



Quick answers to common problems

Cocos2d-x Cookbook

Over 50 hands-on recipes to help you efficiently develop, administer, and maintain your games with Cocos2d-x

Akihiro Matsuura

[PACKT] open source*
PUBLISHING community experience distilled

www.allitebooks.com

Cocos2d-x Cookbook

Over 50 hands-on recipes to help you efficiently develop, administer, and maintain your games with Cocos2d-x

Akihiro Matsuura

[PACKT] open source 
PUBLISHING community experience distilled

BIRMINGHAM - MUMBAI

Cocos2d-x Cookbook

Copyright © 2015 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: October 2015

Production reference: 1261015

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78328-475-7

www.packtpub.com

Credits

Author

Akihiro Matsuura

Project Coordinator

Izzat Contractor

Reviewers

Luma

Pranav Pahlaria

Sergio Martínez-Losa Del
Rincón

Vamsi Krishna Veligatla

Chatchai Mark Wangwiwattana

Proofreader

Safis Editing

Indexer

Tejal Daruwale Soni

Graphics

Jason Monteiro

Acquisition Editor

Kevin Colaco

Production Coordinator

Aparna Bhagat

Content Development Editor

Priyanka Mehta

Cover Work

Aparna Bhagat

Technical Editor

Ryan Kochery

Copy Editors

Imon Biswas

Tani Kothari

About the Author

Akihiro Matsuura has five years of experience as a Cocos2d-x developer. He founded his own company called Syuhari, Inc. five years ago. He has more than 20 years of experience as a programmer. He has written three technical books in Japanese. He also authored *Cocos2d-x Recipe Book*, Shuwa System Co., Ltd, the first Cocos2d-x book in Japan, *iPhone SDK Recipe Book*, Shuwa System Co., Ltd, and *Cocos2d-x Guide Book*, Mynavi Corporation.

7 years ago, iPhone3G was released in Japan. This was when he began to develop its software and ended up developing a lot of applications for iPhone. First, he developed them using Cocos2d for iPhone; however, he had to port to Android. So, he decided to use Cocos2d-x to develop cross-platform applications. At that time, Cocos2d-x was at version 1.0.

Since then, he has developed a lot of applications using Cocos2d-x.

Firstly, I would like to thank Packt Publishing for giving me the opportunity to write this book, especially Priyanka Mehta, Anish Sukumaran, and Kevin Colaco for helping me to improve this book's quality.

The biggest thank you goes to the Cocos2d-x development team. I think Cocos2d-x is a really great game framework. I really love Cocos2d-x.

Finally, a special thank you goes to my wife, Noriko, and daughters, Miu and Yui.

About the Reviewers

Luma has several years of experience on iOS and Android. He focuses on game development on mobile platforms. He is the creator of WiEngine, cocos2dx-better, and cocos2dx-classical. His Github page is <https://github.com/stubma>.

Pranav Paharia is a game developer who makes games for mobile platforms and PCs. He has experience in working on numerous game technologies, such as Unity3D, Cocos2dx, Unreal Engine 4, Construct2, RPG Maker, and so on. He has also worked on many genres such as platformers, infinite runners, RPGs, casual games, turn-based games, point and click games, multiplayer action games, and so on.

After finishing his graduation in information technology, he took the decision of turning his passion into profession and pursued a course in videogame development. Exploring the diversity and depth in emerging game technologies, he worked with various multicultural teams, participating in game jams and working on his personal experiments. He faced every challenge with a "never give up" attitude. Gradually, his hard work and constant commitment led him to Nasscom GDC in 2013 for the game "Song of Swords" with his team winning the "People's Choice of the Year" award. Later in his career timeline, another game *Chotta Bheem Laddoo Runner* entered the limelight in India, being the most popular game among Indian kids. He has also reviewed Packtpub's *Mastering Cocos2d Cross-Platform Game Development* and *Unity 2D Game Development Cookbook*.

Since childhood, he has been in love with computer games. Growing up as the first generation of gamers, he was consumed by Mario, Dave Contra, and other 8-bit pixel art games. Being a lefty, he is creative at heart. His endless curiosity set him on a contrasting journey, from the sciences to a myriad of art forms. Playing computer games, making pencil sketches, and reading books were a few of his many hobbies. By the end of his school days, he was dedicating a lot of his time to playing Counter Strike. In the junior college and graduation era, he competed in many gaming tournaments, and succeeded in making his team, the best team in his college.

After entering into game development, he became fascinated with the other side of the coin, that is, the science of creating great games. For him, a game is a form of art, which stimulates the player's psychology and thereby raising various kind of emotions in him. Manipulating these emotions using colors and code seems like wizardry to him. His main aim in life is to create great games that can stimulate a positive transformation in people. He invests his free time in photography, writing, and the philosophical reasoning behind life. You can find him at pranavpaharia@gmail.com.

I would sincerely like to pay homage and thanks to my parents and to my brother, Nikhil, for their cooperation and motivation while working on this book and to beloved Krishna for giving me the secret knowledge of knowing my inner passion, which has led me to this journey of becoming a Game Wizard.

Sergio Martínez-Losa Del Rincón lives in Spain. He is a software engineer and a serial entrepreneur. He likes to write technical documents as well as programming in several languages.

He is always learning new programming languages and facing new challenges. Currently, he is creating applications and games for iPhone, Macintosh, Android, GoogleGlass, and Ouya. You can see part of his work at <http://goo.gl/k5tOSX>

Vamsi Krishna Veligatla is the director of engineering at Hike Messenger Pvt Ltd. He was the lead developer on some iconic games, such as *Shiva: The Time Bender* and *Dadi vs Jellies* developed at Tiny Mogul Games.

He has a master's degree in computer science from the International Institute of Information Technology, Hyderabad. Previously, he worked at Nvidia Graphics Pvt Ltd, AMD (ATI), and the University of Groningen, Netherlands.

He's also worked on *Cocos2d-x by Example: Beginner's Guide - Second Edition* as a reviewer

I would like to thank my family for their love and support.

Chatchai Mark Wangwiwattana is a game researcher and designer. His work is related to developing and designing computer games for changing human behavior and facilitating learning by utilizing behavioral psychology and artificial intelligence. To learn more about his work and publications, visit www.chatchaiwang.com.

He's also worked on *Cocos2d-x by Example: Beginner's Guide - Second Edition* as a reviewer

I would like to thank my family, professors, and friends for having faith in me and supporting me.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit www.PacktPub.com.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www2.packtpub.com/books/subscription/packtlib>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why Subscribe?

- ▶ Fully searchable across every book published by Packt
- ▶ Copy and paste, print, and bookmark content
- ▶ On demand and accessible via a web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface	v
Chapter 1: Getting Started with Cocos2d-x	1
Introduction	1
Setting up our Android Environment	2
Installing Cocos2d-x	5
Using the Cocos command	8
Building the project using Xcode	11
Building the project using Eclipse	13
Implementing multi-resolution support	19
Preparing your original game	21
Chapter 2: Creating Sprites	25
Introduction	25
Creating sprites	26
Getting the sprite's position and size	28
Manipulating sprites	30
Creating animations	34
Creating actions	37
Controlling actions	40
Calling functions with actions	44
Easing actions	46
Using a texture atlas	48
Using a batch node	53
Using 3D modals	55
Detecting collisions	57
Drawing a shape	59

Chapter 3: Working with Labels	65
Creating system font labels	65
Creating true type font labels	69
Creating bitmap font labels	71
Creating rich text	73
Chapter 4: Building Scenes and Layers	77
Introduction	77
Creating scenes	78
Transitioning between scenes	82
Transitioning scenes with effects	83
Making original transitions for replacing scenes	85
Making original transitions for popping scenes	91
Creating layers	93
Creating modal layers	94
Chapter 5: Creating GUIs	97
Introduction	97
Creating menus	98
Creating buttons	101
Creating checkboxes	103
Creating loading bars	106
Creating sliders	108
Creating text fields	111
Creating scroll views	113
Creating page views	115
Creating list views	116
Chapter 6: Playing Sounds	119
Playing background music	119
Playing a sound effect	121
Controlling volume, pitch, and balance	122
Pausing and resuming background music	123
Pausing and resuming sound effects	124
Playing background music and a sound effect by using AudioEngine	125
Playing movies	127
Chapter 7: Working with Resource Files	131
Selecting resource files	131
Managing resource files	133
Using SQLite	141
Using .xml files	145

Using .plist files	147
Using .json files	149
Chapter 8: Working with Hardware	151
Introduction	151
Using native code	151
Changing the processing using the platform	156
Using the acceleration sensor	157
Keeping the screen on	158
Getting dpi	159
Getting the maximum texture size	160
Chapter 9: Controlling Physics	163
Introduction	163
Using the physics engine	163
Detecting collisions	166
Using joints	168
Changing gravity by using the acceleration sensor	174
Chapter 10: Improving Games with Extra Features	177
Introduction	177
Using Texture Packer	177
Using Tiled Map Editor	183
Getting the property of the object in the tiled map	191
Using Physics Editor	195
Using Glyph Designer	202
Chapter 11: Taking Advantages	207
Introduction	207
Using encrypted sprite sheets	207
Using encrypted zip files	210
Using encrypted SQLite files	213
Creating Observer Pattern	219
Networking with HTTP	223
Index	227

Preface

Cocos2d-x is a suite of open source, cross-platform game-development tools used by thousands of developers all over the world. Cocos2d-x is a game framework written in C++, with a thin platform-dependent layer. Completely written in C++, the core engine has the smallest footprint and the fastest speed of any other game engine, and is optimized to be run on all kinds of devices.

With this book, we aim to provide you with a detailed guide to create 2D games with Cocos2d-x from scratch. You will learn everything, from the fundamental stage, all the way up to an advanced level. We will help you successfully create games with Cocos2d-x.

What this book covers

Chapter 1, Getting Started with Cocos2d-x, covers the installation process for Cocos2d-x, also teaches you how to create a project, and talks about how to build a project for multi-platform.

Chapter 2, Creating Sprite, teaches you to create the sprites, animations and actions.

Chapter 3, Working with Labels, shows how to display the strings, and create labels.

Chapter 4, Building Scenes and Layers, teaches you to create scenes and layers, and how to change the scenes.

Chapter 5, Creating GUIs, talks about creating the GUI parts such as button and switches that are essential to a game.

Chapter 6, Playing Sounds, gives information on playing the background music and sound effects.

Chapter 7, Working with Resource files, teaches you how to manage the resource files, also talks about how to using the database.

Chapter 8, Working with the Hardware, guides you on how to access native features.

Chapter 9, Controlling Physics, tells you how to use physics on sprites.

Chapter 10, Improving Games with Extra Features, teaches you to use extra features on Cocos2d-x, and using various tools.

Chapter 11, Taking Advantage, talks about using practical tips on games, and improving the games.

What you need for this book

You will need a Mac that runs on OS X 10.10 Yosemite. Most of the tools that we will use throughout this book are free to download and try. We've explained how to download and install them.

Who this book is for

If you are a game developer and want to learn more about cross-platform game development in Cocos2d-x, then this book is for you. Knowledge of C++, Xcode, Eclipse, and how to use commands in the terminal are the prerequisites for this book.

Sections

In this book, you will find several headings that appear frequently (Getting ready, How to do it, How it works, There's more, and See also).

To give clear instructions on how to complete a recipe, we use these sections as follows:

Getting ready

This section tells you what to expect in the recipe, and describes how to set up any software or any preliminary settings required for the recipe.

How to do it...

This section contains the steps required to follow the recipe.

How it works...

This section usually consists of a detailed explanation of what happened in the previous section.

There's more...

This section consists of additional information about the recipe in order to make the reader more knowledgeable about the recipe.

See also

This section provides helpful links to other useful information for the recipe.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "If you need to compile a file other than the extension `.cpp` file."

A block of code is set as follows:

```

CPP_FILES := $(shell find $(LOCAL_PATH)/../../Classes -name *.cpp)
LOCAL_SRC_FILES := hellocpp/main.cpp
LOCAL_SRC_FILES += $(CPP_FILES:$(LOCAL_PATH)/%=%)


LOCAL_C_INCLUDES := $(shell find $(LOCAL_PATH)/../../Classes -type
d)


```

Any command-line input or output is written as follows:

```
$ ./build_native.py
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "You select **Android Application** and click on **OK**."

 Warnings or important notes appear in a box like this.

 Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail feedback@packtpub.com, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files from your account at <http://www.packtpub.com> for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from http://www.packtpub.com/sites/default/files/downloads/B00561_Graphics.pdf.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at questions@packtpub.com, and we will do our best to address the problem.

1

Getting Started with Cocos2d-x

In this chapter, we're going to install Cocos2d-x and set up the development environment. The following topics will be covered in this chapter:

- ▶ Setting up our Android environment
- ▶ Installing Cocos2d-x
- ▶ Using the Cocos command
- ▶ Building the project using Xcode
- ▶ Building the project using Eclipse
- ▶ Implementing multi-resolution support
- ▶ Preparing your original game

Introduction

Cocos2d-x is an open source, cross-platform game engine, which is free and mature. It can publish games for mobile devices and desktops, including iPhone, iPad, Android, Kindle, Windows, and Mac. Cocos2d-x is written in C++, so it can build on any platform. Cocos2d-x is open source written in C++, so we can feel free to read the game framework. Cocos2d-x is not a black box, and this proves to be a big advantage for us when we use it. Cocos2d-x version 3, which supports C++11, was only recently released. It also supports 3D and has an improved rendering performance. This book focuses on using version 3.4, which is the latest version of Cocos2d-x that was available at the time of writing this book. This book also focuses on iOS and Android development, and we'll be using Mac because we need it to develop iOS applications. This chapter explains how to set up Cocos2d-x.

Setting up our Android Environment

Getting ready

We begin by setting up our Android environment. If you wish to build only on iOS, you can skip this step. To follow this recipe, you will need some files.

The following list provides the prerequisites that need to be downloaded to set up Android:

- ▶ Eclipse ADT (Android Developer Tools) with the Android SDK:

https://dl.google.com/android/adt/adt-bundle-mac-x86_64-20140702.zip

Eclipse ADT includes the Android SDK and Eclipse IDE. This is the Android development tool that is used to develop Android applications. Android Studio is an Android development IDE, but it is not supported to build NDK. The official site states that a version of Android Studio that supports NDK will be released soon. That's why we use Eclipse in this book.

- ▶ Android NDK (Native Development Kit):

https://dl.google.com/android/ndk/android-ndk-r10c-darwin-x86_64.bin

The NDK is required to build an Android application. You have to use NDK r10c. This is because compiling and linking errors may occur when using NDK r9 or an earlier version.

- ▶ Apache ANT:

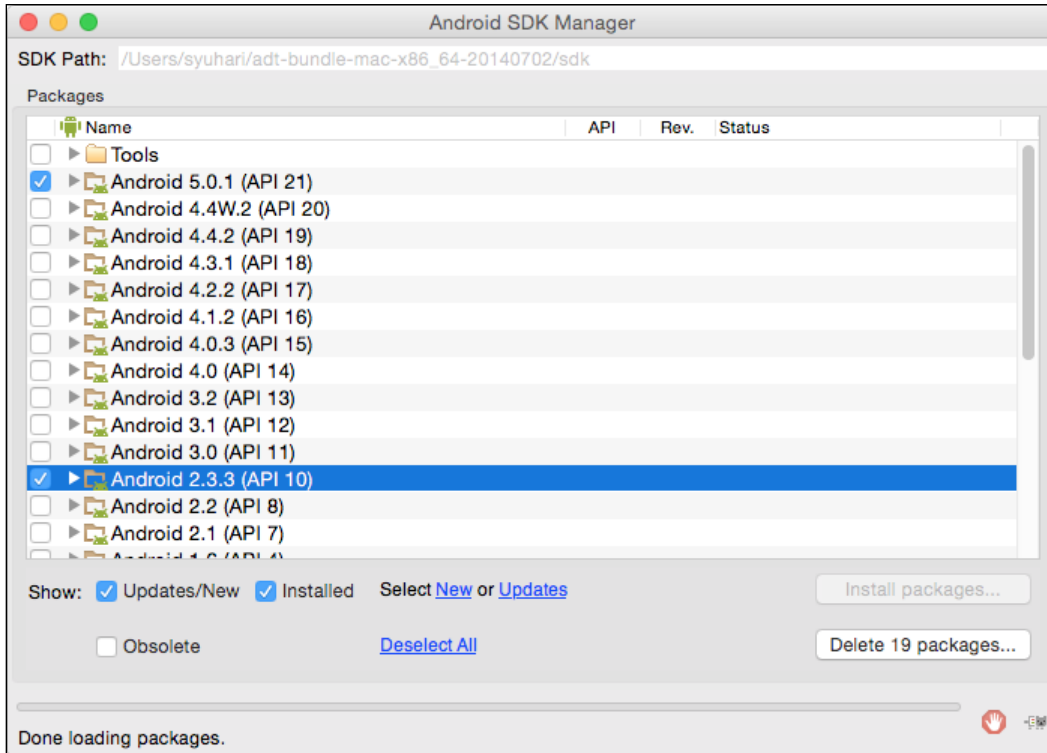
You can download Apache ANT from <http://ant.apache.org/bindownload.cgi>

This is a java library that aids in building software. At the time of writing this book, version 1.9.4 was the latest stable version available.

How to do it...

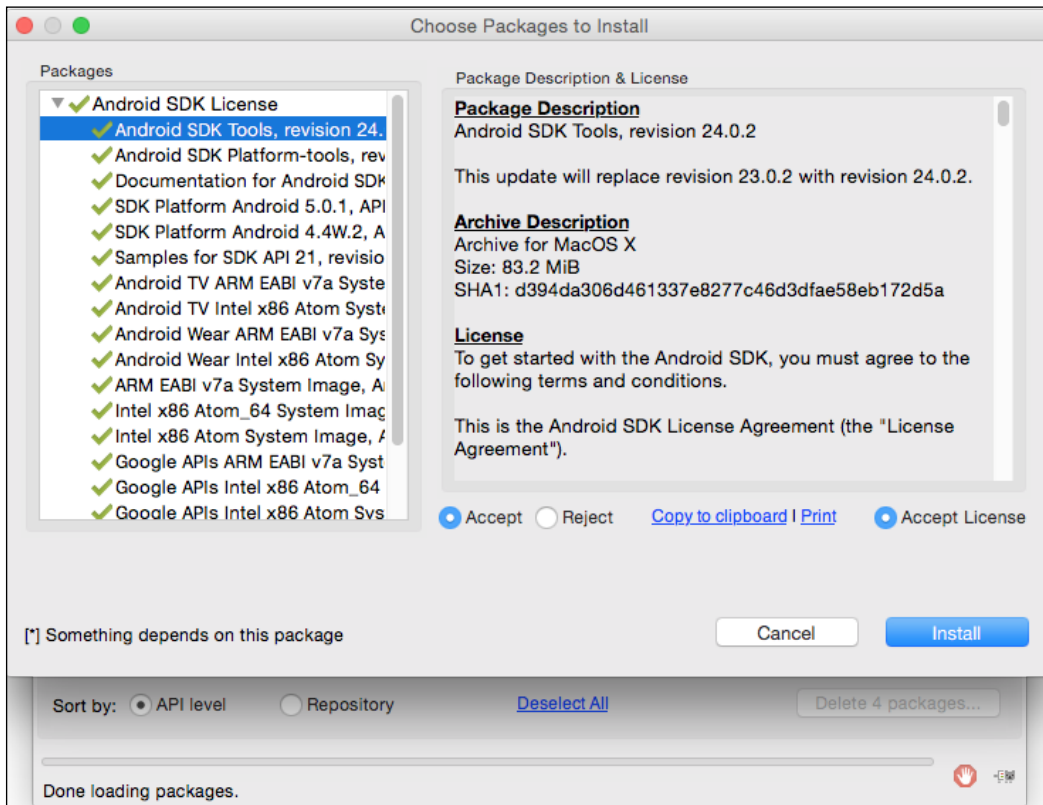
1. You begin by installing Eclipse ADT with the Android SDK, and then continue to unzip the zip file to any working directory you are aware of. I recommend that you unzip it to the Documents folder (`~/adt-bundle-mac-x86_64-20140702`). ADT includes Android SDK and Eclipse. The SDK and Eclipse folders are located under the ADT folder. We call the SDK folder path that is located under the ADT folder `ANDROID_SDK_ROOT`. You have to remember it because you will use it the next recipe. Now, you can launch Eclipse from `~/adt-bundle-mac-x86_64-20140702/eclipse/Eclipse.app`.

2. The next step is to update Android SDK:
 - ❑ Open Eclipse from the `eclipse` folder located in ADT.
 - ❑ Go to **Window | Android SDK Manager**.
 - ❑ After opening **Android SDK Manager**, check **Tools** and the latest Android SDK (API21), Android 2.3.3 (API10) , and any other SDK if necessary, as shown in the following screenshot:



- ❑ Click on **Install packages....**

- ❑ Select each license and click on **Accept**, as shown in the following screenshot:



- ❑ After you accept all licenses, you will see that the **Install** button is enabled. Click on it.
- ❑ You have to wait for a long time to update and install the SDKs.

3. Installing NDK:

Open the terminal window and change the directory to the path from which you downloaded the package. Change the permission on the downloaded package and execute the package. For example:

```
$ chmod 700 android-ndk-r10c-darwin-x86_64.bin
$ ./android-ndk-r10c-darwin-x86_64.bin
```

Finally, you move the NDK folder to the `Documents` folder. We call the installation path for NDK `NDK_ROOT`. `NDK_ROOT` is the address of the folder that contains the files, it helps the Cocos2dx engine to locate the native files of Android. You have to remember `NDK_ROOT` because you will use it in the next recipe.

4. Installing Apache ANT:

Unzip the file to the `Documents` folder. That's all. We call `ANT_ROOT` the installation path for ANT. You have to remember `ANT_ROOT`, as we'll be using it in the next recipe.

5. Installing Java:

By entering the following command in the terminal, you can automatically install Java (if you haven't installed it earlier):

```
$ java --version
```

After installing it, you can check that it was successfully installed by entering the command again.

How it works...

Let's take a look at what we did throughout the recipe:

- ▶ Installing Eclipse: You can use Eclipse as an editor for Cocos2d-x
- ▶ Installing ADT: You can develop Android applications on Eclipse
- ▶ Installing NDK: You can build a C++ source code for Java
- ▶ Installing ANT: You can use command line tools for Cocos2d-x

Now you've finished setting up the Android development environment. At this point, you know how to install them and their path. In the next recipe, you will use them to build and execute Android applications. This will be very useful when you want to debug Android applications.

Installing Cocos2d-x

Getting ready

To follow this recipe, you need to download the zip file from the official site of Cocos2d-x (<http://www.cocos2d-x.org/download>).

At the time of writing this book, version 3.4 was the latest stable version that was available. This version will be used throughout this book.

How to do it...

1. Unzip your file to any folder. This time, we will install the user's home directory. For example, if the user name is `syuhari`, then the install path is `/Users/syuhari/cocos2d-x-3.4`. In this book, we call it `COCOS_ROOT`.


2. The following steps will guide you through the process of setting up Cocos2d-x:

- ❑ Open the terminal
- ❑ Change the directory in terminal to `COCOS_ROOT`, using the following command:

```
$ cd ~/cocos2d-x-v3.4
```
- ❑ Run `setup.py`, using the following command:

```
$ ./setup.py
```
- ❑ The terminal will ask you for `NDK_ROOT`. Enter into `NDK_ROOT` path.
- ❑ The terminal will then ask you for `ANDROID_SDK_ROOT`. Enter the `ANDROID_SDK_ROOT` path.
- ❑ Finally, the terminal will ask you for `ANT_ROOT`. Enter the `ANT_ROOT` path.
- ❑ After the execution of the `setup.py` command, you need to execute the following command to add the system variables:

```
$ source ~/.bash_profile
```

[ Open the `.bash_profile` file, and you will find that `setup.py` shows how to set each path in your system. You can view the `.bash_profile` file using the `cat` command:

```
$ cat ~/.bash_profile
```

]

3. We now verify whether Cocos2d-x can be installed:

- ❑ Open the terminal and run the `cocos` command without parameters:

```
$ cocos
```
- ❑ If you can see a window like the following screenshot, you have successfully completed the Cocos2d-x install process:

```

cocos2d-x-3.4 — bash — 80x24
syuhari cocos2d-x-3.4$ source ~/.bash_profile
syuhari cocos2d-x-3.4$ cocos

/Users/syuhari/cocos2d-x-3.4/tools/cocos2d-console/bin/cocos.py 1.5 - cocos console: A command line tool for cocos2d-x

Available commands:
  run           Compiles & deploy project and then runs it on the target
  luacompile    minifies and/or compiles lua files
  deploy        Deploy a project to the target
  package       Do a package operation
  compile       Compiles the current project to binary
  framework     Do a framework operation
  new           Creates a new project
  jscompile     minifies and/or compiles js files

Available arguments:
  -h, --help    Show this help information
  -v, --version Show the version of this command tool

Example:
  /Users/syuhari/cocos2d-x-3.4/tools/cocos2d-console/bin/cocos.py new --help
  /Users/syuhari/cocos2d-x-3.4/tools/cocos2d-console/bin/cocos.py run --help

```

How it works...

Let's take a look at what we did throughout the above recipe. You can install Cocos2d-x by just unzipping it. You know `setup.py` is only setting up the `cocos` command and the path for Android build in the environment. Installing Cocos2d-x is very easy and simple. If you want to install a different version of Cocos2d-x, you can do that too. To do so, you need to follow the same steps that are given in this recipe, but they will be for a different version.

There's more...

Setting up the Android environment is a bit tough. If you recently started to develop Cocos2d-x, you can skip the settings part of Android. and you can do it when you run on Android. In this case, you don't have to install Android SDK, NDK, and Apache ANT. Also, when you run `setup.py`, you only press *Enter* without entering a path for each question.

Using the Cocos command

The next step is using the `cocos` command. It is a cross-platform tool with which you can create a new project, build it, run it, and deploy it. The `cocos` command works for all Cocos2d-x supported platforms and you don't need to use an IDE if you don't want to. In this recipe, we take a look at this command and explain how to use it.

How to do it...

1. You can use the `cocos` command help by executing it with the `--help` parameter, as follows:

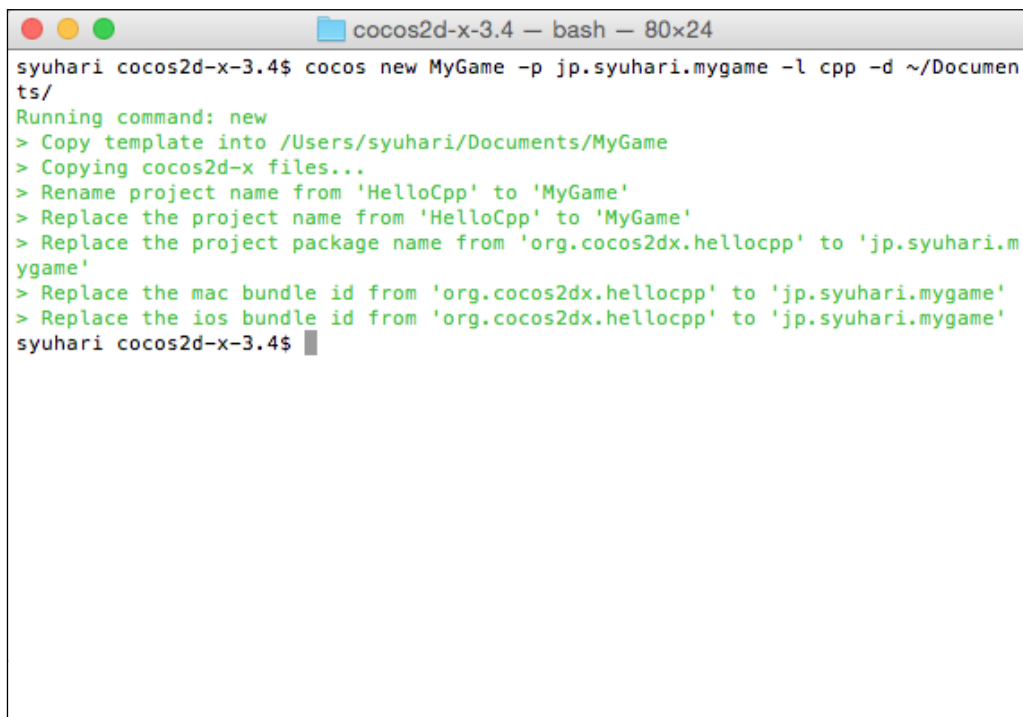
```
$ cocos --help
```

2. We then move on to generating our new project:

First, we create a new Cocos2d-x project with the `cocos new` command, as shown here:

```
$ cocos new MyGame -p com.example.mygame -l cpp -d  
~/Documents/
```

The result of this command is shown the following screenshot:



```
cocos2d-x-3.4 — bash — 80x24  
syuhari cocos2d-x-3.4$ cocos new MyGame -p jp.syuhari.mygame -l cpp -d ~/Documents/  
Running command: new  
> Copy template into /Users/syuhari/Documents/MyGame  
> Copying cocos2d-x files...  
> Rename project name from 'HelloCpp' to 'MyGame'  
> Replace the project name from 'HelloCpp' to 'MyGame'  
> Replace the project package name from 'org.cocos2dx.hellocpp' to 'jp.syuhari.m  
ygame'  
> Replace the mac bundle id from 'org.cocos2dx.hellocpp' to 'jp.syuhari.mygame'  
> Replace the ios bundle id from 'org.cocos2dx.hellocpp' to 'jp.syuhari.mygame'  
syuhari cocos2d-x-3.4$ █
```

Behind the `new` parameter is the project name. The other parameters that are mentioned denote the following:

- `MyGame` is the name of your project.
- `-p` is the package name for Android. This is the application ID in the Google Play store. So, you should use the reverse domain name as the unique name.
- `-l` is the programming language used for the project. You should use `cpp` because we will use C++ in this book.
- `-d` is the location in which to generate the new project. This time, we generate it in the user's documents directory.

You can look up these variables using the following command:

```
$ cocos new --help
```

Congratulations, you can generate your new project. The next step is to build and run using the `cocos` command.

3. Compiling the project:

If you want to build and run for iOS, you need to execute the following command:

```
$ cocos run -s ~/Documents/MyGame -p ios
```

The parameters that are mentioned are explained as follows:

- `-s` is the directory of the project. This could be an absolute path or a relative path.
- `-p` denotes which platform to run on. If you want to run on Android you use `-p android`. The available options are IOS, Android, Win32, Mac, and Linux.
- You can run `cocos run --help` for more detailed information.

The result of this command is shown in the following screenshot:



```
cocos2d-x-3.4 — bash — 80x24
Touch /Users/syuhari/Documents/MyGame/bin/debug/ios/MyGame\ iOS.app
cd /Users/syuhari/Documents/MyGame/proj.ios_mac
export PATH="/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimu
lator.platform/Developer/usr/bin:/Applications/Xcode.app/Contents/Developer/usr/
bin:/usr/local/Cellar/ant/1.9.2/libexec/bin:/Users/syuhari/adt-bundle-mac-x86_64
-20140702/sdk/tools:/Users/syuhari/adt-bundle-mac-x86_64-20140702/sdk/platform-t
ools:/Users/syuhari/adt-bundle-mac-x86_64-20140702/sdk:/Users/syuhari/android-nd
k-r10c:/Users/syuhari/cocos2d-x-3.4/templates:/Users/syuhari/cocos2d-x-3.4/tools
/cocos2d-console/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin"
/usr/bin/touch -c /Users/syuhari/Documents/MyGame/bin/debug/ios/MyGame\ iOS.
app

** BUILD SUCCEEDED **

build succeeded.
Running command: deploy
Deploying mode: debug
Running command: run
starting application
running: '/Users/syuhari/cocos2d-x-3.4/tools/cocos2d-console/plugins/project_run
/bin/ios-sim-xcode6 launch "/Users/syuhari/Documents/MyGame/bin/debug/ios/MyGame
iOS.app" &'

syuhari cocos2d-x-3.4$
```

4. You can now build and run iOS applications on cocos2d-x. However, you have to wait for a long time if this is your first time building an iOS application. It takes a long time to build a Cocos2d-x library, depending on if it was a clean build or a first build.



How it works...

The `cocos` command can create a new project and build it. You should use the `cocos` command if you want to create a new project. Of course, you can build using Xcode or Eclipse. You can easily develop and debug using these tools.

There's more...

The `cocos run` command has other parameters. They are the following:

- ▶ `--portrait` will set the project as a portrait. This command has no argument.
- ▶ `--ios-bundleid` will set the bundle ID for the iOS project. However, it is not difficult to set it later.

The `cocos` command also includes some other commands, which are as follows:

- ▶ The `compile` command: This command is used to build a project. The following patterns are useful parameters. You can see all parameters and options if you execute the `cocos compile [-h]` command:

```
cocos compile [-h] [-s SRC_DIR] [-q] [-p PLATFORM] [-m MODE]
```

- ▶ The `deploy` command: This command only takes effect when the target platform is Android. It will re-install the specified project to the android device or simulator:

```
cocos deploy [-h] [-s SRC_DIR] [-q] [-p PLATFORM] [-m MODE]
```



The `run` command continues to compile and deploy commands.

Building the project using Xcode

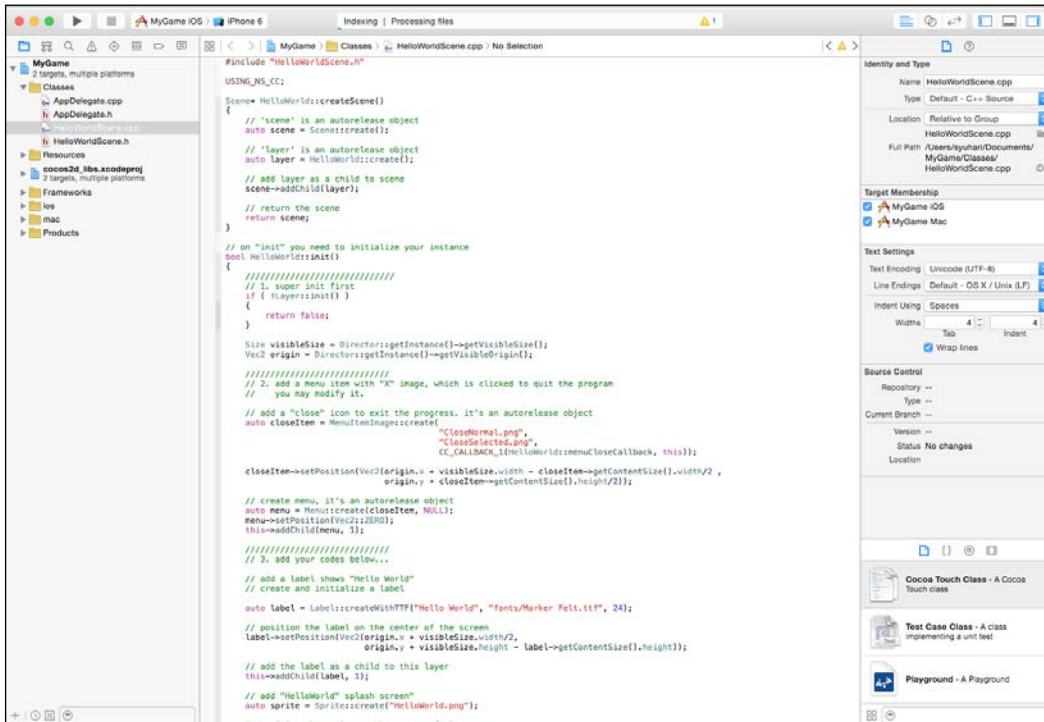
Getting ready

Before building the project using Xcode, you require Xcode with an iOS developer account to test it on a physical device. However, you can also test it on an iOS simulator. If you did not install Xcode, you can get it from the Mac App Store. Once you have installed it, get it activated.

How to do it...

1. Open your project from Xcode:

You can open your project by double-clicking on the file placed at: `~/Documents/MyGame/proj.ios_mac/MyGame.xcodeproj`:



2. Build and Run using Xcode:

You should select an iOS simulator or real device on which you want to run your project.

How it works...

If this is your first time building, it will take a long time but continue to build with confidence as it's the first time. You can develop your game faster if you develop and debug it using Xcode rather than Eclipse.

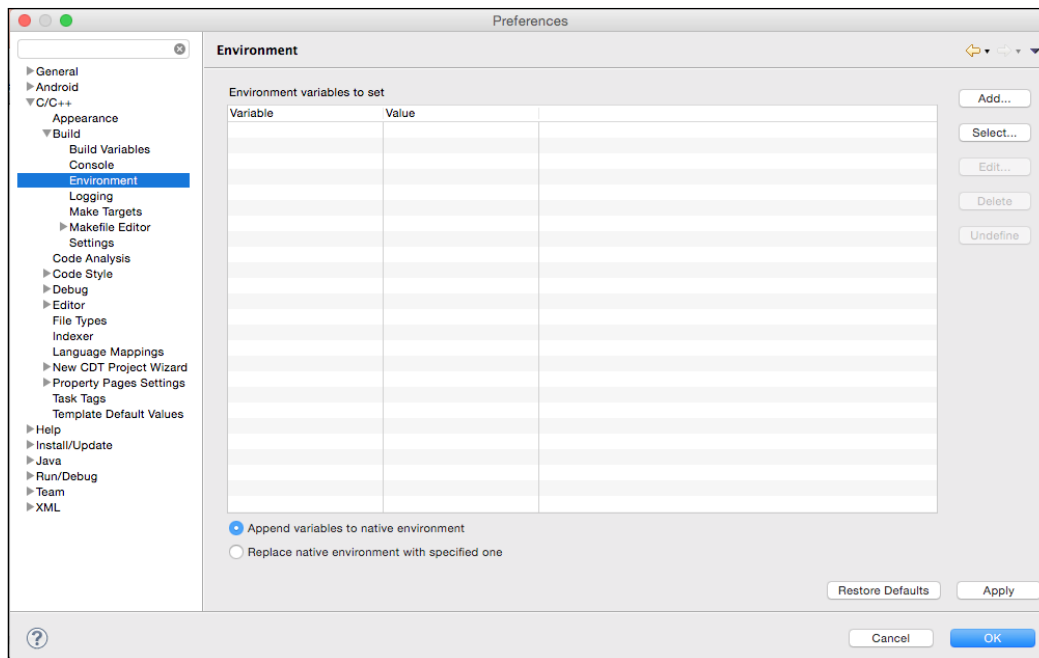
Building the project using Eclipse

Getting ready

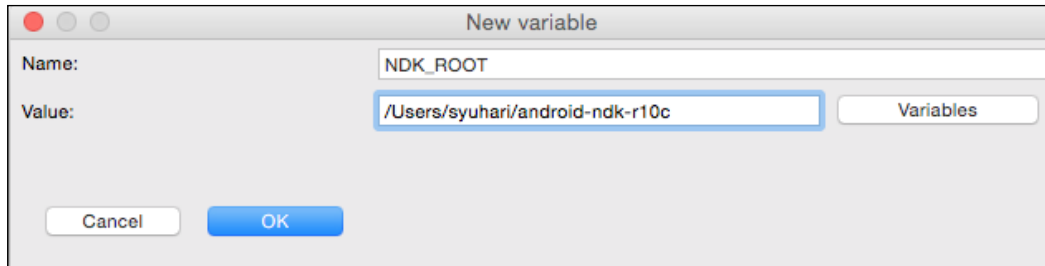
You must finish the first recipe before you begin this step. If you have not finished it yet, you will need to install Eclipse.

How to do it...

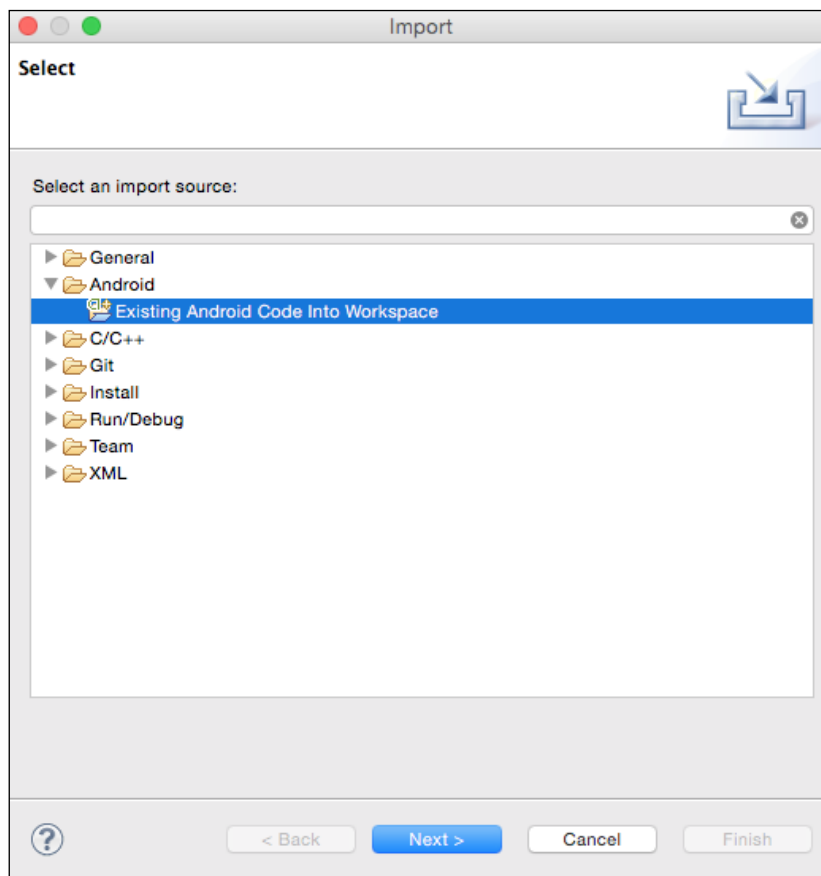
- Setting up NDK_ROOT:
 - Open the preference of Eclipse
 - Open **C++** | **Build** | **Environment**



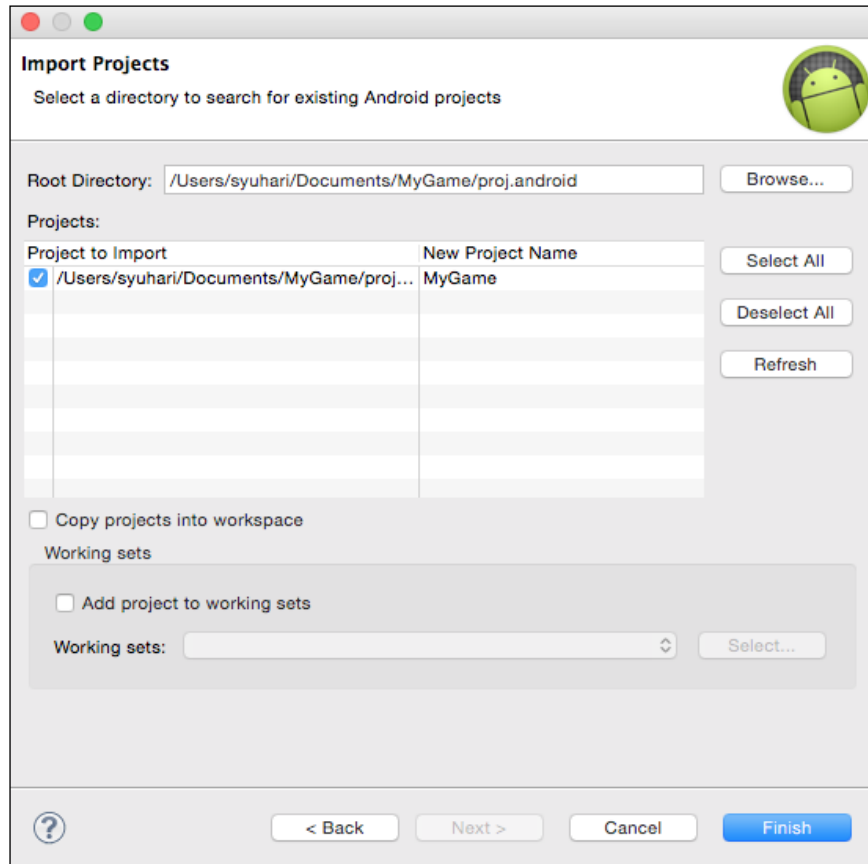
2. Click on **Add** and set the new variable, the name is NDK_ROOT, and the value is NDK_ROOT path:



3. Importing your project into Eclipse:
 - ❑ Open the file and click on **Import**
 - ❑ Go to **Android | Existing Android Code into Workspace**
 - ❑ Click on **Next**



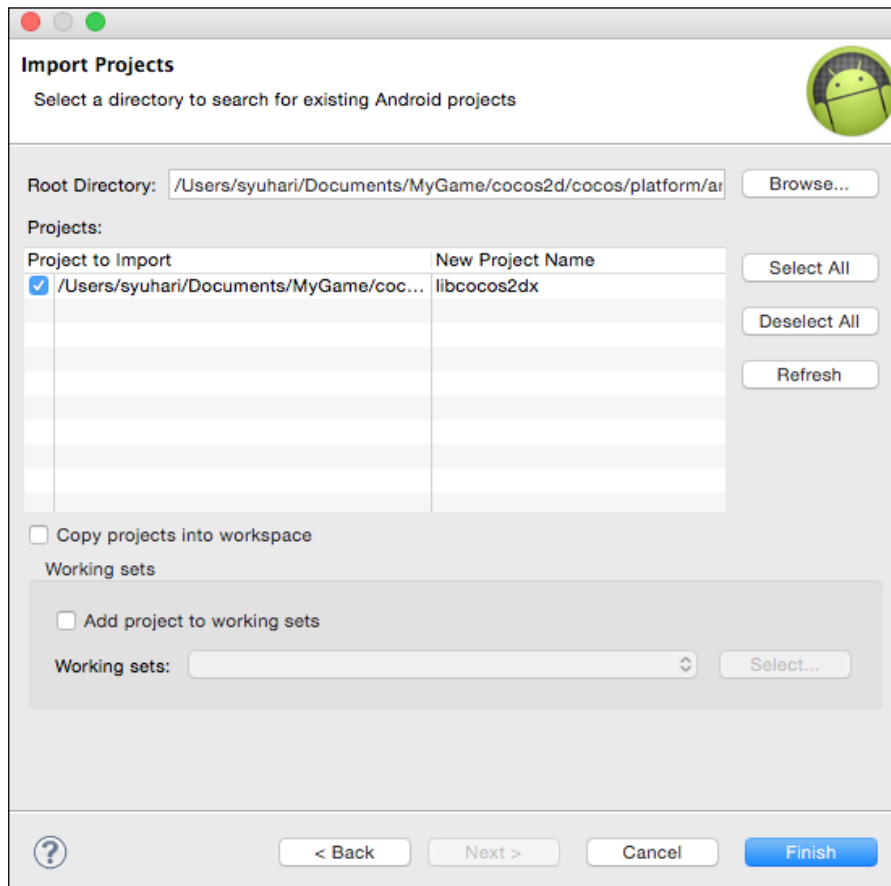
4. Import the project into Eclipse at `~/Documents/MyGame/proj.android`:



5. Importing the Cocos2d-x library into Eclipse:
 - Perform the same steps from Step 3 to Step 4.

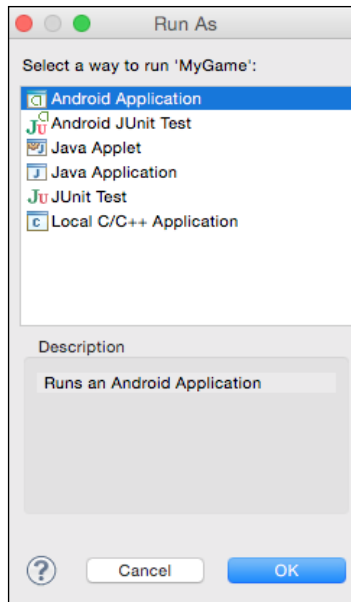
- Import the project `cocos2d lib` at `~/Documents/MyGame/cocos2d/cocos/platform/android/java`, using the following command:

