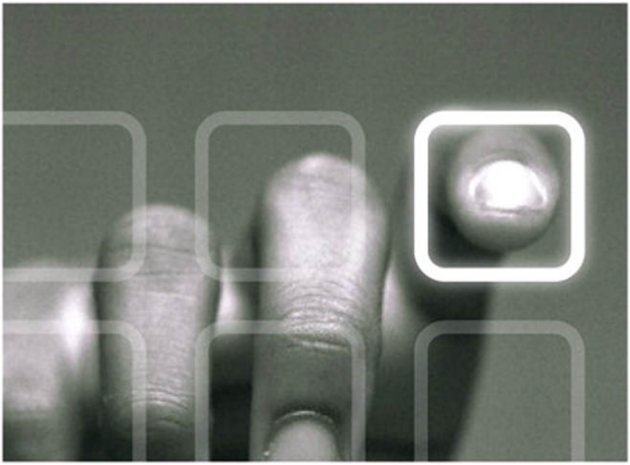


LEARNING iOS GAME PROGRAMMING

A Hands-on Guide to Building Your First iPhone Game

MICHAEL DALEY

Library of Congress cataloging-in-publication data is on file.

ISBN-13: 978-0-321-69942-8
ISBN-10: 0-321-69942-4
Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.
First printing September 2010

Senior Acquisitions Editor
Chuck Toporek

Senior Development Editor
Chris Zahn

Managing Editor
Kristy Hart

Project Editors
Barbara Campbell and Jovana San Nicolas-Shirley

Copy Editor
Water Crest Publishing

Indexer
Lisa Stumpf

Proofreader
Sheri Cain

Publishing Coordinator
Romny French

Cover Designer
Chuti Prasertsith

Contents at a Glance

Preface xxi

1 Game Design 1

2 The Three Ts: Terminology, Technology, and Tools 13

3 The Journey Begins 39

4 The Game Loop 73

5 Image Rendering 97

6 Sprite Sheets 137

7 Animation 153

8 Bitmap Fonts 165

9 Tile Maps 183

10 The Particle Emitter 223

11 Sound 249

12 User Input 285

13 The Game Interface 299

14 Game Objects and Entities 325

15 Collision Detection 357

16 Putting It All Together 369

Index 395

Table of Contents

Preface . . . . . **xxi**

1 Game Design **1**

The Game That Started It All (For Me) 3

So, What's the Big Idea? 4

A Game That Fits *with* the iPhone 4

The Storyline 5

What's in a Name? 5

The Game's Objective 6

Game Play Components 7

Time 7

Lives 7

Health 8

Objects 8

Doors 9

Weapons 10

Entities 10

Player 11

Summary 11

2 The Three Ts: Terminology, Technology, and Tools **13**

Terminology 14

Sprite 14

Sprite Sheet 16

Animation 18

Bitmap Fonts 19

Tile Maps 20

Particle System 21

Collision Detection 22

Artificial Intelligence (AI) 23

Game Loop 24

Technology 26

Objective-C 26

Cocoa Touch 27

OpenGL ES 27

OpenAL 30

Tools 31
The iPhone SDK 32
Summary 38

3 The Journey Begins 39
Creating the Project in Xcode 39
Running the Project 42
Under the Hood 43
Application Delegate 43
Examining the Header File 44
Examining the Implementation File 46
EAGLView 49
EAGLView.h 49
EAGLView.m 50
ES1Renderer 58
Examining ES1Renderer.h 58
Examining ES1Renderer.m 59
Creating the Framebuffer and Renderbuffer 60
Defining the Color Values 66
Positioning 67
How OpenGL Works 68
Applying Transformations on the Model 69
Rendering to the Screen 70
Summary 72

4 The Game Loop 73
Timing Is Everything 73
Collision Detection 74
The Game Loop 75
Frame-Based 75
Time-Based, Fixed Interval 77
Getting Started 78
Inside the EAGLView Class 79
Inside the EAGLView.m File 79
ES1Renderer Class 82
Configuring the View Port 85
Game Scenes and the Game Controller 86

Creating the Game Controller 87
The GameController Class 87
Creating the Singleton 89
Inside GameController.m 89
AbstractScene Class 92
GameScene Class 93
Summary 95
Exercises 95

5 Image Rendering 97
Introduction to Rendering 97
Rendering a Quad 98
Texture Mapping 101
Texture Coordinates 101
Interleaved Vertex Arrays 104
Structures 106
Image Rendering Classes 107
Texture2D Class 108
TextureManager Class 116
ImageRenderManager Class 119
The Image Class 126
Initialization 126
Retrieving a Sub-Image 129
Duplicate an Image 130
Rendering an Image 130
Getters and Setters 134
Summary 134
Exercise 135

6 Sprite Sheets 137
Introduction to Sprite Sheets 137
Simple Sprite Sheet 138
Complex Sprite Sheets 139
Using Zwoptex 141
The SpriteSheet Class 142
Initialization 143
Retrieving Sprites 146

PackedSpriteSheet Class 147
  Initialization 147
  Parsing the Control File 148
  Retrieving a Sprite 149
Summary 150
Exercise 151

7 Animation 153
Animation Chapter Project 153
Introduction to Animation 154
  Frames 154
  State 155
  Type 155
  Direction 155
  Bounce Frame 155
Animation Class 156
  Initialization 156
  Adding Frames 157
  Animation Updates 158
  Animation Rendering 160
  Finishing Things Off 161
Summary 163
Exercise 163

8 Bitmap Fonts 165
Bitmap Font Project 165
Introduction to Bitmap Fonts 166
Creating the Bitmap Font Sprite Sheet 167
The BitmapFont Class 170
  Header File 170
What's with the C? 171
  Initializer 171
  Parsing the Control File 172
Rendering Text 176
  Rendering Justified Text 178
  Text Width and Height 180
  Deallocation 181

Summary . . . 181
Exercise . . . 182

9 Tile Maps . . . 183
Getting Started with the Tile Map Project . . . 183
Introduction to Tile Maps . . . 184
Tile Map Editor . . . 186
  Tile Palette . . . 188
  Layers . . . 188
Creating a Tile Map . . . 189
  Create a New Tile Set . . . 190
  Creating Map Layers . . . 191
  Creating Object Layers . . . 191
  Drawing the Map . . . 192
  Placing Objects . . . 192
Understanding the Tiled Configuration File . . . 193
  Map Element . . . 193
  Tileset Element . . . 193
  Layer Element . . . 194
  Object Group Element . . . 195
Tile Map Classes . . . 196
  Layer Class . . . 196
  TileSet Class . . . 202
  TiledMap Class . . . 204
  Initialization . . . 205
  Parsing a Map File . . . 207
  Creating the Layer Images . . . 216
  Rendering a Layer . . . 218
  Getting Tile Informaiton . . . 220
Summary . . . 220
Exercise . . . 221

10 The Particle Emitter . . . 223
Particle Emitter Project . . . 224
  Introduction to Particle Systems . . . 225
  Particle System Parameters . . . 226

Life Cycle of a Particle . . . 227
A Particle Is Born . . . 227
A Particle Lives . . . 228
A Particle Dies . . . 229
A Particle Is Reborn . . . 229
Particle Emitter Configuration . . . 230
Particle Emitter Classes . . . 231
TBXMLParticleAdditions Class . . . 231
ParticleEmitter Class . . . 233
Have a Play . . . 247
Summary . . . 248

11 Sound . . . 249
Sound Project . . . 249
Introduction to Sound on the iPhone . . . 250
Audio Sessions . . . 250
Playing Music . . . 252
Playing Sound Effects . . . 252
Creating Sound Effects . . . 254
Stereo Versus Mono . . . 256
Sound Manager Classes . . . 256
SoundManager Class . . . 257
Sound Effect Management . . . 273
Loading Sound Effects . . . 274
Playing Sound Effects . . . 276
Stopping Sound Effects . . . 279
Setting Sound Effect and Listener Position . . . 281
Handling Sound Interruptions . . . 281
Summary . . . 284

12 User Input . . . 285
User Input Project . . . 285
Introduction to User Input . . . 287
Touch Events . . . 287
Processing Touch Events . . . 289
The touchesBegan Phase . . . 290
The touchesMoved Phase . . . 292
The touchesEnded Phase . . . 294

Processing Taps 294
Accelerometer Events 296
Summary 298

13 The Game Interface 299
Game Interface Project 299
OpenGL ES Interface 300
Rendering the Interface 301
Defining Button Bounds 304
Handling Touches 304
Handling Transitions 308
OpenGL ES Orientation 308
UIKit Interfaces 312
Creating the Interface 312
Wiring Up the Interface 315
UIKit Orientation 318
Showing and Hiding a UIKit Interface 320
Summary 323

14 Game Objects and Entities 325
Game Objects and Entities Project 325
Game Objects 326
AbstractObject Class 327
EnergyObject Class 329
Game Entities 338
AbstractEntity Class 339
Artificial Intelligence 341
Player Entity Class 343
Saving a Game Object or Entity 352
Summary 355

15 Collision Detection 357
Introduction to Collision Detection 357
Collision Pruning 358
Frame-Based Versus Time-Based 359
Axis-Aligned Bounding Boxes 360

Detecting Collisions 361
Collision Map 362
Entity-to-Map Collision Detection 365
Entity-to-Entity Collision Detection 367
Summary 368

16 Putting It All Together 369
The "Camera" 369
Saving the Game State and Settings 371
  Saving Game State 371
  Loading Game State 373
  Saving Game Settings 375
  Loading Game Settings 376
Saving High Scores 377
  Adding a Score 379
  Saving High Scores 380
  Loading High Scores 381
Performance and Tuning 382
  Using Instruments 383
  Leaks Instrument 384
  Using the OpenGL ES Instrument 387
  Compiling for Thumb 389
Beta Testing 390
  Multiple Device Types 391
  Feedback 392
Summary 392

Index 395

Preface

Writing a game can be a daunting task. Even if you're an experienced programmer, the design patterns, terminology, and thought processes can seem strange and unusual. Having spent most of my working life creating business applications, writing games has been a hobby that has seen me create many games my children have played and enjoyed over the years. With the release of the iPhone and iPod touch, it was time to unleash one of my creations on the world.

My first task was to find a good book on developing games on the iPhone. After a lot of research, I decided that the book I wanted just didn't exist, and having had great feedback on a number of online tutorials I had created, I decided to write my own book. This was a perfect opportunity for me to create the game programming book I've always wanted myself.

Over the years, I've read many game development books and have been left wanting. Although they provide information on the individual components required to make a game and include small examples, they never go all the way to creating a complete game good enough to publish. I've always believed that a good book should both tell the reader what is required to make a game but also demonstrate how those components can be implemented inside a complete game project.

