# 第7章 Cocos2D世界的声音

游戏最大的乐趣就是具有极强的交互性。所以，熟练掌握Cocos2D的事件处理非常重要。

前面的游戏示例中已经简单使用了游戏音效和背景音乐处理，但没有系统地介绍Cocos2D处理声音的方法。因此本章将学习如何让Cocos2D的世界充满各种音乐或音效。很多时候，仅仅是游戏场景中令人心潮澎湃的音乐，就可以把玩家深深地吸引。

学习完本章将会了解：什么是CocosDenshion音效引擎、如何使用SimpleAudioEngine和CDSoundEngine播放背景音乐和音效、如何暂停游戏中的所有音效和背景音乐，如何在游戏中使用音效设置界面来开启和关闭声音选项。最后介绍如何保存游戏分数，以及更多的游戏优化。本章是垂直射击游戏示例项目的最后一章，所以会在章节的最后完善此游戏。

## 认识CocosDenshion音效引擎

苹果公司在iOS系统中提供两个框架用于播放音乐，分别是AVAudioPlayer和OpenAL。使用AVAudioPlayer播放音乐非常简单易行，但对音乐的控制方面很有限。OpenAL是一个低级API，支持更多特性，但使用起来要困难得多。

Cocos2D提供的CocosDenshion音效引擎，可以让开发者很方便地混合使用以上两种框架。本节将对Cocos2D的CocosDenshion音效引擎进行简单介绍。

### CocosDenshion重要API

CocosDenshion引擎由Steve Oldmeadow开发，是一套用于声音控制的类库，支持iOS和Mac OS X系统，从Cocos2D早期版本开始已内置该引擎。CocosDenshion名字的来历也挺有意思，前面Cocos是Cocos2D的简写，Denshion是一个日文单词的音译，意思是“电子音乐”。

CocosDenshion内置一个低延时音效引擎，用来播放游戏中的音效，同时可以修改音调、音高等。此外CocosDenshion提供了一个音效管理器，可以播放多声道音乐，同时负责iOS的声音进程管理。

CocosDenshion音效引擎提供多个API。

##### CDSoundEngine

在最底层，提供低延时的音效播放引擎，以及一些面向对象的API。

##### CDAudioManager

基于CDSoundEngine构建，添加多声道音乐播放功能，支持一些压缩格式的长时音效或音乐，如MP3、IMA4或AAC。

此外，其声音进程管理功能，可以提供一些更丰富的性能，比如让用户听自己的音乐，或者当设备的静音开关打开时确保音效播放被暂停。

##### SimpleAudioEngine

提供一个简化的声音控制API，可以极大地简化游戏中的声音控制。这是本章要重点学习的内容。

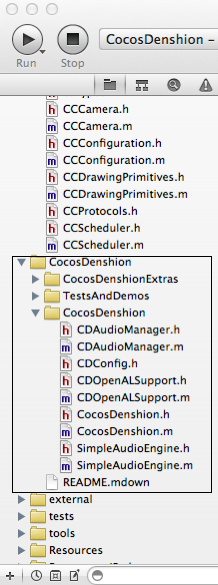
当然，如果要利用一些更丰富的性能，也还是要用到CDAudioManager和CDSoundEngine的API。

使用CocosDenshion，可以充分发挥AVAudioPlayer和OpenAL的强大威力，同时不用考虑如何使用低级别的API。

注意 虽然CocosDenshion引擎包含在Cocos2D引擎中，但可以用在非Cocos2D引擎所开发的一般UIKit项目中，只是部分功能仍然需要Cocos2D框架，如音效逐渐消失等。此外，CocosDenshion引擎和PASoundMgr引擎不能同时使用。

### CocosDenshion相关类

和其他特性一样，CocosDenshion源代码也包含在Cocos2D中，开发者可以根据自己的需要去研究。在Xcode中打开Cocos2D官方模板cocos2d-ios.xcodeproj，从左侧面板中找到CocosDenshion，单击其左侧的下三角，展开后可以看到和CocosDenshion相关的类文件，如图7-1所示。



1. 在Xcode中查看CocosDenshion的相关类

大家可能已经习惯看到Cocos2D的类名前面都加上CC。但CocosDenshion中的类名则会在前面加上CD前缀（SimpleAudioEngine除外）。

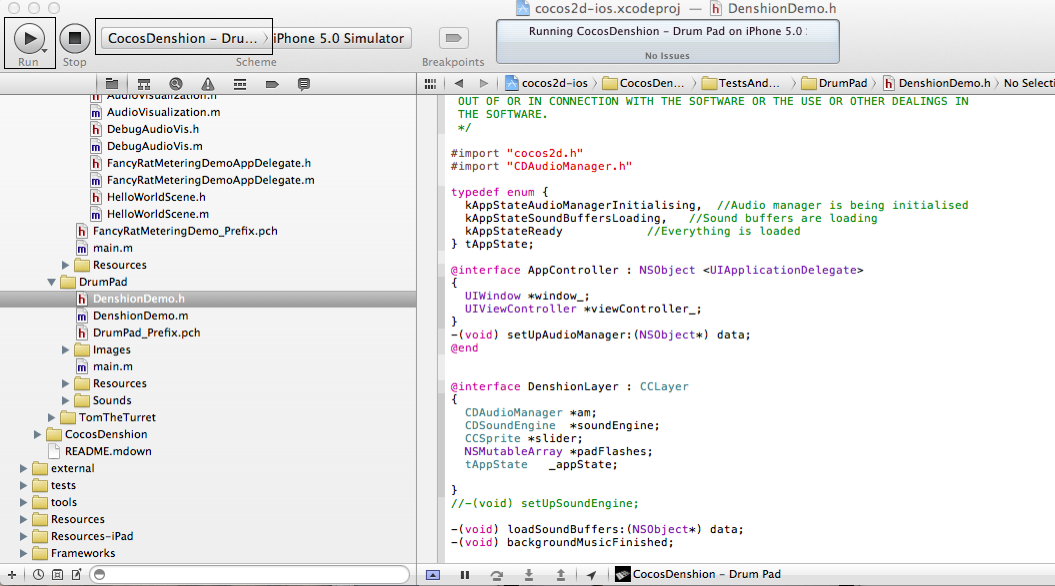
限于篇幅，这里不对相关类的定义、方法及属性进行详细说明，感兴趣的读者可以打开相关类的.h和.m文件，更深入地研究。

Cocos2D官方模板中提供4个相关的示例项目，分别是：

* Fade To Grey
* FancyRat Metering
* DrumPad
* Tom The Turret

其中最后一个项目Tom The Turret存在bug，而前三个示例项目将有助于大家更好地学习CocosDenshion。

在一步学习之前，可以在Xcode中打开Cocos2D官方模板cocos2d-ios.xcodeproj，在上部工具栏的Scheme中选择测试项目名称，如CocosDenshion-Drum Pad，单击Run按钮编译运行，如图7-2所示。



1. 在Scheme中选择CocosDenshion-Drum Pad

Cocos2D官方模板中提供了以上几个项目的详细源代码，感兴趣的读者可以仔细研究。

### Cocos2D支持的CocosDenshion

Cocos2D支持3种方式使用CocosDenshion，根据开发者所选择的方式，需要在项目中添加不同的文件和框架。下面对这3种方式分别进行介绍。

#### SimpleAudioEngine引擎

SimpleAudioEngine是最简单同时也是最常用的API，可以实现一些最基本的声音控制操作。如果开发者只想在游戏中播放音效和背景音乐，而不需要进行一些更低级的操作，那么无疑它是最佳选择。

使用SimpleAudioEngine的必备条件：

* 系统为iOS 3.0或以上版本，如果是Mac系统，应是Mac OS X 10.6或以上版本。
* 在项目中添加的文件包括SimpleAudioEngine.\*、CocosDenshion.\*、CDAudioManager.\*、CDOpenALSupport.h和CDConfig.h等。
* 需要在项目中添加的框架包括OpenAL、Audio Toolbox、AVFoundation（如果在Mac OS X中则应包含CDXMacOSXSupport.\*文件）。

#### CDAudioManager引擎

其实SimpleAudioEngine中已经用到CDAudioManager。使用该引擎可以播放多声道的长时音效或音乐，同时支持声音进程管理。CDAudioManager和SimpleAudioEngine的区别在于，CDAudioManager中的声音使用数字id来标识，而SimpleAudioEngine中的声音则使用文件名来标识。同时，CDAudioManager可以更灵活的同时播放多个声音。

使用CDAudioManager的必备条件：

* 支持的系统为iOS 3.0及以上版本，如果是Mac系统，则应是Mac OS X 10.6或以上版本。
* 在项目中需要添加的文件包括CDAudioManager.\*、CocosDenshion.\*、CDOpenALSupport.h和CDConfig.h。
* 在项目中所需添加的框架包括OpenAL、Audio Toolbox、AVFoundation（如果在Mac OS X中则应包含CDXMacOSXSupport.\*文件）。

#### CDSoundEngine引擎

CDSoundEngine声音引擎在OpenAL基础上构建，可以同时播放32个声音，还可以控制音调、音高和音量。如果需要支持iOS 2.1.x设备，可以考虑使用CDSoundEngine。此外，如果想要在项目中直接使用CDSoundEngine，需要根据iPhone Audio Programming Guide中的指导进行声音进程的中断管理。

使用CDSoundEngine的必备条件：

* 支持的系统为iOS 2.1.x及以上版本，如果是Mac系统，应是Mac OS X 10.6或以上版本。
* 项目中需要添加的文件包括CocosDenshion.\*、CDOpenALSupport.h和CDConfig.h。
* 项目中所需添加的框架包括OpenAL、Audio Toolbox。