```
importing cocos2d lib
```

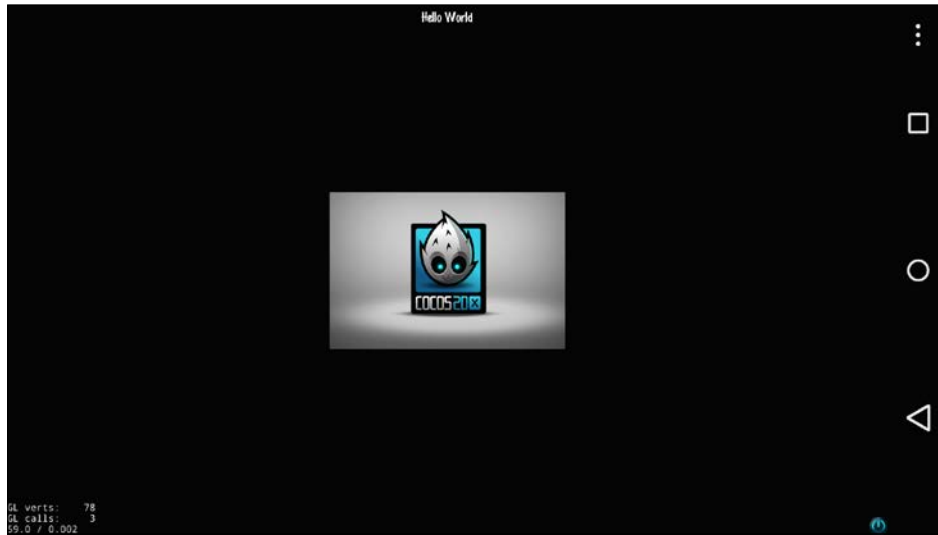


6. Build and Run:

- Click on the Run icon
- The first time, Eclipse will ask you to select a way to run your application. Select **Android Application** and click on **OK**, as shown in the following screenshot:



- If you connected to the Android device on your Mac, you can run your game on your real device or an emulator. The following screenshot shows that it is running on Nexus5:



7. If you added `cpp` files into your project, you have to modify the `Android.mk` file at `~/Documents/MyGame/proj.android/jni/Android.mk`. This file is needed to build the NDK. This fix is required to add files.

8. The original `Android.mk` would look as follows:

```
LOCAL_SRC_FILES := hellocpp/main.cpp \  
                  ../../Classes/AppDelegate.cpp \  
                  ../../Classes/HelloWorldScene.cpp
```

9. If you added the `TitleScene.cpp` file, you have to modify it as shown in the following code:

```
LOCAL_SRC_FILES := hellocpp/main.cpp \  
                  ../../Classes/AppDelegate.cpp \  
                  ../../Classes/HelloWorldScene.cpp \  
                  ../../Classes/TitleScene.cpp
```

The preceding example shows an instance of when you add the `TitleScene.cpp` file. However, if you are also adding other files, you need to add all the added files.

How it works...

You get lots of errors when importing your project into Eclipse, but don't panic. After importing the Cocos2d-x library, errors soon disappear. This allows us to set the path of the NDK, Eclipse could compile C++. After you have modified the C++ codes, run your project in Eclipse. Eclipse automatically compiles C++ codes, Java codes, and then runs.

It is a tedious task to fix `Android.mk` again to add the C++ files. The following code is the original `Android.mk`:

```
LOCAL_SRC_FILES := hellocpp/main.cpp \  
                  ../../Classes/AppDelegate.cpp \  
                  ../../Classes/HelloWorldScene.cpp  
  
LOCAL_C_INCLUDES := $(LOCAL_PATH)/../../Classes
```

The following code is the customized `Android.mk` that adds C++ files automatically:

```
CPP_FILES := $(shell find $(LOCAL_PATH)/../../Classes -name *.cpp)  
LOCAL_SRC_FILES := hellocpp/main.cpp  
LOCAL_SRC_FILES += $(CPP_FILES:$(LOCAL_PATH)/%=%)  
  
LOCAL_C_INCLUDES := $(shell find $(LOCAL_PATH)/../../Classes -type  
d)
```

The first line of the code gets C++ files to the `Classes` directory into the `CPP_FILES` variable. The second and third lines add C++ files into the `LOCAL_C_INCLUDES` variable. By doing so, C++ files will be automatically compiled in the NDK. If you need to compile a file other than the extension `.cpp` file, you will need to add it manually.

There's more...

If you want to manually build C++ in NDK, you can use the following command:

```
$ ./build_native.py
```

This script is located in `~/Documents/MyGame/proj.android`. It uses `ANDROID_SDK_ROOT` and `NDK_ROOT` in it. If you want to see its options, run `./build_native.py -help`.

Implementing multi-resolution support

You may notice a difference in screen appearance on different devices. In some previous recipes, there is an iOS's screenshot and a Nexus 5's screenshot. It shows different image sizes. This image is `HelloWorld.png` located at `MyGame/Resources`. It is 480 x 320 pixels. In this recipe, we explain how to maintain the same size regardless of screen size.

How to do it...

Open `AppDelegate.cpp` through Xcode, and modify the `AppDelegate::applicationDidFinishLaunching()` method by adding the code after the `director->setAnimationInterval(1.0/60.0);` line, as shown in the following code:

```
director->setAnimationInterval(1.0 / 60);  
glview->setDesignResolutionSize(640, 960,  
ResolutionPolicy::NO_BORDER);
```

In this book, we design the game with a screen size of iPhone's 3.5 inch screen. So, we set this screen size to the design resolution size by using the `setDesignResolutionSize` method. The last parameter is resolution policy. The following screenshot is the Nexus 5's screenshot after implementing multi-resolution:



The following screenshot is the iPhone 5 simulator's screenshot. You now know that both screenshots have the same appearance:



How it works...

The resolution policy has `EXACT_FIT`, `NO_BORDER`, `SHOW_ALL`, `FIXED_HEIGHT`, and `FIXED_WIDTH`. These are explained as follows:

- ▶ `EXACT_FIT`: The entire application is visible in the specified area without trying to preserve the original aspect ratio.
- ▶ `NO_BORDER`: The entire application fills the specified area, without distortion but possibly with some cropping, while maintaining the original aspect ratio of the application.
- ▶ `SHOW_ALL`: The entire application is visible in the specified area without distortion, while maintaining the internal the aspect ratio of the application. Borders can appear on two sides of the application.
- ▶ `FIXED_HEIGHT`: The application takes the height of the design resolution size and modifies the width of the internal canvas so that it fits the aspect ratio of the device. No distortion will occur, however, you must make sure your application works on different aspect ratios.
- ▶ `FIXED_WIDTH`: The application takes the width of the design resolution size and modifies the height of the internal canvas so that it fits the aspect ratio of the device. No distortion will occur, however, you must make sure your application works on different aspect ratios.

By implementing multi-resolution, regardless of screen size, you will maintain the image on the screen.

Preparing your original game

In the next chapter, we will start the original game. You know there are a lot of comments and codes in `HelloWorldScene.cpp` and the `HelloWorldScene.h` file. That's why we will remove unnecessary codes in the template project and get started with the original game right away.

How to do it...

1. Open `HelloWorldScene.h` and remove the `menuCloseCallback` method and unnecessary comments. Now `HelloWorldScene.h` should look like the following code:

```
#ifndef __HELLOWORLD_SCENE_H__
#define __HELLOWORLD_SCENE_H__
#include "cocos2d.h"

class HelloWorld : public cocos2d::Layer
{
```



```
public:
    static cocos2d::Scene* createScene();
    virtual bool init();
    CREATE_FUNC(HelloWorld);
};
#endif // __HELLOWORLD_SCENE_H__
```

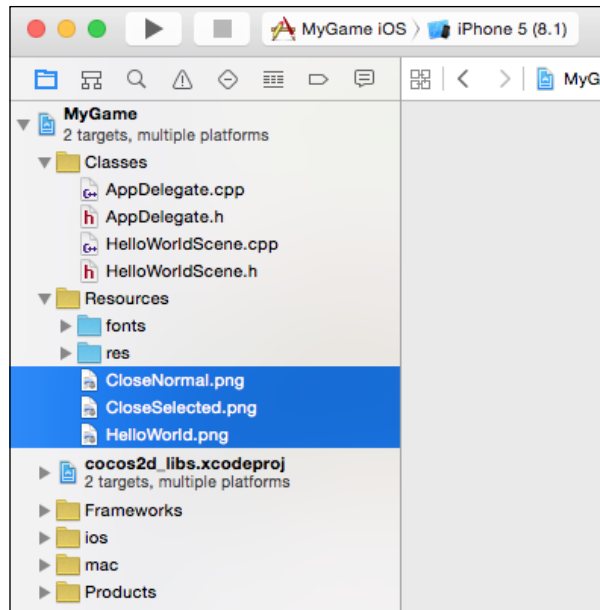
2. The next step is to open `HelloWorldScene.cpp` and remove unnecessary comments, codes, and methods. Now `HelloWorldScene.cpp` should look like the following code:

```
#include "HelloWorldScene.h"
USING_NS_CC;

Scene* HelloWorld::createScene()
{
    auto scene = Scene::create();
    auto layer = HelloWorld::create();
    scene->addChild(layer);
    return scene;
}

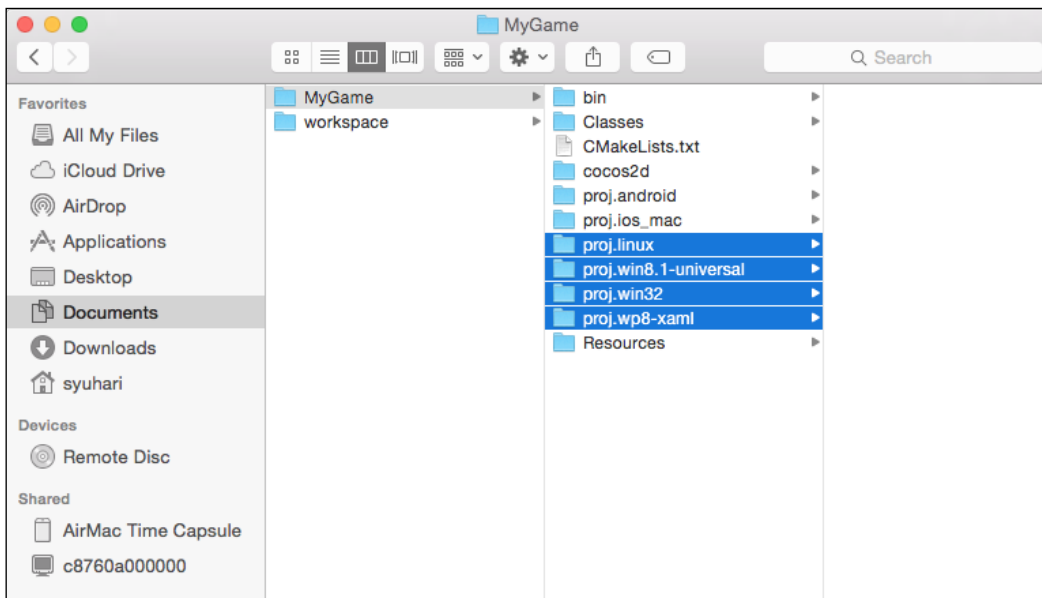
bool HelloWorld::init()
{
    if ( !Layer::init() )
    {
        return false;
    }
    return true;
}
```

3. The next step is to remove unnecessary images in `resources`. Remove `CloseNormal.png`, `CloseSelected.png` and `HelloWorld.png` from the `Resources` folder in Xcode:

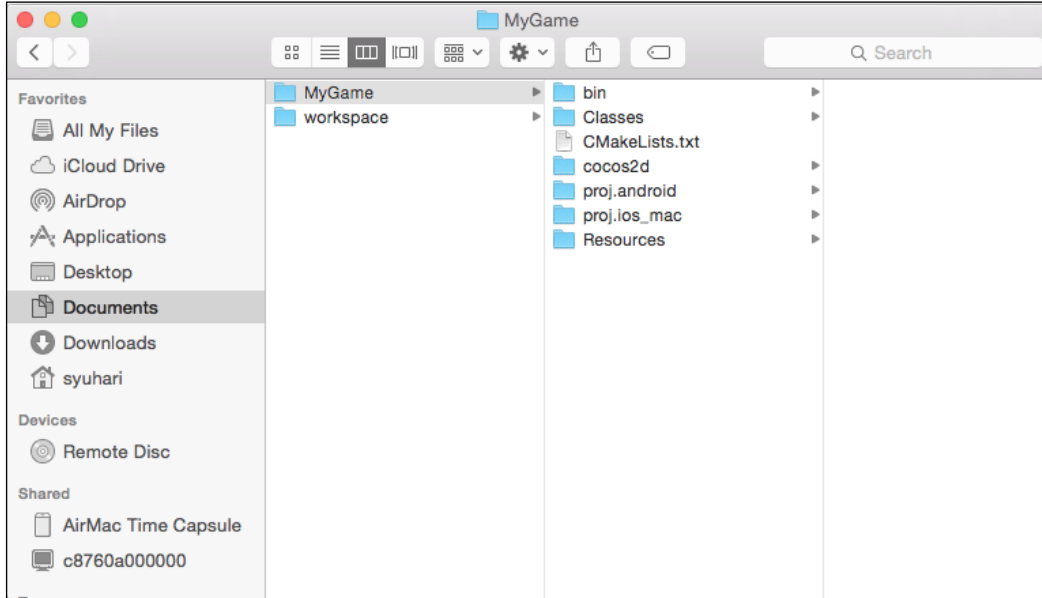


4. Finally, if you are developing only iOS and Android applications, you don't need files for other platforms such as Linux, Windows, and Windows Phone. You should remove these files.

Before removing platform files, it should look like the following screenshot:



After removing platform files, it should look like the following screenshot:



How it works...

With this recipe, you can get the simplest project ready before removing unnecessary comments, codes, and methods. Removing unnecessary platform codes and resources is important for reducing the size of your application. If you start building your original game from scratch, you will need to follow this recipe or chances are, you may get a black screen if you build and run this project. In the next chapter, you can start coding within this simple project.

2

Creating Sprites

In this chapter we're going to create sprites, animations, and actions. The following topics will be covered in this chapter:

- ▶ Creating sprites
- ▶ Getting the sprite's position and size
- ▶ Manipulating sprites
- ▶ Creating animations
- ▶ Creating actions
- ▶ Controlling actions
- ▶ Calling functions with actions
- ▶ Easing actions
- ▶ Using a texture atlas
- ▶ Using a batch node
- ▶ Using 3D models
- ▶ Detecting collisions
- ▶ Drawing a shape

Introduction

Sprites are a 2D image. We can animate and transform them by changing their properties. Sprites are basically, items and your game is not complete without them. Sprites are not only displayed, but also transformed or moved. In this chapter, you will learn how to create sprites using 3D models in Cocos2d-x, and then, we will go through the advantages of sprites.

Creating sprites

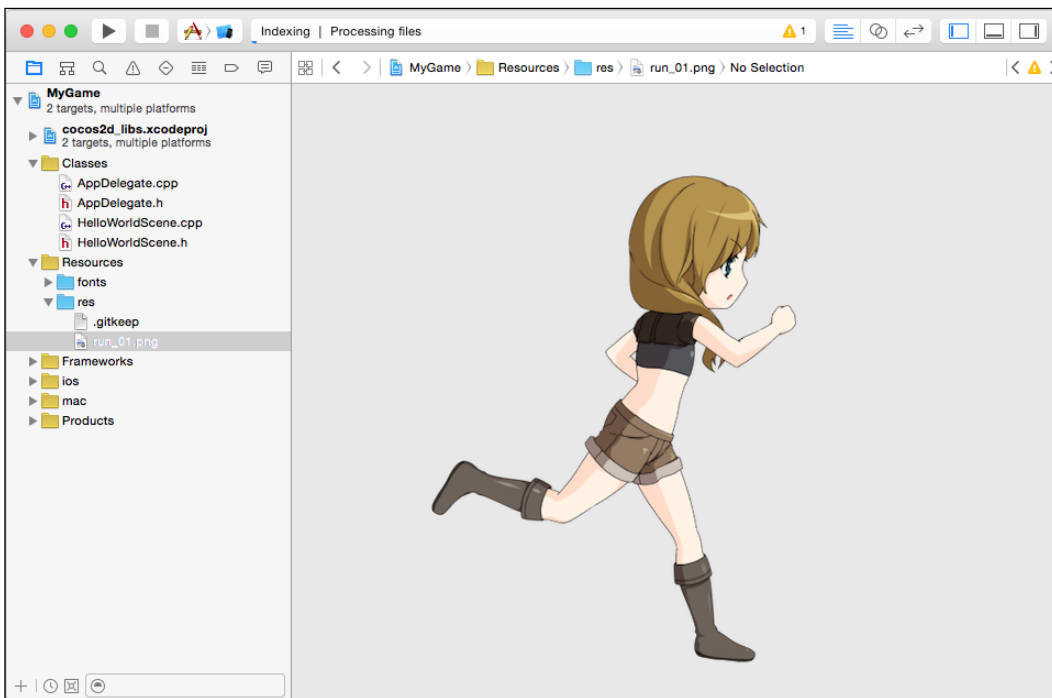
Sprites are the most important things in games. They are images that are displayed on the screen. In this recipe, you will learn how to create a sprite and display it.

Getting ready

You can add the image that you made in the previous chapter into your project, by performing the following steps:

1. Copy the image into the Resource folder `MyGame/Resources/res`.
2. Open your project in Xcode.
3. Go to **Product** | **Clean** from the Xcode menu.

You have to clean and build when you add new images into the `resource` folder. If you did not clean after adding new images, then Xcode will not recognize them. Finally, after you add the `run_01.png` to your project, your project will be seen looking like the following screenshot:

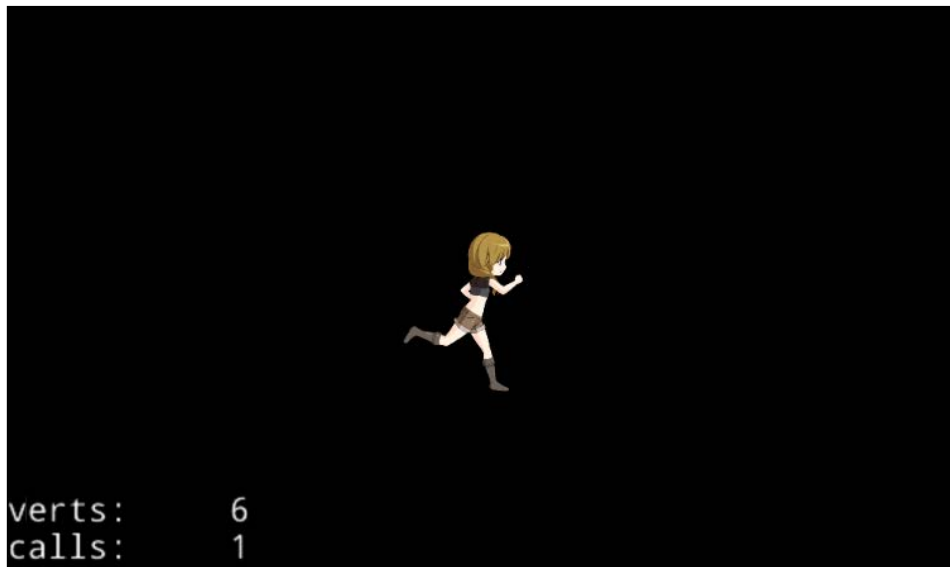


How to do it...

We begin with modifying the `HelloWorld::init` method in the following code:

```
bool HelloWorld::init()
{
    if ( !Layer::init() )
    {
        return false;
    }
    Size size = Director::getInstance()->getWinSize();
    auto sprite = Sprite::create("res/run_01.png");
    sprite->setPosition(Vec2(size.width/2, size.height/2));
    this->addChild(sprite);
    return true;
}
```

And then, after we build & run the project, we can see the following:



How it works...

You can get the screen size from the `Director::getWinSize` method. The `Director` class is a singleton class. You can get the instance using the `getInstance` method. So you can get the screen size by `Director::getInstance()->getWinSize()`.



Please note that you can get an instance of a singleton class in Cocos2d-x using the `getInstance` method.

Sprites are made from images. You can create a sprite by specifying the image. In this case, you create the sprite by `run_01.png` in the `res` folder.

Next, you need to specify the coordinates of the sprite. In this case, you set the sprite in the center of the screen. The `Size` class has the width and height property. You can specify the location of the sprite using the `setPosition` method. The argument of the `setPosition` method is `Vec2`. `Vec2` has two properties as floating point vector, `x` axis coordinate and `y` axis coordinate.

The last step is to add the sprite on the layer. A layer is like a transparent sheet on the screen. You will learn about layers in *Chapter 4. Building Scenes and Layers*.

All objects that are displayed on the screen are **node**. Sprite and Layer are types of node. If you haven't added it in the other nodes, the node does not appear on the screen. You can add a node in the other nodes by the `addChild` method.

There's more...

You can set the sprite using the static coordinate. In the following case we see that the Sprite position is `(100, 200)`.

```
sprite->setPosition(Vec2(100, 200));
```

Also, you can set the sprite in the center of the screen using C++ operator overloading.

```
sprite->setPosition(size/2);
```

If you want to remove the sprite from the layer, you can remove it by the following code:

```
sprite->removeFromParent();
```

See also

The `Sprite` class has a lot of properties. You can manipulate them and change the sprite's appearance. You will also learn more about layer and the scene, which will be explained in *Chapter 4. Building Scenes and Layers*.

Getting the sprite's position and size

There is a certain size and position of the sprite. In this recipe, we explain how to view the size and position of the sprite.

How to do it...

To get the sprite position, use the following code:

```
Vec2 point = sprite->getPosition();  
float x = point.x;  
float y = point.y;
```

To get the sprite size, use the following code:

```
Size size = sprite->getContentSize();  
float width = size.width;  
float height = size.height;
```

How it works...

By default, the sprite position is (0, 0). You can change the sprite position using the `setPosition` method and get it using the `getPosition` method. You can get the sprite size using the `getContentSize` method. However, you cannot change the sprite size by the `setContentSize` method. The `contentSize` is a constant value. If you want to change the sprite size, you have to change the scale of the sprite. You will learn about the scale in the next recipe.

There's more...

Setting anchor points

Anchor point is a point that you set as a way to specify what part of the sprite will be used when setting its position. The anchor point uses a bottom-left coordinate system. By default, the anchor point of all Node objects is (0.5, 0.5). This means that the default anchor point is the center.

To get the anchor point at the center of the sprite, we use the following code:

```
sprite->setAnchorPoint(Vec2(0.5, 0.5));
```

To get the anchor point at the bottom-left of the sprite, we use the following code:

```
sprite->setAnchorPoint(Vec2(0.0, 0.0));
```

To get the anchor point at the top-left of the sprite, we use the following code:

```
sprite->setAnchorPoint(Vec2(1.0, 0.0));
```

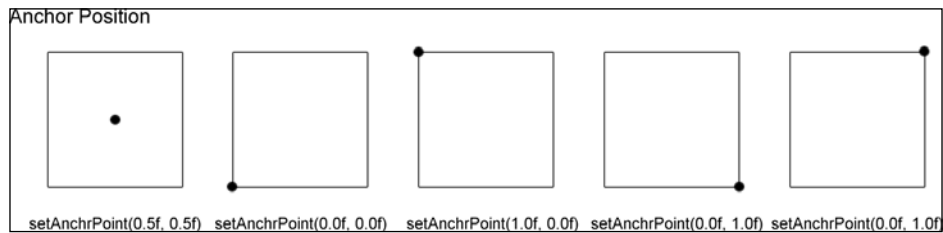

To get the anchor point at the bottom-right of the sprite, we use the following code:

```
sprite->setAnchorPoint(Vec2(0.0, 1.0));
```

To get the anchor point at the top-right of the sprite, we use the following code:

```
sprite->setAnchorPoint(Vec2(1.0, 1.0));
```

The following image shows the various positions of the anchor point:



Rectangle

To get the sprite rectangle, use the following code:

```
Rect rect = sprite->getBoundingBox();
```

```
Size size = rect.size;
```

```
Vec2 point = rect.origin;
```

Rect is the sprite rectangle that has properties such as Size and Vec2. If the scale is not equal to one, then Size in Rect will not be equal to the size, using getContentSize method. Size of getContentSize is the original image size. On the other side, Size in Rect using getBoundingBox is the size of appearance. For example, when you set the sprite to half scale, the Size in Rect using getBoundingBox is half the size, and the Size using getContentSize is the original size. The position and size of a sprite is a very important point when you need to specify the sprites on the screen.

See also

- ▶ The *Detecting collisions* recipe, where you can detect collision using rect.

Manipulating sprites

A Sprite is a 2D image that can be animated or transformed by changing its properties, including its rotation, position, scale, color, and so on. After creating a sprite you can obtain access to the variety of properties it has, which can be manipulated.

How to do it...

Rotate

You can change the sprite's rotation to positive or negative degrees.

```
sprite->setRotation(30.0f);
```

You can get the rotation value using `getRotation` method.

```
float rotation = sprite->getRotation();
```

The positive value rotates it clockwise, and the negative value rotates it counter clockwise. The default value is zero. The preceding code rotates the sprite 30 degrees clockwise, as shown in the following screenshot:



Scale

You can change the sprite's scale. The default value is `1.0f`, the original size. The following code will scale to half size.

```
sprite->setScale(0.5f);
```

You can also change the width and height separately. The following code will scale to half the width only.

```
sprite->setScaleX(0.5f);
```

The following will scale to half the height only.

```
sprite->setScaleY(0.5f);
```

The following code will scale that width to double and the height to half.

```
sprite->setScale(2.0f, 0.5f);
```



Skew

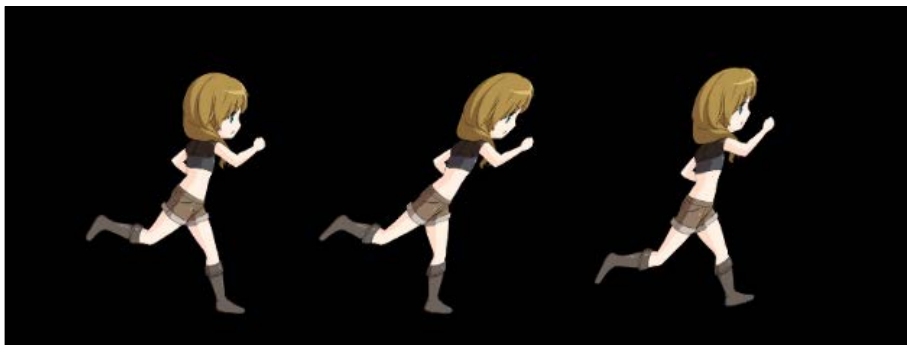
You can change the sprite's skew, either by x, y or uniformly for both x and y. The default value is zero for both x and y.

The following code adjusts the x skew by 20.0:

```
sprite->setSkewX(20.0f);
```

The following code adjusts the y skew by 20.0:

```
sprite->setSkewY(20.0f);
```



Color

You can change the sprite's color by passing in a Color3B object. Color3B has an RGB value.

```
sprite->setColor(Color3b(255, 0, 0));
```

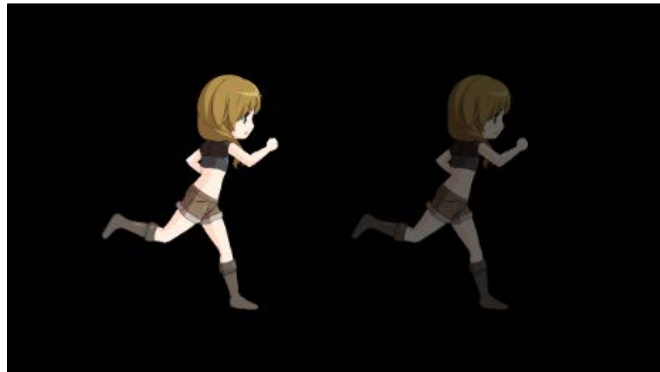


Opacity

You can change the sprite's opacity. The opacity property is set between a value from 0 to 255.

```
sprite->setOpacity(100);
```

The sprite is fully opaque when it is set to 255, and fully transparent when it is set to zero. The default value is always 255.



Visibility

You can change the sprite's visibility by passing in a Boolean value. If it is `false`, then the sprite is invisible; if it is `true`, then the sprite is visible. The default value is always `true`.

```
sprite->setVisible(false);
```



If you want to check the sprite's visibility, use the `isVisible` method rather than the `getVisible` method. The `sprite` class does not have the `getVisible` method.

```
if (sprite->isVisible()) {  
    // visible  
} else {  
    // invisible  
}
```

How it works...

A `Sprite` has a lot of properties. You can manipulate a `sprite` using the `setter` and `getter` methods.

RGB color is a 3 byte value from zero to 255. `Cocos2d-x` provides predefined colors.

```
Color3B::WHITE  
Color3B::YELLOW  
Color3B::BLUE  
Color3B::GREEN  
Color3B::RED  
Color3B::MAGENTA  
Color3B::BLACK  
Color3B::ORANGE  
Color3B::GRAY
```

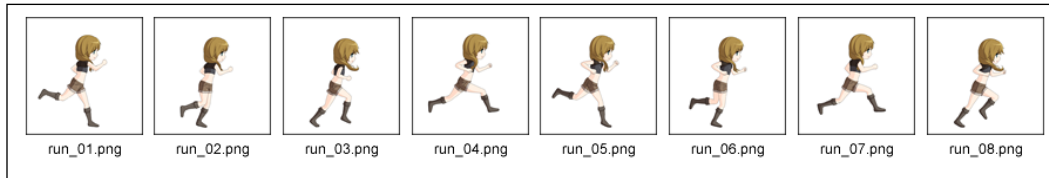
You can find them by looking at the `ccType.h` file in `Cocos2d-x`.

Creating animations

When the characters in a game start to move, the game will come alive. There are many ways to make animated characters. In this recipe, we will animate a character by using multiple images.

Getting ready

You can create an animation from a series of the following image files:



You need to add the running girl's animation image files to your project and clean your project.



Please check the recipe *Creating sprites*, which is the first recipe in this chapter, on how to add images to your project.

How to do it...

You can create an animation using a series of images. The following code creates the running girl's animation.

```
auto animation = Animation::create();
for (int i=1; i<=8; i++) { // from run_01.png to run_08.png
    std::string name = StringUtils::format("res/run_%02d.png", i);
    animation->addSpriteFrameWithFile(name.c_str());
}
animation->setDelayPerUnit(0.1f);
animation->setRestoreOriginalFrame(true);
animation->setLoops(10);
auto action = Animate::create(animation);
sprite->runAction(action);
```

How it works...

You can create an animation using the `Animation` class and the `Animate` class. They change multiple images at regular intervals. The names of the series image files have the serial number, we have added a file name to the `Animation` class in the for loop. We can create the formatted string using the `StringUtils` class in `Cocos2d-x`.

StringUtil is a very useful class. The StringUtil::toString method can generate the std::string value from a variety of values.



```
int i = 100;
std::string int_string = StringUtil::toString(i);
CCLOG("%s ", int_string.c_str());
float j = 123.4f;
std::string float_string = StringUtil::toString(j);
CCLOG("%s", float_string.c_str());
```

StringUtil::format method can generate the std::string value using the printf format.

You can view the log by using CCLOG macro. CCLOG is very useful. You can check the value of the variable in the log during the execution of your game. CCLOG has the same parameters as a sprintf function.

We will add the file name into the Animation instance using the addSpriteFrameWithFile method. It sets the units of time which the frame takes using setDelayPerunit method. It is set to restore the original frame when the animation finishes using the setRestoreOriginalFrame method. True value is to restore the original frame. It is set to the number of times the animation is going to loop. Then, create the Animate instance by passing it with the Animation instance that you created earlier. Finally, run the runAction method by passing in the Animate instance.

If you want to run the animation forever, set -1 using the setLoops method.

```
animation->setLoops(-1);
```

There's more...

In the preceding code, you cannot control each animation frame. In such cases, you can use the AnimationFrame class. This class can control each animation frame. You can set the units of time the frame takes using the second argument of the AnimationFrame::create method.

```
auto rect = Rect::ZERO;
rect.size = sprite->getContentSize();
Vector<AnimationFrame*> frames;
for (int i=1; i<=8; i++) {
    std::string name = StringUtil::format("res/run_%02d.png", i);
    auto frame = SpriteFrame::create(name.c_str(), rect);
    ValueMap info;
    auto animationFrame = AnimationFrame::create(frame, i, info);
    frames.pushBack(animationFrame);
}
```

```
auto animation = Animation::create(frames, 0.1f);
animation->setDelayPerUnit(0.1f);
animation->setRestoreOriginalFrame(true);
animation->setLoops(-1);
auto action = Animate::create(animation);
sprite->runAction(action);
```

See also

- ▶ The *Using a texture atlas* recipe to create an animation using texture atlas

Creating actions

Cocos2d-x has a lot of actions, for example, move, jump, rotate, and so on. We often use these actions in our games. This is similar to an animation, when the characters in a game start their action, the game will come alive. In this recipe you will learn how to use a lot of actions.

How to do it...

Actions are very important effects in a game. Cocos2d-x allows you to use various actions.

Move

To move a sprite by a specified point over two seconds, you can use the following command:

```
auto move = MoveBy::create(2.0f, Vec2(100, 100));
sprite->runAction(move);
```

To move a sprite to a specified point over two seconds, you can use the following command:

```
auto move = MoveTo::create(2.0f, Vec2(100, 100));
sprite->runAction(move);
```

Scale

To uniformly scale a sprite by 3x over two seconds, use the following command:

```
auto scale = ScaleBy::create(2.0f, 3.0f);
sprite->runAction(scale);
```

To scale the x axis by 5x, and y axis by 3x over two seconds, use the following command:

```
auto scale = ScaleBy::create(2.0f, 5.0f, 3.0f);
sprite->runAction(scale);
```


To uniformly scale a sprite to 3x over two seconds, use the following command:

```
auto scale = ScaleTo::create(2.0f, 3.0f);
sprite->runAction(scale);
```

To scale X axis to 5x, and Y axis to 3x over two seconds, use the following command:

```
auto scale = ScaleTo::create(2.0f, 5.0f, 3.0f);
sprite->runAction(scale);
```

Jump

To make a sprite jump by a specified point three times over two seconds, use the following command:

```
auto jump = JumpBy::create(2.0f, Vec2(20, 20), 20.0f, 3);
sprite->runAction(jump);
```

To make a sprite jump to a specified point three times over two seconds, use the following command:

```
auto jump = JumpTo::create(2.0f, Vec2(20, 20), 20.0f, 3);
sprite->runAction(jump);
```

Rotate

To rotate a sprite clockwise by 40 degrees over two seconds, use the following command:

```
auto rotate = RotateBy::create(2.0f, 40.0f);
sprite->runAction(rotate);
```

To rotate a sprite counterclockwise by 40 degrees over two seconds, use the following command:

```
auto rotate = RotateTo::create(2.0f, -40.0f);
sprite->runAction(rotate);
```

Blink

To make a sprite blink five times in two seconds, use the following command:

```
auto blink = RotateTo::create(2.0f, -40.0f);
sprite->runAction(blink);
```

Fade

To fade in a sprite in two seconds, use the following command:

```
auto fadein = FadeIn::create(2.0f);
sprite->runAction(fadein);
```

To fade out a sprite in two seconds, use the following command:

```
auto fadeout = FadeOut::create(2.0f);
sprite->runAction(fadeout);
```

Skew

The following code skews a sprite's x axis by 45 degrees and y axis by 30 degrees over two seconds:

```
auto skew = SkewBy::create(2.0f, 45.0f, 30.0f);
sprite->runAction(skew);
```

The following code skews a sprite's x axis to 45 degrees and y axis to 30 degrees over two seconds.

```
auto skew = SkewTo::create(2.0f, 45.0f, 30.0f);
sprite->runAction(skew);
```

Tint

The following code tints a sprite by the specified RGB values:

```
auto tint = TintBy::create(2.0f, 100.0f, 100.0f, 100.0f);
sprite->runAction(tint);
```

The following code tints a sprite to the specified RGB values:

```
auto tint = TintTo::create(2.0f, 100.0f, 100.0f, 100.0f);
sprite->runAction(tint);
```

How it works...

Action objects make a sprite perform a change to its properties. `MoveTo`, `MoveBy`, `ScaleTo`, `ScaleBy` and others, are Action objects. You can move a sprite from one position to another position using `MoveTo` or `MoveBy`.

You will notice that each Action has a `By` and `To` suffix. That's why they have different behaviors. The method with the `By` suffix is relative to the current state of sprites. The method with the `To` suffix is absolute to the current state of sprites. You know that all actions in Cocos2d-x have `By` and `To` suffix, and all actions have the same rule as its suffix.

There's more...

When you want to execute a sprite action, you make an action and execute the `runAction` method by passing in the `action` instance. If you want to stop the action while sprites are running actions, execute the `stopAllActions` method or `stopAction` by passing in the `action` instance that you got as the return value of the `runAction` method.

```
auto moveTo = MoveTo::create(2.0f, Vec2(100, 100));
auto action = sprite->runAction(moveTo);
sprite->stopAction(action);
```

If you run `stopAllActions`, all of the actions that sprite is running will be stopped. If you run `stopAction` by passing the `action` instance, that specific action will be stopped.

Controlling actions

In the previous recipe, you learned some of the basic actions. However, you may want to use more complex actions; for example, rotating a character while moving, or moving a character after jumping. In this recipe, you will learn how to control actions.

How to do it...

Sequencing actions

`Sequence` is a series of actions to be executed sequentially. This can be any number of actions.

```
auto move = MoveBy::create(2.0f, Vec2(100, 0));
auto rotate = RotateBy::create(2.0f, 360.0f);
auto action = Sequence::create(move, rotate, nullptr);
sprite->runAction(action);
```

The preceding command will execute the following actions sequentially:

- ▶ Move a sprite 100px to the right over two seconds
- ▶ Rotate a sprite clockwise by 360 degree over two seconds

It takes a total of four seconds to execute these commands.

Spawning actions

`Spawn` is very similar to `Sequence`, except that all actions will run at the same time. You can specify any number of actions at the same time.

```
auto move = MoveBy::create(2.0f, Vec2(100, 0));
auto rotate = RotateBy::create(2.0f, 360.0f);
```

```
auto action = Spawn::create(move, rotate, nullptr);
sprite->runAction(action);
```

It will execute the following actions at the same time:

- ▶ Moved a sprite 100px to the right over two seconds
- ▶ Rotated a sprite clockwise by 360 degree over two seconds

It takes a total of two seconds to execute them.

Repeating actions

Repeat object is to repeat an action the number of specified times.

```
auto rotate = RotateBy::create(2.0f, 360.0f);
auto action = Repeat::create(rotate, 5);
sprite->runAction(action);
```

The preceding command will execute a `rotate` action five times.

If you want to repeat forever, you can use the `RepeatForever` action.

```
auto rotate = RotateBy::create(2.0f, 360.0f);
auto action = RepeatForever::create(rotate);
sprite->runAction(action);
```

Reversing actions

If you generate an action instance, you can call a `reverse` method to run it in the reverse action.

```
auto move = MoveBy::create(2.0f, Vec2(100, 0));
auto action = Sequence::create(move, move->reverse(), nullptr);
sprite->runAction(action);
```

The preceding code will execute the following actions sequentially:

- ▶ Move a sprite 100px to the right over two seconds.
- ▶ Move a sprite 100px to the left over two seconds.

In addition, if you generate a sequence action, you can call a `reverse` method to run it in the opposite order.

```
auto move = MoveBy::create(2.0f, Vec2(100, 0));
auto rotate = RotateBy::create(2.0f, 360.0f);
auto sequence = Sequence::create(move, rotate, nullptr);
auto action = Sequence::create(sequence, sequence->reverse(),
    nullptr);
sprite->runAction(action);
```

The preceding code will execute the following actions sequentially:

- ▶ Move a sprite 100px to the right over two seconds.
- ▶ Rotate a sprite clockwise by 360 degree over two seconds
- ▶ Rotate a sprite counterclockwise by 360 degree over two seconds
- ▶ Move a sprite 100px to the left over two seconds.

DelayTime

DelayTime is a delayed action within the specified number of seconds.

```
auto move = MoveBy::create(2.0f, Vec2(100, 0));  
auto delay = DelayTime::create(2.0f);  
auto rotate = RotateBy::create(2.0f, 360.0f);  
auto action = Sequence::create(move, delay, rotate, nullptr);  
sprite->runAction(action);
```

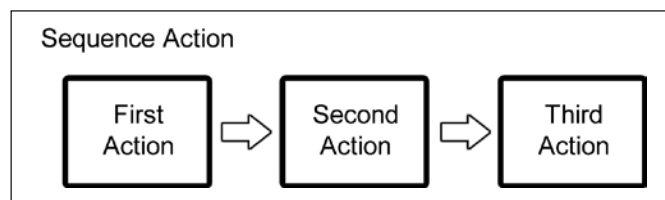
The preceding command will execute the following actions sequentially:

- ▶ Move a sprite 100px to the right over two seconds
- ▶ Delay the next action by two seconds
- ▶ Rotate a sprite clockwise by 360 degree over two seconds

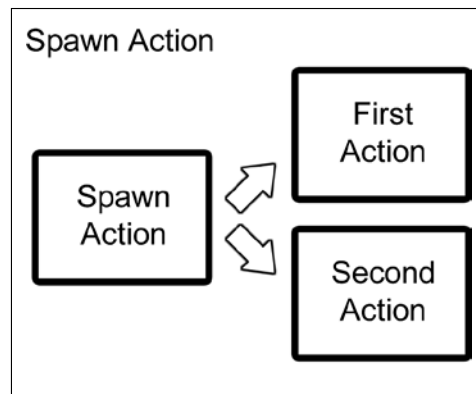
It takes a total of six seconds to execute it.

How it works...

Sequence action runs actions sequentially. You can generate a Sequence instance with actions sequentially. Also, you need to specify nullptr last. If you did not specify nullptr, your game will crash.



Spawn action runs actions at the same time. You can generate a Spawn instance with actions and nullptr like Sequence action.



`Repeat` and `RepeatForever` actions can run, repeating the same action. `Repeat` action has two parameters, the repeating action and the number of repeating actions. `RepeatForever` action has one parameter, the repeating action, which is why it will run forever.

Most actions, including `Sequence`, `Spawn` and `Repeat`, have the `reverse` method. But like the `MoveTo` method that has the suffix `To`, it does not have the `reverse` method; that's why it cannot run the reverse action. `Reverse` method generates its reverse action. The following code uses the `MoveBy::reverse` method.

```
MoveBy* MoveBy::reverse() const
{
    return MoveBy::create(_duration, -_positionDelta);
}
```

`DelayTime` action can delay an action after this. The benefit of the `DelayTime` action is that you can put it in the `Sequence` action. Combining `DelayTime` and `Sequence` is a very powerful feature.

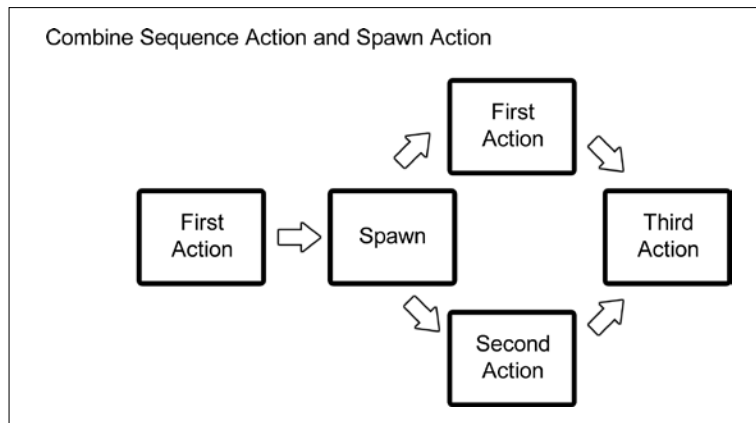
There's more...

`Spawn` produces the same results as running multiple consecutive `runAction` statements.

```
auto move = MoveBy::create(2.0f, Vec2(100, 0));
auto rotate = RotateBy::create(2.0f, 360.0f);
sprite->runAction(move);
sprite->runAction(rotate);
```

However, the benefit of `Spawn` is that you can put it in the `Sequence` action. Combining `Spawn` and `Sequence` is a very powerful feature.

```
auto move = MoveBy::create(2.0f, Vec2(100, 0));
auto rotate = RotateBy::create(2.0f, 360.0f);
auto fadeout = FadeOut::create(2.0f);
auto spawn = Spawn::create(rotate, fadeout, nullptr);
auto fadein = FadeIn::create(2.0f);
auto action = Sequence::create(move, spawn, fadein, nullptr);
sprite->runAction(action);
```



Calling functions with actions

You may want to call a function by triggering some actions. For example, you are controlling the sequence action, jump, and move, and you want to use a sound for the jumping action. In this case, you can call a function by triggering this jump action. In this recipe, you will learn how to call a function with actions.

How to do it...

Cocos2d-x has the `CallFunc` object that allows you to create a function and pass it to be run in your `Sequence`. This allows you to add your own functionality to your `Sequence` action.

```
auto move = MoveBy::create(2.0f, Vec2(100, 0));
auto rotate = RotateBy::create(2.0f, 360.0f);
auto func = CallFunc::create([] () {
    CCLOG("finished actions");
});
auto action = Sequence::create(move, rotate, func, nullptr);
sprite->runAction(action);
```

The preceding command will execute the following actions sequentially:

- ▶ Move a sprite 100px to the right over two seconds
- ▶ Rotate a sprite clockwise by 360 degrees over two seconds
- ▶ Execute CCLOG

How it works...

The `CallFunc` action is usually used as a callback function. For example, if you want to perform a different process after finishing the `move` action. Using `CallFunc`, you can call the method at any time. You can use a lambda expression as a callback function.

If you get a callback with parameters, its code is the following:

```
auto func = CallFuncN::create([](Ref* sender) {
    CCLOG("callback");
    Sprite* sprite = dynamic_cast<Sprite*>(sender);
});
```

The instance of this parameter is the sprite that is running the action. You can get the sprite instance by casting to `Sprite` class.

Then, you can also specify a callback method. `CallFunc` has `CC_CALLBACK_0` macro as an argument. `CC_CALLBACK_0` is a macro for calling a method without parameters. If you want to call a method with one parameter, you need to use the `CallFuncN` action and `CC_CALLBACK_1` macro. `CC_CALLBACK_1` is a macro for calling a method with one argument. A parameter of a method that is called by `CallFuncN` is the `Ref` class.

You can call a method using the following code:

```
bool HelloWorld::init() {
    ...
    auto func =
    CallFunc::create(CC_CALLBACK_0(HelloWorld::finishedAction, this));
    ...
}

void HelloWorld::finishedAction()
{
    CCLOG("finished action");
}
```


To call a method with an argument, you can use the following code:

```
bool HelloWorld::init() {
    ...
    auto func = CallFuncN::create(CC_CALLBACK_1(HelloWorld::callback,
    this));
    ...
}

void HelloWorld::callback(Ref* sender)
{
    CCLOG("callback");
}
```

There's more...

To combine the `CallFuncN` and the `Reverse` action, use the following code:

```
auto move = MoveBy::create(2.0f, Vec2(100, 0));
auto rotate = RotateBy::create(2.0f, 360.0f);
auto func = CallFuncN::create( [=](Ref* sender) {
    Sprite* sprite = dynamic_cast<Sprite*>(sender);
    sprite->runAction(move->reverse());
});
auto action = Sequence::create(move, rotate, func, nullptr);
sprite->runAction(action);
```

The preceding command will execute the following actions sequentially:

- ▶ Move a sprite 100px to the right over two seconds
- ▶ Rotate a sprite clockwise by 360 degree over two seconds
- ▶ Move a sprite 100px to the left over two seconds

Easing actions

Easing is animating with a specified acceleration to make the animations smooth. Ease actions are a good way to fake physics in your game. If you use easing actions with your animations, your game looks more natural with smoother animations.

How to do it...

Let's move a `Sprite` object from (200,200) to (500,200) with acceleration and deceleration.

```

auto sprite = Sprite::create("res/run_01.png");
sprite->setPosition(Vec2(200, 200));
this->addChild(sprite);

auto move = MoveTo::create(3.0f, Vec2(500, 200));
auto ease = EaseInOut::create(move, 2.0f);
sprite->runAction(ease);

```

Next, let's drop a `Sprite` object from the top of the screen and make it bounce.

