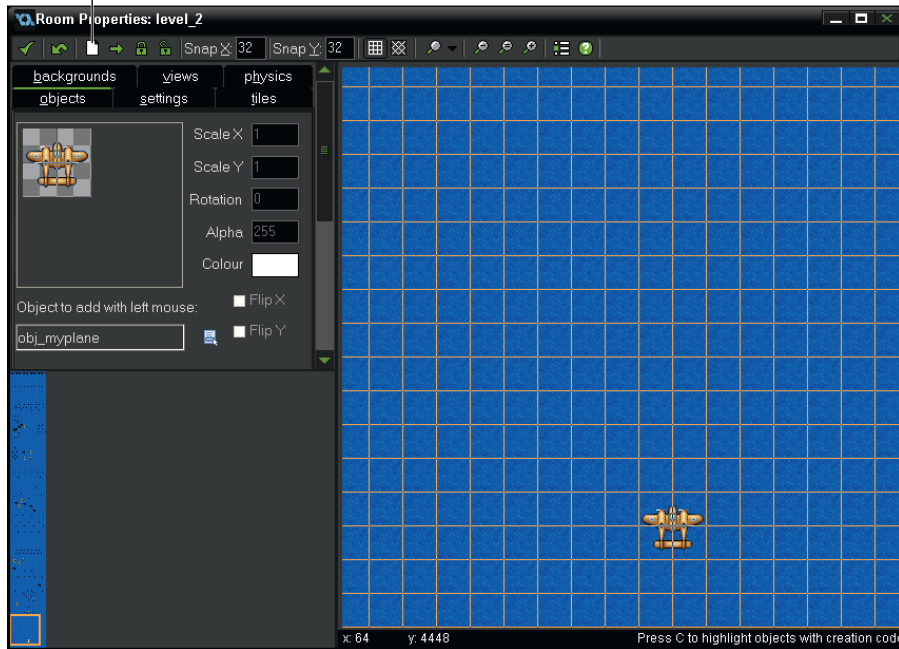


Figure 6-5: The Clear All Instances from the Room icon in the Room Properties window.

To create an Object for the Time Line, which you'll place in the Room, follow these steps:

1. **From the main menu, choose Resources⇨Create Object.**

The Object Properties window appears.

2. **In the Name field, type in, say, obj_timeline.**

3. **Choose Add Event⇨Create.**

A Create Event appears in the Events section.



4. **From the main2 tab, drag and drop a Set Time Line Action into the Actions section.**

The Set Time Line Action properties window appears (see Figure 6-6).

5. **In the Applies To section, select the Self radio button.**
6. **From the Time Line drop-down list, select, say, TimeLine0.**
7. **In the Position field, leave the default at 0.**

8. From the **Start** drop-down list, choose **Start Immediately**.
9. From the **Loop** drop-down list, choose **Don't Loop**.
10. Choose **OK** to save and close.

The Set Time Line Action properties window is saved.

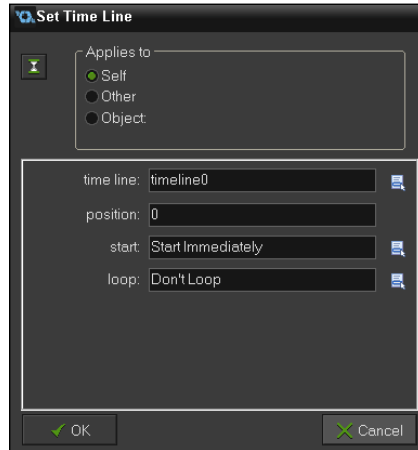


Figure 6-6: The Set Time Line Action properties window.

You've created an Object for the Time Line you created earlier. You can place an Instance of this Object in a Room, so that your Time Line is triggered during the game. In the next section, I show you how to create a new Room in which you can place your Time Line Object.

Creating the Time Line Room

To get your Time Line to trigger during a game, you need to place an Instance of a Time Line Object (see the preceding section) into a Room. To create a Room for a Time Line Object, follow these steps:

1. From the main menu, choose **Resources** → **Create Room**.

The Room Properties window appears.

2. Select the **Settings** tab, and in the **Name** field, type a name for the Room, such as **Level_One** (see Figure 6-7).

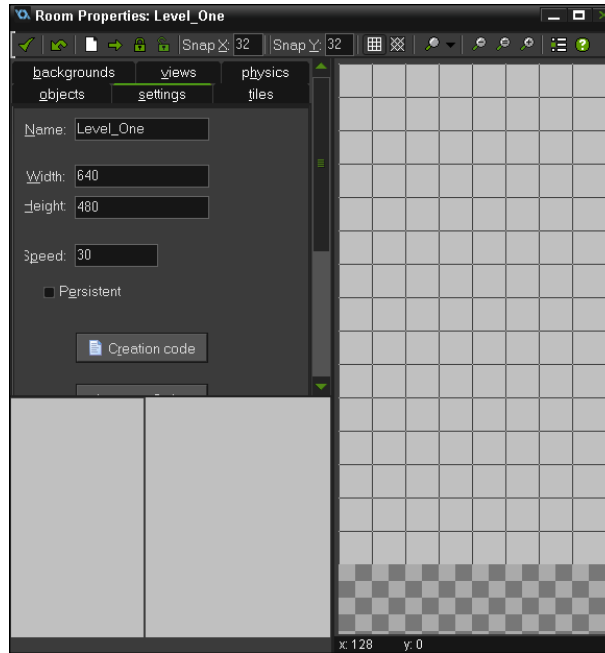


Figure 6-7: The Settings tab of the Room Properties window.

3. **Select the Objects tab.**
4. **Adjust the grid so that you see the bottom of the Room, by clicking and dragging in the bottom-left section of the Room Properties window (see Figure 6-8).**

The Room is represented by a square grid on a gray background (unless you replaced it with a background image), and you'll see a transparency pattern behind that (refer to Figure 6-8).

5. **From the left side of the Room Properties window, click the drop-down icon indicated in Figure 6-8, and select, say, `obj_timeline`.**

The name of the Object appears in the field.

6. **To place an Instance of the Object in the Room, click the grid that represents the Room.**

A blue circle with a question mark appears in the Room where you clicked (refer to Figure 6-8). The blue circle is shown because you didn't load a Sprite to the Object. This blue circle doesn't actually appear in the game.

7. Click the green check mark on the icon bar (refer to Figure 6-8).

The Room Properties window saves and closes.

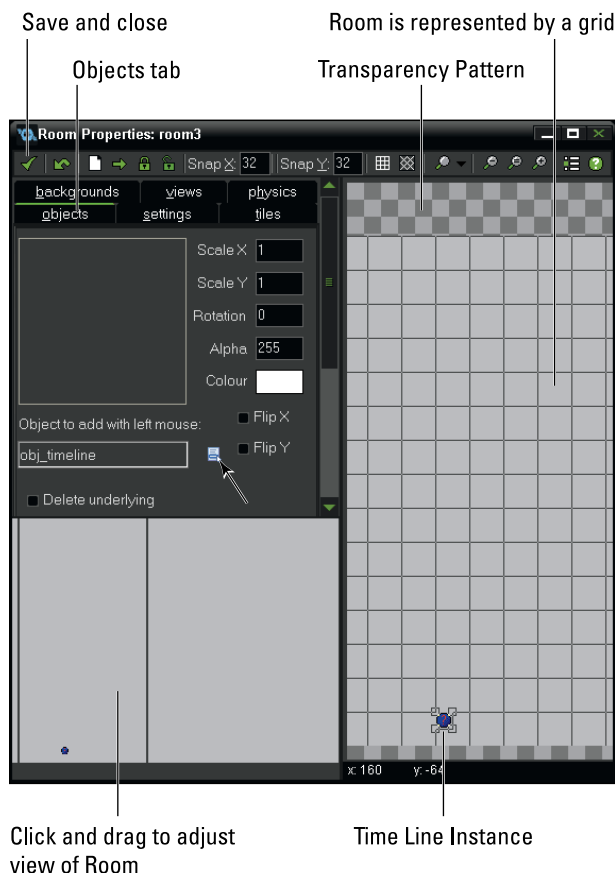


Figure 6-8: The Objects tab of the Room Properties window.

Now is an excellent time to test your game. You should see the enemy planes appear as you configured the Time Line.

At this point, you can go back to your Time Line and add more Moments with additional Objects (enemy planes and islands). Now that you have a Room set up, you can reference the X and Y coordinates to help you plan formations and so forth for the enemy planes.

If you decide you want to change when a Moment happens — say, you want more enemy planes to show up sooner — you can easily do that by changing the Step number. To change the Step number of a Moment, follow these steps:

1. **Select the Moment that you want to change — for example, you may want the Moment to happen sooner in your game.**

The name of the Moment appears with a blue background.

2. **Click Change (see Figure 6-9).**

The Changing a Moment dialog box appears.

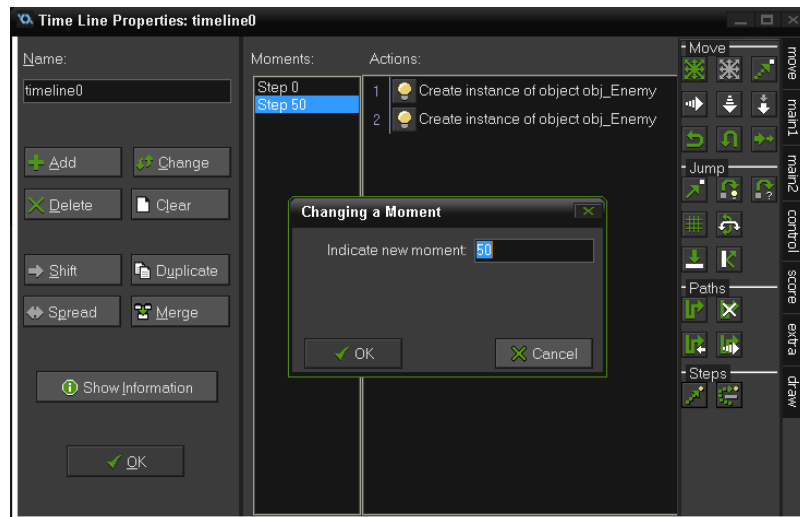


Figure 6-9: Click Change.

3. **In the Indicate New Moment field, type, say, 25.**

Now the Actions defined for this Moment will trigger twice as quickly as when it was set to 50.

4. **Click OK to save and close the Changing a Moment dialog box.**

The updated Step number appears in the Moments section of the Time Line Properties window.



If you're making the Scrolling Shooter game from the in-software tutorial, don't forget to add some Instances of the different islands to the Time Line as well.

To make new Actions for Moments quickly, follow these steps:

1. **From the Resource tree, in the Time Line folder, double-click a Time Line.**

The Time Line Properties window appears.

2. **You should already have a few Moments and Actions configured; if not, see “Creating a Time Line,” earlier in this chapter, for instructions on how to do so.**
3. **Select a Moment from the Moments section of the Time Line Properties window.**

The Actions for that Moment appear in the Actions section.

4. **Right-click an Action, such as a Create Instance Action, and choose Copy (see Figure 6-10).**

That Create Instance Action is now copied to your Clipboard.

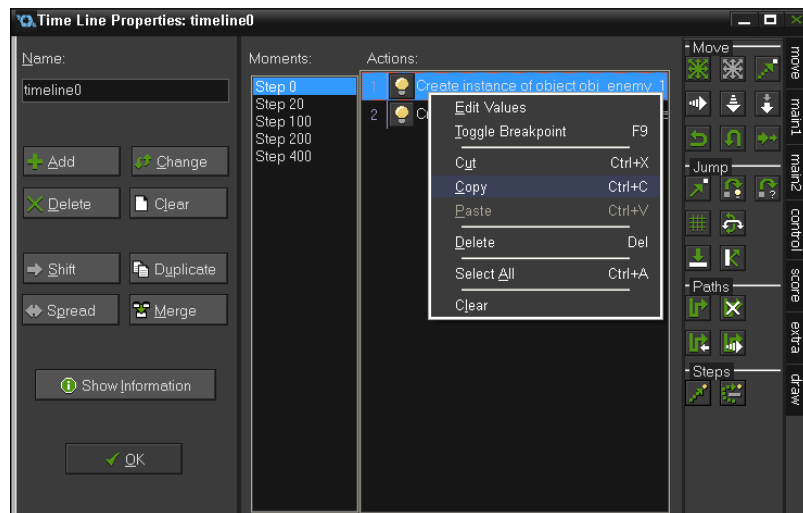


Figure 6-10: Choose Copy.

5. **In an empty part of the Actions section, right-click and choose Paste.**

The Action, such as the Create Instance Action, you copied earlier appears.

6. **Open the pasted Action by double-clicking.**

The Create Instance Action properties window, for example, appears.

7. From the Object drop-down list, choose, say, an island Object rather than an enemy Object.

The name of the Object you chose appears in the field.

8. In the X field, change the value slightly — say, from 300 to 250.
9. In the Y field, change the value slightly — say, from 4400 to 4300.
10. Click OK to save and close the Create Instance Action properties window (or whichever type of Action you copied and pasted).

In this way, you can quickly place Instances of Objects (such as the islands from the Scrolling Shooter tutorial) into the game. Plus, by only slightly changing the values of the coordinates, you make sure that the islands won't appear directly beneath the planes. Unless, of course, you want to place the islands directly beneath the planes.

If you think that Views are a much easier way to place the enemy planes in the Room than using Time Lines, you aren't necessarily wrong. However, Time Lines allow you to do much more than just create Instances. You can also end the game through a Time Line, or slow time down or speed it up. You can even pause a Time Line until a certain requirement happens. In the next section, I show you how to use the Time Line to allow the player to progress to the next Room.

Transitioning to the Next Room with Time Lines

Before you add an entire fleet of enemy planes to the Room, try adding a Moment that transitions the player to the next Room. To use the Time Line to transition to a new Room, follow these steps:

1. With the Time Lines Properties window open, click Add.

The Adding a Moment dialog box appears.

2. In the Indicate the Moment field, type 300 (or at whichever Step of the game you want the player to transition).

3. Click OK to save and close the dialog box.

A new Moment appears in the Moments section of the Time Line Properties window.



4. Drag and drop a Next Room Action from the main1 tab into the Actions section of the Time Line Properties window.

The Next Room dialog box appears.

5. Click OK to save and close the Next Room dialog box.

The Next Room Action appears in the Actions section of the Time Line Properties window.

Now when you test your game, when Moment 300 happens, the game smoothly transitions to the next Room, and it's transparent to the player. If your game is set up like mine, the player starts playing the Room with a View (see Chapter 5), which then progresses to the random generator Room, which I made from following the in-software tutorial, Scrolling Shooter.

Adding a Boss Battle to a Time Line

Boss battles typically occur at the end of a level or Room. A boss battle usually involves the player taking on a stronger-than-normal enemy. In this section, I show you how to add a boss battle in your Time Line for your Room. I was inspired to create this boss battle after I finished the in-software Scrolling Shooter tutorial.

In my boss battle, which I encourage you to improve upon, the boss is represented by the battleship image (see Figure 6-11) that you can find in the Assets folder for the Scrolling Shooter tutorial. In the Assets folder (which is created when you install the tutorial), there is an image called 1945_sprites. You can edit that image to create a new image of the Sprite that you want to use.



Figure 6-11: The battleship Sprite for a boss battle.



If you can't find the Assets folder after you've saved your tutorial, start a brand-new tutorial. Then you can find the Assets folder by choosing Help⇨Open Project in Explorer.

The boss appears near the end of the Time Line and moves horizontally across the room instead of vertically. If it reaches the right edge of the Room before the player can destroy it, the boss reverses directions. The boss can take five hits from the player before it's destroyed, and then the player earns 100 points. The boss shoots guided (tracking) missiles (see Chapter 5).

You create the boss in three main parts. First, you create the Sprite. Then you create the boss Object. Finally, you add the boss to the Time Line. The boss Object includes a Create Event, a Step Event, and a Collision Event.

Creating the boss Sprite

To create the boss Sprite, follow these steps:

1. **With your boss battle image ready, choose Resources → Create Sprite from the main menu.**

The Sprite Properties window appears (see Figure 6-12).

2. **In the Name field, type, spr_boss, or something similar.**
3. **Click Load Sprite.**

You can then navigate to where you saved the image on your PC.

4. **Navigate to your saved image, select the image, and click Open.**

The image appears in the Sprite Editor.

5. **In the X field, type 126 (use a value that places the crosshair close to the right side of the image — refer to Figure 6-12).**

Setting the Origin in this way, so that the X axis is at the far right of the image, will help turn the battleship around, because the Origin of the Sprite is at the far right. When the Origin of the image reaches the edge of the Room, the battleship turns around (you configure that in the boss Object). If the Origin was in the center of the Sprite, half of the battleship would go off screen before turning around.

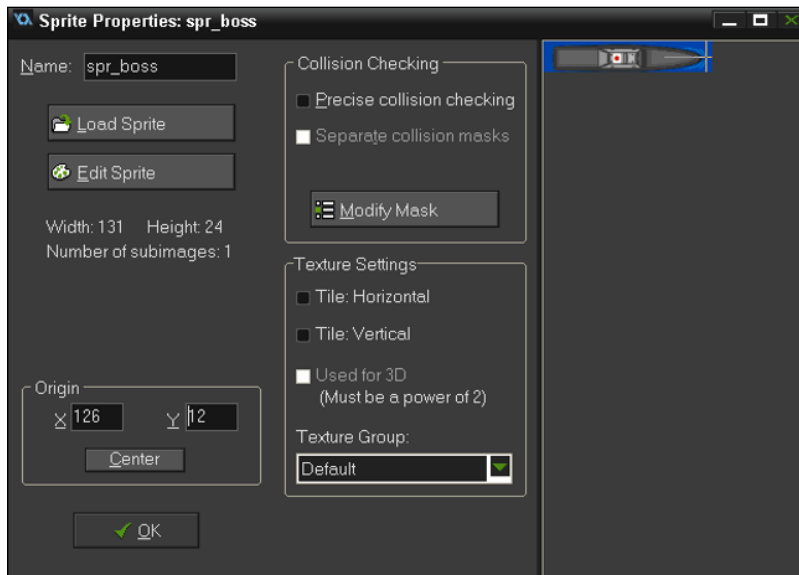


Figure 6-12: The Sprite Properties window.

6. In the Y field, type 12 (use a value that places the crosshair close to the middle of the image — refer to Figure 6-12).
7. Click OK to save and close the Sprite Properties window.



You've configured the boss Sprite.

You can click and drag on the axis points to adjust the Origin.

With the boss Sprite set up, it's time to create an Object for the boss.

Creating the boss Object

In this section, I show you how to create the boss Object, starting with the Create Event. Within the Create Event, you give the boss motion and create a variable to give the boss health (so that it takes more than one shot to destroy the Instance).

To configure the Create Event, follow these steps:

1. From the main menu, choose **Resources** → **Create Object**.

The Object Properties window appears (see Figure 6-13).

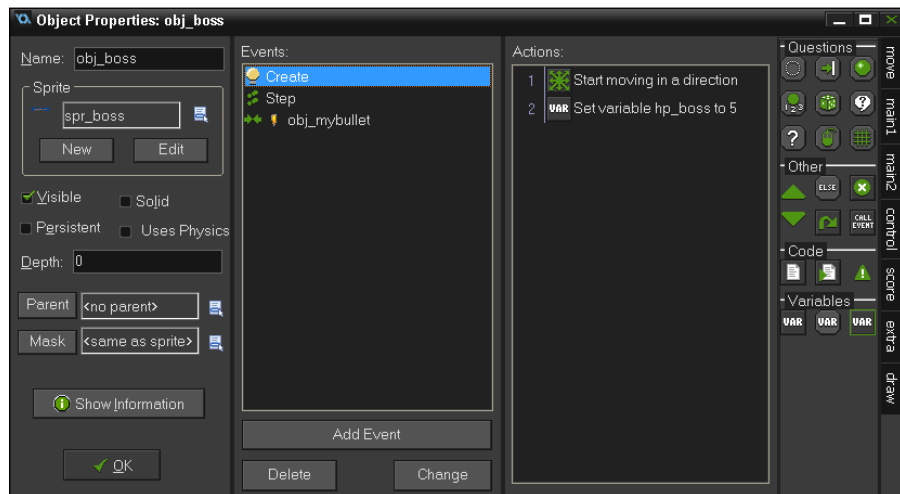


Figure 6-13: The Object Properties window.

2. In the Name field, type `obj_boss`, or something similar.
3. From the Sprite section, click to open the Sprite drop-down list and select the boss Sprite.

A thumbnail of the selected Sprite appears in the Sprite section, and the name of the Sprite appears in the field.

4. Choose Add Event → Create.

The Create Event appears in the Events section of the Object Properties window.



5. From the Move tab, drag and drop a Move Fixed Action into the Actions section.

The Move Fixed Action properties window appears (see Figure 6-14).

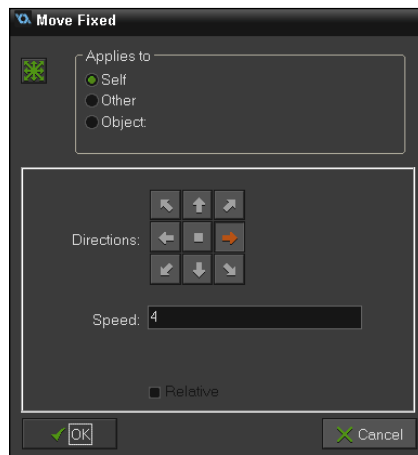


Figure 6-14: The Move Fixed Action properties window.

6. In the Applies To section, select the Self radio button.

7. For the Directions, choose the right arrow.

The Object will move to the right during game play.

8. In the Speed field, type 4.

This determines how fast the Object moves during the game.

9. Click OK to save and close the Move Fixed dialog box.



10. Drag and drop a Set Variable Action from the Control tab into the Actions section.

The Set Variable Action properties window appears (see Figure 6-15).

11. In the Applies To section, select the Self radio button.

12. In the Variable field, type hp_boss to create a variable that represents the health of the boss Object.

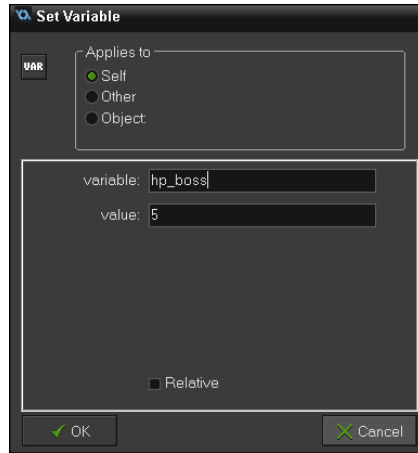


Figure 6-15: The Set Variable Action properties window.

13. In the Value field, type, say, 5.

This gives the boss a health of 5, meaning it will take five hits to destroy an Instance of this Object — that is, if you set up the Collision Effect to cause -1 damage. I get to this later.

14. Click OK to save and close the Set Variable dialog box.

This completes the Create Event, which should look like Figure 6-16.

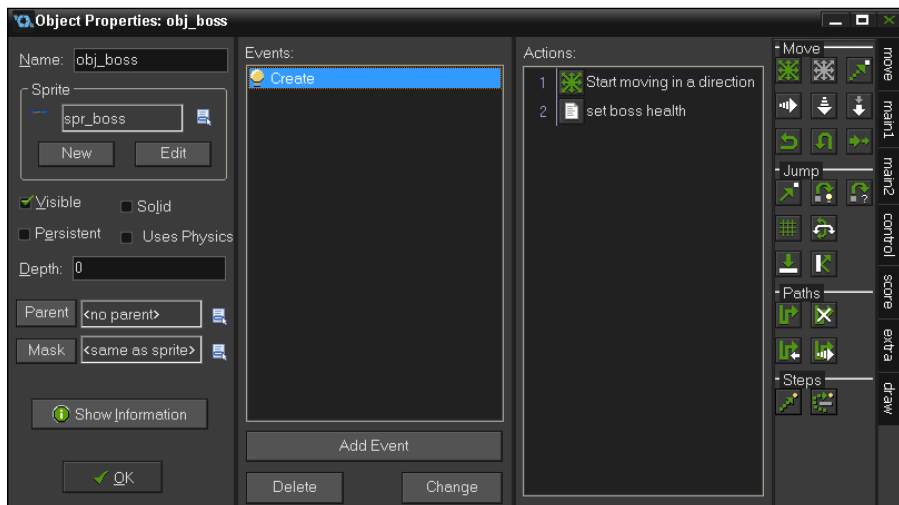


Figure 6-16: The Create Event for the boss Object.

Next, you create the Step Event. This involves setting Test Variables to check for where the Instance of the Object is in the Room, adjusting the direction of the Object, transforming the Sprite, arming the Object, and setting a block of Actions for when the boss loses all health.

To configure the Step Event, follow these steps:

1. Choose Add Event⇨Step⇨Step.

A Step Event appears in the Events section.



2. Select the Control tab, and drag and drop a Test Variable Action into the Actions section.

The Test Variable Action properties window appears.

3. In the Applies To section, select the Self radio button.

4. In the Variable field, type x.

This tests to see when the boss reaches the edge of the Room as denoted by the X axis.

5. In the Value field, type room_width.

The value of room_width equals the width of the Room. So, you can test when the Object's X variable has a value greater than or equal to the X axis of the Room. This will test to see when the Object reaches the edge of the Room.



By using room_width, you're directly referencing the width of the Room, regardless of the value. If you typed the actual value of the width of the Room — say, 544 — and then later you change the width of the Room to 600, you have to go back and update the value in the Action. But by using room_width, that isn't an issue.

6. From the Operation drop-down list, choose Greater Than or Equal To.

7. Click OK to save and close the Test Variable Action properties window.

Now you need to configure the Move Fixed Action for the Step Event:



1. Select the Move tab, and drag and drop a Move Fixed Action into the Actions section.

The Moved Fixed Action properties window appears.

2. For the Directions, select the left arrow.

3. In the Speed field, type 4.

4. Click OK to save and close the Move Fixed Action properties window.

Between the Move Fixed Action and the preceding Test Variable Action, you just told GameMaker to change the direction of the Instance from moving right to moving left when it reaches the edge of the Room.



5. **Select the Control tab, and drag and drop a Test Variable Action into the Actions section.**

The Test Variable Action properties window appears.

6. **In the Applies To section, select the Self radio button.**

7. **In the Variable field, type x.**

This tests to see when the boss reaches the edge of the Room, as denoted by the X axis.

8. **In the Value field, type 0.**

The value of 0 equals the left edge the Room.

9. **From the Operation drop-down list, choose Less Than or Equal To.**

10. **Click OK to save and close the Test Variable Action properties window.**

Now to configure the Move Fixed Action for the Step Event:



1. **Select the Move tab, and drag and drop a Move Fixed Action into the Actions section.**

The Moved Fixed Action properties window appears.

2. **For the Directions, select the right arrow.**

3. **In the Speed field, type 4.**

4. **Click OK to save and close the Move Fixed Action properties window.**

Between the Move Fixed Action and the preceding Test Variable Action, you just told GameMaker to change the direction of the Instance from moving left to moving right when it reaches the edge of the Room.

Now to configure the Transform Sprite Action for the Step Event:



1. **From the Main1 tab, drag and drop a Transform Sprite Action into the Actions section.**

The Transform Sprite Action properties window appears.

This Action forces the Sprite to face the direction it's moving, so if the boss reaches the edge of the Room and needs to turn around, the front of the ship will face to the right instead of the left.

2. **In the Applies To section, select the Self radio button.**

3. **In the Xscale field, type 1.**

4. **In the Yscale field, type 1.**

5. In the Angle field, type direction.
6. From the Mirror drop-down list, choose No Mirroring.
7. Click OK to save and close the Transform Sprite Action properties window.
8. Select the Control tab, and drag and drop a Test Chance Action into the Actions section.



The Test Chance Action Properties window appears.

9. In the Sides field, type 30.

This determines how often the boss will shoot at the player. In this case, the boss has a 1 in 30 chance (based on a 30-sided dice roll) of performing the next Action for every step of the game.

10. Click OK to save and close the Test Chance Action properties window.

Now to configure the Create Instance Action for the Step Event:



1. Select the Main1 tab, and drag and drop a Create Instance Action into the Actions section.

The Create Instance Action properties window appears.

2. In the Applies To section, select the Self radio button.
3. From the Object drop-down list, choose obj_enemy_bullet (or whatever Object you want created).

In this case, the Object will fire a bullet 1 chance out of 30 for every Step, as determined in the Test Chance Action you previously set.

4. In the X field, type 0.

In this Action, you're going to use Relative and not a set coordinate.

5. In the Y field, type 0.

In this Action, you're going to use Relative and not a set coordinate.

6. Select the Relative check box.

You want the bullet to appear from the location of the Instance — in this case, the boss battleship.

7. Click OK to save and close the Create Instance dialog box.

You're almost done with the Step Event. The next procedure shows you how to finish the Step Event by adding a Test Variable to see when the boss's health reaches 0. When that happens, the player's score should increase, the boss should be destroyed, and you can display a message declaring victory!

The following procedure is a continuation of the previous procedure. However, it does provide a great example of setting up a code block to trigger when a Test Variable tests positive. To create a code block for when the boss's health reaches 0, follow these steps:



1. **From the Control tab, drag and drop a Test Variable Action into the Actions section.**

The Test Variable Action properties window appears.

2. **In the Applies To section, select the Self radio button.**
3. **In the Variable field, type hp_boss (the variable used to set the boss's health).**
4. **In the Value field, type 0.**

You're testing for when the boss has 0 health.

5. **From the Operation drop-down list, choose Less Than or Equal To.**
6. **Click OK to save and close the Test Variable properties window.**

Now, when the boss health equals 0, the Test Variable Action will trigger the following block of Actions.



7. **Select the Control tab, and drag and drop a Start Block Action into the Actions section.**

A Start Block Action appears in the Actions section.



8. **Select the Score tab, and drag and drop a Set Score Action into the Actions section.**

The Set Score properties window appears.

9. **In the new score field, type, say, 100.**

The player will earn 100 points when the boss's health reaches 0.

10. **Toggle Relative so that the box is selected.**

You want the player's score to increase by 100, and not remain at 100 each time the Set Score Action is triggered.

11. **Choose OK.**

The Set Score Action properties window is saved and closed.



12. **From the main1 tab, drag and drop a Destroy Instance Action into the Actions section.**

The Destroy Instance properties window appears.

13. **In the Applies To section, select the Self radio button.**

14. Click OK.

The Destroy Instance Action will destroy the Instance of the boss Object when the health reaches 0.

**15. Select the Main2 tab, and drag and drop a Display Message Action into the Actions section.**

The Display Message dialog box appears.

16. In the Message field, type Boss Defeated, or something similar.**17. Click OK to save and close the Display Message dialog box.****18. Select the Control tab, and drag and drop an End Block Action to the Actions section.**

An End Block Action appears in the Actions section. Now, all the Actions within the code block will be triggered when the Test Variable Action checks that the boss's health has reached 0. The Actions triggered include increasing the player's score, destroying the Instance of the boss Object, and displaying a message.

That concludes the Step Event, which should look like Figure 6-17.

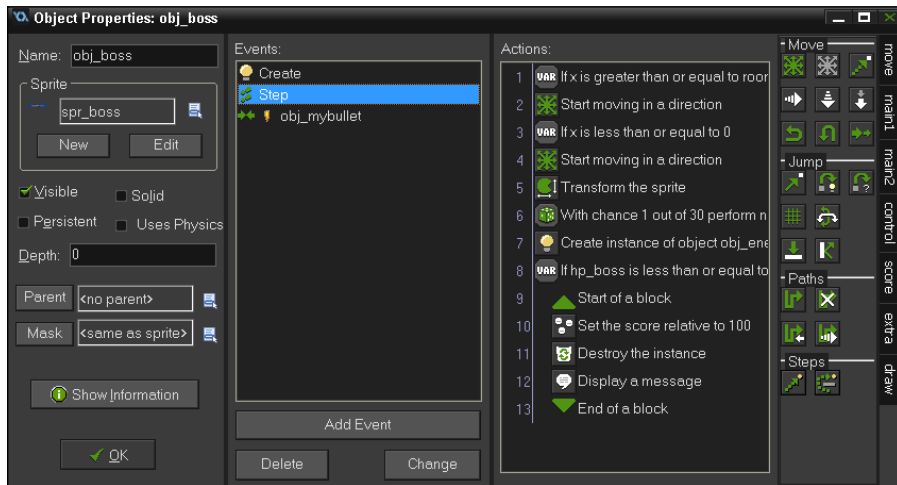


Figure 6-17: The Step Event for obj_boss.

Next, you create the Collision Event. This involves destroying the Instance, creating the explosion Sprite, adding some code for reducing the boss's health,

and increasing the player's score if the boss is destroyed. To configure the Collision Event, follow these steps:

1. **From the Object Properties window, choose Add Event→Collision→obj_mybullet.**

A Collision Event for when the player's bullet hits the boss appears in the Events section.



2. **Select the main1 tab, and drag and drop a Destroy Instance Action into the Actions section.**

The Destroy Instance dialog box appears.

3. **In the Applies To section, select the Other radio button.**

This destroys the Instance of the player's bullet that hit the boss (the Instance that caused the collision). Any other Instances of the player's bullet will remain in the Room.

4. **Click OK to save and close the Destroy Instance dialog box.**



5. **Select the Main1 tab, and drag and drop a Create Instance Action into the Actions section.**

The Create Instance properties window appears.

6. **In the Applies To section, select the Self radio button.**

7. **From the Object drop-down list, choose obj_explosion.**

Choose an Object that represents an explosion.

8. **In the X and Y fields, type 0.**

You're going to use Relative.

9. **Select the Relative check box.**

The explosion appears where the bullet hits the boss.

10. **Click OK to save and close the Create Instance properties window.**



11. **Select the Control tab, and drag and drop a Set Variable Action into the Actions section.**

The Set Variable Action properties window appears.

12. **In the Applies To section, select the Self radio button.**

13. **In the Variable field, type hp_boss (the variable used to set the boss's health).**

14. **In the Value field, type -1.**

The Instance of the boss will lose 1 health for every bullet collision.

15. Select the Relative check box.

You want the boss to lose only 1 health for each collision with a bullet.

16. Click OK to save and close the Set Variable Action properties window.

That concludes configuring the boss Object. The Collision Event should look like Figure 6-18. Next, you need to add the boss to the Time Line.

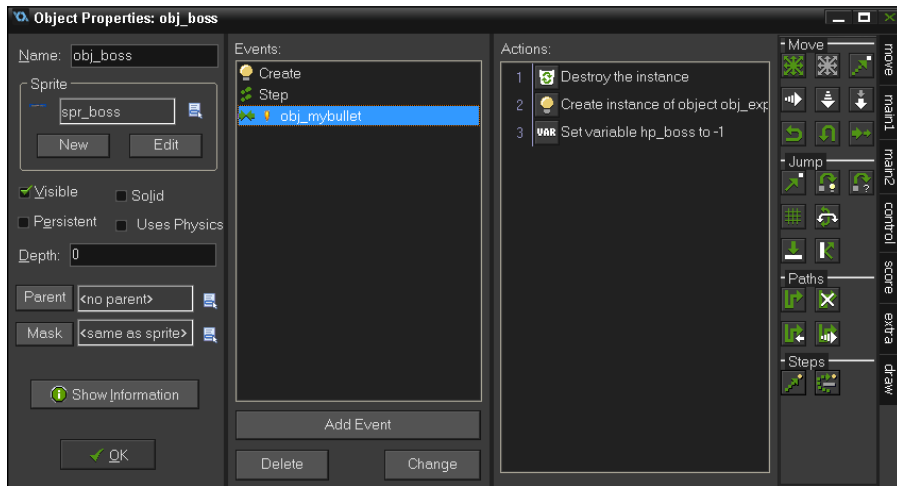


Figure 6-18: The Collision Event for obj_boss.

Adding the Boss to the Time Line

To add the boss Object to a Time Line, follow these steps:

- 1. With a Time Line open (see “Creating a Time Line,” earlier in this chapter), click Add.**

The Adding a Moment dialog box appears.

- 2. In the Indicate a Moment field, type 200 (or whatever Step in the game you want the boss Object to appear).**

- 3. Click OK to save and close the Adding a Moment dialog box.**



- 4. Select the Main1 tab, and drag and drop a Create Instance Action into the Actions section.**

The Create Instance properties window appears.

- 5. In the Applies To section, select the Self radio button.**

6. From the Object drop-down list, choose, say, `obj_boss`.
7. In the X field, type, say, 10.
8. In the Y field, type, say, 3800.
9. Click OK to save and close the Create Instance properties window.

You just added the boss battle to your game!

Congratulations! You've created a boss battle in your Scrolling Shooter game using a Time Line. Now is the time to dig into the details and refine it to meet your satisfaction. For example, what if the player doesn't destroy the boss but instead flies past it? Then you could add another Moment in the Time Line — say, at position 300 — to either restart the room (by using the Restart Room Action) or repeat the battle (by using the Time Line Position Action — in the Position field, type **200**, or wherever the boss battle is set to begin). Those are just a couple ideas to get your creative juices flowing.

Part III

Creating Special Effects



Find out how to add an Effect Action to your GameMaker: Studio game in an article at www.dummies.com/extras/gamemakerstudio.

In this part . . .

- ✓ Make special effects the drag-and-drop way.
- ✓ Code special effects.
- ✓ Use Surfaces for things like lighting and shadows.
- ✓ Apply scaling and resolution code to your games.

Using Actions to Create Special Effects with Particles

In This Chapter

- ▶ Creating buttons to trigger particles
- ▶ Using the Create Effect Action
- ▶ Creating several types of fireworks effects
- ▶ Coding simple and complex particles
- ▶ Coding emitters

In GameMaker, you can make some very special effects, such as explosions and fireworks. You create these effects through the use of *particles*. Particles are fast graphics resources that are generated based on random values. You can create special effects for your game by

- ✓ Using the Create Effect Action
- ✓ Using the 13 different Actions in the Particles section on the Extra tab
- ✓ Writing your own code
- ✓ Creating an Emitter Burst or an Emitter Stream

In this chapter, I walk you through these various methods by using the Particles tutorial that is included with GameMaker. For the coding aspect, you can go ahead and skip to the next chapter, because this chapter primarily discusses using the drag-and-drop Actions. However, I do show you how to create a button to trigger these effects in this chapter, which you can also use in Chapter 8.

As with the other chapters in this book, you can find the assets for this tutorial by choosing Help⇨Open Project in Explorer⇨Assets, from the main menu. Also, when you're done with this tutorial, you can install the demo for using particles to compare your work.

Figure 7-1 shows an example of a special effect that you can make by following along with this chapter.



Figure 7-1: An example of using Particles.

To get started with particles, I show you how to create a fireworks display using the Create Effect Action. First, I describe how to create a button to trigger the effect.

Coding a Button to Trigger the Particles

Before you get started with creating particles, I provide a procedure from the in-software Particles tutorial on how to create a button that the player can use to control the particles. Basically, when the player presses the button, a fireworks effect will appear. You duplicate this button and use it to trigger the other methods discussed in this chapter and Chapter 8, on how to create special effects.

To create a button for triggering the particles, follow these steps:

1. **From the Resource tree, right-click Objects and then choose Create Object.**

The Object Properties window appears.

2. **In the Name field, type `obj_button_builtin`.**
3. **From the Sprite drop-down list, select `spr_button_builtin` (available from the tutorial).**

A thumbnail of the image appears in the Sprite section and the field is populated with the name of the image.

4. Choose Add Event⇨Create.

The Create Event appears in the Events section.

**5. Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

6. In the code window, type the following code:

```
image_speed = 0;  
image_index = 0;
```

This code stops the button from animating (if you opened the Sprite, you may have noticed that the assigned Sprite is made up of three different sub-images) and tells GameMaker to display only the first indexed sub-image.



GameMaker starts counting at 0, just as in programming. Each sub-image represents a different state of the button — image 0 is the unselected state of the button, image 1 shows the button when the player hovers over it with the mouse, and image 2 is a slightly smaller version of the image to represent when the button is actually clicked on by the player. The final result is a button that reacts when the player either hovers over it or clicks it. (You write the code for that in Step 14.)

7. Click the green check mark from the icon menu.

The code window is saved and closed.

8. Choose Add Event⇨Alarm⇨Alarm 0.

The Alarm 0 Event appears in the Events section.

9. Drag and drop an Execute Code Action from the Control tab to the Actions section.

The code window appears.

10. In the code window, type the following code:

```
image_index = 0;
```

You just set an Alarm that when triggered, displays the first sub-image.

11. Click the green check mark from the icon menu.

The code window is saved and closed.

12. Click Add Event⇨Step⇨Step.

The Step Event appears in the Events section.

13. Drag and drop an Execute Code Action from the Control tab to the Actions section.

The code window appears.

14. In the code window, type the following code (see Figure 7-2):

```
if instance_position(mouse_x, mouse_y, id)
{
    if image_index != 2
    {
        image_index = 1;
        if mouse_check_button_pressed(mb_left)
        {
            alarm[0] = room_speed / 2;
            image_index = 2;
            instance_create(random(room_width), 400, obj_button_firework);
        }
    }
}
else
{
    if image_index != 2
    {
        image_index = 0;
    }
}
```

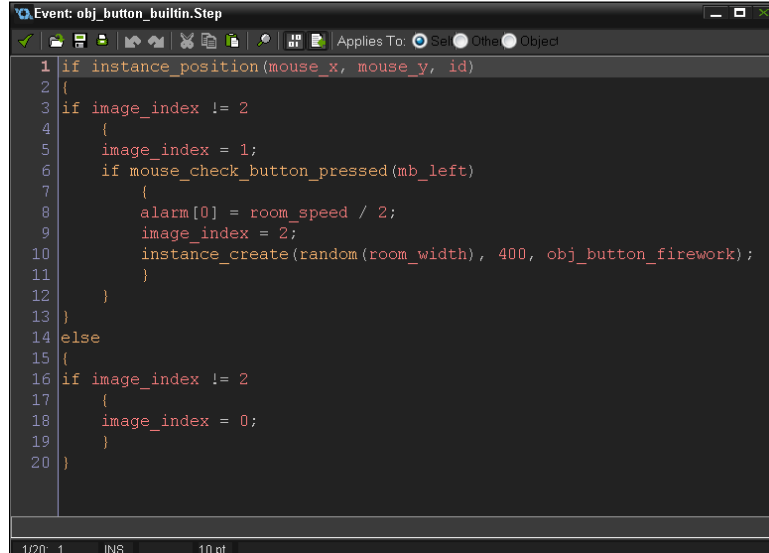


Figure 7-2: The code window for the Step Event, Execute Code Action.

This code determines which sub-image is displayed, depending on the state of the mouse (is the player hovering, is the player not hovering, or is the player clicking the button). The code also tells GameMaker to check if the mouse position is over the button, to check if the mouse button has



been pressed, and if so, to create the firework Object. The player sees a hover and pressed state for the button. At the end of the code, it resets the image index back to 0 (the player is not hovering or clicking).

The preceding code creates an Instance of the firework Object, but because you haven't created this Object yet, the code window displays the Object text (obj_button_firework) in white. After you've created the object, the Object text in the code window appears in red, as shown in Figure 7-2.

15. Click the green check mark.

The code window is saved and closed.

16. Click OK from the Object Properties window.

The Object Properties window is saved and closed.

Using the Create Effect Action

In the preceding procedure, you created a button that will trigger a fireworks effect. In this section, I show you how to create the effect itself by using the Create Effect Action, which does require some coding by hand. In the following procedure, you create the firework Object.



Each Object can have its own set of sequential Alarms. For example, you created Alarm[0] for the button Object, and in the next procedure you create Alarm[0] and Alarm[1] for the firework Object.

To create the firework Object, follow these steps:

1. From the Resource tree, right-click Objects and then choose Create Object.

The Object Properties window appears.

2. In the Name field, type obj_button_firework.

3. Choose Add Event → Create.

The Create Event appears in the Events section.

4. Drag and drop an Execute Code Action from the Control tab to the Actions section.

A code window appears.

5. In the code window, type the following code:

```
direction = point_direction(x, y, room_width / 2, 100);
speed = 15 + random(5);
gravity = 0.5;
gravity_direction = 270;
alarm[0] = 1;
alarm[1] = room_speed
```

6. Click the green check mark from the icon menu.

The code window is saved and closed.

The first line of this code sets the direction of the firework so that it moves toward the center of the Room, it has speed, and there's also gravity so that it appears as if the firework's sparkles start to fall naturally. This code also has two Alarms — one to create a smoke trail and the other to generate the actual firework. You configure the Alarms next.

1. With `obj_button_firework` still open from the last procedure, choose `Add Event`→`Alarm`→`Alarm[0]`.

The Alarm 0 Event appears in the Events section.



2. Drag and drop a Comment Action from the Control tab into the Actions section.

The Comment window appears.

3. In the Comment field, type `Create smoke trail`.

This is a note to yourself for easy reference later on what the Alarm entails.

4. Click OK.

The Comment window is saved and closed. In the Actions section, you can now see the Comment you just wrote, as shown in Figure 7-3.

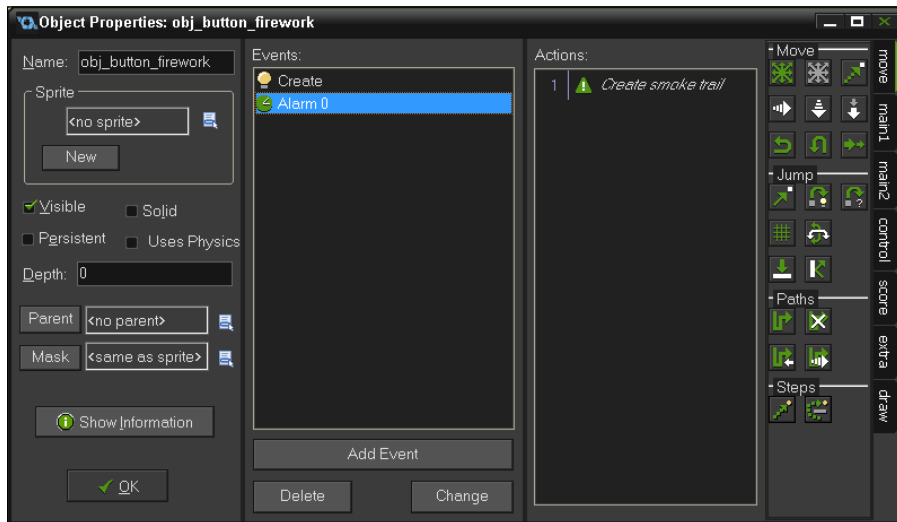


Figure 7-3: The Object Properties window with an Alarm Event and Comment Action.

Now you need to add the Create Effect Action.



1. **Drag and drop a Create Effect Action from the Draw tab into the Actions section.**

The Create Effect properties window appears.

2. **In the Applies To section, select the Self radio button.**
3. **From the Type drop-down list, select Smoke.**

Smoke appears in the Type field (see Figure 7-4).

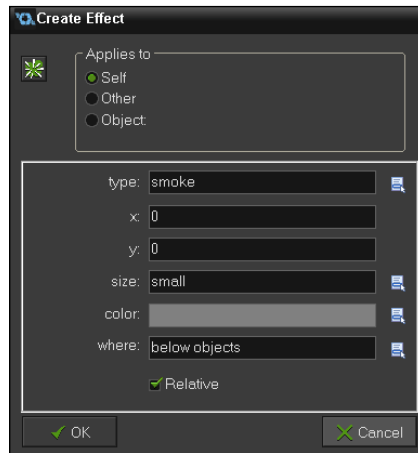


Figure 7-4: The Create Effect properties.

4. **Leave the X and Y fields at 0 (so that it appears Relative to the Instance with the Action).**
5. **From the Size drop-down list, select Small.**
Small appears in the Size field.
6. **Click the Color drop-down.**
The Color window appears.
7. **From the Color window, select a gray color, as shown in Figure 7-5.**
8. **Click OK.**

The color gray appears in the Color field.

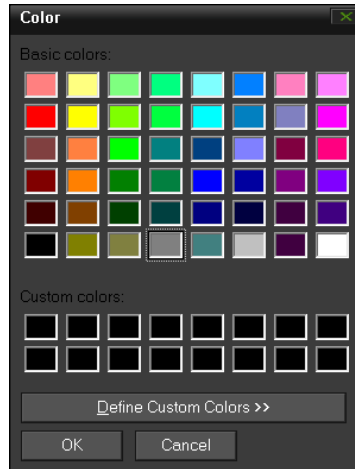


Figure 7-5: The Color window.

9. From the Where drop-down list, select Below Objects.

Below Objects means that this Object will be drawn beneath other Objects at a depth of 100,000. If you choose above, then this Object would appear over other Objects at a depth of -100,000.

10. Select the Relative check box.

Now the properties are relative to what is happening in the game.

11. Click OK.

The Create Effect properties is saved and closed.

Now you need to set an Alarm Action.



1. Drag and drop a Set Alarm Action from the Main2 tab to the Actions section.

You're going to use this Set Alarm Action to reset Alarm 0 every step of the game. When Alarm 0 is triggered (you set this up in the last procedure), a smoke particle is created, which creates a trail as the Instance moves across the Room.

The Set Alarm Action window appears.

2. In the Applies To section, select the Self radio button.

3. In the Number of Steps field, type 1.

This value determines how often the Alarm is triggered. In this case, the Alarm will be triggered once for every step of the game.

4. From the In Alarm No drop-down list, select Alarm 0.

This is the Alarm that will be triggered every step of the game.

5. Click OK.

The Set Alarm Action window is saved and closed.

Now add the second Alarm Event.

1. Choose Add Event⇨Alarm⇨Alarm[1].

The Alarm 1 Event appears in the Events section.

**2. Drag and drop a Comment Action from the Control tab to the Actions section.**

The Comment Action window appears.

3. In the Comment field, type Explode the firework.**4. Click OK.**

The Comment Action window is saved and closed. You can now see the Comment Action in the Actions section.

**5. Drag and drop a Create Effect Action from the Draw tab into the Actions section.**

A Create Effect Action window appears.

6. In the Applies To section, select the Self radio button.**7. From the Type drop-down list, select Firework.****8. Leave X and Y at 0.**

The firework will appear relative to the Instance with the Action.

9. From the Size drop-down list, select Large.**10. Click the Color drop-down.**

The Color window appears.

11. From the Color window, pick a yellow color.**12. Click OK.**

The Color window is saved and closed. The color you picked appears in the Color field.

13. From the Where drop-down list, select Below Objects.

This particle will now be drawn underneath other Instances.

14. Select the Relative check box.**15. Click OK.**

The Create Effect Action window is saved and closed.

Now add a Set Alarm Action to the Alarm[1] Event.



1. **Drag and drop a Set Alarm Action from the Main2 tab to the Actions section.**

The Set Alarm Action window appears.

2. **In the Applies To section, select the Self radio button.**
3. **In the Number of Steps field, type -1.**
4. **From the In Alarm No drop-down list, select Alarm 0.**
5. **Click OK.**

The Set Alarm Action window is saved and closed.

Now to add a second Set Alarm Action for the Alarm[1] Event.

1. **Drag and drop a Set Alarm Action from the Main2 tab to the Actions section.**

A Set Alarm Action window appears.

2. **In the Applies To section, select the Self radio button.**
3. **In the Number of Steps field, type room_speed * 3.**

You just set this Alarm to trigger three times for every step of the game.

4. **From the In Alarm No drop-down list, select Alarm 2.**

Your Alarm 1 Event should look like Figure 7-6.