So, this book not only describes the components and technology needed to create a game on the iPhone, but it does so through the creation of a complete game: *Sir Lamorak's Quest: The Spell of Release*. This game is currently available for free download from the App Store, and is the game you learn how to build as you work your way through this book.

Download the Game!

You can download *Sir Lamorak's Quest* from the App Store: **http://itunes.apple.com/us/app/sir-lamoraks-quest-the-spell/id368507448?mt=8**. The game is freely available, so go ahead and download the game, start playing around with it, and help Sir Lamorak escape from the castle!

This book describes the key components needed to create this 2D game. It covers both the technology, such as OpenGL ES and OpenAL, as well as the key game engine components required, including sprite sheets, animation, touch input, and sound.

Each chapter describes in detail a specific component within the game, along with the technology required to support it, be it a tile map editor, or some effect we're trying to create with OpenGL ES. Once an introduction to the functionality and technology is complete, the chapter then provides details on how the component has been implemented within *Sir Lamorak's Quest*. This combination of theory and real-world implementation helps to fill the void left by other game development books.

About Sir Lamorak's Quest

My game-playing experiences started when I was given a Sinclair Spectrum 48k for Christmas in 1982. I was hooked from that moment, and I have had a close relationship with computers ever since.

While thinking about the game I wanted to develop for this book, my mind kept wandering back to the games I played in the 1980s. They may not have been visually stunning, although at the time I was impressed, but they were fun to play.

I spent some time working on the design of the game, which included not only the features I wanted in the game, but also how it should be implemented on the iPhone. One key aspect of the game is that it should be casual—that is, the concept of the game should be simple and easy to pick up, and players should be able to start and stop the game easily without losing their progress.

I also wanted the controls to be easily recognizable and therefore decided to implement an onscreen joypad to control the main character. It was important, though, to allow the player to swap the position of this joypad so that both left- and right-handed players found the game comfortable.

As for the game play itself, I decided to take a number of design ideas from games I played in the '80s and went with a top-down scroller, in which the player is trapped in a haunted castle and has to find a magic spell so that he can escape.

Organization of This Book

There are 16 chapters in the book, each of which deals with a specific area of creating Sir Lamorak's Quest, as follows:

- *Chapter 1, "Game Design"*—This chapter describes the design considerations I made while designing *Sir Lamorak's Quest*. It provides an insight into the kind of thought process required when sitting down to create a game. It doesn't cover *every* possible design decision needed for all genres of games, but it does cover the important ones.

- *Chapter 2, "The Three Ts: Terminology, Technology, and Tools"*—Even experienced programmers can become confused by the three Ts used within game development. This chapter runs through the common technology, terminology, and tools used to create *Sir Lamorak's Quest* and games in general. This chapter helps you understand the terms and technology covered throughout the book.
- *Chapter 3, "The Journey Begins"*—This is where we start to get our hands on some code and get the iPhone to render something to the screen. This chapter covers the process of creating our first project using the OpenGL ES template project within Xcode. The template is described in detail and sets the scene for the chapters that follow.
- *Chapter 4, "The Game Loop"*—The heartbeat of any game is the game loop. This loop is responsible for making sure that all the core elements of the game, such as AI and rendering, are done at the right time and in the right order. This may sound simple, but there are a number of different approaches to the game loop, and this chapter discusses them and details the approach taken *for Sir Lamorak's Quest*.
- *Chapter 5, "Image Rendering"*—Drawing images to the screen is a fundamental requirement for any game. This chapter provides an overview of OpenGL ES and runs through a number of classes created to simplify the creation and rendering of images to the screen.
- *Chapter 6, "Sprite Sheets"*—Sprite sheets are images that contain a number of smaller images. These sheets can be used to reduce the number of individual images held in memory and the number of different textures OpenGL ES needs to bind to improving performance. They are also commonly used when creating animated sprites. This chapter covers how to create sprite sheets that contain the images used in the game, regardless of whether they have fixed or variable dimensions.
- *Chapter 7, "Animation"*—Having created the means to store the different frames needed in an animation using sprite sheets, this chapter describes how separate images can be played in sequence to provide you with animation, such as the player character running.
- *Chapter 8, "Bitmap Fonts"*—The most common way to interact with your game's user is through the use of text. Being able to render instructions and information (such as the player's score or instructions on how to use the game) is important. This chapter describes how you can use open source tools to take any font and turn it into a bitmap font. Once the bitmap font is created, you'll see how to create a sprite sheet that contains all the images needed to render the characters in that font. It also details the `Bitmap` font class used in *Sir Lamorak's Quest*, which provides a simple API for rendering text to the screen.

- *Chapter 9, "Tile Maps"*—Tile maps allow large game worlds to be created from reusing a small number of tile images. This common approach has been used in the past to create large game worlds (think of the original *Super Mario Brothers* game for Nintendo) when memory is limited, back in the early days of home game systems. This technique is still popular today, and this chapter describes the use of an open source tile-editing tool to create tile maps, along with a class that can render these maps to the screen.
- *Chapter 10, "The Particle Emitter"*—Many games have impressive effects, such as fire, explosions, smoke, and sparks. These are created using a particle system. The particle system is responsible for creating and controlling a number of particles; each has its own properties, such as size, shape, direction, color, and lifespan. During a particle's life cycle, its position, speed, color, and size are changed based on the particle's configuration. This chapter details how to create a particle system that can be used to generate any number of organic effects.
- *Chapter 11, "Sound"*—Giving the player feedback using sound is important in today's modern games. This chapter describes how the media player functionality of the iPhone, along with OpenAL, can be used to play a cool soundtrack in the game, as well as 3D (surround) sound effects.
- *Chapter 12, "User Input"*—This chapter describes how to use the iPhone's unique touch and accelerometer capabilities to control your game. It details how to capture and process multiple touches at the same time and also how data from the accelerometer can be used within your own games.
- *Chapter 13, "The Game Interface"*—In this chapter, we start to look at how the game interface for *Sir Lamorak's Quest* was implemented. This includes how to deal rotation events to make sure that the user interface is always oriented correctly. It also describes how to mix both OpenGL ES and UIKit interface controls.
- *Chapter 14, "Game Objects and Entities"*—As the player runs around the castle in *Sir Lamorak's Quest*, we want him to be able to find objects, pick them up, and fight baddies. This chapter describes how objects and entities have been implemented within *Sir Lamorak's Quest*.
- *Chapter 15, "Collision Detection"*—Having the player and baddies run through walls and doors would really spoil the game, so it's important to be able to register collisions between either the player and the map or objects and entities within the castle. This chapter describes different types of collision detection and how this has been implemented within *Sir Lamorak's Quest*.
- *Chapter 16, "Pulling It All Together"*—At this point, a great deal of ground has been covered. There is, however, a number of things you can do to the game to add polish. This chapter covers how to save the player's game state for when he quits or leaves the game when he has an incoming call. Chapter 16 also covers performance tuning using instruments and tips for getting your game beta tested.

Audience for This Book

This book has been written for people who are already programmers but who have never written computer games before. Although it assumes that you already have some experience with Objective-C, each chapter provides enough information on both Objective-C and other technologies so you can follow the concepts and implementations.

By the time you complete this book, you will have an in-depth understanding of the game engine that was built for *Sir Lamorak's Quest* and the key capabilities and considerations are needed to create a 2D game engine. This enables you to take the same game engine developed in this book and use it in your own games, or simply use the knowledge you have gained about creating games in general and use one of the many game engines available for the iPhone, such as Cocos2D.

Who This Book Is For

If you are already developing applications for the iPhone for other platforms, but want to make a move from utility applications to games, this book is for you. It builds on the development knowledge you already have and leads you into game development by describing the terminology, technology, and tools required, as well as providing real-world implementation examples.

Who This Book Isn't For

If you already have a grasp of the workflow required to create a game or you have a firm game idea that you know requires OpenGL ES for 3D graphics, this is not the book for you.

It is expected that before you read this book, you are already familiar with Objective-C, C, Xcode, and Interface Builder. Although the implementations described in this book have been kept as simple as possible and the use of C is limited, a firm foundation in these languages is required.

The following titles can help provide you with the grounding you need to work through this book:

- *Cocoa Programming for Mac OS X, Third Edition*, by Aaron Hillegass (Addison-Wesley, 2008).
- *Learning Objective-C 2.0*, by Robert Clair (Addison-Wesley, 2011).
- *Programming in Objective-C 2.0*, by Stephen G. Kochan (Addison-Wesley, 2009).
- *Cocoa Design Patterns*, by Erik M. Buck and Donald A. Yacktman (Addison-Wesley, 2009).

- *The iPhone Developer's Cookbook, Second Edition*, by Erica Sadun (Addison-Wesley, 2010).
- *Core Animation: Simplified Animation Techniques for Mac and iPhone Development*, by Marcus Zarra and Matt Long (Addison-Wesley, 2010).
- *iPhone Programming: The Big Nerd Ranch Guide*, by Aaron Hillegass and Joe Conway (Big Nerd Ranch, Inc., 2010).

These books, along with other resources you'll find on the web, will help you learn more about how to program for the Mac and iPhone, giving you a deeper knowledge about the Objective-C language and the Cocoa frameworks.

Download the Source Code

Access to information is not only limited to the book. The complete, fully commented source code to *Sir Lamorak's Quest* is also available for download on InformIT.com.

There is plenty of code to review throughout this book, along with exercises for you to try out, so it is assumed you have access to the Apple developer tools, such as Xcode and the iPhone SDK. Both of these can be downloaded from the Apple iPhone Dev Center.[2]

[2] Apple's iPhone DevCenter: developer.apple.com/iphone.

6 Sprite Sheets

Chapter 5, "Image Rendering," was large and covered a number of complex concepts. Having done all that hard work, and with the classes in place for representing and rendering images, we can move on to the other components needed in the game engine for Sir Lamorak's Quest.

As the title suggests, this chapter is all about sprite sheets. If you remember from Chapter 2, "The Three Ts: Terminology, Technology, and Tools," a *sprite sheet* is a large image that contains a number of smaller images.

There are two key benefits to using sprite sheet, as follows:

- You reduce the number of times you need to ask OpenGL ES to bind to a new texture, which helps with performance.
- You gain the ability to easily define and reuse image elements of the game, even in animations.

This chapter reviews the `SpriteSheet` and `PackedSpriteSheet` classes and shows how to extract specific images from within a larger image sprite sheet.