```

auto sprite = Sprite::create("res/run_01.png");
sprite->setPosition(Vec2(size.width/2, size.height));
sprite->setAnchorPoint(Vec2(0.5f, 0.0f));
this->addChild(sprite);

auto drop = MoveTo::create(3.0f, Vec2(size.width/2, 0));
auto ease = EaseBounceOut::create(drop);
sprite->runAction(ease);

```

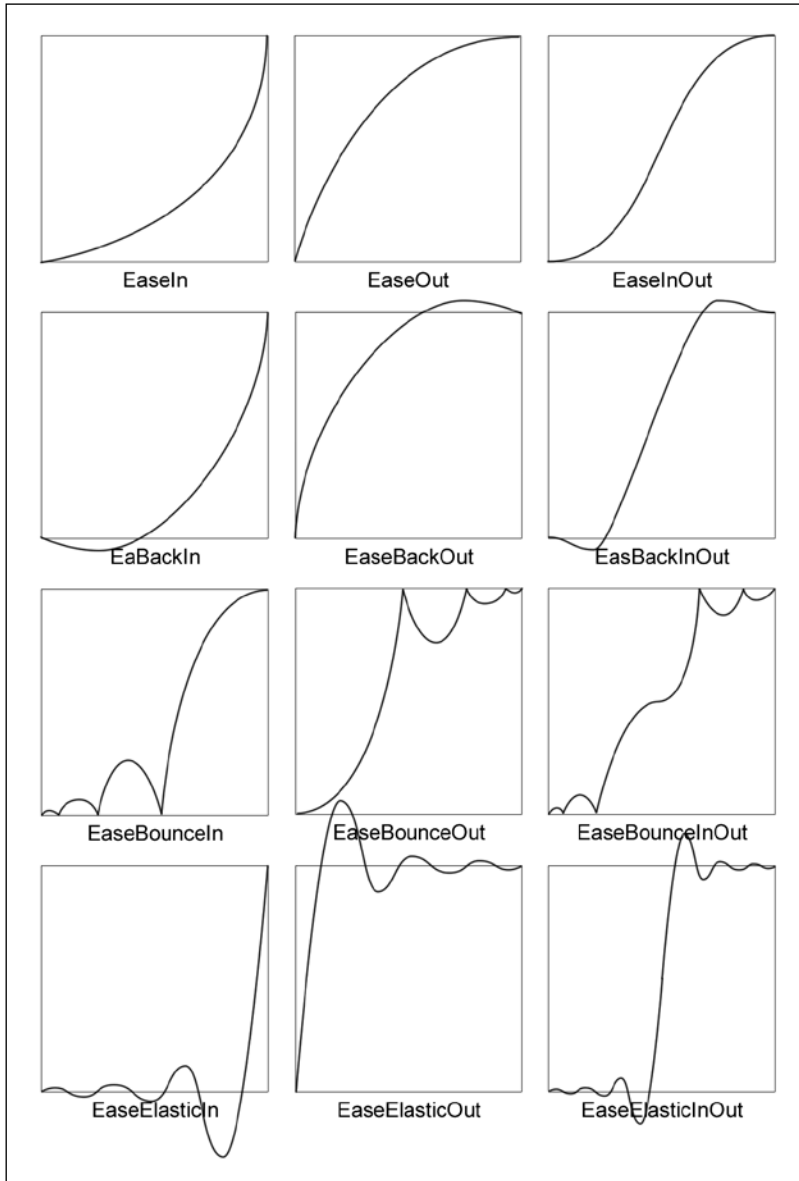
How it works...

The animation's duration time is the same time regardless of whether you use easing. `EaseIn`, `EaseOut` and `EaseInOut` have two parameters—the first parameter is the action by easing, the second parameter is rate of easing. If you specified this parameter to `1.0f`, this easing action is the same without easing. Anything over `1.0f`, means easing is fast, under `1.0f`, and easing will be slow.

The following table are typical easing types.

Class Name	Description
<code>EaseIn</code>	Moves while accelerating.
<code>EaseOut</code>	Moves while decelerating.
<code>EaseInOut</code>	Start moving while accelerating, stop while decelerating.
<code>EaseExponentialIn</code>	It's similar to <code>EaseIn</code> , but meant to accelerate at a rate of exponential curve. It is also used with <code>Out</code> and <code>InOut</code> like <code>EaseIn</code> .
<code>EaseSineIn</code>	It's similar to <code>EaseIn</code> , but meant to accelerate at a rate of sin curve. It is also used with <code>Out</code> and <code>InOut</code> like <code>EaseIn</code> .
<code>EaseElasticIn</code>	Moves after shaking slowly, little by little. It is also used with <code>Out</code> and <code>InOut</code> like <code>EaseIn</code> .
<code>EaseBounceIn</code>	Moves after bouncing. It is also used with <code>Out</code> and <code>InOut</code> like <code>EaseIn</code> .
<code>EaseBackIn</code>	Moves after moving in the opposite direction. It is also used with <code>Out</code> and <code>InOut</code> like <code>EaseIn</code> .

This is a graph that displays typical easing functions:



Using a texture atlas

A **texture atlas** is a large image containing a collection of each sprite. We often use a texture atlas rather than individual images. In this recipe, you will learn how to use a texture atlas.

Getting ready

You have to add the texture atlas files into your project and clean your project.

- ▶ `running.plist`
- ▶ `running.png`

How to do it...

Let's try to read the texture atlas file and make a sprite from it.

```
auto cache = SpriteFrameCache::getInstance();
cache->addSpriteFramesWithFile("res/running.plist");
auto sprite = Sprite::createWithSpriteFrameName("run_01.png");
sprite->setPosition(size/2);
this->addChild(sprite);
```

How it works...

Firstly, we loaded the texture atlas file, when the `SpriteFrameCache` class cached all the images that are included in it. Secondly, you generated a sprite. Do not use the `Sprite::create` method to generate it, use the `Sprite::createWithSpriteFrameName` method instead. Then, you can handle the sprite as a normal sprite.

A texture atlas is a large image containing a collection of images. It is composed of a `plist` file and a texture file. You can create a texture atlas by using tools. You will learn how to make a texture atlas using tools in *Chapter 10, Improving Games with Extra Features*. A `plist` file is defined as the original file name of the image and it is located within the texture file. It also defines the image that will be used by the texture atlas. The `plist` file for the texture atlas is xml format as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>frames</key>
    <dict>
      <key>run_01.png</key>
      <dict>
        <key>frame</key>
        <string>{{2,2},{356,474}}</string>
        <key>offset</key>
        <string>{-62,-26}</string>
```

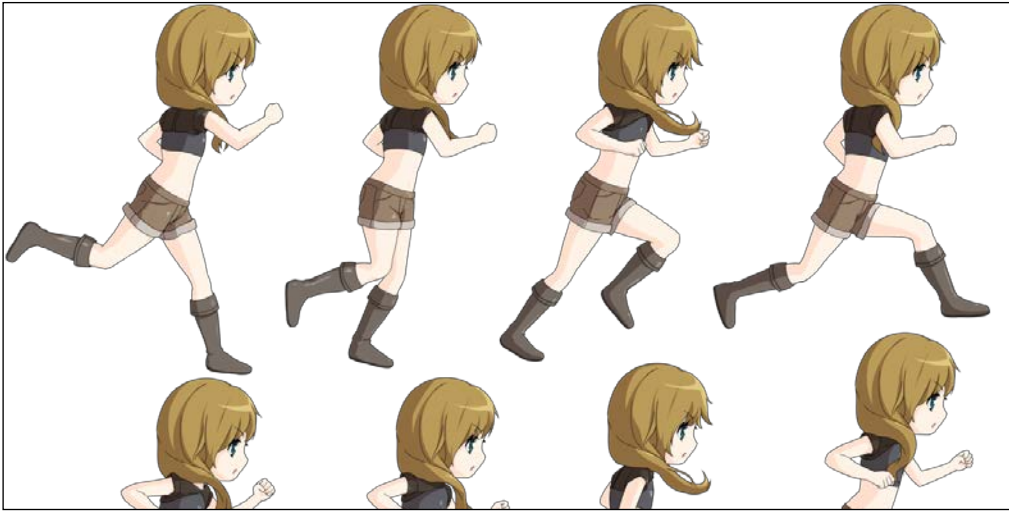
```

        <key>rotated</key>
        <false/>
        <key>sourceColorRect</key>
        <string>{{60,89},{356,474}}</string>
        <key>sourceSize</key>
        <string>{600,600}</string>
    </dict>
    <key>run_02.png</key>
    <dict>
        <key>frame</key>
        <string>{{360,2},{272,466}}</string>
        <key>offset</key>
        <string>{-30,-33}</string>
        <key>rotated</key>
        <false/>
        <key>sourceColorRect</key>
        <string>{{134,100},{272,466}}</string>
        <key>sourceSize</key>
        <string>{600,600}</string>
    </dict>

omit

    </dict>
    <key>metadata</key>
    <dict>
        <key>format</key>
        <integer>2</integer>
        <key>realTextureFileName</key>
        <string>running.png</string>
        <key>size</key>
        <string>{2048,1024}</string>
        <key>smartupdate</key>
        <string>$TexturePacker:SmartUpdate
:e4468ff02abe538ce50e3e1448059f78:1/1$</string>
        <key>textureFileName</key>
        <string>running.png</string>
    </dict>
</dict>
</plist>

```



Why would we use the texture atlas? Because using the memory efficiently is good. Double the memory size is required when the computer loads the image into the memory. For example, there are ten images that are 100x100 size. We will use nine images, but one image requires memories for 128x128 size. On the other hand, texture atlas is one image containing a collection of nine images, where the image size is 1000x1000. It requires a memory size of 1024x1024. This is why texture atlas is used to save wasting unnecessary memory usage.

There's more...

The size of the texture atlas can vary in usage depending on the devices. You can check the maximum texture size of the device in the following codes:

```
int max;  
glGetIntegerv(GL_MAX_TEXTURE_SIZE, &max);  
CLOG("texture size = %d", max);
```

You can generate an animation using a texture atlas and a `plist` file. Firstly, you have to add `run_animation.plist` file into your project. The file is shown in the following screenshot:

Key	Type	Value
▼ Root	Dictionary	(2 items)
▼ animations	Dictionary	(1 item)
▼ run	Dictionary	(4 items)
delayPerUnit	Number	0.1
restoreoriginalFrame	Boolean	YES
loops	Number	-1
▼ frames	Array	(8 items)
▼ Item 0	Dictionary	(3 items)
spriteframe	String	run_01.png
delayUnits	Number	1
▼ notification	Dictionary	(1 item)
firstframe	Boolean	YES
▼ Item 1	Dictionary	(2 items)
spriteframe	String	run_02.png
delayUnits	Number	1
▼ Item 2	Dictionary	(2 items)
spriteframe	String	run_03.png
delayUnits	Number	1
▼ Item 3	Dictionary	(2 items)
spriteframe	String	run_04.png
delayUnits	Number	1
▼ Item 4	Dictionary	(2 items)
spriteframe	String	run_05.png
delayUnits	Number	1
▼ Item 5	Dictionary	(2 items)
spriteframe	String	run_06.png
delayUnits	Number	1
▼ Item 6	Dictionary	(2 items)
spriteframe	String	run_07.png
delayUnits	Number	1
▼ Item 7	Dictionary	(3 items)
spriteframe	String	run_08.png
delayUnits	Number	1
▼ notification	Dictionary	(1 item)
lastframe	Boolean	YES
▼ properties	Dictionary	(2 items)
▼ spritesheets	Array	(1 item)
Item 0	String	running.plist
format	Number	2

This `plist` defines a frame animation. In this case, we defined an animation called `run` using images from `run_01.png` to `run_08.png`. And the animation will loop forever if you specify `-1` to `loop` key's value. The texture atlas was specified `running.plist`.

Secondly, you need to generate an animation using the `plist` file.

```
auto cache = AnimationCache::getInstance();
cache->addAnimationsWithFile("res/run_animation.plist");
auto animation = cache->getAnimation("run");
auto action = Animate::create(animation);
sprite->runAction(action);
```

You also need to cache animation data using the `AnimationCache::addAnimationWithFile` method with the animation `plist`. Next, you will generate an `Animation` instance by specifying `run` that was defined as an animation name in the `plist`. And then, you generate an action from the animation. After that, you can animate using `runAction` method with the action instance.

See also

It is very difficult to create a texture atlas manually. You had better use a tool such as the `TexturePacker`, which you will learn about in *Chapter 11, Taking Advantages*.

Using a batch node

Renderer speed will be slow if there are a lot of sprites on the screen. However, a shooting game needs a lot of images such as bullets, and so on. In this time, if renderer speed is slow, the game earns a bad review. In this chapter, you will learn how to control a lot of sprites.

How to do it...

Let's try to display a lot of sprites using `SpriteBatchNode`.

```
auto batchNode = SpriteBatchNode::create("res/run_01.png");
this->addChild(batchNode);
for (int i=0; i<300; i++) {
    auto sprite = Sprite::createWithTexture(batchNode->getTexture());
    float x = CCRANDOM_0_1() * size.width;
    float y = CCRANDOM_0_1() * size.height;
    sprite->setPosition(Vec2(x,y));
    batchNode->addChild(sprite);
}
```


How it works...

The `SpriteBatchNode` instance can be used to do the following:

- ▶ Generate a `SpriteBatchNode` instance using a texture
- ▶ Add the instance on the layer
- ▶ Generate sprites using the texture in the `SpriteBatchNode` instance
- ▶ Add these sprites on the `SpriteBatchNode` instance

`SpriteBatchNode` can reference only one texture (one image file or one texture atlas). Only the sprites that are contained in that texture can be added to the `SpriteBatchNode`. All sprites added to a `SpriteBatchNode` are drawn in one OpenGL ES draw call. If the sprites are not added to a `SpriteBatchNode` then an OpenGL ES draw call will be needed for each one, which is less efficient.

There's more...

The following screenshot is an executing screen image. You can see three lines of information for Cocos2d-x on the left bottom corner. The top line is the number of polygon vertices. The middle line is the number of OpenGL ES draw call. You understand that a lot of sprites are drawn by one OpenGL ES draw call. The bottom line is FPS and seconds per frame.





If you want to hide this debug information, you should set a false value to the `Director::setDisplayStats` method. You will find it in the `AppDelegate.cpp` in your project.

```
director->setDisplayStats(false);
```

Since Cocos2d-x version 3, the `auto_batch` function of draw calls has been added, Cocos2d-x can draw a lot of sprites with one OpenGL ES draw call, without `SpriteBatchNode`. However, it has the following conditions:

- ▶ Same texture
- ▶ Same BlendFunc

Using 3D modals

Cocos2d-x version 3 supports an exciting new function called **3D modals**. We can use and display 3D modals in Cocos2d-x. In this recipe, you will learn how to use 3D modals.

Getting ready

You have to add the 3D object data into your project and clean your project. The resource files present in the `COCOS_ROOT/test/cpp-tests/Resources/Sprite3DTest` folder are—`body.png` and `girl.c3b`

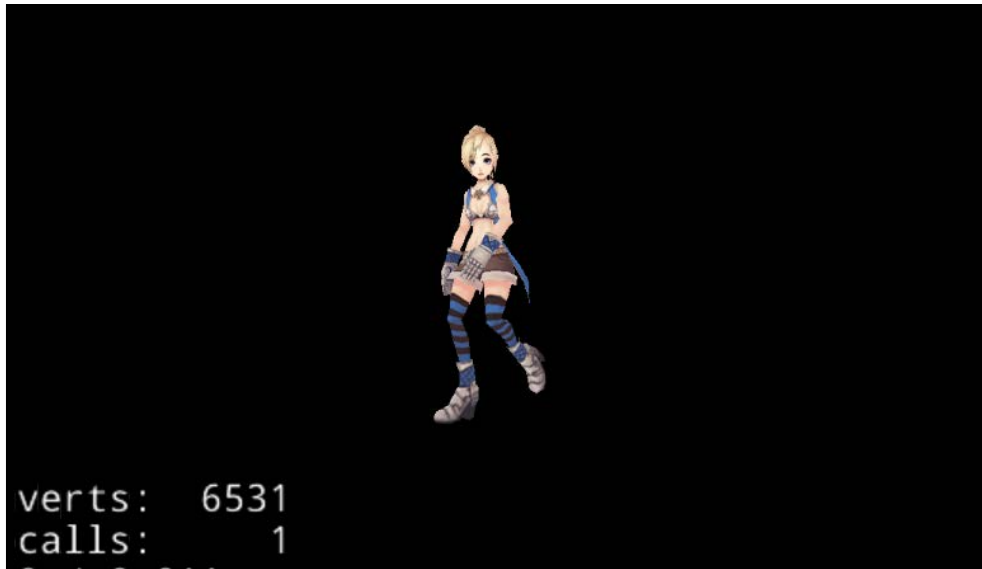
How to do it...

Let's try to display a 3D model and move it.

```
auto size = Director::getInstance()->getWinSize();

// create 3D modal
auto sprite3d = Sprite3D::create("res/girl.c3b");
sprite3d->setPosition(Vec2(size.width/2, 100));
this->addChild(sprite3d);
```

```
// action 3D modal
auto animation3d = Animation3D::create("res/girl.c3b");
auto animate3d = Animate3D::create(animation3d);
auto repeat = RepeatForever::create(animate3d);
sprite3d->runAction(repeat);
```



How it works...

You can create the 3D sprite from a 3D model in the same way as we made a 2D sprite and displayed it. The `Placement` method and the `action` method is exactly the same as seen in a 2D sprite. You can create the `Animation3D` instance from the animation data that is defined in the 3D model.

There's more...

Finally you will try to move the 3D sprite to the left or right. You will notice that 3D sprites differ in appearance depending on their position on the screen when you run the following code:

```
Sprite3d->setPositionX(size.width);
// move fro right to left
auto move1 = MoveBy::create(5.0f, Vec2(-size.width, 0));
auto move2 = MoveBy::create(5.0f, Vec2(size.width, 0));
auto seq = Sequence::create(move1, move2, NULL);
auto loop = RepeatForever::create(seq);
sprite3d->runAction(loop);
```

See also

You can use 3D data formats such as `obj`, `c3b`, and `c3t`. “c3t” stands for Cocos 3d binary. You can get this formatted data by converting `fbx` files.

Detecting collisions

In an action game, a very important technique is to detect collisions between each sprite. However, it is pretty complicated to detect collisions between `rect` and `rect` or `rect` and `point`. In this recipe, you will learn how to detect collisions easily.

How to do it...

There are two ways to detect collisions. The first method checks whether a point is contained within the rectangle of the sprite.

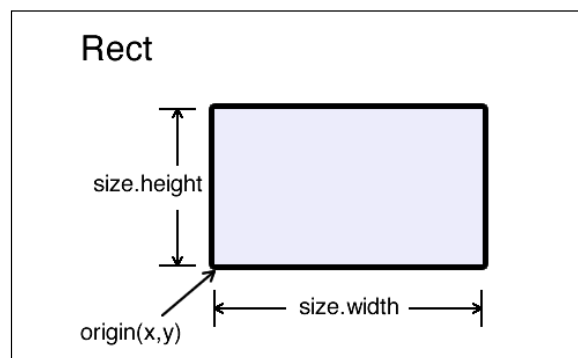
```
Rect rect = sprite->getBoundingBox();
if (rect.containsPoint(Vec2())) {
    CCLOG("the point bumped rectangle");
}
```

The second method checks whether two sprite's rectangles have overlapped.

```
if (rect.intersectsRect(Rect(0, 0, 100, 100))) {
    CCLOG("two rectangles bumped");
}
```

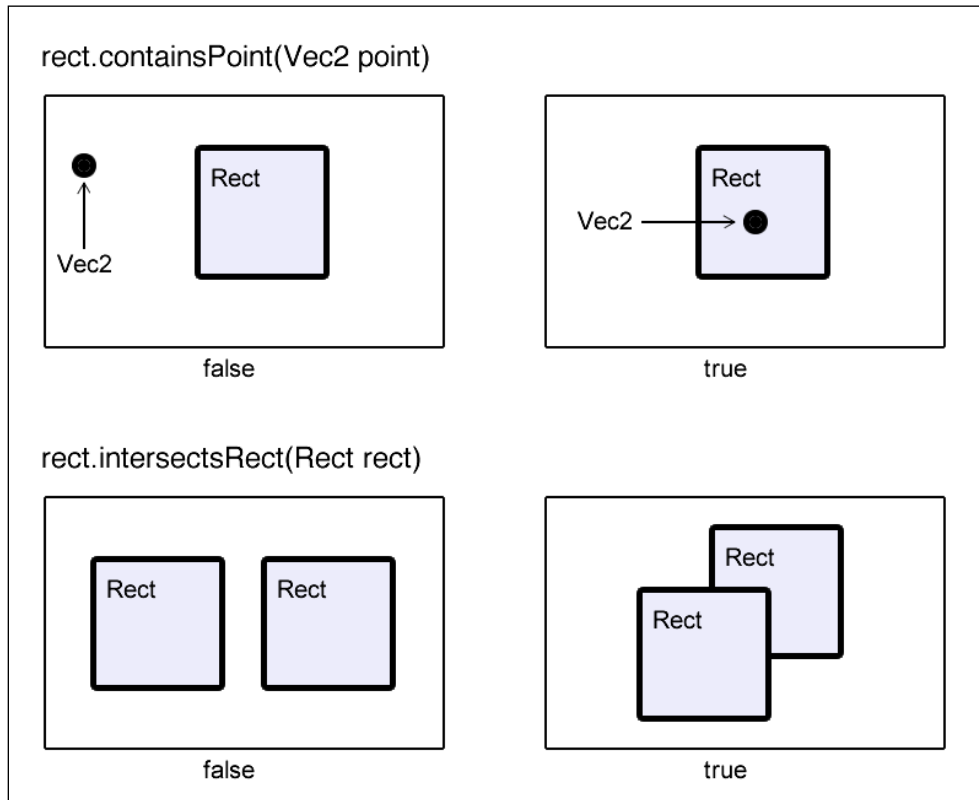
How it works...

The `Rect` class has two properties—`size` and `origin`. The `size` property is the sprite's size. The `origin` property is the sprite's left-bottom coordinate. Firstly, you get the sprite's `rect` using the `getBoundingBox` method.



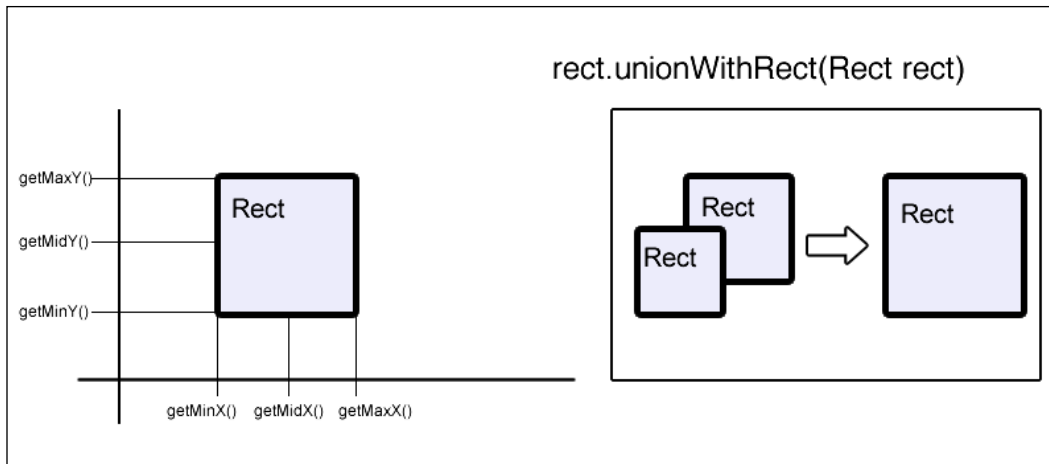
Using the `Rect::containsPoint` method by specifying the coordinate, it is possible to detect whether it contains the rectangle. If it contains it, the method returns `true`. Using `Rect::intersectsRect` method by specifying another rectangle, it is possible to detect whether they overlap. If they overlap, the method returns `true`.

The following image shows a collision between `rect` and `point` or `rect` and `rect`:



There's more...

The `Rect` class has more methods including `getMinX`, `getMidX`, `getMaxX`, `getMinY`, `getMidY`, `getMaxY` and `unionWithRect`. You can obtain the value in the following figure using each of these methods.



See also

- ▶ If you used the physics engine, you can detect collision in a different way. Take a look at *Chapter 9, Controlling Physics*.
- ▶ If you want to detect collision with consideration of the transparent parts of an image, take a look at *Chapter 11 Taking Advantages*.

Drawing a shape

Drawing a shape in Cocos2d-x can be easy using the `DrawNode` class. If you can draw various shapes using `DrawNode`, you will need to prepare textures for such shapes. In this section, you will learn how to draw shapes without textures.

How to do it...

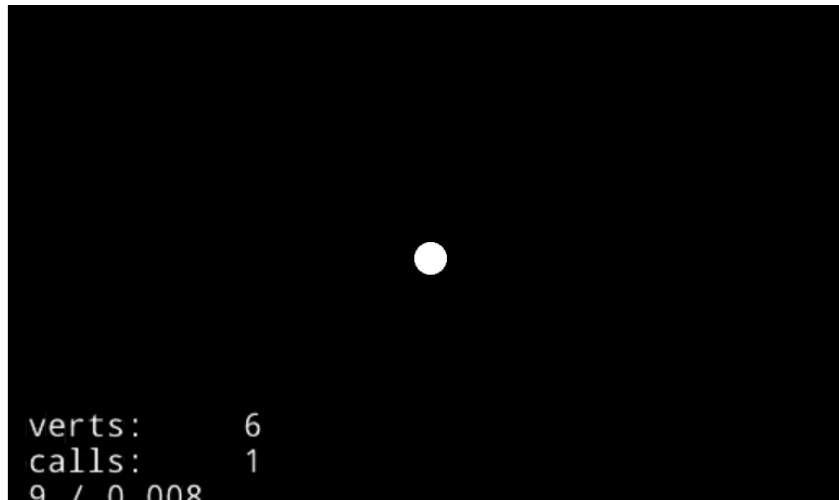
Firstly, you made a `DrawNode` instance as shown in the following codes. You got a window size as well.

```
auto size = Director::getInstance() ->getWinSize();
auto draw = DrawNode::create();
this->addChild(draw);
```

Drawing a dot

You can draw a dot by specifying the point, the radius and the color.

```
draw->drawDot(Vec2(size/2), 10.0f, Color4F::WHITE);
```



Drawing lines

You can draw lines by specifying the starting point, the destination point, and the color. A 1px thick line will be drawn when you use the `drawLine` method. If you want to draw thicker lines, use the `drawSegment` method with a given radius.

```
draw->drawLine(Vec2(300, 200), Vec2(600, 200), Color4F::WHITE);  
draw->drawSegment(Vec2(300, 100), Vec2(600, 100), 10.0f,  
Color4F::WHITE);
```

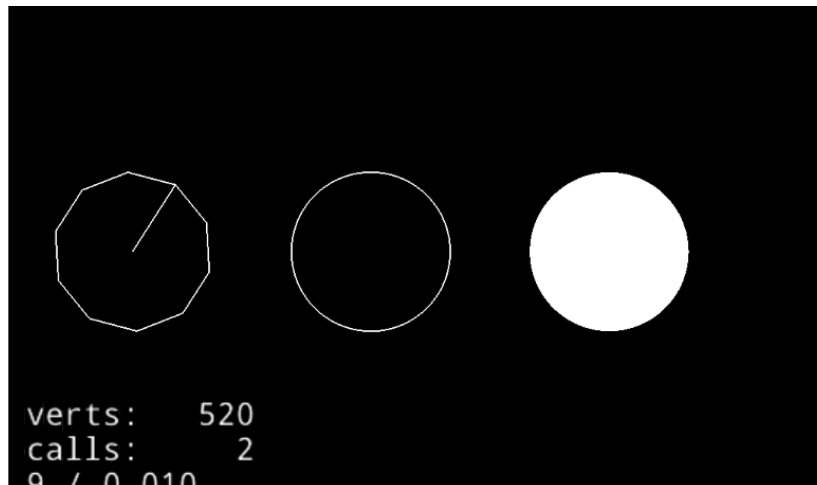


Drawing circles

You can draw circles as shown in the following codes. The specification of the arguments is as follows:

- ▶ center position
- ▶ radius
- ▶ angle
- ▶ segments
- ▶ draw a line to center or not
- ▶ scale x axis
- ▶ scale y axis
- ▶ color

```
draw->drawCircle(Vec2(300, size.height/2), 50.0f, 1.0f, 10, true,  
1.0f, 1.0f, Color4F::WHITE);  
draw->drawCircle(Vec2(450, size.height/2), 50.0f, 1.0f, 100, false,  
1.0f, 1.0f, Color4F::WHITE);  
draw->drawSolidCircle(Vec2(600, size.height/2), 50.0f, 1.0f, 100,  
1.0f, 1.0f, Color4F::WHITE);
```

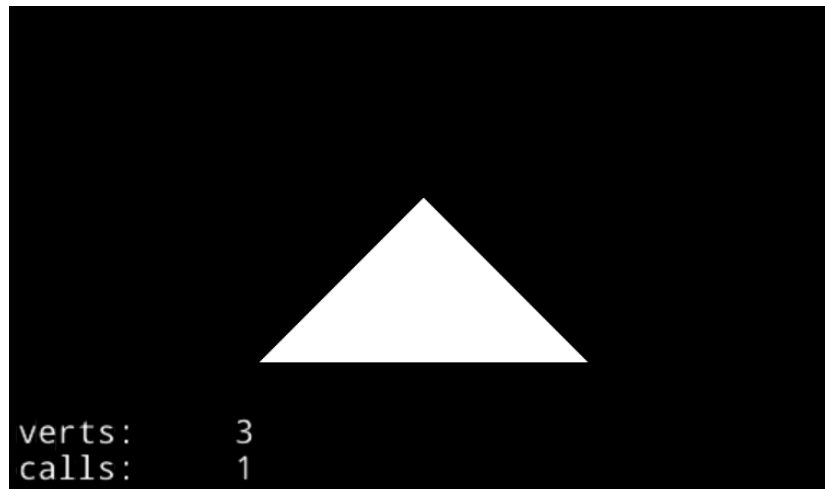


Segment is the number of vertices of the polygon. As you know, the circle is a polygon that has a lot of vertices. Increasing the number of vertices is close to a smooth circle, but the process load goes up. Incidentally, you should use `drawSolidCircle` method if you want to get a solid circle.

Drawing a triangle

You can draw a triangle as in the following code with three vertices and the color.

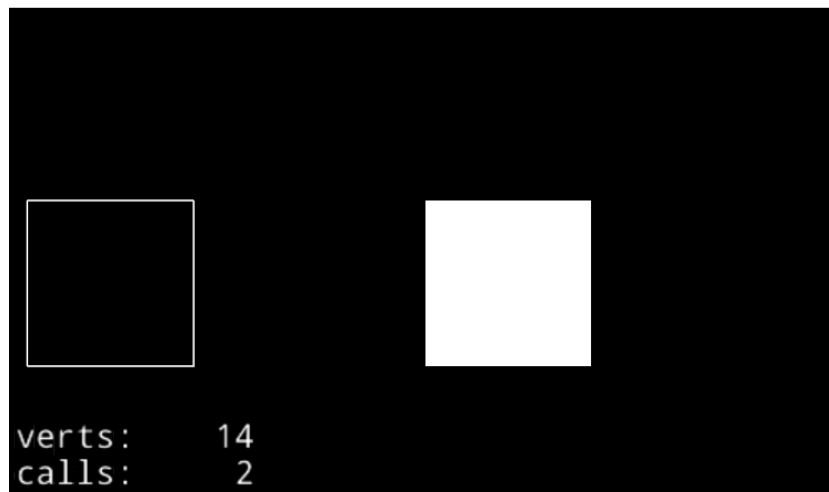
```
draw->drawTriangle(Vec2(380,100), Vec2(480, 200), Vec2(580, 100),  
Color4F::WHITE);
```



Drawing rectangles

You can draw rectangles using the following code with the left-bottom point, the right-top point, and the color. You can draw fill color if you use the `drawSolidRect` method.

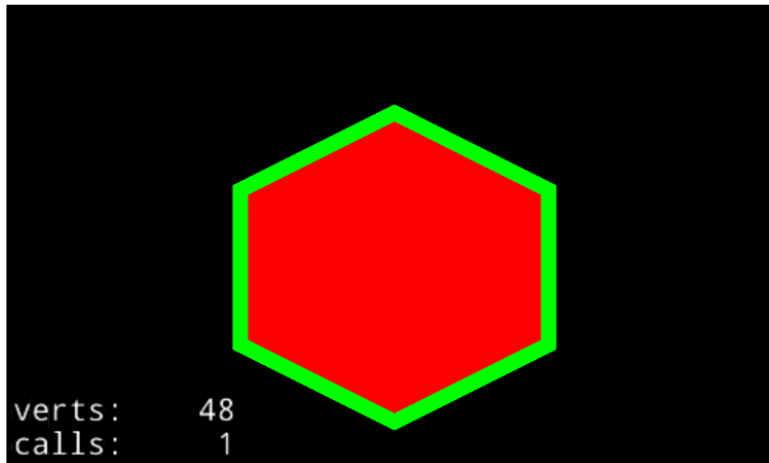
```
draw->drawRect(Vec2(240, 100), Vec2(340,200), Color4F::WHITE);  
draw->drawSolidRect(Vec2(480, 100), Vec2(580, 200), Color4F::WHITE);
```



Drawing a polygon

You can draw a polygon using the following code with the given vertices, the number of vertices, filling color, border's width, and border's color.

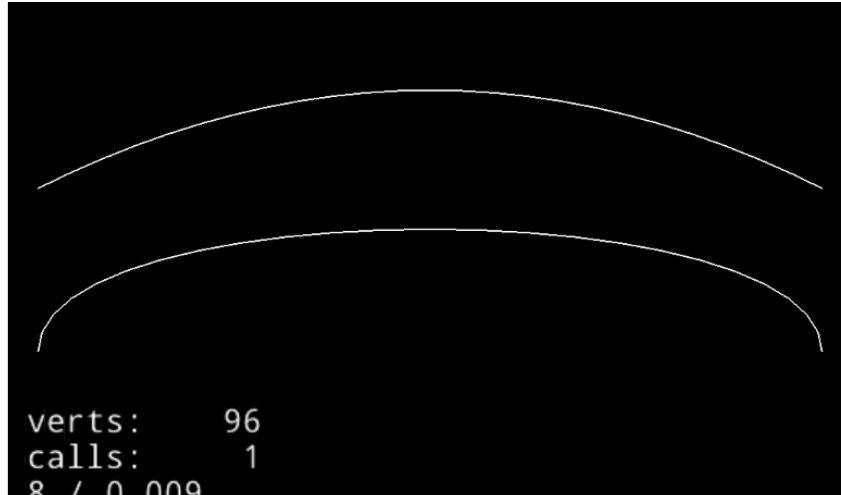
```
std::vector<Vec2>verts;  
verts.push_back(Vec2(380,100));  
verts.push_back(Vec2(380,200));  
verts.push_back(Vec2(480,250));  
verts.push_back(Vec2(580,200));  
verts.push_back(Vec2(580,100));  
verts.push_back(Vec2(480,50));  
draw->drawPolygon(&verts[0], verts.size(), Color4F::RED, 5.0f,  
Color4F::GREEN);
```



Drawing a Bezier curve

You can draw a Bezier curve as shown in the following code. Using `drawQuadBezier` method, you can draw a quadratic Bezier curve, and using `drawCubicBezier` method you can draw a cubic Bezier curve. The third argument of the `drawQuadBezier` method and the fourth argument of the `drawCubicBezier` method is the number of vertices in the same way as the circle.

```
draw->drawQuadBezier(Vec2(240, 200), Vec2(480, 320), Vec2(720, 200),  
24, Color4F::WHITE);  
draw->drawCubicBezier(Vec2(240, 100), Vec2(240, 200), Vec2(720, 200),  
Vec2(720, 100), 24, Color4F::WHITE);
```



How it works...

`DrawNode` is like a mechanism that enables Cocos2d-x to process at a high speed, by making drawing shapes all at once and not separately, or one by one. When you draw multiple shapes, you should use one `DrawNode` instance, instead of multiple `DrawNode` instances and then add multiple shapes in it. Also `DrawNode` does not have the concept of depth. Cocos2d-x will draw to the order of the added shapes in `DrawNode`.

3

Working with Labels

In this chapter, we're going to create labels. To display labels on the screen, you can use the `Label` class with system fonts, true type fonts, and bitmap fonts. The following topics will be covered in this chapter:

- ▶ Creating system font labels
- ▶ Creating true type font labels
- ▶ Creating bitmap font labels
- ▶ Creating rich text

Creating system font labels

Firstly, we will explain how to create a label with system fonts. System fonts are the fonts already installed on your devices. Since they are already installed, there is no need to go through the installation process. Therefore we will skip the installation instructions for system fonts in this recipe, and dive directly into creating labels.

How to do it...

Here's how to create a label by specifying a system font. You can create a single-line label by using the following code:

```
auto label = Label::createWithSystemFont("Cocos2d-x", "Arial",
40);
label->setPosition(size/2);
this->addChild(label);
```



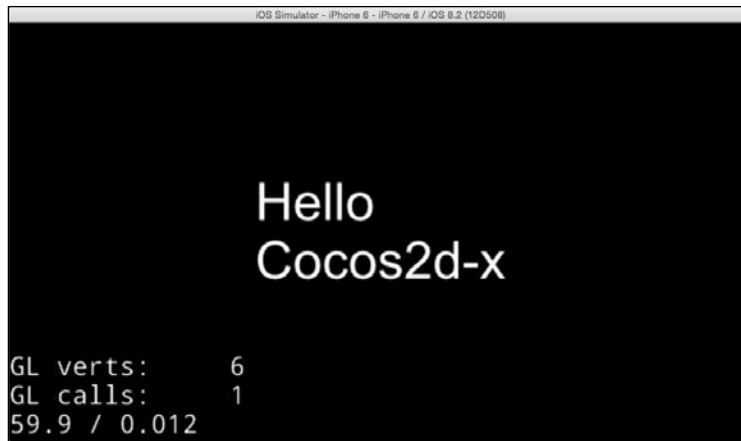
How it works...

You should use the `Label` class to display strings by specifying a string, a system font, and the font size. The `Label` class will display a string that is converted into an image. After creating a `Label` instance, you can use it in the same way as you use `Sprite`. Because `Label` is also a `Node`, we can use properties such as actions, scaling, and opacity functions to manipulate the labels.

Line break

You can also add a new line at any position by putting a line feed code into a string:

```
auto label = Label::createWithSystemFont("Hello\nCocos2d-x",
"Arial", 40);
label->setPosition(size/2);
this->addChild(label);
```



Text align

You can also specify the text alignment in both the horizontal and the vertical directions.

Text alignment type	Description
<code>TextHAlignment::LEFT</code>	Aligns text horizontally to the left. This is the default value for horizontal alignment.
<code>TextHAlignment::CENTER</code>	Aligns text horizontally to the center.
<code>TextHAlignment::RIGHT</code>	Aligns text horizontally to the right.
<code>TextVAlignment::TOP</code>	Aligns text vertically to the top. This is the default value for vertical alignment.
<code>TextVAlignment::CENTER</code>	Aligns text vertically to the center.
<code>TextVAlignment::BOTTOM</code>	Aligns text vertically to the bottom.

The following code is used for aligning text horizontally to the center:

```
label-> setHorizontalAlignment (TextHAlignment::CENTER);
```



There's more...

You can also update the string after creating the label. If you want to update the string once every second, you can do so by setting the timer as follows:

First, edit `HelloWorld.h` as follows:

```
class HelloWorld : public cocos2d::Layer
{
private:
    int sec;
public:
    ...
};
```

Next, edit `HelloWorld.cpp` as follows:

```
sec = 0;
std::string secString = StringUtils::toString(sec);
auto label = Label::createWithSystemFont(secString, "Arial", 40);
label->setPosition(size/2);
this->addChild(label);

this->schedule( [=](float dt) {
    sec++;
    std::string secString = StringUtils::toString(sec);
    label->setString(secString);
}, 1.0f, "myCallbackKey");
```

First, you have to define an integer variable in the header file. Second, you need to create a label and add it on the layer. Next, you need to set the scheduler to execute the function every second. Then you can update the string by using the `setString` method.



You can convert an int or float value to a string value by using the `StringUtil::toString` method.

A scheduler can execute the method at a specified interval. We will explain how the scheduler works in *Chapter 4, Building Scenes and Layers*. Refer to it for more details on the scheduler.

Creating true type font labels

In this recipe, we will explain how to create a label with true type fonts. True type fonts are fonts that you can install into your project. Cocos2d-x's project already has two true type fonts, namely `arial.ttf` and `Marker Felt.ttf`, which are present in the `Resources/fonts` folder.

How to do it...

Here's how to create a label by specifying a true type font. The following code can be used for creating a single-line label by using a true type font:

```
auto label = Label::createWithTTF("True Type Font", "fonts/Marker
Felt.ttf", 40.0f);
label->setPosition(size/2);
this->addChild(label);
```



How it works...

You can create a `Label` with a true type font by specifying a label string, the path to the true type font, and the font size. The true type fonts are located in the `font` folder of `Resources`. Cocos2d-x has two true type fonts, namely `arial.ttf` and `Marker Felt.ttf`. You can generate `Label` objects of different font sizes from one true type font file. If you want to add a true type font, you can use a original true type font if you added it into the `font` folder. However, a true type font is slower than a bitmap font with respect to rendering, and changing properties such as the font face and size is an expensive operation. You have to be careful to not update it frequently.

There's more...

If you want to create a lot of `Label` objects that have the same properties from a true type font, you can create them by specifying `TTFConfig`. `TTFConfig` has properties that are required by a true type font. You can create a label by using `TTFConfig` as follows:

```
TTFConfig config;
config.fontFilePath = "fonts/Marker Felt.ttf";
config.fontSize = 40.0f;
config.glyphs = GlyphCollection::DYNAMIC;
config.outlineSize = 0;
config.customGlyphs = nullptr;
config.distanceFieldEnabled = false;

auto label = Label::createWithTTF(config, "True Type Font");
label->setPosition(size/2);
this->addChild(label);
```

A `TTFConfig` object allows you to set some labels that have the same properties.

If you want to change the color of `Label`, you can change its color property. For instance, by using the following code, you can change the color to `RED`:

```
label->setColor(Color3B::RED);
```

See also

- ▶ You can set effects to labels. Please check the last recipe in this chapter.

Creating bitmap font labels

Lastly, we will explain how to create a label with bitmap type fonts. Bitmap fonts are also fonts that you can install into your project. A bitmap font is essentially an image file that contains a bunch of characters and a control file that details the size and location of each character within the image. If you use bitmap fonts in your game, you can see that the bitmap fonts will be the same size on all devices.

Getting ready

You have to prepare a bitmap font. You can create it by using a tool such as *GlyphDesigner*. We will explain this tool after *Chapter 10, Improving Games with Extra Features*. Now, we will use a bitmap font in *Cocos2d-x*. It is located in the `COCOS_ROOT/tests/cpp-tests/Resources/Fonts` folder. To begin with, you will have to add the files mentioned below to your `Resources/fonts` folder in your project.

- ▶ `future-48.fnt`
- ▶ `future-48.png`

How to do it...

Here's how to create a label by specifying a bitmap font. The following code can be used for creating a single-line label using a bitmap font:

```
auto label = Label::createWithBMFont("fonts/futura-48.fnt",
    "Bitmap Font");
label->setPosition(size/2);
this->addChild(label);
```



How it works...

You can create `Label` with a bitmap font by specifying a `label` string, the path to the true type font, and the font size. The characters in a bitmap font are made up of a matrix of dots. This font renders very fast but is not scalable. That's why it has a fixed font size when generated. A bitmap font requires the following two files: an `.fnt` file and a `.png` file.

There's more...

Each character in `Label` is a `Sprite`. This means that each character can be rotated or scaled and has other changeable properties:

```
auto sprite1 = label->getLetter(0);  
sprite1->setRotation(30.0f);  
  
auto sprite2 = label->getLetter(1);  
sprite2->setScale(0.5f);
```



Creating rich text

After creating `Label` objects on screen, you can create some effects such as a drop shadow and an outline on them easily without having your own custom class. The `Label` class can be used for applying the effects to these objects. However, note that not all label types support all effects.

How to do it...

Drop shadow

Here's how to create `Label` with a drop shadow effect:

```
auto layer = LayerColor::create(Color4B::GRAY);
this->addChild(layer);
auto label = Label::createWithTTF("Drop Shadow", "fonts/Marker
Felt.ttf", 40);
label->setPosition(size/2);
this->addChild(label);
// shadow effect
label->enableShadow();
```



Outline

Here's how to create `UILabel` with an outline effect:

```
auto label = UILabel::createWithTTF("Outline", "fonts/Marker  
Felt.ttf", 40);  
label->setPosition(size/2);  
this->addChild(label);  
// outline effect  
label->enableOutline(Color4B::RED, 5);
```



Glow

Here's how to create `UILabel` with a glow effect:

```
auto label = UILabel::createWithTTF("Glow", "fonts/Marker Felt.ttf",  
40);  
label->setPosition(size/2);  
this->addChild(label);  
// glow effect  
label->enableGlow(Color4B::RED);
```



How it works...

Firstly, we generate a gray layer and change the background color to gray because otherwise we will not be able to see the shadow effect. Adding the effect to the label is very easy. You need to generate a `Label` instance and execute an effect method such as `enableShadow()`. This can be executed without arguments. The `enableOutline()` has two arguments, namely the outline color and the outline size. The outline size has a default value of `-1`. If it has a negative value, the outline does not show. Next, you have to set the second argument. The `enableGlow` method has only one argument, namely glow color.

Not all label types support all effects, but all label types support the drop shadow effect. The `Outline` and `Glow` effects are true type font effects only. In previous versions, we had to create our own custom fonts class if we wanted to apply effects to labels. However, the current version of Cocos2d-x, version 3, supports label effects such as drop shadow, outline, and glow.

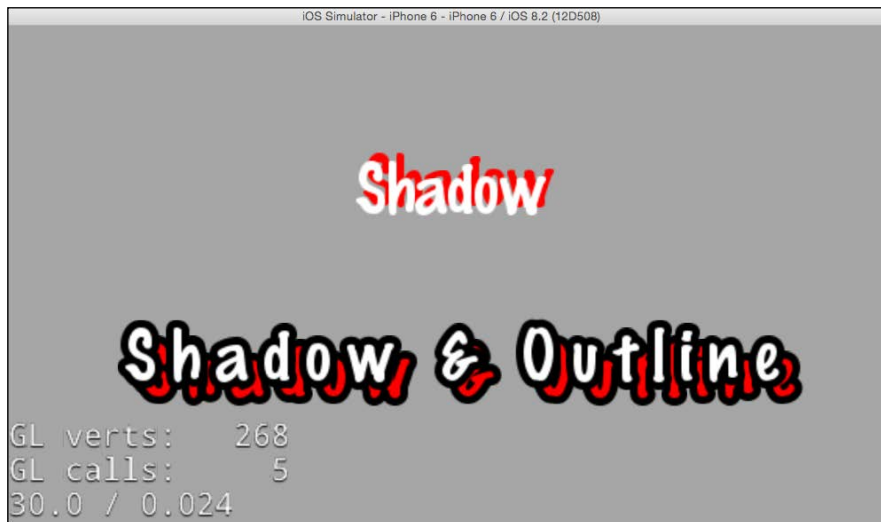
There's more...

You can also change the shadow color and the offset. The first argument is shadow color, the second argument is the offset, and third argument is the blur radius. However, unfortunately, changing the blur radius is not supported in Cocos2d-x version 3.4.

```
auto label = Label::createWithTTF("Shadow", "Fonts/Marker  
Felt.ttf", 40);  
label->setPosition(Vec2(size.width/2, size.height/3*2));  
this->addChild(label);  
label->enableShadow(Color4B::RED, Size(5,5), 0);
```

It is also possible to set two or more of these effects at the same time. The following code can be used for setting the shadow and outline effects for a label:

```
auto label2 = Label::createWithTTF("Shadow & Outline",  
    "fonts/Marker Felt.ttf", 40);  
label2->setPosition(Vec2(size.width/2, size.height/3));  
this->addChild(label2);  
label2->enableShadow(Color4B::RED, Size(10,-10), 0);  
label2->enableOutline(Color4B::BLACK, 5);
```



4

Building Scenes and Layers

The following topics will be covered in this chapter:

- ▶ Creating scenes
- ▶ Transitioning between scenes
- ▶ Transitioning scenes with effects
- ▶ Making original transitions for replacing scenes
- ▶ Making original transitions for popping scenes
- ▶ Creating layers
- ▶ Creating modal layers

Introduction

One screen has one scene. A scene is a container that holds Sprite, Labels, and other objects. For example, a scene can be a title scene, a game scene, or an option menu scene. Each scene has multiple layers. A layer is a transparent sheet similar to Photoshop's layer. Objects that added to layers are displayed on the screen. In this chapter, we will explain how to use the `Scene` class and the `Layer` class and how to transition between scenes. Finally, by the end of this chapter, you will be able to create original scenes and layers.

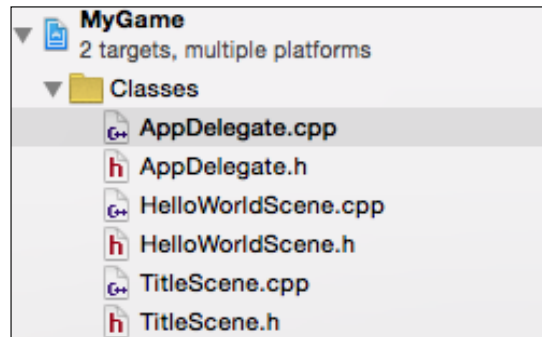
Creating scenes

In Cocos2d-x, your games should have one or more scenes. A scene is basically a node. In this recipe, we will explain how to create and use a `Scene` class.

How to do it...

In this recipe, we will use the project that was created in *Chapter 1, Getting Started with Cocos2d-x*.