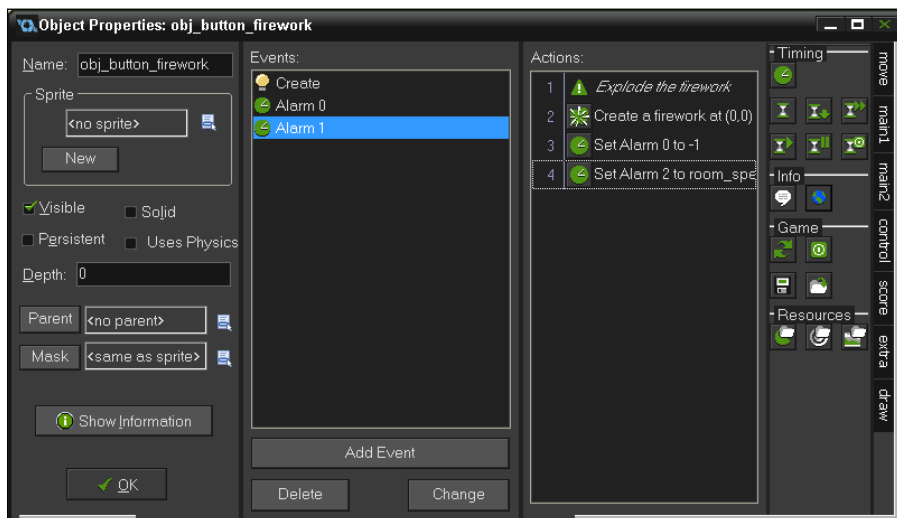


Figure 7-6: The Alarm 1 Event.

Next, you need to create an Event for Alarm[2] that destroys the Instance of the firework effect. It's important to destroy the Instance — you don't want anything in the Room that doesn't do anything because it will eventually build up and slow down the game.

To destroy the firework effect, follow these steps:

1. **With the `obj_button_firework` open (see the previous procedure), choose `Add Event` → `Alarm` → `Alarm 2`.**

The Alarm 2 Event appears in the Events section.



2. **Drag and drop a `Destroy Instance Action` from the `Main1` tab to the `Actions` section.**

The Destroy Instance properties window appears.

3. **In the `Applies To` section, select the `Self` radio button.**
4. **Click `OK`.**

The Destroy Instance window is saved and closed.

5. **Click `OK` from the `Object Properties` window.**

The Object Properties are saved and closed.

Now it is time to add an Instance of the button to the Room. If you're following along with the in-software tutorial, a Room was already created for you.

To add an Instance of the button to the Room, follow these steps:

1. **With the `Particles` tutorial open, from the `Resource tree`, in the `Rooms` section, double-click `Room0`.**

The Room Properties window for Room0 appears.

2. **Click the `Objects` tab (see Figure 7-7).**
3. **Click the drop-down icon and choose `obj_button_builtin` (refer to Figure 7-7).**

A thumbnail of the Sprite appears and the name of the Sprite appears in the field.

4. **Click in the bottom-left corner of the Room.**

An Instance of the Object appears in the Room (refer to Figure 7-7).

5. **Click the green check mark from the icon menu.**

The Room Properties window is saved and closed.

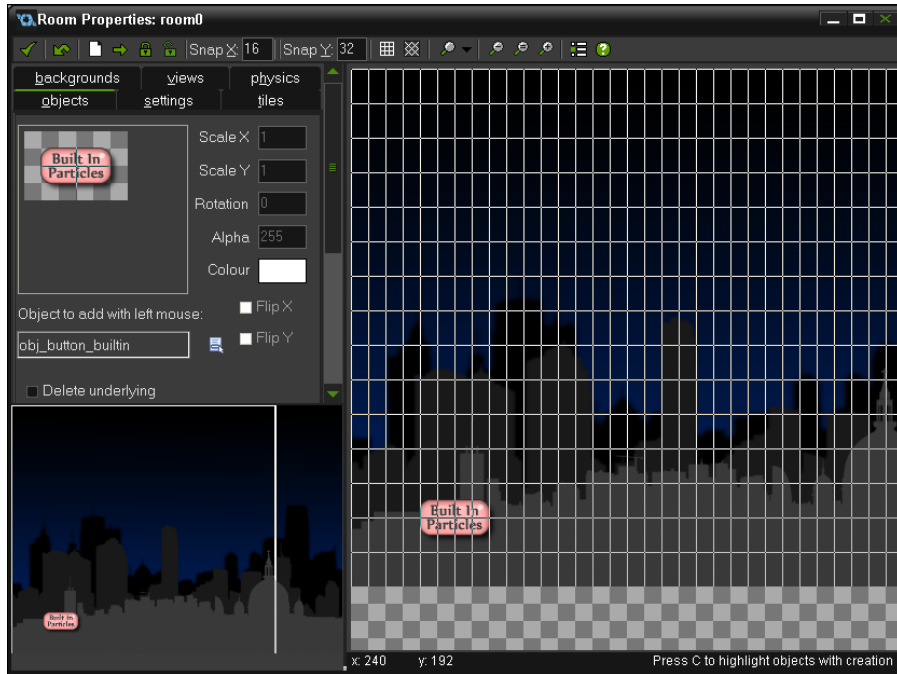


Figure 7-7: The Room Properties window.



Now test your game (choose Run→Run Normally from the main menu of the project window) to see your fireworks light up the sky. When you run the game, you should see the button change when you hover over it. When you click the button, you should see a trail of smoke generate from a random position in the Room that travels toward the center of the Room, and then finally you should see an explosion of yellow color. If you repeatedly press the button, you see multiple firework explosions.

Using the Particle Actions and Emitters

The Create Effect Action is great for simple special effects, but it's limited to a single color at a time, and it lacks detailed customization. Because of this, in this section I show you how to use the Particles Actions. You can find them on the Extra tab of the Object Properties window.

The following procedures show you how to create a new Object that you can use to display a Catherine wheel firework (an effect that rotates quickly and produces a display of sparks and colored flame), complete with puffs of smoke.

Creating puffs of smoke

To create a puff of smoke, follow these steps:

1. **From the Resource tree, right-click Objects and choose Create Object.**

The Object Properties window appears.

2. **In the Name field, type obj_catherine_firework.**

3. **Choose Add Event → Create.**

The Create Event appears in the Events section.

4. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

5. **In the code window, type the following code:**

```
speed = 0;  
alarm[0] = 1;  
alarm[1] = room_speed * 3;
```

This code prevents the Object from moving and sets the necessary Alarms.

6. **Click the green check from the icon menu.**

The code window is saved and closed.

Now add a Create Particle System Action to the obj_catherine_firework Object.



1. **Drag and drop a Create Particle System Action from the Extra tab to the Actions section.**

The Create Particle System properties window appears.

2. **In the Depth field, type 0, which is the default value.**

3. **Click OK.**

The Create Particle System Action window is saved and closed. You've now created a (necessary) container in which you place the particles and emitters that you want to use.

Now add a Create Particle Action, which is different from the Create Particle System Action from the last procedure.



1. **Drag and drop a Create Particle Action (you can find it directly below the Create Particle System Action) from the Extra tab to the Actions section.**

2. **From the Type ID drop-down list, select Type 0 (because it is the first particle effect you're creating).**

Type 0 is the default value.

3. From the **Shape drop-down list**, select **Cloud**.
4. From the **Sprite drop-down list**, select **No Sprite**, which is the default value.

You can use a custom Sprite if you want.

5. In the **Min Size field**, type 0.05.
6. In the **Max Size field**, type 0.2.

These last two properties create a range of sizes between a minimum of 5 percent and a maximum of 20 percent. If you want all the particles to be the same size, use the same value for both Min and Max.

7. In the **Size Increment field**, type 0.02.

This value determines the amount the particle will increase in size per step of the game. If you don't want a change in size, set this to 0, or use a negative number to decrease the size of the particle.

8. Click **OK**.

The Create Particle properties window is saved and closed.

Now add a Particle Color Action to the Object.



1. Drag and drop a **Particle Color Action** from the **Extra tab** to the **Actions section**.

The Particle Color properties window appears.

2. From the **Type ID drop-down list**, select **Type 0**.

This determines the particle you want to work with. In this case, you want to extend what you started with the Create Particle Action.

3. From the **Color Mix drop-down list**, select **Changing**.

This creates a blend of color over each particle's lifetime from Color1 to Color2, which you'll set next.

4. Click the **Color1 drop-down**.

The Color window appears.

5. Select a **light gray color** from the color graph and then choose **OK**.

This color appears as a smoke effect.

6. Click the **Color2 drop-down**.

The Color window appears with a color graph.

7. Select a **dark gray color** and then choose **OK**.

You now have a changing color effect that blends between light gray and dark gray for the smoke effect.

8. In the Start Alpha field, type 1.

9. In the End Alpha field, type 0.

The Start and End Alpha properties, as configured here, allow the particles to fade out over time.

10. Click OK.

The Particle Color properties window is saved and closed.

Now add a Particle Life Action to the Object.



1. Drag and drop a Particle Life Action from the Extra tab to the Actions section.

This Action determines the life span of the particle.

The Particle Life properties window appears.

2. From the Type ID drop-down list, select Type 0.

3. In the Min Life field, type 5.

Setting a different value for the minimum length of the particle's life and the maximum creates a range of time that the particle will exist.

4. In the Max Life field, type 30.

The lifetime of a particle is calculated in steps, so these particles will be shown in the Room for a minimum of 5 steps and a maximum of 30.

5. Click OK.

The Particle Life properties window is saved and closed.

Now add a Particle Gravity Action to the Object.



1. Drag and drop a Particle Gravity Action from the Extra tab to the Actions section.

The Particle Gravity properties window appears.

2. From the Type ID drop-down list, select Type 0.

3. In the Amount field, type 0.3.

4. In the Direction field, type 90.

These settings configure the smoke to rise at a 90-degree angle.

5. Click OK.

The Particle Gravity properties window is saved and closed.

You've now completed creating the smoke effect. In the next section, I show you how to create the Catherine wheel firework effect.

Creating the Catherine wheel firework

In the preceding section, I show you how to create a smoke puff that will be attached to the firework effect that I show you how to make in this section. This time, you use Type 1 instead of Type 0, because this is the second particle effect you're creating for this Object, `obj_catherine_firework`.

I highly recommend that you complete the preceding smoke trail procedure if you haven't already done so. The following procedure builds off the smoke trail procedure and uses that effect in the firework effect.

To create the Catherine wheel firework effect, follow these steps:



1. **With the `obj_catherine_firework` Object open, drag and drop a Create Particle Action.**

The Create Particle properties window appears.

2. **From the Type ID drop-down list, select Type 1.**

You already used Type 0 in the last procedure, so now it's time to use Type 1.

3. **From the Shape drop-down list, select Flare.**
4. **From the Sprite drop-down list, select No Sprite.**
5. **In the Min Size field, type 0.1.**
6. **In the Max Size field, type 0.5.**
7. **In the Size Increment field, type 0.01.**
8. **Click OK.**

The Create Particle properties window is saved and closed.

Now add a Particle Color Action to the Object.



1. **Drag and drop a Particle Color Action from the Extra tab to the Actions window.**

The Particle Color properties window appears.

2. **From the Type ID drop-down list, select Type 1.**
 3. **From the Color Mix drop-down list, select Mixed.**
 4. **Choose the Color1 drop-down.**
- The Color window appears with a color graph.
5. **From the color graph, select a light red color.**
 6. **Choose OK to save and close the color graph.**
 7. **Choose the Color2 drop-down.**

A window with a color graph appears.

8. From the color graph, select a yellow color.
9. Choose OK to save and close the color graph.
10. In the Start Alpha field, type 1.
11. In the End Alpha field, type 0.
You get a mix of colors between light red and yellow.
12. Click OK.

The Particle Color properties window is saved and closed.

Now add a Particle Life Action to the Object.



1. Drag and drop a Particle Life Action from the Extra tab to the Actions section.

The Particle Life properties window appears.

2. From the Type ID drop-down list, select Type 1.
3. In the Min Life field, type 15.
4. In the Max Life field, type 30.

You just set the life span of the particle to last between 15 and 30 steps.

5. Click OK.

The Particle Life properties window is saved and closed.

Now add a Particle Speed Action to the Object.



1. Drag and drop a Particle Speed Action from the Extra tab to the Actions tab.

The Particle Speed properties window appears.

2. From the Type ID drop-down list, select Type 1.
3. In the Min Speed field, type 10.
4. In the Max Speed field, type 10.

All the particles will travel at the same speed because you set the Min and Max Speed with the same value.

5. In the Min Dir field, type 0.
6. In the Max Dir field, type 359.

Setting a minimum of 0 degrees and a maximum of 359 degrees will create a circular effect when enough particles are created at one time.

7. In the Friction field, type 1.

The speed of the particles will slow down at a rate of 1 pixel per step and stop when the speed reaches 0. This effectively limits the distance the particles can travel. Friction will never reduce the speed to less than 0.

8. Click OK.

The Particle Speed properties window is saved and closed.

Now add a Particle Secondary Action to the Object.



1. Drag and drop a Particle Secondary Action from the Extra tab to the Actions section.

The Particle Secondary properties window appears.

2. From the Type ID drop-down list, select Type 1.

3. From the Step Type drop-down list, select Type 0.

4. In the Step Count field, type 0.

5. From the Death Type drop-down list, select Type 0.

6. In the Death Count field, type 1.

7. Click OK.

The Particle Secondary properties window is saved and closed.

This procedure creates a number of secondary particles (Type 0, which is the smoke trail from the last section) when the particle reaches the end of its lifetime (the death count), which means you'll see smoke when the sparking and flashing is done.



The Particle Secondary Action can create many thousands of particles, or even an infinite loop of particle creation, which can slow down and crash your game.

Creating an emitter for the firework

Now that you've created a smoke trail and the firework effect, you need to create an *emitter*, which is used to create the particles you defined earlier. Emitters can either *burst* or *stream* particles. Bursting is a one-off explosion of particles, and streaming means that particles are generated every step of the game.



Emitters must be stored within a Particle System in the same fashion as particles are. In this case, you can use the same Particle System as you created in the previous procedures.

To create an emitter, follow these steps:



1. With the `obj_catherine_firework` Object open, drag and drop a Create Emitter Action from the Extra tab into the Actions section.

It's the third icon from the bottom in the left column — it looks just like the Create Particle Action.

The Create Emitter properties window appears.

2. From the Emitter ID drop-down list, select Emitter 0.
3. From the Shape drop-down list, select Ellipse.
4. In the Xmin field, type $x - 5$.
5. In the Xmax field, type $x + 5$.
6. In the Ymin field, type $y - 5$.
7. In the Ymax field, type $y + 5$.

The X and Y positions define the areas in which the Instances will be created. The configurations shown here provide a slight variation in the distance each particle travels.



If you want the emitter to move with an Instance, place this Action in a Step Event of the Instance so that the Emitter position is updated every step.



8. Drag and drop a Set Alarm Action from the Main2 tab to the Actions section.

The Set Alarm Action properties window appears.

9. In the Applies To section, select the Self radio button.
10. In the Number of Steps field, type 1.
11. From the In Alarm No drop-down list, select Alarm 0.

You just set an Alarm Action to trigger Alarm 0 for every step. You set the Alarms in the following procedures.

At this point, your Actions for the Create Event should look like Figure 7-8.

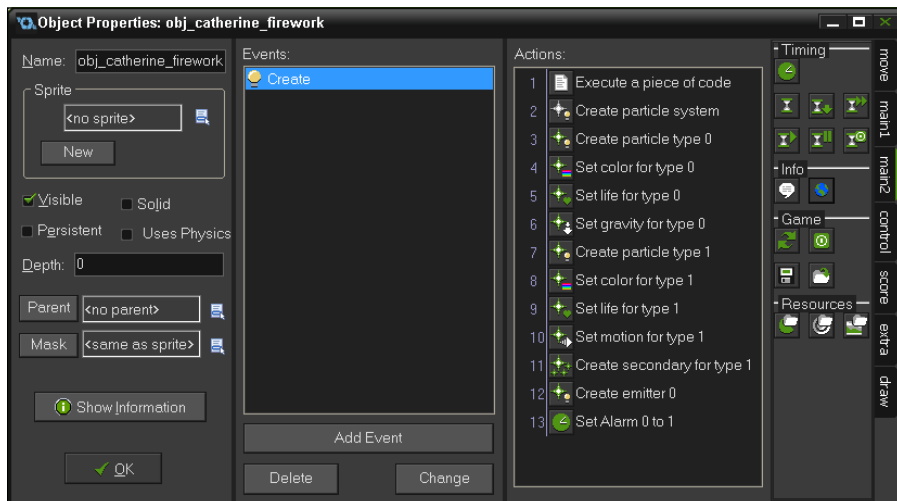


Figure 7-8: The Actions list for the Create Event.

Now you need to add an Alarm[0] Event to control the Create Event. To create the Alarm Event, follow these steps:

1. With the `obj_catherine_firework` Object open, choose **Add Event** ⇨ **Alarm** ⇨ **Alarm 0**.



2. Drag and drop a **Burst From Emitter Action** from the **Extra** tab to the **Actions** section.

The Burst From Emitter properties window appears.

3. From the **Emitter ID** drop-down list, select **Emitter 0**.
4. From the **Particle Type** drop-down list, select **Type 1**.

Remember that Type 1 is the Catherine wheel effect from earlier in this chapter.

5. In the **Number** field, type 50.

This will burst 50 particles of Type 1.

6. Click **OK**.

The Burst From Emitter properties window is saved and closed.

Now add a Set Alarm Action to the Object.



1. Drag and drop a **Set Alarm Action** from the **Main2** tab to the **Actions** section.

The Set Alarm properties window appears.

2. In the **Applies To** section, select the **Self** radio button.
3. In the **Number of Steps** field, type `room_speed / 5`.
4. From the **In Alarm No** drop-down list, select **Alarm 0**.
5. Click **OK**.

The Set Alarm properties window is saved and closed. You just set an Action to trigger Alarm 0 once for every five steps of the game.

Now add an Alarm[1] Event to the Object.

1. Choose **Add Event** ⇨ **Alarm** ⇨ **Alarm 1**.

The Alarm 1 Event appears in the Events section.

2. Drag and drop a **Set Alarm Action** from the **Main2** tab to the **Actions** section.

The Set Alarm properties window appears.

3. In the **Applies To** section, select the **Self** radio button.
4. In the **Number of Steps** field, type `-1`.

A negative number essentially turns off the Alarm selected, which means the Event won't be triggered.

5. From the In Alarm No drop-down list, select Alarm 0.
6. Click OK.

The Set Alarm Action window is saved and closed. You just set an Action to turn off Alarm 0.

Now add another Set Alarm Action to the Object.

1. Drag and drop a Set Alarm Action from the Main2 tab to the Actions section.

The Set Alarm properties window appears.

2. In the Applies To section, select the Self radio button.
3. In the Number of Steps field, type `room_speed * 2`.
4. From the In Alarm No drop-down list, select Alarm 2.

Now Alarm 2 will be triggered twice every step.

5. Click OK.

The Set Alarm properties window is saved and closed.

Now add an Alarm[2] Event to the Object.

1. Choose Add Event⇨Alarm⇨Alarm 2.

The Alarm 2 Event appears in the Events section.



2. Drag and drop a Destroy Emitter Action from the Extra tab to the Actions section.

The Destroy Emitter properties window appears.

3. From the Emitter ID drop-down list, select Emitter 0.
4. Click OK.

The Destroy Emitter properties window is saved and closed. You just set an Action to destroy the emitter when Alarm 2 is triggered.



5. Drag and drop a Destroy Particle System Action from the Extra tab to the Actions window.

The Destroy Particle System Action appears in the Actions section. There is nothing to set here — you're essentially creating an Action to destroy the particle system when Alarm 2 is triggered.



6. Drag and drop a Destroy Instance Action from the Main1 tab to the Actions section.

The Destroy Instance properties window appears.

7. In the Applies To section, select the Self radio button.

This Alarm will stop the effect and destroy the Emitter, the Particle System, and the Instance.

8. Click OK.

The Destroy Instance properties window is saved and closed.

9. Click OK from the Object Properties window.

The Object Properties window is saved and closed.

Your Alarm 2 Event should look like Figure 7-9.

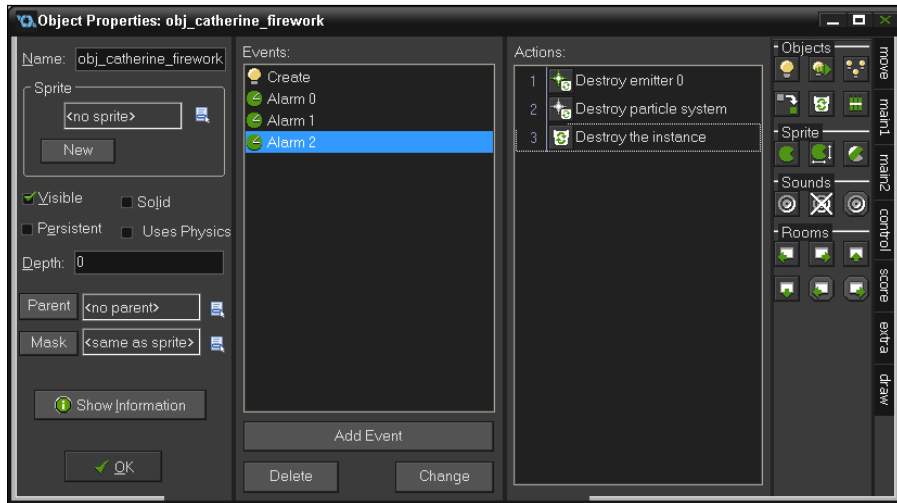


Figure 7-9: The Alarm 2 Event destroys the firework Effect.

With these procedures complete, you can move on to create a button so that the player can trigger this firework. You can duplicate the button you made for the Create Effect Action from the beginning of this chapter to create a new button quickly.

To create a button to trigger the Catherine wheel effect, follow these steps:

- 1. From the Resource tree, in the Object section, right-click `obj_button_builtin`, and choose Duplicate (you made this Object earlier in the chapter).**

A duplicate of the Object is created and appears in the Resource tree as Object2, or something similar. The Object Properties window also appears.

- 2. In the Name field, type `obj_button_catherine`.**

- 3. From the Sprite drop-down list, select `spr_Button_DnD`.**

A thumbnail of the Sprite appears and the name of the Sprite appears in the field.

4. From the Events section of the Object Properties window, select the Step Event.

The Step Event Actions appear in the Actions section.

5. Double-click the Execute Code Action.

The code window appears.

6. In the code window, type the following code (see Figure 7-10):

```
if instance_position(mouse_x, mouse_y, id)
{
  if image_index != 2
  {
    image_index = 1;
    if mouse_check_button_pressed(mb_left) && !
      instance_exists(obj_catherine_firework)
    {
      alarm[0] = room_speed / 2;
      image_index = 2;
      instance_create(random(room_width), 350, obj_catherine_firework);
    }
  }
}
else
{
  if image_index != 2
  {
    image_index = 0;
  }
}
```

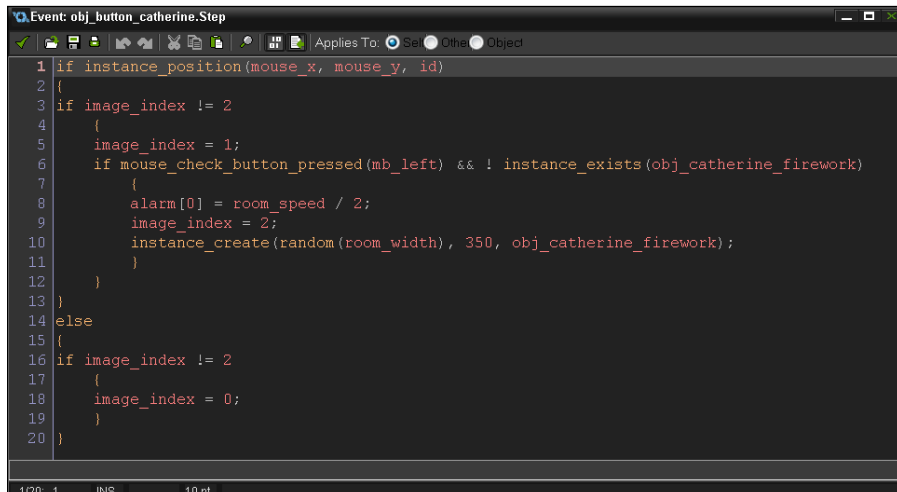


Figure 7-10: The Code window for the Step Event.

This code is basically the same code as the previous button, but it allows a single Instance of the firework at a time. Go ahead and place an Instance of the button in the Room now and run the game.



To add an Instance of the button, open the Room by double-clicking Room0 from the Resource tree in the Rooms section. In the Room Properties window, select the Objects tab. Make sure that `obj_button_builtin` appears and then click in the Room to place it. For more details, see Chapter 2 and earlier in this chapter.

When you test the game now, you might notice that you can only click the Catherine wheel button (labeled D'n'D) once and then you have to wait for the effect to finish before you can trigger it another time. For the Built In Particles effect, you can click that button multiple times to have the effect appear multiple times.

Coding Special Effects with Particles

In This Chapter

- ▶ Coding particle systems
- ▶ Creating a simple firework using code
- ▶ Coding both burst and stream emitters
- ▶ Putting it all together into one complex particle effect

In Chapter 7, I show you how to create special effects using both the Create Effects Action and the Particles Action. In this chapter, I move on to coding particle effects, which requires more work than the drag-and-drop Actions.

Throughout this chapter, I show you how to re-create the first firework that you made using the Create Effects Action, but this time using code. You can see an example of the effect in Figure 8-1.

First, I step you through coding the particle system and the smoke trail. Then I explain how to code the firework effect itself. I move on to coding burst and stream emitters. And I wrap up the chapter by showing you how to code a complex particle effect using everything I've showed you so far.



Throughout this chapter, I refer to a button that you can use to trigger your effects while playing the game. I described how to create this button at the beginning of Chapter 7. Also, this chapter is based on the in-software tutorial, Particles.



Figure 8-1: The simple code firework effect.

Coding the Particle System and the Smoke Trail

To get started coding the firework effect, which you can use for, say, explosions in your game, you need to create a particle system. The following procedure shows you how to do this through code, along with creating a smoke trail for the effect.



You may want to start by opening the in-software Particles tutorial, but it's not necessary.

To code a firework effect, follow these steps:

1. **From the Resource tree, choose Object→Create Object.**

The Object Properties window appears.

2. **In the Name field type, obj_simple_firework (or something along those lines).**

3. **To start creating the particle system, choose Add Event→Create.**

The Create Event appears in the Events section.

4. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

5. **In the code window, type the following code (see Figure 8-2).**

```
///particle system
Sname = part_system_create();
```



The first line of code is a type of comment that you can use to name the Action, so to speak. When you use three backslashes, you're creating a comment that appears as the name of the Action in the Actions section of the Object Properties window.

The second line of code creates a particle system and assigns it an index variable of `Sname`, which you refer to later, when defining the particles.

When you only use two backslashes in your code, GameMaker treats it as a normal comment, which can serve as a useful reminder of what the following code does (or at least what it's supposed to do). In the following code, you're creating a particle type for the smoke trail and you're assigning its index to a variable (trail):

```
//trail  
trail = part_type_create();
```

The following code assigns a particle shape (spark) for the particle type (trail).

```
part_type_shape(trail, pt_shape_spark);
```

You can press F1 within the software to search for a full list of shapes to choose from.

The following code sets the particle scale for the trail.

```
part_type_scale(trail, 1, 1);
```

The particle scale is the base value that the particle is based on, where 1 is a 1:1 scale. So, if you want your particle (before any size is changed) to be scaled by 2 along the X axis and by half along the Y axis, you would set the scale to 2, 0.5 for the X and Y axis.

The following code sets the size of the particle:

```
part_type_size(trail, 0.20, 0.50, 0, 0);
```

You specify a minimum (for example, 0.20) and maximum (for example, 0.50) range for the particle size, and each particle will be set to a random size within that range. Also, you can apply an increment to make the size grow (positive value), shrink (negative value), or remain constant (0).

The final value is referred to as a *wiggle*. Wiggle adds an amount to the target function for each step, but the change lasts only for a step, so the particle will appear to jump in size (or speed or direction). You can specify a value from 0 to 20 for wiggle, and a random number will be generated from 0 to *n* each step. This will be added onto the final particle property. This is not necessary for this procedure, so leave it as 0 for now.

There are three functions for color with particles (straight color, two-color gradient, and three-color gradient), but for this, use the `color2` function to create a gradient from one color (white) to the other (gray) over the lifetime of the particle:

```
part_type_color2(trail, c_white, c_gray);
```

There are also three alpha functions available, depending on what effect you want to use (straight alpha, alpha change from a start alpha to an end alpha, and a three-alpha fade over the particle lifetime). In the following code, you're using the two-alpha fade (`alpha2`), with an alpha of 1 (opaque) at the start of the particle life that transitions to 0 (transparent) at the end.

```
part_type_alpha2(trail, 1, 0);
```

The following line of code sets the speed of the particle (`trail`) and is a range from a minimum (for example, 2) to a maximum (for example, 5) and can be made to move faster or slower by changing the increment (for example, `-0.20`).

```
part_type_speed(trail, 2, 5, -0.20, 0);
```

This function also has a wiggle, which is added to the base speed of each particle and can be from 0 to 20. This particle will have no wiggle (0). If you do want it to have wiggle, use 1.

For the direction of the Instance, you set a minimum (for example, 0) and maximum (for example, 359) direction, as well as an increment (for example, 0) and a wiggle (0). The increment will add (or subtract) an amount to the current direction, providing nice swirling and curling effects. For this effect, you're setting a random direction from 0 degrees to 359 degrees with no increment or wiggle:

```
part_type_direction(trail, 0, 359, 0, 0);
```

The tail should be affected by gravity, so use the following function to add a small amount of gravity to pull the particles down. First, you set the force of the gravity (for example, `0.5`), and then you set the direction (for example, `270`).

```
part_type_gravity(trail, 0.5, 270);
```

The orientation of the particle works the same as the image angle of a Sprite, and gives the particle a rotational angle. For this effect, a random value between 0 degrees and 360 degrees is fine:

```
part_type_orientation(trail, 0, 359, 0, 0, 0);
```

The following function controls *blending*, which uses an *additive* blend mode, meaning that the light parts of the particle will have their luminosity added to that of any other particles that they may overlap with while in the Room. It gives particles a glow effect when there are many of them, and it's very useful. But for the trail, leave it set to false (0).

```
part_type_blend(trail, 0);
```

The following function is the last one. It controls the lifetime of the trail particle. It has a random value in that all the trail particles will last from between 8 and 30 steps.

```
part_type_life(trail, 8, 30);
```

6. Click the green check mark as shown in Figure 8-2.

The code window is saved and closed.

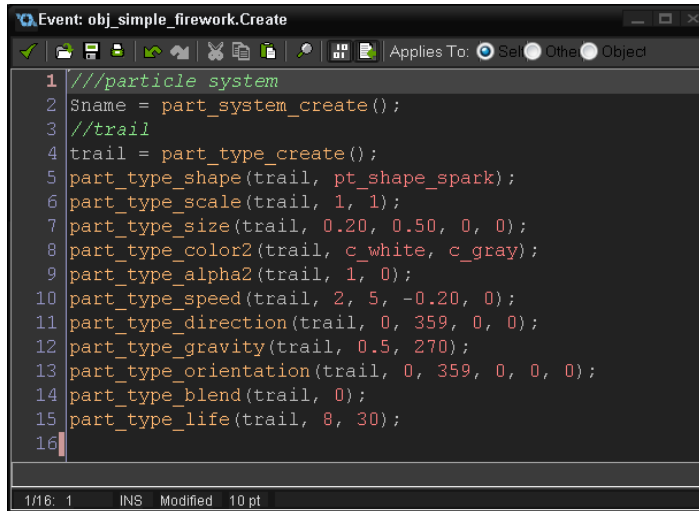


Figure 8-2: Code for the particle system and smoke trail.

Coding the Simple Firework Effect

This section continues from the previous section, in which you wrote the code to create a particle system and smoke trail for the firework. In this section, I pick up on the simple coding tutorial by showing you the code for the firework effect. This code is very similar to what you did in the last section for the trail. So, I won't explain each function in detail like I did in the last procedure. If you want to learn about each function, see the preceding section.

The following code picks up from the last procedure; if you didn't follow the last procedure, simply pick up by creating a new Object, adding a Create Event, and then dragging and dropping an Execute Code Action.

To code the firework effect, follow these steps:

1. Open the firework Object that you started in the last procedure (I named mine obj_simple_firework).

You should see the Create Event in the Events section and an Execute Code Action in the Actions section.

2. Double-click the Execute Code Action from the Action section (if it isn't already open from the last procedure).

The code window appears populated with the code from the preceding section.

3. In the code window, type the following code (see Figure 8-3).

```
//firework (this is a comment to denote where the code for the firework
starts)
stars = part_type_create();
part_type_shape(stars, pt_shape_star);
part_type_scale(stars, 1, 1);
part_type_size(stars, 0.5, 1, -0.02, 0);
part_type_color3(stars, c_lime, c_fuchsia, c_blue);
part_type_alpha3(stars, 0, 1, 0);
part_type_speed(stars, 2, 3, 0, 0);
part_type_direction(stars, 0, 359, 0, 0);
part_type_gravity(stars, 0.20, 270);
part_type_orientation(stars, 0, 359, 0, 0, 0);
part_type_blend(stars, 1);
part_type_life(stars, 50, 70);
```



I simply copied and pasted the trail code from the last procedure and then updated the differences (changing “trail” to “stars,” and then tweaking the values).

A key difference in this code is that the additive blend was turned on (1) in the second-to-last line of code. This causes the firework to appear brighter and more luminous than the trail. Also, you probably noticed that the color scheme got an upgrade by using `color3` and `alpha3` functions to provide a wider range of colors and effects.

```
13 part_type_orientation(trail, 0, 359, 0, 0, 0);
14 part_type_blend(trail, 0);
15 part_type_life(trail, 8, 30);
16 //firework
17 stars = part_type_create();
18 part_type_shape(stars, pt_shape_star);
19 part_type_scale(stars, 1, 1);
20 part_type_size(stars, 0.5, 1, -0.02, 0);
21 part_type_color3(stars, c_lime, c_fuchsia, c_blue);
22 part_type_alpha3(stars, 0, 1, 0);
23 part_type_speed(stars, 2, 3, 0, 0);
24 part_type_direction(stars, 0, 359, 0, 0);
25 part_type_gravity(stars, 0.20, 270);
26 part_type_orientation(stars, 0, 359, 0, 0, 0);
27 part_type_blend(stars, 1);
28 part_type_life(stars, 50, 70);
29
```

Figure 8-3: The firework code.

3. Now add in the following code to configure how the Instance moves (see Figure 8-4):

```
//movement
direction = point_direction(x, y, room_width / 2, 100);
speed = 15 + random(5);
gravity = 0.5;
gravity_direction = 270;
alarm[0] = 1;
alarm[1] = room_speed;
```

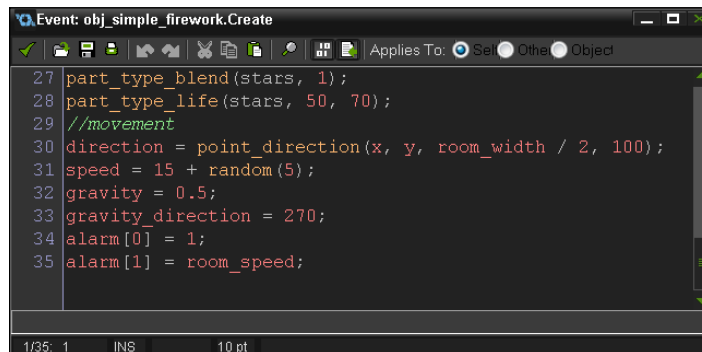


Figure 8-4: The movement code.

4. Click the green check mark.

The code window is saved and closed.

5. From the Object Properties window, choose Add Event→Alarm→Alarm 0.

The Alarm 0 Event appears in the Events section.

6. Drag and drop an Execute Code Action from the Control tab to the Actions section.

A code window appears.

7. In the code window, type the following code:

```
part_particles_create(Sname, x, y, trail, 1);
alarm[0] = 1;
```

The function, `part_particles_create()`, enables you to create particles at any point in the Room without using emitters. It's fast and easy to set up (as you can see in this example). When used every step, it can create stream effects; when used once, it can create burst effects, just like an emitter. You can also use the random functions to create the particles over an area, which can simulate a rectangular emitter with a linear distribution.



When you're working with particles, always keep in mind this function (as well as its extended function `part_particles_create_color`, which permits you to create particles of a specific color) due to its ease of use and small memory footprint.

8. Click the green check mark.

The code window is saved and closed.

9. Choose Add Event ⇨ Alarm ⇨ Alarm 1.

The Alarm 1 Event appears in the Events section.

10. Drag and drop an Execute Code Action from the Control tab to the Actions section.

A code window appears.

11. In the code window, type the following code:

```
alarm[0] = -1;
part_particles_create(Sname, x, y, stars, 200);
alarm[2] = 1;
```

This Alarm uses the create particles function to burst the star particles from the firework and sets Alarm 2, which destroys the Instance.

12. Click the green check mark.

The code window is saved and closed.

13. Choose Add Event ⇨ Alarm ⇨ Alarm 2.

The Alarm 2 Event appears in the Events section.

14. Drag and drop an Execute Code Action from the Control tab to the Actions section.

A code window appears.

15. In the code window, type the following code:

```
if part_particles_count(Sname) > 0
{
    alarm[2] = 1;
}
else
{
    part_type_destroy(trail);
    part_type_destroy(stars);
    part_system_destroy(Sname);
}
```

This code destroys all the Instances and adds another new function, `part_particles_count`. This function is extremely useful for things like timing an Instance's destruction as it counts the number of particles

currently in the Room from a given system. The preceding code checks this value, and when it returns that no particles are present, it destroys the particles and the system. It's important to do it this way, because if you destroy the particles and system while there are particles visible in the Room, they'll just disappear and that does not look good. It kinda takes away the "special" from the special effect.

16. Click the green check mark.

The code window is saved and closed.

At this point, you have a simple coded particle firework complete with smoke trail. Now you just need to create a button for it, place it in the Room, and test it out. The following procedure describes how to duplicate and update the button I showed you how to create in Chapter 7.

To create a button for the simple code firework, follow these steps:

1. From the Resource tree, right-click `obj_button_builtin` (see Chapter 7) and select **Duplicate.**

The duplicated Object Properties window appears.

2. In the Name field, type `obj_button_simple`.

3. In the Sprite section, click the drop-down list and select `spr_Simple_Button` (from the Particles tutorial assets).

The thumbnail and name of the Sprite appear.

4. Select the Step Event from the Events section.

The Actions for the Step Event appear in the Actions section.

5. From the Actions section, double-click the **Execute Code Action.**

The code window appears.

6. Update the code so that it creates an Instance of `obj_simple_firework` (most likely, line 10 of your code, see Figure 8-5).

```
instance_create(random(room_width), 400, obj_simple_firework);
```

You can now open the Room and add an Instance of the simple button Object (see Chapter 7 for details). Then save and test your game. You should now have three working special effects if you've been following along starting from Chapter 7.

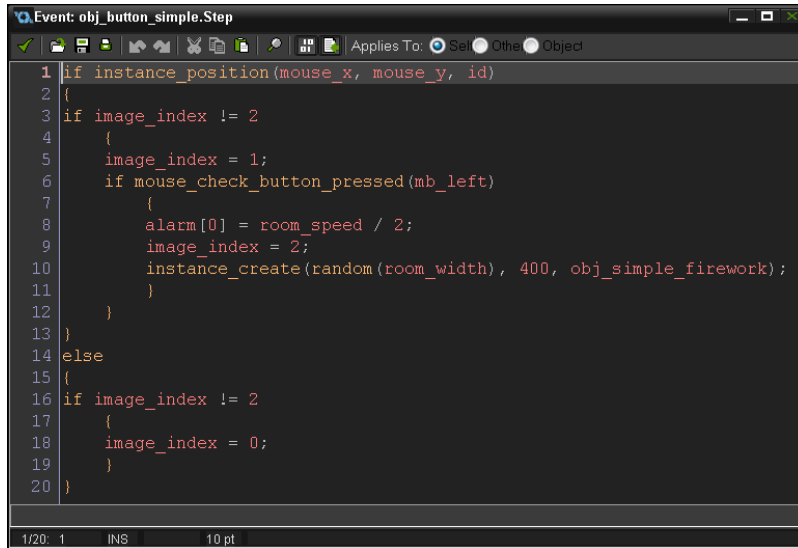


Figure 8-5: The Step Event code for the new button.

Coding Emitters

Emitters are used to create particles at specific positions and areas in the Room. In Chapter 7, I show you how to use the Create Emitter Action from the Extra tab. And, earlier in this chapter, I show you how to burst/stream particles without an emitter using the function `part_particles_create`. In this section, I describe how to code an emitter.



Emitters, just like particles and particle systems, are stored in memory during gameplay and indexed into a variable, so use them carefully to prevent memory leaks or other errors. It's always considered best practice to destroy all Instances of particles, emitters, and systems after they're no longer a part of the Room.

Coding burst emitters

For this firework effect, the only Object you need is a button Object. You place an Instance of this button Object in the Room, and when the player presses it, the particles burst.

The following procedure is a continuation of the other procedures in this chapter and Chapter 7, so the first step has you duplicate a button that I show you how to create in Chapter 7. If you haven't read Chapter 7 yet, skip back and follow the procedure to create `obj_button_builtin` there.

To code a burst emitter, follow these steps:

1. **From the Resource tree, right-click `obj_button_builtin` and select **Duplicate**.**

The duplicated Object Properties window appears.

2. **In the Name field, type `obj_button_burst`.**
3. **From the Sprite section, click the drop-down list and choose `spr_Button_Burst` (from the Particles tutorial assets).**

The thumbnail and name of the Sprite appear.

4. **From the Events section, select the **Create Event**.**

The Actions for the Create Event appear in the Actions section.

5. **From the Actions section, double-click the **Execute Code Action**.**

The code window appears.

6. **In the code window, type the following code to define the particle, which will move quickly upward and then will slowly fall due to the gravity function (see Figure 8-6):**

```
image_speed = 0;
image_index = 0;
Sname = part_system_create();
explode = part_type_create();
part_type_shape(explode, pt_shape_flare);
part_type_size(explode, 0.50, 1.50, 0.02, 1);
part_type_scale(explode, 0.25, 0.25);
part_type_color3(explode, 4227327, 8454143, 4210688);
part_type_alpha3(explode, 0, 1, 0);
part_type_speed(explode, 10, 20, -0.4, 5);
part_type_direction(explode, 45, 135, 0, 10);
part_type_gravity(explode, 0.40, 270);
part_type_orientation(explode, 0, 359, 0, 0, 0);
part_type_blend(explode, 1);
part_type_life(explode, 60, 120);
```

This code defines the colors by using absolute numeric values (such as 4227327). The code also turns on additive blending to improve the effect (`part_type_blend(explode, 1)`). In addition to blending, the code uses the wiggle effect (the last value in the size, speed, and direction functions).



Wiggle adds an amount to the target function for each step, but the change only lasts a step, so the particle appears to jump in size (or speed or direction). You can specify a value from 0 to 20 for wiggle, and a random number will be generated from 0 to n each step, and this will be added onto the final particle property.

7. **To define the emitter, add this code to assign a variable (`emitter1`) to the emitter (see Figure 8-6).**

```
emitter1 = part_emitter_create(Sname);
```

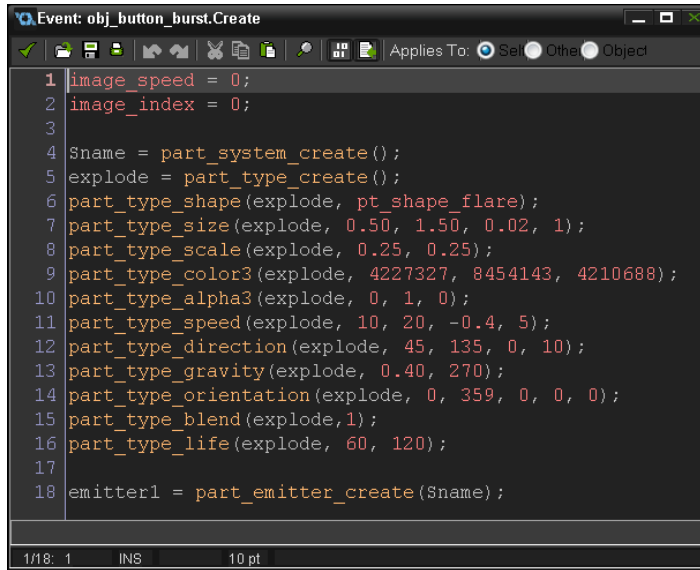


Figure 8-6: Create Event code for the burst emitter.

8. Click the green check mark.

The code window is saved and closed.

9. Select the Step Event from the Events section.

The Actions for the Step Event appear in the Actions section.

10. Double-click the Execute Code Action.

The code window appears.

11. Modify the code so that the particle bursts from the emitter as shown here and in Figure 8-7:

```

if instance_position(mouse_x, mouse_y, id)
{
if image_index != 2
{
image_index = 1;
if mouse_check_button_pressed(mb_left)
{
alarm[0] = room_speed / 2;
image_index = 2;
var posx, posy;
posx = choose(194, 305, 520);
if posx = 194 posy = 264 else if posx = 305 posy = 235 else
posy = 292;

```

```

        part_emitter_region(Sname, emitter1, posx - 64, posx + 64, posy
        - 32, posy + 32, ps_shape_ellipse, ps_distr_gaussian);
        part_emitter_burst(Sname, emitter1, explode, 100);
    }
}
else
{
    if image_index != 2
    {
        image_index = 0;
    }
}

```

This code moves the emitter around the Room based on a random value of 194, 305, or 520) and will burst 100 particles. The shape of the particle is also defined (here, it's ellipse, but you can also use rectangle, diamond, or line). The code also defines the distribution of the particle, which can be Gaussian (more in the middle), linear (equal distribution), or inverse Gaussian (more at the edges).

12. Click the green check mark.

The code window is saved and closed.

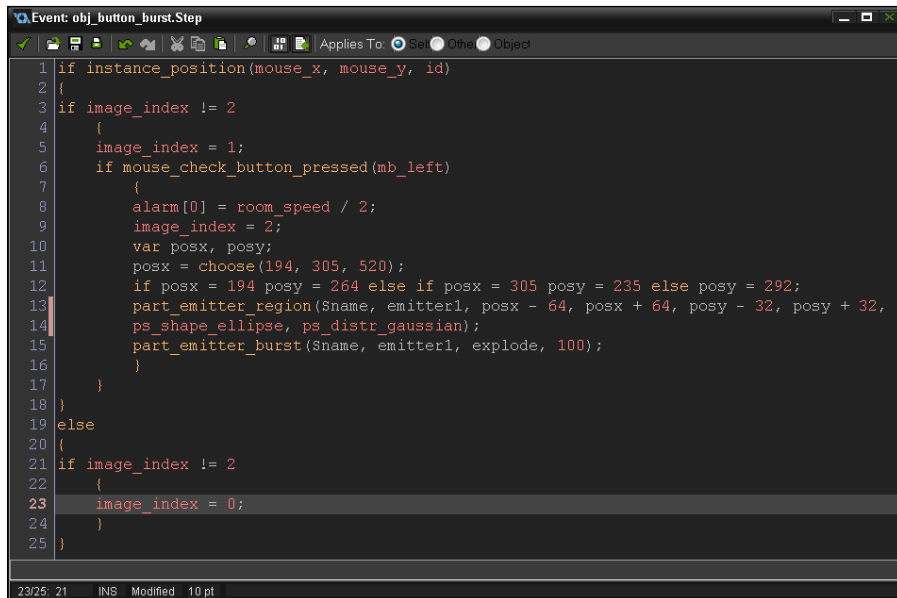


Figure 8-7: The Step Event code for the burst emitter.

13. Choose Add Event→Other→Room End.

The Room End Event appears in the Events section.

14. Drag and drop an Execute Code Action from the Control tab to the Actions section

A code window appears.

15. In the code window, type the following code:

```
part_type_destroy(explode);  
part_emitter_destroy(Sname, emitter1);  
part_system_destroy(Sname);
```

This code effectively destroys the particle, the emitter, and the system to help keep the game from crashing due to memory leaks and so on.

16. Click the green check mark.

The code window is saved and closed.

Now you're ready to add this button to the Room to test your newly coded emitter (see Chapter 7). When you click the Emitter Burst button, you should see a brilliant display of yellows and whites of varying degrees of color that fall slowly downward with lots of wiggle, as shown in Figure 8-8.

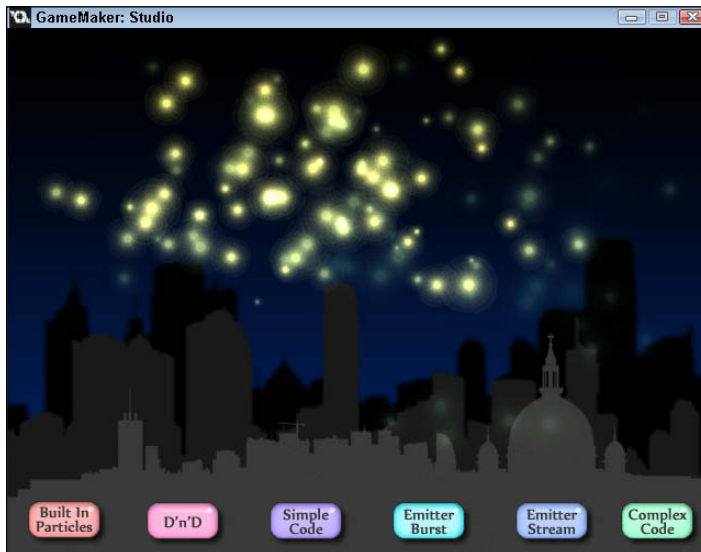


Figure 8-8: The emitter burst special effect.

This burst emitter works great for effects that need to be created for a specific moment. In the next section, I show you how to code a stream emitter, which works well for effects that need to trigger every step.

Coding stream emitters

The main difference between a stream emitter and a burst emitter is that you can use a special function in a stream emitter so that it continues to emit particles until you want it to stop. This provides a means to create excellent smoke effects, trail effects, and flame effects. In the following procedure, I show you how to code a stream emitter so that it displays a waterfall-style firework effect, as shown in Figure 8-9.



Figure 8-9: The emitter stream effect.



The following procedure is a continuation of previous procedures, in that the first step has you duplicate a button (`obj_button_builtin`) from Chapter 7.

To code a stream emitter, follow these steps:

1. **From the Resource tree, right-click `obj_button_builtin` and choose Duplicate.**

The duplicated Object Properties window appears.

2. **In the Name field, type `obj_button_stream`.**
3. **From the Sprite section, click the drop-down list and choose `spr_Button_Stream` (from the particles tutorial assets).**

The thumbnail and name of the Sprite appear.

4. **Select the Create Event from the Events section.**

The Actions for the Create Event appear in the Actions section.

5. Double-click the Execute Code Action.

The code window appears.