Introduction to Sprite Sheets

As mentioned in Chapter 2, there are two different types of sprite sheets, as follows:

- Basic, where all the images in the sprite sheet have the same dimensions.
- Complex, where the images in the sprite sheet could all have different dimensions.

For *Sir Lamorak's Quest*, we are going to be using both kinds of sprite sheets. Although it is possible to merge both the simple and complex sprite sheet functionality into a single class, I have split them into two different classes to make things easier to understand. Basic sprite sheets are handled in a class called `SpriteSheet`, whereas the `PackedSpriteSheet` class handles complex sprite sheets.

> **Note**
> I use the term *packed* because you can place smaller sprite sheets within this larger sprite sheet, thus reducing the number of separate sprite sheets used in the game.

Another term for a sprite sheet is a *texture atlas*, but I will continue to use the old-school term of "sprite sheet" throughout this book.

Simple Sprite Sheet

The `SpriteSheet` class takes the image provided and chops it up into equally sized sub-images (sprites). The dimensions to be used when dividing up the sprite sheet will be provided when a new sprite sheet is instantiated. Information is also provided about any spacing that has been used within the provided sprite sheet image. Spacing is an important property within a sprite sheet. Without going into detail, when defining texture coordinates within an image for OpenGL ES, it is possible to sample a pixel beyond the edge of the texture you are defining. This can cause your textures to have an unwanted border that is made up of pixels from the image around the image defined with your texture coordinates. This is known as *texture bleeding*.

To reduce the risk of this happening, you can place a transparent border around each image within a sprite sheet. If OpenGL ES then goes beyond the edge of your texture, it will only sample a transparent pixel, and this should not interfere with the sprite you have defined. Zwoptex[1] enable you to specify the number of pixels you would like to use as a border around your sprites. Figure 6.1 shows a simple sprite sheet image with single pixel border between each sub-image. If you are drawing non-square triangles, the spacing may need to be more than one pixel to help eliminate texture bleeding.

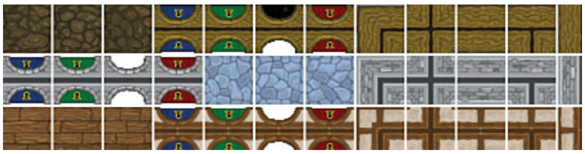


Figure 6.1 Sprite sheet with spacing between each sprite.

In terms of how we are going to access the sprites on a simple sprite sheet, we're going to use its grid location. A simple sprite sheet makes a nice grid because all the images are the

[1] Zwoptex (**www.zwoptexapp.com/flashversion/**) is a Flash-based sprite sheet builder. There is also a Cocoa-based version of this tool available. This Cocoa version generates output the same as the flash version, but was not available during the writing of this book.

same size. This makes it easy to retrieve a sprite by providing its row and column number. Figure 6.2 shows a sprite sheet of twelve columns and three rows with the sprite at location `{5, 1}` highlighted.

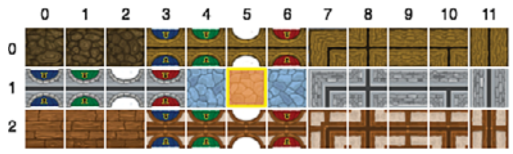


Figure 6.2 Sprite sheet grid with location {5, 1} highlighted.

Complex Sprite Sheets

The `PackedSpriteSheet` class takes an image and the name of the control file. The control file is parsed to obtain the location and size of every sprite within the sprite sheet image.

The control file is the key difference between a basic (`SpriteSheet`) and complex (`PackedSpriteSheet`) sprite sheet. With the basic sprite sheet, you can work out where each sprite is by performing a simple calculation using its grid position. This is harder to do with a complex sprite sheet because the sprites can be different sizes and are often placed randomly throughout the image to make the best use of space.

To help identify the coordinates of the sprites in a complex sprite sheet, the control file provides information on where each sprite is located inside the sprite sheet, along with its dimensions. The control file also gives each image a *key*, usually the name of the image file of the original sub-image, which then allows the `PackedSpriteSheet` class to reference each sprite. Figure 6.3 shows the complex sprite sheet that we use in *Sir Lamorak's Quest*.

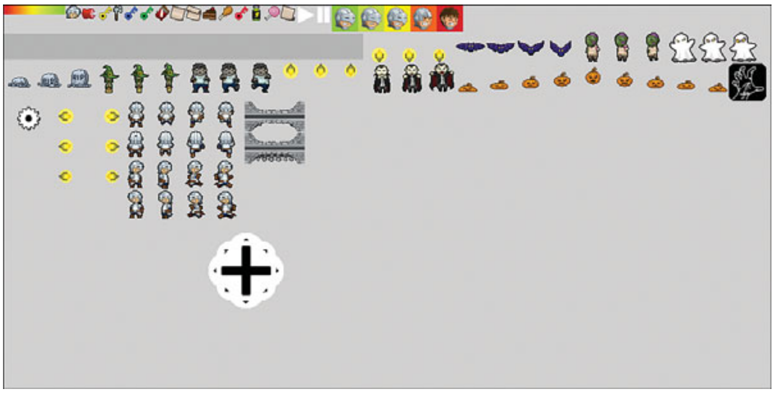


Figure 6.3 Complex sprite sheet from *Sir Lamorak's Quest*.

As you can see from Figure 6.3, a complex sprite sheet has many images that are all different sizes and shapes—thus the need for a control file to make sense of it all.

You could create your own control file for these files, providing the information on the pixel locations within the image and its dimensions, but to be honest, that is a really tedious job. Luckily for us, there are tools that can help.

The Zwoptex tool (mentioned earlier, and discussed in Chapter 2) is one such tool. It not only produces a PNG image of the generated sprite sheet, but it also creates the control file you need to identify the individual images within.

Zwoptex has a number of different algorithms that can help pack images, but it also enables you to move the images around, making it possible for you to pack as many images as possible into a single sheet. There are some good algorithms out there for optimizing the packing of variably sized images, but you'll always get the best results doing this manually.

Figure 6.4 shows the flash version of Zwoptex editing the complex sprite sheet.

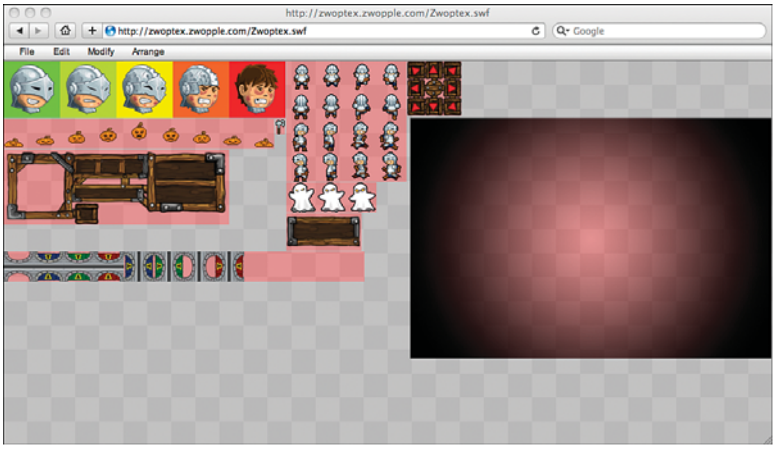

http://zwoptex.zwopple.com/Zwoptex.swf
http://zwoptex.zwopple.com/Zwoptex.swf
Google
File
Edit
Modify
Arrange

Figure 6.4 The Flash-based Zwoptex tool, used for editing a complex sprite sheet.

Zwoptex has three different outputs, as follows:

- A project file that stores your settings and images for a particular sprite sheet
- A PNG image of the sprite sheet
- A *plist* control file, which you can add to your game

The thing I like the most about Zwoptex is that it gave me the control file as a *plist* file. Although you can obviously handle raw XML if needed (or any other format, for that matter), having a *plist* file makes things so much easier (and I like to take the easy route whenever possible).

Now that you know what Zwoptex is, let's show you how to use it.

Using Zwoptex

Using Zwoptex is really easy. Just point your browser to **www.zwoptexapp.com/flashversion/**. Once there, Zwoptex opens, and you can start creating your sprite sheet.

The first step is to import images. Start by going to the menu **File > Import Images** (see Figure 6.5), and you see an **Open File** panel for you to navigate to the file(s) you want to import.

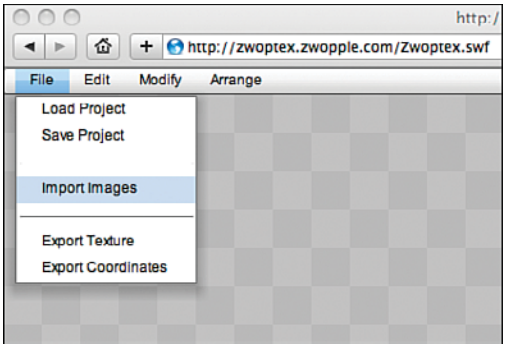


Figure 6.5 Import images into the sprite sheet.

After you select your images, hit the **Select** button to load the images into Zwoptex. All the images you've selected will be placed at the top-left corner of the screen, as shown in Figure 6.6.

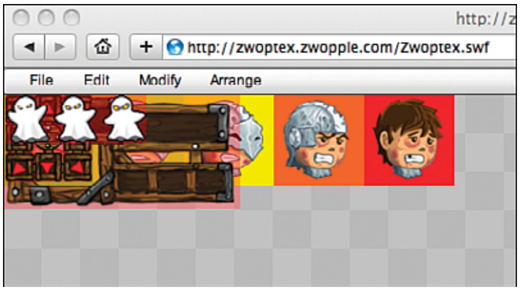


Figure 6.6 Zwoptex imports the images in the top-left corner of the canvas.

Now that you've placed the images in Zwoptex, there are a number of ways to arrange the sprites on the canvas. Under the **Arrange** menu, you will find different options for laying out the sprites. Figure 6.7 shows the sprites having been laid out using the **Complex By Width (no spacing)** option.

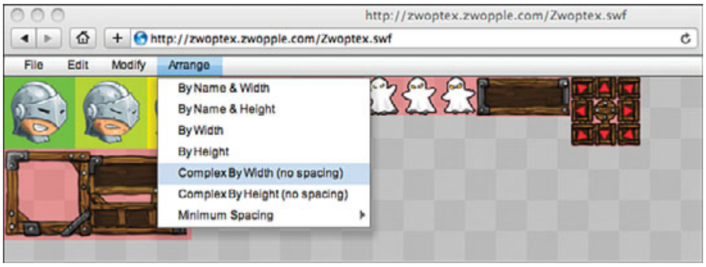

http://zwoptex.zwopple.com/Zwoptex.swf
http://zwoptex.zwopple.com/Zwoptex.swf
File
Edit
Modify
Arrange
By Name & Width
By Name & Height
By Width
By Height
Complex By Width (no spacing)
Complex By Height (no spacing)
Minimum Spacing

Figure 6.7 Sprite options menu and arranged sprites.