1. Firstly, duplicate the `HelloWorldScene.cpp` and `HelloWorldScene.h` files at Finder and rename them as `TitleScene.cpp` and `TitleScene.h`. Secondly, add them to the Xcode project. The result is shown in the following image:



2. Next, we have to change `HelloWorldScene` to `TitleScene` and place the search and replace method in the tips section.



How to search for and replace a class name?

In this case, select `TitleScene.h` and then the **Find | Find and Replace ...** menu in Xcode. Then, enter `HelloWorld` in the **String Matching** area and `TitleScene` in the **Replacement String** area. Execute all replacements. Follow the same process for `TitleScene.cpp`. The result is the following code for `TitleScene.h`:

The result obtained for `TitleScene.h` is as follows:

```
#ifndef __TitleScene_SCENE_H__
#define __TitleScene_SCENE_H__
```

```
#include "cocos2d.h"
class TitleScene : public cocos2d::Layer
{
public:
    static cocos2d::Scene* createScene();
    virtual bool init();
    CREATE_FUNC(TitleScene);
};

#endif // __TitleScene_SCENE_H__
```

Next, the result for TitleScene.cpp is as follows:

```
#include "TitleScene.h"

USING_NS_CC;

Scene* TitleScene::createScene()
{
    auto scene = Scene::create();
    auto layer = TitleScene::create();
    scene->addChild(layer);
    return scene;
}

// on "init" you need to initialize your instance
bool TitleScene::init()
{
    if ( !Layer::init() )
    {
        return false;
    }

    return true;
}
```

3. Next, add a label to the difference between TitleScene and HelloWorldScene. Add it before the return line in the TitleScene::init method as follows:

```
bool TitleScene::init()
{
    if ( !Layer::init() )
    {
        return false;
    }

    auto size = Director::getInstance()->getWinSize();
```

```
        auto label =
        Label::createWithSystemFont("TitleScene", "Arial",
        40);
        label->setPosition(size/2);
        this->addChild(label);

        return true;
    }
}
```

4. Similarly, add the label in the `HelloWorld::init` method.

```
bool HelloWorld::init()
{
    if ( !Layer::init() )
    {
        return false;
    }

    auto size = Director::getInstance()->getWinSize();
    auto label = Label::createWithSystemFont("HelloWorld",
    "Arial", 40);
    label->setPosition(size/2);
    this->addChild(label);

    return true;
}
}
```

5. Next, to display the `TitleScene` class, change `AppDelegate.cpp` as follows:

```
#include "TitleScene.h"

bool AppDelegate::applicationDidFinishLaunching() {
    // initialize director
    auto director = Director::getInstance();
    auto glview = director->getOpenGLView();
    if(!glview) {
        glview = GLViewImpl::create("My Game");
        director->setOpenGLView(glview);
    }

    // turn on display FPS
    director->setDisplayStats(true);

    // set FPS. the default value is 1.0/60 if you don't
    call this director->setAnimationInterval(1.0 / 60);
    glview->setDesignResolutionSize(640, 960,
    ResolutionPolicy::NO_BORDER);
}
```

```
// create a scene. it's an autorelease object
auto scene = TitleScene::createScene();

// run
director->runWithScene(scene);

return true;
}
```

The result is shown in the following image:



How it works...

First, you need to create `TitleScene` by duplicating the `HelloWorldScene` class files. It is pretty difficult to create an original `Scene` class from a blank file. However, a basic class of `Scene` is patterned. So, you can create it easily by duplicating and modifying the `HelloWorldScene` class files. While you are developing your game, you need to execute this step when you need a new scene.

Finally, we change the `AppDelegate.cpp` file. The `AppDelegate` class is the first class to be executed in Cocos2d-x. The `AppDelegate::applicationDidFinishLaunching` method is executed when the application is ready to run. This method will prepare the execution of Cocos2d-x. Then, it will create the first scene and run it.

```
auto scene = TitleScene::createScene();
// run
director->runWithScene(scene);
```

The `TitleScene::createScene` method is used to create a title scene, and the `runWithScene` method is used to run it.

Transitioning between scenes

Your games have to transition between scenes. For example, after launching your games, the title scene is displayed. Then, it is transitioned into the level selection scene, game scene, and so on. In this recipe, we will explain how to facilitate transition between scenes, which would improve the gameplay and the flow of the game.

How to do it...

A game has a lot of scenes. So, you might need to move between scenes in your game. Perhaps, when a game is started, a title scene will be displayed. Then, a game scene will appear in the next title scene. There are two ways to transition to a scene.

1. One is to use the `Director::replaceScene` method. This method replaces a scene outright.

```
auto scene = HelloWorld::createScene();
Director::getInstance()->replaceScene(scene);
```

2. The other is to use the `Director::pushScene` method. This method suspends the execution of the running scene and pushes a new scene on the stack of the suspended scene.

```
auto scene = HelloWorld::createScene();
Director::getInstance()->pushScene(scene);
```

In this case, the old scene is suspended. You can get back to the old scene to pop up a new scene.

```
auto Director::getInstance()->popScene();
```

How it works...

`Layer`, `Sprite`, and other nodes can be displayed by using the `addChild` method. However, `Scene` cannot be displayed by using the `addChild` method; it can be displayed by using the `Director::replaceScene` or `Director::pushScene` methods. That's why `Scene` is visible only on one screen at the same time. `Scene` and `Layer` are similar, but there is a significant difference.

Usually, you will use the `replaceScene` method when you change the scene from the title scene to the game scene. Further, you can use the `pushScene` method to display a modal scene, as in the case of pausing a scene during a game. In this case, an easy way to suspend a game scene is to pause the game.



When scenes are replaced in a game, the applications will release the memory used by the old scenes. However, if games push scenes, they will not release the memory used by the old scenes because it will suspend it. Further, games are resumed when new scenes are popped. If you added a lot of scenes by using the `pushScene` method, the device memory will no longer be enough.

Transitioning scenes with effects

Popular games display some effects when transitioning scenes. These effects can be natural, dramatic, and so on. Cocos2d-x has a lot of transitioning effects. In this recipe, we will explain how to use a transitioning effect and the effect produced.

How to do it...

You can add visual effects to a scene transition by using the `Transition` class. Cocos2d-x has many kinds of `Transition` classes. However, there is only one pattern for how to use them.

```
auto nextScene = HelloWorld::createScene();
auto transition = TransitionFade::create(1.0f, nextScene);
Director::getInstance()->replaceScene(transition);
```

It can be used when a scene was pushed.

```
auto nextScene = HelloWorld::createScene();
auto transition = TransitionFade::create(1.0f, nextScene);
Director::getInstance()->pushScene(transition);
```

How it works...

Firstly, you need to create the `nextscene` object. Then, you need to create a `transition` object with a set duration and an incoming scene object. Lastly, you need to run `Director::pushScene` with the `transition` object. This recipe sets the duration for the transition scene and the fade action as one second. The following table lists some of the major `Transition` classes:

Transition Class	Description
<code>TransitionRotoZoom</code>	Rotates and zooms out of the outgoing scene, and then, rotates and zooms into the incoming scene.
<code>TransitionJumpZoom</code>	Zooms out and jumps the outgoing scene, and then jumps and zooms into the incoming scene.

Transition Class	Description
TransitionMoveInL	Moves scene in from right to the left.
TransitionSlideInL	Slides in the incoming scene from the left border.
TransitionShrinkGrow	Shrinks the outgoing scene while enlarging the incoming scene.
TransitionFlipX	Flips the screen horizontally.
TransitionZoomFlipX	Flips the screen horizontally by doing a zoom out/in. The front face shows the outgoing scene, and the back face shows the incoming scene.
TransitionFlipAngular	Flips the screen half horizontally and half vertically.
TransitionZoomFlipAngular	Flips the screen half horizontally and half vertically by zooming out/in a little.
TransitionFade	Fades out of the outgoing scene, and then, fades into the incoming scene.
TransitionCrossFade	Cross-fades two scenes by using the <code>RenderTarget</code> object.
TransitionTurnOffTiles	Turns off the tiles of the outgoing scene in an random order.
TransitionSplitCols	The odd columns go upwards, while the even columns go downwards.
TransitionSplitRows	The odd rows go to the left, while the even rows go to the right.
TransitionFadeTR	Fades the tiles of the outgoing scene from the left-bottom corner to the top-right corner.
TransitionFadeUp	Fades the tiles of the outgoing scene from the bottom to the top.
TransitionPageTurn	Peels back the bottom right-hand corner of a scene to transition to the scene beneath it, thereby simulating a page turn.
TransitionProgressRadialCW	Counterclockwise radial transition to the next scene.

There's more...

You can also learn the beginning and the end of the transition scene by using the `onEnterTransitionDidFinish` method and the `onExitTransitionDidStart` method. When your game shows the new scene completely, the `onEnterTransitionDidFinish` method is called. When the old scene starts disappearing, the `onExitTransitionDidStart` method is called. If you'd like to do something during the time that the scenes appear or disappear, you will need to use these methods.

Let's now look at an example of using the `onEnterTransitionDidFinish` and `onExitTransitionDidStart` methods. `HelloWorldScene.h` has the following code:

```
class HelloWorld : public cocos2d::Layer
{
public:
    static cocos2d::Scene* createScene();
    virtual bool init();
    CREATE_FUNC(HelloWorld);

    virtual void onEnterTransitionDidFinish();
    virtual void onExitTransitionDidStart();
};
```

`HelloWorldScene.cpp` has the following code:

```
void HelloWorld::onEnterTransitionDidFinish()
{
    CCLOG("finished enter transition");
}

void HelloWorld::onExitTransitionDidStart()
{
    CCLOG("started exit transition");
}
```

Making original transitions for replacing scenes

You know that Cocos2d-x has a lot of transitioning effects. However, if it does not have an effect that you need, it is difficult to create an original transitioning effect. However, you can create it if you have the basic knowledge of transitioning effects. In this recipe, we will show you how to create original transitions.

How to do it...

Even though Cocos2d-x has a lot of different types of `Transition` classes, you may not find a transition effect that you need. In this recipe, you can create an original transition effect such as opening a door. When the replacement of a scene begins, the previous scene is divided into two and open to the left or right.

You have to create new files named `"TransactionDoor.h"` and `"TransactionDoor.cpp,"` and add them to your project.

TransactionDoor.h has the following code:

```
#ifndef __TRANSITIONDOOR_H__
#define __TRANSITIONDOOR_H__

#include "cocos2d.h"

NS_CC_BEGIN

class CC_DLL TransitionDoor : public TransitionScene , public
TransitionEaseScene
{
public:
    static TransitionDoor* create(float t, Scene* scene);

    virtual ActionInterval* action();
    virtual void onEnter() override;
    virtual ActionInterval * easeActionWithAction(ActionInterval *
action) override;
    virtual void onExit() override;
    virtual void draw(Renderer *renderer, const Mat4 &transform,
uint32_t flags) override;
    CC_CONSTRUCTOR_ACCESS:
    TransitionDoor();
    virtual ~TransitionDoor();

protected:
    NodeGrid* _gridProxy;
private:
    CC_DISALLOW_COPY_AND_ASSIGN(TransitionDoor);
};

class CC_DLL SplitDoor : public TiledGrid3DAction
{
public:
    /**
     * creates the action with the number of columns to split and
     the duration
     * @param duration in seconds
     */
    static SplitDoor* create(float duration, unsigned int cols);

    // Overrides
    virtual SplitDoor* clone() const override;
    /**
     * @param time in seconds
     */
}
```

```

    virtual void update(float time) override;
    virtual void startWithTarget(Node *target) override;

CC_CONSTRUCTOR_ACCESS:
    SplitDoor() {}
    virtual ~SplitDoor() {}

    /** initializes the action with the number of columns to split
    and the duration */
    bool initWithDuration(float duration, unsigned int cols);

protected:
    unsigned int _cols;
    Size _winSize;

private:
    CC_DISALLOW_COPY_AND_ASSIGN(SplitDoor);
};

NS_CC_END

#endif /* defined(__TRANSITIONDOOR_H_) */

```

Use the following code for `TransitionDoor.cpp`:

```

#include "TransitionDoor.h"

NS_CC_BEGIN

TransitionDoor::TransitionDoor()
{
    _gridProxy = NodeGrid::create();
    _gridProxy->retain();
}
TransitionDoor::~TransitionDoor()
{
    CC_SAFE_RELEASE(_gridProxy);
}

TransitionDoor* TransitionDoor::create(float t, Scene* scene)
{
    TransitionDoor* newScene = new (std::nothrow) TransitionDoor();
    if(newScene && newScene->initWithDuration(t, scene))
    {
        newScene->autorelease();
        return newScene;
    }
    CC_SAFE_DELETE(newScene);
}

```

```
    return nullptr;
}

void TransitionDoor::onEnter()
{
    TransitionScene::onEnter();

    _inScene->setVisible(true);

    _gridProxy->setTarget(_outScene);
    _gridProxy->onEnter();

    ActionInterval* split = action();
    ActionInterval* seq = (ActionInterval*)Sequence::create
    (
        split,
        CallFunc::create(CC_CALLBACK_0(TransitionScene::finish,this)),
        StopGrid::create(),
        nullptr
    );

    _gridProxy->runAction(seq);
}

void TransitionDoor::draw(Renderer *renderer, const Mat4
&transform, uint32_t flags)
{
    Scene::draw(renderer, transform, flags);
    _inScene->visit();
    _gridProxy->visit(renderer, transform, flags);
}

void TransitionDoor::onExit()
{
    _gridProxy->setTarget(nullptr);
    _gridProxy->onExit();
    TransitionScene::onExit();
}

ActionInterval* TransitionDoor::action()
{
    {
        return SplitDoor::create(_duration, 3);
    }
}

ActionInterval*
TransitionDoor::easeActionWithAction(ActionInterval * action)
{
    {
        return EaseInOut::create(action, 3.0f);
    }
}
```

```
}

SplitDoor* SplitDoor::create(float duration, unsigned int cols)
{
    SplitDoor *action = new (std::nothrow) SplitDoor();

    if (action)
    {
        if (action->initWithDuration(duration, cols))
        {
            action->autorelease();
        }
        else
        {
            CC_SAFE_RELEASE_NULL(action);
        }
    }

    return action;
}

bool SplitDoor::initWithDuration(float duration, unsigned int
cols)
{
    _cols = cols;
    return TiledGrid3DAction::initWithDuration(duration, Size(cols,
1));
}

SplitDoor* SplitDoor::clone() const
{
    // no copy constructor
    auto a = new (std::nothrow) SplitDoor();
    a->initWithDuration(_duration, _cols);
    a->autorelease();
    return a;
}

void SplitDoor::startWithTarget(Node *target)
{
    TiledGrid3DAction::startWithTarget(target);
    _winSize = Director::getInstance()->getWinSizeInPixels();
}

void SplitDoor::update(float time)
{
    for (unsigned int i = 0; i < _gridSize.width; ++i)
```

```

    {
        Quad3 coords = getOriginalTile(Vec2(i, 0));
        float direction = 1;

        if ( ( i % 2 ) == 0 )
        {
            direction = -1;
        }

        coords.bl.x += direction * _winSize.width/2 * time;
        coords.br.x += direction * _winSize.width/2 * time;
        coords.tl.x += direction * _winSize.width/2 * time;
        coords.tr.x += direction * _winSize.width/2 * time;

        setTile(Vec2(i, 0), coords);
    }
}
NS_CC_END

```

The following code will allow us to use the TransitionDoor effect:

```

auto trans = TransitionDoor::create(1.0f,
HelloWorld::createScene());
Director::getInstance()->replaceScene(trans);

```

How it works...

All types of transitions have TransitionScene as SuperClass. TransitionScene is a basic class and has a basic transition process. If you would like to create the original transition effect in an easier way, you would look for a similar transition effect in Cocos2d-x. You can then create your class from a similar class. The TransitionDoor class is created from the TransitionSplitCol class. Then, add and modify them where necessary. However, it is necessary to have basic knowledge about them in order to fix them.

The following are some of the important properties of the Transition class:

Properties	Description
_inScene	Pointer of the next scene.
_outScene	Pointer of the out scene.
_duration	Duration of the transition, a float value specified by the create method.
_isInSceneOnTop	Boolean value; if it is true, the next scene is the top of the scene graph.

Some of the important properties of the `transition` class are as follows:

Properties	Description
<code>onEnter</code>	To start the transition effect.
<code>Action</code>	To create an effect action.
<code>onExit</code>	To finish the transition effect and clean up.

In the case of the `TransitionDoor` class, the next scene is set to be visible and the previous scene is split into two grids in the `onEnter` method. Then, an effect such as opening a door is started. In the action method, an instance of the `Action` class is created by using the `SplitDoor` class. The `SplitDoor` class is based on the `SplitCol` class in Cocos2d-x. The `SplitDoor` class moves two grids of the previous scene to the left or the right.

There's more...

There are some necessary methods in addition to those described above. These methods are defined in the `Node` class.

Properties	Description
<code>onEnter</code>	Node starts appearing on the screen
<code>onExit</code>	Node disappears from the screen
<code>onEnterTransitionDidFinish</code>	Node finishes the transition effect after appearing on the screen
<code>onExitTransitionDidStart</code>	Node starts the transition effect before disappearing from the screen

If you want to play background music when the scene appears on the screen, you can play it by using the `onEnter` method. If you want to play it before finishing the transition effect, use the `onEnterTransitionDidFinish` method. Other than these, the initial process in the `onEnter` method starts the animation in the `onEnterTransitionDidFinish` method, cleans up the process in the `onExit` method, and so on.

Making original transitions for popping scenes

Cocos2d-x has transition effects for pushing a scene. For some reason, it does not have transition effects for popping scenes. We'd like to transition with an effect for popping scenes after pushing scenes with effects. In this recipe, we will explain how to create an original transition for popping scenes.

Getting ready

In this recipe, you will understand how to pop a transition scene with effects. You will need a new class, so you have to make new class files called `DirectorEx.h` and `DirectorEx.cpp` and add them to your project.

How to do it...

Cocos2d-x has a transition scene with effects for pushing scenes. However, it does not have transition effects for popping scenes. Therefore, we create an original class called `DirectorEx` to create a transition effect for popping scenes. The code snippet is given next.

`DirectorEx.h` has the following code:

```
class DirectorEx : public Director
{
public:
    Scene* previousScene(void);
    void popScene(Scene* trans);
};
```

`DirectorEx.cpp` has the following code:

```
#include "DirectorEx.h"

Scene* DirectorEx::previousScene()
{
    ssize_t sceneCount = _scenesStack.size();
    if (sceneCount <= 1) {
        return nullptr;
    }
    return _scenesStack.at(sceneCount-2);
}

void DirectorEx::popScene(Scene* trans)
{
    _scenesStack.popBack();
    ssize_t sceneCount = _scenesStack.size();
    if (sceneCount==0) {
        end();
    } else {
        _sendCleanupToScene = true;
        _nextScene = trans;
    }
}
```

This class can be used as follows:

```
DirectorEx* directorEx =
static_cast<DirectorEx*>(Director::getInstance());
Scene* prevScene = directorEx->previousScene();
Scene* pScene = TransitionFlipX::create(duration, prevScene);
directorEx->popScene(pScene);
```

How it works...

If we customized the `Director` class in Cocos2d-x, it can transition with the effect for popping a scene. However, this is not a good idea. Therefore, we create a sub-class of the `Director` class called the `DirectorEx` class and use this class as follows:

1. Firstly, you can get an instance of the `DirectorEx` class to cast an instance of the `Director` class.

```
DirectorEx* directorEx =
static_cast<DirectorEx*>(Director::getInstance());
```

2. Further, you have to get an instance of the previous scene.

```
Scene* prevScene = directorEx->previousScene();
```

3. Next, you have to create a transition effect.

```
Scene* pScene = TransitionFlipX::create(duration,
prevScene);
```

4. Finally, you can pop a scene with this effect by using the `DirectorEx::popScene` method.

```
directorEx->popScene(pScene);
```

Creating layers

A layer is an object that can be used on `Scene`. It is a transparent sheet similar to Photoshop's layer. All the objects are added to `Layer` in order to be displayed on the screen. Further, a scene can have multiple layers. Layers are also responsible for accepting inputs, drawing, and touching. For example, in the game, a scene has a background layer, hud layer, and a player's layer. In this recipe, we will explain how to use `Layer`.

How to do it...

The following code shows how to create a layer and add it to a scene:

```
auto layer = Layer::create();
this->addChild(layer);
```


That's easy. If you have a color layer, you can do it.

```
auto layer = LayerColor::create(Color4B::WHITE);
this->addChild(layer);
```

How it works...

The Scene class is the one displayed on the screen, but the Layer class can be stacked in many layers. Scene has one or more layers, and Sprite has to be on a layer. The Layer class is a transparent sheet. In addition, a transparent node needs more CPU power. So, you need to be careful not to stack too many layers.

Creating modal layers

In user interface design, a modal layer is an important layer. A modal layer is like a child window. When a modal layer is showing, players cannot touch any other button outside the modal layer. They can only touch the button on the modal layer. We will need modal layers when we confirm something with the players. In this recipe, we will explain how to create modal layers.

How to do it...

Firstly, you have to two new files named ModalLayer.h and ModalLayer.cpp. They should contain the following code:

ModalLayer.h should have the following code:

```
#include "cocos2d.h"

USING_NS_CC;

class ModalLayer : public Layer
{
public:
    ModalLayer();
    ~ModalLayer();
    bool init();
    CREATE_FUNC(ModalLayer);
    void close(Ref* sender=nullptr);
};
```

ModalLayer.cpp should have the following code:

```
#include "ModalLayer.h"
```

```

USING_NS_CC;

ModalLayer::ModalLayer()
{
}

ModalLayer::~~ModalLayer()
{
}

bool ModalLayer::init()
{
    if (!Layer::init())
    {
        return false;
    }

    auto listener = EventListenerTouchOneByOne::create();
    listener->setSwallowTouches(true);
    listener->onTouchBegan = [](Touch *touch, Event*event) ->bool{
        return true; };
    this->getEventDispatcher()->addEventListenerWithSceneGraphPriority(listener, this);

    return true;
}

void ModalLayer::close(Ref* sender)
{
    this->removeFromParentAndCleanup(true);
}

```

You should create a sub-class from the `ModalLayer` class and add a menu button or some design that you need. You then have to create an instance of it and add it to the running scene. Then, it should enable the buttons on the modal layer but disable the buttons at the bottom of the modal layer.

```

// add modal layer
auto modal = ModalLayer::create();
this->addChild(modal);

// close modal layer
modal->close();

```

How it works...

It is easy to create a modal layer in Cocos2d-x version 3. In version 3, a touch event occurs from the top of the layer. So, if the modal layer picks up all the touch events, the nodes under the modal layer are notified of these. The modal layer is picking up all of the events. Refer to the following code:

```
listener->onTouchBegan = [] (Touch *touch, Event*event) ->bool{  
    return true; };
```



This modal layer can pick up all touching events. However, Android has key events like the back key. When a player touches the back key when the modal layer is displayed, you have to decide to do it. In one of the cases, the modal is closed, and in another, the back key is ignored.

5

Creating GUIs

In this chapter, we're going to create various UI parts. The following topics will be covered in this chapter:

- ▶ Creating menus
- ▶ Creating buttons
- ▶ Creating checkboxes
- ▶ Creating loading bar
- ▶ Creating sliders
- ▶ Creating text fields
- ▶ Creating scroll views
- ▶ Creating page views
- ▶ Creating list views

Introduction

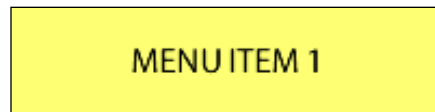
Games have a lot of GUI parts, for example, there are menus, buttons, checkboxes, loading bars, and so on. We cannot make our game without these parts. Further, these are a little different from the node we've discussed until now. In this chapter we will see how to create various GUI parts such as menus, sliders, text fields etc. for a game.

Creating menus

In this recipe, we will create a menu. A menu has various buttons such as a start button and a pause button. A Menu is a very important component for any game and they are really useful too. The steps to use a menu are little complex. In this recipe, we will have a glance over creating menus to understand its complexity and to get used to them.

Getting ready

We prepared the following image as a button image and added them to the `Resources/res` folder in our project. We will use the following image of the button to use it as menu:

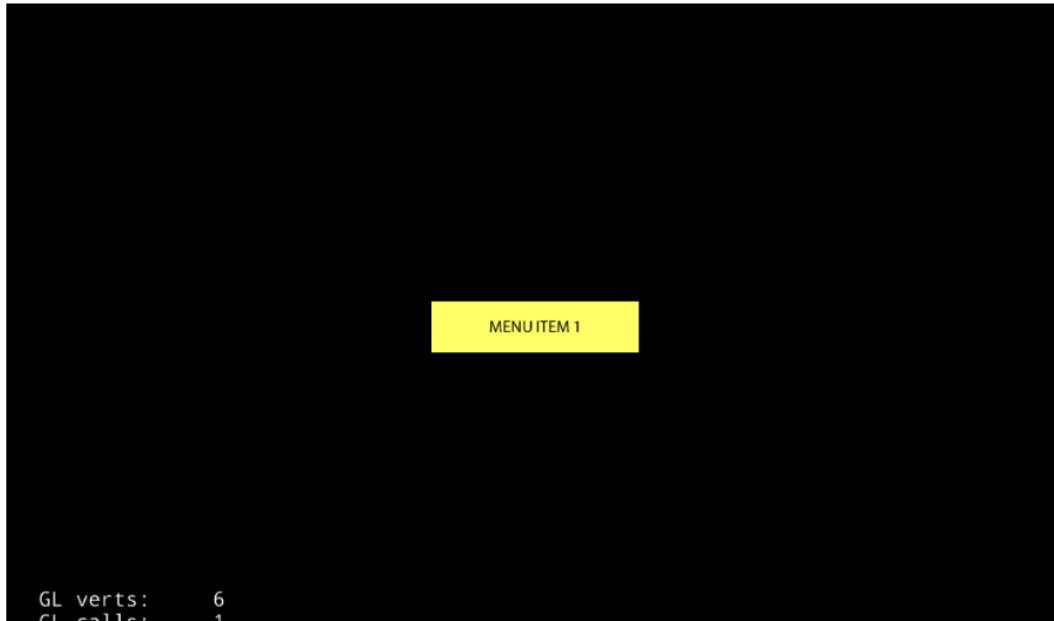


How to do it...

Firstly, we will create a simple menu that has one item for a button. We will use the `item1.png` file as the button image. Create the menu by using the code here.

```
auto normalItem = Sprite::create("res/item1.png");
auto selectedItem = Sprite::create("res/item1.png");
selectedItem->setColor(Color3B::GRAY);
auto item = MenuItemSprite::create(normalItem, selectedItem,
[] (Ref* sender){
    CCLOG("tapped item");
});
auto size = Director::getInstance()->getVisibleSize();
item->setPosition(size/2);
auto menu = Menu::create(item, nullptr);
menu->setPosition(Vec2());
this->addChild(menu);
```

The execution result of this code is shown in the following image:



Further, you can see the `tapped item` text in the log after tapping the menu item. You will notice that the button becomes a little dark when you tap it.

How it works...

1. Create a sprite of the normal status when the button is not operated.
2. Create a sprite of the selected status when the button is pressed. In this case, we used the same images for both the normal status and the selected status, but players could not understand the change in status when they tapped the button. That's why we changed the selected image to a slightly darker image by using the `setColor` method.
3. Create an instance of the `MenuItemSprite` class by using these two sprites. The third argument specifies the lambda expression to be processed when the button is pressed.

This time, we created only one button in the menu, but we can add more buttons in the menu. To do so, we can enumerate several items in the `Menu::create` method and specify `nullptr` at the end. To add multiple buttons in the menu, use the following code:

```
auto menu = Menu::create(item1, item2, item3, nullptr);
```

In addition, it is possible to add an item by using the `addChild` method of the menu instance.

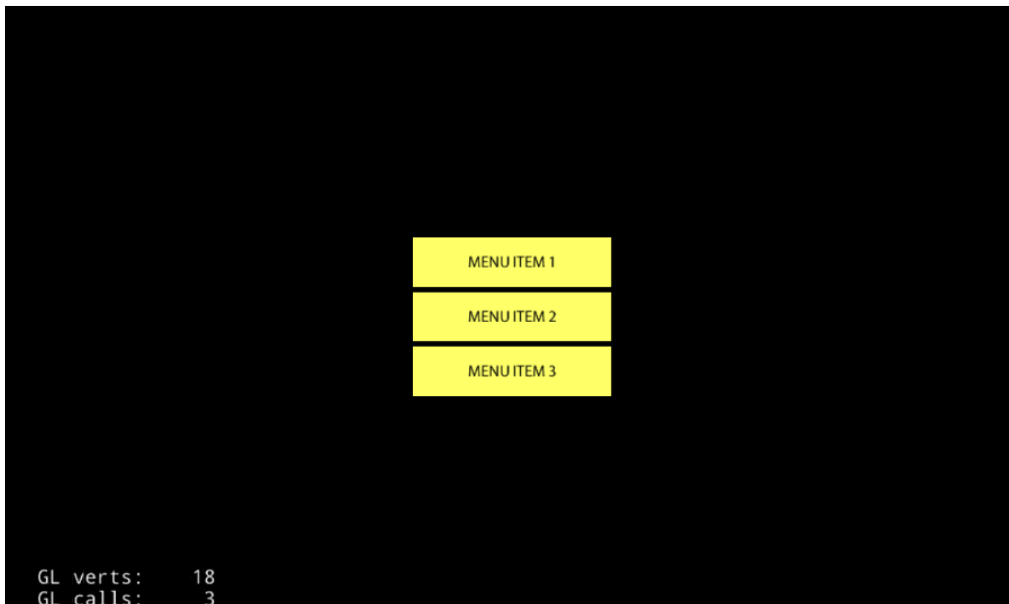
```
menu->addChild(item);
```

If the button is pressed, the lambda expression process that you specify when you create an instance of `MenuItemSprite` starts running. The argument is passed an instance of the `MenuItemSprite` that was pressed.

There's more...

It is also possible to automatically align multiple buttons. We created three items in the `Resources/res` folder. These are named `item1.png`, `item2.png`, and `item3.png`. You can create three buttons and use the following code to align these buttons vertically in the center of the screen:

```
Vector<MenuItem*> menuItems;
for (int i=1; i<=3; i++) {
    std::string name = StringUtils::format("res/item%d.png", i);
    auto normalItem = Sprite::create(name);
    auto selectedItem = Sprite::create(name);
    selectedItem->setColor(Color3B::GRAY);
    auto item = MenuItemSprite::create(normalItem, selectedItem,
[] (Ref* sender) {
    auto node = dynamic_cast<Node*>(sender);
    if (node!=nullptr) {
        CCLOG("tapped item %d", node->getTag());
    }
    });
    item->setTag(i);
    menuItems.pushBack(item);
}
auto size = Director::getInstance()->getVisibleSize();
auto menu = Menu::createWithArray(menuItems);
menu->setPosition(size/2);
menu->alignItemsVertically();
this->addChild(menu);
```



If you want to align these items horizontally, you can use the following code:

```
menu->alignItemsHorizontally();
```

Until now, the alignment of intervals has been adjusted automatically; however, if you want to specify the padding, you can use another method.

The following code will specify the intervals side by side in a vertical manner:

```
menu->alignItemsVerticallyWithPadding(20.0f);
```

The following code will specify the intervals side by side in a horizontal manner:

```
menu->alignItemsHorizontallyWithPadding(20.0f);
```

Creating buttons

In this recipe, we will explain how to create buttons. Before the `Button` class was released, we created a button by using the `Menu` class that was introduced in the previous recipe. Due to the `Button` class, it has become possible to finely control the button press.

Getting ready

To use the `Button` class and other GUI parts mentioned in this chapter, you must include the `CocosGUI.h` file. Let's add the following line of code in `HelloWorldScene.cpp`:

```
#include "ui/CocosGUI.h"
```

How to do it...

Let's create a button using the `Button` class. Firstly, you will generate a button instance by using `item1.png` image that was used in the previous recipe. We will also specify the callback function as a lambda expression by using the `addEventListener` method when the button is pressed. You can create the button by using the following code:

```
auto size = Director::getInstance()->getVisibleSize();
auto button = ui::Button::create("res/item1.png");
button->setPosition(size/2);
this->addChild(button);
button->addEventListener(
    [](Ref* sender, ui::Widget::TouchEvent type){
        switch (type) {
            case ui::Widget::TouchEvent::BEGAN:
                CLOG("touch began");
                break;
            case ui::Widget::TouchEvent::MOVED:
                CLOG("touch moved");
                break;
            case ui::Widget::TouchEvent::ENDED:
                CLOG("touch ended");
                break;
            case ui::Widget::TouchEvent::CANCELED:
                CLOG("touch canceled");
                break;

            default:
                break;
        }
    });
```

How it works...

You can now run this project and push the button. Further, you can move your touch position and release your finger. Thus, you will see that the touch status of the button will change in the log. Let's take a look at it step-by-step.

When you use the `Button` class and other GUI parts mentioned in this chapter, you have to include the `CocosGUI.h` file as this file defines the necessary classes. Further, please note that these classes have their own namespace such as `"cocos2d:ui."`

It is easy to create an instance of the `Button` class. You only need to specify the sprite file name. Further, you can create a callback function as a lambda expression by using the `addTouchListener` method. This function has two parameters. The first parameter is a button instance that was pressed. The second parameter is the touch status. Touch statuses are of four types. `TouchEventType::BEGAN` is the status at the moment that the button is pressed. `TouchEventType::MOVE` is the event type that occurs when you move your finger after you press it. `TouchEventType::ENDED` is the event that occurs at the moment you release your finger from the screen. `TouchEventType::CANCELED` is the event that occurs when you release your finger outside of the button.

There's more...

It is possible to create a button instance by specifying the selected status image and the disabled status image. Create this button by using the code here.

```
auto button = ui::Button::create(
    "res/normal.png",
    "res/selected.png",
    "res/disabled.png");
```

Unlike the `MenuItemSprite` class, you won't be able to specify the selection status by changing the normal image color that was set using the `setColor` method. You have to prepare the images as selected image and disabled image.

Creating checkboxes

In this recipe, we will create a checkbox. In Cocos2d-x version 2, a checkbox was created by using the `MenuItemToggle` class. However, doing so was quite cumbersome. In Cocos2d-x version 3, we can create a checkbox by using the `Checkbox` class that can be used in Cocos Studio.

Getting ready

So let's prepare the images of the checkbox before you start. Here, we have prepared the images of the required minimum `On` and `Off` status. Please add these images to the `Resources/res` folder.

The Off status image will look something like this:



The On status image will look something like this:

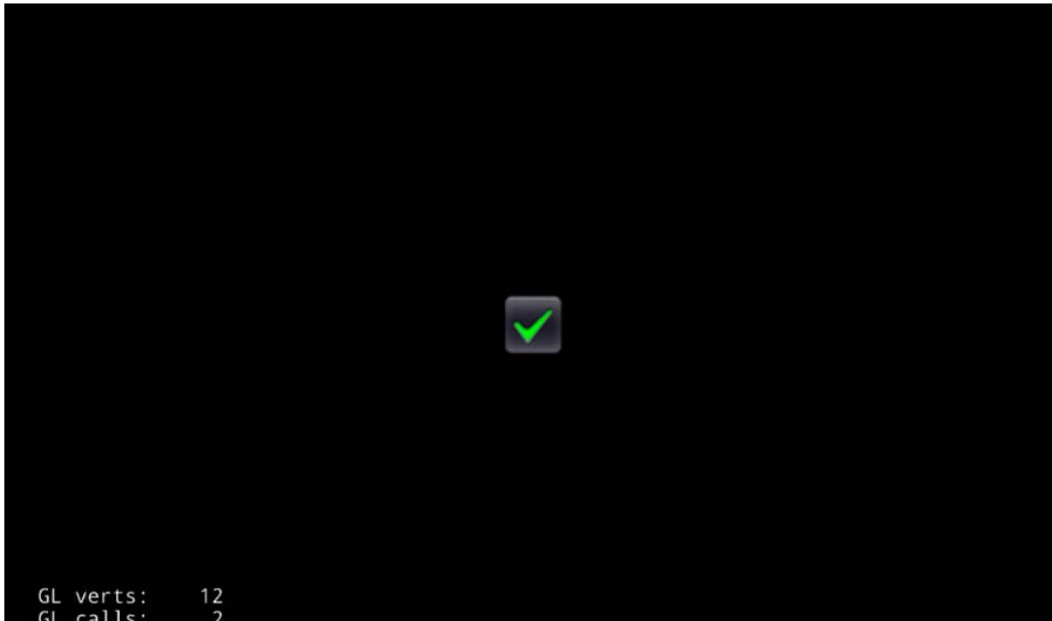


How to do it...

Let's create a checkbox by using the `CheckBox` class. First, you will generate a checkbox instance by using the `check_box_normal.png` image and the `check_box_active.png` image. You will also specify the callback function as a lambda expression by using the `addEventListener` method when the checkbox status is changed. Create the checkbox by using the following code:

```
auto size = Director::getInstance()->getVisibleSize();
auto checkbox = ui::CheckBox::create(
    "res/check_box_normal.png",
    "res/check_box_active.png");
checkbox->setPosition(size/2);
this->addChild(checkbox);
checkbox->addEventListener([](Ref* sender, ui::CheckBox::EventType
type){
    switch (type) {
        case ui::CheckBox::EventType::SELECTED:
            CLOG("selected checkbox");
            break;
        case ui::CheckBox::EventType::UNSELECTED:
            CLOG("unselected checkbox");
            break;
        default:
            break;
    }
});
```

The following figure shows that the checkbox was selected by running the preceding code.



How it works...

It generates the instance of a checkbox by specifying the `On` and `Off` images. Further, the callback function was specified in the same way as the `Button` class was in the previous recipe. A checkbox has two `EventType` options, namely `ui::Checkbox::EventType::SELECTED` and `ui::Checkbox::EventType::UNSELECTED`.

You can also get the status of the checkbox by using the `isSelected` method.

```
If (checkbox->isSelected()) {
    CLOG("selected checkbox");
} else {
    CLOG("unselected checkbox");
}
```

You can also change the status of the checkbox by using the `setSelected` method.

```
checkbox->setSelected(true);
```

There's more...

In addition, it is possible to further specify the image of a more detailed checkbox status. The `CheckBox::create` method has five parameters. These parameters are as follows:

- ▶ Unselected image
- ▶ Unselected and pushing image
- ▶ Selected image
- ▶ Unselected and disabled image
- ▶ Selected and disabled image

Here's how to specify the images of these five statuses:

```
auto checkbox = ui::CheckBox::create(
    "res/check_box_normal.png",
    "res/check_box_normal_press.png",
    "res/check_box_active.png",
    "res/check_box_normal_disable.png",
    "res/check_box_active_disable.png");
```

To disable the checkbox, use the following code:

```
checkbox->setEnabled(false);
```

Creating loading bars

When you are consuming a process or downloading something, you can indicate that it is not frozen by showing its progress to the user. To show such progresses, Cocos2d-x has a `LoadingBar` class. In this recipe, you will learn how to create and show the loading bars.

Getting ready

Firstly, we have to prepare an image for the progress bar. This image is called `loadingbar.png`. You will add this image in the `Resources/res` folder.



How to do it...

It generates an instance of the loading bar by specifying the image of the loading bar. Further, it is set to 0% by using the `setPercent` method. Finally, in order to advance the bar from 0% to 100% by 1% at 0.1 s, we will use the `schedule` method as follows:

```
auto loadingbar = ui::LoadingBar::create("res/loadingbar.png");
loadingbar->setPosition(size/2);
loadingbar->setPercent(0);
this->addChild(loadingbar);
this->schedule([=](float delta){
    float percent = loadingbar->getPercent();
    percent++;
    loadingbar->setPercent(percent);
    if (percent>=100.0f) {
        this->unsubscribe("updateLoadingBar");
    }
}, 0.1f, "updateLoadingBar");
```

The following figure is an image of the loading bar at 100%.



How it works...

You have to specify one image as the loading bar image to create an instance of the `LoadingBar` class. You can set the percentage of the loading bar by using the `setPercent` method. Further, you can get its percentage by using the `getPercent` method.

There's more...

By default, the loading bar will progress toward the right. You can change this direction by using the `setDirection` method.

```
loadingbar->setDirection(ui::LoadingBar::Direction::RIGHT);
```

When you set the `ui::LoadingBar::Direction::RIGHT` value, the start position of the loading bar is the right edge. Then, the loading bar will progress in the left direction.

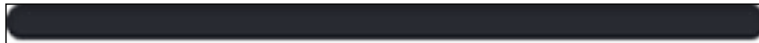
Creating sliders

In this recipe, we will explain the slider. The slider will be used for tasks such as changing the volume of the sound or music. Cocos2d-x has a `Slider` class for it. If we use this class, we can create a slider easily.

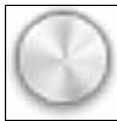
Getting ready

So, let's prepare the images of the slider before we start. Please add these images in the `Resources/res` folder.

- ▶ `sliderTrack.png`: Background of the slider



- ▶ `sliderThumb.png`: Image to move the slider

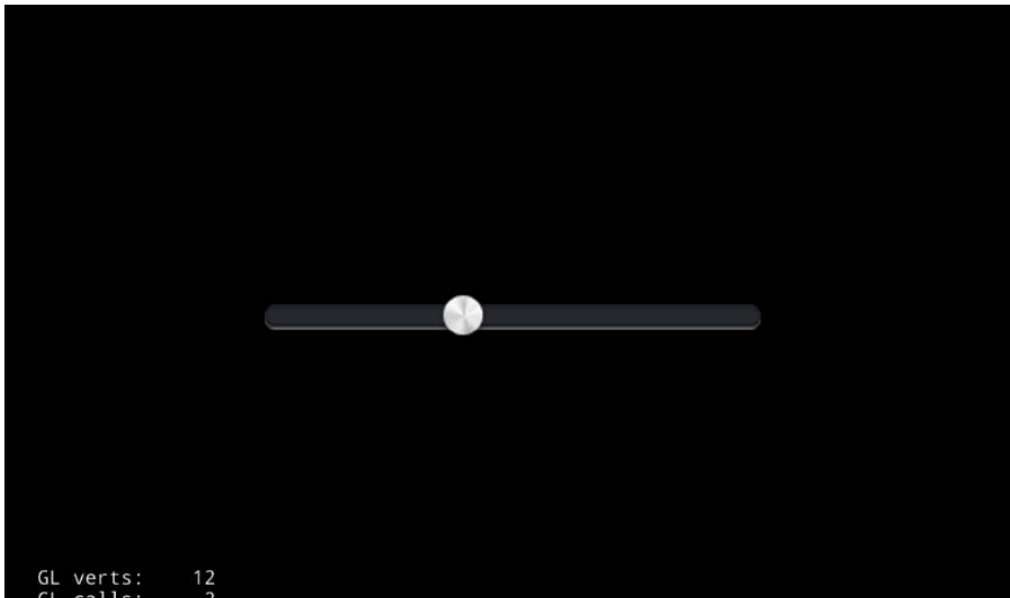


How to do it...

Let's create a slider by using the `Slider` class. First, you will generate a slider instance by using the `sliderTrack.png` image and the `sliderThumb.png` image. You will also specify the callback function as a lambda expression by using the `addEventListener` method when the slider value is changed.

```
auto slider = ui::Slider::create("res/sliderTrack.png",
    "res/sliderThumb.png");
slider->setPosition(size/2);
this->addChild(slider);
slider->addEventListener([](Ref* sender, ui::Slider::EventType
type) {
    auto slider = dynamic_cast<ui::Slider*>(sender);
    if (type==ui::Slider::EventType::ON_PERCENTAGE_CHANGED) {
        CCLOG("percentage = %d", slider->getPercent());
    }
});
```

The following figure shows the result of the preceding code.



How it works...

You have to specify two images as the slider's bar image and the slider's thumb image to create an instance of the `Slider` class. The callback function was specified in the same way as the `Button` class was in the previous recipe. The slider has only one `EventType` as `ui::Slider::EventType::ON_PERCENTAGE_CHANGED`. That's why the status is the only changing value. You can get the percentage shown on the slider by using the `getPercent` method.

There's more...

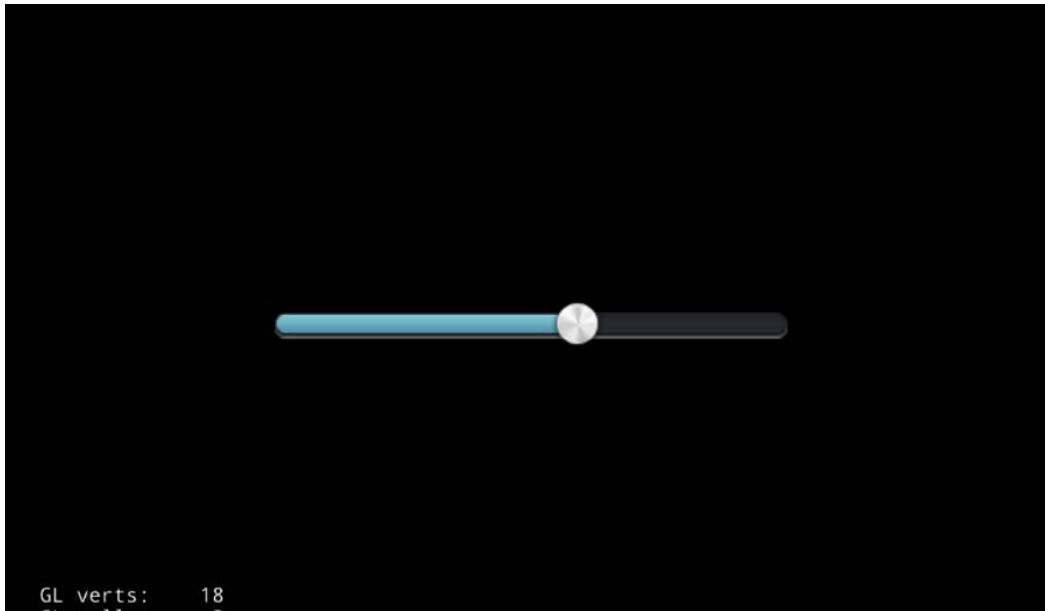
If you want to see the progress on the slider, you can use the `loadProgressBarTexture` method. We will require an image for the progress bar. The following image shows the progress bar image. Let's add it to the `Resources/res` folder.



Then, we use the `loadProgressbarTexture` method by specifying this image.

```
slider->loadProgressBarTexture("res/sliderProgress.png");
```

Let's run the code that has been modified so far. You will see it with the color on the left side of the bar as shown in the following screenshot:



Creating text fields

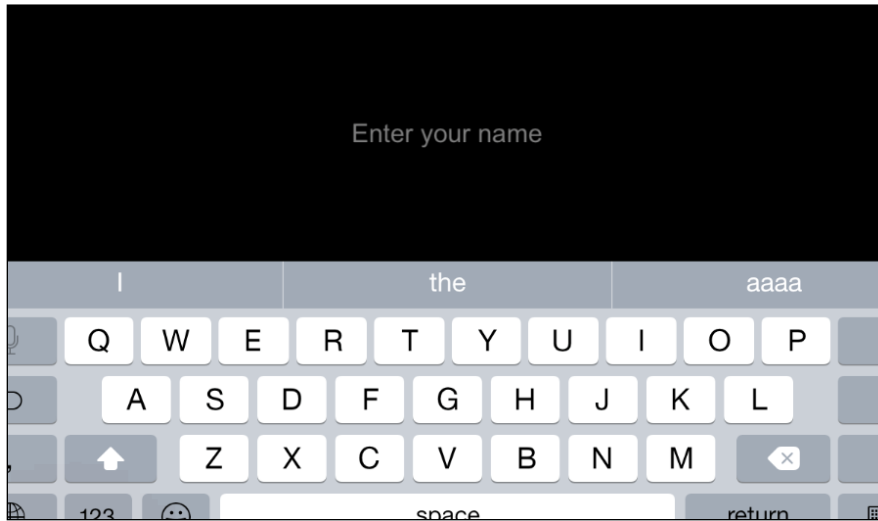
You may want to set a nickname in your game. To set nicknames or to get the player's input text, you can use the `TextField` class. In this recipe, we will learn about a simple `TextField` sample and how to add a textfield in a game.

How to do it...