6. In the code window, update the code to match (see Figure 8-10):

```
image_speed = 0;
image_index = 0;
Sname = part_system_create();
waterfall = part_type_create();
part_type_shape(waterfall, pt_shape_pixel);
part_type_size(waterfall, 1, 2, -0.10, 0);
part_type_scale(waterfall, 8, 16);
part_type_color3(waterfall, 16777088, 16744703, 16776960);
part_type_alpha2(waterfall, 0, 1);
part_type_speed(waterfall, 2, 4, -0.30, 5);
part_type_direction(waterfall, 0, 180, 0, 10);
part_type_gravity(waterfall, 2, 270);
part_type_orientation(waterfall, 0, 359, 5, 0, 0);
part_type_blend(waterfall, 1);
part_type_life(waterfall, 5, 30);
emitter1 = part_emitter_create(Sname);
part_emitter_region(Sname, emitter1, 0, 640, 0, 16, ps_shape_rectangle,
    ps_distr_linear);
```

Of note for this code is that the particle shape is pixel and has an X scale of 8 and a Y scale of 16. The values for the X and Y scales stretch the particle by a large amount, which will have a dramatic effect.

7. Click the green check mark.

The code window is saved and closed.

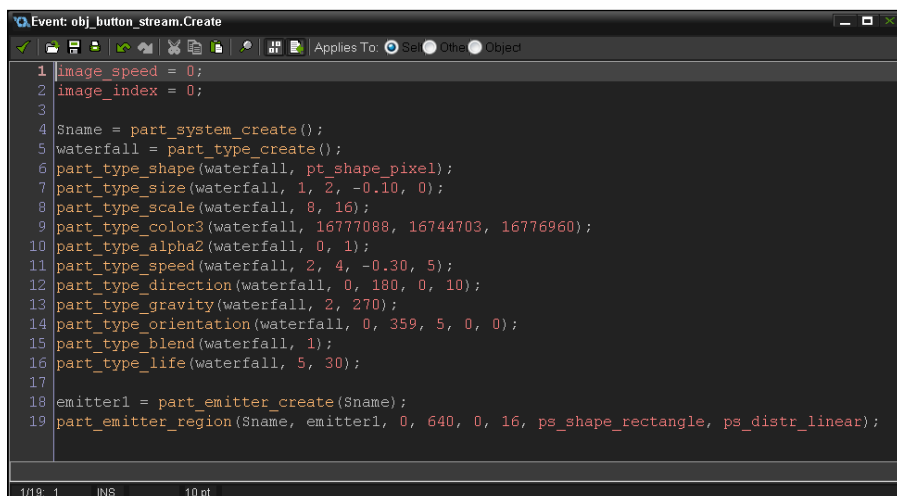


Figure 8-10: The Create Event code for the stream emitter.

8. Select the Step Event from the Events section.

The Actions for the Step Event appear in the Actions section.

9. Double-click the Execute Code Action from the Action section.

The code window appears.

10. In the code window, update the code to match the following (see Figure 8-11):

```
if instance_position(mouse_x, mouse_y, id)
{
  if image_index != 2
  {
    image_index = 1;
    if mouse_check_button_pressed(mb_left)
    {
      alarm[0] = room_speed * 3;
      image_index = 2;
      part_emitter_stream(Sname, emitter1, waterfall, 50);
    }
  }
}
else
{
  if image_index != 2
  {
    image_index = 0;
  }
}
```



In this code, the emitter will stream 50 particles each step. That means 50 new particles are created for each step of the game. Be careful with this, though: If too many particles are created without being destroyed, the multitude of particles can slow down your game.

11. Click the green check mark.

The code window is saved and closed.

12. Select the Alarm 0 Event from the Events section.

The Actions for the Alarm 0 Event appear in the Actions section.

13. Double-click the Execute Code Action from the Actions section.

The code window appears.

14. Update the code to match the following:

```
image_index = 0;
part_emitter_stream(Sname, emitter1, waterfall, 0);
```

The 0 tells the emitter to not emit any particles at all. By placing this in the Alarm Event, you're effectively stopping the emitter after a given amount of time. If you use a negative number here, you're telling the emitter to stream particles at a random number. For example, if you use -5, you'll have a one in five chance of emitting a particle each step.

```

1 if instance_position(mouse_x, mouse_y, id)
2 {
3   if image_index != 2
4   {
5     image_index = 1;
6     if mouse_check_button_pressed(mb_left)
7     {
8       alarm[0] = room_speed * 3;
9       image_index = 2;
10      part_emitter_stream(Sname, emitter1, waterfall, 50);
11    }
12  }
13 }
14 else
15 {
16   if image_index != 2
17   {
18     image_index = 0;
19   }
20 }

```

Figure 8-11: The Step Event code for the stream emitter.



The length of time the emitter streams the particles is based on when the Alarm is triggered from the Step Event. Within the Step Event is a line of code, `alarm[0] = room_speed * 3;`, which says, after three steps of the game, trigger the Alarm (the Alarm Event is set up to stop Step 15). If you want to test this, run the game (after you complete Step 15) and count one-one thousand, two-one thousand, three-one thousand, and then the effect should stop. Then change the code from 3 to 6. Now when you press the button, you can count up to six seconds before the particles stop emitting.

15. Choose Add Event→Other→Room End.

The Room End Event appears in the Events section.

16. Drag and drop an Execute Code Action from the Control tab to the Actions section.

A code window appears.

17. In the code window, type the following:

```

part_type_destroy(waterfall);
part_emitter_destroy(Sname, emitter1);
part_system_destroy(Sname);

```

18. Click the green check mark.

The code window is saved and closed.

You can now add `obj_button_stream` to the Room to test it (see Chapter 7).

Coding Complex Particle Effects



This section shows you how to put everything in this chapter into one firework with some new, advanced functions that I haven't discussed yet.

The first procedure starts with your duplicating the button that I show you how to create in the beginning of Chapter 7.

To create a complex particle effect, follow these steps:

1. **From the Resource tree, right-click `obj_button_builtin` and choose **Duplicate**.**

The duplicated Object Properties window appears.

2. **In the Name field, type `obj_button_complex`.**

3. **From the Sprite section, click the drop-down and choose `spr_Button_Complex` (from the Particles tutorial assets).**

The thumbnail and name of the Sprite appear.

4. **Select the **Create Event** from the Events section.**

The Actions for the Create Event appear in the Actions section.

5. **Double-click the **Execute Code Action** from the Actions section.**

The code window appears.

6. **Update the code in the window to match the following:**

```
image_speed = 0;  
image_index = 0;  
//create particle system  
Sname = part_system_create();
```

7. **Type in the following code for the smoke particle:**

```
//create smoke particle  
smoke = part_type_create();  
part_type_shape(smoke,pt_shape_cloud);  
part_type_size(smoke,0.10, 0.50,0.01,0);  
part_type_scale(smoke,0.50,0.50);  
part_type_color2(smoke,12632256,8421504);  
part_type_alpha2(smoke,1,0);  
part_type_speed(smoke,1,1,0,0);  
part_type_direction(smoke,0,359,0,0);  
part_type_gravity(smoke,0.05,90);  
part_type_orientation(smoke,0,0,0,0,1);  
part_type_blend(smoke,0);  
part_type_life(smoke,15,45);
```

8. Type in the following code for the firework particle:

```
//create firework particle
firework = part_type_create();
part_type_sprite(firework,spr_Custom_Particle, false, false, true);
part_type_size(firework,0.50,1,0,1);
part_type_scale(firework,0.5,0.5);
part_type_color3(firework,8454143,8453888,8421631);
part_type_alpha3(firework,1,0,1);
part_type_speed(firework,1,2,0,0);
part_type_direction(firework,0,359,0,0);
part_type_gravity(firework,0.03,270);
part_type_orientation(firework,0,0,0,0,1);
part_type_blend(firework,1);
part_type_life(firework,60,120);
```

9. Type in the following code for the flare particle:

```
//create flare particle
flare = part_type_create();
part_type_shape(flare,pt_shape_flare);
part_type_size(flare,1,1,0,0.50);
part_type_scale(flare,0.5,0.5);
part_type_color1(flare,8454143);
part_type_alpha2(flare,0,1);
part_type_speed(flare,7,9,0,0);
part_type_direction(flare,50,130,0,0);
part_type_gravity(flare,0.1,270);
part_type_orientation(flare,0,0,0,0,1);
part_type_blend(flare,1);
part_type_life(flare,50,70);
```

You just coded three different particles into a single Create Event using functions I cover earlier in this chapter. One of the particles is going to be used as a carrier for the other two, which will take advantage of two special functions that can make your special effects truly special.

10. Type the code for the special functions:

```
//special functions
part_type_step(flare,10,smoke);
part_type_death(flare,100,firework);
```

These two functions are used to tell a particle to create another particle every step that it exists, and to create another particle when its lifetime is over. So, in the context of this firework, these lines of code tell the flare particle to generate 10 smoke particles every step and then, at the end of its life, it creates 100 firework particles.

These functions can quickly create many thousands of particles or even an infinite loop of particle creation, which can slow down and crash your game, so take great care with these!





The firework particle uses a custom Sprite for the particle and not one of the built-in constants. You may find it very useful to use your own images to animate particles or to have random images for each particle (as is the case here).

11. Type in the code for the emitter:

```
emitter1 = part_emitter_create(Sname);
part_emitter_region(Sname,emitter1,220,420,400,400, ps_shape_rectangle,
ps_distr_linear);
```

12. Click the green check mark.

The code window is saved and closed.

13. Select the Step Event from the Events section.

The Actions for the Step Event appear in the Actions section.

14. Double-click the Execute Code Action from the Actions section.

The code window appears.

15. Update the code to match the following:

```
if instance_position(mouse_x, mouse_y, id)
{
if image_index != 2
{
image_index = 1;
if mouse_check_button_pressed(mb_left)
{
alarm[0] = room_speed / 2;
image_index = 2;
part_emitter_burst(Sname, emitter1, flare, 3);
}
}
}
else
{
if image_index != 2
{
image_index = 0;
}
}
}
```

16. Click the green check mark.

The code window is saved and closed.

17. Choose Add Event → Other → Room End.

The Room End Event appears in the Events section.

18. Drag and drop an Execute Code Action from the Control tab to the Actions section.

A code window appears.

19. In the code window, type the following code:

```
part_type_destroy(flare);  
part_type_destroy(smoke);  
part_type_destroy(firework);  
part_emitter_destroy(Sname, emitter1);  
part_system_destroy(Sname);
```

This code destroys the particles and the particle system to help prevent your game from crashing or slowing down due to many particles in the Room at once.

20. Click the green check mark.

The code window is saved and closed.

Go ahead and add the button for this complex firework effect into the Room (see Chapter 7) and watch a rather spectacular display that is only partly captured in Figure 8-12.



Figure 8-12: The complex code effect.



In this chapter, I show you several different ways to create special effects for your game. To further help you create impressive visual displays, *particle editors* are available from people who use GameMaker: Studio and like to contribute to the community. These particle editors are text files that contain the necessary code, which you can copy and paste into an Execute Code Action. Two of them are

- ✓ **Particle Designer 2.5 by Alert Games:** <http://gmc.yoyogames.com/index.php?showtopic=290449>
- ✓ **Sparks by Nocturne Games:** <http://gmc.yoyogames.com/index.php?showtopic=539511>

You can also search the Internet for more particle editors.



Be aware that particles in HTML5 are not quite as fast as they are in Windows. Also, if the player does not have WebGL switched on in her browser, she won't have color blending. Finally, on Android/iOS devices, particles can slow down games if they involve too much overdraw of the screen.

Using Surfaces to Create Effects

In This Chapter

- ▶ Working with the Surface function
- ▶ Creating shadows
- ▶ Placing decals into the game
- ▶ Using lighting effects
- ▶ Using Views with Surfaces

In GameMaker: Studio, a *Surface* provides you another layer, so to speak, on which you can draw. So far in this book, I've only discussed using Sprites and background colors to create a background effect in a Room (see Chapter 2). Essentially, by using Surfaces, you can draw on top of your background (the screen) if you want to add effects during the game.

For example, say you have a shooter game in which the player can toss grenades. When the player tosses the grenade, an explosion occurs. When the explosion occurs, you want a portion of the screen to appear damaged. You create the damage effect by using Surfaces, in that the damage appears on the screen after the grenade explodes. Other uses for Surfaces include adding shadows and lights to complete the look and feel of your game.

Note: This chapter dives into using the GameMaker Language. Creating Surfaces requires writing code. The code examples provided in this chapter are intended to get you started with the possibilities of what you can do with Surfaces.

Understanding the Surface Function

In Chapters 3 and 4, I discuss Events and Actions, including Events and Actions for drawing. The basic drawing function draws straight to the screen. That is, if you have a `draw_circle` function in the Draw Event, you see a circle appear on the screen. If you want to draw an effect on top of the

screen, you need to change the drawing target to be a Surface. Then all subsequent drawing will be drawn on the Surface layer instead of the screen. To change the drawing target, you can use `draw_surface()` functions; these functions also determine when the effect will be drawn.

The application Surface

GameMaker draws the whole game to a Surface, known as the *application Surface*. The application Surface is the only Surface that is not volatile and will always exist while your game is running. However, you can create your own Surface, which I discuss in this chapter.

You want to keep your Surface drawings small — the bigger they are, the more memory they'll use. And the more memory you use, the greater the odds that your game will lag or even crash. To make Surfaces even more complex, GameMaker stores Surfaces as a power of two textures, so that a 640-x-480-pixel Surface is treated as a 1,024-x-512-pixel texture in memory. Try to keep your Surface sizes small so as not to use all the VRAM of the player's computer or device.



The default position for Surfaces is always at the (0,0) coordinate within the Room. If you have a large Room with Views, or if you only want a small Surface for a special effect, you need to specify the drawing coordinates, offset within the Room (as discussed in the following procedures).

Creating Shadows with Surfaces

This chapter uses examples from the in-software tutorial, *Surfaces_Part 1*. Before you start working on creating a Surface, take a moment to become familiar with the tutorial and how it works. In the game, you can use the arrow keys to move and the mouse to aim and shoot. The right mouse button throws a grenade.

To open the tutorial, from the New Project window, click the Tutorials tab. An expandable list of tutorials appears on the left. Click to open a list of Intermediate tutorials and then choose *Surfaces_Part 1*.

While playing the game, you may have noticed that the player and the enemies have subtle shadows. The following procedures show you how to add shadows to the walls to create a more 3D effect. The following procedure does not use Surfaces, but it does provide an example of how to add a shadow effect to Instances. Then I step you through how to improve this effect with Surfaces.

To add shadows to the walls, follow these steps:

1. **With the Surfaces tutorial open, from the Resource tree, in the Objects folder, double click obj_Wall.**

The Object Properties appear.

2. **Choose Add Event⇨Draw⇨Draw.**

The Draw Event appears in the Events section.

3. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

4. **In the code window, type the following:**

```
draw_sprite_ext(sprite_index, 0, x - 16, y + 16, 1, 1, 0, c_black, 0.5);  
draw_self();
```

5. **Click the green check mark.**

The code window is saved and closed.

If you run the game now, you should see a very basic shadow effect on the walls, as shown in Figure 9-1. To run the game, from the main menu, choose Run⇨Run Normally.

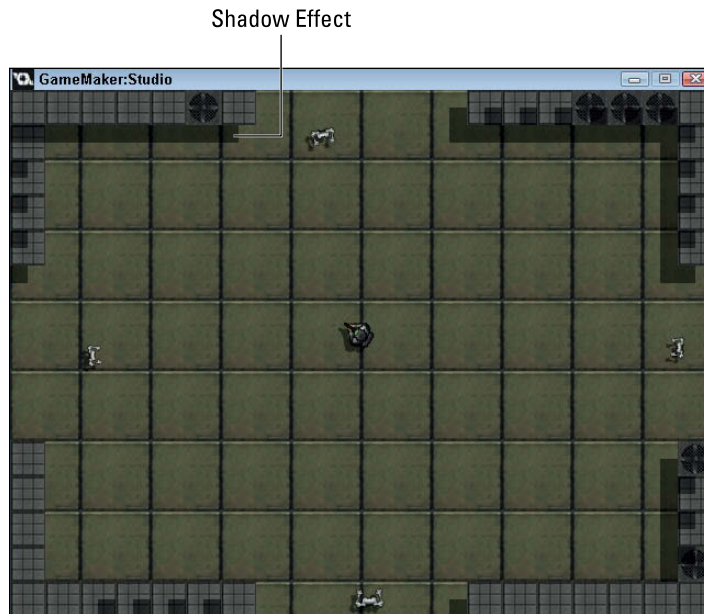


Figure 9-1: A basic shadow effect.

The problem here is that shadows don't tend to be square with rigid lines. To fix this, go ahead and delete the Draw Event you just created for obj_Wall and proceed to the next procedure. To delete the Draw Event, right-click over the Event and choose Delete Event. Go ahead and close obj_Wall by clicking OK.

The following procedures show you how to create a Surface to draw shadows for the walls.

To create a new Object for the Surface effect, follow these steps:

1. **With the Surfaces tutorial open, from the Resource tree, right-click Objects and select Create Object.**

The Object Properties appear.

2. **In the Name field, type obj_Shadow_Surface.**

3. **In the Depth field, type -900.**

This Depth will draw the shadow under the wall Instances so that you won't ever see the shadow effect on or over the wall itself.

4. **Choose Add Event⇒Create.**

The Create Event appears in the Events section.

5. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

6. **In the code window, type the following:**

```
surf = -1;
```

This code creates a variable for the Surface but does not create the Surface itself.

7. **Click the green check mark.**

The code window is saved and closed.

You'll create the Surface effect in the Draw Event, which is triggered every step of the game.

First, though, the following procedure shows how to remove the Surface effect when you no longer need it.

1. **From the obj_Shadow_Surface Object Properties, choose Add Event⇒Other⇒Room End.**

The Room End Event appears in the Events section.

2. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

3. In the code window, type the following:

```
if surface_exists(surf)
{
    surface_free(surf);
}
```

4. Click the green check mark.

The code window is saved and closed.

This code frees up the memory that the Surface was using.



If the memory is not freed when the Surface is no longer needed, the game could develop a memory leak, which will eventually crash your game.

The next procedure has you add the Draw Event to check if the Surface effect exists, and if it doesn't, GameMaker creates a surface area and draws the effect.

To create the Draw Event for the Surface effect, follow these steps:

1. From the obj_Shadow_Surface window, choose Add Event⇨Draw⇨Draw.

The Draw Event appears in the Events section.

2. Drag and drop an Execute Code Action from the Control tab to the Actions section.

A code window appears.

3. In the code window, type the code in Figure 9-2.

The following code checks if the Surface exists.

```
if surface_exists(surf)
```

The following code is triggered if the Surface does exist. If the Surface doesn't exist, GameMaker creates a Surface the size of the Room and assigns the unique ID, or index, of the Surface to the variable; that code is shown below.

```
{
    draw_surface(surf, 0, 0); //the surface exists, so draw it
}
```

The following code sets the drawing target to the surface instead of the screen. The Surface is then cleared to draw something new (more information on this follows the procedure).

```
else
{
    surf = surface_create(room_width, room_height);
    surface_set_target(surf)
    draw_clear_alpha(c_black, 0);
}
```

The following code draws the walls, in black, at different positions thanks to the incrementing value of *i* (that's where the `with (obj_Wall)` function comes into play) and with a low alpha onto the Surface.

```
draw_set_blend_mode_ext(bm_one, bm_inv_src_alpha);

var a = 1 / 16;
for (var i = 1; i < 32; i++ ;)
{
    with (obj_Wall)
    {
        draw_sprite_ext(sprite_index, image_index, x - i, y + i, 1, 1, 0,
            c_black, a);
    }
}

draw_set_blend_mode(bm_normal);
```

The following code resets the drawing target so that any other Draw Events will draw to the screen. If you don't do this, you won't see anything when you draw the Surface.

```
surface_reset_target();
}
```

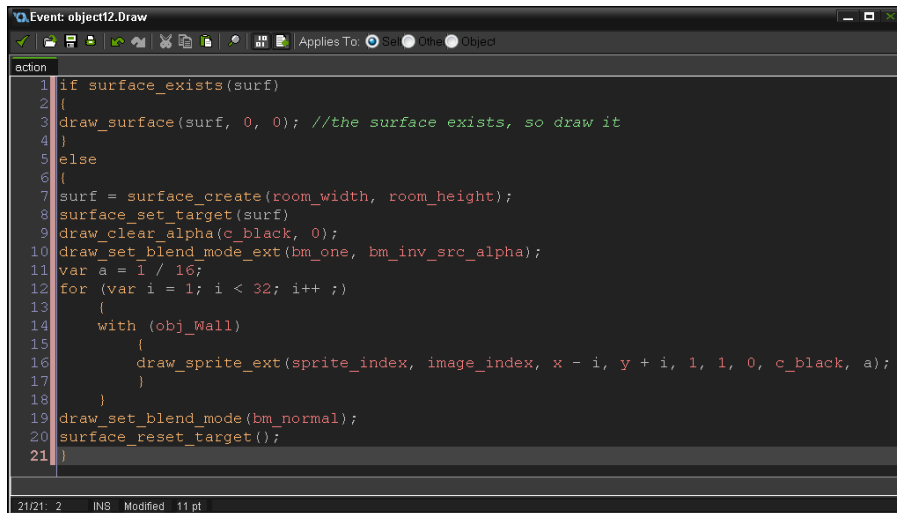


Figure 9-2: Code for drawing the shadow effect to the Surface.

4. Click the green check mark.

The code window is saved and closed.

5. Click OK to save and close the Object Properties.



The code from this procedure has you clear the Surface before drawing. What this means is, you're really clearing the VRAM to make space for the Surface code. You want to clear the VRAM of any other existing code because that code could appear when you draw to the Surface. This code also uses a blend mode, which I discuss in more detail in Chapter 8.

Now you need to add an Instance of your shadow Object to the Room. To add an Instance of the Object to the Room, follow these steps:

1. **With the Surfaces tutorial open, from the Resource tree, in the Rooms folder, double-click room0.**

The Room Properties appear.

2. **Click the Objects tab.**

3. **Click the Object to add drop-down and select obj_Shadow_Surface.**

The name of the Object appears in the field.

4. **Click within the Room.**

An Instance of the Object appears in the Room.

For more information on adding Instances of Objects to a Room, see Chapter 2.

If you run your game now, you should see a nice gradient for the shadow effect, as shown in Figure 9-3.



Figure 9-3: A gradient shadow effect on the walls.

Notice how there is no shadow effect on the walls in the bottom-left corner in Figure 9-3. That's because the shadow is drawn at a depth below the walls, so those shadows don't appear in the game (because the shadow is drawn as if the light source originates from the top right and shines at an angle toward the bottom left).



Using Surfaces to create a shadow effect eliminates the need to create multiple wall Objects and Instances, each with a different Sprite showing a different angle of shadow.

In the next section, I show you how to expand the procedures on creating a shadow effect to create a debris effect.

Using Decals with Surfaces

In the Surfaces tutorial game, there are Instances called *decals*. Decals can provide graphical effects but don't interfere with the gameplay. Just as with Objects and Instances, you want to keep the number of decals that appear in the game to a minimum; otherwise, your game may run very slowly and will eventually crash. In this section, I show you how to draw decals on the Surface to help stop your game from slowing down or having issues.

The following procedure picks up from the in-software Surfaces tutorial. To start, you create an Object that draws the Sprites for the decals that will last a very short amount of time, before being destroyed. This will allow the game to appear to have a million Instances, but it will really have very few.

To draw decals on the Surface, follow these steps:

1. **With the Surfaces tutorial open, from the Resource tree, right-click Objects and select Create Object.**

The Object Properties appear.

2. **In the Name field, type obj_Decal_Surface.**

3. **Choose Add Event → Create.**

The Create Event appears in the Events section.

4. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

5. **In the code window, type the following:**

```
surf = -1;
```

This code creates a variable for the Surface.

6. Click the green check mark.

The code window is saved and closed.

To free up the memory for when you no longer want the decal effect, follow these steps:

1. From the obj_Decal_Surface Object Properties, choose Add Event⇨Other⇨Room End.

The Room End Event appears in the Events section.

2. Drag and drop an Execute Code Action from the Control tab to the Actions section.

A code window appears.

3. In the code window, type the following:

```
if surface_exists(surf)
{
    surface_free(surf);
}
```

4. Click the green check mark.

The code window is saved and closed.

The code here helps with memory issues. The next procedure has you write the code to make the decals appear. To draw the decals, follow these steps:

1. From the obj_Decal_Surface window, choose Add Event⇨Draw⇨Draw.

The Draw Event appears in the Events section.

2. Drag and drop an Execute Code Action from the Control tab to the Actions section.

A code window appears.

3. In the code window, type the following (see Figure 9-4):

```
if surface_exists(surf)
{
    draw_surface(surf, 0, 0)
    if instance_exists(obj_Decal)
    {
        surface_set_target(surf);
        with (obj_Decal)
        {
            if speed < 0.1
            {
```

```

        if dist > 0 draw_sprite_ext(sprite_index, image_index, x -
        dist,y + dist, image_xscale, image_yscale, image_angle,
        c_black, 0.3);
        draw_sprite_ext(sprite_index, image_index, x, y, image_xscale,
        image_yscale, image_angle, c_white, 1);
        instance_destroy();
    }
    surface_reset_target();
}
else
{
    surf = surface_create(room_width, room_height);
    surface_set_target(surf)
    draw_clear_alpha(c_black, 0);
    surface_reset_target();
}

```

```

Event: object13.Draw
1 if surface_exists(surf)
2 {
3 draw surface(surf, 0, 0)
4 if instance_exists(obj_Decal)
5 {
6 surface_set_target(surf);
7 with (obj_Decal)
8 {
9 if speed < 0.1
10 {
11 if dist > 0 draw_sprite_ext(sprite_index, image_index, x - dist,y + dist,
12 image_xscale, image_yscale, image_angle, c_black, 0.3);
13 draw_sprite_ext(sprite_index, image_index, x, y, image_xscale, image_yscale,
14 image_angle, c_white, 1);
15 instance_destroy();
16 }
17 }
18 surface_reset_target();
19 }
20 }
21 else
22 {
23 surf = surface_create(room_width, room_height);
24 surface_set_target(surf)
25 draw_clear_alpha(c_black, 0);
26 surface_reset_target();
27 }

```

Figure 9-4: Code to display the decals on the Surface.

4. Click the green check mark.

The code window is saved and closed.

This code is very similar to the code from the preceding section. One difference in this code, though, is that the code now draws to the Surface every step of the game when an Instance of `obj_Decal_Surface` exists, rather than when the effect is first created.

Go ahead and place an Instance of `obj_Decal_Surface` in the Room. For a reminder on how to do that, see earlier in this chapter or Chapter 2. Now, when you play the game, note how much debris is left behind when the player blows up the enemy. Also note how much longer that debris stays in view as compared to before without the code.

Lighting Effects with Surfaces

This section continues with the in-software tutorial on Surfaces. Here, I step you through how to add lighting effects to your game using Surfaces. In the following procedures, you see how to mix additive blending on Sprites and how to use subtractive blending on a Surface, to achieve a lighting effect. To get started, though, I show you how to create a lighting effect without using Surfaces, as shown in Figure 9-5.

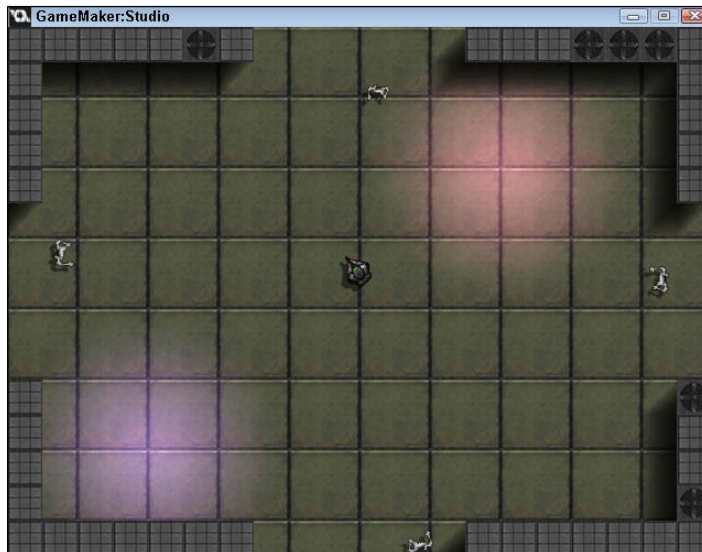


Figure 9-5: Basic lighting effects.

To create the Object for a basic light effect, follow these steps:

1. **With the Surfaces tutorial open, from the Resource tree, in the Objects section, right-click Objects and choose Create Object.**

The Object Properties appear.

2. **In the Name field, type obj_Light.**
3. **From the Sprite drop-down list, select spr_Round_Light.**

The thumbnail and name of the Sprite appear.

4. **In the Depth field, type -1600.**

This effect should be drawn on top of other Instances.

5. **Click Add Event⇨Create.**

The Create Event appears in the Events section.

6. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

7. **In the code window, type the following:**

```
image_xscale = 2;  
image_yscale = 2;  
image_alpha = 0.5;  
image_blend = make_color_hsv(random(255), 128, 255);
```

8. **Click the green check mark.**

The code window is saved and closed.

This code doubles the size of the Sprite and sets the alpha at 0.5, which is a relatively low value, so the light won't be too bright. The last line of code blends in a random color to provide a little something special to the effect, which is purely optional.

To create the Draw Event for the basic light effect, follow these steps:

1. **From the obj_Light properties window, choose Add Event⇨Draw⇨Draw.**

The Draw Event appears in the Events section.

2. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

3. In the code window, type the following:

```
draw_set_blend_mode(bm_add);
draw_self();
draw_set_blend_mode(bm_normal);
```

4. Click the green check mark.

The code window is saved and closed.

This code draws the Sprite at its position within the Room using an additive blend mode. Additive blend modes work great for when you want a bright effect, such as explosions and lighting.

Go ahead and place a couple Instances of `obj_Light` in the Room, as shown in Figure 9-6. For a reminder on how to place an Instance in the Room, see Chapter 2 or earlier in this chapter.

Note: In the Room, the Instances appear white, but in the game they appear colored. That's because you set the code to have the light effect appear with a random color.

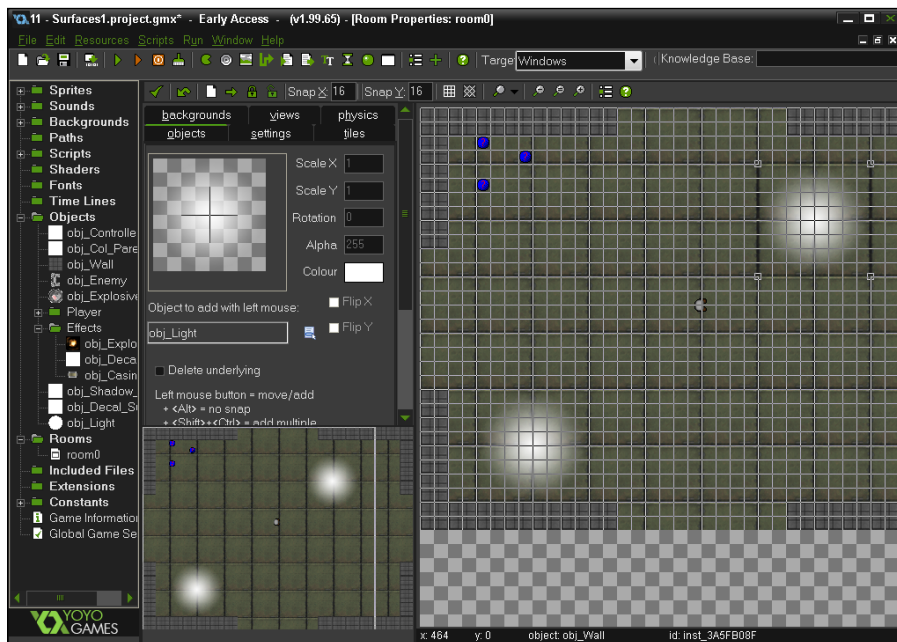


Figure 9-6: Instances of the light Object in the Room.

At this point, the light effect looks pretty cool, but you can make it look cooler. For example, in the following procedure you create some darkness to offset a light effect using a Surface.

Creating a flashlight effect

The next procedure shows how to create a flashlight effect using a Sprite. The Object you create for the flashlight will be a child Object that inherits properties from the parent `obj_Light`, which you created earlier in this chapter. This section continues from the in-software tutorial, *Surfaces*.

To create the Object for the flashlight effect, follow these steps:

1. **With the Surfaces tutorial open, from the Resource tree, right-click Objects and select Create Object.**

The Object Properties appear.

2. **In the Name field, type `obj_Player_Light`.**

3. **From the Sprite drop-down list, select `spr_Player_Light`.**

The thumbnail and name of the Sprite appear.

4. **In the Depth field, type `-1600`.**

This Depth will make sure this effect appears over other Instances.

5. **From the Parent drop-down list, select `obj_Light`.**

See Chapters 2 and 12 for more information on parent and child Objects.

This Object will now inherit the properties of the parent Object, in this case, `obj_Light`.

6. **Choose Add Event→Create.**

The Create Event appears in the Events section.

7. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

8. **In the code window, type the following:**

```
image_xscale = 1.5;  
image_yscale = 1.5;  
image_alpha = 0.5;
```

9. **Click the green check mark.**

The code window is saved and closed.

10. Choose Add Event⇄Step⇄Step.

The Step Event appears in the Events section.

11. Drag and drop an Execute Code from the Control tab to the Actions section.

A code window appears.

12. In the code window, type the following:

```
if instance_exists(obj_Player)
{
x = obj_Player.x;
y = obj_Player.y;
image_angle = obj_Player.b_angle;
}
else instance_destroy();
```

13. Click the green check mark.

The code window is saved and closed.

14. Click OK.

The Object Properties are saved and closed.

This code makes the light (represented by a Sprite) follow the player Object and always point in the direction the player is pointing (aiming); this is handled by the `b_angle` portion of the code. Now you need to add a Create Event in `obj_Player` so that an Instance of `obj_Player_Light` is placed in the Room.

To create the light effect, follow these steps:

1. From the Resource tree, in the Player folder within the Objects folder, double-click `obj_Player`.

The Object Properties appear.

2. From the Events section, with the Create Event highlighted, drag and drop a Create Instance Action from the Main1 tab to the Actions section.

The Create Instance properties appear.

3. From the Object drop-down list, select `obj_Player_Light`.**4. Select the Relative check box.**

Now the light will be created at the same X and Y coordinates as the player.

5. Click OK.

The Create Instance properties are saved and closed.

If you test the game now, you see the flashlight effect, as shown in Figure 9-7.

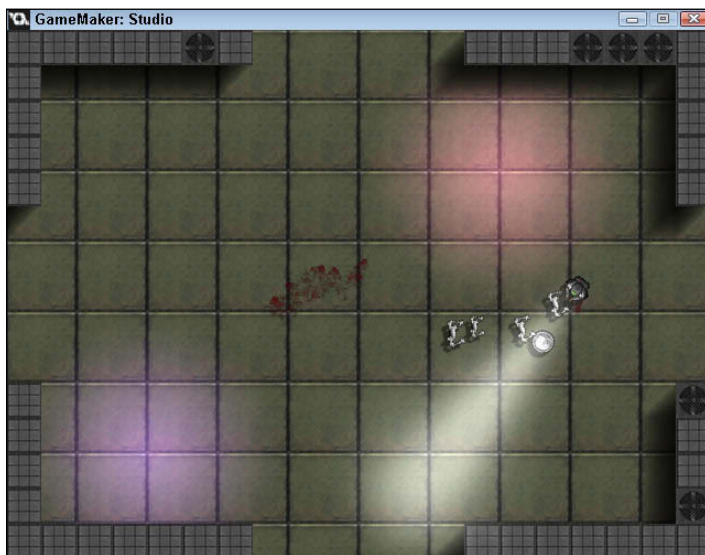


Figure 9-7: The flashlight effect using Surfaces.

To make this flashlight effect even more effective, you can add some darkness to the area surrounding the flashlight. I show you how to do this in the next section.

Adding darkness to the Surface

Earlier in this chapter, I walk you through a couple of different exercises to create some lighting effects, including stationary lights that display a random color each time the game is played and a flashlight effect that follows the player. Here, I show you how to add darkness to the Surface to make the light effects really stand out.

The following procedures continue from the in-software tutorial, *Surfaces_Part1*. In the next procedure, the Create Event and the Room End Event are the same as in earlier procedures, so if you haven't completed the earlier sections, you can flip back for details and explanations.

To create an Object for the darkness, follow these steps:

1. **With the Surfaces tutorial open, from the Resource tree, right-click Objects and select Create Object.**

The Object Properties appear.

2. **In the Name field, type obj_Light_Surface.**

3. **In the Depth field, type -1600.**

4. **Choose Add Event→Create.**

The Create Event appears in the Events section.

5. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

6. **In the code window, type the following:**

```
surf = -1;
```

7. **Click the green check mark.**

The code window is saved and closed.

8. **Choose Add Event→Other→Room End.**

The Room End Event appears in the Events section.

9. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

10. **In the code window, type the following:**

```
if surface_exists(surf)
{
    surface_free(surf);
}
```

11. **Click the green check mark.**

The code window is saved and closed.

12. **Choose Add Event→Draw→Draw.**

The Draw Event appears in the Events section.

13. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

14. In the code window, type the code in Figure 9-8.

The following code sets the Surface target.

```
if surface_exists(surf)
```

The following code clears the surface to black, but with an alpha of 0.85 (you can change this value to achieve different degrees of darkness).

```
{
    surface_set_target(surf);
    draw_clear_alpha(c_black, 0.85);
}
```

The following code sets the blend mode to subtractive blending; this takes the luminosity of what's drawn and sets the alpha relative to the brightness of individual pixels.

```
draw_set_blend_mode(bm_subtract);
```

The following code goes through the light Objects and draws them. This punches a “hole” in the darkness thanks to the subtractive blending.

```
with (obj_Light)
{
    draw_sprite_ext(sprite_index, image_index, x / 2, y / 2, image_yscale,
        image_xscale, image_angle, c_white, 1);
}
draw_set_blend_mode(bm_normal);
```

The following code resets the drawing target to the screen.

```
surface_reset_target();
```

The following code draws the surface, scaled to the size of the screen.

```
draw_surface_ext(surf, 0, 0, 2, 2, 0, c_white, 1);
}
else
{
    surf = surface_create(room_width / 2, room_height / 2);
}
```

15. Click the green check mark.

The code window is saved and closed.

In the preceding code, you've created the Surface at half the Room width and height. The main reason for doing this is because the game is starting to use a few different Surface effects, and that eats up memory. By reducing the Surface to half the size of the Room, you're reducing the amount of memory needed for storage. However, the darkness effect will still appear in the entire Room because of the `draw_surface_ext()` function. That function scales the effect to fit the Room.

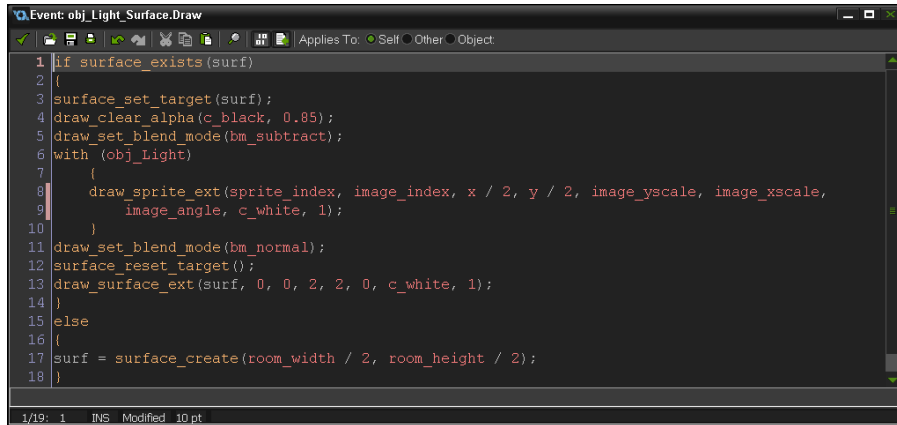


Figure 9-8: Code for the darkness effect.

Now you can add an Instance of `obj_Light_Surface` to the Room. For instructions on how to do that, see earlier in this chapter or Chapter 2. You should now have a darkness effect, as shown in Figure 9-9, which highlights the light effects you created earlier in this chapter.



Figure 9-9: The darkness effect highlights the light effects.

Feel free to change the values for the different effects — such as scaling down the brightness of the lights, or making the darkness darker, and so on. You can fine-tune and customize the game to your liking. At this point, your game is looking more and more like a survival horror game!

Using Surfaces as a View

In this section, I show you how to create special effects that affect the entire screen. This is a radical departure from earlier in this chapter, where the effects you made with Surfaces had you drawing to the Surface to create a new visual effect. Now, you create effects that appear across the entire screen. You achieve this by assigning a View to a Surface.

Setting up the Room and an Object for Surface Views

Basically, for this effect to work, you tell GameMaker to draw to a Surface the visible parts of a View, instead of drawing the View to the screen. You can tell GameMaker to draw the whole screen or a portion of the screen to the Surface every step of the game. When you do this, you need to have two Views active simultaneously. If you don't, then when the Surface is not drawn, the screen appears black, which is why you need the second View to display the screen.



Think of it this way: The first View appears initially, but when an Event is triggered — say, a shock wave rocks the player — the Surface effect kicks in and displays the Surface View with the shock wave effect. When the shock wave effect is finished, then the first View appears again.

The following procedure picks up with the in-software tutorial, Surfaces_Part2, found under the Advanced section of the Tutorials tab from the New Project window. To start, you set up the tutorial game to work with two different Views.

To set up two different Views, follow these steps:

1. **With the Surfaces tutorial open, from the Resource tree, double-click room0.**

The Room Properties appear.

2. **Click the Views tab.**

3. **Select the Enable the Use of Views check box (see Figure 9-10).**

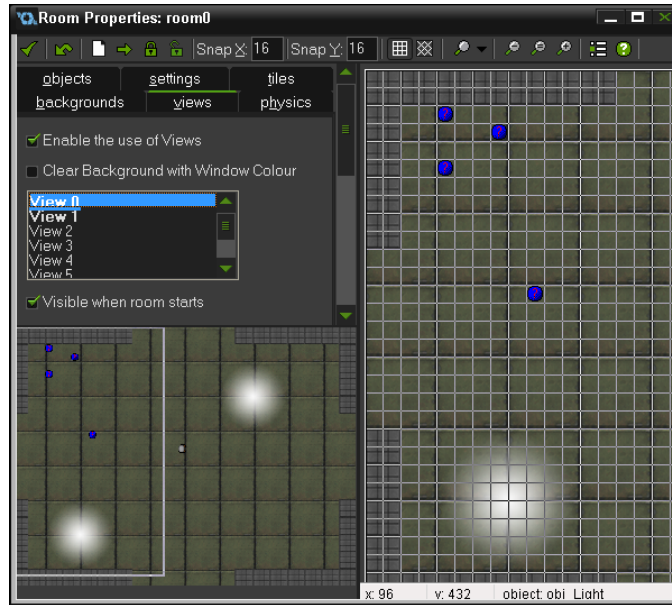


Figure 9-10: The Room Properties with Views enabled and visible.

4. **Select View 0 and make sure Visible When Room Starts is selected.**
5. **Click to select View 1 and make sure Visible When Room Starts is selected.**
6. **Click the green check mark.**

The Room Properties window is saved and closed.

With the Views set up, it's time to create a new Surface Object. This Surface Object checks to see if the screen surface exists, and if so, it holds the values of the screen.

To create the Surface Object, follow these steps:

1. **From the Resource tree, right-click Objects and select Create Object.**
The Object Properties appear.
2. **In the Name field, type `obj_Effects_Surface`.**
3. **Choose Add Event → Create.**

The Create Event appears in the Events section.

4. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

5. **In the code window, type the following:**

```
surf = -1;
```

6. **Click the green check mark.**

The code window is saved and closed.

7. **Choose Add Event→Other→Room End.**

The Room End Event appears in the Events section.

8. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

9. **In the code window, type the following:**

```
if surface_exists(surf)
{
    surface_free(surf);
}
```

10. **Click the green check mark.**

The code window is saved and closed.

11. **Choose Add Event→Draw→Draw.**

The Draw Event appears in the Events section.

12. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

13. **In the code window, type the following:**

```
if !surface_exists(surf)
{
    surf = surface_create(room_width,room_height);
    surface_set_target(surf);
    draw_clear_alpha(c_black,0);
    surface_reset_target();
    surface_reset_target();
    view_surface_id[0] = surf;
}
```

14. **Click the green check mark.**

The code window is saved and closed.

Much of the preceding code is covered in detail earlier in this chapter. However, the last line of this code block creates a variable that holds the Surface ID that GameMaker will draw to. Now, whenever `view[0]` is drawn, it will be drawn to the Surface and not the screen.

If you add this Object to the Room now, you won't see anything different, because you haven't created any effects for the Surface. You can do that in the next section.

Creating the Surface View effect

In the preceding section, you set up the Room and an Object so that you can have the View drawn on the Surface for the purpose of displaying effects. In this section, you create the effect that will be drawn on the Surface.

The following procedures continue with the in-software tutorial Surfaces_Part2. Here, you create a shock wave effect that is shown with every explosion in the game.

To create the Object for the shock wave effect, follow these steps:

1. **With the Surfaces tutorial open, from the Resource tree, right-click Objects and select Create Object.**

The Object Properties appear.

2. **In the Name field, type `obj_Shockring`.**

3. **In the Depth field, type `-800`.**

The shock wave effect appears over most of the Instances in the game, but not all of them.

4. **Choose Add Event → Create.**

The Create Event appears in the Events section.

5. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

6. **In the code window, type the following:**

```
ww = 50;           // width of the ring effect
radius = 0;        // size check variable
r_max = 120;       // maximum radius of the effect
spd = 1;           // expansion speed
d = 360 / 15;      // ring segments to be drawn

if instance_exists(obj_Effect_Surface)
{
    tex = surface_get_texture(obj_Effects_Surface.surf);
```

```
srf_w = surface_get_width(obj_Effects_Surface.surf);
srf_h = surface_get_height(obj_Effects_Surface.surf);
tex_w = texture_get_width(tex);
tex_g = texture_get_height(tex);
}
else instance_destroy();
```

7. Click the green check mark.

The code window is saved and closed.

This code checks that the Surface Object exists. Then GameMaker stores the Surface as a texture. The code also stores the width and height of the Surface in variables, as well as the texture width and height. This is necessary for the effect you're about to create, in which you create basic polygons with a texture. For this, you need variables for the texture as well as the width and height. These variables are different from the Surface width and height because they're always expressed as a value from 0 to 1. Next, you create the Draw Event.

To create the Draw Event for the shock wave effect, follow these steps:

1. With the **obj_Shockring Object Properties** open, choose **Add Event** → **Draw** → **Draw**.

The Draw Event appears in the Events section.

2. Drag and drop an **Execute Code Action** from the **Control** tab to the **Actions** section.

A code window appears.

3. In the code window, type the following:

```
if radius < r_max radius += spd else instance_destroy();

var o, c, xx, yy, a, sx, sy;

o = (-radius + r_max) / 5;           //Distortion offset
c = 0;                               //Multiplier for ring segments
a = 1 / (r_max / spd);               //Alpha modifier
sx = tex_w / srf_w;                  //Texture x coord
sy = tex_h / srf_h;                  //texture y coord

//Draw the inner part of the ring
draw_primitive_begin_texture(pr_trianglestrip, tex);
repeat(16)
{
    xx = x + lengthdir_x(radius, c * d);
    yy = y + lengthdir_y(radius, c * d);
    draw_vertex_texture_color(xx, yy, sx * (xx + lengthdir_x(o, c * d)),
        sy * (yy + lengthdir_y(o, c * d)), c_white, image_alpha);
    xx = x + lengthdir_x(radius - ww, c * d);
    yy = y + lengthdir_y(radius - ww, c * d);
    draw_vertex_texture_color(xx, yy, sx * xx, sy * yy, c_white, 0);
```



```

    c += 1;
  }
draw_primitive_end();
c = 0
//Draw the outer part of the ring
draw_primitive_begin_texture(pr_trianglestrip, tex);
repeat(16)
{
  xx = x + lengthdir_x(radius + ww, c * d);
  yy = y + lengthdir_y(radius + ww, c * d);
  draw_vertex_texture_color(xx, yy, sx * xx, sy * yy, c_white, 0);
  xx = x + lengthdir_x(radius, c * d);
  yy = y + lengthdir_y(radius, c * d);
  draw_vertex_texture_color(xx, yy, sx * (xx + lengthdir_x(o, c * d)),
    sy * (yy + lengthdir_y(o, c * d)), c_white, image_alpha);
  c += 1;
}
draw_primitive_end();
image_alpha -= a;

```

4. Click the green check mark.

The code window is saved and closed.

This code creates two separate rings from polygons. A polygon is the base block for almost all drawing and GameMaker (and most games) use polygons to draw graphics. Polygons are usually simple triangles, but here, you're creating a series of triangles to make a more complex shape. In this example, you're creating a 15-sided polygon to form a ring, as shown in Figure 9-11.



Even Sprites are drawn using two simple polygons to form a quad.

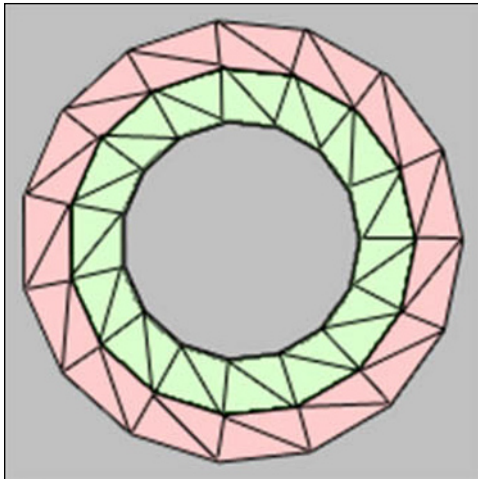


Figure 9-11: The polygon you created in the code block.

To finish this exercise, you have to create this effect in the Create Event of `obj_Explosion`.

To add the shock wave effect to the game, follow these steps.

1. With the `obj_Shockring` Object Properties open, click the **Create Event**.

The Actions for the Create Event appear in the Actions section.

2. From the **Actions** section, double-click the **Execute Code Action**.

The code window appears.

3. In the code window, type the following after the existing code:

```
instance_create(x, y, obj_Shockring);
```

4. Click the **green check mark**.

The code window is saved and closed.

The shock wave effect adds something special to when the barrels and grenades explode. The effect is shown in Figure 9-12.

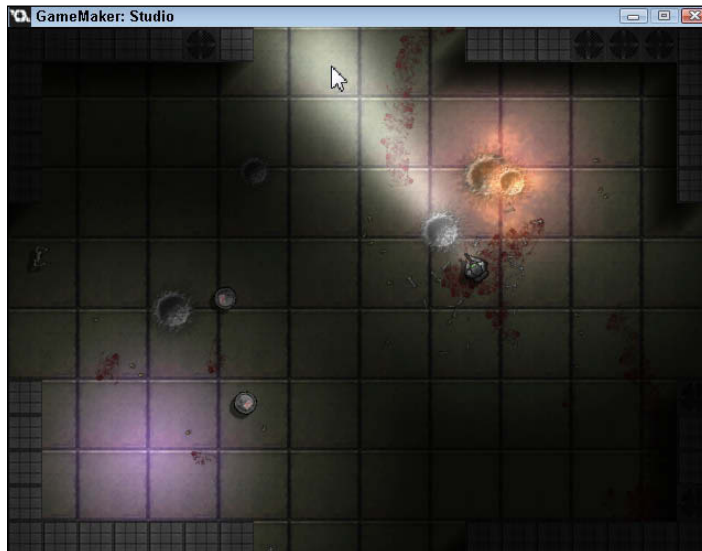


Figure 9-12: The shock wave effect from an explosion.