## SimpleAudioEngine使用方法

SimpleAudioEngine由Joao Caxaria开发，作为CocosDenshion的接口使用。

注意 对于开发者来说，只需知道：使用SimpleAudioEngine是在Cocos2D中添加音乐的最简单方式。

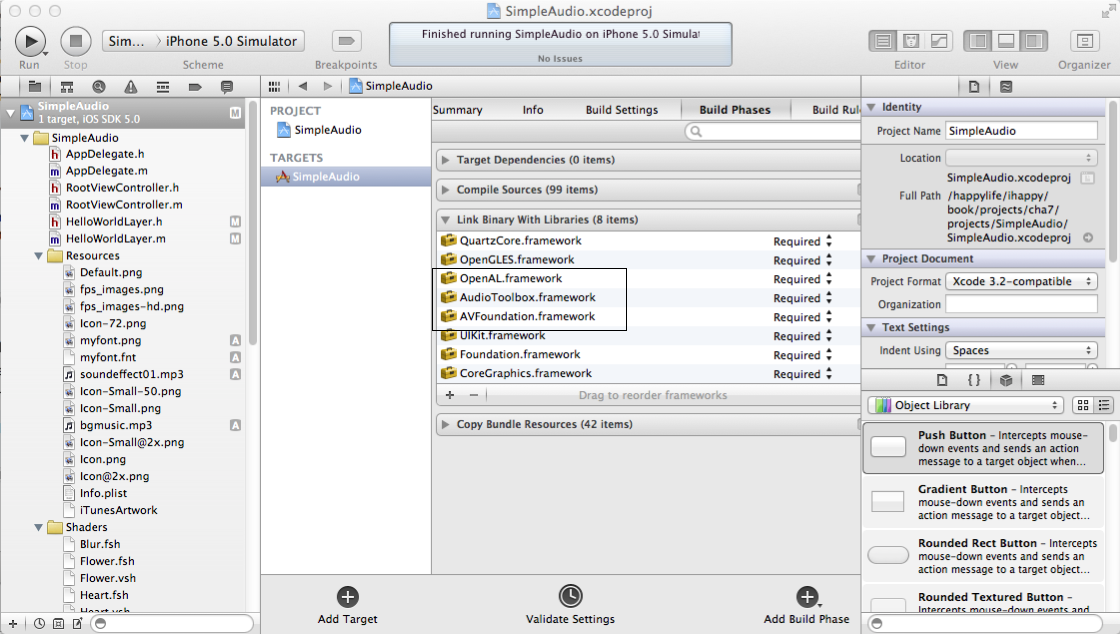
1. 添加背景音乐

为了说明如何使用SimpleAudioEngine，首先使用Cocos2D模板创建一个新项目，名为CocosAudio。单击File→New Project，选择iOS→Cocos2D→Cocos2D模板，然后单击Next，在Product Name处输入SimpleAudio，单击Next，选择存放的目录后，单击Create。

使用SimpleAudioEngine来播放背景音乐非常简单。通常需要使用以下4个步骤完成：

1. 在项目中添加OpenAL、AudioToolbox和AVFoundation框架。

具体操作：在Xcode中单击项目名称，单击TARGETS下面的CocosAudio，然后在右边单击Link Binary With Libraries的下三角。如果其中没有上述3个框架，则单击“+”，从弹出的列表中选择相应的框架添加进去，如图7-3所示。



1. 添加和声音相关的系统框架
2. 加载资源素材。

将事先准备好的声音和字体资源素材（chapter7/resource/audio/SimpleAudio）拖到Xcode的Resources中，记得选择“Copy items into destination group’s folder(if needed)”。

1. 导入SimpleAudioEngine的类文件。

在Xcode中切换到HelloWorldLayer.h，添加以下代码以导入SimpleAudioEngine的类文件：

#import "SimpleAudioEngine.h"

1. 修改init方法。

在Xcode中切换到HelloWorldLayer.m，并在init方法中修改代码如代码清单7-1所示。

1. 在init方法中修改代码

-(id) init

{

// always call "super" init

// Apple recommends to re-assign "self" with the "super" return value

if( (self=[super init])) {

// create and initialize a Label

CCLabelTTF \*label = [CCLabelTTF labelWithString:@"Cocos Audio World" fontName:@"Marker Felt" fontSize:64];

// ask director the the window size

CGSize size = [[CCDirector sharedDirector] winSize];

// position the label on the center of the screen

label.position = ccp( size.width /2 , size.height/2 );

// add the label as a child to this Layer

[self addChild: label];

//添加一行代码以播放背景音乐：

[[SimpleAudioEngine sharedEngine]playBackgroundMusic:@"bgmusic.mp3" loop:YES];

}

return self;

}

此时编译运行，就可以听到背景音乐了。

1. 控制音乐的播放和停止

如何手动控制音乐的播放和停止呢？这里要利用之前所学的文本渲染系统。

1. 添加控制标签。

首先在当前场景中添加两个控制标签，分别用于播放和停止音乐。修改init方法中的代码如代码清单7-2所示。

1. 修改init方法中的代码

-(id) init

{

// always call "super" init

// Apple recommends to re-assign "self" with the "super" return value

if( (self=[super init])) {

//使用CCDirector单例对象来获取屏幕大小

CGSize winSize = [[CCDirector sharedDirector] winSize];

//定义一个字体

NSString \*fontName = @"myfont.fnt";

//创建播放标签，当触碰播放标签时，会调用playSong方法

CCLabelBMFont \*playLabel = [CCLabelBMFont labelWithString:@"Play" fntFile:fontName];

CCMenuItemLabel \*playItem = [CCMenuItemLabel itemWithLabel:playLabel target:self selector:@selector(playSong:)];

playItem.scale = 0;

playItem.position = ccp(winSize.width/2,winSize.height \*0.7);

//创建停止标签，当触碰播放标签时，会调用stopPlaying方法

CCLabelBMFont \*stopLabel = [CCLabelBMFont labelWithString:@"Stop" fntFile:fontName];

CCMenuItemLabel \*stopItem = [CCMenuItemLabel itemWithLabel:stopLabel target:self selector:@selector(stopPlaying:)];

stopItem.scale = 0;

stopItem.position = ccp(winSize.width/2,winSize.height \*0.2);

//创建控制菜单，并将播放标签和停止标签添加进去

CCMenu \*menu =[CCMenu menuWithItems:playItem,stopItem, nil];

menu.position = CGPointZero;

[self addChild:menu];

//让播放标签执行一个动作，出现在屏幕中

[playItem runAction:

[CCSequence actions:

[CCDelayTime actionWithDuration:3.0],

[CCEaseIn actionWithAction:

[CCScaleTo actionWithDuration:2.0 scale:0.9]rate:4.0],

nil]];

//让停止标签执行一个动作，出现在屏幕中

[stopItem runAction:

[CCSequence actions:

[CCDelayTime actionWithDuration:3.0],

[CCEaseIn actionWithAction:

[CCScaleTo actionWithDuration:2.0 scale:0.8]rate:4.0],

nil]];

}

return self;

}

首先在屏幕中添加一个菜单，由两个菜单项组成，分别用于开始播放音乐和停止播放背景音乐。最后，为播放按钮和停止按钮各定义一个动作，从而让其逐渐出现在屏幕中。

1. 在init方法下添加playSong和stopPlaying两个方法的实现代码：

-(void)playSong:(id)sender{

//添加一行代码以播放背景音乐：

[[SimpleAudioEngine sharedEngine]playBackgroundMusic:@"bgmusic.mp3" loop:YES];

}

该方法直接使用SimpleAudioEngine的单例对象来播放背景音乐，并不断循环，如代码下所示：

-(void)stopPlaying:(id)sender{

//停止播放背景音乐

[[SimpleAudioEngine sharedEngine] stopBackgroundMusic];

}

1. 添加几个标签。

前面介绍了直接使用SimpleAudioEngine的单例对象停止播放背景音乐。为了更加理解如何使用SimpleAudioEngine控制音乐的播放，可以在项目中添加几个标签。

在Xcode中切换到HelloWorldLayer.m，更改init方法。修改后的init方法如代码清单7-3所示。

1. 修改后的init方法

// on "init" you need to initialize your instance

-(id) init

{

// always call "super" init

// Apple recommends to re-assign "self" with the "super" return value

if( (self=[super init])) {

//使用CCDirector单例对象来获取屏幕大小

CGSize winSize = [[CCDirector sharedDirector] winSize];

//定义一个字体

NSString \*fontName = @"SpaceGameFont.fnt";

//创建播放标签，当触碰播放标签时，会调用playSong方法

CCLabelBMFont \*playLabel = [CCLabelBMFont labelWithString:@"Play" fntFile:fontName];

CCMenuItemLabel \*playItem = [CCMenuItemLabel itemWithLabel:playLabel target:self selector:@selector(playSong:)];

playItem.scale = 0.5;

playItem.position = ccp(winSize.width/2,winSize.height \*0.5);

//创建停止标签，当触碰播放标签时，会调用stopPlaying方法

CCLabelBMFont \*stopLabel = [CCLabelBMFont labelWithString:@"Stop" fntFile:fontName];

CCMenuItemLabel \*stopItem = [CCMenuItemLabel itemWithLabel:stopLabel target:self selector:@selector(stopPlaying:)];

stopItem.scale = 0.5;

stopItem.position = ccp(winSize.width/2,winSize.height \*0.3);

//创建暂停标签，当触碰播放标签时，会调用pausePlaying方法

CCLabelBMFont \*pauseLabel = [CCLabelBMFont labelWithString:@"Pause" fntFile:fontName];

CCMenuItemLabel \*pauseItem = [CCMenuItemLabel itemWithLabel:pauseLabel target:self selector:@selector(pausePlaying:)];

pauseItem.scale = 0.5;

pauseItem.position = ccp(winSize.width/2,winSize.height \*0.1);

//创建继续标签，当触碰播放标签时，会调用resumePlaying方法

CCLabelBMFont \*resumeLabel = [CCLabelBMFont labelWithString:@"Resume" fntFile:fontName];

CCMenuItemLabel \*resumeItem = [CCMenuItemLabel itemWithLabel:resumeLabel target:self selector:@selector(resumePlaying:)];

resumeItem.scale = 0.5;

resumeItem.position = ccp(winSize.width/2,winSize.height \*0.85);