You will create a text field by specifying the placeholder text, font name, and font size. Then, you set a callback function by using `addEventListener`. In the callback function, you can get the text that the player input in the `textField`. Create the `textField` by using the following code:

```
auto textField = ui::TextField::create("Enter your name", "Arial",
30);
textField->setPosition(Vec2(size.width/2, size.height*0.75f));
this->addChild(textField);
textField->addEventListener([](Ref* sender,
ui::TextField::EventType type){
    auto textField = dynamic_cast<ui::TextField*>(sender);
    switch (type) {
        case ui::TextField::EventType::ATTACH_WITH_IME:
            CCLOG("displayed keyboard");
            break;
        case ui::TextField::EventType::DETACH_WITH_IME:
            CCLOG("dismissed keyboard");
            break;
        case ui::TextField::EventType::INSERT_TEXT:
            CCLOG("inserted text : %s",
                textField->getString().c_str());
            break;
        case ui::TextField::EventType::DELETE_BACKWARD:
            CCLOG("deleted backward");
            break;
        default:
            break;
    }
});
```

Let's run this code. You will see it within the placeholder text, and it will show the keyboard automatically as shown in the following screenshot:



How it works...

1. You create an instance of the `TextField` class. The first argument is the placeholder string. The second argument is the font name. You can specify only a true type font. The third argument is the font size.
2. You can get the event by using the `addEventListener` method. The following list provides the event names and their descriptions:

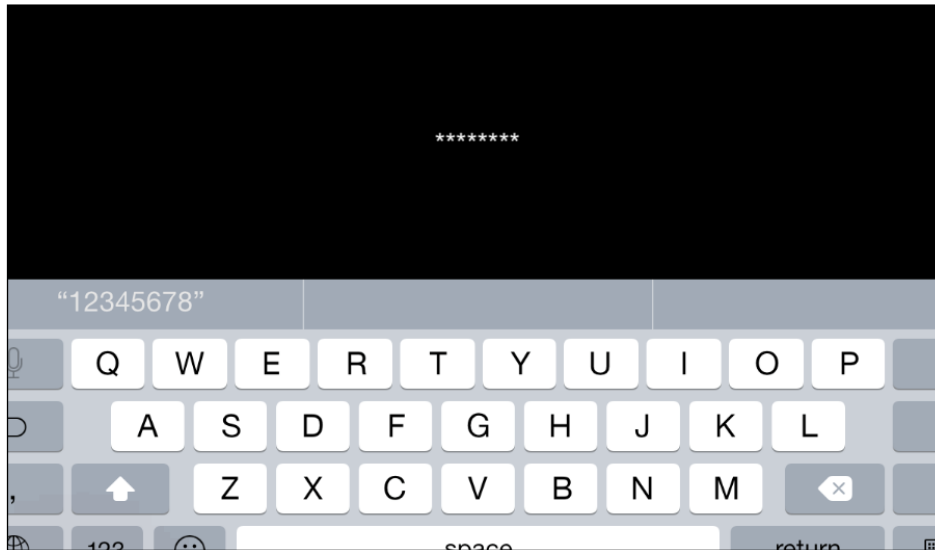
Event Name	Description
<code>ATTACH_WITH_IME</code>	The keyboard will appear.
<code>DETACH_WITH_IME</code>	The keyboard will disappear.
<code>INSERT_TEXT</code>	The text was input. You can get the string by using the <code>getString</code> method.
<code>DELETE_BACKWARD</code>	The text was deleted.

There's more...

When a player enters a password, you have to hide it by using the `setPasswordEnabled` method.

```
textField->setPasswordEnabled(true);
```

Let's run the code that has been modified so far. You will see how to hide a password that you enter, as shown in the following screenshot:



Creating scroll views

When a huge map is displayed in your game, a scroll view is required. It can be scrolled by a swipe, and bounce at the edge of the area. In this recipe, we explain the `ScrollView` class of Cocos2d-x.

How to do it...

Let's implement it right away. In this case, we doubled the size of `HelloWorld.png`. Further, we try to display this huge image in `ScrollView`. Create the scroll view by using the following code:

```
auto scrollView = ui::ScrollView::create();
scrollView->setPosition(Vec2());
scrollView->setDirection(ui::ScrollView::Direction::BOTH);
scrollView->setBounceEnabled(true);
this->addChild(scrollView);

auto sprite = Sprite::create("res/HelloWorld.png");
sprite->setScale(2.0f);
sprite->setPosition(sprite->getBoundingBox().size/2);
scrollView->addChild(sprite);
```

```
scrollView->setInnerContainerSize(sprite->getBoundingBox().size);  
scrollView->setContentSize(sprite->getContentSize());
```

Let's run this code. You will see the huge `HelloWorld.png` image. Further, you will see that you can scroll it by swiping.



How it works...

1. You create an instance of the `ScrollView` class by using the `create` method without arguments.
2. You set the direction of the scroll view by using the `setDirection` method. In this case, we want to scroll up and down, and left and right, so you should set `ui::ScrollView::Direction::BOTH`. This implies that we can scroll in both the vertical and the horizontal directions. If you want to scroll just up and down, you set `ui::ScrollView::Direction::VERTICAL`. If you want to scroll just left and right, you set `ui::ScrollView::Direction::HORIZONTAL`.
3. If you want to bounce when it is scrolled at the edge of the area, you should set `true` by using the `setBounceEnabled` method.
4. You will provide the content to be displayed in the scroll view. Here, we have used `HelloWorld.png` that has been scaled twice.

5. You have to specify the size of the content in the scroll view by using the `setInnerContainerSize` method. In this case, we specify double the size of `HelloWorld.png` in the `setInnerContainerSize` method
6. You have to specify the size of the scroll view by using the `setContentSize` method. In this case, we specify the original size of `HelloWorld.png` by using the `setContentSize` method.

Creating page views

A page view is similar to a scroll view, but it will be scrolled on a page-by-page basis. `PageView` is also a class in Cocos2d-x. In this recipe, we will explain how to use the `PageView` class.

How to do it...

Let's immediately get it implemented. Here, we will arrange three images of `HelloWorld.png` side-by-side in the page view. Create the page view by using the following code:

```
auto pageView = ui::PageView::create();
pageView->setPosition(Vec2());
pageView->setContentSize(size);
this->addChild(pageView);

for (int i=0; i<3; i++) {
    auto page = ui::Layout::create();
    page->setContentSize(pageView->getContentSize());

    auto sprite = Sprite::create("res/HelloWorld.png");
    sprite->setPosition(sprite->getContentSize()/2);
    page->addChild(sprite);
    pageView->insertPage(page, i);
}

pageView->addEventListener([](Ref* sender, ui::PageView::EventType
type){
    if (type==ui::PageView::EventType::TURNING) {
        auto pageView = dynamic_cast<ui::PageView*>(sender);
        CCLOG("current page no =%zd",
            pageView->getCurPageIndex());
    }
});
```

When you run this code, you will see one `HelloWorld.png`. You will see that you can move to the next page by using a swiping movement.

How it works...

Create an instance of the `PageView` class by using the `create` method without arguments. Here, we set it as the same size as that of the screen.

Display three `HelloWorld.png` images side-by-side. You must use the `Layout` class to set the page layout in `PageView`.

Set the page size and add the image by using the `addChild` method.

Insert an instance of the `Layout` class to the page view by using the `insertPage` method. At this time, you specify the page number as the second argument.

Get the event when the page has changed, you use the `addEventListener` method. `PageView` has only one event, `PageView::EventType::TURNING`. You can get the current page number by using the `getCurPageIndex` method.

Creating list views

`ListView` is a class in `Cocos2d-x`. It is like `UITableView` for iOS or `List View` for Android. `ListView` is useful for creating a lot of buttons as required in the case of setting a scene. In this recipe, we will explain how to use the `ListView` class.

How to do it...

Here, we try to display `ListView` that has 20 buttons. Each button is identified with a number such as "list item 10." In addition, we display the number of the button that you selected on the log when you tap any button. Create the list view by using the following code:

```
auto listView = ui::ListView::create();
listView->setPosition(Vec2(size.width/2 - 200, 0.0f));
listView->setDirection(ui::ListView::Direction::VERTICAL);
listView->setBounceEnabled(true);
listView->setContentSize(size);
this->addChild(listView);

for (int i=0; i<20; i++) {
    auto layout = ui::Layout::create();
    layout->setContentSize(Size(400, 50));
    layout->setBackgroundColorType(ui::Layout::BackgroundColorType::SOLID);
    layout->setBackgroundColor(Color3B::WHITE);

    auto button = ui::Button::create();
    button->setPosition(layout->getContentSize()/2);
    std::string name = StringUtils::format("list item %d", i);
```

```

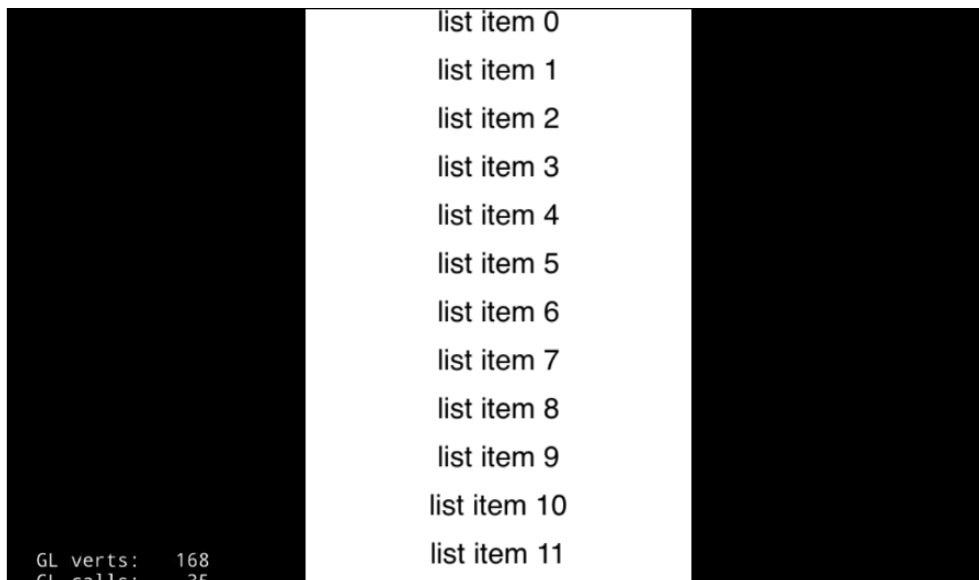
        button->setTitleText(name);
        button->setTitleFontSize(30);
        button->setTitleColor(Color3B::BLACK);

        layout->addChild(button);
        listView->addChild(layout);
    }

    listView->addEventListener([](Ref* sender, ui::ListView::EventType
type){
        auto listView = dynamic_cast<ui::ListView*>(sender);
        switch (type) {
            case ui::ListView::EventType::ON_SELECTED_ITEM_START:
                CCLOG("select item started");
                break;
            case ui::ListView::EventType::ON_SELECTED_ITEM_END:
                CCLOG("selected item : %zd", listView-
>getCurSelectedIndex());
                break;
            default:
                break;
        }
    });

```

When you run this code, you will see some buttons. You will see that you can scroll it by swiping and you can get the number of the button you tapped.



How it works...

1. Create an instance of the `ListView` class. It is possible to specify the scroll direction in the same way as `ScrollView`. Since we want to scroll only in the vertical direction, you specify `ui::ListView::Direction::VERTICAL`. Also, you can specify the bounce at the edge of the area by using the `setBounceEnabled` method.
2. Create 20 buttons to display in the list view. You have to use the `Layout` class to display the content in the list view as in the case of `PageView`. You add an instance of the `Button` class to the instance of the `Layout` class.
3. Get the event by using the `addEventListener` method. `ListView` has two events, namely `ON_SELECTED_ITEM_START` and `ON_SELECTED_ITEM_END`. When you touch the list view, `ON_SELECTED_ITEM_START` is fired. When you release the finger without moving it, `ON_SELECTED_ITEM_END` is fired. If you move your finger, `ON_SELECTED_ITEM_END` is not fired and it will be a scrolling process. You can get the button number by using the `getCurSelectedIndex` method.

6

Playing Sounds

A game without sound will be boring and lifeless. Background music and sound effects that suit the visuals will lighten up the game. Initially, we used a very famous audio engine called `SimpleAudioEngine`, but Cocos2d-x version 3.3 has now come up with an all-new `AudioEngine`. In this chapter, we're going to talk about both `SimpleAudioEngine` and `AudioEngine`. The following topics will be covered in this chapter:

- ▶ Playing background music
- ▶ Playing a sound effect
- ▶ Controlling volume, pitch, and balance
- ▶ Pausing and resuming background music
- ▶ Pausing and resuming sound effects
- ▶ Playing background music and a sound effect by using `AudioEngine`
- ▶ Playing movies

Playing background music

By using `SimpleAudioEngine`, we can play background music very easily. `SimpleAudioEngine` is a shared singleton object that can be called from anywhere in your code. In `SimpleAudioEngine`, we can play only one background score.

Getting ready

We have to include the header file of `SimpleAudioEngine` to use it. Therefore, you will need to add the following code:

```
#include "SimpleAudioEngine.h"
```

How to do it...

The following code is used to play background music called `background.mp3`.

```
auto audio = CocosDenshion::SimpleAudioEngine::getInstance();
audio->preloadBackgroundMusic("background.mp3");

// play the background music and continuously play it.
audio->playBackgroundMusic("background.mp3", true);
```

How it works...

`SimpleAudioEngine` has a namespace called `CocosDenshion`. For `SimpleAudioEngine`, you just have to get an instance by using the `getInstance` method. You can play the background music without preloading it, but this could result in a delay in playback. That's why you should preload the music before playing it. If you want the playback to be continuous, you need to set the `true` value as the second argument.

There's more...

`SimpleAudioEngine` supports a number of formats, including MP3 and Core Audio format. It can play the following formats:

Format	iOS (BGM)	iOS (SE)	Android (BGM)	Android (SE)
IMA (.caf)	○	○	□	□
Vorbis (.ogg)	□	□	○	○
MP3 (.mp3)	○	○	○	○
WAVE (.wav)	○	○	△	△

If you want to play a sound in a different format on iOS and Android, you can play it by using the following macro code:

```
#if (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)
#define MUSIC_FILE      "background.ogg"
#else
#define MUSIC_FILE      "background.caf"
#endif

audio->playBackgroundMusic(MUSIC_FILE, true);
```

In this code, if the device is Android, it plays a `.ogg` file. If the device is iOS, it plays a `.caf` file.

Playing a sound effect

By using `SimpleAudioEngine`, we can play sound effects; to play them, we need to perform only two steps, namely preload and play. Sound effects are not background music; note that we can play multiple sound effects but only one background score at the same time. In this recipe, we will explain how to play sound effects.

Getting ready

As in the case of playing background music, you have to include a header file for `SimpleAudioEngine`.

```
#include "SimpleAudioEngine.h"
```

How to do it...

Let's try to immediately play a sound effect. The audio format is changed depending on the operating system by using the macro that was introduced at the time of playing the background music. The code for playing sound effects is as follows:

```
#if (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)
#define EFFECT_FILE      "effect.ogg"
#else
#define EFFECT_FILE      "effect.caf"
#endif

auto audio = CocosDenshion::SimpleAudioEngine::getInstance();
audio->preloadEffect( EFFECT_FILE );
audio->playEffect( EFFECT_FILE );
```

How it works...

The overall flow is the same as that for playing background music. You need to preload a sound effect file before playing it. The sound effect file is smaller than the background music file. So, you can preload a lot of sound effects before playing them.

There's more...

The number of sound effects that we can play at the same time on Android is less than that on iOS. So, we will now explain how to increase this number for Android. The maximum number of simultaneous playbacks is defined in `Cocos2dxSound.java`.

The path of `Cocos2dxSound.java` is `cocos2d/cocos/platform/android/java/src/org/cocos2dx/lib`. Then, in line 66, the maximum number of simultaneous playbacks is defined.

```
private static final int MAX_SIMULTANEOUS_STREAMS_DEFAULT = 5;
```

If we changed this value to 10, we can play 10 sound effects at the same time.

Controlling volume, pitch, and balance

You can control the volume, pitch, and balance for sound effects. The right blend of these three factors makes your game sound more fun.

How to do it...

Let's try to immediately play a sound effect by controlling its volume, pitch, and balance. The following is the code snippet to do so:

```
auto audio = CocosDenshion::SimpleAudioEngine::getInstance();
// set volume
audio->setEffectsVolume(0.5);

// set pitch, pan, gain with playing a sound effect.
float pitch = 1.0f;
float pan = 1.0f;
float gain = 1.0f;
audio->playEffect(EFFECT_FILE, true, pitch, pan, gain);
```

How it works...

You can control the volume for sound effects by using the `setEffectsVolume` method. The maximum value for the volume is 1.0, and the minimum value is 0.0. If you set the volume to 0.0, the sound effect is muted. The default value of the volume is 1.0.

You can play multiple sound effects at the same time, but you cannot set the volume for these effects individually. To change the master volume for sound effects, set a volume by using the `setEffectsVolume` method. If you want to change the volume individually, you should use a `gain` value; which we will explain later.

The second argument in the `playEffect` method is the flag for continuously playing the sound effects. For the third and the subsequent arguments, please check the following table:

Arguments	Description	Minimum	Maximum
Third argument (<code>pitch</code>)	Playing speed	0.0	2.0
Fourth argument (<code>pan</code>)	Balance of left and right	-1.0	1.0
Fifth argument (<code>gain</code>)	Distance from a sound source	0.0	1.0

The `pitch` is the quality that allows us to classify a sound as relatively high or low. By using this `pitch`, we can control the playing speed in the third argument. If you set the `pitch` to less than 1.0, the sound effect is played slowly. If you set it to more than 1.0, the sound effect is played quickly. If you set it to 1.0, the sound effect plays at the original speed. The maximum value of the `pitch` is 2.0. However, you can set the `pitch` to more than 2.0 in iOS. On the other hand, the maximum value of the `pitch` in Android is 2.0. Therefore, we adopted the maximum value as the lower.

You can change the balance of the left and the right speakers by changing the `pan` in the fourth argument. If you set it to -1.0, you can hear it only from the left speaker. If you set it to 1.0, you can hear it from only the right speaker. The default value is 0.0; you can hear it at the same volume from both the left and the right speakers. Unfortunately, you will not be able to figure out much difference in the speaker of the device. If you use the headphones, you can hear this difference.

You can change the volume of each sound effect by changing the `gain` in the fifth argument. You can set the master volume by using the `setEffectVolume` method and the volume of each effect by changing the `gain` value. If you set it to 0.0, its volume is mute. If you set it to 1.0, its volume is the maximum. The final volume of the sound effects will be a combination of the `gain` value and the value specified in the `setEffectsVolume` method.

Pausing and resuming background music

This recipe will help you better understand the concept of pausing and resuming background music.

How to do it...

It is very easy to stop or pause the background music. You don't specify the argument by using these methods. The code for stopping the background music is as follows:

```
auto audio = CocosDenshion::SimpleAudioEngine::getInstance();
// stop the background music
audio->stopBackgroundMusic();
```

Code for pausing:

```
// pause the background music
audio->pauseBackgroundMusic();
```

Code for resuming the paused background music:

```
// resume the background music
audio->resumeBackgroundMusic();
```

How it works...

You can stop the background music that is playing by using the `stopBackgroundMusic` method. Alternatively, you can pause the background music by using the `pauseBackgroundMusic` method. Once you stop it, you can play it again by using the `playBackgroundMusic` method. Further, if you pause it, you can resume playing the music by using the `resumeBackgroundMusic` method.

There's more...

You can determine whether the background music is playing by using the `isBackgroundMusicPlaying` method. The following code can be used for doing so:

```
auto audio = CocosDenshion::SimpleAudioEngine::getInstance();
if (audio->isBackgroundMusicPlaying()) {
    // background music is playing
} else {
    // background music is not playing
}
```



However, you are required to be careful while using this method. This method always returns a true value that specifies the playing status in the iOS simulator. At line 201 of `audio/ios/CDAudioManager.m` in Cocos2d-x, if the device is the iOS simulator, `SimpleAudioEngine` sets the volume to zero and plays it continuously. That's why there is a problem in the iOS simulator. However, we tested the latest iOS simulator before commenting out this process and found that there was no problem. If you want to use this method, you should comment out this process.

Pausing and resuming sound effects

You might want to stop sound effects too. Also, you may want to pause them and then resume them.

How to do it...

It is very easy to stop or pause a sound effect. The following is the code for stopping it:

```
auto audio = CocosDenshion::SimpleAudioEngine::getInstance();
```

```
unsigned int _soundId;
// get the sound id as playing the sound effect
_soundId = audio->playEffect(EFFECT_FILE);
// stop the sound effect by specifying the sound id
audio->stopEffect(_soundId);
```

The following is the code for pausing it:

```
// pause the sound effect
audio->pauseEffect(_soundId);
```

You can resume the paused code as follows:

```
// resume the sound effect
audio->resumeEffect(_soundId);
```

How it works...

`SimpleAudioEngine` can play multiple sound effects. Therefore, you have to specify the sound effect if you want to stop or pause it individually. You can get the sound ID when you play the sound effect. You can stop, pause, or resume the specific sound effect by using this ID.

There's more...

You can stop, pause, or resume all the playing sound effects. The code to do so is as follows:

```
auto audio = CocosDenshion::SimpleAudioEngine::getInstance();
// stop all sound effects
audio->stopAllEffects();
// pause all sound effects
audio->pauseAllEffects();
// resume all sound effects
audio->resumeAllEffects();
```

Playing background music and a sound effect by using `AudioEngine`

`AudioEngine` is a new class from Cocos2d-x version 3.3. `SimpleAudioEngine` cannot play multiple background scores, but `AudioEngine` can play them. Furthermore, `AudioEngine` can call a callback function when it finishes playing the background music. In addition, we can get the playtime by using the callback function. In this recipe, we will learn more about the brand new `AudioEngine`.

Getting ready

We have to include the header file of `AudioEngine` to use it. Further, `AudioEngine` has a namespace called `experimental`. To include the header file, you will need to add the following code:

```
#include "audio/include/AudioEngine.h"
USING_NS_CC;
using namespace experimental;
```

How to do it...

`AudioEngine` is much easier than `SimpleAudioEngine`. Its API is very simple. The following code can be used to play, stop, pause, and resume the background music.

```
// play the background music
int id = AudioEngine::play2d("sample_bgm.mp3");
// set continuously play
AudioEngine::setLoop(id, true);
// change the volume, the value is from 0.0 to 1.0
AudioEngine::setVolume(id, 0.5f);
// pause it
AudioEngine::pause(id);
// resume it that was pausing
AudioEngine::resume(id);
// stop it
AudioEngine::stop(id);
// seek it by specifying the time
AudioEngine::setCurrentTime(int id, 12.3f);
// set the callback when it finished playing it
AudioEngine::setFinishCallback(int id, [](int audioId, std::string
filePath){
    // this is the process when the background music was finished.
});
```

How it works...

`AudioEngine` no longer needs the `preload` method. Further, `AudioEngine` does not distinguish between background music and sound effects. You can play both background music and sound effects by using the same method. When you play it, you can get a sound ID as the return value. You have to specify the sound ID when you change the volume, stop it, pause it, and so on.

There's more...

If you want to unload audio files from the memory, you can `uncache` by using the `AudioEngine::uncache` method or the `AudioEngine::uncacheAll` method. In the case of the `uncache` method, you have to specify the path that you want to unload. In the case of the `uncacheAll` method, all audio data is unloaded from the memory. While unloading files, you have to stop the related music and sound effects.

Playing movies

You might want to play a movie in your game in order to enrich the representation. Cocos2d-x provides a `VideoPlayer` class for this purpose. This class makes it easy to play a movie; however, it is still an `experimental` class. So, you have to be very careful while using it.

Getting ready

You have to prepare something before using the `VideoPlayer` class.

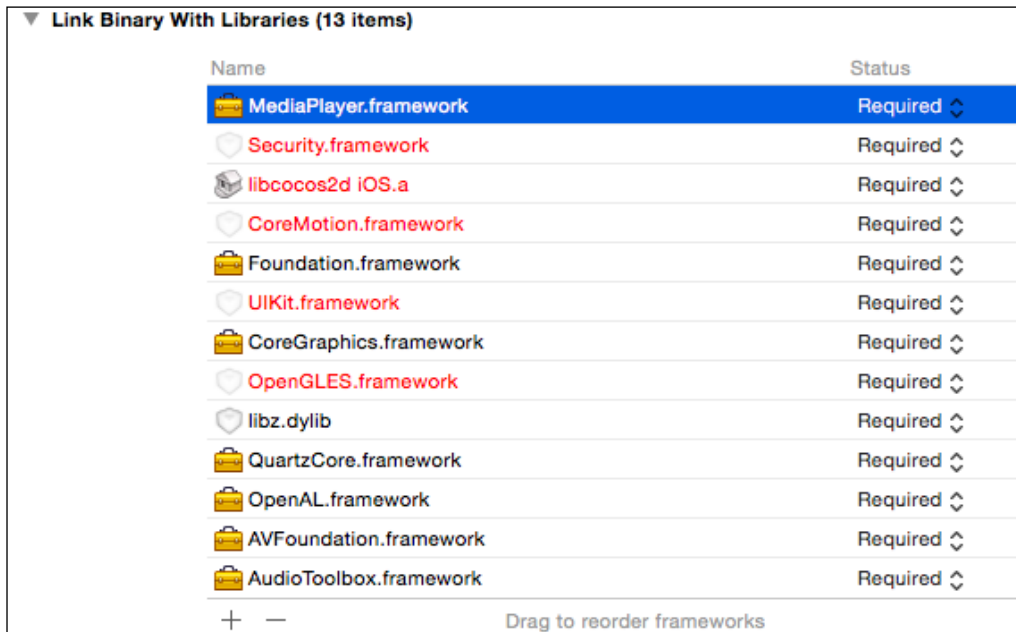
1. You have to add the movie file to the `Resources/res` folder. In this case, we add the video called `splash.mp4`.
2. Next, you have to including a header file. The code to do so is as follows:

```
#include "ui/CocosGUI.h"  
USING_NS_CC;  
using namespace experimental::ui;
```

3. Then, you have to add the following code to the `proj.android/jni/Android.mk` file for building an Android application.

```
LOCAL_WHOLE_STATIC_LIBRARIES += cocos_ui_static  
$(call import-module,ui)
```

- In Xcode, you have to add `MediaPlayer.framework` for iOS, as shown in the following image:



How to do it...

Let's try to play the video in your game. Here, it is:

```

auto visibleSize = Director::getInstance()->getVisibleSize();
auto videoPlayer = VideoPlayer::create();

videoPlayer->setContentSize(visibleSize);
videoPlayer->setPosition(visibleSize/2);
videoPlayer->setKeepAspectRatioEnabled(true);
this->addChild(videoPlayer);

videoPlayer->addEventListener([](Ref *sender,
VideoPlayer::EventType eventType) {
    switch (eventType) {
        case VideoPlayer::EventType::PLAYING:
            CCLOG("PLAYING");
            break;
        case VideoPlayer::EventType::PAUSED:
            CCLOG("PAUSED");
            break;
    }
});
    
```

```
        case VideoPlayer::EventType::STOPPED:
            CCLOG("STOPPED");
            break;
        case VideoPlayer::EventType::COMPLETED:
            CCLOG("COMPLETED");
            break;
        default:
            break;
    }
});

videoPlayer->setFileName("res/splash.mp4");
videoPlayer->play();
```

How it works...

Basically, the `VideoPlayer` class is the same as the other nodes. First, you create an instance, specify its location, and then add it on a layer. Next, you set the content size by using the `setContentSize` method. If you set a false value by using the `setKeepAspectRatioEnabled` method, the video player's size becomes equal to the content size that you specify by using the `setContentSize` method. In contrast, if you set a true value, the video player retains the aspect ratio for the movie.

You can get the event of the playing status by adding an event listener.

`VideoPlayer::EventType` has four types of events, namely `PLAYING`, `PAUSED`, `STOPPED`, and `COMPLETED`.

Finally, you set the movie file by using the `setFileName` method and you can play it by using the `play` method.



There are a lot of video formats. However, the video format that you can play on both iOS and Android is mp4. That's why you should use the mp4 format to play videos in your games.

7

Working with Resource Files

Games have a lot of resources such as images and audio files. Cocos2d-x has a management system of resources. The following topics will be covered in this chapter:

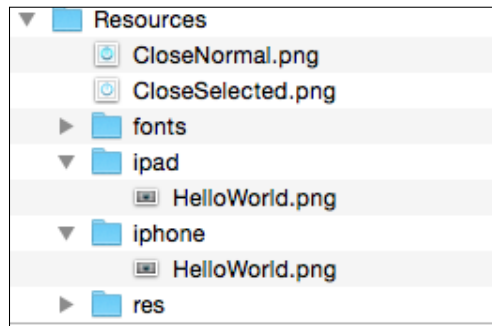
- ▶ Selecting resource files
- ▶ Managing resource files
- ▶ Using SQLite
- ▶ Using .xml files
- ▶ Using .plist files
- ▶ Using .json files

Selecting resource files

Your game has images of each resolution for multiresolution adaption. If you have resolved to find an image for each resolution, your application logic is very complicated. Cocos2d-x has a search path mechanism for solving this problem. In this recipe, we will explain this search path mechanism.

Getting ready

If you want to share some resources between different resolutions, then you can put all the shared resources in the `Resources` folder, and put the resolution-specified resources in different folders as shown in the following image.



`CloseNormal.png` and `CloseSelected.png` are shared resources between different resolutions. However, `HelloWorld.png` is a resolution-specified resource.

How to do it...

You can set the priority to search resources for Cocos2d-x as follows:

```
std::vector<std::string> searchPaths;
searchPaths.push_back("ipad");
FileUtils::setSearchPaths(searchPaths);
Sprite *sprite = Sprite::create("HelloWorld.png");
Sprite *close = Sprite::create("CloseNormal.png");
```

How it works...

Cocos2d-x will find `HelloWorld.png` in `Resources/ipad`. Cocos2d-x will use `HelloWorld.png` in this path; that's why it can find this resource in `Resources/ipad`. However, Cocos2d-x cannot find `CloseNormal.png` in `Resources/ipad`. It will find the `Resources` folder that is the next order path. The system can find it in the `Resources` folder and use it.

You should add this code in the `AppDelegate::applicationDidFinishLaunching` method before creating the first scene. Then, the first scene can use this search path setting.

See also

- ▶ The search path mechanism in the next recipe called *Managing resource files*.

Managing resource files

Cocos2d-x has an extension that manages resources. It is called `AssetsManagerExtension`. This extension is designed for a hot update of resources such as images and audio files. You can update a new version of resources on your games by using this extension without updating your applications.

Getting ready

Before using `AssetsManagerExtension`, you should learn about it. This extension has many useful features to help you make the hot update. Some of these features are as follows:

- ▶ Multithread downloading support
- ▶ Two-level progression support—File-level and byte-level progression
- ▶ Compressed ZIP file support
- ▶ Resuming download
- ▶ Detailed progression information and error information
- ▶ Possibility to retry failed assets

You have to prepare a web server, and hence, your application will download resources.

How to do it...

You need to upload resources and manifest files. In this case, we will update `HelloWorld.png` and a `.zip` file called `test.zip`. This `.zip` file includes some new images. `AssetsManagerExtension` will download resources according to the manifest files. The manifest files are `version.manifest` and `project.manifest`.

The `version.manifest` file contains the following code:

```
{
  "packageUrl" : "http://example.com/assets_manager/",
  "remoteVersionUrl" :
  "http://example.com/assets_manager/version.manifest",
  "remoteManifestUrl" :
  "http://example.com/assets_manager/project.manifest",
  "version" : "1.0.1",
}
```


The `project.manifest` file contains the following code:

```
{
  "packageUrl" : "http://example.com/assets_manager/",
  "remoteVersionUrl" : "http://example.com/assets_manager/version.
manifest",
  "remoteManifestUrl" : "http://example.com/assets_manager/project.
manifest",
  "version" : "1.0.1",
  "assets" : {
    "HelloWorld.png" : {
      "md5" : "b7892dc221c840550847eaffa1c0b0aa"
    },
    "test.zip" : {
      "md5" : "c7615739e7a9bcd1b66e0018aff07517",
      "compressed" : true
    }
  }
}
```

Then, you have to upload these manifest files and new resources.

Next, you have to prepare your application for a hot update. You have to create the `local.manifest` file in your project. The local manifest file should contain the following code:

```
{
  "packageUrl" : "http://example.com/assets_manager/",
  "remoteVersionUrl" :
"http://example.com/assets_manager/version.manifest",
  "remoteManifestUrl" :
"http://example.com/assets_manager/project.manifest",
  "version" : "1.0.0",
}
```

You should make a class that manages `AssetsManagerExtension` in your project. Here, we create a class called `ResourceManager`. Firstly, you will create a header file of `ResourceManager`. It is called `ResourceManager.h`. This file contains the following code:

```
#include "cocos2d.h"
#include "extensions/cocos-ext.h"

class ResourceManager {
private:
  ResourceManager();
  static ResourceManager* instance;
```

```

coccos2d::extension::AssetsManagerEx* _am;
coccos2d::extension::EventListenerAssetsManagerEx* _amListener;

public:
    // custom event name
    static const char* EVENT_PROGRESS;
    static const char* EVENT_FINISHED;

    virtual ~ResourceManager();
    static ResourceManager* getInstance();

    void updateAssets(std::string manifestPath);
};

```

The next step is to create a `ResourceManager.cpp` file. This file contains the following code:

```

#include "ResourceManager.h"

USING_NS_CC;
USING_NS_CC_EXT;

// custom event name
const char* ResourceManager::EVENT_PROGRESS =
    "__cc_Resource_Event_Progress";
const char* ResourceManager::EVENT_FINISHED =
    "__cc_Resource_Event_Finished";

ResourceManager* ResourceManager::instance = nullptr;

ResourceManager::~ResourceManager() {
    CC_SAFE_RELEASE_NULL(_am);
}

ResourceManager::ResourceManager(
    :_am(nullptr)
    ,_amListener(nullptr)
    {
}

ResourceManager* ResourceManager::getInstance() {
    if (instance==nullptr) {
        instance = new ResourceManager();
    }
    return instance;
}

```

```
void ResourceManager::updateAssets(std::string manifestPath)
{
    std::string storagePath = FileUtils::getInstance() -
    >getWritablePath();
    CCLOG("storage path = %s", storagePath.c_str());

    if (_am!=nullptr) {
        CC_SAFE_RELEASE_NULL(_am);
    }
    _am = AssetsManagerEx::create(manifestPath, storagePath);
    _am->retain();

    if (!_am->getLocalManifest()->isLoading()) {
        CCLOG("Fail to update assets, step skipped.");
    } else {
        _amListener = EventListenerAssetsManagerEx::create(_am,
        [this](EventAssetsManagerEx* event){
            static int failCount = 0;
            switch (event->getEventCode())
            {
                case
EventAssetsManagerEx::EventCode::ERROR_NO_LOCAL_MANIFEST:
            {
                CCLOG("No local manifest file found, skip
                assets update.");
                break;
            }
                case
EventAssetsManagerEx::EventCode::UPDATE_PROGRESSION:
            {
                std::string assetId = event->getAssetId();
                float percent = event->getPercent();
                std::string str;
                if (assetId == AssetsManagerEx::VERSION_ID) {
                    // progress for version file
                } else if (assetId ==
AssetsManagerEx::MANIFEST_ID) {
                    // progress for manifest file
                } else {
                    // dispatch progress event
                    CCLOG("%.2f Percent", percent);
                    auto event =
EventCustom(ResourceManager::EVENT_PROGRESS);
                    auto data = Value(percent);
                    event.setUserData(&data);
                    Director::getInstance()->getEventDispatcher()-
                    >dispatchEvent(&event);
                }
            }
        }
    }
}
```

```

        break;
    }
    case
EventAssetsManagerEx::EventCode::ERROR_DOWNLOAD_MANIFEST:
    case
EventAssetsManagerEx::EventCode::ERROR_PARSE_MANIFEST:
    {
        CCLOG("Fail to download manifest file, update
skipped.");
        break;
    }
    case
EventAssetsManagerEx::EventCode::ALREADY_UP_TO_DATE:
    case
EventAssetsManagerEx::EventCode::UPDATE_FINISHED:
    {
        CCLOG("Update finished. %s",
event->getMessage().c_str());
        CC_SAFE_RELEASE_NULL(_am);
        // dispatch finished updating event
        Director::getInstance()->getEventDispatcher()-
>dispatchCustomEvent(ResourceManager::EVENT_FINISHED);
        break;
    }
    case
EventAssetsManagerEx::EventCode::UPDATE_FAILED:
    {
        CCLOG("Update failed. %s", event-
>getMessage().c_str());

        // retry 5 times, if error occurred
        failCount ++;
        if (failCount < 5) {
            _am->downloadFailedAssets();
        } else {
            CCLOG("Reach maximum fail count, exit
update process");
            failCount = 0;
        }
        break;
    }
    case
EventAssetsManagerEx::EventCode::ERROR_UPDATING:
    {
        CCLOG("Asset %s : %s", event-
>getAssetId().c_str(), event->getMessage().c_str());
        break;
    }
    case

```

```

EventAssetsManagerEx::EventCode::ERROR_DECOMPRESS:
    {
        CLOG("%s", event->getMessage().c_str());
        break;
    }
    default:
        break;
}
});

// execute updating resources
Director::getInstance()->getEventDispatcher()-
>addEventListenerWithFixedPriority(_amListener, 1);
_am->update();
}
}

```

Finally, to start updating the resource, use the following code:

```

// label for progress
auto size = Director::getInstance()->getWinSize();
TTFConfig config("fonts/arial.ttf", 30);
_progress = Label::createWithTTF(config, "0%",
    TextHAlignment::CENTER);
_progress->setPosition( Vec2(size.width/2, 50) );
this->addChild(_progress);

// progress event
getEventDispatcher()-
>addCustomEventListener(ResourceManager::EVENT_PROGRESS,
    [this](EventCustom* event){
        auto data = (Value*)event->getUserData();
        float percent = data->asFloat();
        std::string str = StringUtils::format("%.2f", percent) + "%";
        CLOG("%.2f Percent", percent);
        if (this->_progress != nullptr) {
            this->_progress->setString(str);
        }
    });

// finished updating event
getEventDispatcher()-
>addCustomEventListener(ResourceManager::EVENT_FINISHED,
    [this](EventCustom* event){
        // clear cache
        Director::getInstance()->getTextureCache()-
        >removeAllTextures();
        // reload scene
        auto scene = HelloWorld::createScene();
    });

```

```

    Director::getInstance()->replaceScene(scene);
});

// update resources
ResourceManager::getInstance()-
>updateAssets("res/local.manifest");

```

How it works...

Firstly, we will explain the manifest file and the mechanism of `AssetsManagerExtension`. The manifest files are in the JSON format. Local manifest and version manifest have the following data:

Keys	Description
<code>packageUrl</code>	The URL where the assets manager will try to request and download all the assets.
<code>remoteVersionUrl</code>	The remote version manifest file URL that permits one to check the remote version to determine whether a new version has been uploaded to the server.
<code>remoteManifestUrl</code>	The remote manifest file URL that contains all the asset information.
<code>version</code>	The version of this manifest file.

In addition, the remote manifest has the following data in the key called `assets`.

Keys	Description
<code>key</code>	Each key represents the relative path of the asset.
<code>Md5</code>	The <code>md5</code> field represents the version information of the asset.
<code>compressed</code>	When the <code>compressed</code> field is <code>true</code> , the downloaded file will be decompressed automatically; this key is optional.

`AssetsManagerExtension` will execute the hot update in the following steps:

1. Read the local manifest in the application.
2. Download the version manifest according to the remote version URL in the local manifest.
3. Compare the version in the local manifest to the version in the version manifest.
4. If both versions do not match, `AssetsManagerExtension` downloads the project manifest according to the remote manifest URL in the local manifest.

5. Compare the md5 value in the remote manifest to the md5 of the asset in the application.
6. If both md5 values do not match, `AssetsManagerExtension` downloads this asset.
7. Next time, `AssetsManagerExtension` will use the version manifest that was downloaded instead of the local manifest.

Next, we will explain the `ResourceManager` class. You can execute the hot update as follows:

```
ResourceManager.getInstance().updateAssets("res/local.manifest");
```

You should call the `ResourceManager.updateAssets` method by specifying the path of the local manifest. `ResourceManager.updateAssets` will create an instance of `AssetsManagerEx`, which is the class name of `AssetsManagerExtension`, by specifying the path of the local manifest and the path of the storage in the application.

It will create an instance of `EventListenerAssetsManagerEx` for listening to the progress of the hot update.

If the compressed value is true, `AssetsManagerExtension` will unzip it after downloading it.

You can update assets by calling the `AssetsManagerEx.update` method. During the update, you can get the following events:

Event	Description
ERROR_NO_LOCAL_MANIFEST	Cannot find the local manifest.
UPDATE_PROGRESSION	Get the progression of the update.
ERROR_DOWNLOAD_MANIFEST	Fail to download the manifest file.
ERROR_PARSE_MANIFEST	Parse error for the manifest file.
ALREADY_UP_TO_DATE	Already updating assets (The version in the local manifest and the version in the version manifest are equal.).
UPDATE_FINISHED	Finished updating assets.
UPDATE_FAILED	Error occurred during updating assets. In this case, the cause of error may be the connection. You should try to update again.
ERROR_UPDATING	Failed to update.
ERROR_DECOMPRESS	Error occurred during unzipping.

`ResourceManager` dispatches the event called `EVENT_PROGRESS` if it catches the event called `UPDATE_PROGRESSION`. If you catch `EVENT_PROGRESS`, you should update the progress label.

Further, it dispatches the event called `EVENT_FINISHED` if it catches the event called `UPDATE_FINISHED`. If you catch `EVENT_FINISHED`, you should refresh all textures. That's why we remove all texture caches and reload the scene.

```
// clear cache
Director::getInstance()->getTextureCache()->removeAllTextures();
// reload scene
auto scene = HelloWorld::createScene();
Director::getInstance()->replaceScene(scene);
```

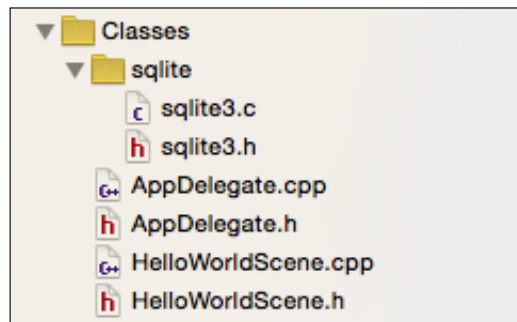
Using SQLite

You can save and load game data easily by using the database in your game. In a smartphone application, the database called SQLite is usually used. SQLite is easy to use. However, you have to set a few things before using it. In this recipe, we will explain how to set up and use SQLite in Cocos2d-x.

Getting ready

Cocos2d-x doesn't have an SQLite library. You have to add SQLite's source code to Cocos2d-x.

You need to download the source code from the site <http://sqlite.org/download.html>. The latest version at the time of writing this book is version 3.8.10. You can download this version's .zip file and expand it. Then, you can add the resulting files to your project as shown in the following image:



In this recipe, we will create an original class called `SQLiteManager`. So, you have to add the `SQLiteManager.h` and `SQLiteManager.cpp` files to your project.

Then, if you build for Android, you have to edit `proj.android/jni/Android.mk` as follows:

```
LOCAL_SRC_FILES := hellocpp/main.cpp \  
                  ../../Classes/AppDelegate.cpp \  
                  ../../Classes/HelloWorldScene.cpp \  
                  ../../Classes/SQLiteManager.cpp \  
                  ../../Classes/sqlite/sqlite3.c
```

How to do it...

First, you have to edit the `SQLiteManager.h` file as follows:

```
#include "cocos2d.h"  
#include "sqlite/sqlite3.h"  
  
class SQLiteManager {  
private:  
    SQLiteManager();  
    static SQLiteManager* instance;  
    sqlite3 *_db;  
    bool open();  
    void close();  
public:  
    virtual ~SQLiteManager();  
    static SQLiteManager* getInstance();  
    void insert(std::string key, std::string value);  
    std::string select(std::string key);  
};
```

Next, you have to edit the `SQLiteManager.cpp` file. This code is a little long. So, we will explain it step by step.

1. Add the following code for the singleton class:

```
SQLiteManager* SQLiteManager::instance = nullptr;  
  
SQLiteManager::~SQLiteManager() {  
}  
  
SQLiteManager::SQLiteManager()  
{  
    if (this->open()) {  
        sqlite3_stmt* stmt;  
        // create table  
        std::string sql = "CREATE TABLE IF NOT EXISTS  
data(key TEXT PRIMARY KEY,value TEXT);";  
        if (sqlite3_prepare_v2(_db, sql.c_str(), -1, &stmt,
```

```

    nullptr) == SQLITE_OK) {
        if (sqlite3_step(stmt) != SQLITE_DONE) {
            CCLOG("Error in CREATE TABLE");
        }
    }

    sqlite3_reset(stmt);
    sqlite3_finalize(stmt);
    this->close();
}
}

```

```

SQLiteManager* SQLiteManager::getInstance() {
    if (instance == nullptr) {
        instance = new SQLiteManager();
    }
    return instance;
}

```

2. Add the method that opens and closes the database:

```

bool SQLiteManager::open()
{
    std::string path = FileUtils::getInstance()-
>getWritablePath()+"test.sqlite";
    return sqlite3_open(path.c_str(), &_db) == SQLITE_OK;
}

void SQLiteManager::close()
{
    sqlite3_close(_db);
}

```

3. Add the method that inserts data to the database:

```

void SQLiteManager::insert(std::string key, std::string
value)
{
    this->open();
    // insert data
    sqlite3_stmt* stmt;
    std::string sql = "INSERT INTO data (key, value)
VALUES(?, ?)";
    if (sqlite3_prepare_v2(_db, sql.c_str(), -1, &stmt,
nullptr) == SQLITE_OK) {
        sqlite3_bind_text(stmt, 1, key.c_str(), -1,
SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 2, value.c_str(), -1,
SQLITE_TRANSIENT);
    }
}

```

```

        if (sqlite3_step(stmt)!=SQLITE_DONE) {
            CLOG("Error in INSERT 1, %s",
                sqlite3_errmsg(_db));
        }
    }
    sqlite3_reset(stmt);
    sqlite3_finalize(stmt);
    this->close();
}

```

4. Add the method that selects data from the database:

```

std::string SQLiteManager::select(std::string key)
{
    this->open();

    // select data
    std::string value;
    sqlite3_stmt* stmt;
    std::string sql = "SELECT VALUE from data where key=?";
    if (sqlite3_prepare_v2(_db, sql.c_str(), -1, &stmt,
        NULL) == SQLITE_OK) {
        sqlite3_bind_text(stmt, 1, key.c_str(), -1,
            SQLITE_TRANSIENT);
        if (sqlite3_step(stmt) == SQLITE_ROW) {
            const unsigned char* val =
                sqlite3_column_text(stmt, 0);
            value = std::string((char*)val);
            CLOG("key=%s, value=%s", key.c_str(), val);
        } else {
            CLOG("Error in SELECT, %s",
                sqlite3_errmsg(_db));
        }
    } else {
        CLOG("Error in SELECT, %s", sqlite3_errmsg(_db));
    }
    sqlite3_reset(stmt);
    sqlite3_finalize(stmt);
    this->close();
    return value;
}

```

5. Finally, here's how to use this class. To insert data, use the following code:

```
SQLiteManager::getInstance()->insert("foo", "value1");
```

To select data, use the following code:

```
std::string value = SQLiteManager::getInstance()-
>select("foo");
```

How it works...

Firstly, in the constructor method of the `SQLiteManager` class, this class creates a table called `data` if it does not already exist. The data table is created in SQL as follows:

```
CREATE TABLE IF NOT EXISTS data(key TEXT PRIMARY KEY,value TEXT);
```

In order to use SQLite, you have to include `sqlite3.h` and use the `sqlite3` API. This API is in the C language. If you would like to learn it, you should check the website <http://sqlite.org/cintro.html>.

We created our database called `test.sqlite` in the sandbox area of the application. If you want to change the location or the name, you should edit the `open` method.

```
std::string path = FileUtils::getInstance()->getWritablePath()+"test.sqlite";
```

You can insert data by using the `insert` method to specify the key and the value.

```
SQLiteManager::getInstance()->insert("foo", "value1");
```

Further, you can select the value by using the `select` method to specify the key.

```
std::string value = SQLiteManager::getInstance()->select("foo");
```

There's more...

In this recipe, we created the `insert` method and the `select` method. However, you can execute other SQL methods such as `delete` and `replace`. Further, you can make the database match your game. So, you will need to edit this class for your game.

Using .xml files

XML is often used as an API's return value. Cocos2d-x has the `TinyXML2` library that can parse an XML file. In this recipe, we will explain how to parse XML files by using this library.

Getting ready

Firstly, you need to create an XML file and save it as `test.xml` in the `Resources/res` folder in your project. In this case, we will use the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <key>key text</key>
  <array>
    <name>foo</name>
```

```
        <name>bar</name>
        <name>hoge</name>
    </array>
</root>
```

To use the TinyXML-2 library, you have to include it and use namespace as follows:

```
#include "tinyxml2/tinyxml2.h"
using namespace tinyxml2;
```

How to do it...

You can parse an XML file by using the TinyXML2 library. In the following code, we parse `test.xml` and log each element in it.