You can do this manually by clicking any sprite and moving it to the position you want. You can also use the **Modify** menu to change the size of the canvas to fit your needs.

By default, Zwoptex trims transparent edges from the imported images. This can be a problem, however, if the image you imported will be used as a simple sprite sheet. These images need to retain their original dimensions or the calculations used to define the position of each sprite will be incorrect.

Within the **Modify** menu is the option to **Untrim Selected Images**. This should be used to ensure that the images are returned to their original size. This is not necessary if the image won't be used as a sprite sheet.

Having arranged your sprites, you can then export both the image (texture) and the control file (coordinates). There are two options within the **File** menu that let you do this: **Export Texture** and **Export Coordinates**. Both options enable you to select the location where you would like the file(s) saved.

That's it! You now have a sprite sheet image file and its accompanying control file.

The SpriteSheet Class

Having looked at the basics of a sprite sheet, we can now look at our implementation of the `SpriteSheet` class. In Xcode, open the *CH06_SLQTSOR* project and look inside the `Game Engine` group. You will see a new group called *Sprite Sheet*, inside of which are the `SpriteSheet` classes header and implementation files.

Initialization

Inside the *SpriteSheet.m* file, you find the following class methods:

- `spriteSheetForImageNamed:spriteSize:spacing:margin: imageFilter`
- `spriteSheetForImage:sheetKey:spriteSize:spacing:margin:`

These methods are used to create new sprite sheets either from an image file or from an `Image` instance that has already been created. Notice that both of these are class methods. This means you don't need an instance of the `SpriteSheet` class to access them. Having also defined a static `NSDictionary` within the class, you can use these class methods to access the dictionary information that only has a single instance.

The idea is that a sprite sheet is cached when it is created. Whenever a new sprite sheet that either uses the same image file or key is requested, a reference to the sprite sheet already created is returned. This helps with performance when you have a large number of entities that share the same sprite sheet (for example, the `Door` class, which you will see soon).

These class methods still make use of the standard initializer methods; they just cache the sprite sheet returned by these methods for later use. Listing 6.1 shows the `spriteSheetForImageNamed:spriteSize:spacing:margin:imageFilter:` method.

Listing 6.1 **The spriteSheetForImageNamed:spriteSize:spacing:margin:imageFilter: Method**

```
static NSMutableDictionary *cachedSpriteSheets = nil;

+ (SpriteSheet*)spriteSheetForImageNamed:(NSString*)aImageName
      spriteSize:(CGSize)aSpriteSize spacing:(NSUInteger)aSpacing
      margin:(NSUInteger)aMargin imageFilter:(GLenum)aFilter {

    SpriteSheet *cachedSpriteSheet;

    if (!cachedSpriteSheets)
        cachedSpriteSheets = [[NSMutableDictionary alloc] init];

    if(cachedSpriteSheet = [cachedSpriteSheets objectForKey:aImageName])
        return cachedSpriteSheet;

    cachedSpriteSheet = [[SpriteSheet alloc]
        initWithImageNamed:aImageName spriteSize:aSpriteSize
        spacing:aSpacing margin:aMargin imageFilter:aFilter];
    [cachedSpriteSheets setObject:cachedSpriteSheet forKey:aImageName];
    [cachedSpriteSheet release];

    return cachedSpriteSheet;
}
```

The first line in Listing 6.1 defines a static `NSMutableDictionary`. This creates a single instance of `NSMutableDictionary` that the class methods use to cache the sprite sheets. This dictionary has been defined at the class level, which means that only a single copy of this dictionary will exist, regardless of how many `SpriteSheet` instances are created. This provides us with a single cache of the sprite sheets.

The rest of the class simply checks to see if an entry already exists in the dictionary for an image name passed in (using `spriteSheetForImageNamed`). If the other method passes in a ready-made image, the `sheetKey` provided is used.

If no match is found, a new sprite sheet is created and added to the dictionary. Otherwise, the matching entry from the dictionary is passed back to the caller.

The initializer used when an image name is provided is shown in Listing 6.2.

Listing 6.2 **SpriteSheet initWithImageNamed:spriteSize:spacing:margin:imageFilter Method**

```
- (id)initWithImageNamed:(NSString*)aImageFileName
    spriteSize:(CGSize)aSpriteSize spacing:(NSUInteger)aSpacing
    margin:(NSUInteger)aMargin imageFilter:(GLenum)aFilter {

    if (self = [super init]) {
        NSString *fileName = [[aImageFileName lastPathComponent]
            stringByDeletingPathExtension];

              self.image = [[Image alloc]
                   initWithImageNamed:filename filter:aFilter];

              spriteSize = aSpriteSize;
              spacing = aSpacing;
              margin = 0;

              [self cacheSprites];
    }
    return self;
}
```

The start of the initializer method is standard, and we have seen it many times already. The first interesting action comes when we create an image instance of the image used as the sprite sheet.

We are using the `Image` class that we created in the last chapter, passing in the image name that has been provided along with the image filter.

Next, the sprite's size, spacing, and margin are defined. At this point, we branch off and call a private method, called `cacheSprites`, which caches the information for each sprite in this sprite sheet. Calculating this information only once is important to help performance. This information should never change during the lifetime of a sprite sheet, so there is no need to calculate each time we request a particular sprite.

We examine the `cacheSprites` method in a moment; first, there is another initializer method to look at, as shown in Listing 6.3.

Listing 6.3 **SpriteSheet initWithImage:spriteSize:spacing:margin Method**

```
- (id)initWithImage:(Image*)aImage spriteSize:(CGSize)aSpriteSize
      spacing:(NSUInteger)aSpacing margin:(NSUInteger)aMargin{
    if (self = [super init]) {
        self.image = aImage;

        spriteSize = aSpriteSize;
        spacing = aSpacing;
        margin = aMargin;

        [self cacheSprites];
    }
    return self;
}
```

The previous initializer took the name of an image file and created the image as part of creating the sprite sheet. This second initializer takes an image that's already been created. Not only is it useful to create a sprite sheet using an image instance that already exists, but it is also the method that's used when we create a sprite sheet from an image held in a complex (or packed) sprite sheet.

The only difference in this initializer from the last is that we set the sprite sheet's image to reference the `Image` instance that has been passed in. This method still calls the `cacheSprites` method, and that's the next method we discuss.

The `cacheSprites` method (shown in Listing 6.4) is a private method, as we only use it internally in the `SpriteSheet` class.

Listing 6.4 **SpriteSheet cacheSprites Method**

```
- (void)cacheSprites {

    horizSpriteCount = ((image.imageSize.width + spacing) + margin) /
        ((spriteSize.width + spacing) + margin);
    vertSpriteCount = ((image.imageSize.height + spacing) + margin) /
        ((spriteSize.height + spacing) + margin);

    cachedSprites = [[NSMutableArray alloc] init];
    CGPoint textureOffset;

    for(uint row=0; row < vertSpriteCount; row++) {
        for(uint column=0; column < horizSpriteCount; column++) {

            CGPoint texturePoint = CGPointMake((column *
                (spriteSize.width + spacing) + margin),
                (row * (spriteSize.height + spacing) + margin));

            textureOffset.x = image.textureOffset.x *
                image.fullTextureSize.width + texturePoint.x;
            textureOffset.y = image.textureOffset.y *
                image.fullTextureSize.height + texturePoint.y;
            CGRect tileImageRect = CGRectMake(textureOffset.x,
                textureOffset.y, spriteSize.width, spriteSize.height);

            Image *tileImage = [[image subImageInRect:tileImageRect]
                retain];

            [cachedSprites addObject:tileImage];

            [tileImage release];
        }
    }
}
```

The first two calculations work out how many sprites there are in the sprite image, and a new `NSMutableArray` is created. This array holds `Image` instances created for each image in the sprite sheet. Again, creating the images at this stage and caching them improves performance. This is not an activity you want to be performing in the middle of game play.

With the array created, we then loop through each row and column, creating a new image for each sprite. We use the information we have about the sprite sheet, such as size, spacing, and margin, to calculate where within the sprite sheet image each sprite will be. With this information, we are now able to use the `subImageInRect` method of the `Image` class to create a new image that represents just the sub-image defined.

Retrieving Sprites

Having set up the sprites on the sprite sheet, the next key activity is to retrieve sprites. We have already discussed that one of the key tasks of the `SpriteSheet` class is to return an `Image` class instance configured to render a single sprite from the sprite sheet, based on the grid location of the sprite.

The `spriteImageAtCoords:` method shown in Listing 6.5 implements the core mechanism for being able to retrieve a sprite.

Listing 6.5 **SpriteSheet spriteImageAtCoords: Method**

```
- (Image*)spriteImageAtCoords:(CGPoint)aPoint {

    if(aPoint.x > horizSpriteCount-1 || aPoint.y < 0 || aPoint.y >
        vertSpriteCount-1 ||
        aPoint.y < 0)
        return nil;

    int index = (horizSpriteCount * aPoint.y) + aPoint.x;

    return [cachedSprites objectAtIndex:index];
}
```

The first check we carry out in this class is on the coordinates that are being passed in. This method takes the coordinates for the sprite in a `CGPoint` variable. `CGPoint` has an `x` and `y` value that can be used to specify the grid coordinates in the sprite sheet.

When we know that the coordinates are within the sprite sheet, we use the coordinates of the sprite to calculate its location within the `NSMutableArray`. It's then a simple task of retrieving the image from that index and passing it back to the caller

That's it for this class. It's not that long or complex, but it does provide an important building block within our game engine.

PackedSpriteSheet Class

As mentioned earlier, the `PackedSpriteSheet` class is responsible for dealing with complex sprite sheets. These sprite sheets contain many variably sized images to which we want to get access. This often includes other sprite sheets. This class can be found in the same group within the *CH06_SLQTSOR* project, as before.

Initialization

This class uses the same caching technique as the `SpriteSheet` class. There is, however, only one initializer, which is shown in Listing 6.6.