//创建重放标签，当触碰重放标签时，会调用rewindPlaying方法

CCLabelBMFont \*rewindLabel = [CCLabelBMFont labelWithString:@"Rewind" fntFile:fontName];

CCMenuItemLabel \*rewindItem = [CCMenuItemLabel itemWithLabel:rewindLabel target:self selector:@selector(rewindPlaying:)];

rewindItem.scale = 0.5;

rewindItem.position = ccp(winSize.width/2,winSize.height \*0.65);

//创建控制菜单，并将播放标签和停止标签添加进去

CCMenu \*menu =[CCMenu menuWithItems:playItem,stopItem,pauseItem, resumeItem,rewindItem, nil];

menu.position = CGPointZero;

[self addChild:menu];

}

return self;

}

1. 添加控制音乐暂停、继续和重放的方法。如代码清单7-4所示。
2. 添加控制音乐暂停、继续和重放的方法

-(void)pausePlaying:(id)sender{

//暂停播放背景音乐

[[SimpleAudioEngine sharedEngine] pauseBackgroundMusic];

}

-(void)resumePlaying:(id)sender{

//继续播放背景音乐

[[SimpleAudioEngine sharedEngine] resumeBackgroundMusic];

}

-(void)rewindPlaying:(id)sender{

//从开始处重放背景音乐

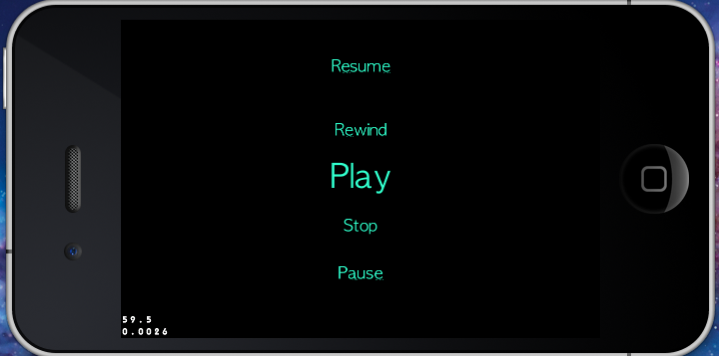
[[SimpleAudioEngine sharedEngine] rewindBackgroundMusic];

}

以上代码首先在屏幕中增加3个标签按钮，用于暂停、继续和重放音乐，并将其添加到菜单中，同时分别定义实现暂停、继续和播放音乐的3个方法。

1. 编译运行程序。

此时编译运行程序，一个简单的音乐播放器就完成了，如图7-4所示。



1. 一个简单的音乐播放器
2. 判断当前是否播放背景音乐

SimpleAudioEngine还提供了以下两个方法：

* preloadBackgroundMusic：预加载某个背景音乐，然后在需要时播放。
* isBackgroundMusicPlaying：判断当前是否在播放背景音乐。

isBackgroundMusicPlaying的具体使用方法如下所示。

1. 在init方法中添加菜单为当前层的子节点之后添加一行代码：

[self scheduleUpdate];

1. 在HelloWorldLayer.h的变量声明中添加以下代码：

@interface HelloWorldLayer : CCLayer

{

BOOL isPlaying;

CCLabelTTF \*label;

}

1. 在HelloWorldLayer.m中添加一个新的方法，如代码清单7-5所示。
2. 在HelloWorldLayer.m中添加一个新的方法

-(void)update:(ccTime)dt{

isPlaying = [[SimpleAudioEngine sharedEngine]isBackgroundMusicPlaying];

if(isPlaying){

if(label !=nil){

[label removeFromParentAndCleanup:YES];

}

label = [CCLabelTTF labelWithString:@"Now it's playing the music" fontName:@"Marker Felt" fontSize:32];

CGSize screenSize = [CCDirector sharedDirector].winSize;

label.position = ccp(screenSize.width/2,screenSize.height \*0.1);

[self addChild:label];

}

else{

if(label !=nil){

[label removeFromParentAndCleanup:YES];

}

label = [CCLabelTTF labelWithString:@"Touch Play to start" fontName:@"Marker Felt" fontSize:32];

CGSize screenSize = [CCDirector sharedDirector].winSize;

label.position = ccp(screenSize.width/2,screenSize.height \*0.1);

[self addChild:label];

}

}

以上代码定义一个BOOL变量，用来获取isBackgroundMusicPlaying信息。然后定义一个update更新方法，并加入判断语句。如果当前正在播放背景音乐，即isBackgroundMusicPlaying为真，首先清除已有标签，然后创建一个标签，显示“Now it’s playing the music”。如果不是，则首先清除已有标签，然后创建一个标签，显示“Touch Play to start ”。

此时编译运行程序，会看到一个小巧的音乐播放器，当触碰控制音乐播放的标签时，下方的文字内容也会发生变化，如图7-5所示。



1. 使用标签来控制音乐的播放
2. 添加音效

音效和背景音乐的区别在于，背景音乐会贯穿游戏某个关卡的始终，而音效则是玩家和游戏产生交互时产生的声音效果。

播放音效和播放背景音乐一样简单，回到刚才的项目，在屏幕中添加一个新的标签用于播放音效。

在Xcode中切换到HelloWorldLayer.m文件，对init方法进行修改，在创建控制菜单这行注释代码的上面添加几行代码，用于创建播放音效的标签，如代码清单7-6所示。

1. 创建播放音效的标签

//创建播放音效标签，当触碰播放音效标签时，会调用soundEffect方法

CCLabelBMFont \*soundEffectLabel = [CCLabelBMFont labelWithString:@"SoundEffect" fntFile:fontName];

CCMenuItemLabel \*soundEffectItem = [CCMenuItemLabel itemWithLabel:soundEffectLabel target:self selector:@selector(soundEffect:)];

soundEffectItem.scale = 0.3;

soundEffectItem.position = ccp(winSize.width\*0.2,winSize.height \*0.5);

//创建控制菜单，并将播放标签和停止标签添加进去

CCMenu \*menu =[CCMenu menuWithItems:playItem,stopItem,pauseItem, resumeItem,rewindItem, soundEffectItem, nil];

menu.position = CGPointZero;

[self addChild:menu];

上面的代码读者应该已经非常熟悉了。首先创建一个用于播放音效的标签，然后创建一个对应的菜单项，并将该菜单项的内容设置为刚才所创建的标签。接下来设置菜单项比例，以及在屏幕中的位置，最后将其添加到菜单中。

添加播放音效的方法如下：

-(void)soundEffect:(id)sender{

//播放音效

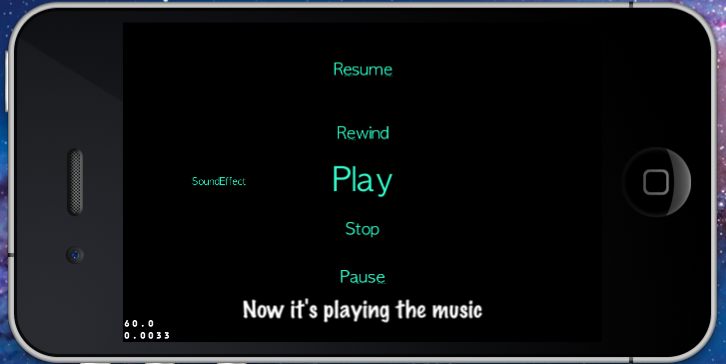
[[SimpleAudioEngine sharedEngine]playEffect:@"soundeffect001.mp3" pitch:12.0 pan:5.0 gain:2.0];

}

需要注意，以上几个参数中，pitch是音高，pan是OpenAL中的音源位置，gain是音量。具体的参数调整读者可以根据自己的需要来实验。

在上面这个方法中，使用短短一行代码充分展示了OpenAL的强大威力。

此时编译运行程序，当触碰soundEffect标签时，就会播放指定的音效。而且背景音乐的播放与否完全不会影响音效，如图7-6所示。



1. 用标签来控制音效的播放

对于音效播放来说，SimpleAudioEngine还提供了几个方法，例如：

* stopEffect：停止播放音效，
* preloadEffect：预加载音效。
* unloadEffect：和preloadEffect方法对应，用于从缓存中清除预加载的音效。

注意 游戏中使用的音效比较多时，可以使用preloadEffect方法来预加载所需要的全部音效，在需要的地方播放指定音效。

## CDSoundEngine使用方法

大多数情况下，SimpleAudioEngine可以满足游戏开发的基本需求，但有时需要更复杂的音效，此时就会用到CDSoundEngine。本节介绍CDSoundEngine的相关知识，以及如何在项目中使用。

1. 设置CDSoundEngine

CDSoundEngine允许同时播放多个声音（只要在iPhone硬件设备可以处理的范围内），可以将这些声音分类，实时修改音高、音源位置和音量等，还可以处理声音进程的中断。使用CDSoundEngine可以灵活地控制各个声道，比如在播放了几个音效后停止播放背景音乐。

和SimpleAudioEngine不同，要使用CDSoundEngine必须预先进行一些设置。

通常来说，开发者希望在游戏开始的界面播放声音，并载入这些声音，以便在后续使用。当然，考虑到内存占用问题，并不希望将多达30个声音文件加载到内存中。如果声音文件不太多，可以在AppDelegate文件中初始化CDSoundEngine，并加载所有的声音文件。当然，也可以在需要使用这些声音的场景中加载相关文件。接下来用一个示例来说明如何设置。