Applying Scaling

In This Chapter

- ▶ Working with the Global Game Settings
- ▶ Scaling to Windows
- ▶ Drawing GUI Elements
- ▶ Scaling for HTML5

In this chapter, I cover resolution and scaling. *Resolution* is all about pixels per inch — a resolution of 1,920 x 1,200 means that a screen has 1,920 pixels horizontally and 1,200 lines vertically. *Scaling* is all about matching your game's graphics to the actual size of the player's screen. This chapter gets you started writing code to make your games fit the screen, whether you're playing on a large monitor, a mobile phone, or a tablet.

This chapter explains how to code for proper resolution and scaling for different screen sizes for the Windows 7 platform, which works great on all desktops, including Mac and Linux. This provides a foundation to learn about additional platforms such as HTML5, iOS, Android, and others.

The procedures in this chapter work together with the in-software tutorial “The_Application_Surface_Scaling.” When creating the proper resolution and scaling for your games, you're manipulating the application surface. I cover working with the application surface in Chapter 9, and I recommend that you familiarize yourself with how the application surface works before continuing with this chapter. You can also review the tutorial “The_Application_Surface_Overview,” which you can find from the New Project window, under the Tutorials tab, in the Beginner section.

Note: You can scale your games for Windows 8 (JavaScript) and Tizen (JavaScript) to maintain the appropriate aspect ratio, but those platforms require a different set of functions that are not covered in this chapter.

Configuring the Global Game Settings

To get started, you should become familiar with the Global Game Settings window, shown in Figure 10-1. You can find the Global Game Settings window under the Resources menu.

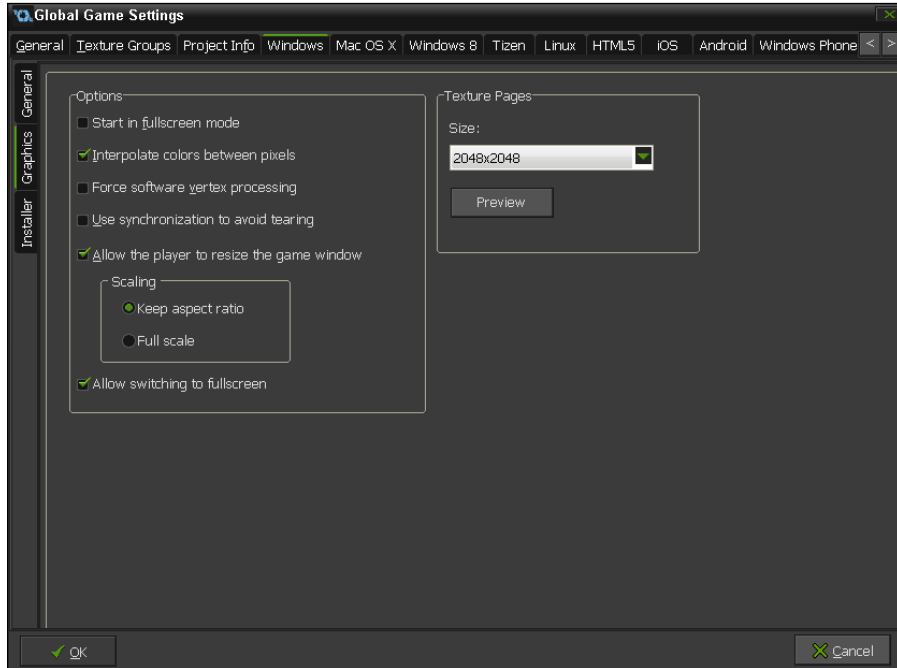


Figure 10-1: The Global Game Settings window.

To open the Global Game Settings window, follow these steps:

1. **Open the Application_Surface_Scaling tutorial from the New Project window.**

The project window appears.

2. **From the main menu, choose Resources → Change Global Game Settings.**

The Global Game Settings appear.

3. **Click the Windows tab.**

The Windows properties appear.



4. Along the left side of the window, click the Graphics tab.
5. From the Scaling section, select the Full Scale radio button.

In the Options section of the Windows Graphics tab, you see a Scaling section with two radio buttons. The default is Keep Aspect Ratio, which is what you'll eventually want for this chapter, but for now choose Full Scale so you can see the differences between the two settings.

GameMaker does its best to fit the game in the full window, without maintaining aspect ratio.

6. Click OK to save and close the Global Game Settings window.

Now is a good time to run the tutorial game to see what it initially looks like. To run the tutorial game, from the main menu, choose Run → Run Normally. At first, the game should have a proper aspect ratio — the game doesn't look stretched to fill the screen. However, if you resize the window, you'll find that the game loses the aspect ratio and appears stretched, as shown in Figure 10-2. To fix this, go back to the Global Game Settings and choose Keep Aspect Ratio (refer to Step 5 in the preceding list).

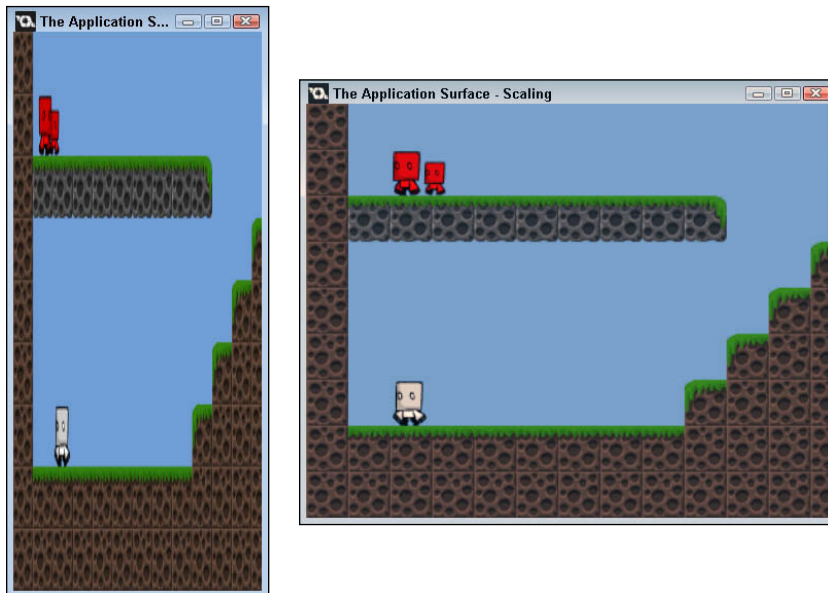


Figure 10-2: Improper aspect ratio makes the game look stretched.

The Full Scale option tells GameMaker to always scale the game to fit the current window size, which can lead to the stretching effect. When you select the Keep Aspect Ratio radio button, the aspect ratio remains correct but you get the letterbox effect (the black bars), as shown in Figure 10-3.

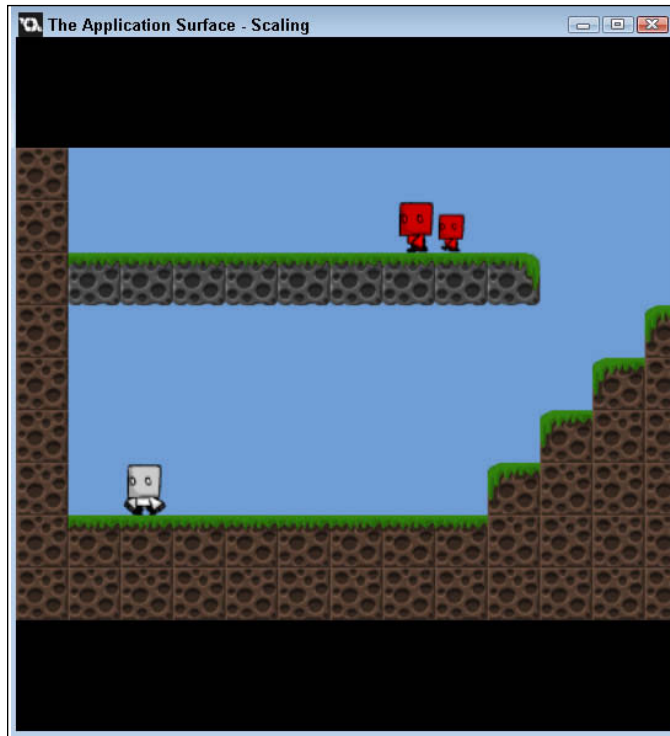


Figure 10-3: The letterbox effect with Keep Aspect Ratio selected.

The Keep Aspect Ratio setting works for all target platforms except HTML5 and is the simplest way to ensure that all your games will display correctly regardless of the device display or window size. The problem right now is those pesky black bars. In the next section, I show you how to cover up the black bars and maintain a proper aspect ratio.

Scaling to the Window

In the following procedures, I show you how to maintain a proper aspect ratio for your games without the letterbox effect appearing. You do this by manipulating the application surface. The initial size of the application surface is based on the size of the Room (if Views are not active) or the View port (if Views are active).

Scaling to best fit

By scaling the application surface to fit the player's window or display size as best as possible, GameMaker adjusts the aspect ratio accordingly as the player resizes the window or triggers full-screen mode. The following procedures show how to scale your game for the best fit for desktop games. Later sections show how to update the code for mobile devices.

The following procedure picks up from the last section with the `Application_Surface_Scaling` tutorial.

To update the controller Object for scaling to best fit, follow these steps:

- 1. With the tutorial open, from the Resource tree, click to open the Objects folder, then double-click `obj_Control`.**

The Object Properties appear.

- 2. Choose Add Event → Create.**

The Create Event appears in the Events section.

- 3. From the Control tab, drag and drop an Execute Code Action.**

The code window appears.

- 4. In the code window type the following code:**

```
application_surface_draw_enable(false);  
min_width = view_wview[0];  
min_height= view_hview[0];  
scr_Resize();
```

This code tells GameMaker that the controller Object will take over the drawing of the application surface. The code also sets variables for the base width and height when the player resizes the window. The last line of the code calls a Script, which you write next. Because you haven't written the Script yet, the last line of code will appear as an error in the code window, but that will fix itself after you write the Script.

- 5. Click the green check mark to save and close the code window.**

Next, you need to write the Script that you called in the last procedure.

- 1. From the Resource tree, right-click Scripts and choose Create Script.**

The Script code window appears.

- 2. In the Name field, type `scr_Resize`.**

3. In the Script code window, type the following:

```
cur_width = max(1, window_get_width());
cur_height = max(1, window_get_height());
var ratio = cur_width / cur_height;
if cur_width < cur_height
{
    view_wview[0] = min_height * ratio;
    view_hview[0] = min_height;
}
else
{
    view_wview[0] = min_width;
    view_hview[0] = min_width / ratio;
}
view_wport[0] = cur_width;
view_hport[0] = cur_height;
surface_resize(application_surface, view_wview[0], view_hview[0]);
```

4. Click the green check mark to save and close the Script code window.

This code gets the current window size and stores those values in the variables. Because windows can be minimized, which would result in values of 0 (and you can't divide by 0), you're using the `max` function to ensure that the variables always have a value of 1 or greater. The variables are then used to get the aspect ratio of the game window. After the aspect ratio is determined, GameMaker can use that to scale the game correctly, which is what the remainder of the code entails.

Now you need to tell the controller Object to detect any changes in the window size for each step of the game.

1. With the `obj_Control` still open from the previous procedure, choose `Add Event` ⇨ `Step` ⇨ `Step`.

The Step Event appears in the Events section.

2. From the Control tab, drag and drop an `Execute Code Action`.

The code window appears.

3. In the code window type the following:

```
if cur_width != window_get_width() || cur_height != window_get_height()
{
    scr_Resize();
}
```

This code detects any changes in the window size for each step of the game and adjusts the aspect ratio as necessary.

4. Click the green check mark to save and close.

Now you need to draw the application surface so that you don't wind up with a black screen:

1. Choose Add Event → Draw → PostDraw.

The Post Draw Event appears in the Events section.

2. From the Control tab, drag and drop an Execute Code Action.

The code window appears.

3. In the code window, type the following:

```
draw_surface_stretched(application_surface, 0, 0, cur_width, cur_height);
```

4. Click the green check mark to save and close the code window.

You can now run the game to test how the game looks in full-screen mode and by adjusting the width and height of the window. The game should now scale correctly with no black bars. To put the game in full-screen mode, press Alt+Enter. Press Alt+Enter a second time to leave full-screen mode.

Scaling the View

In the preceding section, I explain how to scale your game to fit the size of the window, no matter the dimensions of the window. In this section, I show you how to scale the View. When working with Views (see Chapter 5), you tell GameMaker to show only a certain portion of the Room at a time, which is how this tutorial is set up. However, if the player's window is big enough to show more of the View than what you set in the Room, you can update the Script from the last procedure.

1. With the tutorial open from the last section, in the Resource tree, from the Scripts folder, double-click scr_Resize.

The Script code window appears.

2. In the code window, update the code to reflect the following (the code in bold indicates the differences):

```
cur_width = max(1, window_get_width());
cur_height = max(1, window_get_height());
var ratio = cur_width / cur_height;
if cur_width < cur_height
{
    view_wview[0] = median(min_height, cur_height, room_height);
    view_hview[0] = view_hview[0] * ratio;
}
else
```

```
{
    view_wview[0] = median(min_width, cur_width, room_height);
    view_hview[0] = view_wview[0] / ratio;
}
view_wport[0] = cur_width;
view_hport[0] = cur_height;
surface_resize(application_surface, view_wview[0], view_hview[0]);
```

3. Click the green check mark to save and close the Script code window.

GameMaker displays as much of the Room as it can within the window while maintaining the appropriate aspect ratio. You can press the spacebar to change Rooms to see how the window and view adapts to different Room sizes. You may have noticed in the Resource tree that there are two Rooms set up for this tutorial, room0 and room1. When you press the spacebar and switch to room1, you should see much more of the View than you did before.

Scaling on mobile devices

For mobile devices, such as tablets and phones, you're configuring the full display size instead of a window (see the preceding section). Because of this, you only have to deal with a single aspect ratio, unless you allow for landscape and portrait settings from the Global Game Settings window.

The following procedure picks up from the last section and has you update the Script and controller Object:

1. From the Resource tree, in the Objects folder, double-click obj_Control (if it's not open from earlier).

The Object Properties appear.

2. Click the Step Event from the Events section.

The Actions for the Step Event appear in the Actions section.

3. Double-click the Execute Code Action from the Actions section.

The code window appears.

4. In the code window, type the following (the code in bold indicates the original code):

```
switch(os_type)
{
    case os_macosx:
    case os_windows:
    case os_linux:
    case os_win8native:
        if cur_width != window_get_width() || cur_height !=
            window_get_height()
```

```
        {
            scr_Resize();
        }
        break;
    case os_ios:
    case os_android:
    case os_winphone:
    case os_tizen:
        if cur_width != display_get_width() || cur_height != display_get_
            height()
        {
            scr_Resize();
        }
        break;
    }
```

This code checks the operating system of the device running the game.

5. Click the green check mark to save and close the code window.

Now you need to update the Script.

1. From the Resource tree, in the Scripts folder, double-click scr_Resize.

The Script code window appears.

2. Replace the first two lines of code with the following:

```
switch(os_type)
{
    case os_macosx:
    case os_windows:
    case os_linux:
    case os_win8native:
        cur_width = max(1, display_get_width());
        cur_height = max(1, display_get_height());
        break;
    case os_ios:
    case os_android:
    case os_winphone:
    case os_tizen:
        cur_width = max(1, display_get_width());
        cur_height = max(1, window_get_height());
        break;
}
```

3. Click the green check mark to save and close the code window.

Now when you run the game on any mobile device, you should see that the game scales correctly no matter the resolution of the device or if you change the orientation from landscape to portrait. You should also notice that more of the Room is displayed in landscape mode rather than portrait.

Scaling with Draw GUI Events

An important thing to note when configuring your game for a fixed aspect ratio (or if you're positioning the application layer manually) is that the graphical user interface (GUI) layer is normally letterboxed along with the application surface. This makes things very easy to position, as the top-left corner of the GUI layer is the (0,0) position of the application surface and the width and height are scaled by default to fit the application surface, too.

However, this may not be what you want. Fortunately, GameMaker enables you to set the GUI layer to cover the whole window or display, regardless of where the application surface is being drawn, permitting you to override the letterbox effect and draw over the whole area.

Maximizing the GUI

Basically, the GUI layer is dealt with apart from the general Draw Events and has its own resolution and positioning, based on the relative position (0,0) of the application surface. If you choose not to resize the GUI layer, it will be 1:1 with the application surface, but in many games you may want to resize this to better fit heads-up display (HUD) elements, such as virtual keys, ammo count, health status, and so on.

In the following procedures, I show you how to maximize the GUI layer and set it to 1.5 times the size of the View. Then, in the next section, I show you how to hide the letterbox effect. I also show you how to position GUI elements so that they appear over the application surface and not over the letterbox areas. This method can be more complex than leaving the default positioning, but it permits more dynamic GUI elements that change position based on the size of the window or display.

This procedure continues with “The_Application_Surface_Scaling” tutorial. Because of this, you do need to backtrack and remove all the Events from the control Object except for the Keyboard Event as described in the following procedure:

- 1. With obj_Control open from the previous sections, right-click each Event one at a time and choose Delete Event.**

Only the Press <Space> Event and Create Event should appear in the Events section.

- 2. Click to select the Create Event.**

The Create Event Actions appear in the Actions section.

- 3. Double-click the Execute Code Action.**

The code window appears.

4. Update the code to only show the following:

```
display_set_gui_maximise();
```

This code tells GameMaker to start drawing the GUI layer from the (0,0) position of the display (or game window) rather than the position at which it draws the application surface.

5. Click the green check mark to save and close the code window.

Now you need to update the Global Game Settings:

1. From the main menu, choose Resources → Change Global Game Settings.

The Global Game Settings appear.

2. Click the Windows tab.**3. From the left side of the window, click the Graphics tab.****4. In the Options section, select the Start in Fullscreen Mode check box.****5. Make sure that Allow the Player to Resize the Game Window is selected.****6. In the Scaling section, make sure Keep Aspect Ratio is selected.****7. Directly below the Scaling section, make sure Allow Switching to Fullscreen is selected.****8. Click OK to save and close the Global Game Settings.**

Go ahead and run the game now. The game should start in full-screen mode. To exit full-screen mode, press Alt+Enter. If you resize the window, the game should maintain its aspect ratio. You may also notice that the letterbox effect is back. To remove the letterbox effect, move on to the next section.

Hiding the letterbox

The Keep Aspect Ratio option in the Global Game Settings window is a fantastic quick fix for mobile devices or full-screen desktop games. However, if you have the letterbox effect, the GUI elements will be drawn over them because they take all their coordinate positions from the (0,0) screen position.

This is where the `application_get_position()` function comes into play. This function returns an array of values indicating where on the screen the application surface is being drawn by default. You can use these values to define a relative position on the GUI layer, no matter the size of the GUI.

The following procedure picks up from The_Application_Surface_Scaling tutorial.

1. Open obj_Control from the Resource tree.

2. Choose Add Event→Draw→Draw GUI.

The Draw GUI Event appears in the Events section.

3. From the Control tab, drag and drop an Execute Code Action into the Actions section.

The code window appears.

4. In the code window, type the following:

```
draw_set_color(c_white);
var pos = application_get_position();
if pos[0] > 0
{
    var ratio = window_get_width() / window_get_height();
    display_set_gui_size((view_hview[0] * 1.5) * ratio, (view_hview[0] * 1.5));
}
else
{
    var ratio = window_get_height() / window_get_width();
    display_set_gui_size((view_wview * 1.5), (view_wview[0] * 1.5) * ratio);
}
var gui_w = display_get_gui_width();
var gui_h = display_get_gui_height();
switch(os_type)
{
    case os_windows:
    case os_macosx:
    case os_linux:
    case os_win8native:
        var win_w = window_get_width();
        var win_h = window_get_height();
        break;
    case os_android:
    case os_ios:
    case os_winphone:
    case os_tizen:
        var win_w = display_get_width();
        var win_h = display_get_height();
        break;
}
```

This code sets up the appropriate variables based on device type, and also sets the GUI layer to always be 1.5 times the size of the view. (Note that normally you would have a check in here to see if the screen has changed and only set the GUI layer if it has, but for this tutorial that isn't necessary.)

```
var x1 = (pos[0] / win_w) * gui_w;  
var x2 = (pos[2] / win_w) * gui_w;  
var y1 = (pos[1] / win_h) * gui_h;  
var y2 = (pos[3] / win_h) * gui_h;  
if pos[0] > 0  
{  
  draw_rectangle(0, 0, x1, gui_h, false);  
  draw_rectangle(x2, 0, gui_w, gui_h, false);  
}  
else  
{  
  draw_rectangle(0, 0, gui_w, y1, false);  
  draw_rectangle(0, y2, gui_w, gui_h, false);  
}
```

This code draws white rectangles to cover the letterbox.

5. Click the green check mark to save and close the code window.

If you test the game now, you should see white rectangles either on the top and bottom, or on the left and right, depending on how you size the window (see Figure 10-4). You can also change the Global Game Settings to landscape and portrait to see how that affects your game.

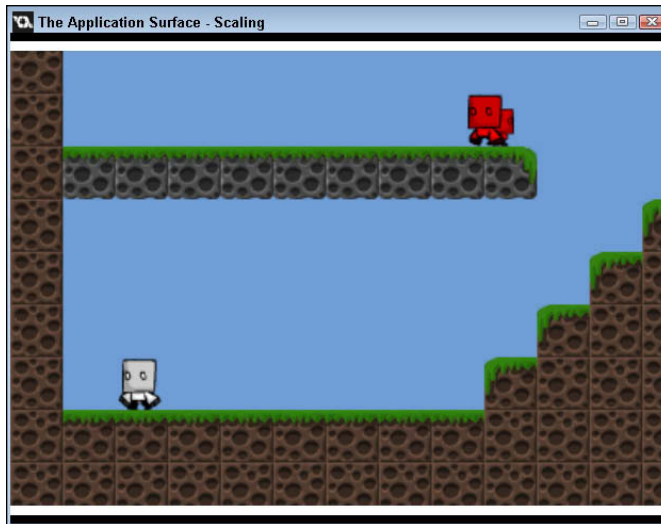


Figure 10-4: The white rectangles appear over the letterbox effect.

You'll most likely still see some of the black bars from the letterbox effect. This shows that the white rectangles didn't replace the letterbox effect; they just attempted to cover them up. Now, it's up to you to go back and fine-tune the code to cover, say, 2 times the area instead of 1.5 times the area, to see what a difference that makes.



You don't have to use white rectangles; you could use any color or even a Sprite. In the next section, I show you how to add Sprites to the GUI layer to create a HUD.

Placing GUI elements to create a heads-up display

This section shows you how to draw GUI elements directly over the application surface. These types of GUI elements typically include such things as the score, number of lives, virtual keys (see Chapter 13), and so on.

To do this, you need to continue from the preceding procedure. Basically, you'll be using the interface values to place different items (score, lives, keys) around the GUI layer to create a heads-up display. A HUD is great for games in which the player needs to know certain information, such as how much health she has left, how much ammo she has, where she is on a map, how strong her shields are, and so on.

To get started, you have to create a new Sprite of your choosing, as described in Chapter 2. Name this Sprite, `spr_Button`. For the purposes of the tutorial, I created a green circle for the Sprite as indicated in the tutorial. The following procedure shows how to draw this Sprite in each corner of the application surface. In this way, you can create your own HUD. This same procedure could also work for drawing text or backgrounds.

The following procedure picks up from `The_Application_Surface_Scaling` tutorial.

- 1. Double-click `obj_Control` from the Resource tree.**

The Object Properties appear.

- 2. Select the Draw GUI Event from the Events section.**

The Actions for the Draw GUI Event appear in the Actions section.

- 3. Double-click the Execute Code Action from the Actions section.**

The code window appears.

- 4. Add the following code after the existing code:**

```
draw_sprite(spr_Button, 0, x1 + 32, y1 + 32);
draw_sprite(spr_Button, 0, x2 - 32, y1 + 32);
draw_sprite(spr_Button, 0, x1 + 32, y2 - 32);
draw_sprite(spr_Button, 0, x2 - 32, y2 - 32);
```

- 5. Click the green check mark to save and close the code window.**

If you run the game now, you should see something similar to Figure 10-5.

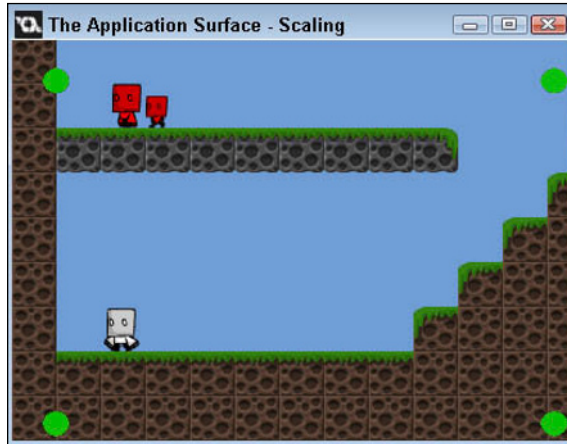


Figure 10-5: The Sprite appears in the four corners.

HTML5 Scaling

The beginning of this chapter covers resolution and scaling for desktop targets, which also works for the most part in the HTML5 target. However, you can fine-tune your code specifically for browser windows instead of desktops. The main differences are that the window is now referred to as a canvas element of the web page and the display becomes the width and height of the browser.

This section works in tandem with the in-software tutorial, `HTML5_Scaling`. You can find this tutorial from the New Project window on the Tutorials tab. From the tree on the left, open `Intermediate` and then select `HTML5_Scaling`. Over the next few procedures, you create your variables, write a Script, and then create a call to the Script that will scale the full Room to fit the browser window.

To create the variables, follow these steps:

1. **With the `HTML5_Scaling` tutorial open, from the Resource tree, in the Objects folder, double-click `obj_HTML5_Scaling`.**

The Object Properties appear.

2. **Choose `Add Event` → `Create`.**

The Create Event appears in the Events section.

3. **Drag and drop an `Execute Code Action` from the Control tab to the Actions section.**

A code window appears.

4. In the code window, type the following:

```
base_width = room_width;
base_height = room_height;
width = base_width;
height = base_height;
```

5. Click the green check mark.

The code window is saved and closed.

This code sets variables for the base width and height of the Room. The second set of variables will account for when the browser height and width change. The beginning value for the width and height variables equal the base width and height (this will change later with a Script you write next). You can leave the Object Properties open — you'll come back to it later after you write the following Script.

To write a Script to scale for HTML5, follow these steps:

1. From the Resource tree, right-click Scripts, and choose Create Script.

The Script code window appears.

2. In the Name field, type scr_HTML5_Scale_Canvas.**3. In the code window, type the following:**

```
aspect = argument0 / argument1;
if argument2 / aspect > argument3
{
    window_set_size(argument3 * aspect, argument3);
}
else
{
    window_set_size(argument2, argument2 / aspect);
}
if argument4
{
    window_center();
}
```

4. Click the green check mark.

The code window is saved and closed.

This code checks the size of the browser window and makes the necessary calculations to maintain aspect ratio (as discussed earlier in this chapter) and scaling if the player changes the size of the window. The last line of code centers the game within the browser, which is optional, and it doesn't currently work with the Windows 8 (JS) target.

To call the Script, follow these steps:

1. **From the Object Properties for obj_HTML5_Scaling (which is probably still open from earlier), choose Add Event→Step→Step.**

The Step Event appears in the Events section.

2. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

3. **In the code window, type the following:**

```
if browser_width != width || browser_height != height
{
    width = browser_width;
    height = browser_height;
    scr_HTML5_Scale_Canvas(base_width, base_height, width, height, true);
}
```

4. **Click the green check mark.**

The code window is saved and closed.

This code checks to see if the browser has changed size, and if so, runs the Script you wrote earlier to scale the canvas.

5. **Click OK from the Object Properties.**

The Object Properties are saved and closed.

6. **From the icon menu, click the Target drop-down and select HTML5.**

Now, when you test the game, GameMaker will generate code specifically for HTML5.

7. **From the main menu, choose Run→Run Normally.**

The game should appear in a browser window. The game will scale and maintain the aspect ratio if you resize the window.

If you don't want the canvas to scale to the full size of the browser, and you'd rather have it scale only to the maximum size of the game Room, you can make a small change to the code (as denoted in the bold text) you wrote for the Step Event as shown here:

```
if browser_width != width || browser_height != height
{
    width = min(base_width, browser_width);
    height = min(base_height, browser_height);
    scr_HTML5_Scale_Canvas(base_width, base_height, width, height, true);
}
```

Now when you test the game, the player can size the browser window smaller than the game Room and the game will maintain aspect ratio and scaling. But the game won't scale bigger than the dimensions set in the Room properties. If you make the browser window bigger than the Room size, the game will appear both horizontally and vertically centered in the window.

Part IV

Coding Physics, Mobile Controls, and More

```
Event: obj_Player.Step
102 //Check for mouse button shooting
103 if device_mouse_check_button(st_pressed[1], mb_left)
104 {
105     var mx, my;
106     mx = scr_mouse_gui_x(device_mouse_raw_x(st_pressed[1]));
107     my = scr_mouse_gui_y(device_mouse_raw_y(st_pressed[1]));
108     vdist = minisc_radius, point_distance(mx, my, gui_x[1], gui_y[1]);
109     vdir = point_direction(gui_x[1], gui_y[1], mx, my);
110     stick_x[1] = gui_x[1] + lengthdir_x(vdist, vdir);
111     stick_y[1] = gui_y[1] + lengthdir_y(vdist, vdir);
112
113     //Check for the shooting variable
114     if c_shoot
115     {
116         //Setup vars
117         var xx, yy, mod_dif;
118         c_shoot=false; //Set shoot var to false
119         alarm[0] = room_speed / 3; //Set shoot alarm
120         audio_play_sound(snd_GunShoot_Pistol, 0, false); //Play shot sound
121         mod_dif = b_angle + 3 - random(6); //Set the random angle for bullet variation
122         xx = x + lengthdir_x(20, mod_dif); //get the x/y position to create the bullet
123         yy = y + lengthdir_y(20, mod_dif);
124         instance_create(xx, yy, obj_Flash); //Create the muzzle flash
125         with (instance_create(xx,yy,obj_Player_Bullet)) //create the bullet
126         {
127             dir = mod_dif; //Set the bullet direction
128         }
129         //Now we will create the bullet casing as a nice special effect
130         with (instance_create(x + lengthdir_x(18, b_angle), y + lengthdir_y(18, b_angle), obj_Casing))
131         {
132             image_index = 0;
133             direction = (other.b_angle + 90) - 15 + random(30);
134             image_angle = direction - 90;
135         }
136     }
137 else st_pressed[1] = -1
138 }
139 //make character point towards mouse
```



Find out how to add code to a GameMaker: Studio project in a free article at www.dummies.com/extras/gamemakerstudio.

In this part . . .

- ✓ Code with GameMaker Language.
- ✓ Use Physics for more realistic gameplay.
- ✓ Create Virtual Keys, Analog Sticks, Swipes, and more.
- ✓ Save player options such as toggling background music.

Coding with GameMaker Language

In This Chapter

- ▶ Looking at coding examples from the breakout tutorial
- ▶ Using GameMaker Language code
- ▶ Coding keyboard controls
- ▶ Looking at example code for collisions
- ▶ Coding score displays
- ▶ Creating an introductory Room and a closing Room

An extremely powerful feature of GameMaker: Studio is the ability to code using GameMaker Language (GML). GameMaker makes it easy for you to add custom code to your game through the Execute Code Action found on the Control tab of the Object Properties window.

To get you acquainted with GML, this chapter uses coding examples from the in-software tutorial, Coding Tutorial, which shows how to create a breakout-style game using GML. Through the example code, you work with Objects that represent a player-controlled bat, a bouncing ball, and a number of other Objects that you can use in your own games.



This chapter isn't a one-stop shop for learning GML or coding a breakout-style game. Instead, it's an introduction with examples on using GML and coding examples.

This chapter also describes how to create introductory and closing screens by setting up more than one Room. I introduce the concept of saving and displaying the current high score. These features are one that you can then add to your own games.

For additional information on coding, turn to Chapter 14. Chapter 7 has a good coding example on how to create a button. And Chapter 12 has coding examples based on physics.



While using GameMaker, at any time, you can press F1 on your keyboard to open the in-software manual. From there, you can read additional information on the functions used in this chapter.

Displaying Sub-Images with Code

The following two procedures describe how to write code to tell GameMaker to

- ✓ Randomly display sub-images within Sprites
- ✓ Display Sprites based on the *sub-image index*, which is the number assigned to the sub-image

The following two examples are based on the in-software tutorial on coding a breakout game. Within that tutorial, you create a Sprite that looks like a brick, and it has several sub-images, each sub-image being a different-colored brick. In the first procedure, you tell GameMaker to randomly display the sub-images. In the second procedure, you tell GameMaker to sort the sub-images by index number.

Randomly displaying sub-images

I get started with coding in a very simple way — by using only two lines of code that can really change how your game looks.

To randomly display a sub-image, follow these steps:

- 1. Create a Sprite with several sub-images (for example, each sub-image could be a different colored brick).**

See Chapter 2 for more on creating Sprites.

The Sprite should appear in the Resource tree.

- 2. Choose Resources → Create Object.**

The Object Properties window appears.

- 3. In the Name field, type something like `obj_brick` (see Figure 11-1).**

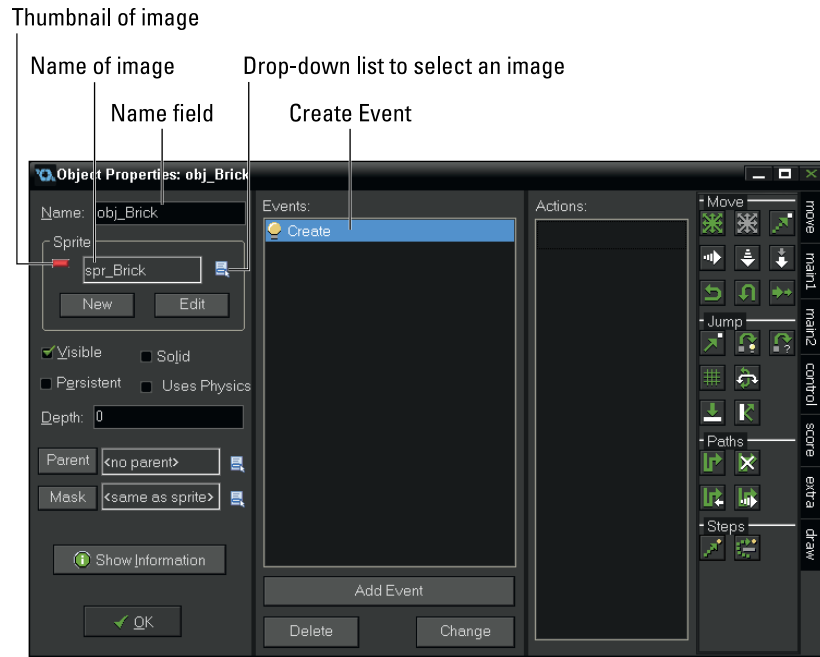


Figure 11-1: The Object Properties window for the brick Object.

4. **From the Sprite drop-down list, select the Sprite with the sub-images that you just created (see Chapter 2).**

A thumbnail of the image appears in the Sprite section, and the name of the image appears in the field (refer to Figure 11-1).

5. **Choose Add Event→Create.**

A Create Event appears in the Events section of the Object Properties window (refer to Figure 11-1).

6. **Drag and drop the Execute Code Action from the Control tab into the Actions section.**

An empty code window appears with its own menu.

7. **In the code window (shown in Figure 11-2), type the following code:**

```
image_speed = 0;
image_index = floor(random(image_number));
```

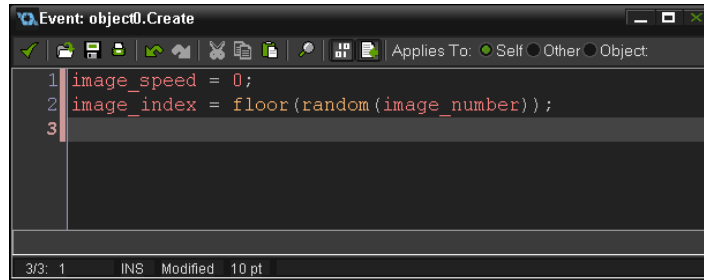


Figure 11-2: The code window.

8. Click the green check mark in the icon menu.

The code window is saved and closed.

9. Click OK in the Object Properties window.

The Object Properties are saved and closed.

10. Place an Instance of the Object in the Room.

See Chapter 2 on creating Rooms and placing Instances of Objects in the Room.

11. From the main menu, choose Run → Run Normally, to test the game.

The compile window displays the compilation of the game. When the game starts, you should see randomly sorted bricks, as shown in Figure 11-3.

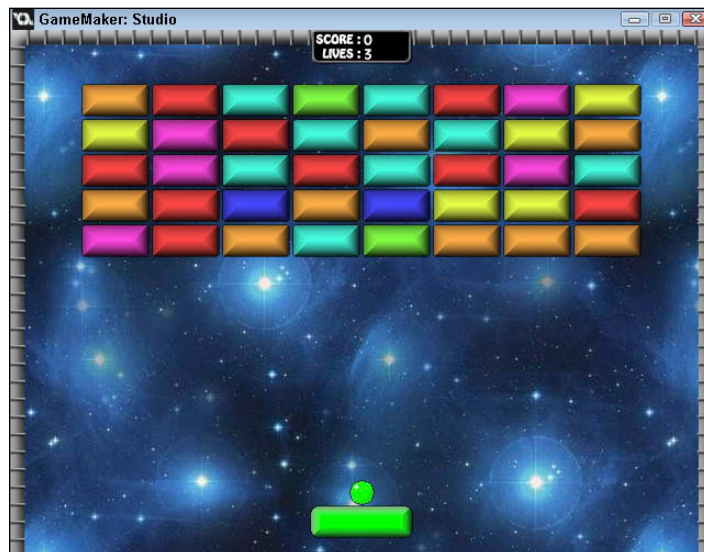


Figure 11-3: Randomly sorted bricks.

This code sets the speed of the image to 0. This is the speed GameMaker uses to show the individual sub-images of the Sprite. The default is 1, which means that normally GameMaker displays a single frame (one of the sub-images) in each step of the game. By using 0 as the image speed, you're telling GameMaker not to animate the Sprite and to show only a single sub-image. The second line of the code randomly selects one of the sub-images that will be displayed during the game.



If you were to use 0.5 as the speed, then the sub-images would appear in slow motion or at half the speed. If you want the sub-images to appear very quickly, you could enter a value of 2.



You may have noticed that the code window is color-coded — that is, certain text appears in different colors than others. That is a useful feature of GameMaker, in that comments, functions, variables, and so on are highlighted in different colors. If you don't like the color coding, or you want to use your own custom colors, you can update the Preferences — from the main menu choose, File⇨Preferences, and then select the Scripts and Code tab as shown in Figure 11-4.

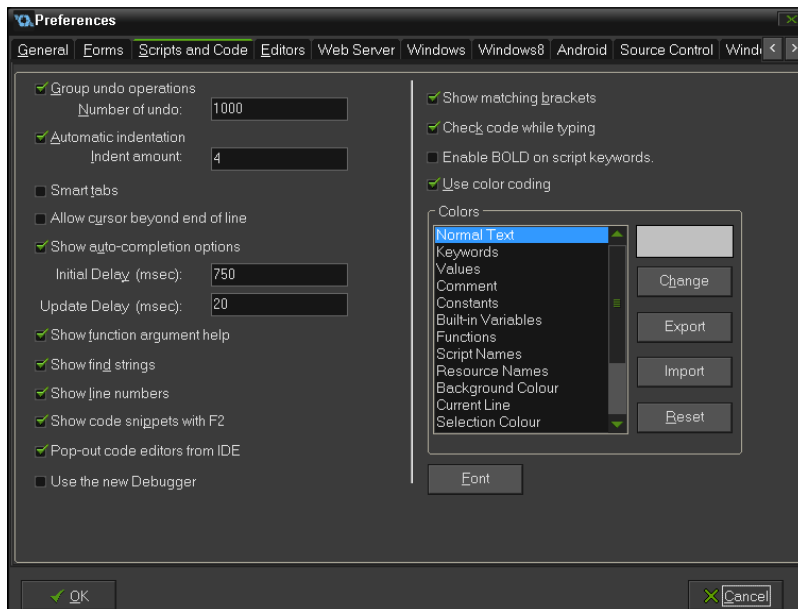


Figure 11-4: The Scripts and Code tab of the Preferences window.

Sorting sub-images by index

If you want to use a single Object to display multiple Instances within a Room, you can assign a Sprite to that Object that has multiple sub-images. Then you can write the following code so that GameMaker displays the sub-images in a sorted manner. The following code example is from the in-software coding tutorial, which shows you how to make a breakout-style game.

To sort sub-images by position (as based on the Room coordinates), follow these steps:

- 1. Create a Sprite with at least four sub-images (for example, each sub-image could be a different colored brick).**

See Chapter 2 for more on creating Sprites.

The Sprite should appear in the Resource tree.

- 2. Choose Resources→Create Object.**

The Object Properties window appears.

- 3. In the Name field, type something like obj_brick.**

- 4. From the Sprite drop-down list, select the Sprite with the sub-images (see Chapter 2).**

A thumbnail of the image appears in the Sprite section, and the name of the image appears in the field.

- 5. Choose Add Event→Create.**

A Create Event appears in the Events section of the Object Properties window (refer to Figure 11-1).

- 6. Drag and drop the Execute Code Action from the Control tab into the Actions section.**

An empty code window appears.

- 7. In the code window, type the following code:**

```
image_speed = 0;
if y <= 32 image_index = 0;
else
if y <= 64 image_index = 1;
else
if y <= 96 image_index = 2;
else
if y <= 128 image_index = 3;
else
if y <= 160 image_index = 4;
```

The code checks for the Y position of the Instance (as determined by the coordinates of the Room) and then assigns the index number of the Sprite to determine which sub-image to display.

8. Click the green check mark to save and close.

Figure 11-5 shows an example of sorting sub-images by index using this code. This game uses a single Sprite with five sub-images, which are displayed based on their index.

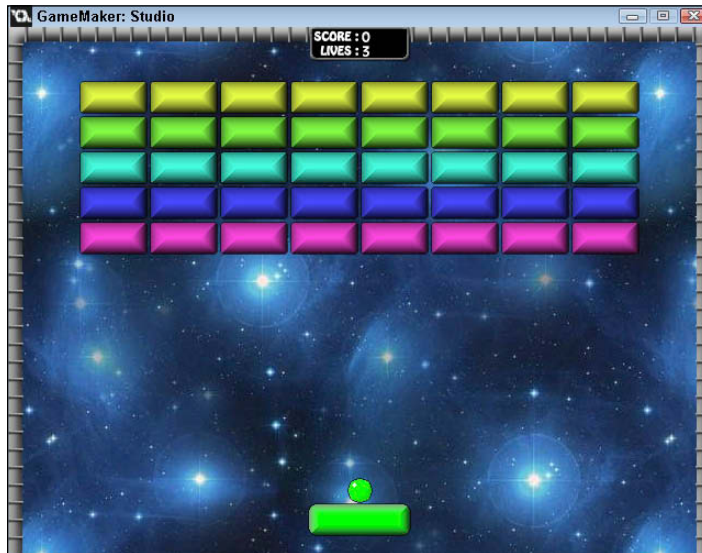


Figure 11-5: An example of sorted bricks using the index of sub-images.

Adding Code to Objects

You add code to Objects through an Execute Code Action from the Control tab. There is an infinite number of ways to use code in games — these examples are just the tip of the iceberg. In the next few sections, I provide sample code for

- ✓ Keyboard controls for moving an Object left or right
- ✓ Launching an Object into the Room through the spacebar
- ✓ Using Collision and Step Events with code
- ✓ A Script that prevents an Object from getting stuck in other Objects

For information on how to create controls for mobile devices that don't have a keyboard, check out Chapter 13.

Coding keyboard controls

You can use the drag-and-drop Actions (see Chapter 4) to create keyboard controls, but you can also write your own code for keyboard controls. The following procedures show you how to write code to make an Object move left and right.

To code left and right keyboard controls, follow these steps:

1. **Create a Sprite that will represent what the player controls, such as the bat Sprite from the breakout tutorial.**

See Chapter 2 for more information on creating Sprites.

The Sprite should appear in the Resource tree.

2. **Choose Resources → Create Object.**

The Object Properties window appears.

3. **In the Name field, type something like obj_bat.**

4. **From the Sprite drop-down list, select the Sprite, such as spr_bat (see Chapter 2).**

A thumbnail of the image appears in the Sprite section, and the name of the image appears in the field.

5. **Choose Add Event → Keyboard → Left.**

A Left Keyboard Event appears in the Events section of the Object Properties.

6. **Drag and drop the Execute Code Action from the Control tab into the Actions section.**

An empty code window appears.

7. **In the code window, type the following code:**

```
if place_meeting(x - 5, y, obj_wall) = false
{
  x -=5;
}
```

The code first checks for a collision between the Object (an Instance of the Object during gameplay) and obj_wall (if there are fewer than 5 pixels separating the Object from the wall). If there is less than 5 pixels of space between the Object and the wall, GameMaker sets the X position of the Object to move 5 pixels over. In that way, the player can't move the Object into the wall.

8. Click the green check mark.

The code window is saved and closed.

9. Choose Add Event→Keyboard→Right.

The Right Keyboard Event appears in the Events section.

10. From the Control tab, drag and drop an Execute Code Action into the Actions section.

An empty code window appears.

11. In the code window, type the following code:

```
if place_meeting(x + 5, y, obj_wall) = false
{
    x +=5;
}
```

The code for the Right arrow key is the same as for the Left arrow key, except you're checking for the right boundary (in this case, `obj_wall`). A good thing to remember is that `+x` is to the right and `-x` is to the left. In the same way, `+y` is down and `-y` is up.

12. Click the green check mark.

The code window is saved and closed.

You could test this code now by placing an Instance of the bat Object in the Room. Of course, you would need the wall Object created as well. To find out how to create the wall Object, you can follow along with GameMaker's coding tutorial.

To code the spacebar to trigger an Action, follow these steps:

1. With your Object open, such as the Object from the last procedure, choose Add Event→Keyboard→Space.

The Space Keyboard Event appears in the Events section.

2. Drag and drop an Execute Code Action from the Control tab to the Actions section.

An empty code window appears.

3. Type the following code:

```
with (obj_ball)
{
    if go = false
    {
        go = true;
        direction = 45 + random(90);
        speed = 5;
    }
}
```

This code does a few things. Basically, when the player presses the spacebar (you set this code in a Keyboard Event), the game starts with a ball Object shooting out into the Room. The ball Object has a speed of 5 pixels per step, and it will travel at a random direction between 45 and 135 degrees (135 is derived from adding 90 to 45).

This code also introduces the `go` variable, which you define in the next procedure. The `go` variable is an example of a *controller variable*. A controller variable, much like a controller Object, controls how and when different events and actions are triggered. Here, the code says that if `go` is set to `false`, and if the spacebar is pressed, set `go` to `true` and give the ball Object speed and direction. In the next procedure, I show you how to define the `go` variable, set it to `false`, and tell it what to do until it's triggered to `true`.

Coding an Object with collisions and Step Events

The following code examples show how to create a ball Object that behaves differently, depending on what it collides with. The Object should react differently if it collides with a wall Object, a brick Object, or a bat Object. Also, if the ball happens to leave the Room, then the game restarts. Figure 11-6 shows the Events in the Events section of the Object Properties window for the ball Object. Each of these Events will have a single Execute Code Action. The Execute Code Action opens a code window into which you enter your code.

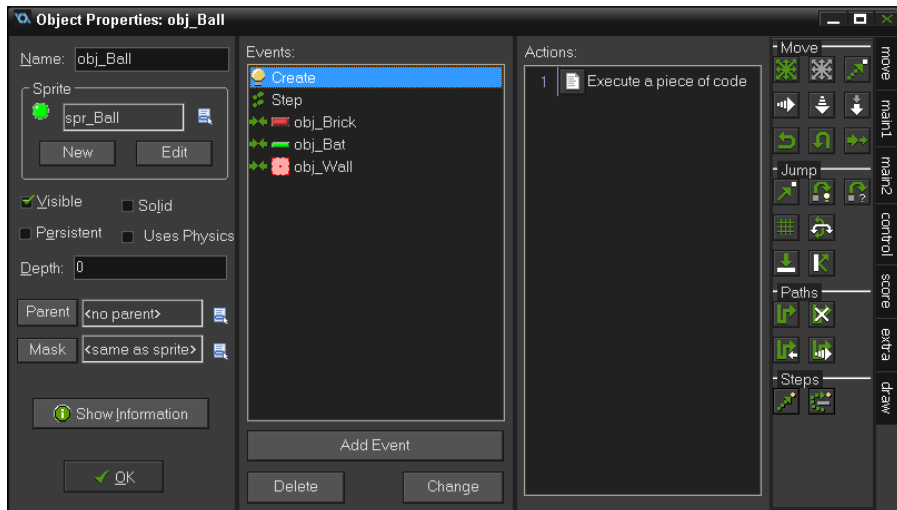


Figure 11-6: The Object Properties window for the ball.

In this procedure, you set the `go` variable to `false`. This `go` variable is linked, so to speak, with the Object from the previous procedure — the line of code with `(obj_ball)`. In the following procedure, you create `obj_ball`. To create the ball Object, follow these steps:

1. Create a Sprite that will represent the ball, such as the ball Sprite from the breakout tutorial.

See Chapter 2 for more information on creating Sprites.

The Sprite should appear in the Resource tree.

2. Choose Resources → Create Object.

The Object Properties window appears.

3. In the Name field, type something like `obj_ball`.

4. From the Sprite drop-down list, select `spr_ball` (see Chapter 2).

A thumbnail of the image appears in the Sprite section, and the name of the image appears in the field.

5. Choose Add Event → Create.

A Create Event appears in the Events section of the Object Properties window.

6. Drag and drop the Execute Code Action from the Control tab into the Actions section.

An empty code window appears.

7. In the code window, type the following code:

```
go = false;
```

You just defined the `go` variable as `false` (see the last procedure).

8. Click the green check mark.

The code window is saved and closed.

9. Choose Add Event → Step → Step.

A Step Event appears in the Events section.

10. Drag and drop an Execute Code Action from the Control tab to the Actions section.

An empty code window appears.

11. In the code box, type the following code:

```
if go = false
{
x = obj_bat.x;
}
else
{
if y > room_height room_restart();
}
```



The first line of code references the control variable `go`, which you used in the previous code example for the spacebar. Here, in the Step Event, for every step of the game, GameMaker will test to see if `go` is `false`; if it is, the X position of the ball Object will equal the X position of the bat Object (meaning, if the player moves the bat back and forth before starting the game, the ball will follow the bat back and forth).

The X position refers to the coordinates of the Room. When the player presses the spacebar, the code from the previous example changes the `go` variable to `true`, which means the `else` statement in this code is triggered (because `go` no longer equals `false`). When the `else` statement is triggered, GameMaker now checks for the Y position of the ball, and if the ball leaves the Room, the game restarts.

12. Click the green check mark.

The code window is saved and closed.

13. Choose Add Event→Collision→obj_wall.

A Collision Event for the wall Object appears in the Events section.