Listing 6.6 **PackedSpriteSheet initWithImageNamed:controlFile:filter Method**

```
- (id)initWithImageNamed:(NSString*)aImageFileName
controlFile:(NSString*)aControlFile
      filter:(GLenum)aFilter {

    if (self = [super init]) {
        NSString *fileName = [[aImageFileName lastPathComponent]
            stringByDeletingPathExtension];

        image = [[[Image alloc] initWithImageNamed:fileName
            filter:aFilter] retain];

        sprites = [[NSMutableDictionary alloc] init];

        controlFile = [[NSDictionary alloc]
            initWithContentsOfFile:[[NSBundle mainBundle]
            pathForResource:aControlFile ofType:@"plist"]];
        [self parseControlFile:controlFile];
        [controlFile release];
    }
    return self;

}
```

Once inside the initializer, we create a new `Image` instance from the details passed in and allocate an `NSMutableDictionary` instance called `sprites` that will hold the details of the sprites in our packed sprite sheet.

The last section of the initializer grabs the contents of the control file that were passed in and loads it into an `NSDictionary` called `controlFile`. It is always assumed that the type of file is a *plist*, so the file type is hard coded. After we have the `controlFile` dictionary populated, we then parse the information inside that dictionary using the private `parseControlFile` method shown in Listing 6.7.

Listing 6.7 **PackedSpriteSheet parseControlFile: Method**

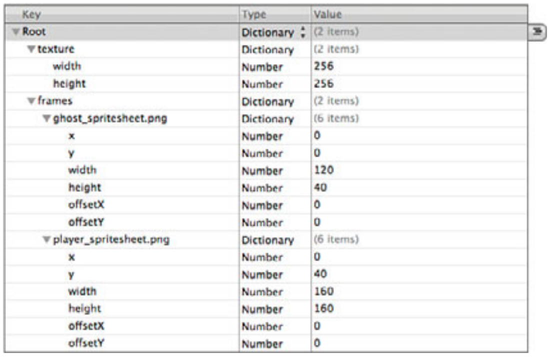
```
- (void)parseControlFile:(NSDictionary*)aControlFile {

    NSDictionary *framesDictionary = [controlFile objectForKey:@"frames"];

    for (NSString *frameDictionaryKey in framesDictionary) {

        NSDictionary *frameDictionary = [framesDictionary
            objectForKey:frameDictionaryKey];

        float x = [[frameDictionary objectForKey:@"x"] floatValue];
        float y = [[frameDictionary objectForKey:@"y"] floatValue];
        float w = [[frameDictionary objectForKey:@"width"] floatValue];
        float h = [[frameDictionary objectForKey:@"height"] floatValue];

        Image *subImage = [image subImageInRect:CGRectMake(x, y, w, h)];
        [sprites setObject:subImage forKey:frameDictionaryKey];
    }
}
```

Parsing the Control File

The `parseControlFile` method creates a dictionary from all the `frames` objects within the dictionary we passed in. There are several objects inside the *plist* file, as follows:

- Texture, which holds the dimensions of the texture.
- Frames, which hold objects keyed on the image's filename for each image in the sprite sheet.

An example of the *plist* file inside the Plist Editor can be seen in Figure 6.8.



Key	Type	Value
▼Root	Dictionary	(2 items)
▼texture	Dictionary	(2 items)
width	Number	256
height	Number	256
▼frames	Dictionary	(2 items)
▼ghost_spritesheet.png	Dictionary	(6 items)
x	Number	0
y	Number	0
width	Number	120
height	Number	40
offsetX	Number	0
offsetY	Number	0
▼player_spritesheet.png	Dictionary	(6 items)
x	Number	0
y	Number	40
width	Number	160
height	Number	160
offsetX	Number	0
offsetY	Number	0

Figure 6.8 Sprite sheet plist control file.

The details we want for the sprites are therefore held in the frame's objects.

Now that we have a dictionary called `frames`, we loop through each of them, extracting the information we need. For each frame we find, we assign another `NSDictionary` that contains the objects for the key we are dealing with. Remember that the key is a string that contains the name of the original image file that was embedded into the larger sprite sheet. This makes it easy later on to reference the image we need.

Once we have the information for the frame, we then add a new object to our `sprites` dictionary. The key is the name of the image file we have just read from the control file, and the object is an `Image` instance.

Getting a sub-image from the full sprite sheet image creates the `Image` instance. Again, we are just making use of functionality we have already built.

This process is repeated for each image in the sprite sheet control file, and we end up with a dictionary that contains an image representing each image in our packed sprite sheet.

Retrieving a Sprite

Having all our sprites in a dictionary now makes retrieving a sprite from our `PackedSpriteSheet` very simple. This is done using the `imageForKey` method. Listing 6.8 shows this method.

Listing 6.8 **PackedSpriteSheet imageForKey Method**

```
- (Image*)imageForKey:(NSString*)aKey {
    Image *spriteImage = [sprites objectForKey:aKey];
    if (spriteImage) {
        return [sprites objectForKey:aKey];
    }
    NSLog(@"ERROR - PackedSpriteSheet: Sprite could not be found for key
        '%@'", aKey);
    return nil;
}
```

We pass an `NSString` into this method containing the key to the sprite's dictionary that we created earlier. If you remember, the key is the filename of the image that was placed inside the packed sprite sheet. If an image is found for the key supplied, a reference to this image is returned. Otherwise, an error is logged, so we know that the sprite we wanted could not be found.

> **Note**
>
> Notice that, in some methods, an error is raised using `NSLog`. This is handy when debugging your game, but this is also a huge performance hog. To reduce the possibility of an `NSLog` message being called in the production code, it would be worth only generating the log messages when running in debug code.

Summary

In this chapter, we have reviewed the `SpriteSheet` and `PackedSpriteSheet` classes that continue to build out our game engine for *Sir Lamorak's Quest*. These classes enable us to retrieve sub-images from within a specified image in a number of ways:

- `SpriteSheet` **class:** As a new `Image` instance based on a sprite's grid location.
- `PackedSpriteSheet` **class:** As an `Image` reference based on a sprite's key (for example, the sub-image's original filename).

These important classes enable us to not only manage the number of textures we need, but also provide us with a mechanism for grabbing the images needed to create animation.

Classes such as `Image`, `SpriteSheet`, and `PackedSpriteSheet` are the building blocks that form the backbone of our game engine. Being comfortable with how they work and how they can be used enable you to get the most out of the game engine itself, as well as a clearer view of how to implement your own games. Although the game engine we are building for *Sir Lamorak's Quest* is not suited to all types of games, it provides you with the basis for any future games you want to develop. This enables you to take the game engine in new directions as your needs and experience grow.

The next chapter covers animation. It's not exactly Pixar Animation,[2] but animation nonetheless.

[2] Pixar Animation is an award-winning computer animation studio responsible for feature films such as *Toy Story*, *Monsters, Inc.*, and *Finding Nemo*, among many others.

Exercise

The example project that is provided with this chapter, *CH06_SLQTSOR*, displays three different images that have been taken from a single sprite sheet. These images are scaled, rotated, and colored using the features of the `Image` class covered in Chapter 5 to show that the `Image` instance returned is an entirely separate image in its own right.

The current project is using a couple of sprite sheets from *Sir Lamorak's Quest* that have been placed inside a complex sprite sheet.

Using this project as a guide, why not try to create your own basic sprite sheet or download one from the Internet? Once you have your sprite sheet, create a complex sprite sheet using Zwoptex and then render your sprites to the screen.

Here are the steps you need to follow:

1. Decide what fancy sprites you want to create.
2. Work out the dimensions each sprite is going to be (for example, 40×40 or 50×80) and any spacing you want to use.
3. Open up your favorite graphics package and draw your sprites, remembering to keep each sprite in a square that has the dimensions you decided.
4. Export your sprite sheet as a PNG file.
5. Open up the Zwoptex link (**www.zwoptexapp.com/flashversion/**), and add the sprite sheets that are included in the project along with your own.
6. Export the texture and coordinates from Zwoptex.
7. Add the two files you have just generated to the Xcode project. This can be done by right-clicking the Images group inside the Game Resources group and selecting **Add > Add Existing File**. Inside the panel that pops up, navigate to the file and select it. You should also select the Copy option to make sure the files are copied to the project folder.
8. Finally, follow the code example in the current project to import and start using your sprite sheet.
9. Once you are rendering your sprites, try to apply some image functions, such as scaling, rotation, and color.

Index

A

A*, 343

AABB (Axis-Aligned Bounding Boxes), 360-361

abstract classes, 79

AbstractEntity class, 339

AbstractObject class, 327-329

AbstractObject Methods, 329

AbstractScene, 92-93

accelerometer, 285

accelerometer events, 296-298

adding
- frames, Animation class, 157-158
- images to render queue, ImageRenderManager, 120-123
- object layers to maps, 335
- particles, ParticleEmitter class, 243-244
- scores, to high scores, 379-380
- tile images, to layers, 199-200
- tiles, to layers, 198-199

AI (artificial intelligence), 23
- game entities, 341-343

alBufferData, 276

alBufferDataStaticProc, 276

alertView:clickedButtonAtIndex: method, 378

alGenSources, 261

AngelCode, 20

angleOfMovement, 348-349

animation, 18-19
bounce frames, 155–156
direction, 155
frames, 154–155
projects, 153–154
rendering with Animation class, 160–161
states, 155
types, 155
updates, Animation class, 158–160
Animation class, 156
animation
rendering, 160–161
updates, 158–160
finishing things off, 161–163
frames, adding, 157–158
initialization, 156–157
animationFrameInterval, 55
applicationWillTerminate, 43
application delegates, 43-44
header files, examining, 44–46
implementation files, examining, 46–49
applicationDidBecomeActive, 43
applicationDidFinishLaunching, 43
applicationWillResignActive, 43
arrays, vertex arrays, 104
artificial intelligence (AI), 23
game entities, 341–343
audio playback formats, 255
audio sessions, 250-251
AudioServices, 273
AVAudioPlayer, 255
AVAudioSessionCategorySoloAmbient, 250-251
Axe class, 339
Axis-Aligned Bounding Boxes (AABB), 360-361

B

beta testing, 390-391
feedback, 392
multiple device types, 391
binding, texture and setting parameters, Texture2D, 113-114
bitmap fonts, 19-20, 165-167
C, 171
initializer, 171–172
parsing control files, 172–174
projects, 165
rendering text, deallocation, 181
sprite sheets, creating, 167–170
BitmapFont class, 170
header files, 170–171
bounce frames, animation, 155-156
boundaries, visualizing, 306-308
button bounds, defining, 304
bitmap fonts, 171
initializer, 171–172
parsing control files, 172–174