打开Xcode，创建一个新的项目，命名为CDSoundTest。

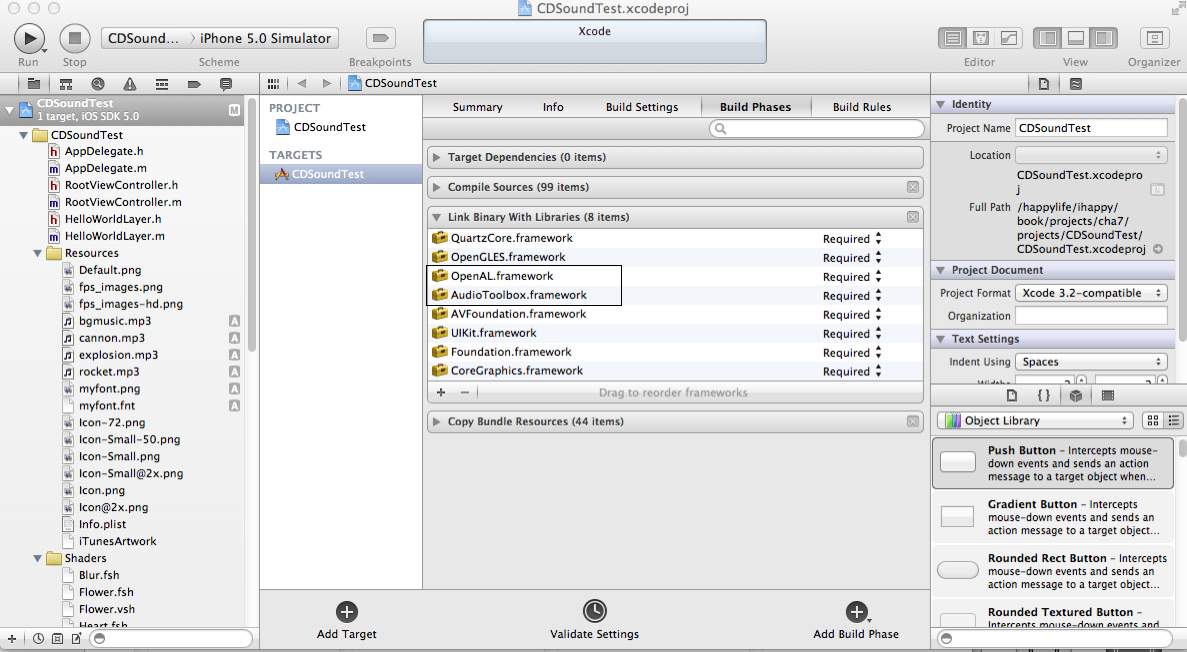
打开Xcode，单击File→New Project，选择iOS→Cocos2D→Cocos2D模板，然后单击Next，在Product Name处输入CDSoundTest，单击Next，选择存放的目录后，单击Create。把chapter7/resource/audio/CDSoundTest/目录下面的音效文件以及chapter7/resource/fonts/目录下面的myfont.fnt和myfont.png文件拖动到Resources中，确保“Copy items into destination group’s folder(if needed)”。

通过以下步骤设置CDSoundEngine。

1. 添加框架。

由于CDSoundEngine需要用到OpenAL和AudioToolbox这两个框架，所以确保项目中已经有这两个框架。

具体操作：在Xcode中单击项目名称，单击TARGETS下面的CDSoundTest，然后在右边单击Link Binary With Libraries的下三角。如果其中没有上述两个框架，则单击“+”，从弹出的列表中选择相应的框架添加进去，如图7-7所示。



1. 添加OpenAL和AudioToolbox 系统框架
2. 定义宏。

切换到AppDelegate.m文件，在文件的顶部添加代码如代码清单7-7所示。

1. AppDelegate.m文件的顶部添加代码

#import "CDAudioManager.h"

#import "CocosDenshion.h"

// 使用以下宏定义来简化播放音乐的代码

/\*\* CDAudioManager 支持左右两个声道

typedef enum {

kASC\_Left = 0,

kASC\_Right = 1

} tAudioSourceChannel; \*/

#define CGROUP\_BG kASC\_Left // 背景音乐所在的声道

#define CGROUP\_EFFECTS kASC\_Right // 音效所在的声道

#define SND\_BG\_LOOP 1 // 背景音乐的标识

#define SND\_ROCKET 2 // 火箭炮音效的标识

#define SND\_CANNON 3 //大炮音效的标识

#define SND\_BLAST 4 //爆炸音效的标识

// 使用下面的宏来播放音乐

#define playEffect(\_\_ID\_\_) [[CDAudioManager sharedManager].soundEngine playSound:\_\_ID\_\_ sourceGroupId:CGROUP\_EFFECTS pitch:1.0f pan:0.0f gain:1.0f loop:NO]

以上代码定义了几个宏。CDAudioManager支持两个声道，分别是左声道和右声道。这里将左声道定义为背景音乐的声道，把右声道定义为音效的声道。接下来为背景音乐和3个音效分别指定对应的ID。最后定义一个用于播放音效的宏，从而在程序中简化相关的代码。

1. 设置CDSoundEngine。

找到applicationDidFinishLaunchingWithOptions方法，其中有一行代码：

[director\_ pushScene: [IntroLayer scene]];

在以上这行代码前面添加代码，如代码清单7-8所示。

1. 设置CDSoundEngine

//SOUND INIT

CDSoundEngine \*sse = [CDAudioManager sharedManager].soundEngine;

/\*\*

A source group is another name for a channel

Here I have 2 channels, the first index allows for only a single effect... my background music

The second channel I have reserved for my sound effects. This is set to 31 because you can

have up to 32 effects at once

\*/

NSArray \*sourceGroups = [NSArray arrayWithObjects:[NSNumber numberWithInt:1], [NSNumber numberWithInt:31], nil];

[sse defineSourceGroups:sourceGroups];

//Initialise audio manager asynchronously as it can take a few seconds

/\*\* Different modes of the engine

typedef enum {

kAMM\_FxOnly, //!Other apps will be able to play audio

kAMM\_FxPlusMusic, //!Only this app will play audio

kAMM\_FxPlusMusicIfNoOtherAudio, //!If another app is playing audio at start up then allow it to continue and don't play music

kAMM\_MediaPlayback, //!This app takes over audio e.g music player app

kAMM\_PlayAndRecord //!App takes over audio and has input and output

} tAudioManagerMode;\*/

[CDAudioManager initAsynchronously:kAMM\_FxPlusMusicIfNoOtherAudio];

//Load sound buffers asynchrounously

NSMutableArray \*loadRequests = [[[NSMutableArray alloc] init] autorelease];

/\*\*

Here we set up an array of sounds to load

Each CDBufferLoadRequest takes an integer as an identifier (to call later)

and the file path. Pretty straightforward here.

\*/

[loadRequests addObject:[[[CDBufferLoadRequest alloc] init:SND\_BG\_LOOP filePath:@"bgmusic.mp3"] autorelease]];

[loadRequests addObject:[[[CDBufferLoadRequest alloc] init:SND\_ROCKET filePath:@"rocket.mp3"] autorelease]];

[loadRequests addObject:[[[CDBufferLoadRequest alloc] init:SND\_CANNON filePath:@"cannon.mp3"] autorelease]];

[loadRequests addObject:[[[CDBufferLoadRequest alloc] init:SND\_BLAST filePath:@"explosion.mp3"] autorelease]];

[sse loadBuffersAsynchronously:loadRequests];

以上代码就是对CDSoundEngine的设置，其作用如下：

1. 在项目中添加必需的框架之后，首先使用CDAudioManager的单例对象初始化了一个CDSoundEngine对象。
2. 定义一个音源组的数组，其中每个音源组都代表一个声道。

这里设置两个声道，该声道只允许播放一种音效，而这个声道将被用来播放背景音乐。同时，另一个声道则设置为允许播放多达31种音效，因为最多只支持32种音效。

1. 使用非同步播放的方式初始化一个CDAudioManager对象，并选择一种特定的方式管理音效。

在这种方式下，如果其他应用正在播放音乐，那么当前应用在启动时不会停止之前应用的音乐，同时不播放当前应用中的音效。

1. 最后定义一个可变数组，并将所需要的背景音乐和音效文件载入到缓存之中。

注意 所载入的音频文件不能超过32个。

1. 播放背景音乐和音效

完成之前的设置工作后，就可以在需要的时候播放这些音频文件了。具体步骤如下所示。

1. 定义宏。

在Xcode中切换到HelloWorldLayer.m文件，在文件顶部添加代码如代码清单7-9所示。

1. HelloWorldLayer.m文件的顶部添加代码

#import "CDAudioManager.h"

#import "CocosDenshion.h"

// 使用以下宏定义来简化播放音乐的代码

/\*\* CDAudioManager 支持左右两个声道

typedef enum {

kASC\_Left = 0,

kASC\_Right = 1

} tAudioSourceChannel; \*/

#define CGROUP\_BG kASC\_Left // 背景音乐所在的声道

#define CGROUP\_EFFECTS kASC\_Right // 音效所在的声道

#define SND\_BG\_LOOP 1 // 背景音乐的标识

#define SND\_ROCKET 2 // 火箭炮音效的标识

#define SND\_CANNON 3 //大炮音效的标识

#define SND\_BLAST 4 //爆炸音效的标识

// 使用下面的宏来播放音乐

#define playEffect(\_\_ID\_\_) [[CDAudioManager sharedManager].soundEngine playSound:\_\_ID\_\_ sourceGroupId:CGROUP\_EFFECTS pitch:1.0f pan:0.0f gain:1.0f loop:NO]

以上代码和在AppDelegate.m中添加的代码完全相同，这里不再赘述。

1. 添加标签。

在游戏界面添加几个标签，当玩家触摸这些标签时，就会产生相应的音效。删除init方法中原有的代码，使用以下代码替代，如代码清单7-10所示。

1. 删除init方法中原有的代码，并使用以下代码替代

-(id) init

{

// always call "super" init

// Apple recommends to re-assign "self" with the "super" return value

if( (self=[super init])) {

//使用CCDirector单例对象来获取屏幕大小

CGSize winSize = [[CCDirector sharedDirector] winSize];

//定义一个字体

NSString \*fontName = @”Myfont.fnt";

//创建大炮标签，当触碰该标签时，会调用shootCannon方法

CCLabelBMFont \*cannonLabel = [CCLabelBMFont labelWithString:@"Cannon" fntFile:fontName];