```
std::string path = util->fullPathForFilename("res/test.xml");
XMLDocument *doc = new XMLDocument();
XMLError error = doc->LoadFile(path.c_str());
if (error == 0) {
    XMLElement *root = doc->RootElement();
    XMLElement *key = root->FirstChildElement("key");
    if (key) {
        CCLOG("key element = %s", key->GetText());
    }
    XMLElement *array = key->NextSiblingElement();
    XMLElement *child = array->FirstChildElement();
    while ( child ) {
        CCLOG("child element= %s", child->GetText());
        child = child->NextSiblingElement();
    }
    delete doc;
}
```

This result is the following log:

```
key element = key text
child element= foo
child element= bar
child element= hoge
```

How it works...

First, you will have to create an instance of `XMLDocument` and then, parse the `.xml` file by using the `XMLDocument::LoadFile` method. To get the root element, you will have to use the `XMLDocument::RootElement` method. Basically, you can get the element by using the `FirstChildElement` method. If it is a continuous element, you can get the next element by using the `NextSiblingElement` method. If there are no more elements, the return value of `NextSiblingElement` will be null.

Finally, you shouldn't forget to delete the instance of `XMLDocument`. That's why you created it using a new operation.

Using .plist files

PLIST used in OS X and iOS is a property list. The file extension is `.plist`, but in fact, the PLIST format is an XML format. We often use `.plist` files to store game settings and so on. Cocos2d-x has a class through which you can easily use `.plist` files.

Getting ready

Firstly, you need to create a `.plist` file and save it as `test.plist` to the `Resources/res` folder in your project. In this case, it has two keys, namely `foo` and `bar`. The `foo` key has an integer value of 1. The `bar` key has a string value of `This is string`. Refer to the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>foo</key>
  <integer>1</integer>
  <key>bar</key>
  <string>This is string</string>
</dict>
</plist>
```

How to do it...

You can parse a `.plist` file by using the `FileUtils::getValueMapFromFile` method. In the following code, we parse `test.plist` and log a key value in it.

```
FileUtils* util = FileUtils::getInstance();
std::string path = util->fullPathForFilename("res/test.plist");
ValueMap map = util->getValueMapFromFile(path);
for (auto element : map) {
    std::string key = element.first;
    Value value = element.second;
    switch (value.getType()) {
        case Value::Type::BOOLEAN:
            CCLOG("%s, %s", key.c_str(),
                value.asBool()?"true":"false");
            break;
        case Value::Type::INTEGER:
            CCLOG("%s, %d", key.c_str(), value.asInt());
            break;
        case Value::Type::FLOAT:
            CCLOG("%s, %f", key.c_str(), value.asFloat());
            break;
        case Value::Type::DOUBLE:
            CCLOG("%s, %f", key.c_str(), value.asDouble());
            break;
        case Value::Type::STRING:
            CCLOG("%s, %s", key.c_str(),
                value.asString().c_str());
            break;
        default:
            break;
    }
}
```

How it works...

You can parse a `.plist` file by specifying the `.plist` file's path to the `FileUtils::getValueMapFromFile` method. After doing so, you get the data from the `.plist` file as a `ValueMap` value. The `ValueMap` class is a wrapper class-based `std::unordered_map`. PLIST's data containers are `Array` and `Dictionary`. After parsing the `.plist` file, `Array` is `std::vector<Value>` and `Dictionary` is `std::unordered_map<std::string, Value>`. Further, you can distinguish the type of value by using the `Value::getType` method. Then, you can get the value by using the `Value::asInt`, `asFloat`, `asDouble`, `asBool`, and `asString` methods.

There's more...

You can save the `.plist` file from `ValueMap`. By doing so, you can save your game data in the `.plist` file. To save the `.plist` file, use the following code:

```
ValueMap map;
for (int i=0; i<10; i++) {
    std::string key = StringUtils::format("key_%d", i);
    Value val = Value(i);
    map.insert(std::make_pair(key, val));
}
std::string fullpath = util->getWritablePath() + "/test.xml";
FileUtils::getInstance()->writeToFile(map, fullpath);
```

First, you need to set the key value in `ValueMap`. In this case, the values are all of the integer type, but you can set mixed-type values as well. Finally, you need to save the file as a `.plist` file by using the `FileUtils::writeToFile` method.

Using .json files

We can use the JSON format like the XML format for saving/loading game-related data. JSON is a simpler format than XML. It takes less space to represent the same data than the XML file format. Further, today, it is used as the value of Web API. Cocos2d-x has a JSON parse library called **RapidJSON**. In this recipe, we will explain how to use RapidJSON.

Getting ready

RapidJSON is usually included in Cocos2d-x. However, you need to include the header files as follows:

```
#include "json/rapidjson.h"
#include "json/document.h"
```

How to do it...

Firstly, we will parse a JSON string as follows:

```
std::string str = "{\"hello\" : \"word\"}";
```

You can parse JSON by using `rapidjson::Document` as follows:

```
rapidjson::Document d;
d.Parse<0>(str.c_str());
```



```
if (d.HasParseError()) {
    CCLOG("GetParseError %s\n", d.GetParseError());
} else if (d.IsObject() && d.HasMember("hello")) {
    CCLOG("%s\n", d["hello"].GetString());
}
```

How it works...

You can parse JSON by using the `Document::Parse` method and specifying the JSON string. You may get a parse error when you use the `Document::HasParseError` method; you can get a description of this error by using the `Document::GetParseError` method for a string. Further, you can get an element by specifying the hash key and using the `Document::GetString` method.

There's more...

In a real application, you can get a JSON string from a file. We will now explain how to get this string from a file. First, you need to add a file called `test.json` to the `Resources/res` folder in your project and save it as follows:

```
[{"name": "Tanaka", "age": 25}, {"name": "Ichiro", "age": 40}]
```

Next, you can get a JSON string from a file as follows:

```
std::string jsonData = FileUtils::getInstance()-
>getStringFromFile("res/test.json");
CCLOG("%s\n", jsonData.c_str());
rapidjson::Document d;
d.Parse<0>(jsonData.c_str());
if (d.HasParseError()) {
    CCLOG("GetParseError %s\n", d.GetParseError());
} else {
    if (d.IsArray()) {
        for (rapidjson::SizeType i = 0; i < d.Size(); ++i) {
            auto name = d[i]["name"].GetString();
            auto age = d[i]["age"].GetInt();
            CCLOG("name-%s, age=%d", name, age);
        }
    }
}
```

You can get the string from the file by using the `FileUtils::getStringFromFile` method. Thereafter, you can parse in the same way. In addition, this JSON string may be an array. You can check whether the format is an array by using the `Document::IsArray` method. Then, you can use a for loop to go through the JSON object in the array.

8

Working with Hardware

The following topics will be covered in this chapter:

- ▶ Using native code
- ▶ Change the processing using the platform
- ▶ Using the acceleration sensor
- ▶ Keeping the screen on
- ▶ Getting dpi
- ▶ Getting the maximum texture size

Introduction

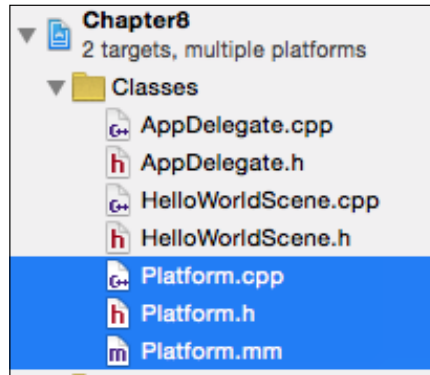
Cocos2d-x has a lot of APIs. However, there are no APIs that we need, for example, In-App purchase, push notification, and so on. In this case, we have to create original APIs and need to write Objective-C code for iOS or Java code for Android. In addition, we want to get the device information that it is running on. When we want to adjust for each device, we have to get the device information such as the running application version, device name, dpi on device, and so on. However, doing so is very difficult and confusing. In this chapter, you can write the native code for iOS or Android and get the device information.

Using native code

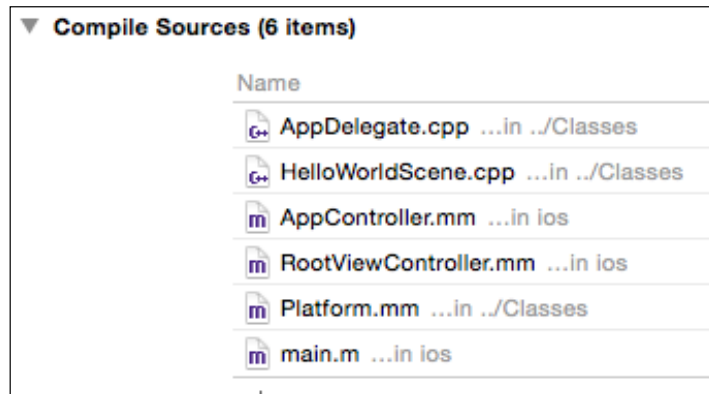
In Cocos2d-x, you can write one source for the cross platform. However, you have to write an Objective-C function or a Java function for the dependency process such as a purchase or push notification. If you want to call Java for Android from C++, you have to use **JNI (Java Native Interface)**. In particular, JNI is very confusing. To call Java from C++, you have to use JNI. In this recipe, we will explain how to call an Objective-C function or a Java function from C++.

Getting ready

In this case, we will make a new class called `Platform`. You can get the application version by using this class. Before writing code, you will make three files called **Platform.h**, **Platform.mm**, and **Platform.cpp** in your project.



It is important that you don't add `Platform.cpp` to **Compile Sources** in Xcode. That's why `Platform.cpp` is for an Android target and doesn't need to be built for iOS. If you added it to **Compile Sources**, you have to remove it from there.



How to do it...

1. Firstly, you have to make a header file called `Platform.h` by using the following code:

```
class Platform
{
public:
    static const char* getAppVersion();
};
```

2. You have to make an execution file called `Platform.mm` for iOS. This code is in Objective-C.

```
#include "Platform.h"

const char* Platform::getAppVersion()
{
    NSDictionary* info = [[NSBundle mainBundle]
infoDictionary];
    NSString* version = [info
objectForKey:(NSString*)kCFBundleVersionKey];
    if (version) {
        return [version UTF8String];
    }
    return nullptr;
}
```

3. You have to make an execution file called `Platform.cpp` for Android. The following code is in C++ and uses Java through JNI:

```
#include "Platform.h"
#include "platform/android/jni/JniHelper.h"
#define CLASS_NAME "org/cocos2dx/cpp/AppActivity"

USING_NS_CC;

const char* Platform::getAppVersion()
{
    JNIEnvInfo t;
    const char* ret = NULL;
    if (JniHelper::getStaticMethodInfo(t, CLASS_NAME,
"getAppVersionInJava", "()Ljava/lang/String;")) {
        jstring jstr = (jstring)t.env-
>CallStaticObjectMethod(t.classID,t.methodID);
        std::string sstr = JniHelper::jstring2string(jstr);
        t.env->DeleteLocalRef(t.classID);
    }
}
```

```
        t.env->DeleteLocalRef(jstr);
        ret = sstr.c_str();
    }
    return ret;
}
```

4. You have to edit `proj.android/jni/Android.mk` to build for Android when you added a new class file in your project.

```
LOCAL_SRC_FILES := hellocpp/main.cpp \
    ../../Classes/AppDelegate.cpp \
    ../../Classes/HelloWorldScene.cpp \
    ../../Classes/Platform.cpp
```

5. Next, you have to write Java code in `AppActivity.java`. This file is named `proj.android/src/org/cocos2dx/cpp/AppActivity.java`.

```
public class AppActivity extends Cocos2dxActivity {
    public static String appVersion = "";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        try {
            PackageInfo packageInfo =
                getPackageManager().getPackageInfo(getPackageName(),
                    PackageManager.GET_META_DATA);
            appVersion = packageInfo.versionName;
        } catch (NameNotFoundException e) {
        }
    }

    public static String getAppVersionInJava() {
        return appVersion;
    }
}
```

6. Finally, you can get a version of your game by using the following code:

```
#include "Platform.h"

const char* version = Platform::getAppVersion();
CCLOG("application version = %s", version);
```

How it works...

1. Firstly, we will look at it for iOS. You will be able to get a version of your game by using Objective-C in `Platform.mm`. You can write C++ and Objective-C in the `.mm` files.
2. Next, we will look for Android. When you call `Platform::getAppversion` on Android devices, the method in `Platform.cpp` is executed. In this method, you can call the `getAppVersionInJava` method in `AppActivity.java`. by using JNI. C++ can connect Java via JNI. That's why you can only get the application version by using Java.
3. In Java, you can get the version of your application by using the `onCreate` method. You can set it to a static variable and then, get it from the `getAppVersionInJava` method in Java.

There's more...

You can use JNI easily by using the `JniHelper` class in Cocos2d-x. How this class manages typos from C++ and creates a bridge between C++ and Java has already been explained. You can use the `JniHelper` class by using the following code:

```
JniMethodInfo t;
JniHelper::getStaticMethodInfo(t, CLASS_NAME,
"getAppVersionInJava",
"()Ljava/lang/String;")
```

You can get the information about the Java method by using `JniHelper::getStaticMethodInfo`. The first argument is a variable of `JniMethodInfo`. The second argument is the name of the class that has the method you want to call. The third argument is the method name. The last argument is the parameter of this method. This parameter is decided by the return value and the arguments. The characters in the bracket are the parameters for the Java method. In this case, this method has no parameters. The characters after the bracket are the return value. `Ljava/lang/String` means that the return value is a string. If you get this parameter easily, you should use the command called `javap`. As the following result will be generated by using this command.

```
$ cd /path/to/project/pro.android/bin/classes
$ javap -s org.cocos2dx.cpp.AppActivity
Compiled from "AppActivity.java"
public class org.cocos2dx.cpp.AppActivity extends
org.cocos2dx.lib.Cocos2dxActivity {
    public static java.lang.String appVersion;
        descriptor: Ljava/lang/String;
    public org.cocos2dx.cpp.AppActivity();
        descriptor: ()V
```

```
protected void onCreate(android.os.Bundle);
    descriptor: (Landroid/os/Bundle;)V

public static java.lang.String getAppVersionInJava();
    descriptor: ()Ljava/lang/String;

static {};
    descriptor: ()V
}
```

From the above result, you can see that the parameter for the `getAppVersionInJava` method is `()Ljava/lang/String;`

As mentioned earlier, you can get the information of the Java method as a `t` variable. So, you can call the Java method by using this variable and the following code:

```
jstring jstr = (jstring)t.env-
>CallStaticObjectMethod(t.classID,t.methodID);
```

Changing the processing using the platform

You can make the program run on specific parts of the source code for each OS. For example, you will change the file name, the file path, or the image scale by the platform. In this recipe, we will introduce the branching code based on the platform of choice in the case of a complication.

How to do it...

You can change the processing by using the preprocessor as follows:

```
#if (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)
    CCLOG("this platform is Android");
#elif (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
    CCLOG("this platform is iOS");
#else
    CCLOG("this platform is others");
#endif
```

How it works...

Cocos2d-x defined the `CC_TARGET_PLATFORM` value in `CCPlatformConfig.h`. If your game is compiled for Android devices, `CC_TARGET_PLATFORM` is equal to `CC_PLATFORM_ANDROID`. If it is compiled for iOS devices, `CC_TARGET_PLATFORM` is equal to `CC_PLATFORM_IOS`. Needless to say, there are other values besides Android and iOS. Please check `CCPlatformConfig.h`.

There's more...

The code that was used in the preprocessor is difficult to read on an editor. Further, you cannot notice the error before compiling your code. You should define a constant value that can be changed by the preprocessor, but you should change the processing by using code as much as possible. You can check the platform by using the `Application` class in Cocos2d-x as follows:

```
switch (Application::getInstance()->getTargetPlatform()) {
    case Application::Platform::OS_ANDROID:
        CCLOG("this device is Android");
        break;
    case Application::Platform::OS_IPHONE:
        CCLOG("this device is iPhone");
        break;
    case Application::Platform::OS_IPAD:
        CCLOG("this device is iPad");
        break;
    default:
        break;
}
```

You can get the value of the platform by using the `Application::getTargetPlatform` method. You will be able to check, not just for iPhone or iPad, but also IOS by using this method.

Using the acceleration sensor

By using an acceleration sensor on the device, we can make the game more engrossing, by using operations such as shaking and tilting the device. For example, move the ball by tilting the screen, the maze game that aims at the goal, and the skinny panda trying to go on a diet, wherein the players shake the device to play the game. You can get the tilt value and the moving speed of the device by using the accelerometer. If you can use it, your game becomes more unique. In this recipe, we learn how to use the acceleration sensor.

How to do it...

You can get the x, y, and z axis values from the acceleration sensor by using the following code:

```
Device::setAccelerometerEnabled(true);
auto listener = EventListenerAcceleration::create([] (Acceleration*
acc, Event* event) {
    CCLOG("x=%f, y=%f, z=%f", acc->x, acc->y, acc->z);
});
this->getEventDispatcher()-
>addEventListenerWithSceneGraphPriority(listener, this);
```


How it works...

1. Firstly, you enable the acceleration sensor by using the `Device::setAccelerometerEnable` method. The methods in the `Device` class are static methods. So, you can directly call a method without an instance like this:

```
Device::setAccelerometerEnable(true);
```
2. You set the event listener for getting the value from the acceleration sensor. In this case, you can get these values by using the lambda function.
3. Finally, you set the event listener in the event dispatcher.
4. You can get the value of the x, y, and z axes from the acceleration sensor, if you run this code on the real device. The x axis is the left and the right of the slope. The y axis is before and after of the slope. The z axis is the vertical motion.

There's more...

The acceleration sensor uses more battery power. When you use it, you set an appropriate interval for when the event occurred. The following code sets the interval as one second.

```
Device::setAccelerometerInterval(1.0f);
```



[If the interval is higher, we might miss some tilt inputs. However, if we use a low interval, we will drain a lot of battery.]

Keeping the screen on

You have to keep the device from entering into the sleep mode while playing your game. For example, in your game, the player can control the game and keep the game going by using the accelerometer. The problem is that if the player does not touch the screen while playing with the accelerometer, the device goes to sleep and enters background mode. In this recipe, you can keep the screen on easily.

How to do it...

You can keep the screen on if you set it to `true` by using the `Device::setKeepScreenOn` method as follows:

```
Device::setKeepScreenOn(true);
```

How it works...

There is a different way for each platform to prevent a device from entering the sleep mode. However, Cocos2d-x can do it for every platform. You can use this method without executing a platform. In the iOS platform, the `setKeepScreenOn` method is as follows:

```
void Device::setKeepScreenOn(bool value)
{
    [[UIApplication sharedApplication]
 setIdleTimerDisabled:(BOOL)value];
}
```

In the Android platform, the method is as follows:

```
public void setKeepScreenOn(boolean value) {
    final boolean newValue = value;
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mGLSurfaceView.setKeepScreenOn(newValue);
        }
    });
}
```

Getting dpi

There are a lot of **dpi (dots per inch)** variations for each device. You can prepare several kinds of images by resolution. You might want to change an image by the dpi running on the device. In this recipe, if you would like to get the dpi that your game is running on, you need to use the Cocos2d-x function.

How to do it...

You can get dpi of the device game is executing on, by using the `Device::getDPI` method as follows:

```
int dpi = Device::getDPI();
CCLOG("dpi = %d", dpi);
```

How it works...

In fact, we checked the dpi of some devices. To use the dpi information, you can further adjust the multiscreen resolution.

Device	Dpi
iPhone 6 Plus	489
iPhone 6	326
iPhone 5s	326
iPhone 4s	326
iPad Air	264
iPad 2	132
Nexus 5	480

Getting the maximum texture size

The maximum texture size that can be used is different for each device. In particular, when you use the texture atlas, you should be careful. That's why a texture atlas that has a lot of images is too large in size. You can't use a texture that is over the maximum size. If you use it, your game will crash. In this recipe, you can get the maximum texture size.

How to do it...

You can get the max texture size easily by using the following code:

```
auto config = Configuration::getInstance();
int textureSize = config->getMaxTextureSize();
CCLOG("max texture size = %d", textureSize);
```

How it works...

The Configuration class is a singleton class. This class has some OpenGL variables. OpenGL is a multiplatform API for rendering 2D and 3D vector graphics. It is pretty difficult to use. Cocos2d-x wraps it and makes it easy to use. OpenGL has a lot of information for graphics. The max texture size is one of the variables providing this information. You can get the max texture size of the device that your application is running on.

There's more...

You can get other OpenGL variables. If you want to check the variables that `Configuration` has, you will use the `Configuration::getInfo` method.

```
auto config = Configuration::getInstance();
std::string info = config->getInfo();
CCLOG("%s", info.c_str());
```

The result of the log on iPhone 6 Plus:

```
{
  gl.supports_vertex_array_object: true  cocos2d.x.version:
  cocos2d-x 3.5
  gl.vendor: Apple Inc.
  gl.supports_PVRTC: true
  gl.renderer: Apple A8 GPU
  cocos2d.x.compiled_with_profiler: false
  gl.max_texture_size: 4096
  gl.supports_ETC1: false
  gl.supports_BGRA8888: false
  cocos2d.x.build_type: RELEASE
  gl.supports_discard_framebuffer: true
  gl.supports_NPOT: true
  gl.supports_ATITC: false
  gl.max_samples_allowed: 4
  gl.max_texture_units: 8
  cocos2d.x.compiled_with_gl_state_cache: true
  gl.supports_S3TC: false
  gl.version: OpenGL ES 2.0 Apple A8 GPU - 53.13
}
```

If you get each variable, and you check the source code of the `Configuration` class, you can understand them easily.

9

Controlling Physics

The following topics will be covered in this chapter:

- ▶ Using the physics engine
- ▶ Detecting collisions
- ▶ Using joints
- ▶ Changing gravity by using the acceleration sensor

Introduction

Physics is really important for games. Players need to simulate real-world situations. You can add physical realism to your game by using a physics engine. As you know, there are two famous physics engines: Box2D and Chipmunk. In Cocos2d-x version 2.x, you have to use these physics engines. However, it is pretty difficult to use them. Since Cocos2d-x version 3.x, Cocos2d-x has a useful physics engine wrapped in Chipmunk. Therefore, the physics engine is no longer a concern for us as it is scalable and CPU friendly. In this chapter, you will learn how to use the physics engine easily in your game.

Using the physics engine

What should you do when you realize that your game needs to simulate real-world situations? You know that the answer is to use a physics engine. When you start using a physics engine, you have to use some new classes and methods. In this recipe, you will learn how to use the basic physics engine in Cocos2d-x.

How to do it...

1. Firstly, you have to create the physics world in your scene. You can create it by using the following code:

```
Scene* HelloWorld::createScene()
{
    auto scene = Scene::createWithPhysics();
    auto layer = HelloWorld::create();
    scene->addChild(layer);
    return scene;
}
```

2. Next, you have to add the physics bodies in the physics world. A physics body is not visible. It is a physical shape such as a square or a circle or a more complex shape. Here, let's create a square shape. You have to create it and set it to the sprite to be visible.

```
bool HelloWorld::init()
{
    if ( !Layer::init() )
    {
        return false;
    }
    Size visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();

    auto wall = Node::create();
    auto wallBody = PhysicsBody::createEdgeBox(visibleSize,
PhysicsMaterial(0.1f, 1.0f, 0.0f));
    wall->setPhysicsBody(wallBody);
    wall->setPosition(Vec2(visibleSize.width/2+origin.x,
visibleSize.height/2+origin.y));
    addChild(wall);

    auto sprite = Sprite::create("CloseNormal.png");
    sprite->setPosition(visibleSize/2);
    auto physicsBody = PhysicsBody::createCircle(sprite-
>getContentSize().width/2);
    physicsBody->setDynamic(true);
    sprite->setPhysicsBody(physicsBody);
    this->addChild(sprite);

    return true;
}
```

3. Finally, you have to run the preceding code. You can then see the sprite falling and bouncing on the ground.

How it works...

1. Firstly, you have to create the physics world in the scene by using the `Scene::createWithPhysics` method. In this way, you can use the physics engine in your game.
2. Next, you have to create the wall upside down and from the left to the right on the screen edge. If you remove this wall and run the code, the sprite object will be falling forever. You can create an edge box by using the `PhysicsBody::createEdgeBox` method with this size and material setting. In this case, the wall will be of the same size as the screen. The material setting is specified as `PhysicsMaterial(0.1f, 1.0f, 0.0f)`. This means that the density is `1.0f`, restitution is `1.0f`, and friction is `0.0f`. Let's try to change this parameter and check it in the given situation.
3. Finally, you can create the physics body with the sprite. In this case, the sprite is circular in shape. So, you need to use the `PhysicsBody::createCircle` method to create the circular physics body. Then, add the physics body to the sprite by using the `Sprite::setPhysicsBody` method.
4. Cocos2d-x has a lot of physics body shapes as listed in the following table:

Shape	Description
<code>PhysicsShapeCircle</code>	Solid circle shape
<code>PhysicsShapePolygon</code>	Solid polygon shape
<code>PhysicsShapeBox</code>	Solid box shape
<code>PhysicsShapeEdgeSegment</code>	Segment shape
<code>PhysicsShapeEdgePolygon</code>	Hollow polygon shape
<code>PhysicsShapeEdgeBox</code>	Hollow box shape
<code>PhysicsShapeEdgeChain</code>	To connect many edges

There's more...

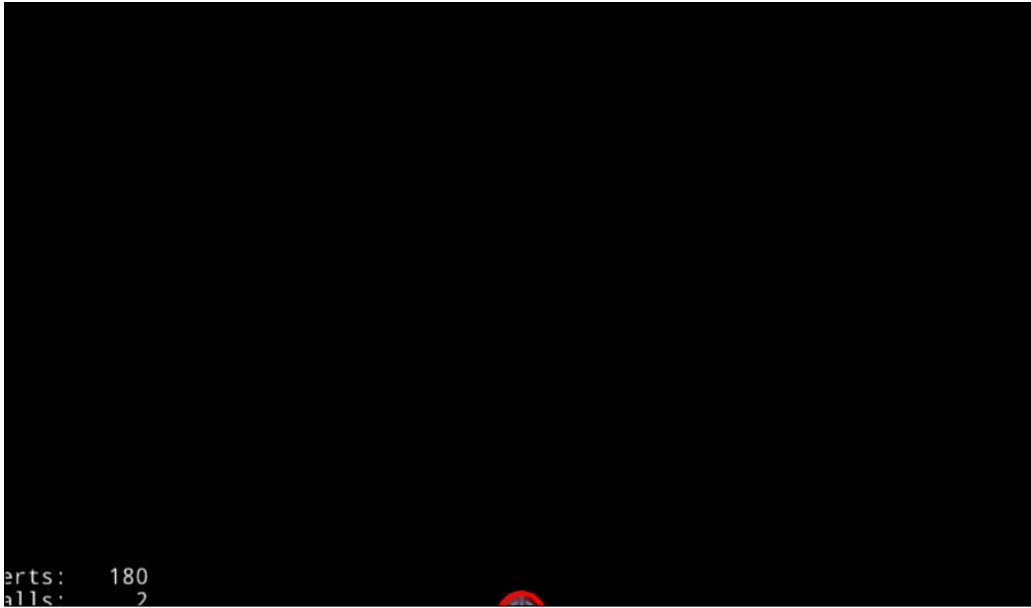
Then, you can get a `PhysicsWorld` instance by using the `Scene::getPhysicsWorld` method. In this recipe, we set `PhysicsWorld::DEBUGDRAW_ALL` to the physics world. That's why you can see that the edges of all physics objects are red lines. When you release your game, you will have to remove this setting.

```
Scene* HelloWorld::createScene()
{
    auto scene = Scene::createWithPhysics();
    auto layer = HelloWorld::create();
    scene->addChild(layer);
}
```



```
PhysicsWorld* world = scene->getPhysicsWorld();
world->setDebugDrawMask(PhysicsWorld::DEBUGDRAW_ALL);

return scene;
}
```



Further, you can set the original gravity value to `PhysicsWorld`. For example, you can change the gravity when the device was tilted. The following code is how to change the gravity:

```
PhysicsWorld* world = scene->getPhysicsWorld();
auto gravity = Vec2(0, 98.0f);
world->setGravity(gravity);
```

The above code is against the force of the earth's gravity. The default gravity value is `Vec2(0, -98.0f)`.

Detecting collisions

When a collision between physics objects occurs, you want to take action against the physics bodies, for example, showing an explosion and showing a particle. In this recipe, you learn how to detect a collision in the physics world.

How to do it...

1. Firstly, you have to create the event listener in the `init` method as follows:

```
auto contactListener =
EventListenerPhysicsContact::create();
contactListener->onContactBegin = [](PhysicsContact& contact){
    CLOG("contact begin");
    auto shapeA = contact.getShapeA();
    auto bodyA = shapeA->getBody();

    auto shapeB = contact.getShapeB();
    auto bodyB = shapeB->getBody();
    return true;
};
this->getEventDispatcher()->addEventListenerWithSceneGraphPriority
(contactListener, this);
```

2. Next, you have to set the contact test bit mask to the physics bodies that you want to check the collisions for. In this recipe, you set the wall body and the sprite body as follows:

```
auto wallBody = PhysicsBody::createEdgeBox(visibleSize,
PhysicsMaterial(0.1f, 1.0f, 0.0f));
wallBody->setContactTestBitmask(1);

auto physicsBody = PhysicsBody::createCircle(sprite-
->getContentSize().width/2);
physicsBody->setContactTestBitmask(1);
```

How it works...

You can detect a collision in the physics world by using the `EventListenerPhysicsContact` class. It will receive all the contact callbacks in the physics world. If you set the `onContactBegin` method in this listener, you can catch the collision of the physics bodies. You can get two physics shapes from the parameter's `PhysicsContact` instance in the `onContactBegin` method by using the `getShapeA`, `getShapeB`, and `getBody` method as follows:

```
contactListener->onContactBegin = [](PhysicsContact& contact){
    CLOG("contact begin");
    auto shapeA = contact.getShapeA();
    auto bodyA = shapeA->getBody();
```

```
    auto shapeB = contact.getShapeB();
    auto bodyB = shapeA->getBody();
    return true;
};
```

The `onContactBegin` method returns true or false. If it returns true, the two physics bodies will collide. If it returns false, there will not be a collision response. So, you decide to check the type of collision of the two bodies any way.

The `setContactTestBitmask` method has a parameter to contact the test bit mask. This mask defines which categories of bodies cause intersection notifications with this physics body. When two bodies share the same space, each body's category mask is tested against the other body's contact mask by performing a logical AND operation. If either comparison results in a non-zero value, the `PhysicsContact` object is created and passed to the physics world's delegate. For best performance, only set bits in the contacts mask for the interactions you need. The bitmask is an integer number. The default value is `0x00000000` (all bits cleared).

`PhysicsContact` has some other events as listed in the following table:

Event	Description
<code>onContactBegin</code>	Called when two shapes start to contact
<code>onContactPreSolve</code>	Two shapes are touching
<code>onContactPostSolve</code>	Two shapes' collision responses have been processed
<code>onContactSeparate</code>	Called when two shapes separate

Using joints

Joints are used to connect two physics bodies to each other. Then, you can create a complex shape to join some shapes. In addition, you can create objects such as a gear or a motor to use joints. Cocos2d-x has many different types of joints. In this recipe, we explain a typical joint type.

Getting ready

You will create a method that creates a physics object. That's why you have to create multiple physics objects. This method is called `makeSprite`. You have to add the following code in `HelloWorld.h`:

```
cocos2d::Sprite* makeSprite();
```

You have to add the following code in `HelloWorld.cpp`:

```
Sprite* HelloWorld::makeSprite()
{
    auto sprite = Sprite::create("CloseNormal.png");
    auto physicsBody = PhysicsBody::createCircle(sprite-
>getContentSize().width/2);
    physicsBody->setDynamic(true);
    physicsBody->setContactTestBitmask(true);
    sprite->setPhysicsBody(physicsBody);
    return sprite;
}
```

How to do it...

In this recipe, we explain `PhysicsJointGear`. This joint works to keep the angular velocity ratio of a pair of bodies.

1. Firstly, you have to add the following code in `HelloWorld.h`:

```
void onEnter();
cocos2d::DrawNode* _drawNode;
cocos2d::PhysicsWorld* _world;
```

2. Secondly, you have to add the `onEnter` method to create a gear joint by using two physics objects and the `PhysicsJointGear` class in `HelloWorld.cpp`:

```
void HelloWorld::onEnter()
{
    Layer::onEnter();

    Size visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();

    _world = Director::getInstance()->getRunningScene()-
>getPhysicsWorld();

    // wall
    auto wall = Node::create();
    auto wallBody = PhysicsBody::createEdgeBox(visibleSize,
PhysicsMaterial(0.1f, 1.0f, 0.0f));
    wallBody->setContactTestBitmask(true);
    wall->setPhysicsBody(wallBody);
    wall->setPosition(Vec2(visibleSize.width/2+origin.x,
visibleSize.height/2+origin.y));
    addChild(wall);

    // gear object 1
    auto sp1 = this->makeSprite();
```

```

    sp1->setPosition(visibleSize/2);
    this->addChild(sp1);
    // gear object 2
    auto sp2 = this->makeSprite();
    sp2->setPosition(Vec2(visibleSize.width/2+2, visibleSize.
height));
    this->addChild(sp2);

    // joint: gear
    auto body1 = sp1->getPhysicsBody();
    auto body2 = sp2->getPhysicsBody();
    auto pin1 = PhysicsJointPin::construct(body1, wallBody, sp1-
>getPosition());
    _world->addJoint(pin1);
    auto pin2 = PhysicsJointPin::construct(body2, wallBody, sp2-
>getPosition());
    _world->addJoint(pin2);
    auto joint = PhysicsJointGear::construct(body1, body2, 0.0f,
2.0f);
    _world->addJoint(joint);
}

```

- Next, you have to be able to touch physics objects. Add in `HelloWorld.h`, the following code:

```

bool onTouchBegan(cocos2d::Touch* touch, cocos2d::Event* event);
void onTouchMoved(cocos2d::Touch* touch, cocos2d::Event* event);
void onTouchEnded(cocos2d::Touch* touch, cocos2d::Event* event);
cocos2d::Node* _touchNode;

```

Then, add to the `HelloWorld::onEnter` method in `HelloWorld.cpp`, the following code:

```

auto touchListener = EventListenerTouchOneByOne::create();
touchListener->onTouchBegan = CC_CALLBACK_2(HelloWorld::onTouchBega
n, this);
touchListener->onTouchMoved = CC_CALLBACK_2(HelloWorld::onTouchMov
ed, this);
touchListener->onTouchEnded = CC_CALLBACK_2(HelloWorld::onTouchEnd
ed, this);
_eventDispatcher->addEventListenerWithSceneGraphPriority(touchList
ener, this);

```

- Next, you write the executing codes in three touch methods as follows:

```

bool HelloWorld::onTouchBegan(Touch* touch, Event* event)
{
    auto location = touch->getLocation();
    auto shapes = _world->getShapes(location);
    if (shapes.size()<=0) {

```

```

        return false;
    }
    PhysicsShape* shape = shapes.front();
    PhysicsBody* body = shape->getBody();
    if (body != nullptr) {
        _touchNode = Node::create();
        auto touchBody = PhysicsBody::create(PHYSICS_INFINITY,
        PHYSICS_INFINITY);
        _touchNode->setPhysicsBody(touchBody);
        _touchNode->getPhysicsBody()->setDynamic(false);
        _touchNode->setPosition(location);
        this->addChild(_touchNode);
        PhysicsJointPin* joint = PhysicsJointPin::construct(touchB
        ody, body, location);
        joint->setMaxForce(5000.0f * body->getMass());
        _world->addJoint(joint);
        return true;
    }
    return false;
}

void HelloWorld::onTouchMoved(Touch* touch, Event* event)
{
    if (_touchNode!=nullptr) {
        _touchNode->setPosition(touch->getLocation());
    }
}

void HelloWorld::onTouchEnded(Touch* touch, Event* event)
{
    if (_touchNode!=nullptr) {
        _touchNode->removeFromParent();
        _touchNode = nullptr;
    }
}

```

5. Finally, you will run and test the gear joint by touching the physics objects.

How it works...

1. Firstly, you have to fix gear objects on the wall, as gear objects will drop to the floor if they are not fixed. To fix them, you use the `PhysicsJointPin` class.

```

auto pin1 = PhysicsJointPin::construct(body1, wallBody,
sp1->getPosition());

```

- Next, you create a gear joint by using the `PhysicsJointGear` class. In the `PhysicsJointGear::construct` method, you specify two physics bodies, namely phase value and ratio value. The phase value is the initial angular offset of the two bodies. The ratio value is the gear ratio. If the ratio value is `2.0f`, one axis will be rotated twice and the other axis will be rotated once.

```
auto joint = PhysicsJointGear::construct(body1, body2, 0.0f,
2.0f);
_world->addJoint(joint);
```

- You were able to create the gear joint in Step 2. However, you cannot move this gear. That's why you enable the touching of the screen and the moving of the physics objects. In the `onTouchBegan` method, we check the physics object in the touch area. If the object didn't exist in the touch location, it returns `false`.

```
auto location = touch->getLocation();
auto shapes = _world->getShapes(location);
if (shapes.size()<=0) {
    return false;
}
```

- If the object existed in the touch location, get the physics body from the physics shape. Then, create a node on the touch location and add a physics body to this node. This node is used in the `onTouchMoved` method.

```
PhysicsShape* shape = shapes.front();
PhysicsBody* body = shape->getBody();
if (body != nullptr) {
    _touchNode = Node::create();
    auto touchBody = PhysicsBody::create(PHYSICS_INFINITY,
PHYSICS_INFINITY);
    _touchNode->setPhysicsBody(touchBody);
    _touchNode->getPhysicsBody()->setDynamic(false);
    _touchNode->setPosition(location);
    this->addChild(_touchNode);
}
```

- To add force to this object, add `PhysicsJointPin` by using `touchBody` and the touch location. Then, set the force by using the `setMaxForce` method.

```
PhysicsJointPin* joint = PhysicsJointPin::construct(touchBody,
body, location);
joint->setMaxForce(5000.0f * body->getMass());
_world->addJoint(joint);
```

6. In the `onTouchMoved` method, move the touch node as follows:

```
void HelloWorld::onTouchMoved(Touch* touch, Event* event)
{
    if (_touchNode!=nullptr) {
        _touchNode->setPosition(touch->getLocation());
    }
}
```

7. In the `onTouchEnded` method, remove the touch node as follows:

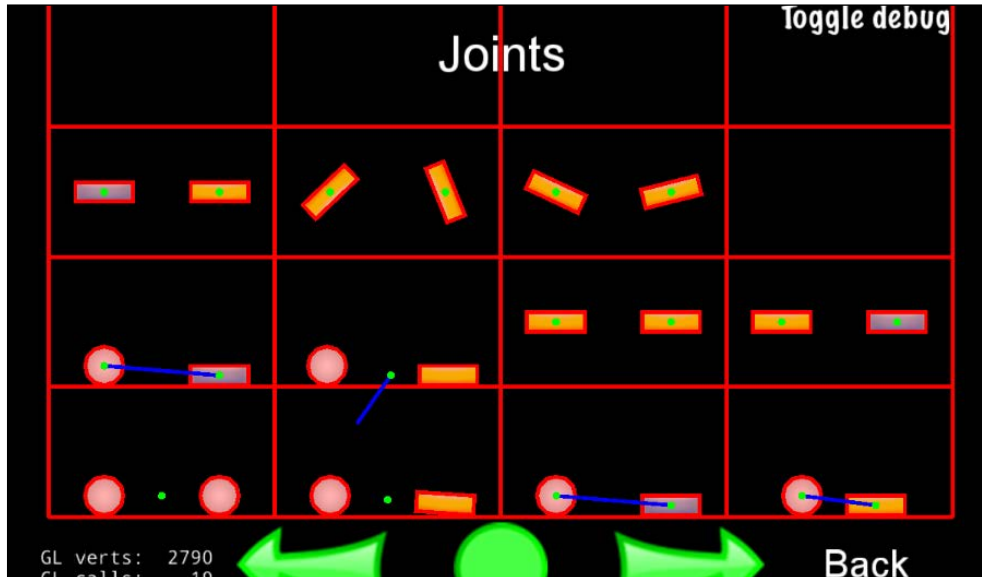
```
void HelloWorld::onTouchEnded(Touch* touch, Event* event)
{
    if (_touchNode!=nullptr) {
        _touchNode->removeFromParent();
        _touchNode = nullptr;
    }
}
```

There's more...

Cocos2d-x has a lot of joints. Each joint has a different task as given in the following table:

Joint	Description
<code>PhysicsJointFixed</code>	A fixed joint connects the two bodies together at a reference point. Fixed joints are useful for creating complex shapes that can be broken apart later.
<code>PhysicsJointLimit</code>	A limit joint imposes the maximum distance between the two bodies.
<code>PhysicsJointPin</code>	Allowing two bodies to independently rotate around the pin
<code>PhysicsJointDistance</code>	Joining two bodies with a fixed distance
<code>PhysicsJointSpring</code>	Connecting two bodies with a spring
<code>PhysicsJointRotarySpring</code>	Like a spring joint which rotates
<code>PhysicsJointRotaryLimit</code>	Like a limit joint which rotates
<code>PhysicsJointRatchet</code>	Like a socket wrench
<code>PhysicsJointGear</code>	Keeps the angular velocity ratio of a pair of bodies
<code>PhysicsJointMotor</code>	Keeps the relative angular velocity of a pair of bodies

This is difficult to explain by text. So, you should check the `cpp-tests` application that was provided by Cocos2d-x. You run the `cpp-tests` application and select `Node::Physics` from the menu. You can check the following image:



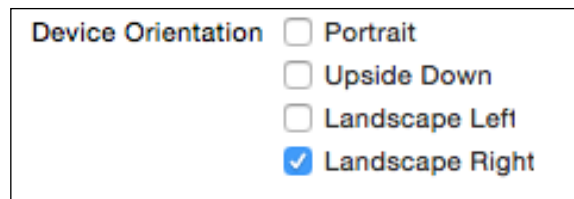
Then, you can touch or drag these physics objects, so, you can see each joint's working.

Changing gravity by using the acceleration sensor

A game with a physics engine will often change the direction of gravity by tilting the device. By doing so, it is possible to add realism in the game. In this recipe, you can change the direction of gravity by using an acceleration sensor.

Getting ready

To avoid screen rotation, you have to change some code and settings. Firstly, you should set **Device Orientation** to only **Landscape Right** as shown in the following image:



Secondly, you change the `shouldAutorotate` method's return value to false in `RootViewController.mm`.

```
- (BOOL) shouldAutorotate {
    return NO;
}
```

How to do it...

You check the acceleration sensor value in `HelloWorld.cpp` as follows:

```
Device::setAccelerometerEnabled(true);
auto listener = EventListenerAcceleration::create( [=] (Acceleration*
acc, Event* event) {
    auto gravity = Vec2(acc->x*100.0f, acc->y*100.0f);
    world->setGravity(gravity);
});
this->getEventDispatcher() ->addEventListenerWithSceneGraphPriority(li
stener, this);
```

How it works...

If you tilt the device, you can get the changing acceleration x and y values. At this time, we have 100 times the value of the x -axis and y -axis. That's why the value of acceleration is pretty small for using gravity.

```
auto gravity = Vec2(acc->x*100.0f, acc->y*100.0f);
```

While rotating the device, the home button is at the right, then it is the home position. At this time, the acceleration y value is negative. While rotating, if the home button is at the left side; the acceleration y value is positive. While rotating, if it is in the portrait position, then the acceleration x value is positive. Or, while rotating, if it is upside down, then the acceleration x value is negative. Then, to change gravity by using the acceleration sensor value, you can realize real gravity in your game.

10

Improving Games with Extra Features

The following topics will be covered in this chapter:

- ▶ Using Texture Packer
- ▶ Using Tiled Map Editor
- ▶ Getting the property of the object in the tiled map
- ▶ Using Physics Editor
- ▶ Using Glyph Designer

Introduction

For a long time, there have been a lot of tools available to you that help you in game development. Some of these tools can be used in Cocos2d-x. With the use of these tools, you can quickly and efficiently develop your game. You can, for example, use original fonts and create sprite sheets, a map like a role-playing game, complex physical objects, and so on. In this chapter, you will learn how to use these extra tools in your game development.

Using Texture Packer

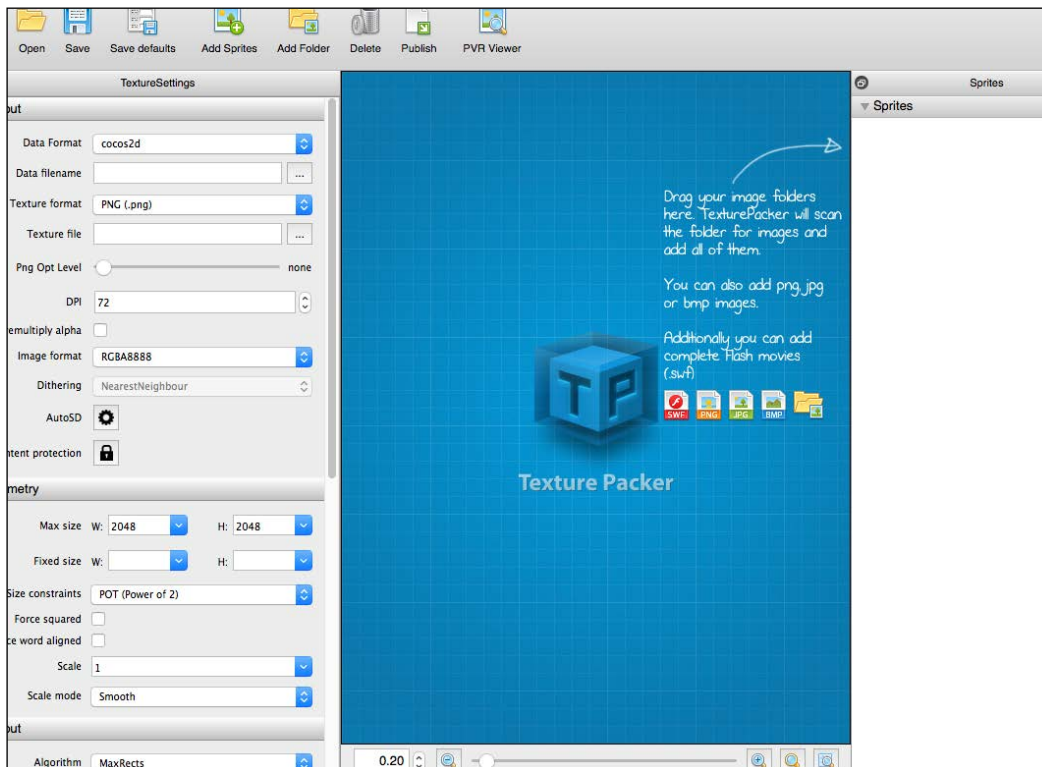
Texture Packer is a tool that can drag and drop images and publish. With the use of this tool, we can not only create sprite sheets, but also export multi sprite sheets. If there are a lot of sprites, then we need to use the command line tool when we create sprite sheets, encrypt them, and so on. In this recipe, you can use Texture Packer.

Getting ready

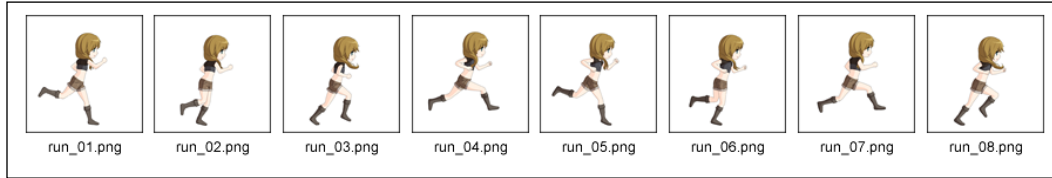
Texture Packer is a paid application. However, you can use the free trial version. If you don't have it, you can download it by visiting <https://www.codeandweb.com/texturepacker>

How to do it...