C

cachedTextures, 118
CADisplayLink, 56
CAEAGLLayer, 52
calloc command, 157
cameras, 369-371
CGPoint, 147
CGRectContainsPoint function, 305
char id, 174
parsing, 175–176
cheating, AI (artificial intelligence), 342
checkForCollisionWithEntity: method, 333-334, 367-368
checkForParchment:pickup: method, 351

checkJoypadSettings method, 322
@class, 45
clipping, 67
Cocoa Design Patterns, 312
Cocoa Touch, 27
Cocoa Touch Class, 312
codecs, 255
collision detection, 22-23, 357-358, 361-362
  Axis-Aligned Bounding Boxes (AABB), 360-361
  entity-to-entity collision detection, 367-368
  entity-to-map collision detection, 365-366
  frame-based versus time-based, 359-360
  game loops, 74-75
collision maps, 362-365
collision method, 334
collisionBounds method, 334
collisions
  detecting, 361-362
  EnergyObject class, 333-334
  pruning, 358-359
color values, ES1Renderer, 66
  positioning, 67-68
common, 173
  parsing, 174
Compiling for Thumb, performance, 389-390
components, 7
  doors, 9
  entities, 10
  health, 8
  lives, 7-8
  objects, 8
    energy items, 9
    keys, 9
    parchment pieces, 9
  players, 11
  time, 7
  weapons, 10
configuration, particle emitters, 230-231
configuring, view ports, 85-86
control files, 17
  parsing
    C, 172-174
    with PackedSpriteSheet class, 148-149
controlling music, SoundManager, 265-266
copyImageDetails method, 121
Core Animation, 321
CPU spike, OpenGL ES instrument, 377
createLayerTileImage: method, 216

D

dealloc method, 162, 181
deallocation, rendering text, 181
delegates. *See* application delegates
design patterns, 312
detecting, collisions, 361-362
direction, animation, 155
directories, 353-354
doors, 9, 338
drawing maps, tile maps, 192
dropInventoryFromSlot: method, 351-352
duplicating images, Image class, 130

E

EAGLView, 49, 290
EAGLView class, 79
EAGLView.h, 49-50
EAGLView.m, 50-58, 79-82
encodeWithCoder: method, 354-355
energy items, 9

EnergyObject class, 329
- collisions, 333–334
- initialization, 329–332
- rendering, 333
- updating, 332–333

entities, 10

entity-to-entity collision detection, 367-368

entity-to-map collision detection, 365-366

ES1Renderer, 58
- color values, 66
  - positioning, 67–68
- framebuffer, creating, 60–66
- game loops, 82–85
- render method, 63–66
- renderbuffer, creating, 60–66

ES1Renderer.h, 58-59

ES1Renderer.m, 59-60

ES2Renderer, 52

examining
- header files, 44–46
- implementation files, 46–49

F

fadeImage, 303

fading music, SoundManager, 266-268

feedback, beta testing, 392

fonts, bitmap fonts, 19-20. *See* bitmap fonts

FPS (Frames Per Second), 74

frame-based collision detection, versus time-based, 359-360

frame-based game loops, 75-76

framebuffer, ES1Renderer, 60-66

frames
- adding with Animation class, 157–158
- animation, 154–155

Frames Per Second (FPS), 74

G

game controllers, 79, 86-87
- creating, 87

game entities, 325, 338-339
- AbstractEntity class, 339–341
- artificial intelligence (AI), 341–343
- Player class, 343–344
  - initialization, 344
  - inventory, 350–352
  - updating, 344–346
  - updating player's location, 346–350
- projects, 325–326
- saving, 352–355

game interfaces
- OpenGL ES interfaces. *See* OpenGL ES interfaces
- projects, 299–300

game loops, 24-26, 75
- collision detection, 74–75
- ES1Renderer class, 82–85
- frame-based, 75–76
- time-based, fixed interval, 77–78
- timing, 73–74
- view ports, configuring, 85–86

game objects, 325-326
- AbstractObject class, 327–329
- EnergyObject class, 329
  - collisions, 333–334
  - initialization, 329–332
  - rendering, 333
  - updating, 332–333
- location, 336
- naming, 336
- projects, 325–326
- saving, 352–355
- tile maps and, 334–337

game scenes, 79, 86-87
AbstractScene, 92–93
game settings
loading, 376–377
saving, 375–376
game state
loading, 373–375
saving, 371–373
GameController class, 87-89, 308
GameController.m, 89-91
games
for the iPhone, special considerations for, 4–5
Manic Miner, 3
naming, 5–6
objectives, 6–7
GameScene class, 93-95
generating
image data, Texture2D, 111–112
texture names, Texture2D, 112–113
getters, 134
getTileCoordsForBoundingRect method, 341
GL_LINE_LOOP, 98
GL_LINE_STRIP, 98
GL_LINEAR, 114
GL_LINES, 99
GL_NEAREST, 114
GL_POINTS, 98
GL_TRIANGLE_FAN, 99
GL_TRIANGLE_STRIP, 98
OpenGL ES, 99
GL_TRIANGLES, 98-100
glDrawArrays, 71, 246
glEnableClientState, 70
glGenBuffers, 274
glGenTextures, 113
GlobalTileID, 199
glTransferlatef, 69-70
glTranslate, 370
glVertexPointer, 70
GPUs (Graphics Processing Units), 30
group headers, 79

H

header files
BitmapFont class, 170–171
examining, 44–46
health, 8
height, text, 180-181
hiding UIKit interfaces, 320-322
Hiero, 36-37, 166
high score list, adding scores to, 379-380
high scores
loading, 381–382
saving, 377–381

I

IBAction keyword, 317
IBAction methods, 317
IBOutlet, 315-316
ideas for games, 4
Image class, 126
images
duplicating, 130
rendering, 130–133
initialization, 126–129
sub-images, creating, 129–130
image data, loading into OpenGL texture, 114-116
image rendering, classes, 97, 107-108
Image class. *See* Image class

ImageRenderManager. *See* ImageRenderManager
Texture2D. *See* Texture2D
TextureManager. *See* TextureManager
ImageRenderManager, 119
images
adding to render queue, 120–123
rendering, 123–126
initialization, 119–120
images
adding to render queue, ImageRenderManager, 120–123
duplicating in Image class, 130
generating data in Texture2D, 111–112
loading, in Texture2D, 108–109
rendering
with Image class, 130–133
ImageRenderManager, 123–126
sizing in Texture2D, 109–111
implementation files, examining, 46-49
initialization
Animation class, 156–157
bitmap fonts, Cocoa Touch, 171–172
EnergyObject class, 329–332
Image class, 126–129
ImageRenderManager, 119–120
Layer class, tile map classes, 196–197
PackedSpriteSheet class, 147–148
ParticleEmitter class, 234–235
Player class, 344
SoundManager, 258–262
SpriteSheet class, 143–146
Texture2D, 108
TextureManager, 117
TiledMap class, 205–207
TileSet class, tile map classes, 202–203
initWithCoder: method, 51, 355
initWithTileLocation: method, 329
Instruments, 35, 382-384
Leaks Instrument, 384–387
OpenGL ES, 387–389
Interface Builder, 32-33, 315
interfaceOrientation, 320
Interleaved Vertex Arrays (IVA), 104-106
interruptions, sound, 281-283
inventory, Player class, 350-352
iOS, 256
iPhone SDK, 32
Hiero, 36–37
Instruments, 35
Interface Builder, 32–33
iPhone Simulator, 34–35
Shark, 36
Tiled, 37
Xcode, 32
iPhone Simulator, 34-35
iPhones, sound, 250
audio sessions, 250–251
creating sound effects, 254–256
playing music, 252
playing sound effects, 252–253
stereo versus mono, 256
isBlocked:y: method, 366
IVA (Interleaved Vertex Arrays), 104-106
structures, 106–107

J

justification values, 179-180
justified text, rendering, 178-180

K

keys, 9

keywords, IBAction, 317

kFadeInterval, 267

L

Layer class, tile map classes, 196-197
- adding tile images to layers, 199-200
- adding tiles to layers, 198-199
- getting and setting tile information, 201-202
- initialization, 197-198

layer elements
- parsing, 212-216
- tiled configuration file, 194-195

layer images, creating, 216-217

layerClass method, 51

layers
- rendering, 218-219
- tile maps, 188-189

Leaks Instrument, 384-387

life cycle of particles
- birth of particles, 227-228
- death, 229
- lives of, 228-229
- rebirth, 229-230

linking, IBAction, 317

listener positions, sound effects, 281

listings
- AbstractEntity encodeWithCoder: Method, 354-355
- AbstractEntity Methods, 340
- AbstractEntity Properties in AbstractEntity.h, 339-340
- AbstractObject Properties, 338
- Action Methods Defined in SettingeViewController.h, 317
- The addFrameWithImage:delay: Method, 158
- Adjusting the Image Size to 1024x1024, 110-111
- Animation init Method, 156
- BitmapFont parseCharacterDefinition: Method, 175
- BitmapFont parseCommon: Method, 174
- BitmapFont parseFont:controlFile: Method, 172-173
- BitmapFont renderStringAt:text: Method, 176
- BitmapFont renderStringJustifiedInFrame: justification:text: Method, 178-179
- BitmapFonts initWithFontImageNamed: controlFile: Method, 171
- CGRect Variables Defined in MenuScene.h, 304
- CH03_SLQTSORAppDelegate.h, 44
- CH03_SLQTSORAppDelegation.m, 46-47
- Checking The Bounds of the Start Button, 305
- Circle-to-Circle Collision Detection Function, 362
- Circle-to-Rectangle Collision Detection Function, 361-362
- Code to Render Interface Element Boundaries Inside MainMenu.m, 307
- Code Used to Convert RGP565 Image Data from 32-16-Bits, 112
- Complete Layer Element within a .tmx File, 194
- Complete objectgroup Element Within a .tmx File, 195
- A Complete Tileset Element within a .tmx File, 194