CCMenuItemLabel \*cannonItem = [CCMenuItemLabel itemWithLabel:cannonLabel target:self selector:@selector(shootCannon:)];

cannonItem.scale = 0.5;

cannonItem.position = ccp(winSize.width/2,winSize.height \*0.5);

//创建火箭标签，当触碰该标签时，会调用shootRocket方法

CCLabelBMFont \*rocketLabel = [CCLabelBMFont labelWithString:@"Rocket" fntFile:fontName];

CCMenuItemLabel \*rocketItem = [CCMenuItemLabel itemWithLabel:rocketLabel target:self selector:@selector(shootRocket:)];

rocketItem.scale = 0.5;

rocketItem.position = ccp(winSize.width/2,winSize.height \*0.35);

//创建爆炸标签，当触碰该标签时，会调用blast方法

CCLabelBMFont \*blastLabel = [CCLabelBMFont labelWithString:@"Blast" fntFile:fontName];

CCMenuItemLabel \*blastItem = [CCMenuItemLabel itemWithLabel:blastLabel target:self selector:@selector(blast:)];

blastItem.scale = 0.5;

blastItem.position = ccp(winSize.width/2,winSize.height \*0.7);

//创建播放标签，当触碰该标签时，会调用playMusic方法

CCLabelBMFont \*playLabel = [CCLabelBMFont labelWithString:@"Play Music" fntFile:fontName];

CCMenuItemLabel \*playItem = [CCMenuItemLabel itemWithLabel:playLabel target:self selector:@selector(playMusic:)];

playItem.scale = 0.5;

playItem.position = ccp(winSize.width/2,winSize.height \*0.9);

//创建控制菜单，并将以上标签添加进去

CCMenu \*menu =[CCMenu menuWithItems:cannonItem,rocketItem,blastItem,playItem, nil];

menu.position = CGPointZero;

[self addChild:menu];

}

return self;

}

以上代码创建一个菜单，包含4个菜单项，其中3个用于播放音效，1个用于播放背景音乐。当玩家触碰其中某个按钮时，会调用相应的方法播放音效或音乐。

1. 添加播放代码。

在init方法下添加playMusic,blast,shootCannon和shootRocket等方法，如代码清单7-11所示。

1. 在init方法的下面添加方法

-(void)playMusic:(id)sender{

[[CDAudioManager sharedManager] playBackgroundMusic:@"bgmusic.mp3" loop:YES];

}

-(void)blast:(id)sender{

playEffect(SND\_BLAST);

}

-(void)shootCannon:(id)sender{

playEffect(SND\_CANNON);

}

-(void)shootRocket:(id)sender{

playEffect(SND\_ROCKET);

}

以上代码一目了然，其中playMusic方法用来播放背景音乐，blast方法用来播放爆炸音效，shootCannon用来播放发射大炮的音效，而shootRocket方法则用来播放发射火箭炮的音效。

1. 编译运行项目。

此时，可以随意播放自己需要的音乐或音效，如图7-8所示。



1. CDSoundTest项目编译运行后的效果

除了使用宏定义中的playEffect播放音效，还可以使用以下代码播放音效。

找到blast方法，使用以下代码替换：

-(void)blast:(id)sender{

// playEffect(SND\_BLAST);

[[CDAudioManager sharedManager].soundEngine playSound:SND\_BLAST sourceGroupId:CGROUP\_BG pitch:1.0f pan:0.0f gain:1.0f loop:NO];

}

编译运行，会发现和使用宏定义的效果一样。这样做的好处是可以根据程序的需要灵活更改相关的参数。

要停止播放该音效，只需使用以下代码：

ALuint blast = [[CDAudioManager sharedManager].soundEngine playSound:SND\_BLAST sourceGroupId:CGROUP\_BG pitch:1.0f pan:0.0f gain:1.0f loop:NO];

[[CDAudioManager sharedManager].soundEngine stopSound:blast];

此时编译运行项目，单击blast标签后不会有任何反应。

注意 上述方法只适用于播放音效，如果需要修改背景音乐的音高、音源位置和音量等信息，可以将背景音乐作为音效来播放。

## 游戏菜单中的声音设置选项

很多游戏在菜单中提供了设置声音的选项，即声音设置界面，可以启用或关闭背景音乐、设置音量的大小等。实际项目中应该如何使用声音设置界面呢？本节通过示例项目AudioSetting详细说明。

打开Xcode，单击File→New Project，选择iOS→Cocos2D→Cocos2D模板，单击Next，在Product Name处输入AudioSetting，单击Next，选择存放的目录后，单击Create。

把chapter7/resource/audio/AudioSetting/目录下面的音效文件以及chapter7/resource/fonts/目录下面的myfont.fnt和myfont.png文件拖动到Resources中，确保选中“Copy items into destination group’s folder(if needed)”。

#### 创建菜单

切换到HelloWorldLayer.m文件，在init方法中清除之前的所有代码，并添加以下代码创建一个菜单，如代码清单7-12所示。

1. 创建一个菜单

-(id) init

{

// always call "super" init

// Apple recommends to re-assign "self" with the "super" return value

if( (self=[super init])) {

//使用CCDirector单例对象来获取屏幕大小

CGSize winSize = [[CCDirector sharedDirector] winSize];

//定义一个字体

NSString \*fontName = @"myfont.fnt";

//创建播放标签，当触碰播放标签时，会调用playSong方法

CCLabelBMFont \*playLabel = [CCLabelBMFont labelWithString:@"Touch to play music" fntFile:fontName];

CCMenuItemLabel \*playItem = [CCMenuItemLabel itemWithLabel:playLabel target:self selector:@selector(playSong:)];

playItem.scale = 0.7;

playItem.position = ccp(winSize.width/2,winSize.height \*0.7);

//创建设置标签，当触碰播放标签时，会调用settingSound方法

CCLabelBMFont \*settingLabel = [CCLabelBMFont labelWithString:@"Setting" fntFile:fontName];

CCMenuItemLabel \*settingItem = [CCMenuItemLabel itemWithLabel:settingLabel target:self selector:@selector(settingSound:)];

settingItem.scale = 0.5;

settingItem.position = ccp(winSize.width/2,winSize.height \*0.35);

//创建控制菜单，并将以上标签添加进去

CCMenu \*menu =[CCMenu menuWithItems:playItem,settingItem,nil];

menu.position = CGPointZero;

[self addChild:menu];

}

return self;

}

以上代码一目了然，只是在屏幕中创建了一个菜单，该菜单共包含两个菜单项，其中一个触碰后会播放一段音乐，而另一个触碰后则会进入音效设置界面。

#### 添加播放音乐方法

在init方法后添加playSong:方法和settingSound:方法，如代码清单7-13所示。

1. 在init方法后添加两个方法

-(void)playSong:(id)sender{

//添加一行代码以播放背景音乐：

[[SimpleAudioEngine sharedEngine]playBackgroundMusic:@"bgmusic.mp3" loop:YES];

}

-(void)settingSound:(id)sender{

[[SimpleAudioEngine sharedEngine]stopBackgroundMusic];

CCTransitionSlideInL \*transitionScene = [CCTransitionSlideInL transitionWithDuration:3.0 scene:[SoundSetting scene]];

[[CCDirector sharedDirector]replaceScene:transitionScene];

}

以上两个方法的作用如下：

* playSong:方法简单地调用了SimpleAudioEngine的单例循环播放背景音乐。
* settingSound:方法先停止播放背景音乐，然后使用一个场景切换效果切换到SoundSetting场景。

#### 创建SoundSetting类

在Xcode中单击File→New→New File，选择Cocos2D→CCNode class，单击Next，选择Subclass of CCLayer，单击Next，保存为SoundSetting.m，单击Create确认。

在Xcode中切换到SoundSetting.h文件，在类声明部分的下面添加以下代码：

+(id)scene;

切换到SoundSetting.m文件，在文件的顶部添加以下代码：

#import "HelloWorldLayer.h"

#import "CDAudioManager.h"

#### 创建并初始化当前场景和层

接着刚才的代码添加scene类方法的实现代码，如代码清单7-14所示。

1. 添加scene类方法的实现代码

+(id)scene{

CCScene \*scene = [CCScene node];

SoundSetting \*layer = [SoundSetting node];

[scene addChild:layer];

return scene;

}

这个方法创建并初始化当前的场景和层，将层添加为当前场景的子节点，最后返回该场景。

#### 添加控制开关的菜单

在init方法中添加一个菜单，其中包括3个菜单项，如代码清单7-15所示。

1. 在init方法中添加一个菜单

-(id)init{

if((self = [super init])){

//使用CCDirector单例对象来获取屏幕大小

CGSize winSize = [[CCDirector sharedDirector] winSize];

//定义一个字体

NSString \*fontName = @"myfont.fnt";

//设置字体和大小

[CCMenuItemFont setFontName: @"Helvetica-Bold"];

[CCMenuItemFont setFontSize:18];

//设置切换音乐的选项。菜单状态：开和关

CCMenuItemFont \*musicTitle = [CCMenuItemFont itemWithString:@"触摸下面的标签开启或关闭音乐"];

[musicTitle setIsEnabled:NO];

musicTitle.position = ccp(winSize.width \*0.5,winSize.height \*0.7);

CCMenuItemFont \*musicOn = [CCMenuItemFont itemWithString:@"On"];

CCMenuItemFont \*musicOff = [CCMenuItemFont itemWithString:@"Off"];

CCMenuItemToggle \*music = [CCMenuItemToggle itemWithTarget:self selector:@selector(changeStatus:) items:musicOff, musicOn,nil];

music.position = ccp(winSize.width \*0.5,winSize.height \*0.6);

//创建返回标签，当触碰该标签时，会调用backToMainMenu方法

CCLabelBMFont \*returnLabel = [CCLabelBMFont labelWithString:@"Back to main menu" fntFile:fontName];