14. Drag and drop an Execute Code Action from the Control tab to the Actions section.

An empty code window appears.

15. In the code window, type the following code:

```
move_bounce_all(true);
```

This code tells the ball to bounce off the wall when the two collide. By using `true`, this creates *precise bouncing*, which provides some realism to the bounce effect.

16. Click the green check mark.

The code window is saved and closed.

17. Choose Add Event→Collision→obj_brick.

A Collision Event for the brick Object appears in the Events section.

18. Drag and drop an Execute Code Action in the Actions section.

An empty code window appears.

19. In the code window, type the following code:

```
move_bounce_all(true);
with (other) instance_destroy();
```

This is the same line of code as for the wall, but this time you want the ball to destroy other Instances in the Room, such as the bricks from the breakout tutorial, and not just bounce off.

The snippet of code with `(other)` is very useful, in that it tells GameMaker that if the Object collides with another Object, that other Object (the Object that collided with this Object) should be destroyed. I provide an example of this in Chapter 6, in the section on creating a boss battle. In that example, if a player's bullet collides with the boss Object, the Instance of the player's bullet that collided with the boss Object is destroyed; however, any other Instances of the player's bullet will remain in the Room.

20. Click the green check mark.

The code window is saved and closed.

21. Choose Add Event→Collision→obj_bat.

A Collision Event with the bat appears in the Event section.

22. Drag and drop an Execute Code Action from the Control tab to the Actions section.

An empty code window appears.

23. In the code window, type the following code to determine how the ball bounces off the bat (or the Object of your choosing).

```
move_bounce_all(true);  
var dir;  
dir = point_direction(other.x, other.y, x, y);  
motion_add(dir, 5);  
speed = 5;
```

This code provides the ball a direction to travel in, based on the other Instance it collides with, whether that be the bat, the wall, or a brick.

24. Click the green check mark.

The code window is saved and closed.

That concludes this procedure. Now, you should have a ball Object with several Events, as shown in Figure 11-6, found at the beginning of the ball procedures.

Fixing a bug with a script

If you play your breakout game enough times, you're bound to notice that sometimes the ball gets stuck in the bat, or the wall, or other Instances. To fix this, you can write a script that stops Instances from getting stuck in other Instances. A *script* is a snippet of code that you can write to create your own functions that can take arguments and return values.

To write a script to fix the ball bug, follow these steps:

1. Right-click Scripts from the Resource tree and choose Create Script.

The Script window (shown in the following figure) appears.

2. In the Name box, type `scr_move_out`.

3. In the code window, type the following code:

```
while (place_meeting(x, y,
    argument0))
{
    x += lengthdir_x(1, argument1);
    y += lengthdir_y(1, argument1);
}
```

4. Click the green check mark.

The Script window is saved and closed.

5. Open `obj_ball` from the Resource tree.

6. Select the wall Collision Event.

The Actions for the wall Collision Event appear in the Actions section.

7. Double-click the Execute Code Action.

The code window appears.

8. Update the code to match the following:

```
var dir;
dir = direction - 180;
scr_move_out(other, dir);
move_bounce_all(true);
```

This code references the script that you wrote earlier.

9. Click the green check mark.

The code window is saved and closed.

10. Select the bat Collision Event.

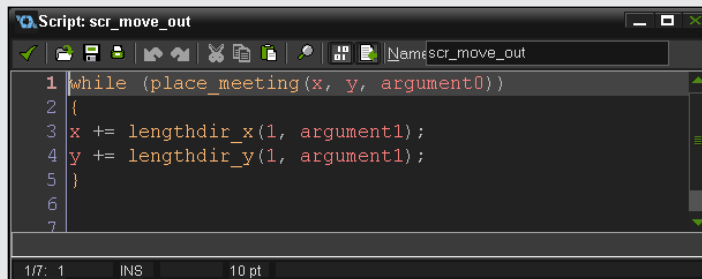
The Actions for the bat Collision Event appear in the Actions section.

11. Double-click the Execute Code Action.

The code window appears.

12. Update the code to match the following:

```
var dir;
dir = point_direction(other.x,
    other.y, x, y);
scr_move_out(other, dir);
move_bounce_all(true);
motion_add(dir, 5);
speed = 5;
```



13. Click the green check mark.

The code window is saved and closed.

14. Select the brick Collision Event.

The Actions for the brick Collision Event appear in the Actions section.

15. Double-click the Execute Code Action.

The code window appears.

16. Update the code to match the following:

```
var dir;
dir = point_direction(other.x,
    other.y, x, y);
scr_move_out(other, dir);
move_bounce_all(true);
```

```
with (other)
    instance_destroy();
score += 10;
```

17. Click the green check mark.

The code window is saved and closed.

18. Click OK to save the Object Properties window.

If you save and run your game now, the ball should no longer get stuck or trapped in Instances.

Using scripts can be a powerful way to enhance your games. You can find many scripts by searching the Internet and then use them in GameMaker.

Coding a Score Display and Lives

By writing code to create the score display, you can customize it to your liking, without necessarily having to use images. With that said, the following example uses the ScoreBox image, which you can find in the Graphics folder of the assets for the in-software coding tutorial.

To load the Sprite for the score display, follow these steps:

1. Choose Resources → Create Sprite.

The Sprite Properties window appears.

2. In the name field, type something like spr_scorebox.**3. Choose Load Sprite.**

A navigation window opens.

4. Navigate to the scorebox.png on your PC, or whichever image you want to use.**5. Select the image file and choose Open from the navigation window.**

A thumbnail of the image appears in the Sprite Properties window.

6. For the Origin, click Center (see Figure 11-7).

GameMaker determines the center of the image — in this case, X becomes 48 and Y becomes 16 (refer to Figure 11-7). This will help you align the text for the score and lives later on.

7. Click OK.

The Sprite Properties window is saved and closed.

You just created a Sprite that you can use as part of your score display. Next, I show you how to create an Object for the score display by using code.

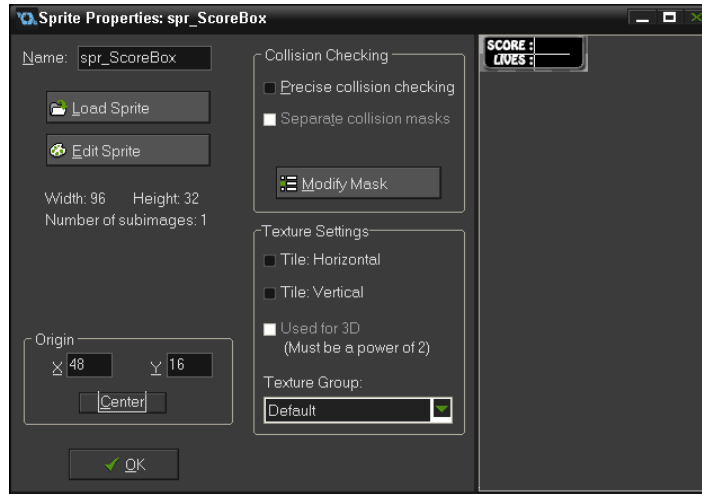


Figure 11-7: The Sprite Properties window.

To create the score display Object, follow these steps:

1. Choose Resources→Create Object.

The Object Properties window appears.

2. In the Name field, type something like obj_score_display.

3. From the Sprite drop-down list, select the Sprite, such as spr_ScoreBox (see Chapter 2).

A thumbnail of the image appears in the Sprite section, and the name of the image appears in the field.

4. Choose Add Event→Create.

A Create Event appears in the Events section of the Object Properties window.

5. Drag and drop the Execute Code Action from the Control tab into the Actions section.

An empty code window appears.

6. In the code window, type the following code:

```
draw_set_halign(fa_left);
draw_set_color(c_white);
go = true;
alpha = 0;
```

You just set up the text alignment for the score display to start on the left and the color to be white. However, when you created the Sprite, you set the Origin to Center, which means the text will be drawn starting from the center of the Sprite, so the left text alignment actually starts in the middle of the Sprite, which is right where you want it.

7. Click the green check mark.

The code window is saved and closed.

8. Choose Add Event → Step → Step.

A Step Event appears in the Events section.

9. Drag and drop an Execute Code Action from the Control tab into the Actions section.

An empty code window appears.

10. In the code window, type the following code:

```
if go = false
{
if alpha < 1
{
alpha += 0.05;
}
else
{
ini_open("Score.ini");
+- if score > ini_read_real("Score","0",0)
{
ini_write_real("Score","0",score);
}
ini_close();
room_goto(room_intro_close);
}
}
else
{
if instance_exists(obj_brick) = false
{
room_restart();
}
}
```

This code creates the transition effect for when the game ends or if the player clears all the bricks. If the game is over, the player is returned to the introduction Room, which you need to create (see Chapter 2). If the bricks are cleared, then the Room resets, but the score remains. It also reads and writes the high score to an INI file; if the player's current score

is higher than the saved high score, then the old high score is updated with the player's current score. You can learn more about reading and writing INI files and more in Chapter 14.

11. Click the green check mark.

The code window is saved and closed.

12. Choose Add Event⇨Draw⇨Draw.

A Draw Event appears in the Events section.

13. Drag and drop an Execute Code Action from the Control tab to the Actions section.

An empty code window appears.

14. In the code window, type the following code:

```
draw_self();  
draw_text(x, 2, score);  
draw_text(x, 15, lives);  
draw_rectangle(0, 0, room_width, room_height, false);  
draw_set_alpha(1);
```

You just told GameMaker to draw the score display Sprite (`draw_self`) and the text. The X coordinate was defined in the Create Event, so you don't have to assign a value here. The 2 and the 15 are the values for the Y coordinate, which determines the height at which to draw the text.

15. Click the green check mark.

The code window is saved and closed.

16. Click OK in the Object Properties window.

The Object Properties window is saved and closed.

Now you should turn your attention back to the ball Object. You need to change the Step Event so that it reduces a life when the ball leaves the screen and the game only restarts after the player loses his last life.

To update the ball Object, follow these steps:

1. Select the ball Object from the Resource tree (see “Coding an Object with collisions and Step Events,” earlier in this chapter).

2. Select the Step Event.

The Step Event Actions appear in the Actions section.

3. Double-click the Execute Code Action.

The code window appears.

4. Update the code to the following:

```
if go = false
{
x = obj_bat.x;
}
else
{
if y > room_height
{
lives -= 1;
if lives < 1
{
room_restart();
}
else
{
instance_create(obj_bat.x, obj_bat.y - 28, obj_ball);
instance_destroy();
}
}
}
```

This code tells GameMaker to reduce the player's lives by 1 when the ball leaves the Room. If the player has additional lives, a new ball Instance is created over the bat. If the player does not have any remaining lives, then GameMaker restarts the game.

5. Click the green check mark.

The code window is saved and closed.

6. Select the Collision Event for the bricks.

The Actions for the Collision Event appear in the Actions section.

7. Double-click the Execute Code Action.

The code window appears.

8. Add the following code at the end of the existing code:

```
score += 10;
```

This code adds 10 points to the player's score each time a brick is destroyed.

9. Click the green check mark.

The code window is saved and closed.

10. Click OK.

The Object Properties window is saved and closed.

Your score display Object and Sprite should now be ready for use. Remember to place the score display Object in the Room for you to see it in action (see Chapter 2 for more information).

Coding to Start and End Games

In this section, I show you how to code Rooms for introducing and closing your games. These introduction screens are useful to display many different types of game features, including:

- ✓ Showing a title graphic
- ✓ Displaying high scores
- ✓ Displaying a Play button

The following procedures do cover these features, but there are other game options you may want to add to your game, such as allowing the player to toggle the sound off and on or allowing the player to configure her own keyboard controls (see Chapter 14).

Introducing the game

In the introduction screen, you show the player several pieces of information. This includes a title graphic, a Play button, and the current high score, as shown Figure 11-8. You create a new Room for this intro screen, which you also use when the game restarts.

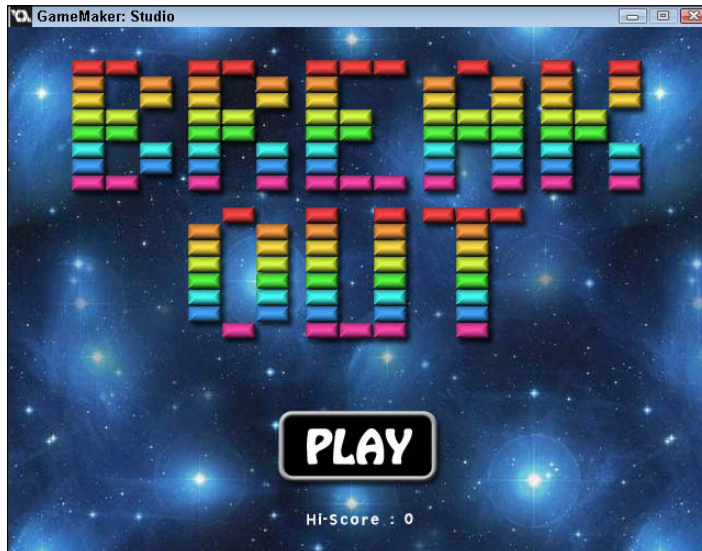


Figure 11-8: An example intro screen showing a Play button and high score.

To save the high score, you create an INI file (see Chapter 14). I walk you through the details in this chapter as well. The following procedures are based on the in-software coding tutorial, but you can adapt them for your own games.

To create two Sprites for the introduction screen, follow these steps:

- 1. Right-click Sprites from the Resource tree, and then choose Create Sprite.**

The Sprite Properties window appears.

- 2. In the Name field, type something like `spr_play`.**

- 3. Choose Load Sprite.**

A navigation window appears.

- 4. Navigate to where you have your image saved on your PC and choose, say, `Play.png`.**

- 5. Choose Open from the navigation window.**

A thumbnail of the Sprite appears in the Sprite Properties window.

- 6. Click OK.**

The Sprite Properties window is saved and closed.

- 7. Create a second Sprite by right-clicking Sprites from the Resource tree and then choosing Create Sprite.**

The Sprite Properties window appears.

- 8. In the Name field, type `spr_title` or `spr_intro` or something appropriate for the title graphic.**

- 9. Choose Load Sprite, navigate to your image, such as `Titles.png`, and choose Open from the navigation window.**

A thumbnail of the image appears in the Sprite Properties window.

- 10. Click OK.**

The Sprite Properties window is saved and closed.

You've now created two different Sprites that you can use for the introduction screen of your game.

To create the Control Object for the introduction screen, follow these steps:

- 1. From the Resource tree, right-click Object and choose Create Object.**

The Object Properties window appears.

2. In the name field, type something like obj_intro_end.

3. From the Sprite drop-down list, select spr_play.

A thumbnail of the image appears in the Sprite section, and the name of the image appears in the field.

4. Choose Add Event→Create.

A Create Event appears in the Events window.

5. Drag and drop an Execute Code Action from the Control tab to the Actions section

An empty code window appears.

6. In the code window, type the following code:

```
alpha = 0;
ini_open("score.ini");
hiscore = ini_read_real("Score","0",0);
ini_close();
draw_set_color(c_white);
```

The draw code defines the text color. This code also records and saves the high score in an INI file. If you didn't set up your own INI file, GameMaker creates one for you.

7. Click the green check mark.

The code window is saved and closed.

8. Choose Add Event→Draw→Draw.

A Draw Event appears in the Events section.

9. Drag and drop an Execute Code Action from the Control tab to the Actions section.

An empty code window appears.

10. In the code window, type the following code:

```
draw_self();
draw_sprite(spr_intro, 0, 64, 160);
draw_text(240, 440, "Hi-Score : " + string(hiscore));
draw_set_alpha(alpha);
draw_rectangle(0, 0, room_width, room_height, false);
draw_set_alpha(1);
```

You may recognize this code to draw the Play button (`draw_self`), the intro graphic (`spr_intro`), and the high score. The alpha variable is used to create a transition effect from the intro screen to actually playing the game. I show you how to do this next.

11. Choose Add Event→Mouse→Left Pressed.

The Left Pressed Mouse Event appears in the Events section.

12. Drag and drop an Execute Code Action from the Control tab to the Actions section.

An empty code window appears.

13. In the code window, type the following code:

```
if alpha = 0
{
    alarm[0] = 1;
    score = 0;
    lives = 3;
}
```

With this code, GameMaker checks the value of alpha, and if it is 0, it runs the Alarm, sets the score to 0, and gives the player three lives. GameMaker performs this check to make sure the Event runs a single time, even if the player repeatedly presses the mouse button.

The preceding code also sets the global variables for score and lives. Global variables are built into GameMaker, and they don't belong to Objects, but rather to everything within the game.

14. Choose Add Event ⇨ Alarm ⇨ Alarm [0].

The Alarm Event appears in the Events section.

15. Drag and drop an Execute Code Action from the Control tab to the Actions section.

An empty code window appears.

16. In the code window, type the following code:

```
alpha += 0.05;
if alpha >= 1
{
    room_goto(room_breakout);
}
else alarm[0] = 1;
```

You've now created the introduction screen for the game. Before I show you how to add it to the game, I'm going to walk you through creating the conclusion to the game.

Ending the game

In this section, I show you how to create a transition effect for when the player runs out of lives and the game ends. This includes saving the high score and restarting the Room.

1. **From the Resource tree, in the Object section, double-click the obj_intro_end Object.**

The Object Properties window appears.

2. **Select the Create Event.**

The Create Event Actions appear in the Actions section.

3. **Double-click the Execute Code Action from the Actions section.**

The code window appears.

4. **Update the code to the following:**

```
alpha = 0;
ini_open("score.ini");
hiscore = ini_read_real("Score", "0", 0);
ini_close();
draw_set_halign(fa_center);
draw_set_color(c_white);
```



You placed these draw functions in the Create Event, and not a Draw Event, because they don't actually draw anything. The draw functions here simply set the values for alignment and color. Because these functions are only needed once, you can place them in the Create Event. If you placed these functions in a Draw Event, GameMaker would be checking them once every step of the game, which isn't necessary.

5. **Click the green check mark.**

The code window is saved and closed.

6. **Click OK from the Object Properties window.**

The Object Properties window is saved and closed.

7. **From the Resource tree, open obj_control.**

The Object Properties window appears.

8. **Select the Draw Event.**

The Draw Event Actions appear in the Actions section.

9. **Double-click the Execute Code Action.**

The code window appears.

10. **Update the code to the following:**

```
draw_self();
draw_text(x, 2, score);
draw_text(x, 15, lives);
draw_rectangle(0, 0, room_width, room_height, false);
draw_set_alpha(1);
```

This code creates the transition effect for closing the game.

11. Click the green check mark.

The code window is saved and closed.

12. In the Object Properties window, in the Depth field, type -1000.

This makes sure that the introduction screen is drawn over everything else.

13. Click OK in the Object Properties window.

The Object Properties window is saved and closed.

14. From the Resource tree, open obj_ball.

The Object Properties window appears.

15. Select the Step Event.

The Step Event Actions appear in the Actions section.

16. Double-click the Execute Code Action.

The code window appears.

17. Update the code to look like the following:

```
if go = false
{
x = obj_bat.x;
}
else
{
if y > room_height
{
lives -= 1;
if lives < 1
{
with (obj_control) go = false;
}
}
else
{
instance_create(obj_bat.x, obj_bat.y - 28, obj_ball);
instance_destroy();
}
}
}
```

18. Click the green check mark.

The code window is saved and closed.

19. Click OK.

The Object Properties are saved and closed.

Now when you run the game, when the game is over, there should be a flash of white as the game transitions back to the introduction screen.

Hopefully, these coding examples will give you an excellent starting point for you to code your own games. They should also give you some ideas on how to work with Rooms to create introduction and closing screens.

If you find yourself having trouble getting the code to work, don't worry — you're not alone. When I did some of the GameMaker tutorials for the first time, I spent an entire day debugging my typos and mistakes, placing Actions in the wrong Events, using the wrong Events, and so on.

Using Physics

In This Chapter

- ▶ Understanding the Physics properties
- ▶ Working with fixtures
- ▶ Wrapping Objects around the Room
- ▶ Adding Physics to Rooms

In this chapter, I introduce you to working with physics in GameMaker: Studio. I draw upon examples from the in-software Physics Tutorial, as shown in Figure 12-1. (Choose the Tutorials tab from the New Project window that appears when you start the software; the Physics Tutorial is found under Advanced.) The tutorial provides information on how to make an Asteroids-style game.

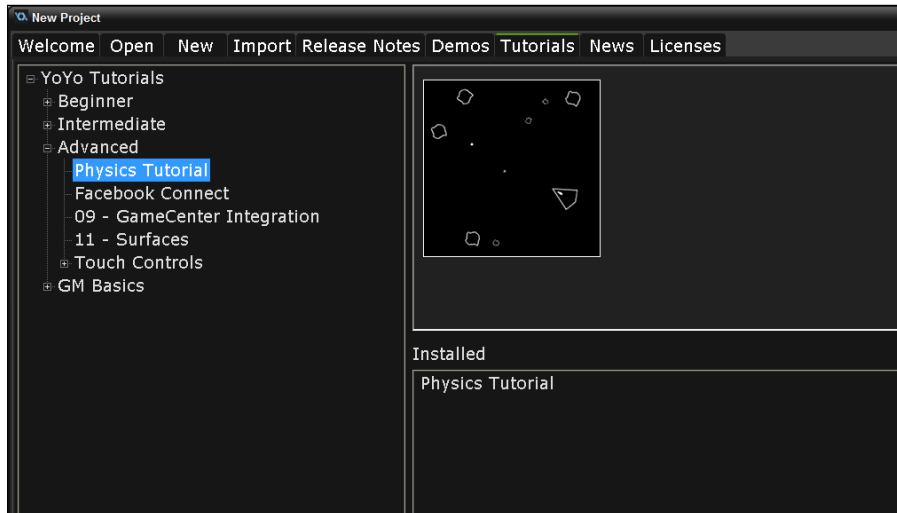


Figure 12-1: The New Project window.

Game physics revolves around the idea of making Objects appear more real to the player. By adding physics to your game, you change how Objects react during collision detection and how Objects move through the Room based on energy and force. Physics provide your game a more natural feel. This chapter doesn't explain everything you can do with physics and GameMaker, but it does provide a solid foundation of how physics works.

The examples in this chapter show you how to make a player's ship that can shoot bullets. You also write a Script that will enable both the ship and the bullets to leave one side of the Room only to reenter on the opposite side of the Room. Finally, this chapter includes a section on creating a Room that uses physics.



The physics system in GameMaker is completely different from the movement and collision systems used in other games you can create through the different in-software tutorials, such as Catch the Clown and Breakout. When you turn the physics system on, all interactions between Objects will be done through the physics system and all the built-in variables like `speed` and `image_direction` are no longer used. Instead, you use a whole new set of physics variables and constants.



This chapter assumes you already know how Objects (Chapter 2), Instances (Chapter 2), Rooms (Chapter 2), Events (Chapter 3), and Actions (Chapter 4) work. The emphasis of this chapter is on using code for physics.

Understanding Fixtures

When creating a physics-based game, you need to create a fixture for the Object, as opposed to a collision mask or `mask_index` (as described in Chapter 2). You create this fixture so that you can assign physical properties to the Object. Fixtures define the area of an Object that becomes the shape the physics uses for collisions; it also contains the physical information of the Object as well.

The basic process for creating a fixture (a detailed, step-by-step procedure is in this chapter) is to give it a shape, define the physical properties, and then apply that fixture to the Object. You can create a fixture and then apply it to multiple Objects so they all have the same base properties. GameMaker has built-in tools to help you create and apply fixtures in the Object Properties window.



When working with a physics-based game, your Sprites must have a specific shape in that they must be *convex* and never *concave*. A convex shape is one that only has an outline or surface curved like the exterior of a circle or sphere and it has only interior angles measuring less than 180 degrees. A concave shape is one where the outline curves inward, as shown in Figure 12-2.

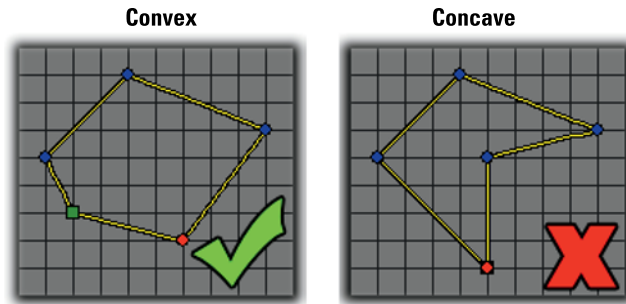


Figure 12-2: Create convex Sprites when creating a physics-based game.

The following procedure has you create a new Object for the player using physics properties. You can use the premade Sprite that is included in the Physics tutorial or make a new one using a convex shape.

Defining the physics properties

Before I get started with the procedure, here's a quick list of the different physics properties that you use when setting up your Object (see Figure 12-3):

- ✓ **Density:** Defines how many molecules are packed into the fixture. Think of a box full of feathers and a box full of books — they both *look* the same size but the one with the feathers is less dense (and so has less mass) than the one with the books. Changing this value will affect the overall mass of your fixture. For a quick and responsive effect, you would want the density low (for example, a value of 0.15).
- ✓ **Restitution:** Defines the bounce of the fixture. A high restitution value means that the Object will bounce a lot when in a collision, and a low restitution value means it will only bounce a little.
- ✓ **Collision group:** This feature enables you to create groups of fixtures, with the default group being 0. If you have two or more fixtures with a negative collision group (for example, a value of -1), these fixtures will never collide, even if they have a Collision Event set up. If you assign two or more fixtures to a positive collision group (for example, a value of 1), they'll always collide, whether they have a Collision Event or not. The default is 0 , which means that they'll collide if they have a Collision Event.



Using collision groups is a processor-heavy task and should only be used when absolutely necessary.

- ✓ **Linear damping:** When you throw something, it doesn't keep going forever because the friction against the air slows it down and eventually stops it (gravity works in this way, too, but that's for another section). Linear damping gives Objects a kind of drag or friction in the Room. To make an Object slow down quickly, set this value to 0.5.
- ✓ **Angular damping:** Differs from linear damping in that it affects rotation instead of direction. If this is set to 0, any bump or movement that affects the angular rotation of the fixture will continue infinitely. To make the Object responsive to the player, set this value to 0.5.
- ✓ **Friction:** Determines how quickly an Object slows down when its speed is greater than zero (can be negative or positive). For example, if the Friction is set to 0.1 and the speed of the Instance is 1 (1 pixel per step), the Instance will slow down and stop after ten steps.
- ✓ **Sensor:** Select this box to ignore the physical properties of the Object but still trigger a Collision Event with the Objects around it. This enables you to have an Instance with no physical presence in the Room, but the Instance can still react to collisions of other Instances and do something in turn.
- ✓ **Start awake:** Represents the initial state of the fixture, and the default is awake. When an Object is awake, it's considered part of the physics world. However, if for some reason this interferes with how your game starts, you deselect this option so that the Object starts off asleep, and it does not participate in the physics world unless something acts upon it.
- ✓ **Kinematic:** If you select Kinematic, the Object will be unaffected by collisions and gravity, but it can still be moved around or rotated using the appropriate values. A Kinematic Object is considered a static Object that happens to move; in contrast, a static Object has a density of 0 and you can position it at any absolute point in the Room.



All these properties behave differently depending on the mass of the fixture. The mass is calculated by GameMaker, based on the surface area of the fixture and the density.

Creating a fixture

To create an Object using physics, follow these steps:

1. **From the Resource tree, right-click Objects, and then select Create Object.**

The Object Properties window appears.

2. **In the name field, type something like `obj_player`.**

3. From the **Sprite** drop-down list, select a **Sprite** for this **Object**, such as **spr_Ship** from the in-software **Physics** tutorial.

A thumbnail of the image appears in the **Sprite** section and the name of the image appears in the field (see Figure 12-3).

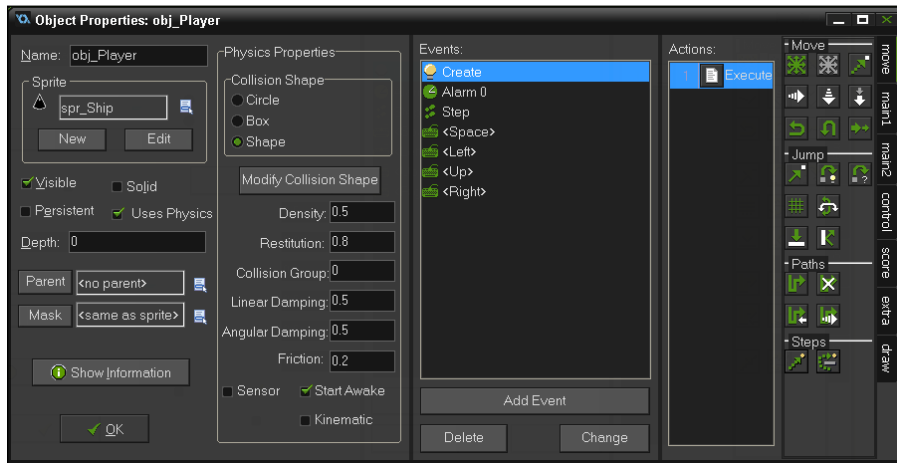


Figure 12-3: The Object Properties window.

4. Toggle **Uses Physics** so that a check appears in the box (refer to **Figure 12-3**).

The **Physics Properties** appear in the **Object Properties** window.

5. In the **Physics Properties** section, for the **Collision Shape**, select the **Shape** radio button.

If you're working in the **Physics** tutorial, **Shape** should have already been selected. If you aren't working from the tutorial, GameMaker will try its best to create a fixture around the shape of your **Sprite**, but you'll probably have to edit this as explained in the next step.

6. Choose **Modify Collision Shape** (refer to **Figure 12-3**).

This opens the **Physics Shape** (also known as the **fixture editor**). This is where you create the **fixture** for the **Object** (see **Figure 12-4**).

In the **Physics** tutorial, the **fixture** (the yellow lines that denote the outline of the player's ship, as indicated in **Figure 12-4**) was already created for you. However, in the **Collision** section of the **Physics Shape**, if you select the **Circle** or **Box** radio button, you'll see how the **fixture** changes. **Figure 12-4** shows how the outline changes if you select the **Circle** shape.

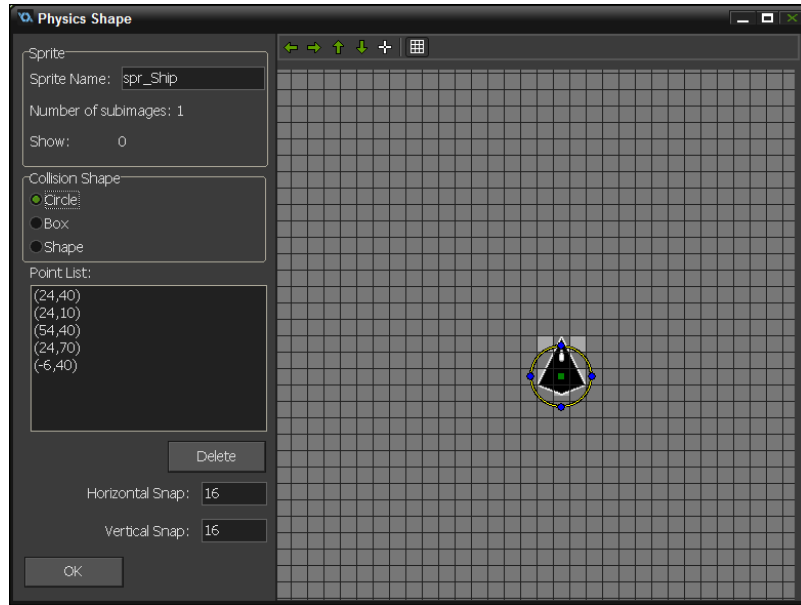


Figure 12-4: The Physics Shape window.



If you change the shape of the fixture and want to get back to the original Shape that was set up for you, click the X in the top-right corner of Physics Shape to close that window. Also choose the X in the Object Properties window and then select No to not save any changes. Then, when you reopen the Physics Shape, the original shape should still be there, as shown in Figure 12-5. If not, you can manually reset the fixture or simply reload a new tutorial.

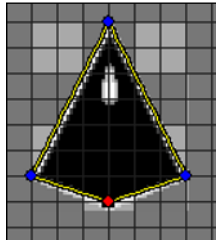


Figure 12-5: The fixture should match the shape of the Sprite.

To reset the fixture manually, keep in mind that you can

- Click in an empty space to add a point (the blue dots shown in Figure 12-4)

- Click and drag an existing point to move it
- Right-click to delete a point
- Zoom by using the mouse wheel
- Hold the middle mouse button to drag the view or change the grid resolution to make adding points easier

When you create a fixture, you want to keep the number of points as low as possible with a minimum of three points (triangle shape) with the overall shape being convex. In the Physics tutorial, the final shape of the fixture should look like Figure 12-5, in that the fixture matches the shape of the Sprite.



The green square (refer to Figure 12-4) signifies the starting point of the fixture. The blue circles represent path points, and the red circle is the currently selected path point.

7. In the **Density field**, type 0.5.
8. In the **Restitution field**, type 0.8.
9. In the **Collision Group field**, type 0.
10. In the **Linear Damping field**, type 0.5.
11. In the **Angular Damping field**, type 0.5.
12. In the **Friction field**, type 0.2.
13. Leave **Sensor** unchecked.
14. Toggle **Start Awake** (so that the check mark appears).
15. Leave **Kinematic** deselected.
16. Click **OK**.



The Object Properties window is saved and closed.

I provide explanations for each of these fields in the section preceding this procedure.

You're now done with setting the physics for the Object.

Adding Movement to an Object with Physics

After you get your Object defined for physics, you might want to allow the player to move the Object around the Room during the game. The following two sections show how to add movement to an Object as well as make an Object turn left or right.

Example code for an Up Keyboard Event to apply local force

To apply force to an Object to make it move, create an Up Keyboard Event as described in the following procedure. Just follow these steps:

1. **From the Resource tree, double-click the obj_player Object or another relevant Object.**

The Object Properties window appears.

2. **Choose Add Event→Keyboard→Up.**

An Up Keyboard Event appears in the Events section.

3. **Drag and drop an Execute Code Action from the Control tab to the Actions window.**

An empty code window appears.

4. **In the code window, type the following code:**

```
physics_apply_local_force(0, 0, 0, -50);
```

5. **Click the green check mark.**

The code window is saved and closed.

This code uses the *local coordinate system* to set a vector for propulsion. The local coordinates are based on the Sprite, where the X and Y axes meet at the Origin (see Chapter 2). So, no matter which way the Object is rotated in the Room, the origin of the Sprite never changes. In that way, X = 0, Y = -1 would always be just below the Sprite, as shown in Figure 12-6. The first value in this function is for xlocal, the second value is ylocal, the third is xforce_local, and the last value represents yforce_local. By using the value -50 as the value for yforce_local, you're placing force behind the ship that pushes the ship forward. The amount of force is based on the value, so a larger number places greater force. Experiment with this by changing the value from -50 to -5000 to see what a difference it makes. If you use a positive number — say, 50 — notice that the ship appears to move backward instead of forward.

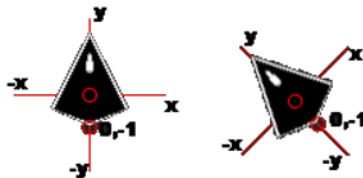


Figure 12-6: Local coordinates are based on the Sprite's Origin as denoted by the red circle.

Example code for Left and Right Keyboard Events

To enable the player to turn the Object, add Left and Right Keyboard Events.

To have the player turn the Object to the left, follow these steps:

1. **From the Resource tree, double-click the obj_player Object or another relevant Object.**

The Object Properties window appears.

2. **Choose Add Event→Keyboard→Left.**

A Left Keyboard Event appears in the Events section.

3. **Drag and drop an Execute Code Action from the Control tab to the Actions window.**

An empty code window appears.

4. **In the code window, type the following code:**

```
phy_angular_velocity=0;  
phy_rotation-=10;
```

5. **Click the green check mark.**

The code window is saved and closed.

During the game, the player can now turn the Object to the left. This code adds or subtracts from the physics rotation, which is what controls the direction and image angle of the Object when the player presses the left-arrow key.

To have the player turn the Object to the right, follow these steps:

1. **From the Resource tree, double-click the obj_player Object (or another relevant object).**

The Object Properties window appears.

2. **Choose Add Event→Keyboard→Right.**

A Right Keyboard Event appears in the Events section.

3. **Drag and drop an Execute Code Action from the Control tab to the Actions window.**

An empty code window appears.

4. **In the code window, type the following code:**

```
phy_angular_velocity=0;  
phy_rotation+=10;
```

Leaving one side and entering the other

In the classic videogame Asteroids, the player's ship could leave one side of the Room only to reenter on the opposite side of the Room (for example, it might leave on the left and reenter on the right). The following procedure shows you how to create a Script to wrap the Room as part of the Step Event:

1. From the Resource tree, right-click Scripts and select Create Scripts (see Chapter 7 for another example on using Scripts).

The Script window appears.

2. In the Name field, type `phy_wrap` (see the following figure).

3. In the code section, type the following code:

```
if phy_position_x < 0 phy_
    position_x += room_width;
if phy_position_x > room_
    width phy_position_x -=
    room_width;
if phy_position_y < 0 phy_
    position_y += room_height;
if phy_position_y > room_
    height phy_position_y -=
    room_height;
```

4. Click the green check mark.

The Script window is saved and closed.

This code is different from what I present elsewhere in the book when working with X and Y

positions, because this example is based on physics and not the basic movement and collision system (non-physics).

Now you need to call this Script from an Object. To apply a Script to an Object, follow these steps:

1. From the Resource tree, in the Objects section, double-click the `obj_Player` Object (or the object you want to call the Script).

The Object Properties window appears.

2. Choose Add Event→Step→Step.

A Step Event appears in the Events section.

3. Drag and drop an Execute Code Action from the Control tab to the Actions section.

An empty code window appears.

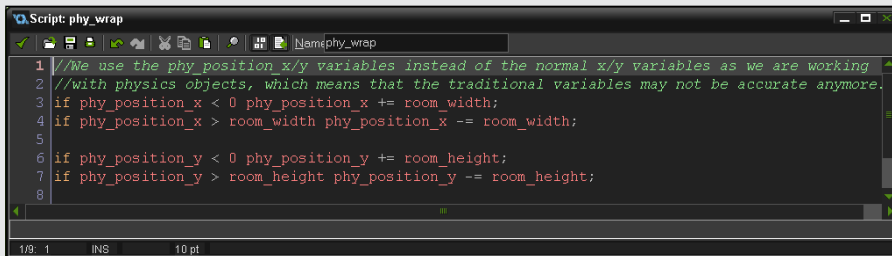
4. In the code window, type the following code:

```
phy_wrap();
```

5. Click the green check mark.

The code window is saved and closed.

This code calls the Script and applies it to the Object. Now, when an Instance of this Object is placed in the Room, the player can fly the Instance through one side of the Room only to reenter on the opposite side.



5. Click the green check mark.

The code window is saved and closed. This code adds or subtracts from the physics rotation, which is what controls the direction and image angle of the Object when the player presses the right-arrow key.



You could give the Object a more natural turning by using `physics_apply_torque` to rotate the Object using *torque*, which is the rotational force. Torque takes into account the mass of the Object and makes turning slower, much like a real-life object; however, in a game, you don't always want it to be too real (it is a game, after all).

Another difference between the physics world in GameMaker: Studio and the traditional functions is that normally the direction and image angle are calculated counterclockwise, so adding 10 to the direction variable would rotate the Object to the left. However, when using physics, directions and angles are calculated clockwise, so adding 10 to the rotational value rotates the Object to the right.

Adding Physics to Rooms

When using physics to create your game, along with using fixtures on your Objects (see “Understanding Fixtures” and “Creating a fixture” earlier in this chapter), you need to set up the Room to accommodate for physics.

There are two properties on the Physics tab of the Room Properties window that you need to learn:

- ✓ **Gravity:** Gravity is calculated as a *vector* (a quantity having direction as well as magnitude, which determines the position of one point in space relative to another) using the X/Y coordinates you input. The length of the vector indicates the force of gravity, and the direction indicates the direction of pull. So, a vector of X = 1, Y = 1 provides a weak gravity that pulls Objects up and to the right, while a vector of X = 0, Y = 10 creates a strong downward gravity.
- ✓ **Pixels to Meters:** GameMaker: Studio uses real-world sizes when computing functions, which means that you need to translate your pixel sizes into meters. This value controls how that ratio works, with (for example) a value of 0.1 being a ratio of 1 pixel = 10 cm. This value should not be any extreme value, such as 10, because it will cause problems.

The in-software Physics tutorial has a Room for physics already set up. I'll walk you through how and what you need to do for your own game. To set up the Room for physics, follow these steps:

1. From the Resource tree, right-click Rooms and then select Create Room.

The Room Properties window appears.

2. Click the Physics tab, as shown in Figure 12-7.

The Physics properties appear in the Room Properties window.

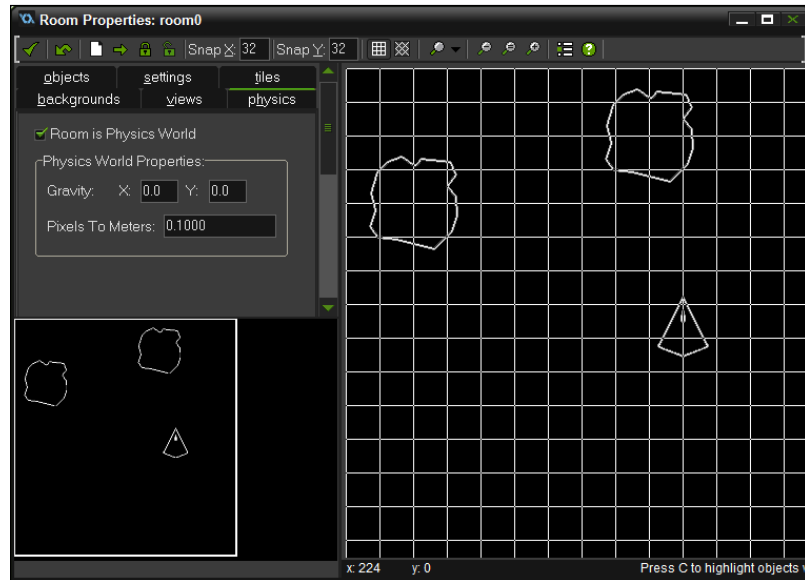


Figure 12-7: The Physics properties in the Room Properties window.

3. Select the Room Is Physics World check box (refer to Figure 12-7).

You just told GameMaker that this Room should use Physics for the Instances of Objects during game play.

4. In the Physics World Properties section, in the Gravity field for X, leave the default at 0.0.**5. In the Physics World Properties section, in the Gravity field for Y, type 0.0.**

You don't want gravity in your asteroids game, because it takes place in outer space. If you do want gravity in your game, so that Instances naturally fall down, leave the default setting of 10.0.

6. In the Pixels to Meters field, leave the default of 0.1000.**7. Click the green check mark.**

The Room Properties window is saved and closed.

The default value for Pixels to Meters works well for this game, but make sure that both the X and Y coordinates for the Gravity vectors are set to 0 — that is, if you're making a space game and you don't want gravity to interfere with the player's ship or the asteroids.

Physics and Shooting Bullets

In the Physics tutorial, the player's ship shoots bullets to break up the asteroids. The following tutorial will step you through that process.

Defining bullets to work with Physics

The following procedure is based on the Physics tutorial, which has some of the Objects and Sprites in the Resource tree, waiting for you to configure. You're encouraged to use your own Sprites for your own Objects for your own game.

To set up bullets in a physics environment, follow these steps:

1. **From the Resource tree, double-click `obj_Bullet` (or start a new Object with an assigned Sprite for the bullet; see Chapter 2).**

The Object Properties window appears.

2. **Toggle `Uses Physics` as shown in Figure 12-8.**

3. **In the Physics Properties section, select the `Box` radio button (refer to Figure 12-8).**

4. **Open the Physics Shape (or Fixture Editor) by clicking `Modify Collision Shape` Shape (refer to Figure 12-8).**

The Physics Shape window appears, as shown in Figure 12-9.

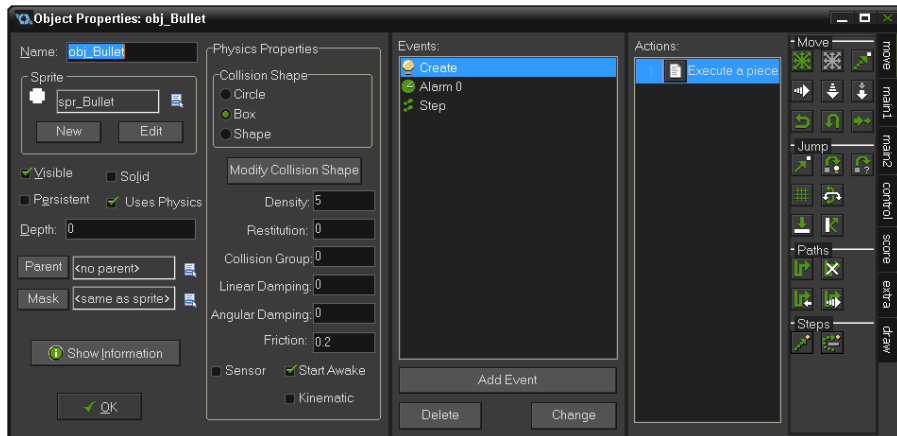


Figure 12-8: The Object Properties window for the bullet Object.

Make sure the yellow outline (known as the fixture) aligns with the bullet (see Figure 12-10). You may have to zoom in to see it properly. To zoom in, you can spin your mouse wheel. The fixture should be box shaped, because you chose Box for the Collision Shape in Step 3.

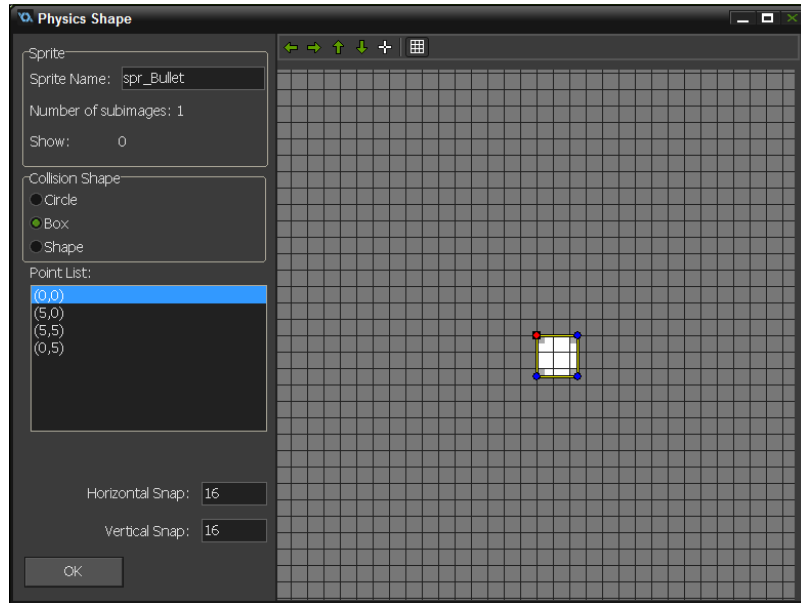


Figure 12-9: The Physics Shape window with a Box fixture for the bullet Sprite.

5. Click OK from the Physics Shape window.

The Physics Shape window is saved and closed.

6. In the Object Properties window, in the Physics Properties section, in the Density field, type 5 (refer to Figure 12-8).

The value of 5 is a relatively high density. You want to use a high density for the bullet because you want the impact to really affect the Instances of Objects that it collides with during gameplay.



GameMaker calculates physics for Objects based on their size. Therefore, because the bullet Sprite is small in size compared to the asteroids, you need the bullet Object to have a high density for it to have an impact during Collision Events. Think of the impact difference a bullet would have if it were made of aluminum instead of lead.

7. In the Restitution, Collision Group, Linear Damping, and Angular Damping fields, type 0 (refer to Figure 12-8).

You set these properties to 0 because you don't want the Object to bounce, slow down over time, or curve during a collision.



8. In the Friction field, type 0.2 (refer to Figure 12-8).

By applying Friction, you're telling the Object to slow down for each step of the game. How much the Object slows down is determined by the value you enter here and the speed of the Object.

9. Select the Start Awake box (refer to Figure 12-8).

All these properties are explained in this chapter in the "Defining the physics properties" section.

10. Choose Add Event→Create Event (if you don't already have a Create Event).

A Create Event appears in the Events section.

11. Drag and drop an Execute Code Action from the Control tab to the Actions section (if an Execute Code Action isn't already there).

The code window appears.

12. In the code window, type the following code:

```
phy_bullet=true;
alarm[0]=60;
```

This code uses the variable `phy_bullet`, which tells GameMaker this is a fast-moving bullet Object. As such, GameMaker provides the extra processing power to ensure proper collisions. This code also sets an Alarm that you set up next.

13. Click the green check mark.

The code window is saved and closed.

14. Choose Add→Alarm→Alarm 0.

The Alarm 0 Event appears in the Events section.

15. Drag and drop an Execute Code Action from the Control tab to the Actions section.

The code window appears.

16. Type in the following code:

```
instance_destroy();
```

This code destroys the Instance of the bullet Object. This code is triggered when the Alarm is triggered, which you set for 60 steps in Step 12.

17. Click the green check mark.

The code window is saved and closed.

18. Choose Add Event→Step→Step.

A Step Event appears in the Events section.

19. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

20. **Type the following code in the code window:**

```
phy_wrap();
```

This code calls the Script, which I describe earlier in this chapter. This Script enables the player Object, and now the bullet Object, to leave one side of the Room and reenter on the opposite side. Because you put this Action in a Step Event, GameMaker will check to see if the Instance leaves the Room for every step of the game.

21. **Click the green check mark.**

The code window is saved and closed.

22. **Click OK from the Object Properties window.**

The Object Properties window for the bullet is saved and closed.

You've now defined a bullet Object using physics. Now you can update the player Object so that the player's ship can shoot the bullets.

Coding the ship to shoot bullets

Now that you have the bullet Object defined to work with physics (see the preceding section), you need to set up the ship so that it will shoot the bullet itself.

To set up the ship to fire bullets, follow these steps:

1. **From the Resource tree, in the Objects section, double-click the obj_Player Object (I describe how to create this Object in the "Creating a fixture" section, earlier in this chapter).**

The Object Properties window appears.

2. **Choose Add Event → Create.**

A Create Event appears in the Events section.

3. **Drag and drop an Execute Code Action from the Control tab to the Actions section (if it already exists, simply double-click to open it).**

The code window appears.

4. **In the code window, type the following code:**

```
canshoot=true;  
phy_fixed_rotation=true;
```


The first line of code creates the variable, `canshoot`, which will control the rate of fire and set it to `true`. When `canshoot` equals `true`, that means the player can shoot bullets. The second line of code stops collisions from spinning the ship.

5. Click the green check mark.

The code window is saved and closed.

6. Choose Add Event→Alarm→Alarm 0.

The Alarm Event appears in the Events section.

7. Drag and drop an Execute Code Action from the Control tab to the Actions section (if it already exists, then simply double-click to open).

The code window appears.

8. In the code window, type the following code:

```
canshoot=true;
```

Again, you are setting the `canshoot` variable to `true`, which enables the player to shoot bullets. You're doing this in an Alarm Event. In the next step, I show you how to trigger this Alarm to rearm the ship after the player shoots.

9. Click the green check mark.

The code window is saved and closed.

10. Choose Add Event→Keyboard→<Space>.

The Space Keyboard Event appears in the Events section.

11. Drag and drop an Execute Code Action from the Control tab to the Actions section (double-click to open it if it already exists).