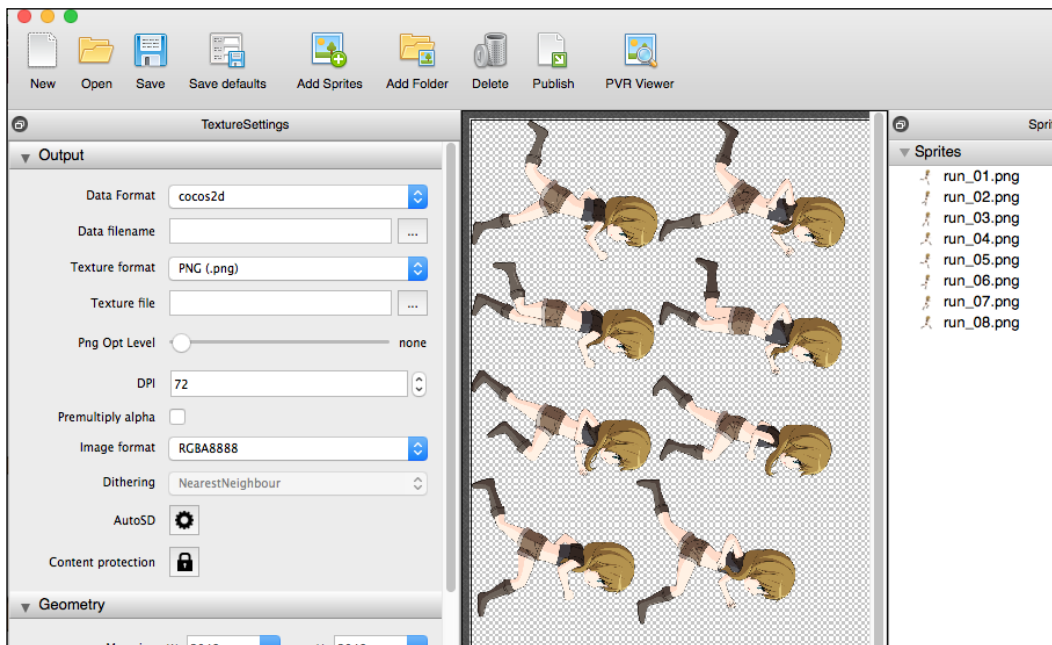
1. You need to launch Texture Packer, after which you will see a blank window appear.



- In this recipe, we will use these sprites as shown in the following screenshot:



- You simply need to drag the images into the Texture Packer window and it will automatically read all the files and arrange them.

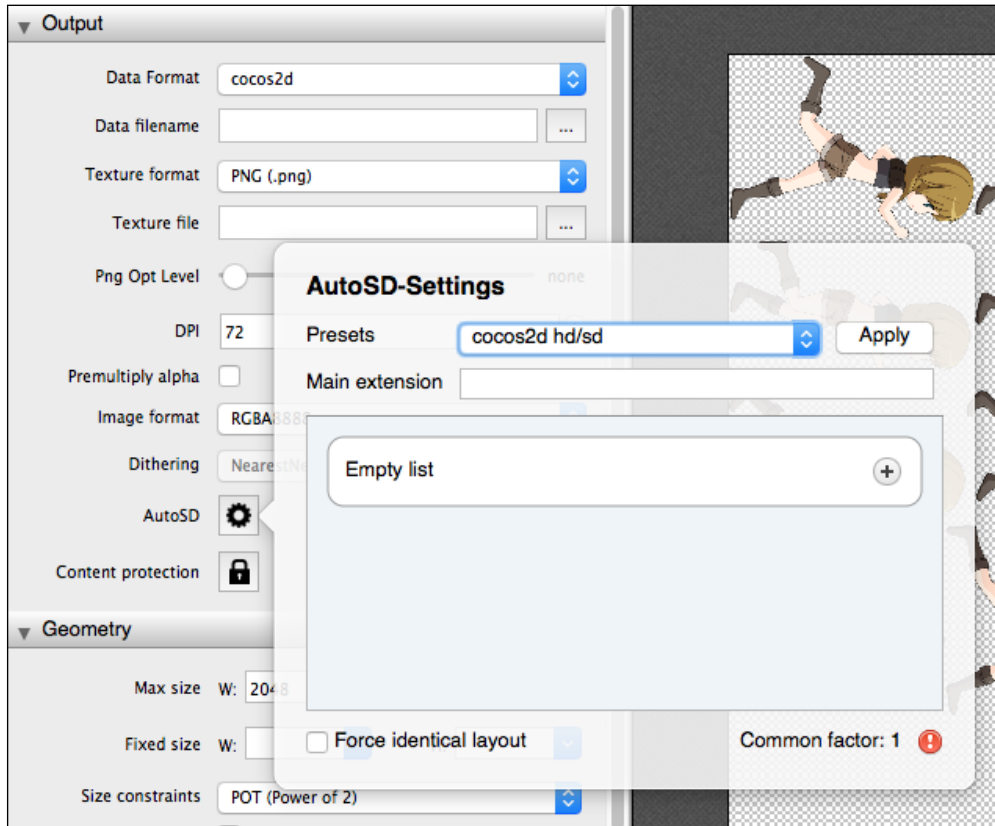


- And that's it. So let's publish the sprite sheet image and `plist` to click the **publish** button. That's how you can get the sprite sheet image and `plist`.

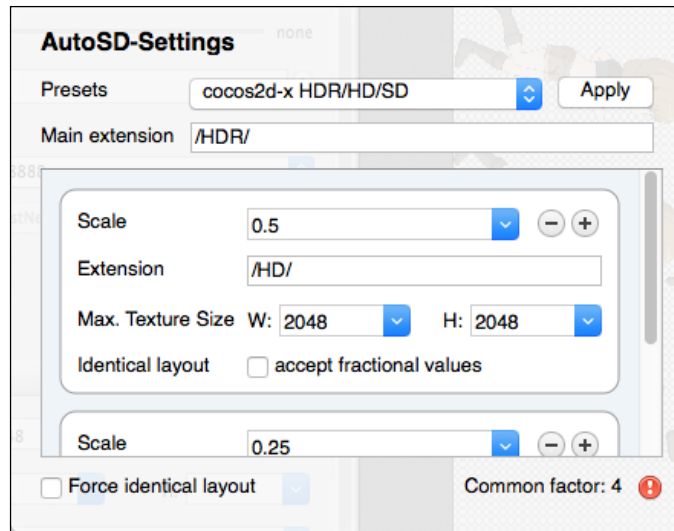
How it works...

You can get the sprite sheet image and `plist` file. In this part, we explain how to publish the sprite sheet for all devices with a single click.

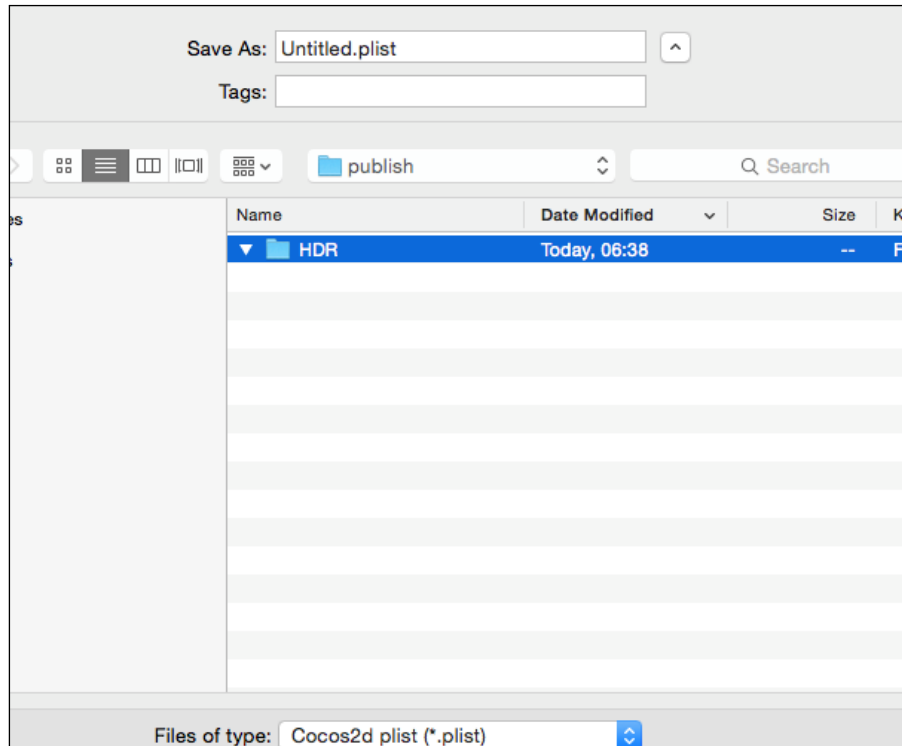
1. Click on the **AutoSD** button with the gear icon, and you will see an additional window appear, as shown in the following screenshot:



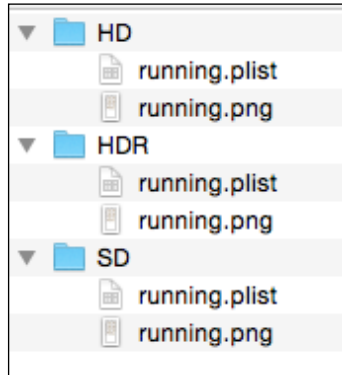
2. Select the **cocos2d-x HDR/HD/SD** and click the **Apply** button. After clicking it, setting the default scale, extension, size and so on like in the following image:



3. Next, you have to click the **publish** button, you will see the window to select the data file name. The important thing is to select the folder named HDR as in the following image:



4. Finally, you will get three size sprite sheets automatically as in the following image:



The sprite sheet in HDR folder is the largest size. The images that were dragged and dropped are HDR images. These images are good for resizing to HD or SD images.

There's more...

You can use the Texture Packer on the command line like this:

```
texturepacker foo_*.png --format cocos2d --data hoge.plist --sheet hoge.png
```

The preceding command is to make a sprite sheet named `hoge.plist` and `hoge.png` by using images named `foo_*.png`. For example, if there were `foo_1.png` to `foo_10.png` in a folder, then the sprite sheet is created from these 10 images.

In addition, the command has other options as in the following table:

Option	Description
<code>--help</code>	Display help text
<code>--version</code>	Print version information
<code>--max-size</code>	Set the maximum texture size
<code>--format cocos2d</code>	Format to write, default is cocos2d
<code>--data</code>	Name of the data file to write
<code>--sheet</code>	Name of the sheet to write

There are a lot of options other than that. You can see another options by using the following command:

```
texturepacker --help
```

Using Tiled Map Editor

A tiled map is a grid of cells where the value in the cell indicates what should be at the location. For example, (0,0) is a road, (0,1) is a grass, (0,2) is a river and so on. Tiled maps are very useful but they are pretty hard to create by hand. **Tiled** is a tool that can be used to just create tiled maps. Tiled is a free application. However, this application is a very powerful, useful and popular tool. There are various kinds of Tiled Map, for example, 2D maps such as Dragon Quest, Horizontal scrolling game map such as Super Mario and so on. In this recipe, you can basically use texture packer.

Getting ready

If you don't have Tiled Map Editor, you can download it from <https://www.mapeditor.org/>.

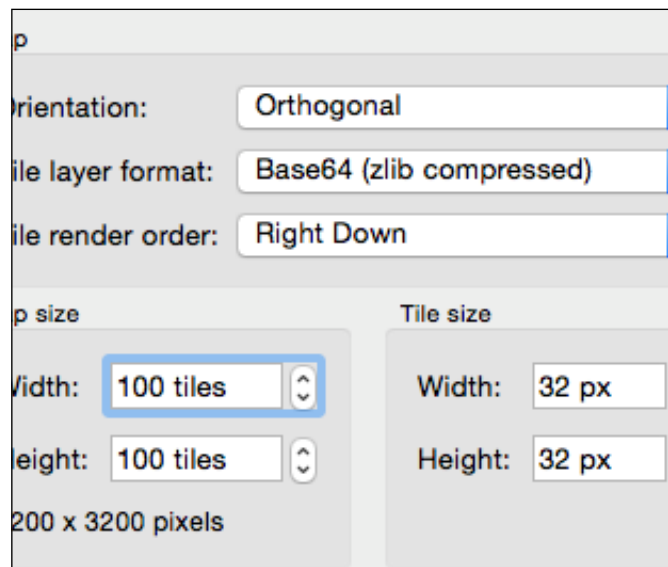
And then, after downloading it, you will install the application and copy the `example` folder in the `.dmg` file, into the working space of your computer.

Tiled Map Editor is free application. However, you can donate to this software if you like.

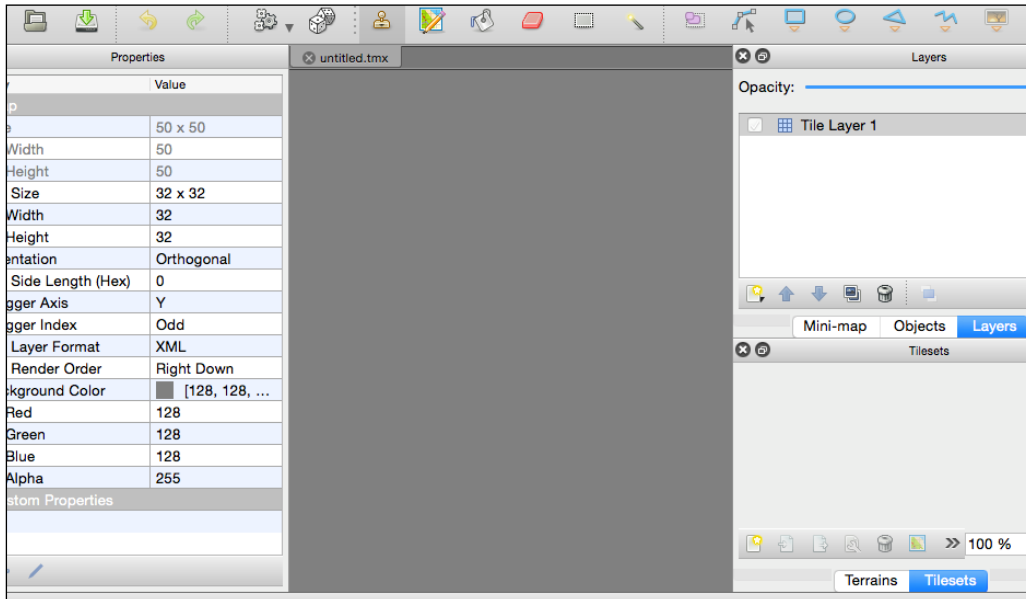
How to do it...

In this part, we explain how to create a new map from scratch with the Tiled tool.

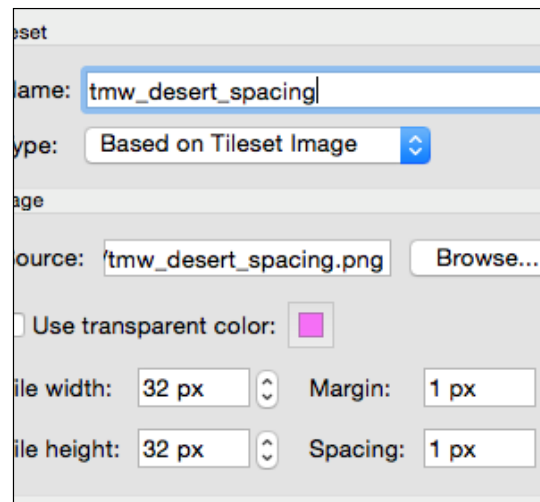
1. Launch Tiled and selecting **File | New** in the menu. Open the new additional window as in the following image:



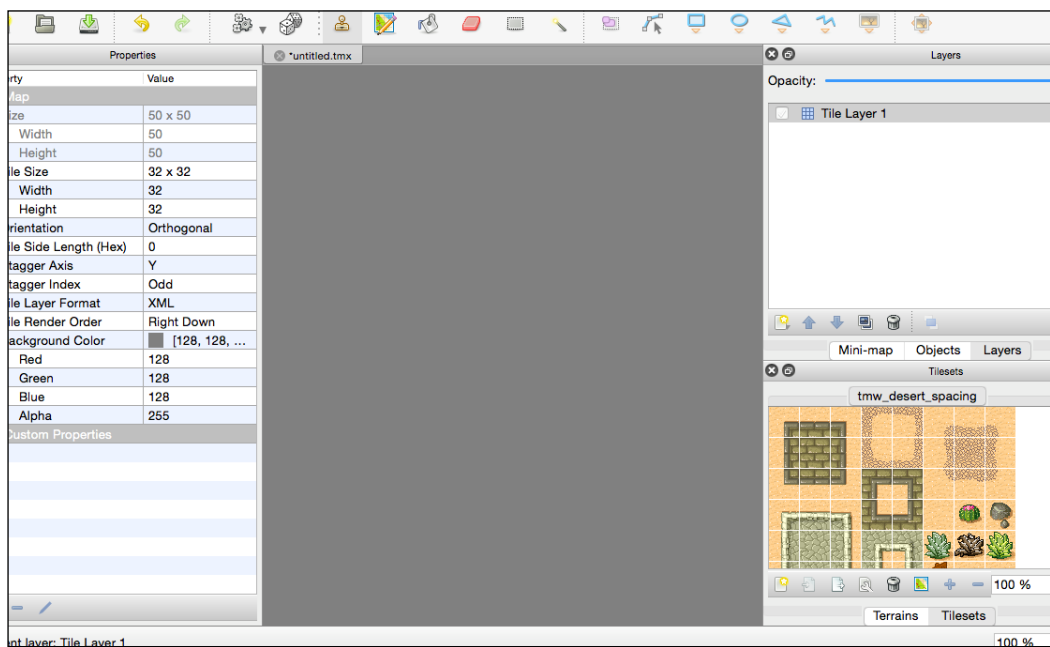
2. Select XML in **Tile layer format** and change **Width** and **Height** in **Map size** to 50 tiles. Finally, click **OK**. So you can see the Tiled's window as in the following image:



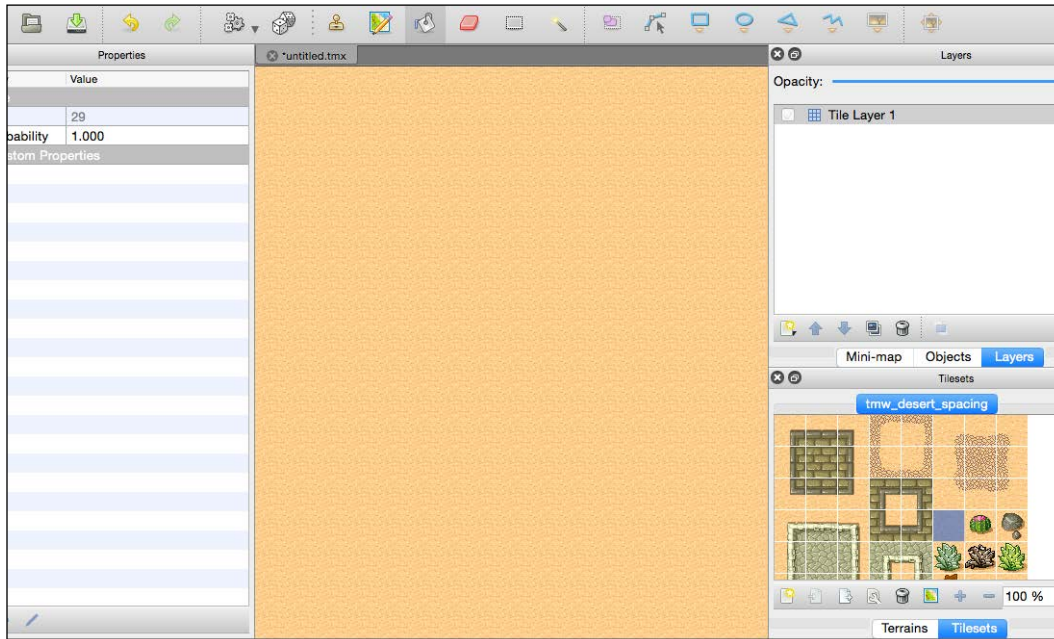
3. Select **Map | New Tileset...** in the menu. You can select the tileset window. Select the tileset image by clicking the **Browse...** button in the middle of the window. In this case, you will select `tmw_desert_spacing.png` file in Tiled's `example` folder. This tileset has tiles with a width and height of 32px and a margin and spacing of 1px. So you have to change these values as shown in the following screenshot:



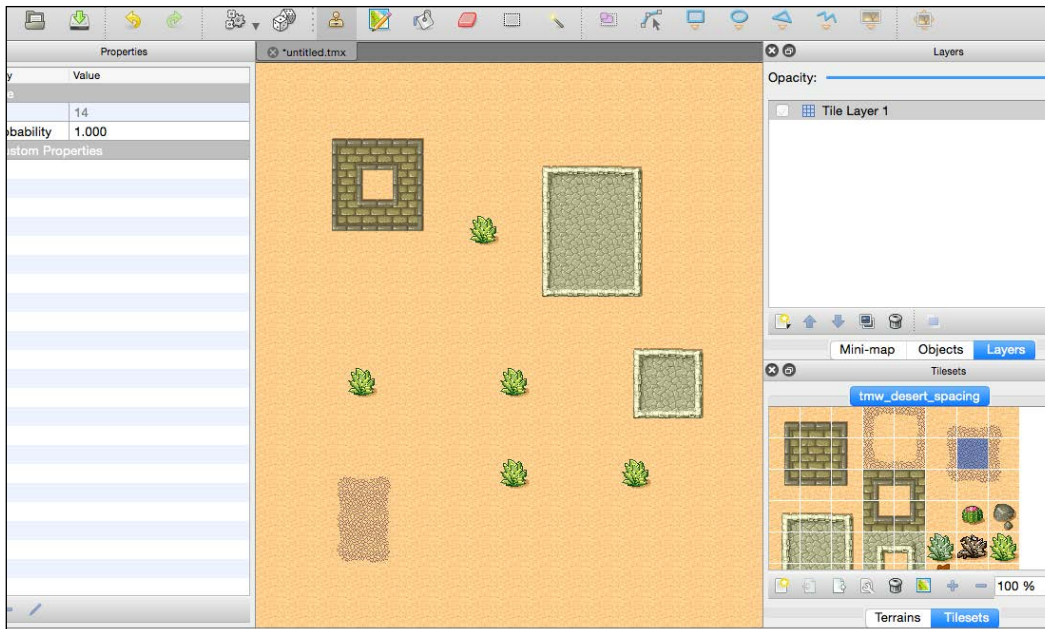
4. Finally, click on the **OK** button, and you will see the new editor window as shown in the following screenshot:



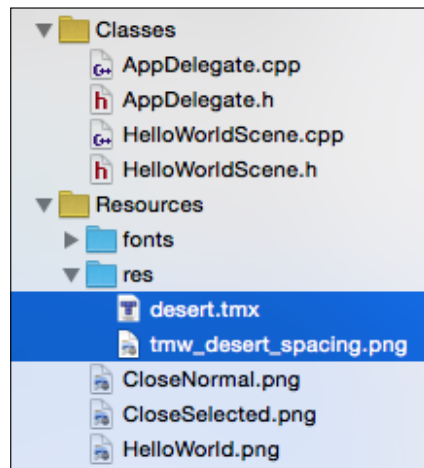
- Next, let's try to paint the ground layer using the tile that you selected. Select the tile from the right and lower panes, and select the bucket icon in the tool bar. Then, click on the map, and you will see the ground painted with the same tile.



- You can arrange the tiles on the map. Select the tile in the lower right pane and select the stamp icon in the tool bar. Then, click on the map. That's how you can put the tile on the map.



7. After you have finished arranging the map, you need to save it as a new file. Go to **File** | **Save as...** in the menu and save the new file that you made. To use Cococs2d-x, you have to add the `tmx` file and tileset image file into the `Resources/res` folder in your project. In this recipe, we added `desert.tmx` and `tmw_desert_spacing.png` in Tiled's `example` folder.

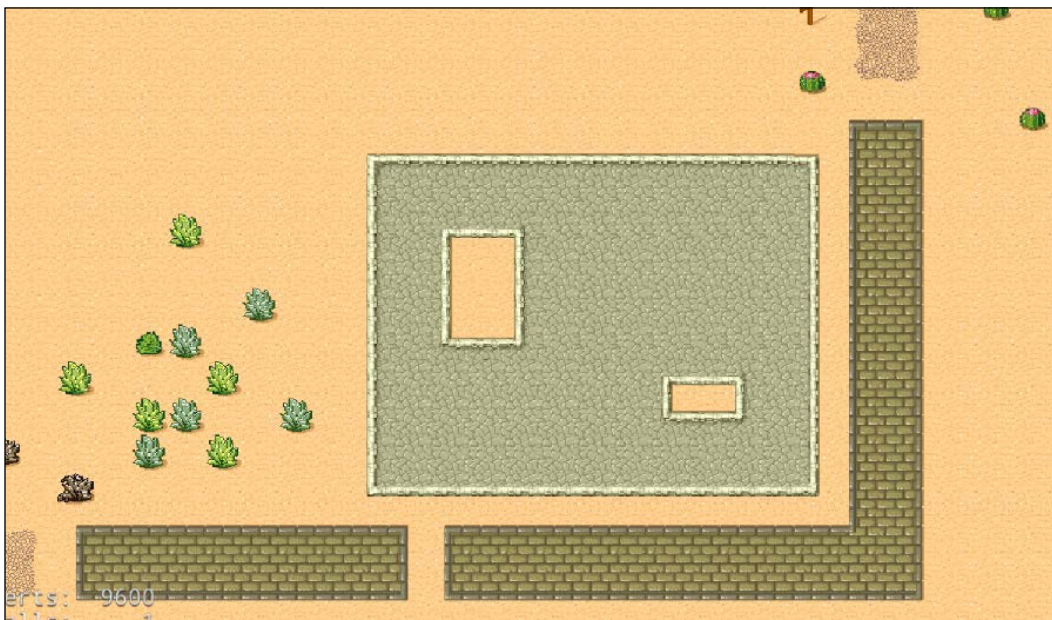


8. From now on, you have to work in Xcode. Edit the `HelloWorld::init` method as shown in the following code:

```
bool HelloWorld::init()
{
    if ( !Layer::init() )
    {
        return false;
    }
    Vec2 origin = Director::getInstance()-
    >getVisibleOrigin();
    _map = TMXTiledMap::create("res/desert.tmx");
    _map->setPosition(Vec2()+origin);
    this->addChild(_map);

    return true;
}
```

9. After building and running, you can see the following image on the simulator or devices:



How it works...

The files that Tiled map needs are the `tmx` file and `tileset` image file. That's why you have to add these files into your project. You can see the Tiled map object using the `TMXTiledMap` class. You have to specify the `tmx` file path to the `TMXTiledMap::create` method. The `TMXTiledMap` object is `Node`. You can see the tiled map only when you add the `TMXTiledMap` object using the `addChild` method.

```
_map = TMXTiledMap::create("res/desert.tmx");
_map->setPosition(Vec2()+origin);
this->addChild(_map);
```



`TMXTiledMap` object's anchor position is `Vec2(0, 0)`. The normal node's anchor position is `Vec2(0.5f, 0.5f)`.

There's more...

The tiled map is huge. So, we try to move the map by scrolling it. In this case, you touch the screen and scroll the map by the distance from the touching point to the center of the screen.

1. Add the following code in the `HelloWorld::init` method:

```
auto touchListener = EventListenerTouchOneByOne::create();
touchListener->onTouchBegan =
CC_CALLBACK_2(HelloWorld::onTouchBegan, this);
touchListener->onTouchEnded =
CC_CALLBACK_2(HelloWorld::onTouchEnded, this);
_eventDispatcher-
>addEventListenerWithSceneGraphPriority(touchListener,
this);
```

2. Define the `touch` method and some properties in `HelloWorldScene.h` as shown in the following code:

```
bool onTouchBegan(cocos2d::Touch* touch, cocos2d::Event*
event);
void onTouchEnded(cocos2d::Touch* touch, cocos2d::Event*
event);
void update(float dt);
cocos2d::Vec2 _location;
cocos2d::TMXTiledMap* _map;
```


3. Add the touch method in `HelloWorldScene.cpp` as shown in the following code:

```
bool HelloWorld::onTouchBegan(Touch* touch, Event* event)
{
    return true;
}

void HelloWorld::onTouchEnded(Touch* touch, Event* event)
{
    auto size = Director::getInstance()->getVisibleSize();
    auto origin = Director::getInstance()-
        >getVisibleOrigin();
    auto center = Vec2(size/2)+origin;
    _location = touch->getLocation() - center;
    _location.x = floorf(_location.x);
    _location.y = floorf(_location.y);
    this->scheduleUpdate();
}
```

4. Finally, add the update method in `HelloWorldScene.cpp` as shown in the following code:

```
void HelloWorld::update(float dt)
{
    auto mapSize = _map->getContentSize();
    auto winSize = Director::getInstance()-
        >getVisibleSize();
    auto origin = Director::getInstance()-
        >getVisibleOrigin();

    auto currentLocation = _map->getPosition();
    if (_location.x > 0) {
        currentLocation.x--;
        _location.x--;
    } else if (_location.x < 0) {
        currentLocation.x++;
        _location.x++;
    }
    if (_location.y > 0) {
        currentLocation.y--;
        _location.y--;
    } else if (_location.y < 0) {
        currentLocation.y++;
        _location.y++;
    }
}
```

```

    if (currentLocation.x > origin.x) {
        currentLocation.x = origin.x;
    } else if (currentLocation.x < winSize.width + origin.x
- mapSize.width) {
        currentLocation.x = winSize.width + origin.x -
        mapSize.width;
    }
    if (currentLocation.y > origin.y) {
        currentLocation.y = origin.y;
    } else if (currentLocation.y < winSize.height + origin.y
- mapSize.height) {
        currentLocation.y = winSize.height + origin.y -
        mapSize.height;
    }

    _map->setPosition(currentLocation);
    if (fabsf(_location.x)<1.0f && fabsf(_location.y)<1.0f) {
        this->unscheduleUpdate();
    }
}
}

```

After that, run this project and touch the screen. This is how you can move the map in the direction that you swipe.

Getting the property of the object in the tiled map

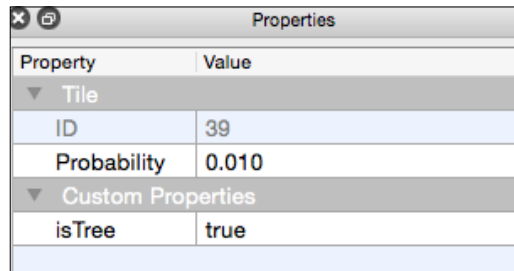
Now, you can move the Tiled map. However, you might notice the object on the map. For example, if there is a wood or wall in the direction of movement, you can't move in that direction beyond that object. In this recipe, you will notice the object on the map by getting the property of it.

Getting ready

In this recipe, you will make a new property of the tree object and set a value to it.

1. Launch the Tiled application and reopen the `desert.tmx` file.
2. Select the tree object in the **Tilesets** window.
3. Add a new property by clicking on the plus icon in the lower left corner in the **Properties** window. Then, a window will pop up specifying the property's name. Enter `isTree` in the text area.

- After you name the new property, it will be shown in the properties list. However, you will find that its value is empty. So, you have to set the new value to it. In this case, you need to set a true value as shown in the following image:



Property	Value
▼ Tile	
ID	39
Probability	0.010
▼ Custom Properties	
isTree	true

- Save it and update `desert.tmx` in your project.

How to do it...

In this recipe, you will get the property of the object that you touched.

- Edit the `HelloWorld::init` method to show the tiled map and add the event listener for touching.

```
bool HelloWorld::init()
{
    if ( !Layer::init() )
    {
        return false;
    }
    Vec2 origin = Director::getInstance()->getVisibleOrigin();
    _map = TMXTiledMap::create("res/desert.tmx");
    _map->setPosition(Vec2()+origin);
    this->addChild(_map);

    auto touchListener = EventListenerTouchOneByOne::create();
    touchListener->onTouchBegan =
    CC_CALLBACK_2(HelloWorld::onTouchBegan, this);
    _eventDispatcher-
    >addEventListenerWithSceneGraphPriority(touchListener, this);

    return true;
}
```

2. Add the `HelloWorld::getTilePosition` method. You can get the tile's grid row/column position if you called this method by specifying the touch position.

```
Vec2 HelloWorld::getTilePosition(Vec2 point)
{
    auto mapContentSize = _map->getContentSize();
    auto tilePoint = point - _map->getPosition();
    auto tileSize = _map->getTileSize();
    auto mapRowCol = _map->getMapSize();
    auto scale = mapContentSize.width / (mapRowCol.width *
tileSize.width);
    tilePoint.x = floorf(tilePoint.x / (tileSize.width * scale));
    tilePoint.y = floorf((mapContentSize.height -
tilePoint.y)/(tileSize.height*scale));
    return tilePoint;
}
```

3. Finally, you can get the properties of the object that you touch. Add the `HelloWorld::onTouchBegan` method as shown in the following code:

```
bool HelloWorld::onTouchBegan(Touch* touch, Event* event)
{
    auto touchPoint = touch->getLocation();
    auto tilePoint = this->getTilePosition(touchPoint);
    TMXLayer* groundLayer = _map->getLayer("Ground");
    int gid = groundLayer->getTileGIDAt(tilePoint);
    if (gid!=0) {
        auto properties = _map-
>getPropertiesForGID(gid).asValueMap();
        if (properties.find("isTree")!=properties.end()) {
            if(properties.at("isTree").asBool()) {
                CLOG("it's tree!");
            }
        }
    }
    return true;
}
```

Let's build and run this project. If you touched the tree to which you set the new `isTree` property, you can see *it's tree!* in the log.

How it works...

There are two points in this recipe. The first point is getting the tile's row/column position on the tiled map. The second point is getting the properties of the object on the tiled map.

Firstly, let's explain how to get the tiles' row/column position on the tiled map.

1. Get the map size using the `TMXTiledMap::getContentSize` method.

```
auto mapContentSize = _map->getContentSize();
```
2. Calculate the point on the map from the touching point and map position.

```
auto tilePoint = point - _map->getPosition();
```
3. Get the tile size using the `TMXTiledMap::getTileSize` method.

```
auto tileSize = _map->getTileSize();
```
4. Get the row/column of the tile in the map using the `TMXTiledMap::getMapSize` method.

```
auto mapRowCol = _map->getMapSize();
```
5. Get the magnification display using the original size called `mapContentSize` and real size calculated by the column's width and tile's width.

```
auto scale = mapContentSize.width / (mapRowCol.width * tileSize.width);
```
6. The origin of coordinates for the tiles is located in the upper left corner. That's why the tile's row/column position of the tile that you touched is calculated using the tile's size, the row, and magnification display as shown in the following code:

```
tilePoint.x = floorf(tilePoint.x / (tileSize.width * scale));  
tilePoint.y = floorf((mapContentSize.height -  
tilePoint.y) / (tileSize.height * scale));
```

`tilePoint.x` is the column position and `tilePoint.y` is row position.

Next, let's take a look at how to get the properties of the object on the Tiled map.

1. Get the row/column position of the tile that you touched using the touching point.

```
auto touchPoint = touch->getLocation();  
auto tilePoint = this->getTilePosition(touchPoint);
```
2. Get the layer called "Ground" from the tiled map.

```
TMXLayer* groundLayer = _map->getLayer("Ground");
```
3. There are the objects on this layer called Ground. Get the `TileGID` from this layer using row/column of the tile.

```
int gid = groundLayer->getTileGIDat(tilePoint);
```

4. Finally, get the properties as `ValueMap` from the map using the `TMXTiledMap::getPropertiesForGID` method. Then, get the `isTree` property's value from them as shown in the following code:

```
auto properties = _map->getPropertiesForGID(gid).asValueMap();
if (properties.find("isTree")!=properties.end()) {
    if (properties.at("isTree").asBool()) {
        CCLOG("it's tree!");
    }
}
```

In this recipe, we showed only the log. However, in your real game, you will add the point to the object, explosions and so on.

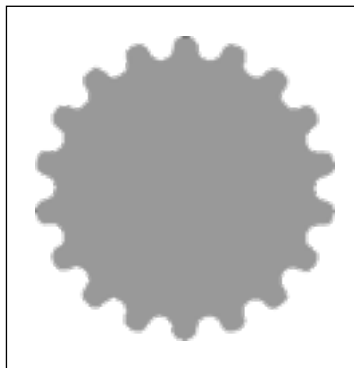
Using Physics Editor

In *Chapter 9, Controlling Physics*, you learned about **Physics Engine**. We can create physics bodies to use Cocos2d-x API. However, we can only create a circle shape or a box shape. Actually, you have to use complex shapes in real games. In this recipe, you will learn how to create a lot of shapes using **Physics Editor**.

Getting ready

Physics Editor is created by the same company that created Texture Packer. Physics Editor is a paid application. But you can use a free trial version. If you don't have it, you can download it by visiting the <https://www.codeandweb.com/physicseditor>

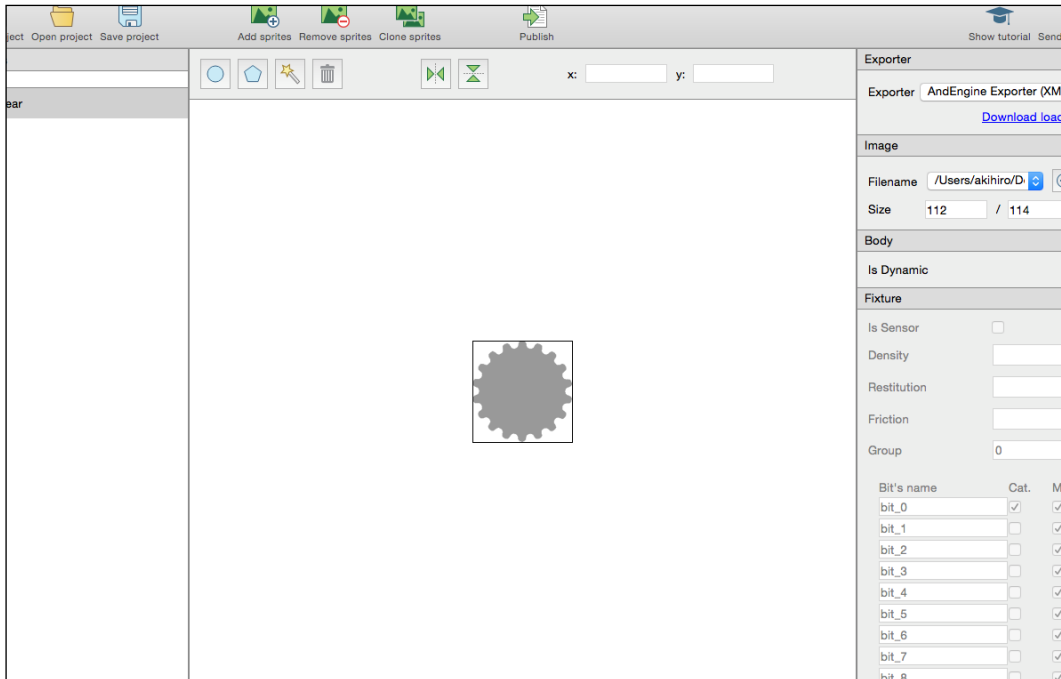
Here, you prepare the image to use this tool. Here, we will use the following image that is similar to a gear. This image's name is `gear.png`.



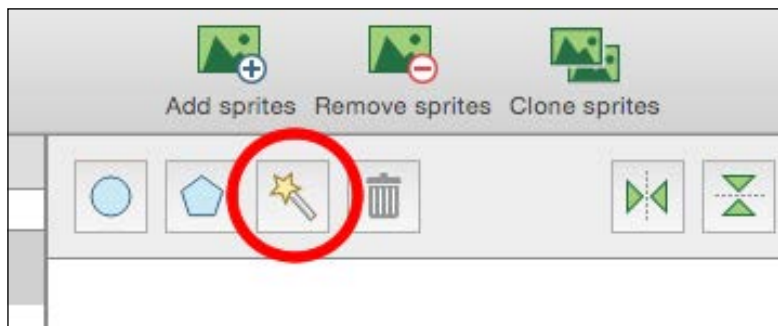
How to do it...

First of all, you will create a physics file to use Physics Editor.

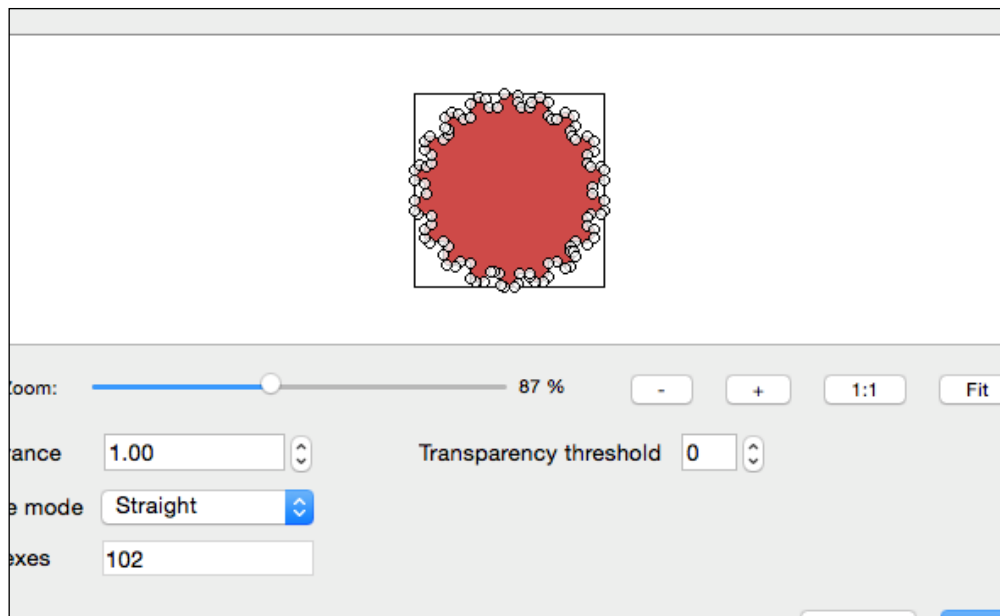
1. Launch Physics Editor. Then, drag the image gear.png to the left pane.



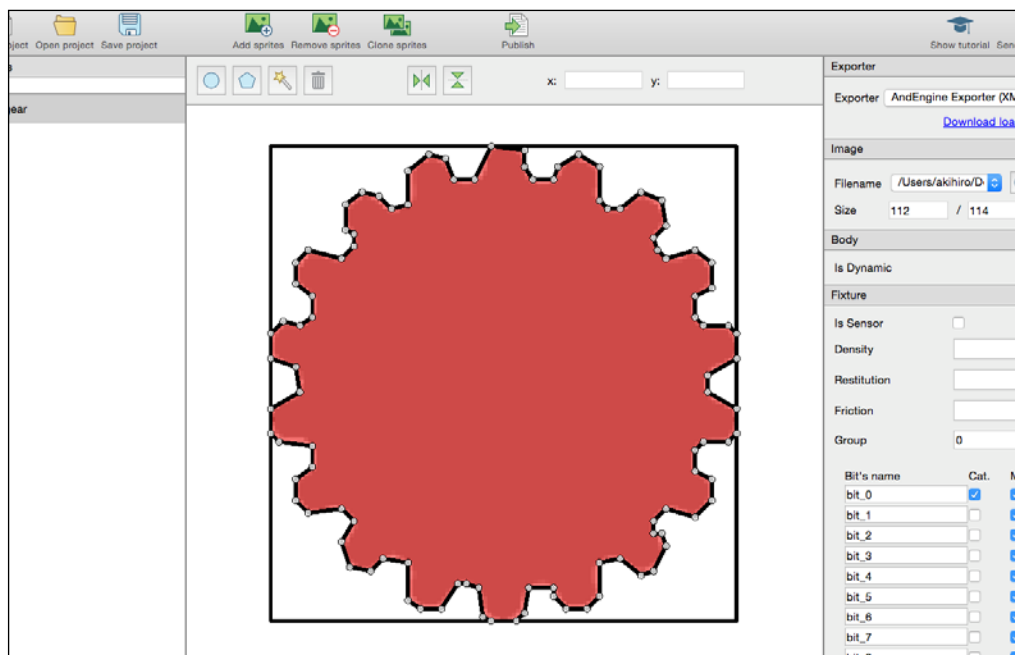
2. Click on the shaper tracer icon that is the third icon from the left in the tool bar. The shaper tracer icon is shown in the following image:



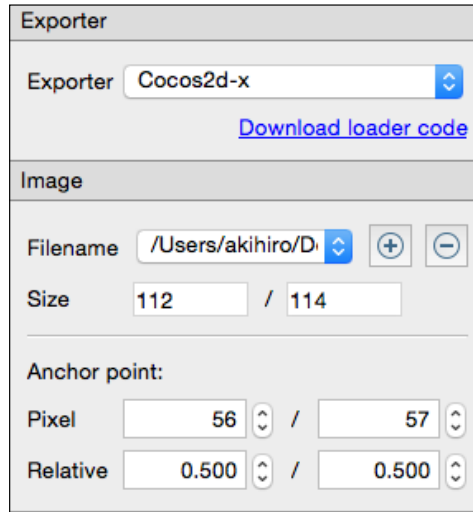
3. After this, you can see the pop-up window as shown in the following image:



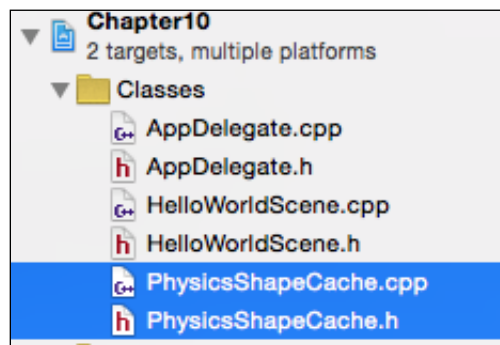
You can change the **Tolerance** value. If the **Vertexes** value is too big, the renderer is slow. So you set the suitable **Vertexes** value to change the **Tolerance** value. Finally, click on the **OK** button. You will see the following:



4. Select `Cocos2d-x` in **Exporter**. In this tool, the anchor point's default value is $\text{Vec2}(0, 0)$. In `Cocos2d-x`, the anchor point's default is $\text{Vec2}(0.5f, 0.5f)$. So you should change the anchor point to the center as shown in the following screenshot:



5. Check the checkboxes for **Category**, **Collision**, and **Contact**. You need to scroll down to see this window in the right pane. You can check all the checkboxes and click all buttons that are in the bottom of the right pane.
6. Publish the `plist` file to use this shape in `Cocos2d-x`. Click on the **Publish** button and save as the previous name.
7. You can see the **Download loader code** link under the **Exporter** selector. Click on the link. After this, open the browser and browse to the github page. `Cocos2d-x` cannot load Physics Editor's `plist`. However, the loader code is provided in github. So you have to clone this project and add the codes in the `Cocos2d-x` folder in the project.



Next, you will write code to create the physics bodies by using the Physics Editor data. In this case, the gear object will appear at the touching point.

1. Include the file `PhysicsShapeCache.h`.

```
#include "PhysicsShapeCache.h"
```

2. Create a scene with the physics world as shown in the following code:

```
Scene* HelloWorld::createScene()
{
    auto scene = Scene::createWithPhysics();
    auto layer = HelloWorld::create();
    PhysicsWorld* world = scene->getPhysicsWorld();
    world->setDebugDrawMask(PhysicsWorld::DEBUGDRAW_ALL);
    scene->addChild(layer);
    return scene;
}
```

3. Create a wall of the same screen size in the scene and add the touching event listener. Then, load the Physics Editor's data as shown in the following code:

```
bool HelloWorld::init()
{
    if ( !Layer::init() )
    {
        return false;
    }

    Size visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();

    auto wall = Node::create();
    auto wallBody = PhysicsBody::createEdgeBox(visibleSize,
        PhysicsMaterial(0.1f, 1.0f, 0.0f));
    wallBody->setContactTestBitmask(true);
    wall->setPhysicsBody(wallBody);
    wall->setPosition(Vec2(visibleSize/2)+origin);
    this->addChild(wall);

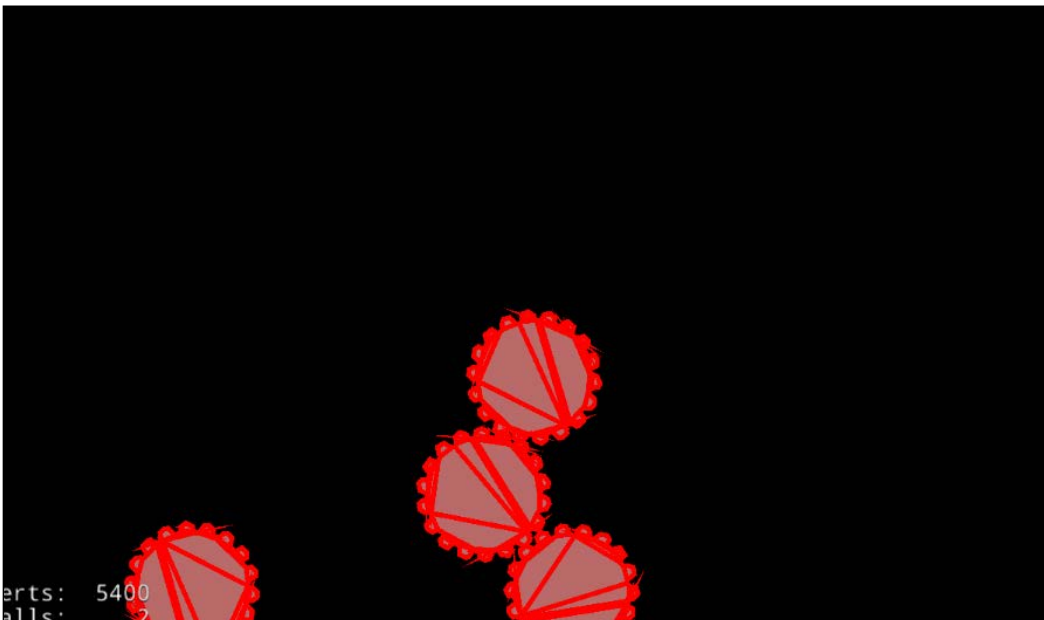
    auto touchListener = EventListenerTouchOneByOne::create();
    touchListener->onTouchBegan =
        CC_CALLBACK_2(HelloWorld::onTouchBegan, this);
    _eventDispatcher-
        >addEventListenerWithSceneGraphPriority(touchListener, this);
}
```

```
PhysicsShapeCache::getInstance() -  
>addShapesWithFile("res/gear.plist");  
  
return true;  
}
```

4. Make the gear objects perform when touching the screen as shown in the following code:

```
bool HelloWorld::onTouchBegan(Touch* touch, Event* event)  
{  
    auto touchPoint = touch->getLocation();  
    auto body = PhysicsShapeCache::getInstance() -  
        >createBodyWithName("gear");  
    auto sprite = Sprite::create("res/gear.png");  
    sprite->setPhysicsBody(body);  
    sprite->setPosition(touchPoint);  
    this->addChild(sprite);  
    return true;  
}
```

5. After this, build and run this project. After touching the screen, the gear objects appear at the touching point.



How it works...