CCMenuItemLabel \*returnItem = [CCMenuItemLabel itemWithLabel:returnLabel target:self selector:@selector(backToMainMenu:)];

returnItem.scale = 0.7;

returnItem.position = ccp(winSize.width\*0.5,winSize.height \*0.35);

//创建控制菜单，并将以上标签添加进去

CCMenu \*menu =[CCMenu menuWithItems:musicTitle,music,returnItem,nil];

menu.position = CGPointZero;

[self addChild:menu];

}

return self;

}

其中musicTitle菜单项提醒玩家“触摸下面的标签开启或关闭音乐”，music切换开关用于实际开启或关闭音乐，最后添加一个返回标签，用来返回主场景。

#### 切换静音状态

在init方法后添加changeStatus方法用以切换项目中是否开启静音状态，如代码清单7-16所示。

1. 添加changeStatus方法以切换静音状态

-(void)changeStatus:(CCMenuItemToggle \*)sender{

if([CDAudioManager sharedManager].mute == TRUE){

[CDAudioManager sharedManager].mute =FALSE;

}else{

[CDAudioManager sharedManager].mute = TRUE;

}

}

这个方法是需要重点了解的，在使用切换开关来切换选项时，使用CDAudioManager的单例对象的mute（静音）属性进行判断，并对该状态进行切换。

#### 返回之前的场景

最后添加一个方法返回之前的场景，如代码清单7-17所示。

1. 返回之前的场景

-(void)backToMainMenu:(id)sender{

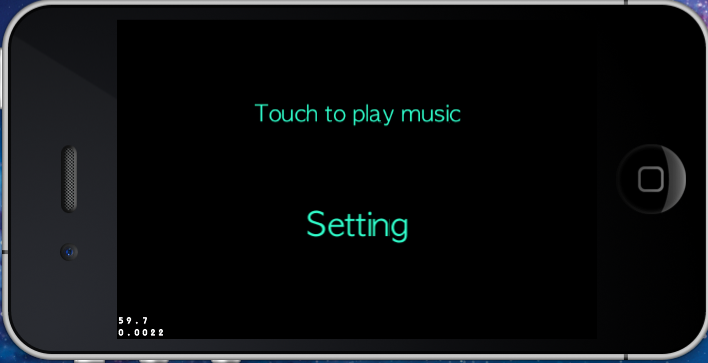
CCTransitionFlipX \*transitionScene = [CCTransitionFlipX transitionWithDuration:3.0 scene:[HelloWorldLayer scene]];

[[CCDirector sharedDirector]replaceScene:transitionScene];

}

这个方法不难理解，这里定义一个CCTransitionFlipX场景切换效果，并使用CCDirector单例对象来切换场景。

此时编译运行程序，主界面如图7-9所示。



1. 编译运行后的主界面

触碰“Touch to play music”标签时，会播放背景音乐；触碰“Setting”标签进入设置界面后，会看到默认状态是On，触碰该标签可以切换音乐开关。然后返回主界面，此时触碰“Touch to play music”标签，就不会再播放背景音乐了。

但这里还有个小问题，如果再次进入设置界面，会发现音乐开关的状态仍然是On。这是因为，虽然之前对音乐开关的状态进行了设置，但是程序界面并没有“记住”这种选择。因此这里需要用到NSUserDefaults进行默认用户设置。

#### 设置默认用户

使用NSUserDefaults方法设置默认用户。

在Xcode中切换到SoundSetting.m，找到init方法，在添加菜单为当前层的子节点这行代码之后添加以下代码：

NSUserDefaults \*usrDef = [NSUserDefaults standardUserDefaults];

if([usrDef boolForKey:@"music"] == NO)

music.selectedIndex = 1;

在changeStatus方法中添加以下代码：

NSUserDefaults \*usrDef = [NSUserDefaults standardUserDefaults];

if(sender.selectedIndex ==1)

[usrDef setBool:NO forKey:@"music"];

if(sender.selectedIndex ==0)

[usrDef setBool:YES forKey:@"music"];

此时编译运行程序，设置界面的显示和实际的设置就完全吻合。当用户开启音乐时，音效开关界面中的开关选项会切换到On；反之，则开关选项切换到Off，如图7-10和图7-11所示。



1. 音效开关选项切换到On



1. 音效开关选项切换到Off

## 垂直射击游戏：GameOver

目前，垂直射击游戏在暂停按钮按下去时，游戏背景音乐并没有停止。本节在玩家按下暂停按钮时把游戏背景音乐也一并暂停。同时，为垂直射击游戏添加声音设置界面。最后游戏结束时，显示本次游戏的分数，并存入档案且保留最高分前三名，玩家可以在高分榜画面查看。

### 实现背景音乐暂停功能

添加背景音乐暂停功能步骤如下。

1. 找到HelloWorldLayer.m的init方法，在“暂停”按钮的回调方法中添加如下代码。

[[SimpleAudioEngine sharedEngine] pauseBackgroundMusic];

1. 找到didRecieveResumeEvent方法，并在该方法的最后添加以下代码。

[[SimpleAudioEngine sharedEngine] resumeBackgroundMusic];

这样可以保证玩家继续游戏之后，背景音乐也能继续。

### 实现菜单场景

现在为游戏添加菜单场景，此场景在Loading场景之后出现。

#### 实现MenuLayer类

新建一个文件MenuLayer继承自CCLayer，用代码清单7-18替换MenuLayer.h文件的内容。

1. MenuLayer.h

#import <Foundation/Foundation.h>

#import "cocos2d.h"

@interface MenuLayer : CCLayer {

}

+(id) scene;

-(void) highScoreScene;

-(void) settingsScene;

-(void) gameScene;

@end

这段代码很常见，静态方法scene供切换场景时调用；下面3个方法分别对应3个菜单项的事件处理方法，触发后将分别进入“高分榜”、“设置”和“游戏画面”。

接下来，看看MenuLayer的实现。

打开MenuLayer.m文件，用代码清单7-19中的所示代码替换其中的内容。

1. MenuLayer.m

#import "MenuLayer.h"

#import "GameLayer.h"

@implementation MenuLayer

+(id) scene{

CCScene \*sc = [CCScene node];

MenuLayer\* layer = [MenuLayer node];

[sc addChild:layer];

return sc;

}

-(id) init{

if ((self = [super init])) {

CCMenuItem \*highScoreItem = [CCMenuItemFont itemWithString:@"高分榜"

target:self selector:@selector(highScoreScene)];

CCMenuItem \*settingsItem = [CCMenuItemFont itemWithString:@"设置"

target:self selector:@selector(settingsScene)];

CCMenuItem \*gameItem = [CCMenuItemFont itemWithString:@"进入游戏"

target:self selector:@selector(gameScene)];

CCMenu \*menu = [CCMenu menuWithItems:gameItem,highScoreItem,settingsItem, nil];

menu.position = ccp(160,240);

[self addChild:menu];

[menu alignItemsVerticallyWithPadding:20];

}

return self;

}

-(void) highScoreScene{

}

-(void) settingsScene{

}

-(void) gameScene{

[[CCDirector sharedDirector] replaceScene:[CCTransitionFade transitionWithDuration:0.5f

scene:[GameLayer scene]

withColor:ccWHITE]];

}

@end

首先看到的是init方法，这里定义3个菜单项，分别是“进入游戏”、“高分榜”和“设置”，每个菜单项对应一个回调方法。最后调用菜单的alignItemsVerticallyWithPadding使这些菜单项按照垂直方向排列，且间距为20像素。

接下来，定义3个方法，其中highScoreScene和settingsScene暂时都是空的。而gameScene的实现也非常简单，就是以淡出的方式进入游戏场景。所以要实现settingsScene。菜单界面效果如图7-12所示。



1. MenuLayer界面运行效果

#### 实现SettingsLayer类

右键单击VerticalShootingGame分组，选择New File，选择Cocos2d.x模板，选择CCLayer作为基类，命名为SettingsLayer。

SettingsLayer.h的定义如代码清单7-20所示。

1. SettingsLayer.h的定义

#import <Foundation/Foundation.h>

#import "cocos2d.h"

@interface SettingsLayer : CCLayer {

}

+(id) scene;

-(void) toggleGameMusic:(id)sender;

-(void) toggleGameSoundEffect:(id)sender;

-(void) goBack;

@end

这段代码也比较简单，首先定义一个scene方法，此方法的实现省略。接下来的3个方法都是菜单的回调方法，分别是开关背景音乐、开关游戏音效和回到主菜单。

接下来，看看SettingsLayer.m的实现。

添加头文件和常量定义，如下所示。

#import "SimpleAudioEngine.h"

#define kMusicKey @"music"

#define kAudioKey @"audio"

更改init方法的实现代码，如代码清单7-21所示。

1. init方法的实现代码

-(id) init{

if ((self = [super init])) {

CCLabelTTF \*musicLabel = [CCLabelTTF labelWithString:@"背景音乐:" fontName:@"Arial" fontSize:20];

musicLabel.position = ccp(100,280);

[self addChild:musicLabel];

CCLabelTTF \*audioLabel = [CCLabelTTF labelWithString:@"游戏音效:" fontName:@"Arial" fontSize:20];

audioLabel.position = ccp(100,240);

[self addChild:audioLabel];

CCMenuItem \*musicOnItem = [CCMenuItemFont itemWithString:@"开" target:self

selector:nil];

musicOnItem.tag = 1;

CCMenuItem \*musicOffItem = [CCMenuItemFont itemWithString:@"关" target:self

selector:nil];

musicOnItem.tag = 0;

CCMenuItemToggle \*musicItem = [CCMenuItemToggle itemWithTarget:self

selector:@selector(toggleGameMusic:) items:musicOffItem,musicOnItem, nil];

BOOL musicState = [[[NSUserDefaults standardUserDefaults] objectForKey:kMusicKey] boolValue];

if (musicState) {

musicItem.selectedIndex = 0;

}else{

musicItem.selectedIndex = 1;

}

musicItem.position = ccp(200,280);