The code window appears.

12. In the code window, type the following code (see Figure 12-10):

```
if canshoot
{
    with (instance_create(x,y,obj_Bullet))
    {
        phy_rotation=other.phy_rotation;
        physics_apply_local_impulse(0, 0, 0, -100);
    }
    alarm[0]=15;
    canshoot=false;
}
```

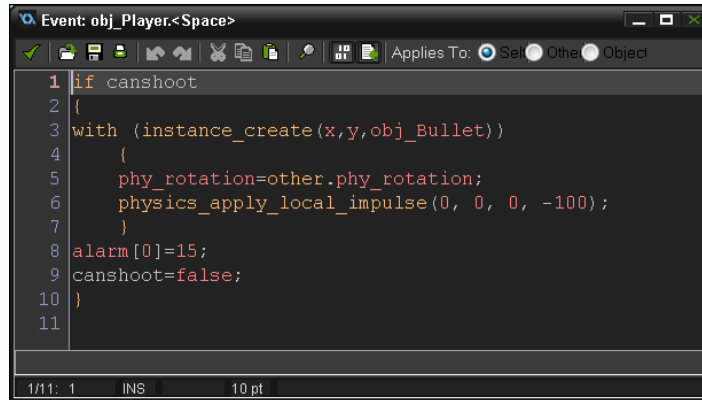


Figure 12-10: The code window for the Space Keyboard Event.

13. Click the green check mark.

The code window is saved and closed.

This code first checks to see if the variable `canshoot` is set to `true`, which it should be at the beginning of the game, because you set `canshoot` to equal `true` in the Create Event. Because `canshoot` does equal `true`, when the player presses the spacebar, the Space Bar Event is triggered and the preceding block of code is triggered. This block of code creates the bullet at the player position through the `with` function. The code also sets the directional angle of the bullet to that of the ship by using the keyword `other` in a function. The code then applies a *local impulse* to the bullet to shoot it off and, thus, sets the speed. Finally, the code sets the Alarm 0 to 15 (which defines the amount of time before the player can shoot again) and sets `canshoot` to `false` (so that the player has to wait for the Alarm to be triggered before he can shoot again).



Impulse differs from force in that it does not take into consideration factors such as velocity or torque.

Now is a good time to save your work and run the game to see how it performs. If you want to experiment with the values you defined, you'll discover what effect that has on your Objects as you play the game.

In Chapter 2, I discuss one way to make the asteroids by using parents and children so that the traits of the parent Object are passed on to the child Objects.

Creating Controls for Mobile Devices

In This Chapter

- ▶ Playing games on mobile devices
- ▶ Creating virtual keys on the screen
- ▶ Adding swipe controls
- ▶ Creating pinch and zoom controls
- ▶ Playing mobile games with virtual analog sticks

If you want to make a game these days, there's a very good chance you want your audience to play it on a mobile device. That's where this chapter, and Chapter 10 on scaling and resolution, come into play. In this chapter, I show you how to add controls to your game so that it's playable on a mobile device such as a tablet or phone. This includes virtual keys such as analog sticks, swiping, and pinching and zooming. You can add all these features to your game through GameMaker: Studio functions.

Putting Your Game on a Mobile Device

During the game creation process, be sure to test your game as you go. Think about how you're going to test your game on a mobile device before you create it. After all, it would be awful to spend hours creating virtual joysticks only to learn you have no way to test your work.

Using an emulator

To test the procedures in this chapter, you need to get your game onto a mobile device. You can do this through an emulator if you don't have a mobile device available. I'm not going to discuss the details behind using emulators, but if you look online, you can find plenty of information.

Putting your game online

If you have a website, you can upload your GameMaker files through FTP to appear on your site. Then you can access the game through your mobile device's browser, whether an iPhone, iPad, Kindle Fire, or any kind of Android tablet or phone.



If you decide to make an HTML5 game geared towards mobile devices, keep those games small and simple.

If you want to test your mobile game online, change the target of your game to HTML5. That way, when you save the game, GameMaker creates the appropriate files.

You can change the *target* (the platform you want to play the game on) of your game from the icon menu, as shown in Figure 13-1. From the Target drop-down list, choose HTML5. (For more information, see Chapter 1.)

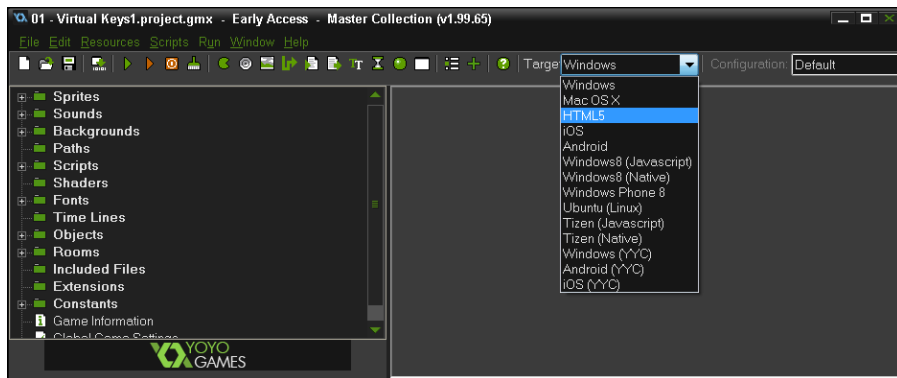


Figure 13-1: Select the target of the game.

Now, when you save your game, GameMaker will create an `index.html` file along with a folder named `html5game`; within that folder, you'll find the files that you need to upload (along with the `index.html` file) through FTP. After that's done, simply point your mobile device's browser to the URL, which ends in `index.html`.



When you change the target to HTML5, when you test the game, the game will open in your default browser.

The reason I suggest using HTML5 as the target is because it'll work on Android, iOS, and Windows devices. You'll be able to test your touch controls on any of those devices using HTML5.

Adding Simple Touch Controls

In this section, I show you how to add touch controls for the game Breakout, which is discussed in Chapter 11. In that chapter, the directions were to add in keyboard commands so that the player could play from a PC. However, the game is unplayable on a mobile device, because mobile devices don't have a traditional hardware keyboard. To remedy this, I show you how to add touch controls to move the bat in the Breakout game.



The Left Mouse Button Event is considered the same as a tap on mobile devices. So, if you use a Left Mouse Button Event in your game (say, for a button), then when you play on a mobile device, tapping that button works just as well as using a mouse.

Understanding virtual keys

Virtual keys (also known as touch controls) respond to a player's touch while on a mobile device. You can map virtual keys to keyboard keys so that any Keyboard Events you set up in GameMaker will have the same effect as a virtual key. For example, in the game Breakout, the player uses the left-and-right arrow keys to move the bat back and forth. You can map those arrow keys to work as virtual keys so that the player can tap a button to move the bat back and forth instead of using the arrows on the keyboard.

Virtual keys are simple rectangular areas and cannot be any other shape (such as ellipses). Virtual keys are mapped to the graphical user interface (GUI) layer; this means that they don't move with the View. Virtual keys are automatically removed when a Room ends. It doesn't matter in which Object you create the virtual keys — they'll trigger a key Event for all Instances equally, like any normal keyboard press.

Creating virtual keys

The following procedure is based on the Virtual Keys tutorial found within GameMaker. To create a virtual key, follow these steps:

- 1. Start the GameMaker: Studio software.**
- 2. From the New Project window, click the Tutorials tab.**
An outline of tutorials appears in the left column.
- 3. From the Beginner section on the left, click to open Touch Controls and then click to open the Virtual Keys tutorial.**

If this is the first time you've opened this tutorial, you'll be prompted to install it.

The Virtual Keys project window appears.

4. From the Target drop-down list, choose HTML5 (refer to Figure 13-1).

Now when you save and test your game, GameMaker will compile the files specifically for HTML5, or whichever Target you chose.

5. From the Resource tree, in the Objects section, double-click obj_Control.

The Object Properties window appears.

6. From the Events section, click the Create Event.

The Actions for the Create Event appear in the Actions section.

7. From the Actions section, double-click the Execute Code Action.

The code window appears.

8. In the code window, type the following code (as shown in Figure 13-2):

```
display_set_gui_size(view_wview[0], view_hview[0]);
key_left = virtual_key_add(32, display_get_gui_height() - 96, 64, 64, vk_
    left);
key_right = virtual_key_add(display_get_gui_width() - 96, display_get_gui_
    height() - 96, 64, 64, vk_right);
key_space = virtual_key_add((display_get_gui_width() / 2) - 32, display_
    get_gui_height() - 96, 64, 64, vk_space);
virtual_key_show(key_left);
virtual_key_show(key_right);
virtual_key_show(key_space);
```

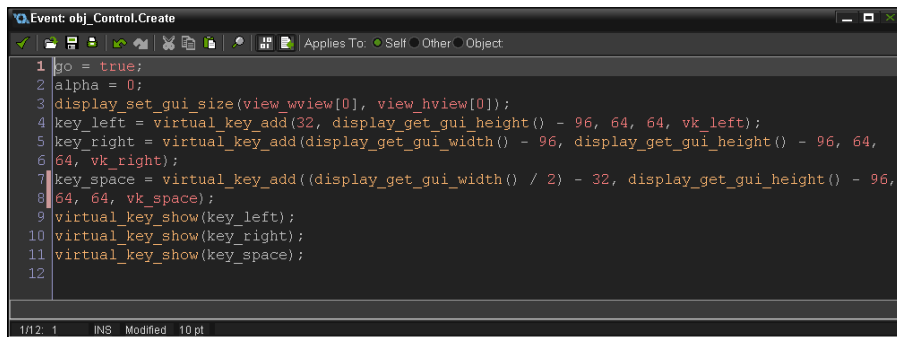


Figure 13-2: The code to create virtual keys.

9. Click the green check mark.

The code window is saved and closed.

This code sets the GUI size to match the View. The code also creates three virtual keys: left, right, and spacebar, which are drawn on the GUI layer as white squares. Each key is assigned a unique ID (`key_left`, `key_right`, and `key_space`), which is stored as a variable, so that you can then change it later.

At this point, you can test the game — from the main menu, choose Run⇄Run Normally. If you updated the Target to HTML5, the game should open in your default browser. After you click Play, you should see the outline of three squares, as shown in Figure 13-3. If you're on a desktop, you can click these squares and they'll work. If you're on a mobile device, you can tap these squares for them to work.



If you see three solid white squares and they don't work with the mouse, check your Target. Most likely, it's still set to Windows.

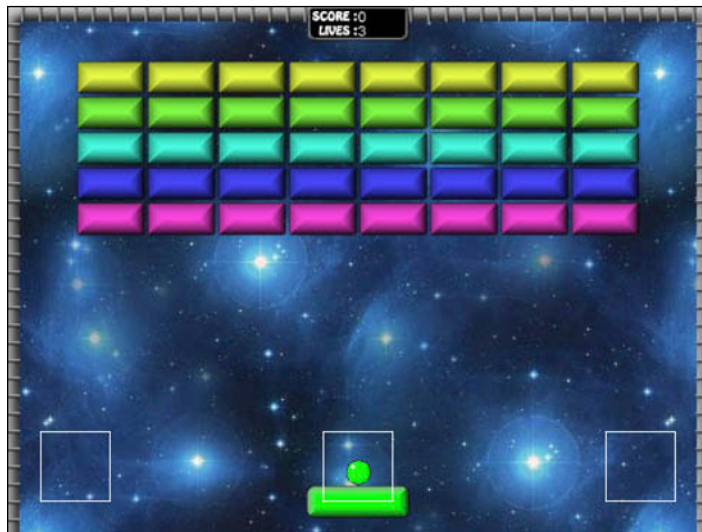


Figure 13-3: Virtual keys as depicted in the Google Chrome browser.

Adding Sprites to virtual keys

In all likelihood, you don't want your virtual keys looking like empty boxes. To remedy this, this section describes how to assign a Sprite to your virtual keys. If you're following along with the tutorial, GameMaker has Sprites ready and waiting for you.

To assign a Sprite to your virtual keys, follow these steps:

- 1. From the Resource tree, double-click `obj_Control` from the Objects section.**

The Object Properties window appears.

- 2. With the Create Event selected in the Events section, double-click the Execute Code Action from the Actions section.**

The code window appears.

3. Delete the last three lines of code:

```
virtual_key_show(key_left);
virtual_key_show(key_right);
virtual_key_show(key_space);
```

4. Type the following code:

```
s_left = 0;
s_right = 0;
s_space = 0;
```

You just set up variables to draw the correct Sprite for each of the buttons, which you'll set up next in the Step Event.

5. Click the green check mark.

The code window is saved and closed.

6. From the Events section, select the Step Event.

The Actions for the Step Event appear in the Actions section.

7. Double-click the Execute Code Action from the Actions section.

The code window appears.

8. Scroll down to line 32 and find the code `room_restart()`; . Type in the following code between the last two brackets so that it looks like Figure 13-4:

```
{
    room_restart();
}
s_left = keyboard_check(vk_left);
s_right = keyboard_check(vk_right);
if keyboard_check_pressed(vk_space)
{
    s_space = -1;
}
```

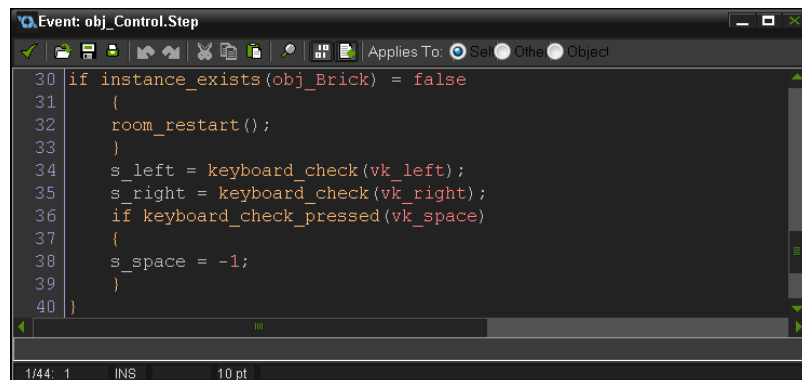


Figure 13-4: New code for the virtual keys.

9. Click the green check mark.

The code window is saved and closed.

10. Select the Draw Event from the Events section.

The Actions for the Draw Event appear in the Actions section.

11. Double-click the Execute Code Action from the Actions section.

The code window appears.

12. In the code window, type the following code at line 15 (after the existing code) to make the Sprites actually appear in the game:

```
if go
{
    draw_sprite(spr_Left, s_left, 32, display_get_gui_height() - 96);
    draw_sprite(spr_Right, s_right, display_get_gui_width() - 96, display_
        get_gui_height() - 96);
    if s_space != -1
    {
        draw_sprite(spr_Space, 0, display_get_gui_width() / 2, display_get_
            gui_height() - 64);
    }
}
```

This code checks the game state (`if go`) and then draws the buttons if there are bricks left. The launch Sprite is also drawn, but only up until the player taps that button. You need this button to reappear when the player loses the ball, which you set up next.

13. Click the green check mark.

The code window is saved and closed.

14. Click OK from the Object Properties window for `obj_Control`.

The Object Properties window is saved and closed.

15. From the Resource tree, in the Objects section, double-click `obj_Ball`.

The Object Properties window appears.

16. From the Events section, click the Step Event.

The Actions for the Step Event appear in the Actions section.

17. Double-click the Execute Code Action from the Actions section.

The code window appears.

18. At line 14, insert the cursor, and type in the following code at line 15, as shown in Figure 13-5.

```
s_space = 0;
```

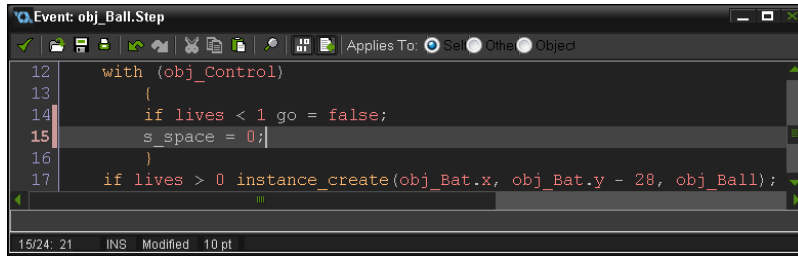


Figure 13-5: New code for the ball.

You can test your game again and you should see the button Sprites, shown in Figure 13-6, that you added for virtual keys. The game should play well on all mobile and touchscreen devices.

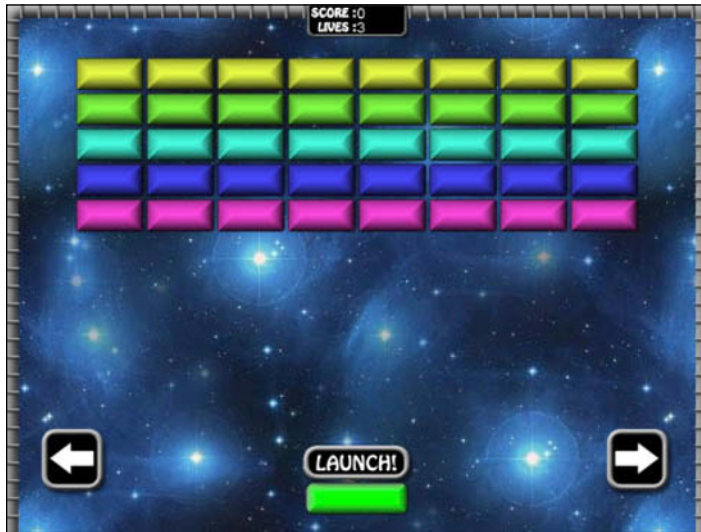


Figure 13-6: The Breakout game with Sprites for the virtual keys.

Adding Swipe Controls

Many games for mobile devices use swipes to control aspects of their game like skipping through menus or different screens. This same effect can easily be achieved using the normal mouse functions in GameMaker: Studio.

In the first part of this chapter, I show you how to set up virtual keys to detect taps on mobile devices, but that's only the beginning. For most mobile games, people expect to use swipe gestures to make certain things happen,

such as navigating menus. In the following procedures, I show you how to add swipe controls to the Breakout game. These procedures are based on the in-software tutorial, Swipes.

Single-direction swipes

For the following procedures, you can follow along with the in-software tutorial, Swipes. In this procedure, you will add a swipe check to the high-score menu to switch between local and online high scores.

To add a swipe check, follow these steps:

1. **Open the GameMaker: Studio software and from the New Project window, choose Tutorials→Intermediate→Touch Controls→Swipes.**

The Swipes Project window appears.

2. **From the Resource tree, in the Objects section, double-click obj_HighScores.**

You'll add a swipe check to this Object.

The Object Properties window appears.

3. **Select the Create Event from the Events section.**

The Actions for the Create Event appear in the Actions section.

4. **Double-click the Execute Code Action from the Actions window.**

The code window appears.

5. **Insert the cursor at line 20 and type the following code:**

```
page = 0;  
mx = 0;  
score_x = 0;  
swipe_min = 100;
```

This code sets up variables for the Object. These variables are defined as:

- **page:** The score page the player is on
- **mx:** The X position of the mouse at the start of the swipe
- **score_x:** The current base position GameMaker uses for drawing
- **swipe_min:** The minimum distance to travel (in pixels) before a swipe is detected (how much a person has to swipe before it's detected as a swipe)

6. **Click the green check mark.**

The code window is saved and closed.

7. Choose Add Event→Mouse→Global Mouse→Global Left Button.

The Global Left Button Event appears in the Events section. You want to use the Global Mouse Event because this detects a touch anywhere on the screen. A Local Mouse Event only detects touches with the Instance collision Mask (which is how the button works).

8. Drag and drop the Execute Code Action from the Control tab to the Actions section.

A code window appears.

9. Type in the following code:

```
if mouse_check_button_pressed(mb_left)
{
    mx = mouse_x;
}
score_x = clamp((page * room_width) + (mouse_x - mx), 0, room_width);
```

10. Click the green check mark.

The code window is saved and closed.

This code detects the touch and then stores the position of the touch on the X axis. The position of the X axis is used to set the position of where the scores are drawn.

The code uses the `clamp()` function so that the player can't drag the scores too far to the left or too far to the right. When the player swipes, she's moving the high-score list. If she swipes far enough, she'll switch to another page, say, from local high scores to global high scores.



You can modify the value of the page variable so that you can add more pages for the player to swipe through by changing the value in the Create Event.

The page itself will be set when the player releases her finger from the screen, which I show you how to set up in the next procedure.

To detect when the player stops swiping, follow these steps:

1. Choose Add Event→Global Mouse→Global Left Released.

The Global Left Released Event appears in the Events section.

2. Drag and drop an Execute Code Action from the Control tab to the Actions section.

A code window appears.

3. In the code window, type the following code:

```
var tx = mouse_x - mx;
if abs(tx) > swipe_min
{
    if sign(tx) == -1
    {
        page = 0;
    }
    else
    {
        page = 1;
    }
}
```

Here, the `abs()` function gets the absolute value of the distance between the start and the finish of the swipe. If that value is greater than the minimum swipe value, that triggers the `sign()` of the swipe value. The sign will be `-1` for a swipe to the left; if the swipe is not to the left, then the value of `1` is assigned to the `page` variable.

4. Click the green check mark.

The code window is saved and closed.

To get a smooth transition so that it looks like the player is sliding the pages across the screen when swiping, follow these steps:

1. With `obj_HighScores` still open from the last procedure, choose `Add Event` → `Step` → `Step`.

The Step Event appears in the Events section.

2. Drag and drop an `Execute Code Action` from the `Control` tab to the `Actions` window.

The code window appears.

3. In the code window, type the following code:

```
var xgo = (page * room_width);
if score_x != xgo //(page * room_width)
{
    score_x += (xgo - score_x) / 5;
}
```

4. Click the green check mark.

The code window is saved and closed.

This code moves the drawing position of the high-score tables based on the length of the player's swipe.

Run the game now — from the main menu, choose `Run` → `Run Normally`. Click the Scores button and you'll see the Local Scores. If you're on a mobile device,

you can swipe the screen to see the Online Scores. When you swipe, the Local Scores move with the swipe. If you're on PC, a click-and-drag is the equivalent of a swipe, and you'll see the Local Scores transition to the Online Scores.



This code works the same for vertical swipes, too. You just need to adjust the code to reflect the Y axis instead of the X axis and use the Room height instead of the Room width.

Directional swipes

In the game Breakout, to start the game, the player needs to launch the ball from the bat up toward the bricks. Currently, the player can launch the ball by either pressing the spacebar (from Chapter 11) or by using virtual keys (earlier in this chapter). These two options launch the ball in a random direction. In this section, I show you how you can launch the ball in a specific direction, based on which direction you swipe. This type of swipe is considered a directional swipe.

The following procedure picks up from the last section on using swipes.

To add a directional swipe to the breakout game, follow these steps:

1. **Open the GameMaker: Studio software and from the New Project window, choose Tutorials⇒Intermediate⇒Touch Controls⇒Swipes.**

The Swipes Project window appears.

2. **From the Resource tree, in the Objects section, double-click obj_Bat.**

The Object Properties window appears.

3. **From the Events section, right-click the Keyboard Pressed Space Event and choose Delete.**

You no longer want the player to launch the ball using the spacebar, so this Event is no longer necessary.

4. **Choose Add Event⇒Create.**

The Create Event appears in the Events section.

5. **Drag and drop an Execute Code Action from the Control tab to the Actions section.**

A code window appears.

6. **In the code window, type the following code:**

```
mx = 0;  
my = 0;
```

This sets two variables, `mx` and `my`, to 0. This represents an initial mouse (in this case, the player's finger) position at the start of the swipe.

7. **Click the green check mark.**

The code window is saved and closed.

8. Choose Add Event⇨Mouse⇨Global Mouse⇨Global Left Pressed.

The Global Left Pressed Event appears in the Events section.

9. Drag and drop an Execute Code Action from the Control tab to the Actions section.

A code window appears.

10. In the code window, type the following code:

```
mx = mouse_x;
my = mouse_y;
```

Now, when the player presses his finger against the screen, the `mx` and `my` variables are updated to match the X and Y coordinates of the touch.

11. Click the green check mark.

The code window is saved and closed.

12. Choose Add Event⇨Mouse⇨Global Mouse⇨Global Left Released.

The Global Left Released Event appears in the Events section.

13. Drag and drop the Execute Code Action from the Control tab to the Actions section.

A code window appears.

14. In the code window, type the following code:

```
if point_distance(mx, my, mouse_x, mouse_y) > 50
{
  with (obj_Ball)
  {
    if go = false
    {
      audio_play_sound(snd_Play, 0, false);
      go = true;
      direction = clamp(point_direction(other.mx, other.my, mouse_x,
        mouse_y), 10, 170);
      speed = clamp(point_distance(other.mx, other.my, mouse_x, mouse_y) /
        50, 2, 10);
    }
  }
  with (obj_Control)
  {
    s_space = -1;
  }
}
```

15. Click the green check mark.

The code window is saved and closed.

This code determines the direction of the swipe and how far, and how fast, the player pressed her finger against the screen. These values are what GameMaker uses to set the speed and direction for the launch of the ball.

Note the use of the `clamp()` function in this code. The `clamp()` function limits the possible angle of the ball during the launch so that the ball can't start at an angle that would cause the player to lose a life. The `clamp()` function also ensures that the ball doesn't launch too fast or too slow.

The last line of code stops the launch Sprite from showing while the game is in play.

Now is a great time to save and test your game — from the main menu, choose Run → Run Normally. If you're testing on a PC, you can click and drag with the mouse to represent a directional swipe of a finger on a mobile device.

Pinching and Zooming

The act of pinching and zooming has become commonplace when interacting with content on a mobile device. Pinching and zooming are when a player touches his thumb and index finger to the screen and then simultaneously pinches them together to zoom in on the content, or spreads the fingers wider apart to zoom out of the content (or vice versa depending on your device). This type of player interaction requires code that enables the player to use multiple touches during a game. This is where the `device_mouse_*()` function comes into play.

The `device_mouse_*()` function is designed to take an argument from 0 to 4. Each of those values (0, 1, 2, 3, 4) represents a finger. This makes it possible for you to tell GameMaker to check up to five simultaneous touches on a screen (remember that in coding, numbering always starts at 0).

In the following procedures, I show you how to detect when two fingers are touching the screen simultaneously. You then tell GameMaker to use the X and Y coordinates of where the player is touching the screen to create a pinch-and-zoom feature to a game.

The pinch function is represented through the `device_mouse_check_button()`. Remember that mobile devices treat fingers as mouse actions, so there are many opportunities to translate mouse functions for use on mobile devices. You use this function to check for the initial position of each finger and then to test if the distance between them changes.

To set up the zoom function, you use `device_mouse_raw_x/y()`. In this function, you use the *raw coordinates* as opposed to the Room coordinates (see Chapter 2 for more on Rooms). The raw coordinates are the X and Y position of the mouse within the screen, and not the Room. This is important to understand, because these coordinates change relative to the new coordinates as the player pinches.



For the Windows Target, the raw position is calculated from a specific window resolution, with the top-left corner being (0,0). However, on mobile devices, the raw position is calculated from the display resolution.

To add zoom functionality to your game, follow these steps:

1. From the New Project window, choose **Tutorials**→**Intermediate**→**Touch Controls**→**Pinch and Zoom**.
2. From the Resource tree, in the Objects section, double-click **obj_Controller**.

The Object Properties window appears.

3. Click the **Create Event in the Events section**.

The Actions for the Create Event appear in the Actions section.

4. Double-click the **Execute Code Action in the Actions section**.

The code window appears.

5. In the code window, type the following code at line 6 (as shown in Figure 13-7):

```
initial_w = view_wview[0];
initial_h = view_hview[0];
v_scale = 1;
v_dist = 0;
```

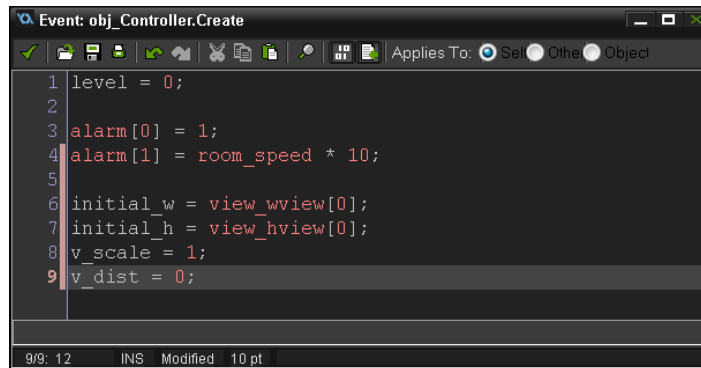


Figure 13-7: Code for the Create Event.

These variables will hold the starting width/height of the view, the current scaling modifier, and the distance between the two touches on the screen.

6. Click the **green check mark**.

The code window is saved and closed.

7. Choose **Add Event**→**Step**→**Step**.

The Step Event appears in the Events section.

8. Drag and drop an Execute Code Action from the Control tab to the Actions section.

A code window appears.

9. In the code window, type the following code:

```
var mx0raw, my0raw, mx1raw, my1raw;
mx0raw = device_mouse_raw_x(0);
my0raw = device_mouse_raw_y(0);
mx1raw = device_mouse_raw_x(1);
my1raw = device_mouse_raw_y(1);
```

This code checks for the raw position (as described right before this procedure) of the first two touches on the screen and stores them in the variables you set up earlier.

10. In the code window, type the following code (shown in Figure 13-8):

```
if device_mouse_check_button(1, mb_left)
{
    if device_mouse_check_button_pressed(1, mb_left)
    {
        v_dist = point_distance(mx0raw, my0raw, mx1raw, my1raw) * v_scale;
    }
    v_scale = v_dist / max(1, point_distance(mx0raw, my0raw, mx1raw,
        my1raw));
    if v_scale < 0.5 v_scale = 0.5 else if v_scale > 1.5 v_scale = 1.5;
    view_wview[0] = initial_w * v_scale;
    view_hview[0] = initial_h * v_scale;
    view_xview[0] = global.Player_x - (view_wview[0] / 2);
    view_yview[0] = global.Player_y - (view_hview[0] / 2);
}
```

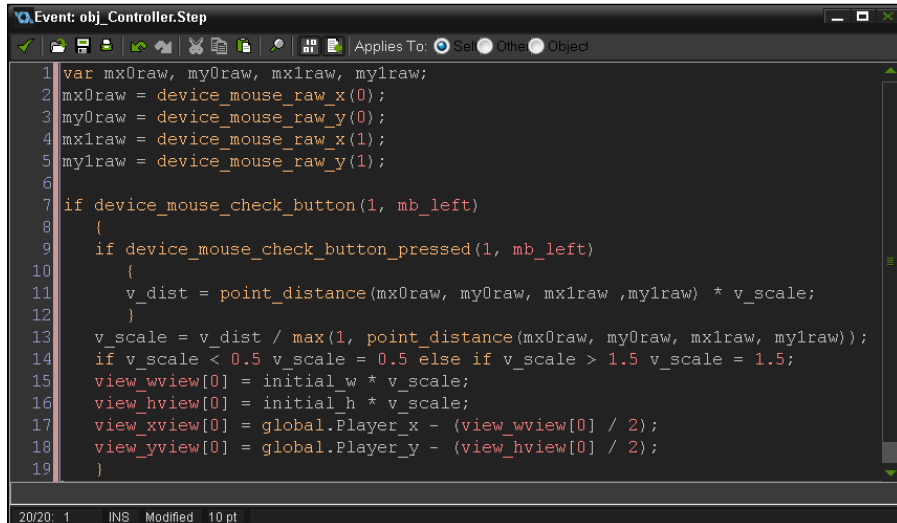


Figure 13-8: Code for the Step Event.

11. Click the green check mark.

The code window is saved and closed.

This code checks to see if a second touch has been detected, and if it has, GameMaker stores in the variable `v_dist` the initial starting distance between the two touch points. GameMaker then uses these values to detect further movement. The resulting change between the initial distance and the new distance is stored in the variable `v_scale`.



Note the use of the `max()` function in the code at line 13 in Figure 13-8. This function prevents any possible divide by zero errors that may occur should the distance be zero.



A *divide by zero error* is when the divisor is 0, which causes an error in your game. Use the `max()` function to prevent the error.

At line 14 of the code, you're telling GameMaker to limit the zoom to half the natural view as a minimum, and 1.5 as maximum (the player can't zoom the view so small that you can't see it and the player can't zoom in so far as to make the pixels blurry). This concept is good to keep in mind when designing your games — different zoom factors will drastically change the end-user experience, especially with the visuals of your game.

The final part of this code — lines 15 through 18 — sets the actual view by multiplying the initial width and height by the scaling value, and then sets the new X and Y position for the view appropriately.

You can test the game now on any mobile device and see that pinching and zooming will cause the screen view to enlarge or shrink. To test the game, from the main menu, choose Run → Run Normally. If you're testing the game on a laptop with a multitouch-capable trackpad enabled, the pinch and zoom functionality should work.

Creating Analog Sticks for Mobile Devices

This section shows you how to create *analog sticks* (virtual joysticks) for mobile devices. You can use analog sticks in your game to greatly improve moving characters around in the game. A benefit to using analog sticks over virtual keys is the enhanced freedom of movement that these virtual joysticks provide.

The procedures in this section are based off the in-software tutorial, *Analogue Sticks*, found on the Tutorials tab of the New Project window. In this section, you see how to add analog sticks to make movement and game play more natural. You'll make two sticks: one to control the player movement and the other to control the player's shooting direction. The player uses her left thumb to control the movement and her right thumb to control the direction.

Drawing the sticks onto the GUI script

You need two Scripts for the analog sticks to work. The Scripts are used to draw the stick onto the GUI layer of the screen. To do so, GameMaker needs to convert the raw mouse positions (as discussed earlier in this chapter) into GUI positions. That's why one Script is for the X position, and the other Script is for the Y position.

To create the X position Script, follow these steps:

1. **From the New Project window, choose Tutorials→Advanced Touch Controls→Analogue Sticks.**

The Analogue Sticks project window appears.

2. **From the Resource tree, right-click Scripts→Create Script.**

A Script code window appears.

3. **In the Name field, type `scr_mouse_gui_x`, as shown in Figure 13-9.**

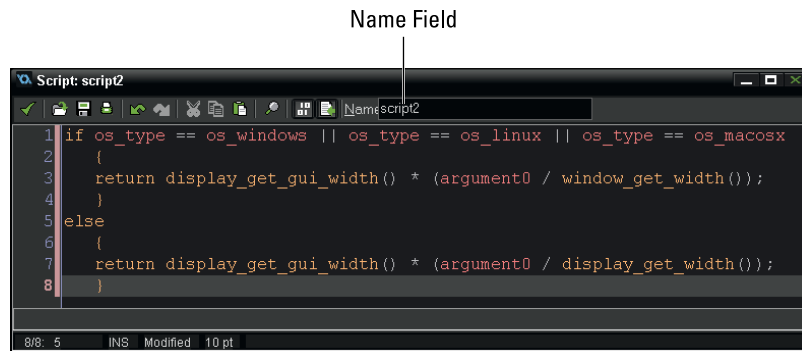


Figure 13-9: Code for the X position Script.

4. **In the Script code window, type the following code (refer to Figure 13-9):**

```
if os_type == os_windows || os_type == os_linux || os_type == os_macosx
{
    return display_get_gui_width() * (argument0 / window_get_width());
}
else
{
    return display_get_gui_width() * (argument0 / display_get_width());
}
```

5. **Click the green check mark from the icon menu.**

The Script code window is saved and closed.

6. From the Resource tree, right-click Scripts and select Create Script.

A Script code window appears.

7. In the Name field, type scr_mouse_gui_y.**8. In the code window, type the following code:**

```
if os_type == os_windows || os_type == os_linux || os_type == os_macosx
{
    return display_get_gui_height() * (argument0 / window_get_height());
}
else
{
    return display_get_gui_height() * (argument0 / display_get_height());
}
```

9. Click the green check mark.

The Script code window is saved and closed.

Later, you call these two Scripts to determine the GUI layer X and Y coordinates from the raw mouse position in the game.



You may have noticed in the first line of code that there are checks to determine the operating system the player is playing on. This is because for Windows and other desktop targets, you need to use the window resolution, whereas on mobile devices, you want the display resolution. These checks help fix any potential errors with placing the sticks on the screen.

Making the player move

Now it's time to enable the player to move. This section picks up where the previous section left off and is a continuation of the Analogue Sticks tutorial.

In the following procedure, you create several new variables for the player Object. These variables are used to translate the mouse position (the player's finger) into movement. The variables are also used to draw the controls on the screen, which is also done in obj_Player.

To create arrays to store values for the two sticks, follow these steps:

1. With the Analogue Sticks tutorial open, from the Resource tree, click to open the Objects folder, and then double-click obj_Player from the Player folder.

The Object Properties window appears.

2. From the Events section, click the Create Event.

The Actions for the Create Event appear in the Actions section.

3. Double-click the Execute Code Action from the Actions section.

The code window appears.

4. At line 12, update `spd=3` to `b_spd=3`.

You need to use the correct variable to get the player to move.

5. In the code window, at line 31, type the following code (as shown in Figure 13-10):

```
gui_x[0] = 64;
gui_y[0] = (display_get_gui_height() - 64);
gui_x[1] = (display_get_gui_width() - 64);
gui_y[1] = (display_get_gui_height() - 64);
stick_x[0] = gui_x[0];
stick_y[0] = gui_y[0];
stick_x[1] = gui_x[1];
stick_y[1] = gui_y[1];
st_pressed[0] = -1;
st_pressed[1] = -1;
st_radius = 40;
vdir = 0;
```

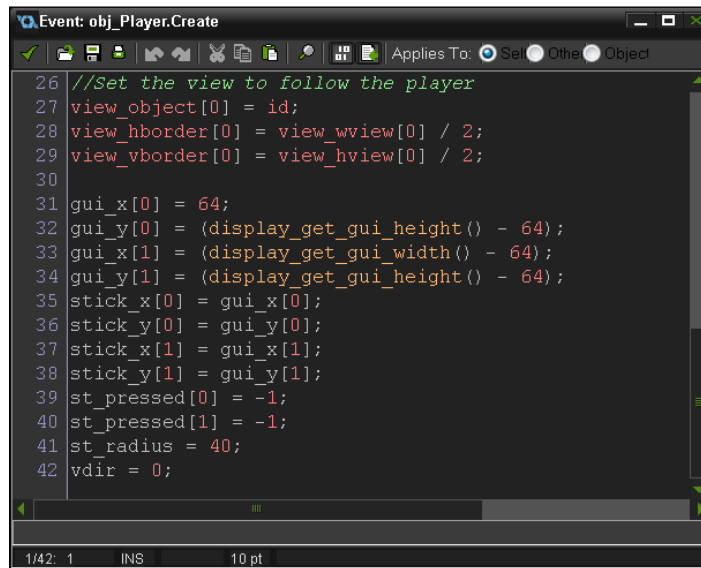


Figure 13-10: Code for the arrays.

6. Click the green check mark.

The code window is saved and closed.

This code creates arrays to hold the X and Y coordinates. Arrays make it easier to group multiple values under the same variable. In this code, you use the array [0] for the left stick, and array [1] for the right stick.



You may need to fix some of the code that came with the tutorial. In line 27, you may see, `/*view_object[0] = id;`. You need to delete the slash and the asterisk to get your code to look like Figure 13-10.

Here's what the different variables do:

- `gui_x/y`: These arrays hold the position on the GUI layer that GameMaker will consider as the center position. They're used to calculate movement. You'll use them to draw the base images of the analog stick on the screen.
- `stick_x/y`: These arrays hold the position of the analog stick while it's being moved by the player, as well as for drawing the stick on the screen.
- `st_pressed`: Use this variable to check for a touch on the screen.
- `st_radius`: This is the radius (in pixels) of the analog stick base. It's used to control the maximum movement of the stick from the base position.
- `vdir`: This variable controls the stick movement for the direction that the player is facing. This is used to set the image angle only when the player is using the stick.

Now you need to update the Step Event. To update the Step Event to support analog sticks, follow these steps:

1. **With the `obj_Player` Object Properties window open, click the Step Event from the Events section.**

The Actions for the Step Event appear in the Actions section.

2. **Double-click the Execute Code Action from the Actions section.**

The code window appears.

3. **Delete lines 8 through 12, highlighted in Figure 13-11.**

Delete the indicated code

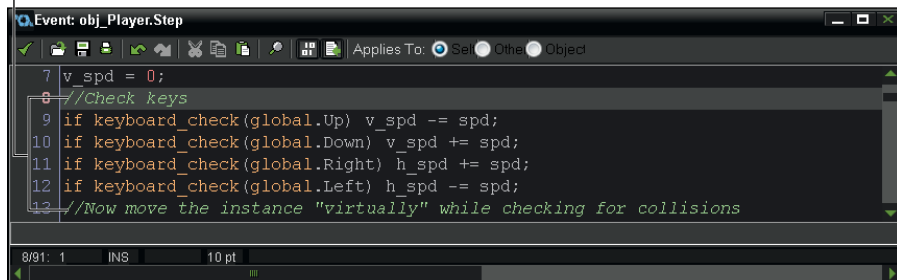


Figure 13-11: Delete this code to remove the keyboard controls.

You just removed the keyboard controls by deleting that code. Now you can add the code for the analog sticks where the code for the keyboard controls used to be. You do this in two parts. The first part of the code detects *which* finger is over the controller and then stores that value in the array `st_pressed`. If the screen is not being touched (at the stick position), this array will hold the value of `-1`. The second part of the code is used to get the position of any detected touch relative to the base GUI position (as defined in the array `gui_x/y`) and that sets the player's horizontal and vertical speed values.

4. At line 8, type in the following code (see Figure 13-12 for the full code block):

```
stick_x[0] = gui_x[0];
stick_y[0] = gui_y[0];
if st_pressed[0] = -1
{
    for (i = 0; i < 5; i++;)
    {
        if device_mouse_check_button(i, mb_left)
        {
            var tx, ty;
            tx = scr_mouse_gui_x(device_mouse_raw_x(i));
            ty = scr_mouse_gui_y(device_mouse_raw_y(i));
            if point_distance(tx, ty, gui_x[0], gui_y[0]) < st_radius
            {
                if st_pressed[1] != i
                {
                    st_pressed[0] = i;
                    break;
                }
            }
        }
    }
}
```

This code block checks to see if there is no current touch on the left stick, and if not, the code loops through the five possible touch points and checks to see if there is a finger there (making sure that it is not the same finger as the one using the shoot controller). If there is a touch within the given radius from the base GUI X/Y point, that touch is detected and the finger number is stored.

Note that this code calls the Scripts you wrote earlier in this chapter to convert the raw mouse position on the screen into the correct GUI layer position.

5. Type in the following code block directly below the code you typed in Step 4 (this code should start at line 30):

```
if device_mouse_check_button(st_pressed[0], mb_left)
{
    var spd, vdist, mx, my;
    mx = scr_mouse_gui_x(device_mouse_raw_x(st_pressed[0]));
    my = scr_mouse_gui_y(device_mouse_raw_y(st_pressed[0]));
```

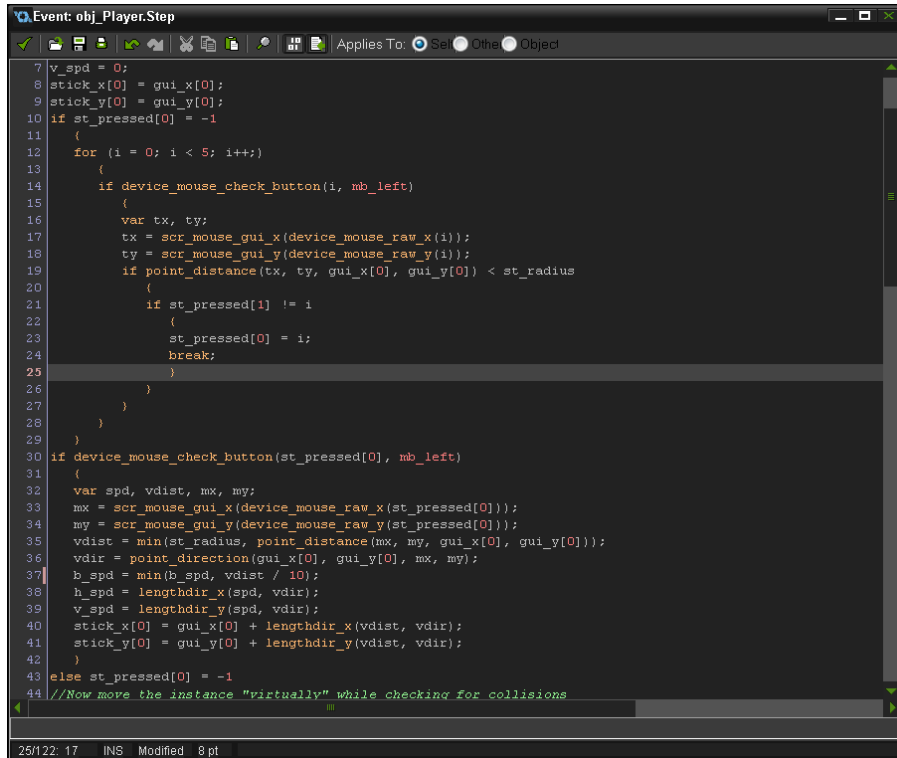



Figure 13-12: The code block for Steps 4 through 8.

This code checks to see if the previously stored finger is still touching the screen. If it is, the local variables defined here get the mouse position on the GUI layer.

6. Next, type in this code at line 35:

```
vdist = min(st_radius, point_distance(mx, my, gui_x[0], gui_y[0]));
vdir = point_direction(gui_x[0], gui_y[0], mx, my);
```

This code captures the distance that the stick has been moved, limiting it to the radius value that you set earlier in the Create Event, and also gets the direction that the stick has been moved in. Both of these values will now be used to calculate the movement speed, which you write the code for next.

7. Type in this code directly below the previous code (at line 37).

```
spd = min(b_spd, vdist / 10);
h_spd = lengthdir_x(sp, vdir);
v_spd = lengthdir_y(sp, vdir);
```

This code sets the local `spd` variable to a value that is limited to a maximum speed set by the base speed (`b_spd`), and then GameMaker uses this value to get the component horizontal and vertical speed from the speed/direction vector. These two values are then used in the movement code of this object.

8. Type in this code directly below the previous code (at line 40):

```
stick_x[0] = gui_x[0] + lengthdir_x(vdist, vdir);
stick_y[0] = gui_y[0] + lengthdir_y(vdist, vdir);
}
else st_pressed[0] = -1
```

This code sets the stick array to hold the current position for drawing the stick head in the Draw GUI Event, and it also resets the `st_pressed` variable to `-1` if the player lifts his finger from the screen.

9. At line 111, you should see the `point_direction` function; update that code to reflect `vdir` instead, as shown in Figure 13-13.

This code change will make the player point in the correct direction when the stick is being moved.

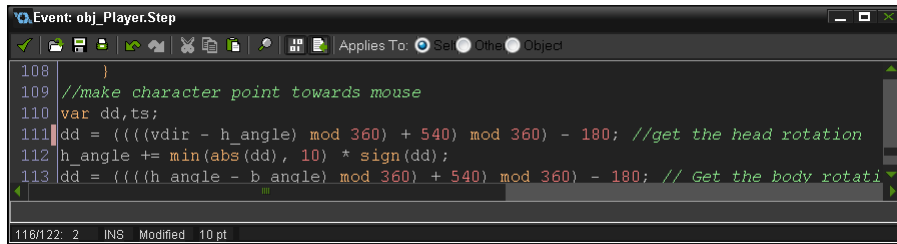


Figure 13-13: Updated code to get the head direction.

10. Click the green check mark.

The code window is saved and closed.

11. Choose Add Event→Draw→Draw GUI.

The Draw GUI Event appears in the Events section.

12. Drag and drop an Execute Code Action from the Control tab to the Actions section.

A code window appears.

13. In the code window, type the following code:

```
draw_set_alpha(0.75);
draw_circle_color(gui_x[0], gui_y[0], st_radius, c_lime, c_lime, true);
draw_circle_color(stick_x[0], stick_y[0], 20, c_lime, c_lime, true);
draw_circle_color(gui_x[1], gui_y[1], st_radius, c_lime, c_lime, true);
draw_circle_color(stick_x[1], stick_y[1], 20, c_lime, c_lime, true);
```

```
draw_set_alpha(0.25);
draw_circle_color(gui_x[0], gui_y[0], st_radius, c_lime, c_lime, false);
draw_circle_color(stick_x[0], stick_y[0], 20, c_lime, c_lime, false);
draw_circle_color(gui_x[1], gui_y[1], st_radius, c_lime, c_lime, false);
draw_circle_color(stick_x[1], stick_y[1], 20, c_lime, c_lime, false);
draw_set_alpha(1);
```

This code draws a representation of the stick on the screen. This allows the player to know where to place his fingers and thumbs.

14. Click the green check mark.

The code window is saved and closed.

You can draw the stick in any way you choose, but in this example, you drew green circles, as shown in Figure 13-14. Note that you also draw the second stick on the right (for shooting) although it's useless at this point. You can test the game now, and you'll see the analog sticks, even if they aren't fully functional yet.

If you run and test your game now, you should see the green circles at the left- and right-bottom corners, as shown in Figure 13-14. If you see the virtual keys (the hollow white squares), then you need to open `obj_Controller`, and in the Create Event, open the Execute Code Action and delete the following code:

```
key_left = virtual_key_add(32, display_get_gui_height() - 160, 64, 64, vk_left);
key_up = virtual_key_add(96, display_get_gui_height() - 224, 64, 64, vk_up);
key_down = virtual_key_add(96, display_get_gui_height() - 96, 64, 64, vk_down);
key_right = virtual_key_add(160, display_get_gui_height() - 160, 64, 64, vk_
    right);
virtual_key_show(key_left);
virtual_key_show(key_up);
virtual_key_show(key_down);
virtual_key_show(key_right);
```

Shooting with the analog stick

In the previous section, you created analog sticks for moving and shooting. Currently, you can turn with the left stick, but the right stick doesn't do anything. You update this so that the player can move in one direction and shoot in another. The procedures in this section pick up from the Analogue Stick tutorial.

To shoot with the analog stick, follow these steps:

1. **With the Analogue Stick tutorial open, from the Resource tree, double-click `obj_Player`.**

The Object Properties window appears.

2. **Click the Step Event from the Events section.**

The Actions for the Step Event appear in the Actions section.

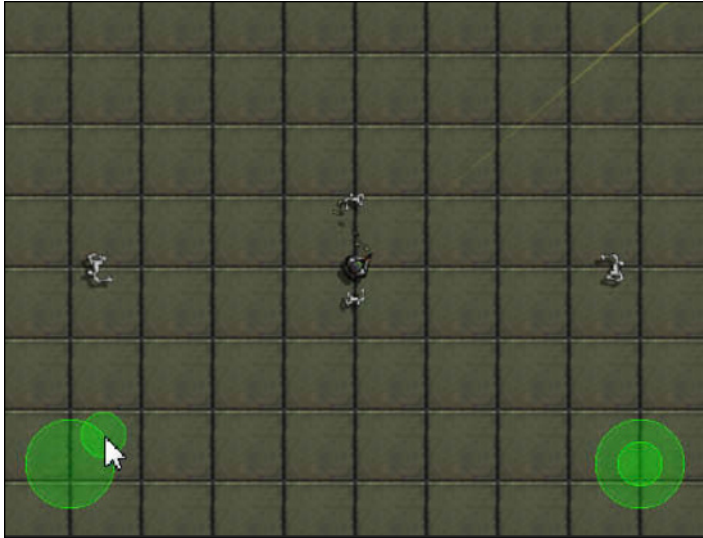


Figure 13-14: The analog sticks.