1. Firstly, you have to add two files, `plist` and `image`. Physics body is defined in the `plist` file that you published with Physics Editor. However, you use the gear image to create a sprite. Therefore, you have to add the `plist` file and `gear.png` into your project.

2. Cocos2d-x cannot read Physics Editor's data. Therefore, you have to add the loader class that is provided in github.

3. To use the Physics Engine, you have to create a scene with Physics World and you should set the debug draw mode to easy, to better understand physics bodies.

```
auto scene = Scene::createWithPhysics();
auto layer = HelloWorld::create();
PhysicsWorld* world = scene->getPhysicsWorld();
world->setDebugDrawMask(PhysicsWorld::DEBUGDRAW_ALL);
```

4. Without border or walls, the physics objects will drop out of the screen. So you have to put up a wall that is the same size as the screen.

```
auto wall = Node::create();
auto wallBody = PhysicsBody::createEdgeBox(visibleSize,
PhysicsMaterial(0.1f, 1.0f, 0.0f));
wallBody->setContactTestBitmask(true);
wall->setPhysicsBody(wallBody);
wall->setPosition(Vec2(visibleSize/2)+origin);
this->addChild(wall);
```

5. Load the physics data's `plist` that was created by Physics Editor. The `PhysicsShapeCache` will load the `plist` at once. After that, the physics data is cached in the `PhysicsShapeCache` class.

```
PhysicsShapeCache::getInstance() -
>addShapesWithFile("res/gear.plist");
```

6. In the `HelloWorld::onTouchBegan` method, create the gear object at the touching point. You can create physics body using the `PhysicsShapeCache::createBodyWithName` method with physics object data.

```
auto body = PhysicsShapeCache::getInstance() -
>createBodyWithName("gear");
```

Using Glyph Designer

In games, you have to use text frequently. In which case, if you used the system font to display the text, you will have some problems. That's why there are different fonts for each device. The bitmap fonts are faster to render than the TTF fonts. So, Cocos2d-x uses the bitmap font to display the fps information in the bottom-left corner. Therefore, you should add the bitmap font into your game to display the text. In this recipe, you will learn how to use **Glyph Designer** which is the tool to make the original bitmap font and how to use the bitmap font in Cocos2d-x.

Getting ready

Glyph Designer is a paid application. But you can use a free trial version. If you don't have it, you can download it by visiting the following URL:

<https://71squared.com/glyphdesigner>

Next, we will find a free font that fits your game's atmosphere. In this case, we will use the font called `Arcade` from the dafont site (<http://www.dafont.com/arcade-ya.font>). After downloading it, you need to install it to your computer.

On the dafont site, there are a lot of fonts. However, the font license is different for each font. If you used the font, you need to check its license.

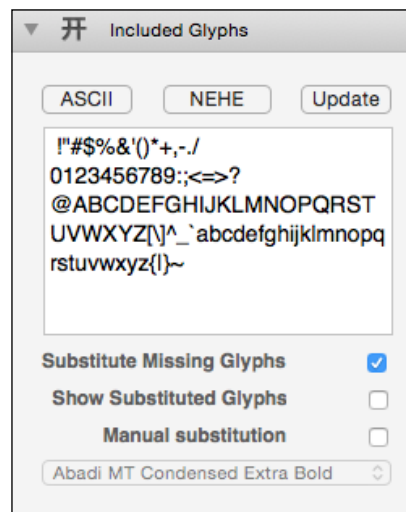
How to do it...

In this section, you will learn how to use Glyph Designer.

1. Launch Glyph Designer. In the left pane, there are all the fonts that are installed on your computer. You can choose the font that you want to use in your game from there. Here we will use `Arcade` font that you downloaded a short time ago. If you didn't install it yet, you can load it. To load the font, you have to click on the **Load Font** button in the tool bar.



- After selecting or loading the font, it is displayed in the center pane. If your game used a part of the font, you have to hold the characters that you need to save memory and the application capacity. To select the characters, you can use the **Include Glyphs** window in the right pane. You need to scroll down to see this window in the right pane.



3. The others, you can specify the size, color, and shadow. In the **font color** option, you can set a gradient.
4. Finally, you can create an original font by clicking on the `Export` icon on the right side of the tool bar.
5. After exporting, you will have the two files that have the extension of `.fnt` and `.png`.

How it works...

The bitmap font has two files, `.fnt` and `.png`. These files are paired for use in the bitmap font. Now, you will learn how to use bitmap fonts in Cocos2d-x.

1. You have to add the font that were created in Glyph Designer, into the `Resources/font` folder in your project.
2. Add the following code to display "Cocos2d-x" in your game.

```
auto label = Label::createWithBMFont("fonts/arcade.fnt",  
"Cocos2d-x");  
label->setPosition(Vec2(visibleSize/2)+origin);  
this->addChild(label);
```

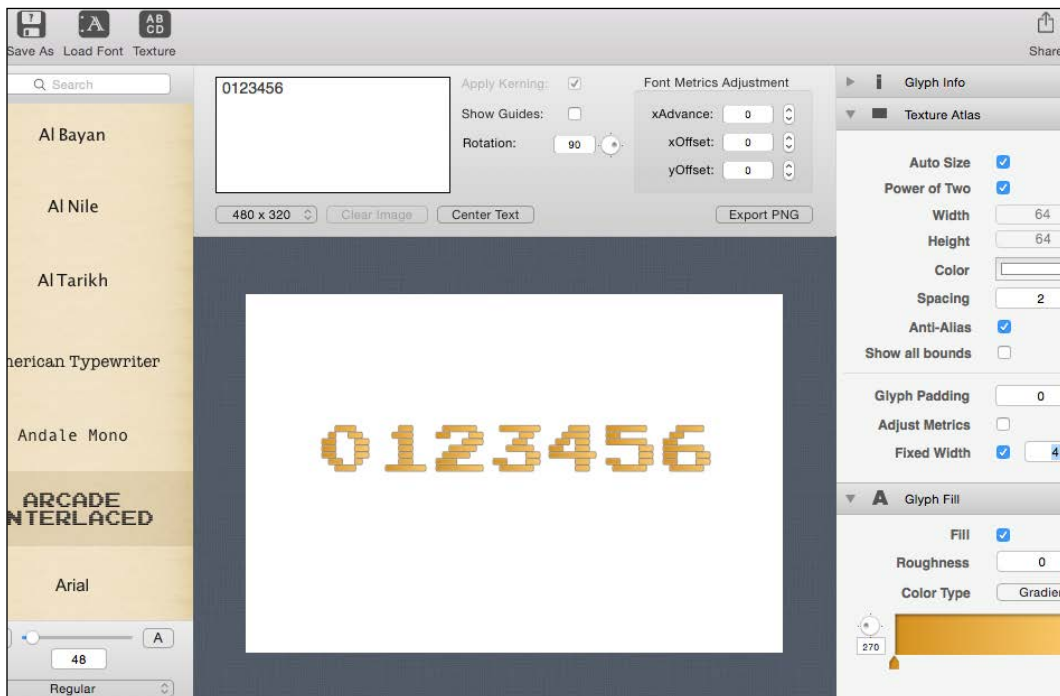
3. After building and running your project, you will see the following:



There's more...

Some fonts aren't monospaced. The true type font is good enough for use in a word-processor. However, the monospaced font is more attractive. For example, the point character needs to use the monospaced font. When you want to make the monospaced font into a non-monospaced font, you can go through the following steps:

1. Check the checkbox named **Fixed Width** in **Texture Atlas** in right pane.
2. Preview your font and click on the **Preview** icon in the tool bar. Then, you can check the characters that you want to check in the textbox.
3. If you want to change the character spacing, then you need to change the number next to the checkbox of **Fixed Width**.



11

Taking Advantages

The following topics will be covered in this chapter:

- ▶ Using encrypted sprite sheets
- ▶ Using encrypted zip files
- ▶ Using encrypted SQLite files
- ▶ Creating Observer Pattern
- ▶ Networking with HTTP

Introduction

Until now, we have explained basic technical information in Cocos2d-x. It supports the development of games on a smartphone. Actually, you can create your original games using basic functions of Cocos2d-x. However, if your game is a major hit, cheaters might attempt to crack the code. Therefore, there are cases where encryption is needed to prevent unauthorized access to your game data. Encryption is an important aspect in game development because it helps you to protect your code and prevent people from ruining the overall experience of the game, and it also prevents illegal hacking of game. In this chapter, you will learn how to encrypt your game resources.

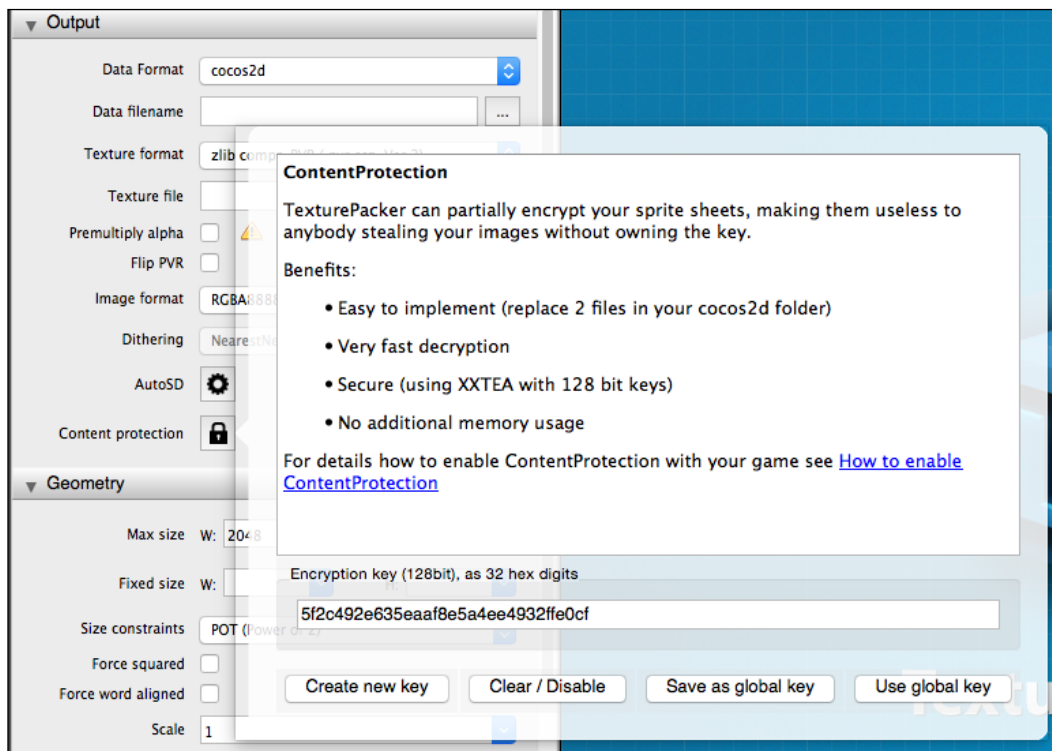
Using encrypted sprite sheets

It is pretty easy for a hacker to extract resource files from the application. This is a huge concern for copyright. Sprite sheets can be encrypted very easily using `TexturePacker`. In this recipe, you will learn how to encrypt your sprites to protect them from hackers and cheaters.

How to do it...

To encrypt sprite sheets using TexturePacker, you need to set it on the left pane of TexturePacker. Then, you need to follow the steps written here to successfully encrypt your sprite.

1. Change the Texture format to `zlib compr. PVR(.pvr.ccz, Ver.2)`
2. Click on the **ContentProtection** icon, and you will see the additional window in which to set the password.
3. Type the encryption key in the text input area as shown in the following screenshot. You can type in your favorite key. However, it is difficult to type in 32 hex digits and thus, you can just click on the **Create new key** button. After clicking it, you will find that it automatically inputs the **Encryption key**.

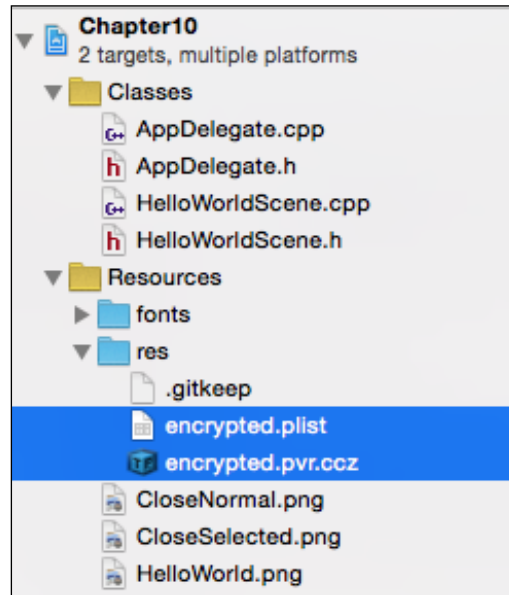


4. Take a note of this encryption key. This is the key you will need to decrypt the files that are encrypted.
5. Finally, you can publish the encrypted sprite sheet.

How it works...

Now, let's have a look on how to use these encrypted sprite sheets.

1. Add the encrypted sprite sheet to your project as shown in the following image:



2. Include the `ZipUtils` class in `HelloWorld.cpp` to decrypt.


```
#include "ZipUtils.h"
```
3. Set the encrypting key that is used for encryption by `TexturePacker`.


```
ZipUtils::setPvrEncryptionKey
    (0x5f2c492e, 0x635eaaaf8, 0xe5a4ee49, 0x32ffe0cf);
```
4. Finally, the sprite is created using the encrypted sprite sheet.

```
Size visibleSize = Director::getInstance() -
>getVisibleSize();
Vec2 origin = Director::getInstance() ->getVisibleOrigin();

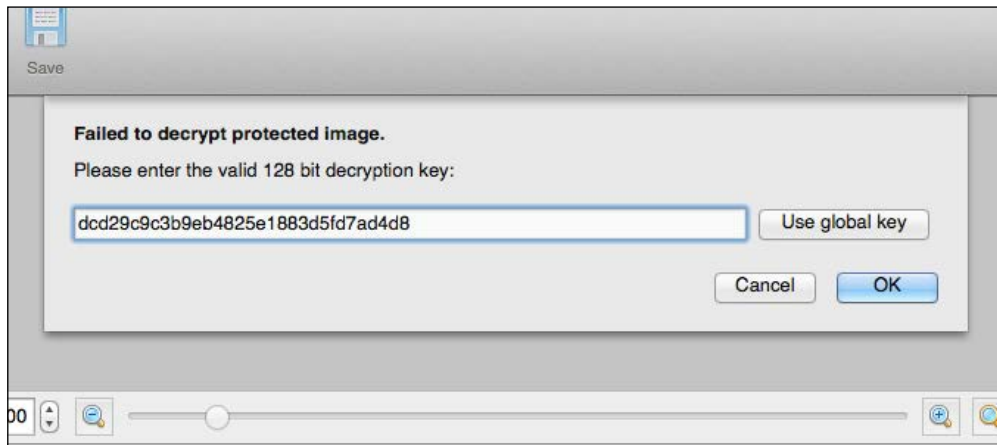
SpriteFrameCache::getInstance() -
>addSpriteFramesWithFile("res/encrypted.plist");
auto sprite =
Sprite::createWithSpriteFrameName("run_01.png");
sprite->setPosition(Vec2(visibleSize/2)+origin);
this->addChild(sprite);
```

There's more...

The application has a lot of sprite sheets normally. You can use each encryption key per sprite sheet. But this might create some confusion. You need to use the same key in all the sprite sheets in your application. The first time, you need to click on the **Create new key** button to create the encryption key. Then, you need to click on the **Save as global key** button to save the encryption key as the global key. Next time, when you create a new encrypted sprite sheet, you can set this encryption key as a global key by clicking on the **Use global key** button.

Now, we will move on to understanding how to check the encrypted sprite sheets. The encrypted sprite sheet's extension is `.ccz`.

1. Double-click the encrypted file that has the `.ccz` extension.
2. Launch Texture Packer and you will see the window where you need to enter the decryption key, as shown in the following screenshot:



3. Enter the decryption key or click on the **Use global key** button. If you have saved the key as the global key, then click on the **OK** button.
4. If the key is the correct key, you will see the sprite sheet as shown in the preceding screenshot:

Using encrypted zip files

In a smartphone, the game frequently downloads a `zip` file from the server to update resources. These assets are generally the main targets for hackers. They can decode these assets to manipulate information in a game system. Hence, security for these assets is very important. In this case, `zip` is encrypted to protect against cheaters. In this recipe, you will learn how to unzip an encrypted `zip` file with a password.

Getting ready

Cocos2d-x has an unzip library. However, encryption/decryption is disabled in this library. That's why we have to enable the crypt option in `unzip.cpp`. This file's path is `cocos2d/external/unzip/unzip.cpp`. You will have to comment out line number 71 of `unzip.cpp` to enable the crypt option.

```

// #ifndef NOUNCRYPT
//     #define NOUNCRYPT
// #endif

```

When we tried to build in Cocos2d-x version 3.7, an error occurred in `unzip.h` in line 46, as shown in the following code:

```
#include "CCPlatformDefine.h"
```

You have to edit the following code to remove this error, as shown:

```
#include "platform/CCPlatformDefine.h"
```

How to do it...

First, include the `unzip.h` file to use the unzip library in `HelloWorld.cpp` as shown in the following code:

```
#include "external/unzip/unzip.h"
```

Next, let's try to unzip the encrypted zip file with the password. This can be done by adding the following code in `HelloWorld.cpp`:

```

#define BUFFER_SIZE 8192
#define MAX_FILENAME 512

bool HelloWorld::uncompress(const char* password)
{
    // Open the zip file
    std::string outFileName = FileUtils::getInstance()-
    >fullPathForFilename("encrypt.zip");
    unzFile zipfile = unzOpen(outFileName.c_str());
    int ret = unzOpenCurrentFilePassword(zipfile, password);
    if (ret!=UNZ_OK) {
        CCLOG("can not open zip file %s", outFileName.c_str());
        return false;
    }

    // Get info about the zip file
    unz_global_info global_info;

```

```
    if (unzGetGlobalInfo(zipfile, &global_info) != UNZ_OK) {
        CCLOG("can not read file global info of %s",
            outFileName.c_str());
        unzClose(zipfile);
        return false;
    }

    CCLOG("start uncompressing");

    // Loop to extract all files.
    uLong i;
    for (i = 0; i < global_info.number_entry; ++i) {
        // Get info about current file.
        unz_file_info fileInfo;
        char fileName[MAX_FILENAME];
        if (unzGetCurrentFileInfo(zipfile, &fileInfo, fileName,
            MAX_FILENAME, nullptr, 0, nullptr, 0) != UNZ_OK) {
            CCLOG("can not read file info");
            unzClose(zipfile);
            return false;
        }

        CCLOG("filename = %s", fileName);

        unzCloseCurrentFile(zipfile);

        // Goto next entry listed in the zip file.
        if ((i+1) < global_info.number_entry) {
            if (unzGoToNextFile(zipfile) != UNZ_OK) {
                CCLOG("can not read next file");
                unzClose(zipfile);
                return false;
            }
        }
    }

    CCLOG("end uncompressing");
    unzClose(zipfile);

    return true;
}
```

Finally, you can unzip the encrypted zip file to use this method by specifying the password. If the password is `cocos2d-x`, you can unzip with the following code:

```
this->uncompress("cocos2d-x");
```

How it works...

1. Open the encrypted zip file using the `unzOpen` function, as shown:

```
unzFile zipfile = unzOpen(outFileName.c_str());
```

2. After opening it with the `unzOpen` function, open it again using the `unzOpenCurrentFilePassword` function, as shown here:

```
int ret = unzOpenCurrentFilePassword(zipfile, password);
if (ret!=UNZ_OK) {
    CLOG("can not open zip file %s", outFileName.c_str());
    return false;
}
```

3. After that, you can continue in the same way that is used to unzip an unencrypted zip file.

Using encrypted SQLite files

We often use SQLite to save the user data or game data. SQLite is a powerful and useful database. However, there is a database file in your game's sand box. Cheaters will get it from your game and they will edit it to cheat. In this recipe, you will learn how to encrypt your SQLite and prevent cheaters from editing it.

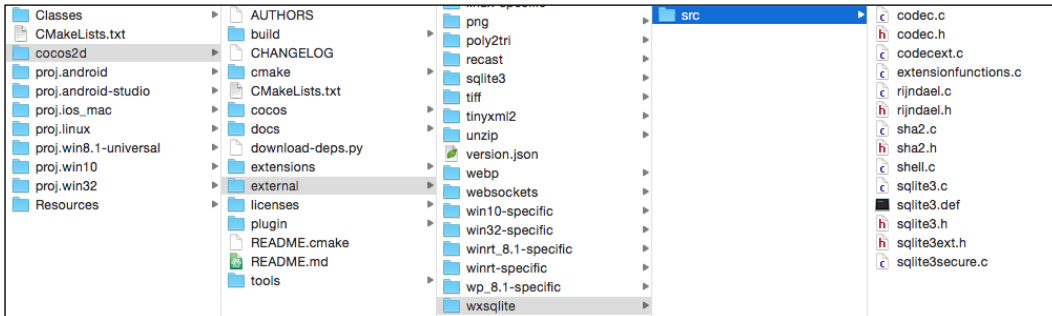
Getting ready

We will use the `wxSqlite` library to encrypt SQLite. This is free software. Firstly, you need to install `wxSqlite` in Cocos2d-x and edit some code and set files in Cocos2d-x.

1. Download the `wxSqlite3` project's zip file. Visit the following url: <http://sourceforge.net/projects/wxcode/files/Components/wxSQLite3/wxsqlite3-3.1.1.zip/download>
2. Expand the zip file.
3. Create a new folder called `wxsqlite` under `cocos2d/external`.

Taking Advantages

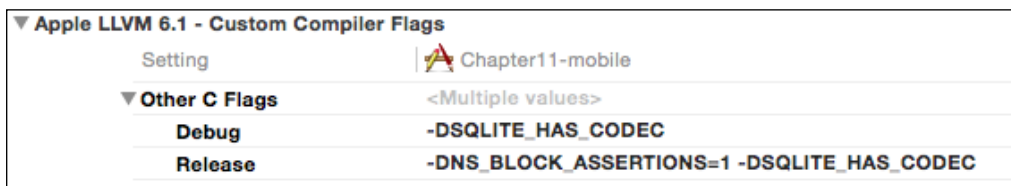
- Copy `sqlite3/secure/src` after expanding the folder to `cocos2d/external/wxsqlite` as shown in the following screenshot:



- Add `sqlite3.h` and `sqlite3secure.c` in `wxsqlite/src` that you added in step 4 to your project, as shown in the following screenshot:



- Add `-DSQLITE_HAS_CODEC` to Other C Flags in **Build Settings** of Xcode, as shown in the following screenshot:



- Create a new file called `Android.mk` in `cocos2d/external/wxsqlite`, as shown in the following code:

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := wxsqlite3_static
LOCAL_MODULE_FILENAME := libwxsqlite3
```

```

LOCAL_CFLAGS += -DSQLITE_HAS_CODEC
LOCAL_SRC_FILES := src/sqlite3secure.c
LOCAL_EXPORT_C_INCLUDES := $(LOCAL_PATH)/src
LOCAL_C_INCLUDES := $(LOCAL_PATH)/src
include $(BUILD_STATIC_LIBRARY)

```

8. Edit `Android.mk` in `cocos2d/cocos/storage/local-storage`, as shown in the following code:

```

LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)

LOCAL_MODULE := cocos_localstorage_static

LOCAL_MODULE_FILENAME := liblocalstorage

LOCAL_SRC_FILES := LocalStorage.cpp

LOCAL_EXPORT_C_INCLUDES := $(LOCAL_PATH)/..

LOCAL_C_INCLUDES := $(LOCAL_PATH)/../..

LOCAL_CFLAGS += -Wno-psabi
LOCAL_CFLAGS += -DSQLITE_HAS_CODEC
LOCAL_EXPORT_CFLAGS += -Wno-psabi

LOCAL_WHOLE_STATIC_LIBRARIES := cocos2dx_internal_static
LOCAL_WHOLE_STATIC_LIBRARIES += wxsqlite3_static

include $(BUILD_STATIC_LIBRARY)

$(call import-module,.)

```

9. Edit `LocalStorage.cpp` in `cocos2d/cocos/storage/local-storage`. Comment out line 33 and line 180, as shown in the following code.

```

LocalStorage.cpp line33:
// #if (CC_TARGET_PLATFORM != CC_PLATFORM_ANDROID)
LocalStorage.cpp line180:
// #endif // #if (CC_TARGET_PLATFORM != CC_PLATFORM_ANDROID)

```

10. Edit `Android.mk` in `proj.android/jni`, as shown in the following code:

```
LOCAL_SRC_FILES := hellocpp/main.cpp \
                  ../../Classes/AppDelegate.cpp \
                  ../../Classes/HelloWorldScene.cpp \
                  ../../cocos2d/external/wxsqlite/src/
sqlite3secure.c

LOCAL_C_INCLUDES := $(LOCAL_PATH)/../../Classes
LOCAL_C_INCLUDES += $(LOCAL_PATH)/../../cocos2d/external/wxsqlite/
src/
LOCAL_CFLAGS += -DSQLITE_HAS_CODEC
```

After this, SQLite is encrypted and can be used in your project.

How to do it...

1. You have to include `sqlite3.h` to use SQLite APIs.

```
#include "sqlite3.h"
```

2. Create the encrypted database, as shown in the following code:

```
std::string dbname = "data.db";
std::string path = FileUtils::getInstance()->getWritablePath() +
dbname;
CCLOG("%s", path.c_str());

sqlite3 *database = nullptr;
if ((sqlite3_open(path.c_str(), &database) != SQLITE_OK)) {
    sqlite3_close(database);
    CCLOG("open error");
} else {
    const char* key = "pass_phrase";
    sqlite3_key(database, key, (int)strlen(key));

    // sql: create table
    char create_sql[] = "CREATE TABLE sample ( "
        "            id      INTEGER PRIMARY KEY, "
        "            key     TEXT    NOT NULL, "
        "            value   INTEGER NOT NULL "
        "            ) "
        ";

    // create table
    sqlite3_exec(database, create_sql, 0, 0, NULL);
```

```

// insert data
char insert_sql[] = "INSERT INTO sample ( id, key, value )"
"          values (%d, '%s', '%d')          ";

char insert_record[3][256];
sprintf(insert_record[0],insert_sql,0,"test",300);
sprintf(insert_record[1],insert_sql,1,"hoge",100);
sprintf(insert_record[2],insert_sql,2,"foo",200);

for(int i = 0; i < 3; i++ ) {
    sqlite3_exec(database, insert_record[i], 0, 0, NULL);
}

sqlite3_reset(stmt);
sqlite3_finalize(stmt);
sqlite3_close(database);
}

```

3. Select the data from the encrypted database, as shown in the following code:

```

std::string dbname = "data.db";
std::string path = FileUtils::getInstance()->getWritablePath() +
dbname;
CCLOG("%s", path.c_str());

sqlite3 *database = nullptr;
if ((sqlite3_open(path.c_str(), &database) != SQLITE_OK)) {
    sqlite3_close(database);
    CCLOG("open error");
} else {
    const char* key = "pass_phrase";
    sqlite3_key(database, key, (int)strlen(key));

    // select data
    sqlite3_stmt *stmt = nullptr;

    std::string sql = "SELECT value FROM sample WHERE key='test'";
    if (sqlite3_prepare_v2(database, sql.c_str(), -1, &stmt, NULL)
== SQLITE_OK) {
        if (sqlite3_step(stmt) == SQLITE_ROW) {
            int value = sqlite3_column_int(stmt, 0);
            CCLOG("value = %d", value);
        } else {
            CCLOG("error , error=%s", sqlite3_errmsg(database));
        }
    }
}

```

```
    }  
  
    sqlite3_reset(stmt);  
    sqlite3_finalize(stmt);  
    sqlite3_close(database);  
}
```

How it works...

Firstly, you have to create the encrypted database with the `pass` phrase. To create it, follow these three steps:

1. Open the database normally.
2. Next, set the pass phrase using the `sqlite3_key` function.

```
const char* key = "pass_phrase";  
sqlite3_key(database, key, (int)strlen(key));
```

3. Finally, execute `sql` to create tables.

After this, you will need the encrypted database file in the application. You can get it from the path that was printed by `CCLOG`.

To select data from there, the same method is used. You can get data from the encrypted database using the same pass phrase after opening the database.

There's more...

You must be wondering whether this database was really encrypted. So let's check it. Open the database using the command line and executing the command as shown:

```
$ sqlite3 data.db  
SQLite version 3.8.4.3 2014-04-03 16:53:12  
Enter ".help" for usage hints.  
sqlite> .schema  
Error: file is encrypted or is not a database  
sqlite>
```

If the database is encrypted, you will not be able to open it and an error message will pop up, as shown:

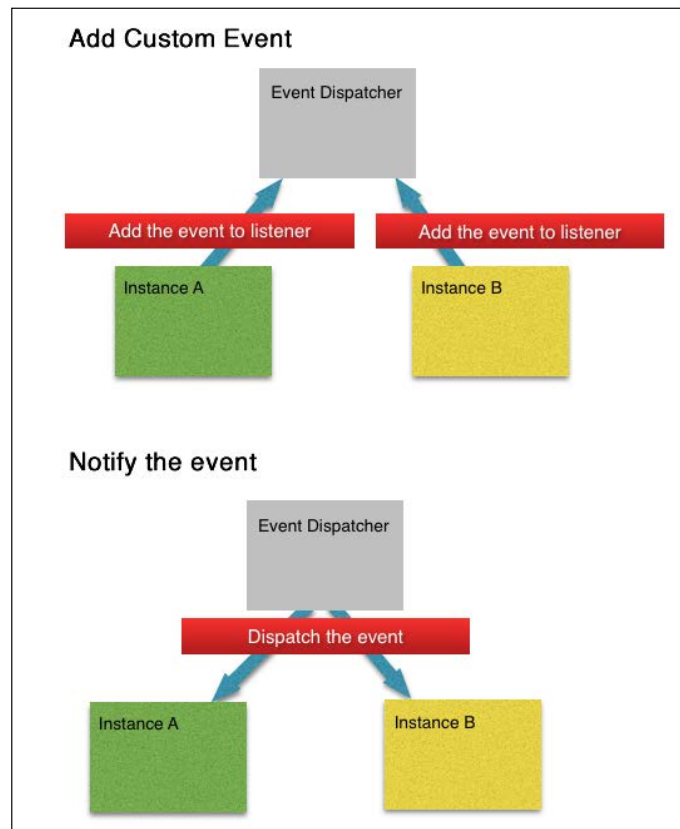
```
"file is encrypted or is not a database".
```

Creating Observer Pattern

Event Dispatcher is a mechanism for responding to events such as touching screen, keyboard events and custom events. You can get an event using Event Dispatcher. In addition, you can create `Observer Pattern` in the design patterns using it. In this recipe, you will learn how to use Event Dispatcher and how to create Observer Pattern in Cocos2d-x.

Getting ready

Firstly, we will go through the details of Observer Pattern. Observer Pattern is a design pattern. When an event occurs, Observer notifies the event about the subjects that are registered in Observer. It is mainly used to implement distributed event handling. Observer Pattern is also a key part in the MVC architecture.



How to do it...

We will create a count up label per second in this recipe. When touching a screen, count up labels are created in this position, and then, count up per second using Observer Pattern.

1. Create Count class that is extended Label class as shown in the following code:

```
Count.h
class Count : public cocos2d::Label
{
private:
    int _count;
    void countUp(float dt);
public:
    ~Count();
    virtual bool init();
    CREATE_FUNC(Count);
};
Count.cpp
Count::~Count()
{
    this->getEventDispatcher()-
>removeCustomEventListeners("TimeCount");
}

bool Count::init()
{
    if (!Label::init()) {
        return false;
    }

    _count = 0;

    this->setString("0");
    this->setFontSize(2.0f);

    this->getEventDispatcher()-
>addCustomEventListener("TimeCount", [=](EventCustom*
event) {
    this->countUp(0);
});

    return true;
}
```

```

void Count::countUp(float dt)
{
    _count++;
    this->setString(StringUtils::format("%d", _count));
}

```

2. Next, when touching a screen, this label will be created at the touching position and will call the `HelloWorld::countUp` method per second using a scheduler as the following code in `HelloWorld.cpp`:

```

bool HelloWorld::init()
{
    if ( !Layer::init() )
    {
        return false;
    }

    auto listener = EventListenerTouchOneByOne::create();
    listener->setSwallowTouches(_swallowsTouches);
    listener->onTouchBegan =
    C_CALLBACK_2(HelloWorld::onTouchBegan, this);
    this->getEventDispatcher()-
    >addEventListenerWithSceneGraphPriority(listener,
    this);

    this->schedule(schedule_selector(HelloWorld::countUp),
    1.0f);

    return true;
}

bool HelloWorld::onTouchBegan(cocos2d::Touch *touch,
cocos2d::Event *unused_event)
{
    auto countLabel = Count::create();
    this->addChild(countLabel);
    countLabel->setPosition(touch->getLocation());

    return true;
}

void HelloWorld::countUp(float dt)
{
    this->getEventDispatcher()-
    >dispatchCustomEvent("TimeCount");
}

```


3. After building and running this project, when you touch the screen, it will create a count up label at the touching position, and then you will see that the labels are counting up per second at the same time.

How it works...

1. Add the custom event called `TimeCount`. If `TimeCount` event occurred, then the `Count::countUp` method is called.

```
this->getEventDispatcher() -
>addCustomEventListener("TimeCount", [=](EventCustom*
event) {
    this->countUp(0);
});
```

2. Don't forget that you need to remove the custom event from `EventDispatcher` when the instance of the `Count` class is removed. If you forget to do that, then the zombie instance will be called from `EventDispatcher` when the event occurs and your game will crash.

```
this->getEventDispatcher() -
>removeCustomEventListeners("TimeCount");
```

3. In `HelloWorld.cpp`, call the `HelloWorld::countUp` method using the scheduler. The `HelloWorld::countUp` method calls the custom event called `TimeOut`.

```
this->getEventDispatcher() -
>dispatchCustomEvent("TimeCount");
```

And then, `EventDispatcher` will notify this event to the listed subjects. In this case, the `Count::countUp` method is called.

```
void Count::countUp(float dt)
{
    _count++;
    this->setString(StringUtils::format("%d", _count));
}
```

There's more...

Using `EventDispatcher`, labels count up at the same time. If you use `Scheduler` instead of `EventDispatcher`, you will notice something different.

Change the `Count::init` method as shown in the following code:

```
bool Count::init()
{
    if (!Label::init()) {
```

```

        return false;
    }

    _count = 0;

    this->setString("0");
    this->setFontScale(2.0f);
    this->schedule(schedule_selector(Count::countUp), 1.0f);

    return true;
}

```

In this code, use a scheduler by calling the `Count::countUp` method per second. You can see that the labels are not counting up at the same time in this way. Each label is counting up per second, however not at the same time. Using Observer Pattern, a lot of subjects can be called at the same time.

Networking with HTTP

In recent smartphone games, we normally use an Internet network to update data, download resources, and so on. There aren't any games developed without networking. In this recipe, you will learn how to use networking to download resources.

Getting ready

You have to include the header file of `network/HttpClient` to use networking.

```
#include "network/HttpClient.h"
```

If you run it on Android devices, you need to edit `proj.android/AndroidManifest.xml`.

```
<user-permission android:name="android.permission.INTERNET" />
```

How to do it...

In the following code, we will get the response from `http://google.com/` and then, print the response data as a log.

```

auto request = new network::HttpRequest();
request->setUrl("http://google.com/ ");
request->setRequestType(network::HttpRequest::Type::GET);
request->setResponseCallback([](network::HttpClient* sender,
network::HttpResponse* response) {
    if (!response->isSucceed()) {
        CLOG("error");
    }
});

```

```

        return;
    }

    std::vector<char>* buffer = response->getResponseData();
    for (unsigned int i = 0; i <buffer-> size (); i ++) {
        printf("%c", (* buffer)[i]);
    }
    printf("\n");
});

network::HttpClient::getInstance()->send(request);
request->release();

```

How it works...

1. Firstly, create an `HttpRequest` instance. The `HttpRequest` class does not have a `create` method. That's why you use `new` for creating the instance.

```
auto request = new network::HttpRequest();
```

2. Specify URL and the request type. In this case, set `http://google.com/` as a request URL and set `GET` as a request type.

```
request->setUrl("http://google.com/ ");
request->setRequestType(network::HttpRequest::Type::GET);
```

3. Set callback function to receive the data from the server. You can check its success using the `HttpResponse::isSucceed` method. And then you can get the response data using the `HttpResponse::getResponseData` method.

```
request->setResponseCallback([](network::HttpClient*
sender, network::HttpResponse* response){
    if (!response->isSucceed()) {
        CCLOG("error");
        return;
    }

    std::vector<char>* buffer = response-
>getResponseData();
    for (unsigned int i = 0; i <buffer-> size (); i ++) {
        printf("%c", (* buffer)[i]);
    }
    printf("\n");
});
```

4. You can request networking by calling the `HttpClient::send` method specifying the instance of the `HttpRequest` class. If you are getting a response via the network, then call the callback function as mentioned in Step3.


```
network::HttpClient::getInstance()->send(request);
```
5. Finally, you have to release the instance of `HttpRequest`. That's why you created it by using `new`.


```
request->release();
```

There's more...

In this section, you will learn how you can get resources from the network using the `HttpRequest` class. In the following code, get the Google log from the network and display it.

```
auto request = new network::HttpRequest();
request-
>setUrl("https://www.google.co.jp/images/branding/googlelogo/2x/
googlelogo_color_272x92dp.png");
request->setRequestType(network::HttpRequest::Type::GET);
request->setResponseCallback([&](network::HttpClient* sender,
network::HttpResponse* response){
    if (!response->isSucceed()) {
        CLOG("error");
        return;
    }

    std::vector<char>* buffer = response->getResponseData();
    std::string path = FileUtils::getInstance()->getWritablePath()
+ "image.png";
    FILE* fp = fopen(path.c_str(), "wb");
    fwrite(buffer->data(), 1, buffer->size(), fp);
    fclose(fp);

    auto size = Director::getInstance()->getWinSize();
    auto sprite = Sprite::create(path);
    sprite->setPosition(size/2);
    this->addChild(sprite);
});

network::HttpClient::getInstance()->send(request);
request->release();
```

You can see the following window after building and running this code.



You have to save the original data in the sandbox. You can get the path of the sandbox using the `FileUtils::getWritablePath` method.



Index

Symbols

3D modals

using 55, 56

.xml files

using 145, 146

A

acceleration sensor

used, for modifying gravity 174, 175

using 157, 158

actions

controlling 40

creating 37

easing 46

functions, calling with 44-46

repeating 41

reversing 41

sequencing 40

spawning 40

anchor points

setting 29, 30

Android environment

setting up 2-5

Android NDK

installing 4

URL 2

animations

creating 34-36

Apache ANT

installing 5

URL, for downloading 2

Arcade, in dafont

reference link 202

assets 139

AssetsManagerExtension 139

AudioEngine

used, for playing background music 125-127

used, for playing sound effect 125-127

B

background music

pausing 123, 124

playing 119, 120

playing, AudioEngine used 125-127

resuming 123, 124

balance

controlling 122, 123

batch node

using 53-55

Bezier curve

drawing 63

bitmap font labels

creating 71, 72

buttons

creating 101-103

C

C++

building, in NDK 19

checkboxes

creating 103-106

circles

drawing 61

Cocos2d-x

about 1

installing 5-7

URL 5

cocos command

- compile command 11
- deploy command 11
- using 8-11

cocos run command, parameters

- ios-bundleid 11
- portrait 11

collisions

- detecting 57, 58, 166-168

D

DelayTime action 42, 43

dot

- drawing 60

dpi

- about 159
- obtaining 159

DrawNode 64

drop shadow effect

- Label, creating with 73

E

easing 46

easing types

- EaseBackIn 47
- EaseBounceIn 47
- EaseElasticIn 47
- EaseExponentialIn 47
- EaseIn 47
- EaseInOut 47
- EaseOut 47
- EaseSinIn 47

Eclipse

- project, building by 13-19

Eclipse ADT, with Android SDK

- URL 2

effects

- scenes, transitioning with 83-85

encrypted sprite sheets

- using 207-210

encrypted SQLite files

- using 213-218

encrypted zip files

- using 210-213

F

functions

- calling, with actions 44-46

G

glow effect

- Label, creating with 74

Glyph Designer

- URL 202
- using 202-205

gravity

- modifying, acceleration sensor used 174, 175

H

HTTP

- networking with 223-225

I

installation, Cocos2d-x 5-7

J

Java

- installing 5

JNI (Java Native Interface) 151

joints

- PhysicsJointDistance 173
- PhysicsJointFixed 173
- PhysicsJointGear 173
- PhysicsJointLimit 173
- PhysicsJointMotor 173
- PhysicsJointPin 173
- PhysicsJointRatchet 173
- PhysicsJointRotaryLimit 173
- PhysicsJointRotarySpring 173
- PhysicsJointSpring 173
- using 168-174

json files

- using 149, 150

L

labels

- bitmap font labels, creating 71, 72
- creating, TTFConfig used 70

- system font labels, creating 65, 66
- true type font labels, creating 69, 70

layers

- creating 93, 94

line break 66

lines

- drawing 60

list views

- creating 116-118

loading bars

- creating 106-108

M

makeSprite 168

max texture size

- obtaining 160, 161

menus

- creating 98-101

modal layer

- creating 94-96

movies

- playing 127-129

multi resolution support

- implementing 19-21

N

native code

- using 151-155

NDK

- C++, building in 19

networking

- with HTTP 223-225

Node class

- properties 91

O

object property

- obtaining, in tiled map 191-194

observer pattern

- creating 219-223

original game

- preparing 21-24

original transitions

- making, for popping scenes 91-93
- making, for replacing scenes 85-91

outline effect

- Label, creating with 74

P

page views

- creating 115, 116

physics 163

PhysicsContact

- events 168

Physics Editor

- URL 195

- using 195-201

physics engine

- using 163-166

pitch

- controlling 122, 123

platform

- processing, modifying by 156, 157

plist files

- using 147, 148

polygon

- drawing 63

position

- obtaining, of sprite 29

processing

- modifying, by platform 156, 157

project

- building, by Eclipse 13-19

- building, by Xcode 11

R

RapidJSON 149

rectangles

- drawing 62

Repeat action 43

RepeatForever action 43

resolution policy

- EXACT_FIT 21

- FIXED_HEIGHT 21

- FIXED_WIDTH 21

- NO_BORDER 21

- SHOW_ALL 21

resource files

- managing 133-140

- selecting, for usage 131, 132

rich text

creating 73-75

S**scenes**

about 77
creating 78-82
transitioning between 82
transitioning, with effects 83-85

screen

keeping on 158, 159

scroll views

creating 113-115

Sequence action 42**shadow color**

modifying 75

shape

drawing 59

SimpleAudioEngine 120**size**

obtaining, of sprite 29

sliders

creating 108-110

sound effects

pausing 124, 125
playing 121
playing, AudioEngine used 125-127
resuming 124, 125

Spawn action 42**sprite rectangle**

obtaining 30

sprites

about 25
blinking 38
code tint 39
creating 26, 27
creating, static coordinate used 28
fading 38
manipulating 30
moving 37
position, obtaining 28
preparing, for jump 38
rotating 38
scaling 37
size, obtaining 28
skewing 39

sprites, properties

Color 32
Opacity 33
Rotate 31
Scale 31
Skew 32
Visibility 33, 34

SQLite

URL 141
using 141-145

static coordinate

used, for setting sprites 28

string

updating, after label creation 68, 69

system font labels

creating 65, 66

T**text alignment**

specifying 67

text fields

creating 111, 112

texture atlas

using 48-53

Texture Packer

URL 178
using 177-182
using, on command 182

tiled map

object property, obtaining in 192-194

Tiled Map Editor

URL 183
using 183-191

Transition Class

TransitionCrossFade 84
TransitionFade 84
TransitionFadeTR 84
TransitionFadeUp 84
TransitionFlipAngular 84
TransitionFlipX 84
TransitionJumpZoom 83
TransitionMoveInL 84
TransitionPageTurn 84
TransitionProgressRadialCW 84
TransitionRotoZoom 83
TransitionShrinkGrow 84

- TransitionSlideInL 84
- TransitionSplitCols 84
- TransitionSplitRows 84
- TransitionTurnOffTiles 84
- TransitionZoomFlipAngular 84
- TransitionZoomFlipX 84

triangle

- drawing 62

true type font labels

- creating 69, 70

TTFConfig

- used, for creating labels 70

V

volume

- controlling 122, 123

W

wxSqlite3 project

- URL 213

X

Xcode

- project, building by 11, 12



Thank you for buying Cocos2d-x Cookbook

About Packt Publishing

Packt, pronounced 'packed', published its first book, *Mastering phpMyAdmin for Effective MySQL Management*, in April 2004, and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern yet unique publishing company that focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website at www.packtpub.com.

About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt open source brand, home to books published on software built around open source licenses, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's open source Royalty Scheme, by which Packt gives a royalty to each open source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, then please contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

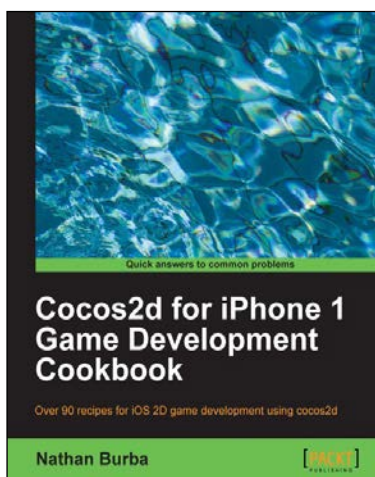


Creating Games with cocos2d for iPhone 2

ISBN: 978-1-84951-900-7 Paperback: 388 pages

Master cocos2d through building nine complete games for the iPhone

1. Games are explained in detail, from the design decisions to the code itself.
2. Learn to build a wide variety of game types, from a memory tile game to an endless runner.
3. Use different design approaches to help you explore the cocos2d framework.



Cocos2d for iPhone 1 Game Development Cookbook

ISBN: 978-1-84951-400-2 Paperback: 446 pages

Over 90 recipes for iOS 2D game development using cocos2d

1. Discover advanced Cocos2d, OpenGL ES, and iOS techniques spanning all areas of the game development process.
2. Learn how to create top-down isometric games, side-scrolling platformers, and games with realistic lighting.
3. Full of fun and engaging recipes with modular libraries that can be plugged into your project.

Please check www.PacktPub.com for information on our titles



Cocos2d for iPhone 0.99 Beginner's Guide

ISBN: 978-1-84951-316-6 Paperback: 368 pages

Make mind-blowing 2D games for iPhone with this fast, flexible, and easy-to-use framework!

1. A cool guide to learning cocos2d with iPhone to get you into the iPhone game industry quickly.
2. Learn all the aspects of cocos2d while building three different games.
3. Add a lot of trendy features such as particles and tilemaps to your games to captivate your players.



Learning Cocos2d-x Game Development

ISBN: 978-1-78398-826-6 Paperback: 266 pages

Learn cross-platform game development with Cocos2d-x

1. Create a Windows Store account and upload your game for distribution.
2. Develop a game using Cocos2d-x by going through each stage of game development process step by step.

Please check www.PacktPub.com for information on our titles