CCMenuItem \*audioOnItem = [CCMenuItemFont itemWithString:@"开" target:self

selector:nil];

audioOnItem.tag = 1;

CCMenuItem \*audioOffItem = [CCMenuItemFont itemWithString:@"关" target:self

selector:nil];

audioOffItem.tag = 0;

CCMenuItemToggle \*audioItem = [CCMenuItemToggle itemWithTarget:self selector:@selector(toggleGameSoundEffect:) items:audioOffItem,audioOnItem, nil];

audioItem.position = ccp(200,240);

BOOL audioState = [[[NSUserDefaults standardUserDefaults] objectForKey:kAudioKey] boolValue];

if (audioState) {

audioItem.selectedIndex = 0;

}else {

audioItem.selectedIndex = 1;

}

CCMenuItem \*backItem = [CCMenuItemFont itemWithString:@"返回"

target:self selector:@selector(goBack)];

backItem.position = ccp(50,400);

CCMenu \*menu = [CCMenu menuWithItems:musicItem,audioItem,backItem, nil];

menu.position = ccp(0,0);

[self addChild:menu];

}

return self;

}

这段代码有几点需要说明，首先，创建菜单的部分使用了CCMenuItemToggle，它需要接受两个MenuItem作为参数，其实就是常见的开关按钮。使用此按钮时，有一个重要属性selectedIndex，从0开始，在初始化时，哪个按钮写在前面，就是selectedIndex为0。

上面例子中，索引值为0的菜单项是musicOffItem和audioOffItem。正如前面已经讲到的，使用了NSUserDefaults保存用户的声音选项。

提示 这里使用一个技巧，因为刚开始第一次设置时，音乐和音效都是开的，而此时NSUserDefaults保存的key所对应的值是false。所以，使用false代表开启状态，而true表示关闭状态。最后需要注意，初始化时要根据NSUserDefaults保存的key所对应的值设置开关按钮的当前状态。

#### 菜单项回调方法的实现

3个菜单项回调方法的具体实现如代码清单7-22所示。

1. 3个菜单项的回调方法

-(void) toggleGameMusic:(id)sender{

CCMenuItemToggle \*item = (CCMenuItemToggle\*)sender;

if (item.selectedIndex == 1) {

CCLOG(@"music on");

[SimpleAudioEngine sharedEngine].backgroundMusicVolume = 1;

[[NSUserDefaults standardUserDefaults] setObject:[NSNumber numberWithBool:NO] forKey:kMusicKey];

}else{

CCLOG(@"music off");

[SimpleAudioEngine sharedEngine].backgroundMusicVolume = 0;

[[NSUserDefaults standardUserDefaults] setObject:[NSNumber numberWithBool:YES] forKey:kMusicKey];

}

}

-(void) toggleGameSoundEffect:(id)sender{

CCMenuItemToggle \*item = (CCMenuItemToggle\*)sender;

if (item.selectedIndex == 1) {

CCLOG(@"audio on");

[SimpleAudioEngine sharedEngine].effectsVolume = 1;

[[NSUserDefaults standardUserDefaults] setObject:[NSNumber numberWithBool:NO] forKey:kAudioKey];

}else{

CCLOG(@"audio off");

[SimpleAudioEngine sharedEngine].effectsVolume = 0;

[[NSUserDefaults standardUserDefaults] setObject:[NSNumber numberWithBool:YES] forKey:kAudioKey];

}

}

-(void) goBack{

[[CCDirector sharedDirector] popScene];

}

这里的代码也较熟悉，但是有几点值得注意。

和之前不同，这里没有使用SimpleAudioEngine的mute属性静音，而是通过修改各自的音量。因为一旦设置了mute属性，那么游戏中所有的背景音乐和音效都会听不到。但是，有许多游戏同时提供单方面禁止背景音乐或者禁止音效，这样就必须使用上面提到的做法，修改音量。最后是goBack方法，因为SettingsLayer通过pushScene进入舞台，所以需要通过popScene退出舞台。

最后，把MenuLayer的settingsScene用下面代码替换，代码如代码清单7-23所示。

1. settingsScene方法

-(void) settingsScene{

[[CCDirector sharedDirector] pushScene:[SettingsLayer scene]];

}

有一点需要注意，玩家第一次进入玩游戏时，默认背景音乐音量和音效音量都是开启的，所以，需要在AppDelegate.m里添加相应的声音设置初始化代码。

找到AppDelegate.m文件，在以下方法返回之前添加代码：

-(BOOL) application:(UIApplication \*)application didFinishLaunching WithOptions:(NSDictionary \*)launchOptions

添加的内容如代码清单7-24所示。

1. AppDelegate.m文件添加代码

//loading audio & music config

BOOL audioState = [[[NSUserDefaults standardUserDefaults] objectForKey:kAudioKey] boolValue];

if (audioState) {

[SimpleAudioEngine sharedEngine].effectsVolume = 0;

}

BOOL musicState= [[[NSUserDefaults standardUserDefaults] objectForKey:kMusicKey] boolValue];

if (musicState) {

[SimpleAudioEngine sharedEngine].backgroundMusicVolume = 0;

}

编译并运行，加载完毕进入菜单系统后，单击设置得到如图7-13所示输出。



1. 编译运行后的输出

### 实现高分榜场景

本节一起来完成高分榜场景。

#### 实现HighScoreLayer类

首先新建一个HighScoreLayer类继承自CCLayer，同时在头文件中定义两个方法，如下所示。

+(id) scene;

-(void) goBack;

scene方法不再赘述。goBack的实现之前也见过，就是调用popScene，可以参考本章附带源程序。

接下来是重要的init方法，此方法完成高分榜前三名的显示，同时还定义一个返回主菜单的按钮。实现代码如代码清单7-25所示。

1. 高分榜前三名的显示

-(id) init{

if ((self = [super init])) {

CGSize winSize = [[CCDirector sharedDirector] winSize];

CCLabelTTF \*titleLabel = [CCLabelTTF labelWithString:@"前三名:" fontName:@"Arial" fontSize:20];

titleLabel.position = ccp(winSize.width / 2, winSize.height - 50);

[self addChild:titleLabel];

//根据游戏存档显示前三名的分数

const int SCORE\_POS\_X = winSize.width / 2;

const int SCORE\_POS\_Y = winSize.height - 100;

const int VERTICAL\_PADDING = 50;

NSArray \*highScores = [GameManager sharedGameManager].top3Scores;

for (int i=0; i < [highScores count]; ++i) {

int score = [[highScores objectAtIndex:i] intValue];

NSString \*scoreString = [NSString stringWithFormat:@"%d",score];

CCLabelTTF \*scoreLabel = [CCLabelTTF labelWithString:scoreString fontName:@"Arial" fontSize:20];

scoreLabel.position = CGPointMake(SCORE\_POS\_X, SCORE\_POS\_Y - i \* VERTICAL\_PADDING);

[self addChild:scoreLabel];

}

CCMenuItem \*backItem = [CCMenuItemFont itemWithString:@"返回主菜单" target:self selector:@selector(goBack)];

backItem.position = CGPointMake(100, 50);

CCMenu \*menu = [CCMenu menuWithItems:backItem, nil];

menu.position = CGPointZero;

[self addChild:menu];

}

return self;

}

需要注意，这里引用了一个新的GameManager类，从GameManager类中获取高分榜前三名的数据。以此数据初始化3个标签显示在屏幕上面。代码和之前介绍的标签用法并无太大差别，关键是以下这行代码耐人寻味：

NSArray \*highScores = [GameManager sharedGameManager].top3Scores

因为游戏的高分榜含历史记录，每当有新的游戏记录产生，都需要更新高分榜。所以，游戏里需要提供一种保存游戏数据的方式。由于保存游戏数据是每一个游戏开发者都需要面临的问题，所以非常重要。同时，为了使这种保存方式具有最大限度的可重用性，代码在组织结构上做了一些规划。HighScoreLayer的最终效果如图7-14所示：



图7-14 高分榜界面运行效果

#### 实现GameManager类

新建一个GameManager类，继承自NSObject。此类将给该游戏程序提供一个全局的访问点。任何模块之间想要交流信息都可以通过这个类做中转。

GameManager类的头文件定义如代码清单7-26所示。

1. GameManager类的头文件定义

@interface GameManager : NSObject<NSCoding>{

NSMutableArray \*highScores; //保存游戏高分榜前三台，需要序列化的

BOOL isWin; //游戏结束时设置，如果赢了则为YES，临时使用

int score; //当前游戏分数，临时使用，偶尔也能用

int playTimes; //总共玩了几把游戏.如果超过3次，则在玩家第3次打开游戏时弹出提示，请玩家为游戏评分

}

@property(nonatomic,copy)NSMutableArray\* highScores;

@property(nonatomic,assign)BOOL isWin;

@property(nonatomic,assign)int score;

@property(nonatomic,assign)int playTimes;

+(GameManager\*) sharedGameManager;

+(void) purge;

+(void) loadState;

+(void) saveState;

+(NSString\*) makeSavePath;

-(void) clear;

-(NSMutableArray\*) top3Scores;

-(void) addNewHighScore:(int)newScore;

@end

这个类定义一个highScores数组，用来保存高分榜前三名。然后是一些游戏过程中产生的临时数据。接下来定义的这些静态方法都可以重用，适用任何需要在不同的场景之间传递数据的情况。其实就是单例模式的运用，配合NSCoding协议达到序列第对象的目的。具体看看下面几个方法的用途，如代码清单7-27所示。

1. 序列第对象的实现

-(void) clear

{

// erase all data

[highScores release];

highScores = nil;

playTimes = 0;

isWin = false;

score = 0;

}

-(NSMutableArray\*) top3Scores{

if (nil == highScores) {

highScores = [NSMutableArray arrayWithCapacity:3];

[highScores retain];

}

while ([highScores count] < 3) {

[highScores addObject:[NSNumber numberWithInt:0]];