Configure the Bitmap Context for Rendering the Texture, 111

EAGLView gameLoop: Method, 80

EAGLView render Method, 83

EAGLView Touch Methods, 290

EnergyGame object checkForCollisionWithEntity: Method, 333-334

EnergyObject collisionBounds Method, 334

EnergyObject initWithTileLocation: Method, 330-331

EnergyObject render Method, 333

EnergyObject updateWithDelta: Method, 332

Entity State enums Defined in Global.h, 340

Entity-to-Map Collision Check, 366

ES1Renderer orientationChanged Method, 310

Example Tile Elements with Properties, 211

Excerpt from the Bitmap Font Control File, 173

The Game Loop, 24-25

Game Object Type and Subtype enums in Global.h, 327-328

GameController addToHighScores:gameTime:playersName:didWin: method, 379

GameController adjustTouchOrientationForTouch: Method, 311

GameController loadHighScores Method, 381-382

GameController loadSettings Method, 376

GameController saveHighScores Method, 381

GameController saveSettings Method, 375

GameController sortHighScores Method, 380

GameScene accelerometer:didAccelerate: Method, 296

GameScene alertView:clickedButtonAtIndex: Method, 378

GameScene checkJoypadSettings Method, 322

GameScene initCollisionMapAndDoors: Method, 363-364

GameScene isBlocked:y: Method, 365

GameScene loadGameState: Method, 374-375

GameScene saveGameState: Method, 372-373

GameScene saveGameState Method NSKeyedArchiver Creation Snippet, 353

GameScene touchesBegan:withEvent: view: Method, 291

GameScene touchesBegan:withEvent: view Method Handling Taps, 294-295

GameScene touchesEnded:withEvent: view Method, 294

GameScene touchesMoved:withEvent: view Method, 292-293

GameScene updateSceneWithDelta: Method, 297

GameScene updateSceneWithDelta: Method—Object Update, 358-359

GameScene's renderScene Method Positions the Player in the Middle of the Screen, 369

GameScene's renderScene Method (Tile Map Rendering), 370

The getHeightForString: Method, 180-181

Getters and Setters, 134
The getWidthForString: Method, 180
Ghost checkforCollisionWithEntity: Method, 367
The ImageDetails Structure, 107
The imageDuplicate Method, 130
ImageRenderManager addImageDetailsToRenderQueue: Method, 121
ImageRenderManager addToTextureList: Method, 122
ImageRenderManager init Method, 120
ImageRenderManager initializeImageDetails Method, 128
ImageRenderManager renderImages Method, 123-124
ImageRenderManager subImageInRect: Method, 130
Initialization of Button Bounds, 304
Layer addTileAt:tileSetID:tileID Method, 198
Layer addTileImageAt: Method, 200
Layer getGlobalTileIDAtX: Method, 201
Layer initWithName: Method, 197
Layer setValueAtX: Method, 201
Layer tileImageAt: Method, 202
MenuScene renderScene Method, 302
PackedSpriteSheet imageForKey Method, 149-150
PackedSpriteSheet initWithImageNamed:controlFile: filter Method, 147-148
PackedSpriteSheet parseControlFile: Method, 148
Particle Emitter XML Configuration File, 230-231
Particle Structure, 234
ParticleEmitter addParticle Method Exert, 244
ParticleEmitter initParticleEmitterWithFile: Method, 235
ParticleEmitter parseParticleConfig: Partial Method (Part 1), 235
ParticleEmitter parseParticleConfig: Partial Method (Part 2), 236
ParticleEmitter renderParticles Method (Part 1), 244-245
ParticleEmitter renderParticles Method (Part 2), 245-246
ParticleEmitter setupArrays Method, 237
ParticleEmitter stopParticleEmitter Method, 246
ParticleEmitter updateWithDelta Method (Part 1), 239
ParticleEmitter updateWithDelta Method (Part 2), 240
ParticleEmitter updateWithDelta Method (Part 3), 242-243
Player checkForCollisionWithEntity: Method, 367-368
Player Class Properties, 343-344
Player dropInventoryFromSlot: Method, 351-352
Player placeInInventory: Method, 350-351
Player updateLocationWithDelta: Method (Part 1), 347
Player updateLocationWithDelta: (Part 2), 349
Player updateWithDelta: Method (Part 1), 344-345
Player updateWithDelta: Method (Part 2), 346
PointSprite Structure, 234
Primitives drawBox Function, 306-307

The render Method, 131-132
The Render Methods Within Animation.m, 160-161
The renderCenteredAtPoint:scale: rotation: Method, 131
Rendering the Texture Image, 111
Scene State as Defined in the Global.h File, 303
Setting the Image Size in Texture2D to a Power-of-Two, 110
SettingsViewController class, 318
SettingsViewController hide Method, 321
SettingsViewController shouldAutorotateToInterfaceOrientat ion: Method, 318-319
SettingsViewController show Method, 320
SettingsViewController.h IBOutlets, 315
SettingsViewControllerviewWill Appear: Method, 319
A Simple Game Loop, 73
SoundManager addToPlayListName: track: Method, 268-269
SoundManager audioPlayerDidFinish Playing:successfully: Method, 270
SoundManager AVAudioSession Delegate Methods, 282
SoundManager fadeMusicVolume From: toVolumen: duration: stop: Method, 267
SoundManager fadeVolume Method, 267-268
SoundManager init Method (Part 1), 258
SoundManager init Method (Part 2), 258-259
SoundManager init Method (Part 3), 259-260
SoundManager initOpenAL Method (Part 1), 260
SoundManager initOpenAL Method (Part 2), 261
SoundManager initOpenAL Method (Part 3), 262
SoundManager isExternalAudio Playing Method, 259
SoundManager loadMusicWithKey: musicFile Method, 263
SoundManager loadSoundWithKey: soundFile: Method, 275-276
SoundManager loadSoundWithKey: soundFile: Method (Part 1), 274
SoundManager nextAvailableSource Method, 277-278
SoundManager playMusicWithKey:timesToRepeat: Method, 264-265
SoundManager playNextTrack Method, 271
SoundManager playSoundWithKey:gain:pitch: location:shouldLoop: Method (Part 1), 277
SoundManager playSoundWithKey: gain:pitch:location:shouldLoop: Method (Part 2), 278
SoundManager removeFromPlaylist Named:track:, 271-272
SoundManager removeMusicWith Key: Method, 264
SoundManager removePlaylistNamed: and clearPlaylistNamed: Method, 272
SoundManager setActivate: Method, 282-283
SoundManager setListenerLocation and setOrientation Methods, 281
SoundManager startPlaylistNamed: Method, 269-270

SoundManager stopMusic, pauseMusic, resumeMusic, and setMusicVolume: Methods, 265-266
SoundManager stopSoundWithKey: Method, 279-280
SpriteSheet cacheSprites Method, 145-146
SpriteSheet InitWithImageNamed: spriteSize:spacing:margin:imageFilter Method, 144
SpriteSheet initWithImage:spriteSize: spacing:margin Method, 145
SpriteSheet spriteImageAtCoords: Method, 146-147
The spriteSheetForImageName: spriteSize:spacing:margin:image Filter: Method, 143
Structure of BitmapFontchar, 170
TBXMLParticleAdditions Header File, 232
Texture and Ratio Calculations, 115
The TexturedColoredQuad Structure, 107
The TexturedColoredVertex Structure, 106
TextureManage textureWithFile Name:filter: Method, 117-118
TiledMap createLayerTileImages: Method, 217
TiledMap initWithFileName: fileExtension Method (Part 1), 206
TiledMap initWithFileName: fileExtension Method (Part 2), 206
TiledMap parseMapFileTBXML: Method (Part 1), 207-208

TiledMap parseMapFileTBXML: Method (Part 2), 208

TiledMap parseMapFileTBXML: Method (Part 3), 209
TiledMap parseMapFileTBXML: Method (Part 4), 210
TiledMap parseMapFileTBXML: Method (Part 5), 211
TiledMap parseMapFileTBXML: Method (Part 6), 212
TiledMap parseMapFileTBXML: Method (Part 7),213
TiledMap parseMapFileTBXML: Method (Part 8), 214
TiledMap parseMapFileTBXML: Method (Part 9), 215
TiledMap renderLayer:mapx:mapy:width:heigh t:useBlending Method, 218-219
TileSet's initWithImageNamed: Method, 203
TXMLParticleAdditions color4fFrom ChildElementNamed:parentElement Method, 232-233
The updateWithDelta: Method, 159
Witch Chase Code in the update WithDelta:scene: Method, 342

lives, 7-8

loadGameState: method, 373-375

loadHighScores method, 381-382

loading

game settings, 376-377
game state, 373-375
high scores, 381-382
image data into OpenGL texture, into OpenGL texture, 114-116
images, Texture2D, 108-109
music, SoundManager, 263-264
sound effects, 274-276

loadSettings method, 376

location

of game objects, 336
of players, updating, 346-350

M

managing
- playlists, 271-272
- sound effects, 273

Manhattan distance, 293

Manic Miner, 3

map element, tile configuration file, 193

map elements
- parsing, 207-209
- tiled configuration file, 193

map files
- parsing, 207
  - map elements, 207-209

map layers, creating, 191

maps
- collision maps, 362-365
- drawing, 192
  - tile maps, 192

message nesting, 48

motion events, 287

multiple device types, beta testing, 391

multiple touches, 288

music
- controlling in SoundManager, 265-266
- fading in SoundManager, 266-268
- loading with SoundManager, 263-264
- playing in SoundManager, 264-265
- removing in SoundManager, 264

music management, Sound, 262-263

music playlists, SoundManager, 268-271

musicPlaylists dictionary, 258, 269

N

naming
- game objects, 336
- games, 5-6

nextAvailableSource method, 278

nonatomic property, 50

notifications, 308

NSDataAdditions class, 205

NSKeyedArchiver, 352

NSNotificationCenter, 308

NSUserDefaults, 375

O

OBB (Oriented Bounding Boxes), 361

object group element
- tiled configuration file, 195-196

object group elements, parsing, 216

object layers
- adding to maps, 335
- creating, 191-192

objectgroups, 196

Objective-C, 26
- @property, 46
- application delegates, 44

Objective-C 2.0, 162

objectives, games, 6-7

objects
- components, 8
  - energy items, 9
  - keys, 9
  - parchment pieces, 9
- placing, in tile maps, 192

OES, 245

Open GL texture, loading image data into, 114-116

OpenAL, 30-31, 253

OpenGL, 68-69
applying transformations on models, 69-70
axis configuration, 64
rendering to screens, 70-72
OpenGL ES, 15, 27-29
XXXX1.1 versus 2.029-30
GL_TRIANGLE_STRIP, 99
instrument, 387-389
OpenGL ES interfaces, 300-301
defining button bounds, 304
handling touches, 304-308
rendering, 301-303
transitions, 308
OpenGL ES orientation, 308
manually setting, 309-311
orientation
OpenGL ES, 308
manually setting, 309-311
UIKit interfaces, 318-320
orientationChanged method, 310
Oriented Bounding Boxes (OBB), 361