3. Double-click the Execute Code Action.

The code window appears.

4. At line 82 in the code window, type in the following code, as shown in Figure 13-15.

```
stick_x[1] = gui_x[1];
stick_y[1] = gui_y[1];
if st_pressed[1] = -1
{
    for (i = 0; i < 5; i++;)
    {
        if device_mouse_check_button(i, mb_left)
        {
            var tx, ty;
            tx = scr_mouse_gui_x(device_mouse_raw_x(i));
            ty = scr_mouse_gui_y(device_mouse_raw_y(i));
            if point_distance(tx, ty, gui_x[1], gui_y[1]) < st_radius
            {
                if st_pressed[0] != i
                {
                    st_pressed[1] = i;
                    break;
                }
            }
        }
    }
}
```

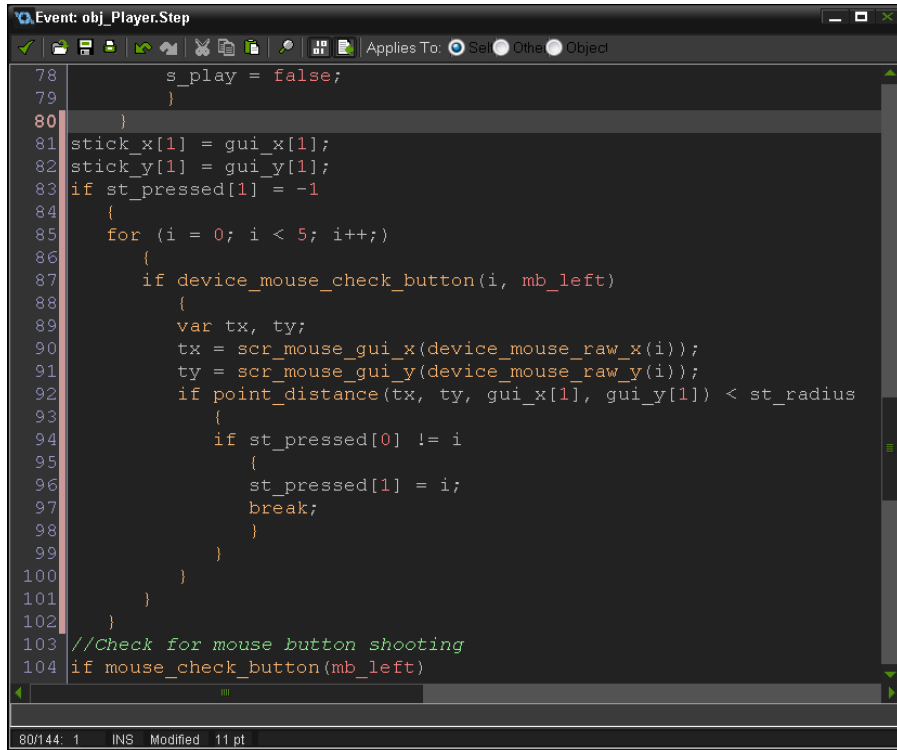


Figure 13-15: Code for the shooting analog stick.

5. Now you need to change the shooting code to accept the touch on the stick and set the correct angles for shooting and drawing; replace the initial `mouse_check_button` function with a device-specific one and set the stick array values like this (see Figure 13-16):

```

if device_mouse_check_button(st_pressed[1], mb_left)
{
    var mx, my;
    mx = scr_mouse_gui_x(device_mouse_raw_x(st_pressed[1]));
    my = scr_mouse_gui_y(device_mouse_raw_y(st_pressed[1]));
    vdist = min(st_radius, point_distance(mx, my, gui_x[1], gui_y[1]));
    vdir = point_direction(gui_x[1], gui_y[1], mx, my);
    stick_x[1] = gui_x[1] + lengthdir_x(vdist, vdir);
    stick_y[1] = gui_y[1] + lengthdir_y(vdist, vdir);
    //check for the shooting variable
    if c_shoot
    {
        //the rest of the code as it is here
    }
}
else st_pressed[1] = -1;

```

6. Click the green check mark.

The code window is saved and closed.

Test the game now and you should see that the left control turns and moves the player and the right control makes him shoot and turn. You may notice that you can't shoot and move at the same time, but perhaps that's a challenge you can take on!

```

102 //Check for mouse button shooting
103 if device_mouse_check_button(st_pressed[1], mb_left)
104 {
105     var mx, my;
106     mx = scr_mouse_gui_x(device_mouse_raw_x(st_pressed[1]));
107     my = scr_mouse_gui_y(device_mouse_raw_y(st_pressed[1]));
108     vdist = min(st_radius, point_distance(mx, my, gui_x[1], gui_y[1]));
109     vdir = point_direction(gui_x[1], gui_y[1], mx, my);
110     stick_x[1] = gui_x[1] + lengthdir_x(vdist, vdir);
111     stick_y[1] = gui_y[1] + lengthdir_y(vdist, vdir);
112
113     //Check for the shooting variable
114     if c_shoot
115     {
116         //Setup vars
117         var xx, yy, mod_dif;
118         c_shoot=false; //Set shoot var to false
119         alarm[0] = room_speed / 3; //Set shoot alarm
120         audio_play_sound(snd_GunShoot_Pistol, 0, false); //Play shot sound
121         mod_dif = b_angle + 3 - random(6); //Set the random angle for bullet variation
122         xx = x + lengthdir_x(20, mod_dif); //Set the x/y position to create the bullet
123         yy = y + lengthdir_y(20, mod_dif);
124         instance_create(xx, yy, obj_Flash); //Create the muzzle flash
125         with (instance_create(xx,yy,obj_Player_Bullet)) //create the bullet
126         {
127             dir = mod_dif; //Set the bullet direction
128         }
129         //Now we will create the bullet casing as a nice special effect
130         with (instance_create(x + lengthdir_x(18, b_angle), y + lengthdir_y(18, b_angle), obj_Casing))
131         {
132             image_index = 0;
133             direction = (other.b_angle + 90) - 15 + random(30);
134             image_angle = direction - 90;
135         }
136     }
137 else st_pressed[1] = -1
138 }
139 //make character point towards mouse

```

Figure 13-16: Code for the Step Event.

Saving and Loading

In This Chapter

- ▶ Making sense of the file system
- ▶ Looking at the ins and outs of INI files and text files
- ▶ Enabling the player to save a game
- ▶ Loading a saved game
- ▶ Mapping the gameplay keys and resetting the default

This chapter provides some basic information on how to add some functionality to your games that the player may find useful. This includes game options, such as turning the background music off or on. This type of functionality is performed using file functions within GameMaker and does require that you have some coding skills with the GameMaker Language (GML). With that said, I provide all the code that you need so that you can set up your game with some useful features for the player.



This chapter is geared toward creating a game for the Windows target module. However, everything I discuss in this chapter works for all other platforms (with slight changes).



This chapter is written in tandem with the Loading and Saving tutorial found within GameMaker. If you start with that tutorial, all the Objects, Sprites, Sounds, and so on come preloaded, which means you can focus on learning GML. I recommend that you play the game first to get an idea of what this tutorial comes preloaded with. Finally, don't be surprised if some of the functionality doesn't work — you're going to fix that as you read through this chapter.

Understanding the File System

Saving and loading functionality — for everything from the player mapping keys to saving games to creating a saved leaderboard — relates to writing to and reading from external files. This chapter focuses on saving and loading game configurations.

The first term that you should learn is the *sandbox*. The sandbox is where GameMaker saves and loads files. These files are found within the *game bundle* (the files GameMaker makes) and the *local storage* (your hard drive). The player has to provide input for data to be saved or loaded.



The inputs discussed here relate only to the Windows target platform.

The sandbox is divided into the game bundle and the save area:

- ✓ **The game bundle:** All the files that are part of the executable game, including any files you added to the Included Files in the Resource tree. The Included Files are external files that you might package with the game. The player can't write to the game bundle.
- ✓ **The save area:** Located on the player's computer. The save area is where game data is written and saved, and can be loaded from later. The save area consists of reading and writing operations within the *working_directory*.
 - **Reading operations** is used when the game wants to save data. GameMaker first checks to see if the file being accessed exists in the save area. If the file doesn't exist, GameMaker searches the game bundle for the Included File.
 - **Writing operations** can only happen in the save area.



The sandbox is available on each target platform, but it behaves differently on each one. Don't worry, though — GameMaker makes it easy to target your games to many different platforms. You can export the games you make with GameMaker to several different platforms:

- ✓ **Windows:** Windows saves all files in the %localappdata%\Game Name directory (on Windows 7 this is the /Users/<User Name>/AppData/Local/Game Name directory).
- ✓ **Mac OS:** Storage depends on whether the application is sandboxed or not (following Apple's rules).
- ✓ **HTML5/Windows 8 (JS):** Everything is done through the local storage.
- ✓ **iOS:** Storage is the standard location, as viewed through iTunes.
- ✓ **Android:** Files are in the standard location (which is invisible unless the device is rooted), /data/data/package name.
- ✓ **Windows Phone:** Files are in the same location as the app.



For Android, iOS, and Windows Phone, you must use the *working_directory* prefix to access stored files. Using just the relative path doesn't work. In addition, the HTML5 and Windows 8 platforms have a limit on how much local storage (between 1 MB and 5 MB) you can use for your games. This means that you need to be cognizant of how large your Sprites and so forth are.

Understanding INI Files

You set up save game options through *INI files*, which are used by all Windows programs to store data (and are perfectly acceptable to use for all platforms). You can create an INI file by using a plain text editor such as Notepad, which comes standard with Windows. You then bundle that INI file into your game. I show you how to do that in this section.



You don't have to include the INI files in your game bundle, but it's considered best practice to do so. If you don't bundle an INI file, then when the player tries to save a game option, an INI file is created automatically. However, the auto-generated INI file does not create any *reset mechanisms*. A reset mechanism enables the player to revert to the game defaults. For example, say that the player mapped B as the jump key, but then later realized that the default use of the spacebar served better for jumping. By providing a reset mechanism to the player (generally in the form of a button), she can revert back to the default game setting. Another important reason for creating your own INI files is that you take control of the format, and this enables players to save information from different localizations.

An example of an INI file for game options could look like this:

```
[Options]
music = 1
sound = 1
diff = 2
name = Anon
```

All INI files are broken up into sections. In this code block, you see the section for Options. Within this section are the *keys* (such as `music`, `sound`, `diff`, and `name`); each key has its own *value* (such as 1, 2, and `Anon`), which can either be a *real* (say, a number) or a *string* (a sequence of characters) data type.



If you open the Loading and Saving tutorial within GameMaker, and then look in the Resource tree under Included Files, you see `Save.ini`. You can open this file by double-clicking on it, which opens Notepad and you should see Figure 14-1.

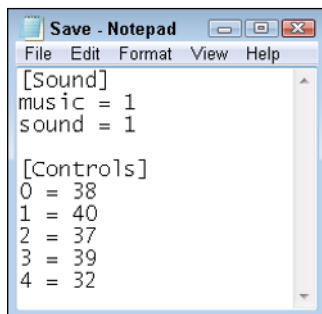


Figure 14-1: The INI file from the Saving and Loading tutorial.



When you're creating your own INI file in Notepad, be sure to save it as *.ini and not *.txt.

To save a TXT file as an INI file with Notepad, follow these steps:

1. From within Notepad, choose File⇒Save As.

The Save As dialog box appears.

2. In the File Name field, change the suffix from .txt to .ini.

3. Click the Save As Type drop-down and choose All Files.

4. From the Encoding drop-down list, choose UTF-8 (as shown in Figure 14-2).

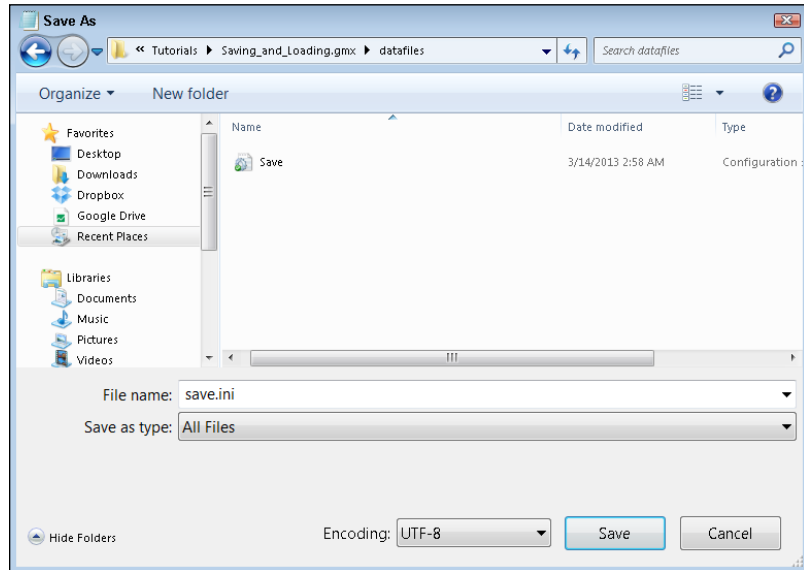


Figure 14-2: The Save As dialog box.



Make sure to save your INI file as UTF-8 from the Encoding drop-down. This will save your game from errors when players from places other than the United States or the United Kingdom try to save data.



During the game, when the player chooses the different options, it's the INI file that's copied over to the player's local storage (the save area as discussed earlier in this chapter). The *base* file — the INI file that you saved in the Included Files and is part of the game bundle — won't be overwritten; instead, the player's choices are saved on the local computer. This follows the sandboxing rules mentioned earlier.

Using Real Values to Save Game Options

In this section, I show you how to create a couple buttons that the player can use to toggle the background music or the sound effects off or on. To do this, you update code blocks in three different files. Two of these code blocks are found within the Objects that were made for the background music and the sound effects. The third block of code is found in the Scripts section of the Resource tree (assuming you're working from the in-software Saving and Loading tutorial).

The following procedures show you how to create code using real values (numbers) that GameMaker uses to determine if the player has selected the sound effects and background music to be off or on (to play or not to play, that is the toggle). In the next section, I show you how to use strings (characters or numbers) so that the player can save his name.

Making a button for the background music

If you haven't done so already, go ahead and open the Saving and Loading tutorial that comes with GameMaker. Run the game and take note of the buttons for toggling the sound effects, as shown in Figure 14-3.



Figure 14-3: The sound options buttons in the Saving and Loading tutorial.

If you click those buttons now, you see them push in, so to speak, but they don't actually do anything. To get them to work, follow the next procedure.

To make the musical note button work, follow these steps:

1. From the New Project window, select the Tutorials tab.
2. Select the Saving and Loading Tutorial under Intermediate in the left-side column.

The Saving and Loading Project window appears.

3. From the Resource tree, in the Objects section, click to open the Start Room folder.

The contents of the Start Room folder appear.

4. Double-click `obj_Music_Button`.

The Object Properties window appears.

5. Select the Step Event from the Events section, as shown in Figure 14-4.

The Actions for the Step Event appear in the Actions section.

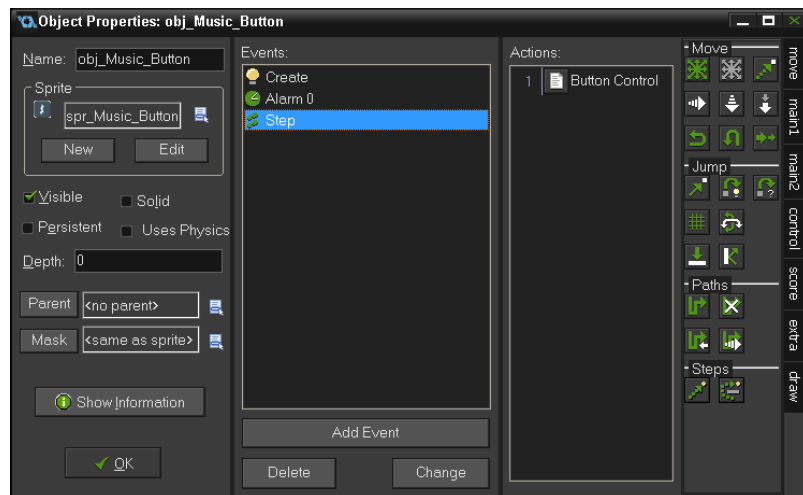


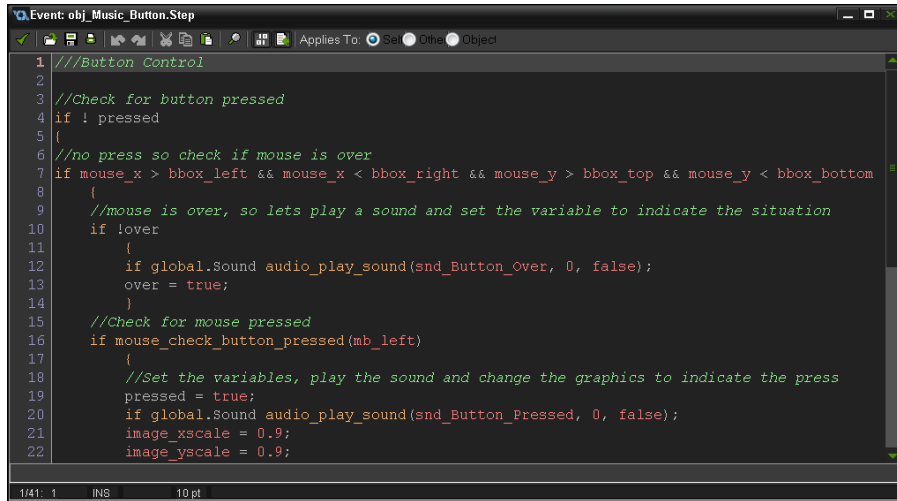
Figure 14-4: Select the Step Event.

6. From the Actions section, double-click Button Control (refer to Figure 14-4).

Button Control is an Execute Code Action that was named by using a comment with three backslashes (see Figure 14-5).

The code window appears. You can use the code window as a code editor to edit, write, and delete code.

The code shown in Figure 14-5 checks for when the player presses the mouse button over the music button in the game. (See Chapter 11 for information on the color coding in the code window.)



```

1  ///Button Control
2
3  //Check for button pressed
4  if ! pressed
5  {
6      //no press so check if mouse is over
7      if mouse_x > bbox_left && mouse_x < bbox_right && mouse_y > bbox_top && mouse_y < bbox_bottom
8      {
9          //mouse is over, so lets play a sound and set the variable to indicate the situation
10         if !over
11         {
12             if global.Sound audio_play_sound(snd_Button_Over, 0, false);
13             over = true;
14         }
15         //Check for mouse pressed
16         if mouse_check_button_pressed(mb_left)
17         {
18             //Set the variables, play the sound and change the graphics to indicate the press
19             pressed = true;
20             if global.Sound audio_play_sound(snd_Button_Pressed, 0, false);
21             image_xscale = 0.9;
22             image_yscale = 0.9;

```

Figure 14-5: The code window for the Button Control Execute Code Action.

- 7. Place the cursor after the `alarm[0] = room_speed / 4;` (line 24), press Enter, and type the following code (see Figure 14-6):**

```

//Check the ini file and write the changes
ini_open("Save.ini");
global.Music = !global.Music
ini_write_real("Sound", "music", global.Music);
if global.Music audio_play_music(snd_Music, true) else
    audio_stop_music();
ini_close();
}

```

The code window should now look like Figure 14-6.

- 8. Click the green check mark from the icon menu.**

The code window is saved and closed.

This code does a few things. When the player presses the button, the global controller variable is toggled between true or false (on or off). The next line checks the value and then either starts or stops the music. From there, the `Save.ini` file is opened, then the new value is written, and then the INI file is closed. If you've been following this chapter from the beginning, you should already be familiar with the `Save.ini` file that came preloaded with this tutorial.



You should always make sure that any `ini_open()` function is followed by `ini_close()`. Failure to close the INI file will cause errors during gameplay when the player tries to make changes to the game settings later on.

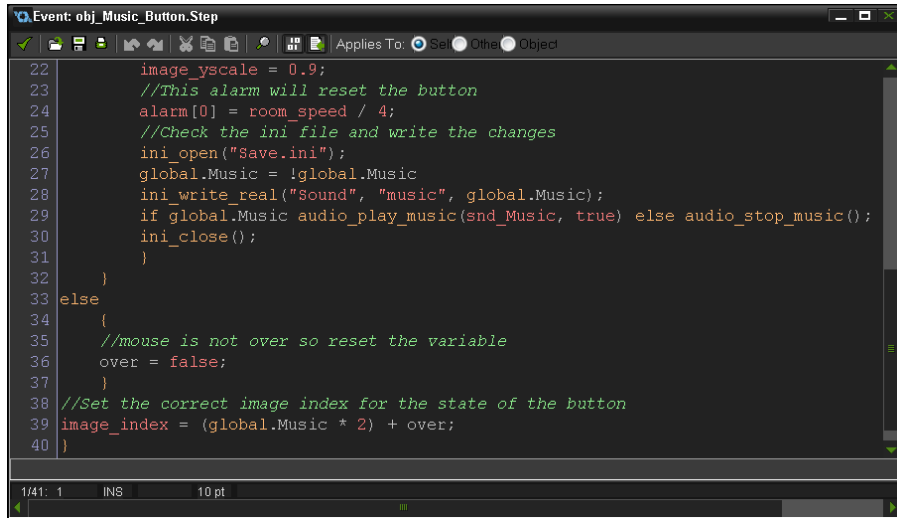


Figure 14-6: The code window for obj_Music_Button in the Step Event.

Now you need for the game to check the saved values to see whether the music should play. I'm also going to show you how to check the saved values for the sound effects button (this will save you the trouble of coming back later and updating the code a second time).

To check the saved value, follow these steps:

- 1. From the Resource tree, under Scripts, double-click scr_Init_Globals.**

The script window appears.

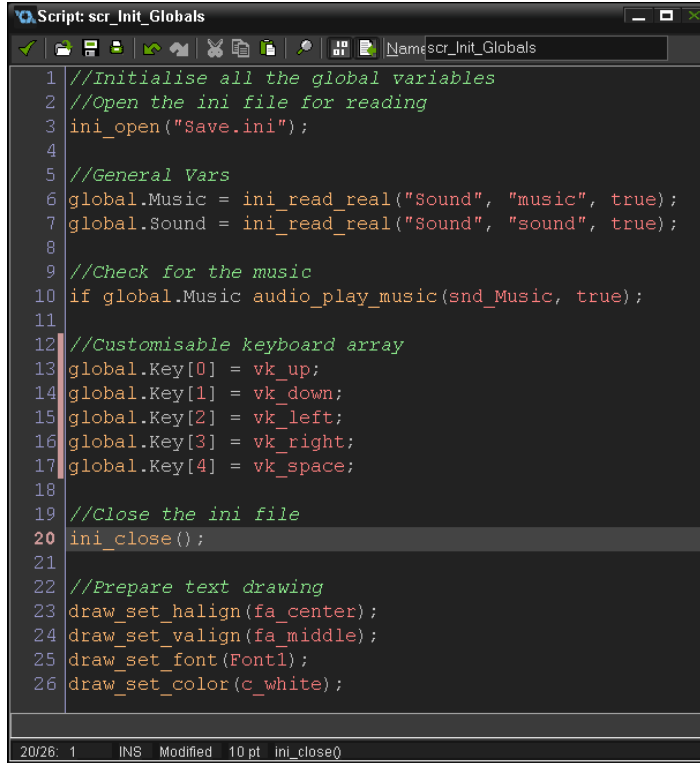
- 2. Update the code to reflect Figure 14-7.**

You won't have to type in all this, but check it line-for-line to see the differences. Don't forget to add in lines 19 and 20 to close the INI file.

- 3. Click the green check mark from the icon menu.**

The script window is saved and closed.

Now you can save and run the tutorial to see if you wrote the code correctly. If you did, when you click the musical note button, the background music should stop playing and the icon on the button should change. If you click the button again, the music should start playing, and you see the musical note icon again. If you click Quit to close the game, and then run the game again, your last option should have been saved. (If you Quit the game with the music playing, it should be playing when you restart; if you Quit the game with the music turned off, then it should still be off when you start the game again).



```

1 //Initialise all the global variables
2 //Open the ini file for reading
3 ini_open("Save.ini");
4
5 //General Vars
6 global.Music = ini_read_real("Sound", "music", true);
7 global.Sound = ini_read_real("Sound", "sound", true);
8
9 //Check for the music
10 if global.Music audio_play_music(snd_Music, true);
11
12 //Customisable keyboard array
13 global.Key[0] = vk_up;
14 global.Key[1] = vk_down;
15 global.Key[2] = vk_left;
16 global.Key[3] = vk_right;
17 global.Key[4] = vk_space;
18
19 //Close the ini file
20 ini_close();
21
22 //Prepare text drawing
23 draw_set_halign(fa_center);
24 draw_set_valign(fa_middle);
25 draw_set_font(Font1);
26 draw_set_color(c_white);

```

Figure 14-7: The script window for scr_Init_Globals.

Making a button for the sound effects

Now, it's time to take what you just learned and apply it to the second button, which toggles the sound effects in the game. In the following procedure, you update the obj_Sound_button Object from the Saving and Loading tutorial.

To make the sound effects button work, follow these steps:

1. **With the Saving and Loading tutorial open, from the Resource tree, in the Objects section, click to open the Start Room folder and then click obj_Sound_Button.**

The Object Properties window appears.

2. **Select the Step Event from the Events section.**

The Actions for the Step Event appear in the Actions section.

3. **Double-click Button Control from the Actions section.**

This opens the code window for the Execute Code Action.

4. Find the section of code that checks for when the mouse button is pressed.

That code starts at line 15 with the comment, *//Check for mouse pressed*.

5. Type in the following code so that it reflects Figure 14-8.

```
//Check the ini file and write the changes
ini_open("Save.ini");
global.Sound = !global.Sound
ini_write_real("Sound", "sound", global.Sound);
ini_close();
```

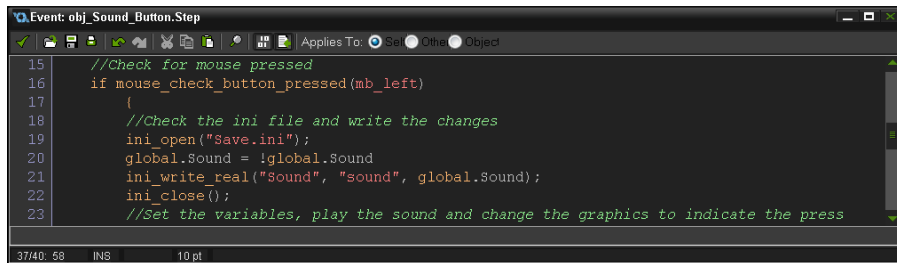


Figure 14-8: The code window for the Sound Button Step Event.

6. Click the green check mark.

The code window is saved and closed.

If you were following along from the last section, you've already updated the `Init_Globals` script. If not, go back and follow the procedure before this one.

If you run the game now, both the music and sound effect buttons should work. When you click the background music or the sound effect button off or on, the icon on the button changes, indicating that the sound is either off or on. When you close the game and reopen it, your last options should be saved.

Using String Values to Save Game Options

In the previous section, I show you how to save game options using real values, which works well if you want to toggle options off or on. In this section, I show you how to write code so that the player can save his name within the game. This is useful if you want to create a saved leaderboard so that different players can compare scores. This can lead to competition and people wanting to play your game more and more to get better and better and to score higher and higher!

To create code that enables the player to save his name, follow these steps:

1. **With the Saving and Loading tutorial open, from the Resource tree, in the Objects section, click to open the Name Room folder and then click obj_Name_Control.**

The Object Properties window appears.

2. **Select the Create Event from the Events section.**

The Actions for the Create Event appear in the Actions section.

3. **Double-click the Execute Code Action from the Actions section.**

The code window appears.

4. **Type in the following code at line 12, as shown in Figure 14-9.**

```
//Get any stored name
ini_open("Save.ini");
keyboard_string = ini_read_string("Player", "Name", "Anon");
ini_close();
```

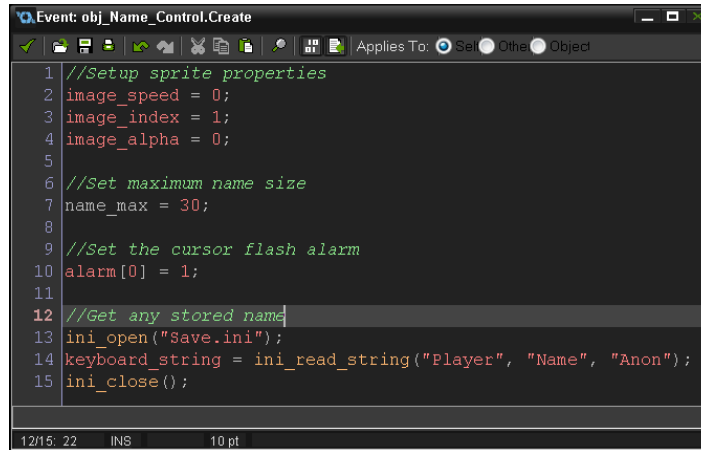


Figure 14-9: The code window for the Create Event.

5. **Click the green check mark.**

The code window is saved and closed.

This code opens the `Save.ini` file, checks for a name, and if there is none, returns the value of `Anon`. After you complete the procedure, when you run the game and click Next, you should now see `Anon` in the name (see Figure 14-10). To test this, go back and edit the code, but replace `Anon` with, say, `Mike`. Now when you run the game, the default name is `Mike`.



Figure 14-10: Update the code to show Anon as the default name.

Now you need to add some code to save the name the player types in.

To save the player's name, follow these steps:

1. **With `obj_Name_Control` still open, click `Add Event` → `Other` → `Room End`.**

The Room End Event appears in the Events section.

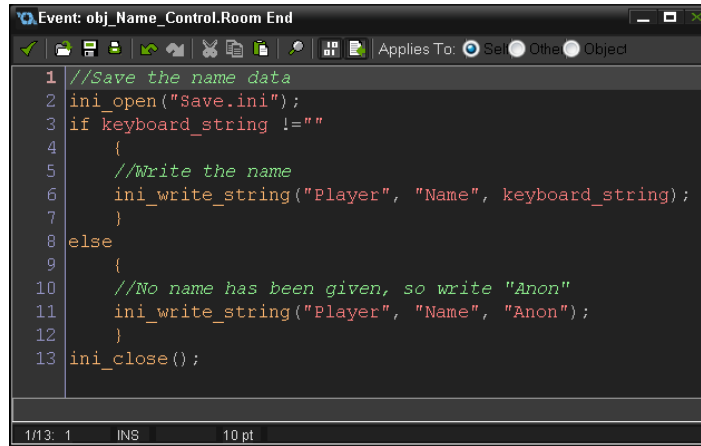
2. **Drag and drop an `Execute Code` Action from the `Control` tab to the `Actions` section.**

A code window appears.

3. **Type in the following code so that the player's name is saved when she leaves that room (for example, clicks `Back` or `Next`), as shown in **Figure 14-11**.**

```
//Save the name data
ini_open("Save.ini");
if keyboard_string != ""
{
    //Write the name
    ini_write_string("Player", "Name", keyboard_string);
}
else
{
    //No name has been given, so write "Anon"
    ini_write_string("Player", "Name", "Anon");
}
ini_close();
```

If the player does not type in a name, GameMaker uses “Anon” as the player’s name.



```

1 //Save the name data
2 ini_open("Save.ini");
3 if keyboard_string != ""
4 {
5     //Write the name
6     ini_write_string("Player", "Name", keyboard_string);
7 }
8 else
9 {
10    //No name has been given, so write "Anon"
11    ini_write_string("Player", "Name", "Anon");
12 }
13 ini_close();
  
```

Figure 14-11: The Room End Event code.

4. Click the green check mark.

The code window is saved and closed.

Test the game now and type in your name. Try clicking Back and Next and see if your name stays. If you close the game and reopen it, your name should still appear.



You must pair your read and write code correctly, or else you’ll get an error during gameplay. If you mix and match your string values with real values (say, you use `ini_read_real()` with `ini_write_string()`), you’ll create an error).

Mapping the Gameplay Keys

In this section, I show you how to update existing code that enables the player to remap keys that are used in the game. Some players may prefer to use W, A, and D to move the character on the screen and some players may prefer the arrow keys. With this code, the player can assign whichever keys they want for the gameplay. This input is also saved to the INI file that you used earlier.

The Saving and Loading tutorial uses a parent button that controls everything. The following procedure shows you how to update the `obj_Key_Parent` to save the player’s choices.

To update the Alarm Event to save mapped keys, follow these steps:

1. **With the Saving and Loading tutorial open, from the Resource tree, click to open the Objects folder, click to open the Options Room folder, and then click obj_Key_Parent.**

The Object Properties window appears.

2. **Select the Alarm 0 Event from the Events section.**

The Actions for the Alarm 0 Event appear in the Actions section.

3. **Double-click the Execute Code Action in the Actions section.**

The code window appears.

4. **Scroll down to line 33, click to insert your cursor at the beginning of the line and press Enter to start a new line, and type the following code (as shown in Figure 14-12):**

```
ini_open("Save.ini");
ini_write_real("Controls", string(keynum), global.Key[keynum]);
ini_close();
```

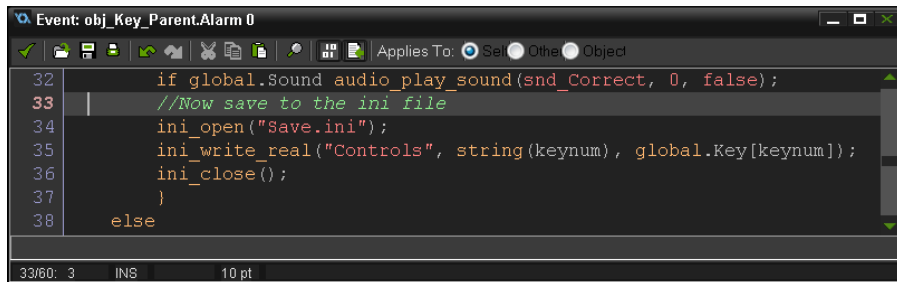


Figure 14-12: The code window for the Alarm 0 Event.

5. **Click the green check mark.**

The code window is saved and closed.

In this code, you're assigning a number (hence, the real value and not a string value) to the key the player chooses. The key can be a letter, a symbol, or a function key, because a number is assigned to each key on the keyboard.

Now, I'll show you how to load the keys that the player assigns. You do this by updating the scr_Init_Globals script. To update the script for loading mapped keys, follow these steps:

1. **With the Saving and Loading tutorial open, from the Resource tree, in Scripts, double-click scr_Init_Globals.**

The script window appears.

2. Scroll down to line 19 (or so) and type in the following code (see Figure 14-13):

```
ini_open("Save.ini")
global.Key[0] = ini_read_real("Controls", "0", vk_up);
global.Key[1] = ini_read_real("Controls", "1", vk_down);
global.Key[2] = ini_read_real("Controls", "2", vk_left);
global.Key[3] = ini_read_real("Controls", "3", vk_right);
global.Key[4] = ini_read_real("Controls", "4", vk_space);
ini_close()
```

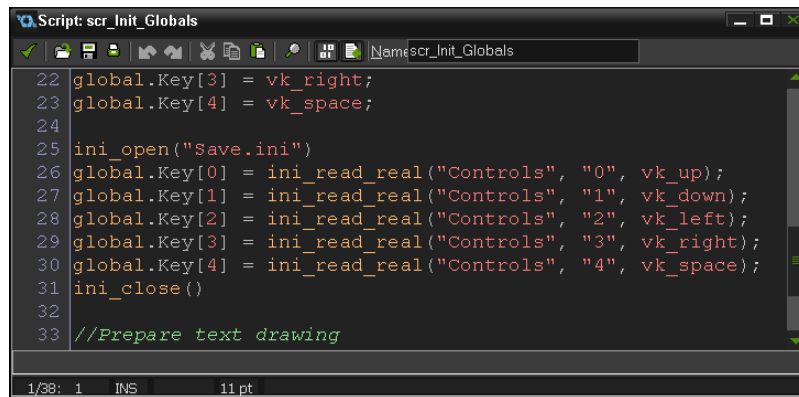


Figure 14-13: The script window for scr_Init_Globals.

3. Click the green check mark.

The script window is saved and closed.

With this procedure complete, the player can now remap the game controls and they'll be saved for later use.

Now it's a good idea to fix the functionality of the Reset Keys button. I show you how to do that in the next section.

Resetting the Default Keys

At this point in the tutorial, the player has the ability to remap the game keys, but the Reset Keys button doesn't do anything. In the next procedure, I show you how to make the Reset Keys button work.



While working in this section, keep in mind the difference between the game bundle files and the saved area files. The game bundle files are never written to; instead, they're copied to the player's local machine and that copy of the file is written to. That creates a player-specific file in the local storage (saved

area), and that's the file GameMaker reads and writes to. At that point, there are two versions of the file: one with the default values in the game bundle and one with player-specific values in the saved area.

When the saved area file is removed, GameMaker can no longer find it to use the saved values. In that case, GameMaker reverts back to using the game bundle file. In that way, the player can revert to the default values.

To make the Reset Keys button work, follow these steps:

- 1. With the Saving and Loading tutorial open, from the Resource tree, click to open the Objects folder, click to open the Options Room folder, and then click obj_Key_Reset.**

The Object Properties window appears.

- 2. Select the Step Event from the Events section.**

The Actions for the Step Event appear in the Actions section.

- 3. Double-click the Execute Code Action from the Actions section.**

The code window appears.

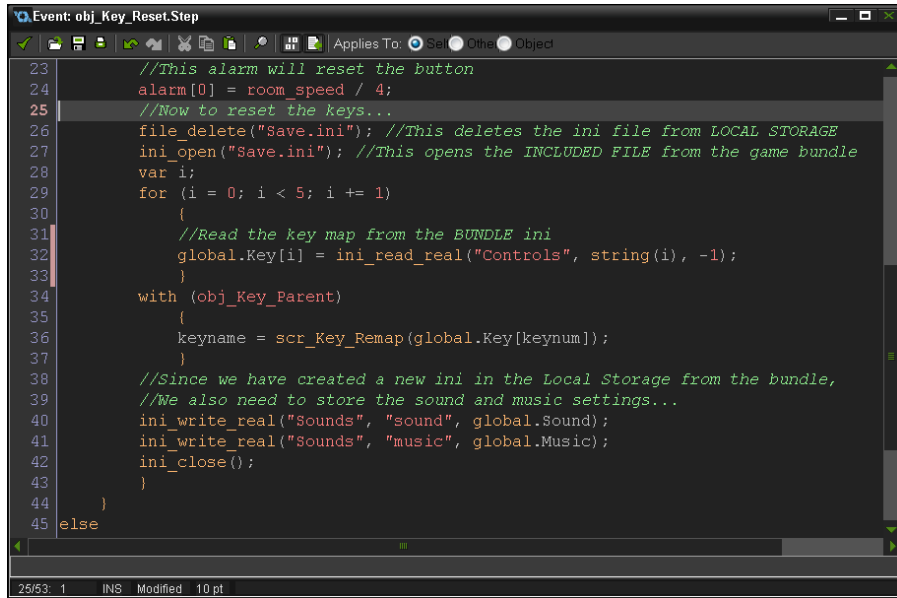
- 4. Insert the cursor at line 25 and type in the following code directly after `alarm[0] = room_speed / 4`, as shown in Figure 14-14:**

```
file_delete("Save.ini");
ini_open("Save.ini");
var i;
for (i = 0; i < 5; i += 1)
{
    global.Key[i] = ini_read_real("Controls", string(i), -1);
}
with (obj_Key_Parent)
{
    keyname = scr_Key_Remap(global.Key[keynum]);
}
ini_write_real("Sounds", "sound", global.Sound);
ini_write_real("Sounds", "music", global.Music);
ini_close();
```

- 5. Click the green check mark.**

The code window is saved and closed.

Now, when the player clicks the Reset Keys button, the default keys should reappear. Figure 14-15 shows a before and after of the mapped keys. Also, if you click the Back button, you should notice the name reset to Anon. Finally, the music options won't change, but if you close and reopen the game, the music options will default back to being on.



```

23 //This alarm will reset the button
24 alarm[0] = room_speed / 4;
25 //Now to reset the keys...
26 file_delete("Save.ini"); //This deletes the ini file from LOCAL STORAGE
27 ini_open("Save.ini"); //This opens the INCLUDED FILE from the game bundle
28 var i;
29 for (i = 0; i < 5; i += 1)
30 {
31     //Read the key map from the BUNDLE ini
32     global.Key[i] = ini_read_real("Controls", string(i), -1);
33 }
34 with (obj_Key_Parent)
35 {
36     keyname = scr_Key_Remap(global.Key[keynum]);
37 }
38 //Since we have created a new ini in the Local Storage from the bundle,
39 //We also need to store the sound and music settings...
40 ini_write_real("Sounds", "sound", global.Sound);
41 ini_write_real("Sounds", "music", global.Music);
42 ini_close();
43 }
44 }
45 else

```

Figure 14-14: The code for the Step Event.

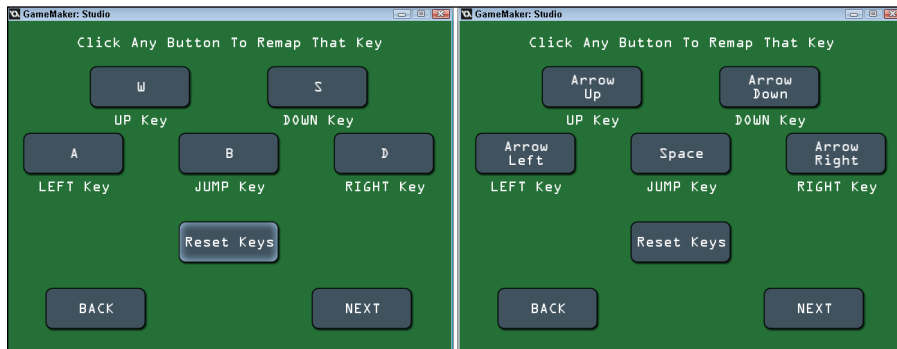


Figure 14-15: Remapped keys (left) and the default values after clicking Reset Keys (right).



If you're not a developer by profession, you may be wondering what this code does. The first part, and probably the most obvious, is that you tell GameMaker to delete the saved INI file on the local storage. From there, GameMaker opens the INI file from the game bundle, and it uses the global key array for the default values. Next, the script is run to show the default values for the keys. Finally, GameMaker writes the current values for the sound as the INI file is rewritten and placed in the saved area (local storage) once again.

Part V

The Part of Tens



For ten great resources for game developers, head to www.dummies.com/extras/gamemakerstudio.

In this part . . .

- ✓ Find more tips on making games with GameMaker: Studio.
- ✓ Get answers to common questions about GameMaker: Studio.

Ten Tips for Game Developers

In This Chapter

- ▶ Thinking ahead as a game developer
- ▶ Knowing where the industry is going

When I was at Game Developers Conference in early 2014, I met members of the YoYo Games team and asked them, “If you had an opportunity to speak with a large audience of people who wanted to make games for a living, what would you tell them?” We talked about the latest trends, advancements in GameMaker: Studio, career advice, and basically what a budding game developer needs to know and do to make it in the industry. This chapter is an offshoot of those conversations.

Developing Cross-Platform Games

One of the first things you need to think about before you start making your game is where and how people are going to play it. Are you making an HTML5 game that people can play on your website? Do you want to make a game that you can submit for inclusion in Apple’s App Store, or Google Play, or even for use on the Sony PS4 or Vita?

Whatever your goal, you need to know the differences between the platforms, such as which code will work where. In addition, you need to be aware of scaling and resolution issues — for example, will your heads-up display (HUD) be just as legible on a large monitor as it will on a handheld device? (Chapter 10 is all about scaling your games.)

You also need to keep in mind how you want the controls to work. For PC games, you probably have keyboard and mouse controls. For a mobile device, you need to think about taps, swipes, and pinching and zooming. (I cover mobile controls in Chapter 13.)



Start with a specific platform and develop your game toward that one platform. Don’t try to make your game work perfectly on all platforms all at once. I recommend developing your game to work on Windows first — that should also work just fine on both iOS and Android platforms.

Knowing Your Audience

Who do you expect to play your game? Are you making a crossword game that you want people to play on Facebook, or are you making a shooter for the PS4? Do you want to make a hardcore game with a high difficulty level, or do you want to make a creative game that allows casual fans to use their imaginations? When you know who your audience is, you'll be able to decide how you want to develop your different levels, if you want a tutorial section, or if you just want players to figure out how to play on their own.

Monetizing Your Games

Making money with your games eventually becomes one of those necessary evils that all developers must face. It would be nice to live in a world in which we could all make and play games for free, but that's not the reality. Companies have salaries to pay and indies have their own bills to pay. So, monetizing your games becomes an important facet that you must incorporate into your games, unless you're independently wealthy and don't care about money.



How you monetize your game is something you want to think about from the beginning. Are you going to charge users to download and install the game? Do you want advertising within the game itself? Or do you want to make a *freemium game*, where the players can buy extras to enhance their game play experience? Many options are available, and you have many advertising services to choose from. Take your time and learn about as many as you can, and then decide for yourself the best option for you and your game.

Choosing Your Game Genre

Some of the best games you can make with GameMaker: Studio are *platformers* (games in which the player controls a character and moves it about the screen, jumping from one place to another, gathering coins, finding loot, avoiding bad guys, and so on). Nintendo's Mario games are excellent examples of the platform genre.

Roguelike games also work well with GameMaker: Studio. Defining a roguelike game is not easy — like many things, the definition depends on who you ask. But at a basic level, a roguelike game is one in which

- ✓ There is random level generation.
- ✓ There is *permadeath*. You don't start with a set number of lives. There are no save points — when you die, you start over from the beginning.

- ✓ It's turn-based, allowing the player to think carefully between each move.
- ✓ The levels typically represent a dungeon, or a room with corridors, halls, and different rooms in which the player encounters bad guys, loot, and perhaps non-playable characters.



Check out the YoYo Games Showcase at www.yoyogames.com/showcase to see examples of many different games that you can make and play with GameMaker: Studio.

Making Your Game User-Friendly

Usability is a concept that goes beyond just platforms and audiences — it dives deep on how the players are to play, and whether you need to teach them how the controls work. For example, if you're making a game for the PC, and you decide to use the letters O, K, and L to move the character around, you may confuse your audience, especially if 99 percent of PC games use W, A, and D to move the character. Think of it this way: If all the other games used the spacebar to fire a gun, and you use the Esc key to fire a gun, you're potentially confusing your audience from the get-go.



Make your game as intuitive as possible, so there's as little guessing as possible for the player to figure out how to play. Another idea is to include an introductory, tutorial level that teaches the player how to play as the game progresses.

Testing Your Game

Testing your game is probably one of the most important things you can do. Quality assurance is what can make the difference between a buggy game that constantly crashes and a bug-free game that you can play without a hitch. So, test your game, then test it again, and then before the end of the day, test it again. Test all aspects of the game as you're developing it. See what happens when you introduce something else to the game — how does it affect aspects of the game that you added before?



Apple and Google won't test your games for you when you submit them to their stores. Those two behemoths are purely interested in having a quantity of games available for sale and they let the best games rise to the top. If your game is full of bugs, it won't rise to the top and your sales rankings will be in the basement with a billion other broken games. On the other hand, if you want to submit your game for sale on the PS4, Sony *will* test your games. But don't use Sony for your quality assurance — you don't want your game to be rejected based on bugs and game crashes.

Considering Limitations for Mobile Devices

In the early days of mobile devices, there were many limitations as to what you could develop. These days, those limitations are quickly disappearing. However, it's still good to know what will crash your game on an iOS device as compared to playing on a PC.



Running out of video memory is the most common reason a game will crash on a mobile device. To decrease the processing power necessary to run your game, there are several things you can do. This includes compressing your images, making use of Texture Pages and Texture Groups, and optimizing your game as much as possible. (I talk about Texture Settings in Chapter 2.)

Thinking Big

YoYo Games wants you thinking big. Like real big. Like Minecraft meets Grand Theft Auto big. YoYo Games is actually making a game that takes elements from Minecraft, such as building structures, and includes aspects from another huge game like Grand Theft Auto, in which you can get in a vehicle to move about. If you're interested in participating in this project, head to the YoYo Labs (www.yoyogames.com/developers/yoyolabs). There, you can find the source code (<http://yourworld.yoyogames.com/downloads>) for this ambitious project.

Try taking the best features of your favorite games and combine them into one. Then add in that little something extra to give the game your own personal touch. If you can create something new that people already enjoy, making it a unique game that you can call your own, you may have a hit game on your hands.

With all that said, though, you should start off slow and simple. Make some silly games to get familiar with the GameMaker: Studio software and then keep adding to it as you go.

Expanding Your Opportunities

While making your game, always be thinking about the future. How could you improve this game later? How can you get the community involved? Gamers are a vocal group — they'll let you know in no uncertain terms how much they love or hate your game. If you can get your audience involved and make them a part of the game development process, you're that much further ahead. If possible, set up a way to receive feedback from your audience. You could also make your game open source, so that people can contribute to it.

Expanding your game could also mean allowing the community to test it. Allow users access to beta versions of the game so they can try to break it for you. That way, you can learn a lot about how people play your game and what works and what doesn't.

Going Indie

Last, but certainly not least, these days the video game industry is all about going independent. Some of the biggest games from the past years have been made by small or independent game developers. Games such as Spelunky, Journey, and Minecraft were all made by indie developers. You may not think of Minecraft as an indie game because of its enormous popularity, but once upon a time, it was just a single developer who wanted to make a game. Now, that indie developer has started his own studio with a staff of employees.

You no longer have to work for companies like Activision, Electronic Arts, or Bethesda to earn a living as a game developer. I'm sure there are lots of happy developers working for those large companies, but the fact is, you don't have to limit yourself to a few big companies. New indie game companies are springing up all the time. And if you're good enough, there's no reason why you can't be among them!

Ten (Or So) Frequently Asked Questions about GameMaker: Studio

In This Chapter

- ▶ Getting more tips and tricks
 - ▶ Looking at different types of games to make
-

This whole book is devoted to demystifying GameMaker: Studio, but you may have a common question and need a quick answer. And if that's the case, you've come to the right spot!

What Types of Games Can I Make with GameMaker: Studio?

The answer to this question is: It's up to you. Within the software, you find tutorials for several types of games — including scrolling shooters, top-down shooters, breakout games, Angry Birds–style games, and platformers. You could make a classic game such as Asteroids or whatever you dream up.

Whatever game you decide to make, whatever Sprites you decide to use, whatever mechanics you want to develop, GameMaker: Studio provides you the means to make your game, your way.

Can I Run GameMaker: Studio on a Mac?

GameMaker: Studio currently does not run on a Mac. I've been told that it's in development and has recently been accelerated, but in the meantime, you can use a Windows emulator on a Mac to install GameMaker: Studio. For more about Windows emulators, go to <http://windows-emulator-software-review.toptenreviews.com>.

How Do I Recover My License Key?

When you first boot up GameMaker: Studio, you'll be asked to enter your license key. But what happens if you lose your license key? Don't worry, you can always retrieve it from the License Recover page: www.yoyogames.com/support. Simply enter your email address, click the Recover button, and YoYo Games will send you your license key.

How Do I Include Advertising in My Game?

At first, while you're just starting, you'll be thinking primarily about making the game itself. However, there will come a time when you're ready to publish your game. If you're independently wealthy and you have no desire to make money with your games, you can skip to the next section. But if you're like most of us, and you have dreams of making a free-to-play game that will be installed a million times over, then you're going to want to include advertising in your game.