}

NSSortDescriptor \*descriptor = [NSSortDescriptor sortDescriptorWithKey:nil

ascending:NO

selector:@selector(compare:)];

[highScores sortUsingDescriptors:[NSArray arrayWithObject:descriptor]];

return highScores;

}

-(void) addNewHighScore:(int)newScore{

if (nil == highScores) {

highScores = [NSMutableArray arrayWithCapacity:3];

[highScores retain];

}

if ([highScores count] < 3) {

[highScores addObject:[NSNumber numberWithInt:newScore]];

}

else{

NSSortDescriptor \*descriptor = [NSSortDescriptor sortDescriptorWithKey:nil

ascending:NO

selector:@selector(compare:)];

[highScores sortUsingDescriptors:[NSArray arrayWithObject:descriptor]];

int lastScore = [[highScores objectAtIndex:2] intValue];

if (newScore > lastScore) {

[highScores removeObjectAtIndex:2];

[highScores addObject:[NSNumber numberWithInt:newScore]];

}

}

}

-(id) initWithCoder:(NSCoder\*)coder

{

self = [super init];

if( self != nil )

{

// decode data

highScores = [[coder decodeObjectForKey:kHightScoreKey] retain];

playTimes = [[coder decodeObjectForKey:kPlayTimesKey] intValue];

NSLog(@"initWithCoder = %@",self);

}

return self;

}

-(void) encodeWithCoder:(NSCoder\*)coder

{

// encode data

[coder encodeObject:highScores forKey:kHightScoreKey];

[coder encodeObject:[NSNumber numberWithInt:playTimes] forKey:kPlayTimesKey];

NSLog(@"encodeWithCoder = %@",self);

}

clear方法的作用就是初始化类中的成员变量，这个方法最终由init方法调用的。接下来的两个方法top3Scores和addNewHighScore是重点。它们都不复杂，但是对于边界条件的判断还是很重要的。比如数组为可以为nil就添加数据，addNewScore要看该数据是否比第三条记录的数值大。如果是，把原来数组第三号位置的元素移除掉。最后，把新的分数添加进来。

添加完以后，就需要按照分数的大小从大到小排序。使用以下代码完成。

NSSortDescriptor \*descriptor = [NSSortDescriptor sortDescriptorWithKey:nil

ascending:NO selector:@selector(compare:)];

[highScores sortUsingDescriptors:[NSArray arrayWithObject:descriptor]];

最后是initWithCoder和encodeWithCoder方法的实现，这里使用经典的序列化数据和反序列化数据。虽然GameManager的实现已经很清楚了，但是对于何时加载游戏数据，何时保存游戏数据，需要明确指出来。

一般程序刚开始启动的时候，肯定加载游戏数据。具体在AppDelegate.m里的applicationDidFinishLouch方法返回之前添加代码，如下所示。

//加载游戏存档数据

[GameManager loadState];

if ([GameManager sharedGameManager].highScores == nil) {

[[GameManager sharedGameManager] addNewHighScore:0];

[[GameManager sharedGameManager] addNewHighScore:0];

[[GameManager sharedGameManager] addNewHighScore:0];

}

这里判断如果highScore为nil，很有可能是第一次运行，游戏存档文件还不存在，所以会返回nil，然后添加3个0分进前三甲。

最后，在applicationWillResignActive、applicationDidEnterBackground和applicationWillTerminate中添加[GameManager saveState]保存游戏数据。

#### 实现GameLayer

接下来，看看最重要的GameLayer（由HelloWorldLayer改名而来）的实现（其实只需要修改部分逻辑即可）。首先是游戏结束和胜利画面。更新后的gameOver和onRestartGame方法的实现如代码清单7-28所示。

1. gameOver和onRestartGame方法的实现

-(void) gameOver{

[\_gameEndLabel setString:@"游戏失败!"];

\_gameEndLabel.visible = YES;

id scaleTo = [CCScaleTo actionWithDuration:1.0 scale:1.2f];

[\_gameEndLabel runAction:scaleTo];

[self unscheduleUpdate];

[\_backgroundNode stopAllActions];

[self unscheduleAllSelectors];

[GameManager sharedGameManager].isWin = NO;

[GameManager sharedGameManager].score = \_totalScore;

[GameManager sharedGameManager].playTimes = [GameManager sharedGameManager].playTimes + 1;

[[SimpleAudioEngine sharedEngine] stopBackgroundMusic];

CCTransitionFade\* transitionScene = [CCTransitionFade transitionWithDuration:2

scene:[GameOverLayer scene]

withColor:ccWHITE];

[[CCDirector sharedDirector] replaceScene:transitionScene];

// [self performSelector:@selector(onRestartGame) withObject:nil afterDelay:3.0f];

}

-(void) onRestartGame{

[GameManager sharedGameManager].isWin = YES;

//计算生命值的得分

while (\_totalLives > 0) {

\_totalScore += 1000;

\_totalLives--;

}

//计算剩余时间所得分

\_totalScore += (\_totalSeconds - \_consumedTime) \* 100;

[GameManager sharedGameManager].score = \_totalScore;

[GameManager sharedGameManager].playTimes = [GameManager sharedGameManager].playTimes + 1;

[[SimpleAudioEngine sharedEngine] stopBackgroundMusic];

CCTransitionFade\* transitionScene = [CCTransitionFade transitionWithDuration:1 scene:[GameOverLayer scene] withColor:ccWHITE];

[[CCDirector sharedDirector] replaceScene:transitionScene];

[[GameManager sharedGameManager] addNewHighScore:\_totalScore];

}

游戏胜利或者失败以后，不是直接就重新开始游戏，而是跳转到另外一个新的界面GameOverLayer。该场景会根据上一局游戏的结果显示成功过关还是失败。同时相应地提供“再来一盘”和“重新开始”选项。

在onRestartGame方法中计算当前游戏的得分，分为3个组成部分：

击中分数所获得分+剩余生命条数×1000+剩余时间×100

### 实现GameOver场景

不管游戏成功或者失败，都调转到一个GameOver场景。该场景可以显示上一局的分数，同时有3个菜单按钮，分别对应“重新开始”或者“再来一盘”（如果是游戏胜利就显示再来一盘，可以刷新记录嘛，如果失败了，则显示重新开始）和“回到主菜单”。

首先建立一个新场景，命名为GameOverLayer，继承到CCLayer。其头文件定义如代码清单7-29所示。

1. GameOverLayer头文件定义

@interface GameOverLayer : CCLayer {

}

+(id) scene;

-(void) restartGame;

-(void) backToMainMenu;

scene方法返回一个自动释放的CCScene对象。后面两个方法的作用也非常明显，就是“重新开始”游戏和“返回主菜单”。

接下来，看看GameOverLayer的实现。

首先是init方法，代码如代码清单7-30所示。

1. init方法代码

-(id) init{

if ((self = [super init])) {

CGSize winSize = [[CCDirector sharedDirector] winSize];

CCLabelTTF \*titleLabel = [CCLabelTTF labelWithString:@"0" fontName:@"Arial" fontSize:20];

BOOL isGameWin = [GameManager sharedGameManager].isWin;

if (isGameWin) {

[titleLabel setString:@"恭喜你！成功过关。"];

}else{

[titleLabel setString:@"很遗憾！您没能成功过关。"];

}

titleLabel.position = CGPointMake(winSize.width/2, winSize.height - 50);

[self addChild:titleLabel];

int score = [GameManager sharedGameManager].score;

NSString \*scoreString = [NSString stringWithFormat:@"您的分数是:%d",score];

CCLabelTTF \*scoreLabel = [CCLabelTTF labelWithString:scoreString fontName:@"Arial" fontSize:20];

scoreLabel.position = ccpSub(titleLabel.position, ccp(0,50));

[self addChild:scoreLabel];

NSString \*promtStr = nil;

if (isGameWin) {

promtStr = @"重新开始";

}else{

promtStr = @"再来一盘";

}

//初始化菜单项

CCMenuItem \*restartItem = [CCMenuItemFont itemWithString:promtStr

target:self

selector:@selector(restartGame)];

CCMenuItem \*backItem = [CCMenuItemFont itemWithString:@"回到主菜单"

target:self

selector:@selector(backToMainMenu)];

CCMenu \*menu = [CCMenu menuWithItems:restartItem,backItem, nil];

menu.position = ccp(winSize.width/2,winSize.height / 2);

[menu alignItemsVerticallyWithPadding:20];

[self addChild:menu];

}

return self;

}

这里唯一需要注意，根据GameManager保存的上一局的游戏状态（胜利or失败），显示不同的标签和菜单项名称。最后，还是初始化菜单项，然后绑定两个回调方法。

两个回调方法的具体实现如代码清单7-31所示。

1. 回调方法的实现

-(void) restartGame{

[[CCDirector sharedDirector] replaceScene:[CCTransitionFade transitionWithDuration:0.5f

scene:[GameLayer scene]

withColor:ccWHITE]];

}

-(void) backToMainMenu{

[[CCDirector sharedDirector] replaceScene:[CCTransitionFade transitionWithDuration:0.5f

scene:[MenuLayer scene]

withColor:ccWHITE]];

}

编译并运行，此时已是一个完整的垂直射击游戏。游戏结束时的GameOver界面如图7-15所示。



图7-15 GameOverLayer界面

## 本章小结

本章重点学习了如何让Cocos2D世界充满鸟语花香。首先对Cocos2D的音效引擎CocosDenshion做了详细介绍；然后学习如何使用SimpleAudioEngine和CDSoundEngine播放背景音乐和音效；最后了解如何在游戏中增加音效设置界面以开启和关闭声音选项，以及保存游戏数据、暂停游戏音效和背景音乐、显示游戏结束画面。

完成1~7章的内容之后，应该对Cocos2D的用法有基本了解，对于垂直射击类游戏的开发也不再陌生。相信大家已经能够开发各种充满乐趣的游戏了。

从第8章开始将学习Cocos2D的TiledMap，届时会使用一个新的游戏示例——超级玛利。