P

PackedSpriteSheet class, 147
bitmap fonts, 167
initialization, 147-148
parsing control files, 148-149
sprites, retrieving, 149-150
parameters, particle systems, 226-227
parchment pieces, 9
parseParticleConfig method, 235
parsing
char id, 175-176
common prefix, 174
control files
C, 172-174
PackedSpriteSheet class, 148-149
layer elements, 212-216
map files, 207
map elements, 207-209
object group elements, 216
particle configuration, 235-237
tile set elements, 209-212
particle arrays, 237-238
particle configuration, parsing, 235-237
Particle Designer, 247
particle emitters
configuration, 230-231
playing with, 247-248
projects, 224-225
stopping, 246-247
particle systems, 21-22
overview, 225-226
parameters, 226-227
ParticleEmitter class, 233
initialization, 234-235
parsing particle configuration, 235-237
particles
adding, 243-244
rendering, 244-246
updating, 239-243
setting up particle and render arrays, 237-238
stopping particle emitters, 246-247
structures, 233-234
ParticleEmitter classes, 231
TBXMLParticleAdditons class, 231-233

particles, 21
- adding in ParticleEmitter class, 243–244
- life cycle of
  - birth of particles, 227–228
  - death, 229
  - lives of, 228–229
  - rebirth, 229–230
- rendering in ParticleEmitter class, 244–246
- updating in ParticleEmitter class, 239–243

performance, Compiling for Thumb, 389-390

phases, tracking touches between, 292

pixelLocation, 327

placeholders, prototyping, 19

placeInInventory: method, 350-351

placing objects, tile maps, 192

Player class, 343-344
- initialization, 344
- inventory, 350–352
- updating, 344–346
- updating player's location, 346–350

players, 11

playing
- music
  - iPhones, 252
  - SoundManager, 264–265
- sound effects, 252–253, 276–279

playlists
- managing, 271–272
- starting, 270
- tracks, removing, 272

playNextTrack: method, 271

point sprites, 238

Portal class, 339

positioning, color values, ES1Renderer, 67-68

prefixes, 173-174
- char id, parsing, 175–176
- common, parsing, 174

processing
- taps, 294–295
- touch events, 289–290
  - touchesBegan phase, 290–292
  - touchesEnded phase, 294
  - touchesMoved phase, 292–294

projects
- creating with Xcode, 39–42
- running, 42–43

@property, 46

prototyping, with placeholders, 19

pruning collisions, 358-359

Q

quads, rendering, 98-101

R

releaseAllTextures, 118

releasing, textures with TextureManager, 118

removePlaylistNamed: method, 272

removing
- music, SoundManager, 264
- tracks from playlists, 272

render arrays, ParticleEmitter class, 237-238

render method, 63-66, 329

render queue, adding images to, ImageRenderManager, 120-123

renderbuffer, ES1Renderer, 60-66

rendering, 97-98
- animation, Animation class, 160–161
- EnergyObject class, 333
- images
  - Image class, 130–133
  - ImageRenderManager, 123–126

justified text, 178-180
layers, 218-219
OpenGL ES interfaces, 301-303
particles, ParticleEmitter class, 244-246
quads, 98-101
to screens, OpenGL, 70-72
text, 176-178
deallocation, 181
width and height, 180-181

renderLayer:mapx:mapy:width:height: useBlending: method, 218

renderScene method, 303, 369-370

retrieving
sprites
with PackedSpriteSheet class, 149-150
SpriteSheet class, 146-147
textures, TextureManager, 117-118

rotation, animation, 161

rotationPoint, animation, 161

running, projects, 42-43

S

saveGameState, 372-373

saveHighScores method, 381

saveSettings method, 375

saving, 371
game entities, 352-355
game objects, 352-355
game settings, 375-376
game state, 371-373
high scores, 377-381

scores, adding to high score list, 379-380

screens, rendering to screens, OpenGL, 70-72

setListenerPosition method, 281

setters, 134

setting parameters, binding with texture parameters, 113-114

Shark, 36

showing, UIKit interfaces, 320-322

singletons, creating, 89

sizing, images, in Texture2D, 109-111

Smith, Matthew, 3

sortHighScores method, 380

Sound, music management, 262-263

sound
handling interruptions, 281-283
iPhones, 250
audio sessions, 250-251
creating sound effects, 254-256
playing music, 252
playing sound effects, 252-253
stereo versus mono, 256
projects, 249

sound effects
creating, 254-256
listener positions, 281
loading, 274-276
managing, 273
playing, 252-253, 276-279
stopping, 279-280

sound manager classes, 256-257
MyOpenALSupport, 256-257
SoundManager, 257
controlling music, 265-266
fading music, 266-268
initialization, 258-262
loading music, 263-264
managing playlists, 271-272
music management, 262-263
music playlists, 268-271
playing music, 264-265
removing music, 264

soundLibrary dictionary, 258

SoundManager, 257

controlling music, 265–266

fading music, 266–268

initialization, 258–262

loading music, 263–264

managing playlists, 271–272

music playlists, 268–271

playing music, 264–265

removing music, 264

SoundManagerAVAudioPlayer, 257

special considerations, for iPhone games, 4-5

speed, update speed, 74

speedOfMovement, 347

sprite sheets, 16-18, 137-138

bitmap fonts, creating, 167–170

complex sprite sheets, 139–141

simple sprite sheets, 138–139

Zwoptex, 141–142

sprites, 14-15

creating, 15

retrieving

with PackedSpriteSheet class, 149–150

with SpriteSheet class, 146–147

SpriteSheet class, 142

initialization, 143–146

sprites, retrieving, 146–147

startAnimation, 56-57

starting playlists, 270

states, animation, 155

stereo versus mono, sound on iPhones, 256

stopping

particle emitters, 246–247

sound effects, 279–280

storylines, 5

structures

IVA (Interleaved Vertex Arrays), 106–107

ParticleEmitter class, 233–234

sub-images, creating with Image class, 129-130

T

taps, processing, 294-295

TBXML, 204

TBXMLParticleAdditions class, 231-233

technology

Cocoa Touch, 27

Objective-C, 26

OpenAL, 30–31

OpenGL ES, 27–29

terminology, 13

AI (artificial intelligence), 23

animation, 18–19

bitmap fonts, 19–20

collision detection, 22–23

game loops, 24–26

particle systems, 21–22

sprite sheets, 16–18

sprites, 14–15

time maps, 20–21

testing, beta testing, 390-391

feedback, 392

multiple device types, 391

text

justified text, rendering, 178–180

rendering, 176–178

deallocation, 181

width and height, 180–181

texture coordinates, texture mapping, 101-103

texture mapping, 101
texture coordinates, 101–103
texture names, generating in Texture2D, 112-113
texture parameters, binding with setting parameters, 113-114
Texture2D, 107, 108
binding texture and setting parameters, 113–114
generating texture names, 112–113
images
generating data, 111–112
loading, 108–109
sizing, 109–111
initialization, 108
loading image data into OpenGL texture, 114–116
TexturedColoredQuad, 200
TextureManager, 116-117
initialization, 117
textures
releasing, 118
retrieving and creating, 117–118
textures
releasing with TextureManager, 118
retrieving and creating, with TextureManager, 117–118
texturesToRender array, 122
textureWithFileName:filter: method, 117-118
tile configuration file, 193
layer elements, 194–195
map element, 193
object group element, 195–196
tileset element, 193–194
tile images, adding to layers, 199-200
tile information, getting, 220
tile map classes, 196
Layer class, 196–197
adding tile images to layers, 199–200
adding tiles to layers, 198–199
getting and setting tile information, 201–202
initialization, 197–198
TiledMap class, 204–205
initialization, 205–207
TileSet class, 202
getting tile set information, 203–208
initialization, 202–203
tile maps, 183, 184-188, 370
creating, 189
drawing maps, 192
map layers, 191
new tile sets, 190–191
object layers, 191–192
drawing maps, 192
game objects and, 334–337
layers, 188–189
placing objects, 192
projects, 183–184
rendering layers, 218–219
tile information, getting, 220
tile palette, Tiled, 188
tile set elements, parsing, 209-212
tile sets, creating, 190-191
Tiled, 37, 187
tile palette, 188
tiled configuration file, 193
layer elements, 194–195
map elements, 193
object group element, 195–196
tileset element, 193–194
Tiled map editor, game objects, 334

Tiled Qt, 186

TiledMap class

- tile map classes, 204–205
  - initialization, 205–207
- XML parsing, 204

TileID, 199

tileImageAt: method, 201

tileLocation, 327

tiles, adding to layers, 198-199

TileSet class, tile map classes, 202

- getting tile set information, 203–208
- initialization, 202–203

tileset element, 193-194

TileSetID:Identifies, 199

time, 7

time maps, 20-21

time-based collision detection, versus frame-based, 359-360

time-based fixed interval game loops, 77-78

timing

- collision detection, 74–75
- game loops, 73–74

tools, 31-32

- iPhone SDK, 32
  - Hiero, 36–37
  - Instruments, 35
  - Interface Builder, 32–33
  - iPhone Simulator, 34–35
  - Shark, 36
  - Tiled, 37
  - Xcode, 32

touch events, 287-289

- processing, 289–290
  - touchesBegan phase, 290–292
  - touchesEnded phase, 294
  - touchesMoved phase, 292–294
- taps, processing, 294–295

touches, 285

- OpenGL ES interfaces, 304–308
- tracking between phases, 292

touchesBegan phase, 290-292

touchesEnded: method, 304

touchesEnded phase, 294

touchesMoved phase, 292-294

tracking touches, between phases, 292

tracks, removing from playlists, 272

transformation, 67

transformations, applying to models, 69-70

transitions, OpenGL ES interfaces, 308

translating, 370

types, animation, 155

U

UDID, 390

UIAccelerometerDelegate, 88

UIApplication, 43

UIInterfaceOrientationIsLandscape macro, 319

UIKit interfaces, 312

- creating, 312–315
- showing/hiding, 320–322
- wiring up interfaces, 315–318

UIKit orientation, 318-320

UISlider controls, 314

UISlider track, 314-315

UITouch objects, 288

UIView, 55

update speed, 74

updateLocationWithDelta: method, 348-349

updates, animation, Animation class, 158-160

updateWithDelta: method, 303, 332, 344-345

updateWithDelta:scene: method, 341-342

updating
  EnergyObject class, 332–333
  particles, ParticleEmitter class, 239–243
  Player class, 344–346
user input, 287
  accelerometer events, 296–298
  projects, 285–286
  taps, 294–295
  touch events, 287–289
    processing, 289–290
userDefaultsSet, 377

V

vertex arrays, 104
vertical synchronization, 52
view ports, configuring, 85-86
Viewport function, 85
visualizing boundaries, 306-308
vsync, 52

W

weapons, 10
width, text, 180-181
wiring up interfaces, UIKit interfaces, 315-318

X

Xcode, 32
  projects, creating, 39–42
XML parsing, TiledMap class, 204

Y-Z

Zwoptex, 17, 138, 140-141
  sprite sheets, 141–142