When you start designing your game, think about where you're going to place your ads within the game. Also think about which advertising platform you want to use. Luckily, YoYo Games has an entire page full of options for your advertising and analytics needs; just go to <http://help.yoyogames.com/forums> and scroll down to the Ads and Analytics section.

How Do I Activate In-App Purchases?

If you think advertising within a game is pretty hideous and only serves to muck up the game, but you still need to make a living, activating in-app purchases (IAPs) may suit you better. For example, if you make a role-playing game (RPG) full of knights and dragons, and you want to enable the player to purchase horse armor, that would be an IAP. Typically, for the better games, IAPs give the player a chance to buy decorative extras for the game. If you make players pay for items they need to advance in the game, they may not like you very much. Whichever way you want to go, you can find a great online resource for IAPs at <http://help.yoyogames.com/forums/22944718-IAPs>.

What's Involved with Publishing My Game?

A topic I don't go into detail on in this book is how to actually publish your game to all the different platforms available through GameMaker: Studio. Within the software, you can choose from several different Targets for your game. This includes Windows, HTML5, iOS, Android, and more. But what happens if and when you do get a game published? You may be wondering if

YoYo Games gets a cut of the profits. Or you may ask whether YoYo Games can help get your game noticed. The answers are all online, my friend. Check out the answers to these questions and more at <http://help.yoyogames.com/forums/22958768-Publishing>.

When Is a New Feature Coming Out?

New features are added to GameMaker: Studio on a rolling basis. To find the latest new features, go to <http://store.yoyogames.com/downloads/gm-studio/release-notes-studio.html>.



Did you also know that you can download and install an Early Access version of GameMaker: Studio? That's right, now you, too, can try out all the new features before they reach the stable build. Got to www.yoyogames.com/studio/download, and click the Download Studio Early Access link to download the Early Access version.

Where Can I Get More Game-Making Tips and Tricks?

This book is filled with tips, tricks, and tutorials for beginners, but a multitude of additional information is available to help you take your game to the next level as you become a Master Game Maker (I made that title up, but I'm sure someone somewhere has declared himself or herself a Master of Game Making). An excellent source for these tips and tricks can be found at <http://help.yoyogames.com/forums/22669697-Tutorials-Tips-and-Tricks>.

If you're interested in learning how to make better Sprites, an article that you might appreciate can be found at <http://help.yoyogames.com/entries/42578606-Skeletal-Animation-Sprites-Using-Spine-GMS-v1-3>. There, Mark Alexander (the technical editor for this book) discusses using the software Spine, which can be integrated for use with GameMaker.

Index

• Symbols •

(hashtag) symbol, 61
// (backslashes), 153

• A •

abs() function, 275
Actions, 47. *See also* names of specific actions
additive blending, 154, 156
advertising options, 314, 320
AI (artificial intelligence), 54
Alarm Action
 Alarm Events versus, 36
 Emitter example, 146–148
 fireworks effect example, 134–135
 overview, 36
 turning off, 146
Alarm Events
 Alarm triggers, 36
 Emitter example, 146–148
 fireworks effect example, 132, 135–137
 maximum number of Alarms, 37
 overview, 36–37
 sequential Alarms, 131
Alert Games, 173
Align to Grid Action, 52
Alpha Tolerance settings, 23
analog sticks
 creating GUI Scripts, 282–283
 finger detection, 285–287
 joystick drawing, 288–289
 overview, 281
 shooting, code for, 289–292
 stick movement capture, 287–288
 testing, 288, 292
 variables, 285
 virtual keys versus, 281

Analogue Sticks tutorial, 281, 285
anchor point, 21–22, 113
Android platform
 drop-down list, 13
 file directory, 294
 HTML5 testing advantages, 266
 screen redrawing issues, 173
angular damping, 250
Animation End Event, 40
application surface
 letterbox effect, eliminating, 204–207, 211–214
 tutorials, 201
application_get_position()
 function, 211
The_Application_Surface_Overview
 tutorial, 201
The_Application_Surface_Scaling tutorial, 201–203, 210, 214
artificial intelligence (AI), 54
aspect ratio
 adjusting to game window, 206–207
 eliminating letterbox effect, 204–207
 Full Scale
 Keep Aspect Ratio, 203–204
 HTML5 scaling, 216
 maximizing GUI layer, 210–211
 retrieving from game window, 206
 View scaling, effect on Rooms, 207–208
Assets folder, 112
Asynchronous Events, 45

• B •

Backgrounds tab (Room Properties), 29
backslashes (//), 153
blending, 154, 156
blue circle icon, 107
boss battles, 112–113

boss Object

- Collision Event configuration, 121–123
- Create Instance Action, 119
- creating, 114–116
- End Block Action, 121
- health check setup, 120
- Move Fixed Action, 117–118
- Start Block Action, 120–121
- Step Event, creating, 117–121
- Test Chance Action, 119
- Test Variable Action, 117
- Time Line, adding to, 123–124
- Transform Sprite Action, 118–119

Bounce Action, 52

bounce effect, 232

boundary settings

- Keyboard Event adding, 87
- right/left boundaries, 89
- Room transitions, adding, 96–97
- testing, 88
- top/bottom vertical, 88–89

Bounding Box settings, 24

Breakout game, 228–233, 267

bugs

- debugging with Compile window, 14
- fixing with scripts, 234–235
- importance of debugging, 315

bullets, 259–264

burst effects, 157–158

burst emitters

- coding, 160–164
- stream versus, 144, 157–158, 164–165

button creation

- background music toggle, 297–300
- fireworks effect example, 134–135, 159–160
- hovering versus clicking, 138
- sound effects, 301–302

button duplication, 148



Call Parent Event Action, 67

Catch the Clown tutorial, 248

Catherine wheel firework

- creating, 142
- Particle Color Action, 142–143
- Particle Life Action, 143
- Particle Secondary Action, 144
- Particle Speed Action, 143–144

Change Instance Action, 56

Change Sprite Action, 57

Check Collision Action, 65

Check Empty Action, 65

Check Grid Action, 66

Check Mouse Action, 66

Check Next Action, 59

Check Object Action, 65

Check Previous Action, 59

Check Question Action, 65

Check Sound Action, 57

clamp() function, 274

Clear Highscore Action, 70

Code Actions. *See also* Execute Code Action

- Comment Action, 68, 131–132, 153
- Execute Script Action, 68
- overview, 67–68

code generation. *See also* Execute Code Action

- advantages of learning, 9
- automatic, 7–8
- code-editing tool, 9
- custom coding flexibility, 67, 99
- fireworks effect example, 131
- project planning, 11
- testing, 11
- use of color, 11

coding. *See also* GML; particles; Particles Actions

- backslashes, 153
- burst emitters, 160–164
- button duplication, 159–160
- complex particle effects, 169–172
- decals, 182–185
- emitter functions, 161–164
- fireworks effect, 155–159
- fixtures, 250–253
- HTML5 scaling, 215–218
- letterbox effect elimination, 211–214
- particle editors, 172–173
- particle functions, 152–154, 157–159
- particle system, 152–153
- programming flexibility, 67, 99
- Room wrapping, 256
- scaling to fit window, 205–207
- scaling Views, 207–208
- smoke trail, 153–155
- stream emitters, 165–168

- Surface drawing functions, 178–180
 - trigger button, 134–138, 159–160
 - Coding Tutorial, 221
 - Collision Checking feature
 - accuracy versus speed, 22–24
 - Alpha Tolerance settings, 23
 - Mask assignments, 23–24
 - overview, 22
 - Collision Events
 - example, 121–123
 - importance of Masks, 38
 - overview, 38
 - collision groups, 249
 - collisions
 - ball collision example, 231–233
 - bounce effect, 232
 - checking for, 228–229
 - coding with GML, 230–231
 - color settings
 - absolute numeric values, 161
 - Color window, 133–134
 - particles, 153
 - Set Color Action, 75
 - WebGL/browser issues, 173
 - Colour Sprite Action, 57
 - Comment Action
 - backslashes, 153
 - fireworks effect example, 131–132
 - overview, 68
 - Compile window, 13–14, 268
 - Control Actions
 - Code Actions, 67–68
 - Other Actions, 66–67
 - overview, 64
 - Questions Actions, 64–66
 - Variables Actions, 68–69
 - Control tab, 68
 - controller Objects
 - checking when Object reaches end of Room, 84–85
 - controlling view movement, 83–84
 - creating, 83–84
 - placing in Rooms, 87
 - scaling to best fit, 205–207
 - score display, 86–87
 - controller variable, 230–233
 - convex shapes, 248
 - coordinates
 - absolute versus relative, 78
 - gravity vectors, 257
 - raw position calculations, 278–280
 - Surface defaults, 176
 - top of Room, 84
 - up, down, left, right, 229
 - counting, 129
 - Create Effect Action
 - fireworks effect example, 131–137
 - limitations, 138
 - overview, 76
 - Create Events, 35, 244
 - Create Instance Action, 35, 56
 - Create Moving Action, 56
 - Create Particle Action, 139–140
 - Create Particle System Action, 139
 - Create Random Action, 56
- D •
- darkness effect, 190–194
 - Death Count field, 144
 - debugging
 - with Compile window, 14
 - importance of, 315
 - using scripts, 234
 - decals
 - coding, 182–185
 - memory leaks, 183
 - overview, 182
 - performance issues, 182
 - Surfaces, 182–185
 - density
 - bullet example, 260
 - Kinematic Objects, 250
 - overview, 249
 - Depth field settings, 28
 - Destroy at Position Action, 56
 - Destroy Events, 36
 - Destroy Instance Action
 - emitter example, 147–148
 - overview, 56
 - settings, 89–90
 - Destroy Particle System Action, 147
 - device_mouse_*() function, 278
 - Different Room Action, 59

- direction
 - controller variables, 230
 - Jump Actions, 52
 - Keyboard Events, 254–255
 - Move Actions, 49–50
 - Objects Actions, 56
 - physics system, 248
 - rotational value calculations, 257
 - Step Actions, 54
 - wiggle values, 153, 161
- Display Message Action
 - adding messages to Objects, 85
 - overview, 61
- divide by zero error, 281
- drag-and-drop Actions
 - advantages for beginners, 7
 - advantages for experienced developers, 8, 74
 - custom coding flexibility, 99
 - Objects controls, 28
 - overview, 7
- Draw Actions
 - Drawing Actions, 74
 - Other Actions, 75–76
 - overview, 73
 - Settings Actions, 75
- Draw Background Action, 74
- Draw Begin Event, 42
- Draw End Event, 42
- Draw Event, 41
- Draw Events
 - automatic triggering, 41–42
 - Create Events versus, 244
 - creating with Surfaces, 179
 - Draw Begin Event, 42
 - Draw End Event, 42
 - Draw Event, 41
 - Draw GUI Begin Event, 42
 - Draw GUI End Event, 42
 - Draw GUI Event, 42, 210–211, 212–213
 - options, 41–42
 - overview, 41
 - PostDraw Event, 42–43
 - PreDraw Event, 42–43
 - Resize Event, 43
 - Draw GUI Begin Event, 42
 - Draw GUI End Event, 42

- Draw GUI Event
 - code example, 212–213
 - maximizing GUI layer, 210–211
 - overview, 42
- Draw Health Action, 72, 86–87
- Draw Life Images Action, 71, 86–87
- Draw Lives Action, 71
- Draw Score Action, 70
- Draw Self Action, 74
- Draw Sprite Action, 74
- Draw Text Action, 74
- Draw Variable Action, 69
- Drawing Actions, 74
- duplicating Rooms, 104–105



- Else Action, 67
- emitters
 - burst versus stream effects, 144, 157–158, 164–165
 - creating, 144–148
 - destroying, 160, 172
 - functions, 161–164
 - memory leaks, 160, 164
 - overview, 144
 - simulating, 157–158
 - stopping particle streams, 167–168
- emulators
 - mobile device, 265
 - Windows (for Mac), 319
- End Block Action, 66–67
- End Game Action, 62
- End of Path Event, 41
- End Path Action, 53
- enemy planes
 - bullets, 94
 - Comment Action, 93
 - Create Instance Action, 94
 - Destroy Instance Action, 94
 - Exit Event Action, 92–93
 - Object creation, 91–92
 - patterns of planes, 95–96
 - Speed Vertical Action, 93
 - Test Chance Action, 94
 - Test Variable Actions, 92–94
 - variable values, 93

Events

- adding to Objects, 33–34, 232–233
- Alarm Events, 36–37
- assigning Actions to, 34
- Asynchronous Events, 45
- Collision Events, 38, 121–123
- Create Events, 35, 244
- deleting, 178
- Destroy Events, 36
- Draw Events, 41–43
- Key Events, 44
- Mouse Events, 39, 267, 269, 272–274, 276–281
- Other Events, 39–41
- Step Events, 37–38, 41–42, 84–85, 109
- timing with Timelines, 36
- triggering with Comment Actions, 68

Execute Code Action

- adding code to Objects, 227
- adding Instances, 150
- button creation, 128–131, 149, 159–160
- Catherine wheel example, 148–149
- code color, 131
- custom coding flexibility, 47, 99
- fireworks effect example, 131, 155–159
- overview, 67
- puff of smoke example, 139
- smoke trail, 153–155

Execute Script Action, 68

Exit Event Action, 67

external files

- base file safety, 296
- game bundle versus local storage, 294, 307
- INI files, 295–296
- sandbox, 294
- save game options, 297–300
- saving and loading, 293
- tutorial, 293
- UTF-8 file warning, 296
- working_directory prefix, 294

Extra Actions, 72–73

• F •

F1 shortcut, 222

favicon icon choices, 10

File Transfer Protocol (FTP) uploads, 10, 266

FileZilla website, 10

fireworks effect

- Alarm Action, 134–135
- Alarm Events, 132, 135–137
- Catherine wheel, 142–144
- Comment Action, 131–132
- Create Effect Action, 131–137
- Execute Code Action, 131, 155–159
- Set Gravity Action, 131–132
- smoke trail. *See* coding
- trigger button, 134–135, 159–160

fixtures

- aligning with Objects, 260
- convex versus concave shapes, 248
- creating, 250–253
- masks versus, 248
- overview, 248–249
- shape design, 253

flashlight effect, 188–190

floating point values, 85

font settings, 75

force. *See also* gravity

- direction/image angle calculations, 257
- impulse versus, 264
- local coordinate system, 254
- torque, 257

Up Keyboard Events, 254

freemium games, 314

friction (physics), 250

FTP (File Transfer Protocol) uploads,
10, 266

full-screen toggle, 75

functions. *See* emitters; GML;
particles

• G •

Game Actions, 62–63

End Game Action, 62

Load Game Action, 63

Restart Game Action, 62

Save Game Action, 63

game bundle files, 294

game creation

- advice for beginners, 8
- future vision, 8
- project planning, 11
- testing, 11

Game Developers Conference (GDC), 7

game development

- beta versions, 317
- combining features, 316
- independent productions, 317
- open-sourcing, 316
- Windows emulators for Mac, 319

Game Start/End Events, 40

GameMaker Language (GML). *See* GML

GameMaker: Studio. *See also* names of specific features; tutorials

- code-editing tool, 9
- color, 11
- F1 shortcut, 222
- future plans, 8, 11
- installation requirements, 9
- platform file naming, 10
- target audience, 7
- website, 8
- windows, 11, 14

GDC (Game Developers Conference), 7

genres, 314–315

Global Game Settings

- aspect ratio, 203
- landscape/portrait settings, 208
- maximizing GUI layer, 211
- opening from `The_Application_Surface_` Scaling tutorial, 202

GML (GameMaker Language)

- adding code to Objects, 227
- bugs, fixing with scripts, 234–235
- collisions and Step Events, 230–231
- color-coded text, 225
- controller variables, 230–233
- draw function placement, 244
- F1 shortcut, 222
- file saving and loading, 293
- introduction screens, 240–243
- keyboard controls, 228–230
- overview, 221
- preference settings, 225
- score displays, 235–239
- sub-images, overview, 222
- sub-images, random, 222–225
- sub-images, sorted by index, 226–227
- transition effects, 243–246

Go to Next Room Action. *See* Next Room Action

Graphical User Interface layer. *See* GUI layer

gravity

- calculating, 257
- fireworks effect example, 131–132
- particle settings, 154
- Set Gravity Action, 50

GUI (Graphical User Interface) layer

- analog stick Scripts, 282–283
- `application_get_position()` function, 211
- creating with HUD, 214–215
- letterbox effect elimination, 211–213
- maximizing, 210–211
- overview, 42
- relative positions, defining, 211
- virtual keys, 267

`gui_x/y` array, 285

guided bullet Object, 90–91, 94–95

• H •

hashtag (#) symbol, 61

heads-up display (HUD), 42, 214–215

Health Actions

- Draw Health Action, 72, 86–87
- overview, 71
- Score Caption Action, 72
- Set Health Action, 72
- Test Health Action, 72

HTML5

- aspect ratio, 204
- display definition, 215
- file directory, 294
- game size considerations, 266
- HTML5_Scaling tutorial, 215
- local storage limit, 294
- scaling, 215–218
- sizing browser window, 218
- testing, 266
- Windows versus, 173, 216

HUD (heads-up display), 42, 214–215

• I •

IAPs (in-app purchases), 320

icon menu

- creating Objects, 26–28
- creating Sprites, 19–20
- creating/defining Rooms, 29–30
- overview, 13

Image Editor, 20
 impulse versus force, 264
 in-app purchases (IAPs), 320
 Info Actions
 Display Message Action, 61, 85
 URL Open Action, 61
 INI files
 base file safety, 296
 creating, 295–296
 game bundle inclusion, 295
 `ini_close()` function, 299
 `ini_open()` function, 299
 overview, 295
 real values versus strings, 297
 reset mechanisms, 295
 save game options, 302–305
 `Save.ini` file, 295, 299
 TXT file conversion, 296
 `ini_close()` function, 299
 `ini_open()` function, 299
 Instance Collision Mask, 274
 Instances
 adding to Rooms, 150
 bugs, fixing with scripts, 234–235
 Check Actions, 65
 Check Grid Action, 66
 clearing Rooms, 104–105
 creating Rooms, 99
 definition, 37, 48
 destroying, 89–90
 Object placement in Rooms, 30, 80
 overview, 11
 Questions Actions, 65
 Room and View relationship,
 77, 79
 Step Event checking, 37
 Test Instance Count, 65
 Time Line Objects, 106–108
 Intersect Boundary Event, 39
 introduction screens
 creating Control Object, 242–243
 creating Sprites, 241
 overview, 240
 iOS platform
 End Game Action, 62
 file directory, 294
 HTML5 testing advantages, 266
 particle speed issues, 173
 screen redrawing issues, 173

• J •

Jump Actions
 Align to Grid Action, 52
 Bounce Action, 52
 Jump to Position Action, 51
 Jump to Random Action, 52
 Jump to Start Action, 51
 Move to Contact Action, 52
 overview, 51
 Wrap Screen Action, 52
 Jump to Position Action, 51
 Jump to Random Action, 52
 Jump to Start Action, 51

• K •

Keep Aspect Ratio setting, 204
 keyboard controls
 adding code to Objects, 227
 remapping keys, 305–307
 resetting default key, 307–309
 writing code for, 228–230
 Keyboard Events
 boundary settings, 87–89
 overview, 44
 physics coding, 254–255
 kinematic setting, 250

• L •

layers. *See* Surfaces
 letterbox effect
 `application_get_position()`
 function, 211–213
 Draw GUI Events, 212–213
 eliminating, 204–207
 mobile devices, 211–214
 overriding for GUI layers, 210
 license key recovery, 320
 lighting effects
 basic, 186–188
 darkness, 190–194
 flashlight, 188–190
 linear damping, 250
 Linux platform
 aspect ratio, 204
 resolution and scaling, 201

Lives Actions

- Draw Life Images Action, 71, 86–87
- Draw Lives Action, 71
- Object updating, 238–239
- overview, 71
- Set Lives Action, 71
- Test Lives Action, 71

Load Game Action, 63**Loading and Saving tutorial, 293****local coordinate system (physics), 254****local folder (%localappdata%), 75****local force, 254–257****Mac platform**

- aspect ratio, 204
- file directory, 294
- resolution and scaling, 201
- Windows emulators, 319

Main 1 Actions

- Objects Actions, 56
- overview, 55
- Room Actions, 58–59
- Sound Actions, 57–58
- Sprite Actions, 57

Main 2 Actions

- Game Actions, 62–63
- Info Actions, 61
- overview, 59
- Resources Actions, 63–64
- Timing Actions, 60–61

manual, in-software, 222**Mask assignments**

- Alpha Tolerance settings, 23
- Bounding Box settings, 24
- multiple sub-images, 23
- Objects, 28
- relationship to Collision Events, 38
- shape choices, 24
- Sprite Properties window, 23

masks

- automatic versus manual settings, 24
- Check Empty Action, 65
- Collision Events, 38
- fixtures versus, 24, 248
- modifying, 21–24
- Object assignments, 28

performance issues, 23**precision shape settings, 24****max () function, 281****memory leaks, 179****menu bars, 13–14****mobile devices****emulators, 265****Game End Event issues, 40****HTML5 considerations, 215–218, 266****letterbox elimination, 211–214****limitations of, 316****online game access, 266****resolution and scaling, 208–210, 257****swipes, directional, 276–278****swipes, single-direction, 273–276****touch controls, 267****virtual keys, creating, 267–269****virtual keys, customizing, 269–272****Moments. *See also* Step Events****adding multiple, 102–104****adding single, 100–101****creating Actions for, 110–111****definition, 100****Time Line editing, 108–109****monetizing games, 314****mouse****assigning Actions to, 39****Check Mouse Action, 66****movement control variables,
283–285****raw position calculations, 278–280,
282–283****Set Mouse Cursor Action, 72–73****Mouse Events****directional swipes, 276–278****global versus local, 274****Instance Collision Mask detection, 274****overview, 39****pinching/zooming, 278–281****swipe controls, 272–274****tapping, 267****testing, 269****Move Actions****Jump Actions, 51–52****Move Directional Actions, 48–50****overview, 48****Path Actions, 52–54****Step Actions, 54–55**

Move Directional Actions

- Move Fixed Action, 49
- Move Free Action, 49
- Move Towards Action, 49
- overview, 48–49
- Reverse Horizontal Action, 50
- Reverse Vertical Action, 50
- Set Friction Action, 50
- Set Gravity Action, 50, 131–132
- Speed Horizontal Action, 50
- Speed Vertical Action, 50
- Move Fixed Action, 49
- Move Free Action, 49
- Move tab, 48–51
- Move to Contact Action, 52
- Move Towards Action, 49



- New Project window, 11
- Next Room Action, 59, 96–97
- Nintendo 3DS, 8
- No More Health Event, 40
- No More Lives Event, 40
- Nocturne Games, 173
- NumLock toggle, 44



- Objects. *See also* boss Object
 - adding code to, 227
 - adding Events to, 28, 33–34, 232–233
 - adding to Time Lines, 108–109
 - adding vertical speed to, 78–79
 - applying fixtures, 248
 - assigning Actions to, 47–48
 - boundary settings, 87–88
 - checking when Object reaches end
 - of Room, 85
 - collision control, 27
 - collision reactions, 230–233
 - controlling with Draw Events, 41
 - creating for Time Lines, 105–106
 - Depth field settings, 28
 - destroying Instances, 89–90
 - drag-and-drop Actions, 28

- drop-down list, 26–27
- guided bullets, 90–91
- linking with variables, 230–232
- loading/creating, 27
- Mask assignments, 28
- overview, 11, 26
- parent/child assignments, 28
- physics settings, 27
- Room transitions, 96–97
- score display, 86–87
- speed effects in Views, 78–79
- updating, 238–239

Objects Actions

- Change Instance Action, 56
- Create Instance Action, 35, 56
- Create Moving Action, 56
- Create Random Action, 56
- Destroy at Position Action, 56
- Destroy Instance Action, 56, 89–90, 147–148

Origin

- code example, 88–89
- overview, 21–22
- X axis choices, 21–22, 113

Other Actions (Control tab)

- Call Parent Event Action, 67
- Else Action, 67
- End Block Action, 66–67
- Exit Event Action, 67
- Repeat Action, 67
- Start Block Action, 66–67, 120–121

Other Actions (Draw tab)

- Create Effect Action, 76, 131–138
- Take Snapshot Action, 75

Other Events menu

- Animation End Event, 40
- End of Path Event, 41
- Game End Event, 40
- Game Start Event, 40
- Intersect Boundary Event, 39
- No More Health Event, 40
- No More Lives Event, 40
- Outside Event, 39
- Room End Event, 40
- Room Start Event, 40
- User Defined Event, 41
- Outside Room Event, 39



Particle Color Action, 140–143
 Particle Editor 2.5, 173
 particle editors, 172–173
 Particle Gravity Action, 141
 Particle Life Action, 141, 143
 Particle Secondary Action, 144
 Particle Speed Action, 143–144
 particles. *See also* coding; emitters
 alpha function choices, 154, 156
 automatic propagation, 170
 burst versus stream effects, 144, 157–158, 164–165
 button creation, 128–131
 color function choices, 153, 156
 destroying, 160, 167, 172
 direction settings, 154
 emitter simulation, 157–158
 functions, 152–154, 157–159
 gravity settings, 154
 Instance movement, 57, 154, 157
 lifetime settings, 154
 memory leaks, 160, 164
 overview, 127
 particle editors, 172–173
 platform differences, 173
 rotational angle settings, 154
 secondary, 144
 shape-size settings, 153
 speed settings, 154
 sprites, animated, 171
 tutorial, 151
 wiggle value settings, 153–154
 Particles Actions. *See also* coding
 Create Particle Action, 139–140
 Create Particle System Action, 139
 custom coding versus drag-and-drop, 151
 Destroy Particle System Action, 147
 emitters, 144
 overview, 138
 Particle Color Action, 140–143
 Particle Gravity Action, 141
 Particle Life Action, 141, 143
 Particle Secondary Action, 144
 Particle Speed Action, 143–144

Path Actions

 End Path Action, 53
 Move tab controls, 48–51
 overview, 52
 Path Position Action, 53
 Path Speed Action, 53
 Set Path Action, 53
 Path Position Action, 53
 Path Speed Action, 53
 Pause Time Line Action, 61
 performance issues
 accuracy versus speed, 22–24
 Android/iOS screen redrawing, 173
 automatic particle propagation, 170
 collision groups, 249
 decal overuse, 182
 destroying unnecessary particles/
 emitters, 167, 172
 HTML5 versus Windows, 173
 mobile device limitations, 316
 Particle Secondary Action, 144
 particle/emitter memory leaks,
 160, 164
 Separate Collision Masks, 23
 speed versus accuracy, 22–24
 Surface drawing size, 176
 Surface memory leaks, 179, 183
 persistency settings, 27
 physics
 bullets, coding to fire, 262–264
 bullets, creating, 259–262
 density, 249
 fixtures, 248–253
 gravity settings, 257
 impulse versus force, 264
 Left/Right Keyboard Events, 255
 local coordinate system, 254
 mass calculations, 250
 overview, 248
 pixel/meter translation, 257–258
 properties listing, 249–250
 Room setup, 257–258
 rotation control, 255
 rotation direction, 257
 Sprites Properties window, 27
 torque control, 257

- traditional movement systems versus, 248, 257
- Up Keyboard Event, 254
- Physics tab (Room Properties), 29
- Physics tutorial, 247
- `physics_apply_torque` variable, 257
- pinching/zooming
 - divide by zero error, 281
 - laptop trackpads, 281
 - overview, 278
- raw position calculations, 278–280
- pixel/meter translation, 257–258
- platform choices
 - drop-down list, 9–10, 13
 - favicon icon choices, 10
 - file naming, 10
 - game publishing, 10
 - listing, 294
 - PS4 and Vista, 10
- platform games, 314
- Play Sound Action, 57
- PostDraw Event, 42–43
- precise bouncing, 232
- PreDraw Event, 42–43
- Previous Room Action, 59
- PS4 future plans, 8
- puff of smoke effect, 139–141



Questions Actions

- Check Collision Action, 65
- Check Empty Action, 65
- Check Grid Action, 66
- Check Mouse Action, 66
- Check Object Action, 65
- Check Question Action, 65
- overview, 64–65
- Test Chance Action, 65
- Test Expression Action, 66
- Test Instance Count Action, 65



- random generators, 100, 112
- random movement, 49, 52, 163

- random Object creation, 56
- random particle generation, 127
- random sub-images, 17
- raw coordinates (mobile devices), 278
- remapping keys, 305–309
- Repeat Action, 67
- Replace Background Action, 64
- Replace Sound Action, 64
- Replace Sprite Action, 64
- reset mechanisms, 295
- Resize Event, 43
- resolution, 201, 283
- Resource tree, 13, 59
- Resources Actions
 - bugs, 64
 - overview, 63
 - Replace Background Action, 64
 - Replace Sound Action, 64
 - Replace Sprite Action, 64
- Restart Game Action, 62
- Restart Room Action, 59
- restitution, 249
- Reverse Horizontal Action, 50
- Reverse Vertical Action, 50
- roguelike games, 314–315
- role-playing games (RPGs), 320
- Room Actions
 - Check Next Action, 59
 - Check Previous Action, 59
 - Different Room Action, 59
 - Next Room Action, 59, 96–97
 - overview, 58
 - Previous Room Action, 59
 - Restart Room Action, 59
- Room End Event, 40
- Room Start Event, 40
- Rooms
 - absolute versus relative coordinates, 78
 - background colors/images, 81
 - creating/defining, 29–30
 - dimension settings, 81
 - duplicating, 104–105
 - multiple for each game, 80
 - naming, 81
 - Object placement, 30
 - overview, 29

Rooms (*continued*)

- physics setup, 257–258
- planning for Views, 79–80
- relationship to Views, 77, 79
- Resource tree menus, 80
- scaling, 207–208, 215–218
- speed settings, 29
- transitions, 96–97
- View movement within, 78–79
- Views tab, 82–83
- wrapping, 256
- rotation control
 - angular damping, 250
 - direction calculations, 257
 - Left/Right Keyboard Events, 255, 257
 - torque application, 257
- RPGs (role-playing games), 320



- sandbox, 294
- Save Game Action, 63
- save game options
 - background music, 297–300
 - key remapping, 305–307
 - matching values, 305
 - sound effects, 301–302
 - string values, 302–305
- Save .ini file
 - closing files, 299
 - location, 295
- scaling
 - to fit window, 201, 203–207
 - for mobile devices, 208–210
 - Views, 207–208
- Score Actions
 - Health Actions, 71–72
 - Lives Actions, 71
 - Score Actions, 69–70
- Score Caption Action, 72
- score display
 - creating with controller Objects, 86–87
 - creating with GML, 235–239
 - global variables, 97
 - local/online high score switching, 273–276

- Object creation, 236–237
 - updating/displaying high score, 237–238
- Score Health Action, 72
- ScoreBox image, 235
- screen buffer, 42
- screen capture, 75
- screens
 - relative Object movements, 79
 - Room and View relationship, 77, 79
- Scripts
 - analog sticks, 282–283
 - definition, 234
 - Execute Script Action, 68
 - finding/using, 235
 - fixing bugs, 234–235
 - HTML5 scaling, 215–216
 - mobile device scaling, 208–209
 - preferences management, 225
 - reusing code blocks, 68
 - Room wrapping, 256
 - scaling Views, 207–208
 - window sizing, 205–206
- Scrolling Shooter tutorial, 77–78
- secondary particles, 144
- sensor settings, 250
- Set Alarm Action, 60
- Set Color Action, 75
- Set Font Action, 75
- Set Friction Action, 50
- Set Full Screen Action, 75
- Set Gravity Action, 50, 131–132
- Set Health Action, 72
- Set Lives Action, 71
- Set Mouse Cursor Action, 72–73
- Set Path Action, 53
- Set Score Action, 69
- Set Time Line Action, 60
- Set Variable Action, 68
- Settings Actions, 75
- shadow creation
 - basic shadows, 177
 - complex shadows, 178–182
 - Surface advantages, 182
- shock wave effect, 197–200
- sign() function, 275
- smoke trail. *See* coding

- Sony Developers License, 8, 10
- Sound Actions, 57–58
- sound effects toggle, 301–302
- Sound resources
 - main menu icon, 15
 - overview, 14
 - sound file editing, 16
 - sound file loading, 15
- Sparks particle editor, 173
- special effects
 - bounce, 232
 - burst, 157–158
 - darkness, 190–194
 - fireworks, 131–137, 142–144, 155–160
 - flashlight effect, 188–190
 - lighting, 186–194
 - puff of smoke, 139–141
 - shock wave, 197–200
 - stream, 157–158
 - stretching, 204
- Speed Horizontal Action, 50
- speed settings
 - negative versus positive, 78–79
 - overview, 29
 - speed versus accuracy, 22–24
- Speed Vertical Action, 50
- Speed Vertical window, 78
- Spine software, 321
- Sprite Actions, 57
- Sprites. *See also* sub-images
 - Animation End Event, 40
 - blue circle icon, 107
 - Collision Checking feature, 22–24
 - configuring, 21
 - Create a New Sprite dialog box, 19
 - custom animated particles, 171
 - Draw Life Images Action, 86–87
 - fixture shape requirements, 248
 - image browsing, 18
 - Image Editor, 20
 - loading, 17–18
 - Mask assignments, 23–24
 - Object placement in Rooms, 30
 - Origin, 21–22
 - overview, 17
 - persistence, 27
 - physics, 27
 - ScoreBox image, 235
 - Sprite Editor, 20
 - Sprite Properties window, 18–19, 22–27
 - Texture Groups, 24–26
 - trigger button, 159–160
 - virtual key customization, 269–272
 - visibility, 27
 - X and Y axis choices, 21, 113–114
- `st_pressed` variable, 285
- `st_radius` variable, 285
- start awake option, 250, 253, 261
- Start Block Action, 66–67, 120–121
- Start Time Line Action, 61
- Step Actions, 54–55
- Step Avoid Action, 54–55
- Step Events. *See also* Moments; Time Lines
 - checking when Object reaches end of Room, 84–85
 - Display Message Action, 85
 - menu options, 38
 - Step number editing, 109
 - Test Variable Action, 84–85
 - testing with, 84–85
 - timing action with, 37
 - Visible toggle effects, 41–42
- Step Towards Action, 54–55
- `stick_x/y` array, 285
- Stop Sound Action, 57
- Stop Time Line Action, 61
- stream effects, 157–158
- stream emitters
 - burst versus, 144, 157–158, 164–165
 - coding, 165–168
- stretching effect, 204
- sub-images
 - Animation End Event, 40
 - counting reminder, 129
 - GML coding, 222–227
 - numbering/displaying, 17
 - precision shape settings, 24
 - separate Masks assignments, 23
 - sorting by index, 226–227
 - speed settings, 225

Surfaces

- application surface, 176
- coordinate defaults, 176
- decals, 182–185
- Draw Events, 179
- drawing functions, 176, 178–180
- drawing memory, 176
- lighting effects, 186–194
- memory leaks, 179, 183
- overview, 175
- removing effects, 178–179
- shadow creation, 176–182
- shock wave effect, 197–200
- texture variables, 198
- Views, 194–200
- VRAM clearing, 181
- swipe controls
 - `abs()` function, 275
 - `clamp()` function, 274
 - directional, 276–278
 - horizontal versus vertical, 276
 - pinching/zooming, 278–281
 - `sign()` function, 275
 - single-direction, 273–276
 - testing, 275, 278



Take Snapshot Action, 75

target platforms

- aspect ratio, 204
- drop-down list, 9–10, 13
- HTML5, 217
- testing options, 266

Test Chance Action, 65

Test Expression Action, 66

Test Health Action, 72

Test Instance Count, 65

Test Lives Action, 71

Test Score Action, 69

Test Variable Action, 68, 85, 87–89

testing

- analog sticks, 288
- background music toggle, 300
- beta versions, 317
- Else Action, 67

error warning, 91

expression evaluation, 66

HTML5 advantages, 266

importance of, 11, 315

Instances, 65

open-sourcing, 316

pinching/zooming, 281

save game options, 305

sound effects toggle button, 302

sounds, 57

Step Events, 84–85

swipe controls, 275, 278

virtual keys, 269, 272

Texture Groups

assigning Sprites, 25–26

creating, 24

naming, 24–25

textures, 176, 198

Tiles tab (Room Properties), 29

Time Line Position Action, 60

Time Line Speed Action, 61

Time Lines. *See also* Step Events

access options, 100

advantages of, 100

Create Instance Action, 101–102

game behavior control, 100

Moments, adding, 123–124

Moments, creating, 100–104

Moments, editing, 108–109

multiple Rooms, 104

Objects, creating, 105–106

overview, 36, 99

Rooms, creating, 106–108

Rooms, duplicating, 104

Views versus, 99, 111

X and Y coordinates, 99

timing, 29

Timing Actions

Pause Time Line Action, 61

Set Alarm Action, 60

Set Time Line Action, 60

Start Time Line Action, 61

Stop Time Line Action, 61

Time Line Position Action, 60

Time Line Speed Action, 61

torque, 257

trails, 43
 Transform Sprite Action, 57
 transitions
 game endings, 243–246
 Next Room Action, 96–97
 Time Lines, 104, 111–112
 trigger button
 fireworks effect example, 134–135, 159–160
 hovering versus clicking, 138
 tutorials
 Analogue Sticks, 281, 285
 The_Application_Surface_Overview, 201
 The_Application_Surface_Scaling, 201–203, 210, 214
 Breakout, 228–233
 Catch the Clown, 248
 Coding, 221
 game inclusion of, 315
 HTML5_Scaling, 215
 listing of, 176
 Loading and Saving, 293
 New Project window, 11
 Particles, 127, 151
 Physics, 247
 Scrolling Shooter, 77–78
 Surfaces_Part 1, 176–178
 Surfaces_Part 2, 194, 197
 tutorial window overview, 14
 website, 3



URL Open Action, 61
 usability concerns, 315
 User Defined Event, 41
 UTF-8 file warning, 296



variables. *See also* X and Y coordinates
 array numbering, 86
 controller Object settings, 83–84
 error warning, 91
 floating point values, 85
 Objects, linking with, 230–232

Variables Actions
 Draw Variable Action, 69
 Set Variable Action, 68
 Test Variable Action, 68, 85, 87–89
 vdir variable, 285
 vectors, 257
 Vertical Speed Action, 78–79
 Views
 alternating Surfaces, 194
 configuring in Rooms, 82
 controller Objects, 83–84
 movement within Rooms, 78–79, 83–84
 overview, 77
 planning, 79–80
 relationship to Rooms, 77, 79
 scaling, 207–208
 speed effects, 78–79
 Surfaces as, 194–200
 Time Lines versus, 99, 111
 Views tab (Room Properties), 29, 82–83
 virtual joystick. *See* analog sticks
 virtual keys
 analog sticks versus, 281
 creating, 267–269
 customizing with Sprites, 269–272
 overview, 267
 testing, 269, 272
 VRAM clearing, 181



WebGL / browser issues, 173
 websites
 advertising options, 320
 FileZilla, 10
 game publishing questions, 320–321
 game uploading/accessing, 266
 GameMaker: Studio, 8
 IAPs, 320
 new features, 321
 Particle Editor 2.5, 173
 Sparks particle editor, 173
 Sprites development, 321
 tips and tricks, 321
 tutorials, 3
 Windows emulators, 319
 YoYo Games, 3

- wiggle value, 153, 161
- Window Colour setting, 43
- window resizing (scaling), 205–207
- Windows 7 platform
 - aspect ratio, 204
 - file directory, 294
 - Global Game settings, 202–203
 - HTML5 versus, 173
 - resolution and scaling, 201
- Windows 8 platform
 - file directory, 294
 - HTML5 scaling issues, 201, 216
 - local storage limit, 294
 - Resize Event, 43
- Windows Phone platform
 - End Game Action, 62
 - file directory, 294
 - HTML5 testing advantages, 266
 - particle speed issues, 173
- working_directory prefix, 294
- Wrap Screen Action, 52
- wrapping Rooms
 - overview, 52
 - Script creation, 256



- X and Y coordinates
 - boundary settings, 88–89
 - Create Instance Action, 94, 101
 - Draw Events, 86–87
 - Instance placement in Moments, 110–111
 - Jump Actions, 51
 - Move Towards Action, 91
 - overview, 99
 - Relative settings, 133
 - Room dimensions, 82
 - triggering Actions with, 92, 97
- X axis choices, 21–22, 113
- xforce_local value, 254



- Y axis choices, 114
- yforce_local value, 254
- YoYo Games website, 3



- zooming, 278–281

About the Author

Mike Rohde has been writing for over 20 years and playing videogames even longer. This is his second *For Dummies* book — *Adobe Edge Animate CC For Dummies* was published in July 2013. He lives in the northern Bay Area of California with his wife, two children, and a black lab named Delilah.

Dedication

To my kids, Max and Julia — may you shine forever bright.

Author's Acknowledgments

I dedicated this book to my kids, and for good reason. When they first learned I was writing a book to teach people how to make videogames, their mouths dropped open. Their excitement, energy, and enthusiasm motivated me to write the best possible book I could. Plus, I might have started a game reviewing career for my son. When my daughter asked him what he was playing while he was testing one of my first games, he answered, “It’s called Catch the Clown. It isn’t very much fun.” With that statement, my 7-year old son upped the ante on what he expected from the book. He did appreciate the special effects I made with particles though.

This book would not have been possible without my wife, without whom I probably would have grown hungry, cold, and dirty because I was too busy writing about, making, and playing games. Thank you for helping make life easier.

I want to thank Connie Santisteban from John Wiley & Sons, who helped make this book a reality. Her energy behind this book was amazing right from the start, straight through to the finish.

I would be remiss if I did not acknowledge the hard work Elizabeth Kuball put into this book. She is responsible for keeping this a traditional *For Dummies* book, and not my version of the Wild, Wild West. Thank you for keeping this book on track.

Mark Alexander was the technical editor for this book and is the man responsible for the in-software manual of GameMaker: Studio. His careful eye saved a few mistakes and his expert advice proved valuable for many of the tips in this book. He was also the guiding force behind getting the analog sticks procedures to work in Chapter 13.

To the YoYo Games staff — in particular, the CEO, Sandy Duncan — I want to thank you for responding to my tweet asking if you’d like a *For Dummies* book for GameMaker: Studio and for making this book possible. To the guys in the GDC booth – Malcolm, Stuart, and Russell – thank you for taking your time to speak with me. Much of our conversations made it into the book.

Publisher's Acknowledgments

Acquisitions Editor: Connie Santisteban

Project Editor: Elizabeth Kuball

Copy Editor: Elizabeth Kuball

Technical Editor: Mark Alexander

Editorial Assistants: Claire Johnson,
Annie Sullivan

Sr. Editorial Assistant: Cherie Case

Project Coordinator: Patrick Redmond

Cover Image: © iStock.com/exdez

Apple & Mac

iPad For Dummies,

6th Edition

978-1-118-72306-7

iPhone For Dummies,

7th Edition

978-1-118-69083-3

Macs All-in-One

For Dummies, 4th Edition

978-1-118-82210-4

OS X Mavericks

For Dummies

978-1-118-69188-5

Blogging & Social Media

Facebook For Dummies,

5th Edition

978-1-118-63312-0

Social Media Engagement

For Dummies

978-1-118-53019-1

WordPress For Dummies,

6th Edition

978-1-118-79161-5

Business

Stock Investing

For Dummies, 4th Edition

978-1-118-37678-2

Investing For Dummies,

6th Edition

978-0-470-90545-6

Personal Finance

For Dummies, 7th Edition

978-1-118-11785-9

QuickBooks 2014

For Dummies

978-1-118-72005-9

Small Business Marketing

Kit For Dummies,

3rd Edition

978-1-118-31183-7

Careers

Job Interviews

For Dummies, 4th Edition

978-1-118-11290-8

Job Searching with Social

Media For Dummies,

2nd Edition

978-1-118-67856-5

Personal Branding

For Dummies

978-1-118-11792-7

Resumes For Dummies,

6th Edition

978-0-470-87361-8

Starting an Etsy Business

For Dummies, 2nd Edition

978-1-118-59024-9

Diet & Nutrition

Belly Fat Diet For Dummies

978-1-118-34585-6

Mediterranean Diet

For Dummies

978-1-118-71525-3

Nutrition For Dummies,

5th Edition

978-0-470-93231-5

Digital Photography

Digital SLR Photography

All-in-One For Dummies,

2nd Edition

978-1-118-59082-9

Digital SLR Video &

Filmmaking For Dummies

978-1-118-36598-4

Photoshop Elements 12

For Dummies

978-1-118-72714-0

Gardening

Herb Gardening

For Dummies, 2nd Edition

978-0-470-61778-6

Gardening with Free-Range

Chickens For Dummies

978-1-118-54754-0

Health

Boosting Your Immunity

For Dummies

978-1-118-40200-9

Diabetes For Dummies,

4th Edition

978-1-118-29447-5

Living Paleo For Dummies

978-1-118-29405-5

Big Data

Big Data For Dummies

978-1-118-50422-2

Data Visualization

For Dummies

978-1-118-50289-1

Hadoop For Dummies

978-1-118-60755-8

Language & Foreign Language

500 Spanish Verbs

For Dummies

978-1-118-02382-2

English Grammar

For Dummies, 2nd Edition

978-0-470-54664-2

French All-in-One

For Dummies

978-1-118-22815-9

German Essentials

For Dummies

978-1-118-18422-6

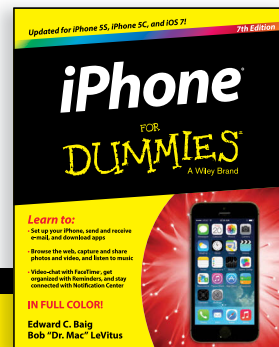
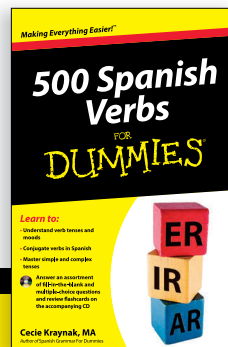
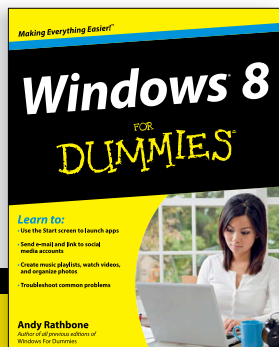
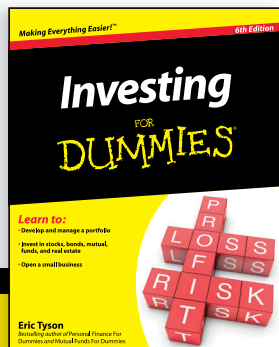
Italian For Dummies,

2nd Edition

978-1-118-00465-4



Available in print and e-book formats.



Available wherever books are sold. For more information or to order direct visit www.dummies.com

www.it-ebooks.info

Math & Science

Algebra I For Dummies,
2nd Edition
978-0-470-55964-2

Anatomy and Physiology
For Dummies, 2nd Edition
978-0-470-92326-9

Astronomy For Dummies,
3rd Edition
978-1-118-37697-3

Biology For Dummies,
2nd Edition
978-0-470-59875-7

Chemistry For Dummies,
2nd Edition
978-1-118-00730-3

1001 Algebra II Practice
Problems For Dummies
978-1-118-44662-1

Microsoft Office

Excel 2013 For Dummies
978-1-118-51012-4

Office 2013 All-in-One
For Dummies
978-1-118-51636-2

PowerPoint 2013
For Dummies
978-1-118-50253-2

Word 2013 For Dummies
978-1-118-49123-2

Music

Blues Harmonica
For Dummies
978-1-118-25269-7

Guitar For Dummies,
3rd Edition
978-1-118-11554-1

iPod & iTunes
For Dummies, 10th Edition
978-1-118-50864-0

Programming

Beginning Programming
with C For Dummies
978-1-118-73763-7

Excel VBA Programming
For Dummies, 3rd Edition
978-1-118-49037-2

Java For Dummies,
6th Edition
978-1-118-40780-6

Religion & Inspiration

The Bible For Dummies
978-0-7645-5296-0

Buddhism For Dummies,
2nd Edition
978-1-118-02379-2

Catholicism For Dummies,
2nd Edition
978-1-118-07778-8

Self-Help & Relationships

Beating Sugar Addiction
For Dummies
978-1-118-54645-1

Meditation For Dummies,
3rd Edition
978-1-118-29144-3

Seniors

Laptops For Seniors
For Dummies, 3rd Edition
978-1-118-71105-7

Computers For Seniors
For Dummies, 3rd Edition
978-1-118-11553-4

iPad For Seniors
For Dummies, 6th Edition
978-1-118-72826-0

Social Security
For Dummies
978-1-118-20573-0

Smartphones & Tablets

Android Phones
For Dummies, 2nd Edition
978-1-118-72030-1

Nexus Tablets
For Dummies
978-1-118-77243-0

Samsung Galaxy S 4
For Dummies
978-1-118-64222-1

Samsung Galaxy Tabs
For Dummies
978-1-118-77294-2

Test Prep

ACT For Dummies,
5th Edition
978-1-118-01259-8

ASVAB For Dummies,
3rd Edition
978-0-470-63760-9

GRE For Dummies,
7th Edition
978-0-470-88921-3

Officer Candidate Tests
For Dummies
978-0-470-59876-4

Physician's Assistant Exam
For Dummies
978-1-118-11556-5

Series 7 Exam For Dummies
978-0-470-09932-2

Windows 8

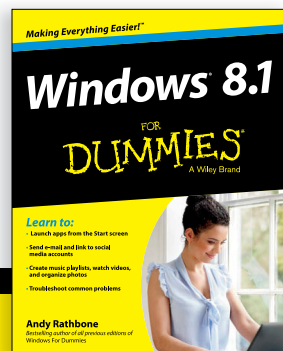
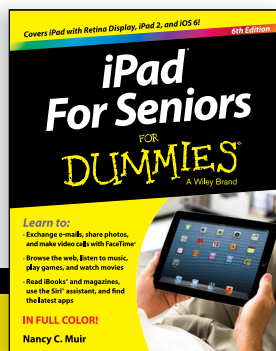
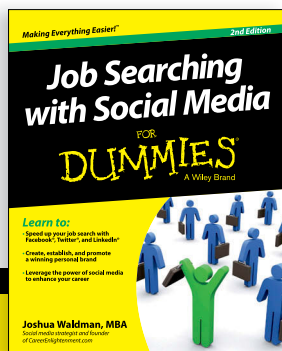
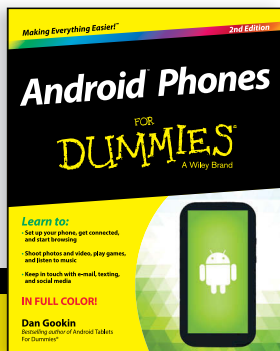
Windows 8.1 All-in-One
For Dummies
978-1-118-82087-2

Windows 8.1 For Dummies
978-1-118-82121-3

Windows 8.1 For Dummies,
Book + DVD Bundle
978-1-118-82107-7



Available in print and e-book formats.



Available wherever books are sold. For more information or to order direct visit www.dummies.com

www.it-ebooks.info

At home, at work, or on the go, Dummies is here to help you go digital!



From eLearning to e-books, test prep to test banks,
language learning to video training, mobile apps, and more,
Dummies makes learning easier.



www.facebook.com/fordummies



www.twitter.com/fordummies

DUMMIES.COM[®]
A Wiley Brand

www.it-ebooks.info

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley's ebook EULA.