# 第2章 你的第一款iPhone游戏：垂直射击游戏

本章将以一个垂直射击游戏为题材，带领大家动手制作一个简单的游戏。主要目的是让大家对Cocos2D开发游戏有一个感性的认识，同时体验Cocos2D的强大以及易用性。之后的章节将引入更多的游戏元素，逐步完善此游戏。

## 准备工作

作为开发者，首先需要有一台iOS设备，iPhone、iPod或者iPad都可以；其次拥有开发者账号，按照第1章介绍的方法下载并安装开发者证书。这样才可以把游戏编译运行到真机上。

为什么非要真机呢？接下来向大家展示的游戏必须有真机才能测试，游戏中将使用加速计控制飞机的飞行，而在模拟器里是无法处理加速计事件的。

打开Xcode，使用Cocos2D iOS模板新建一个项目，将其命名为VerticalShootingGame，单击Next；选择一个目录单击Create。编译运行将看到经典的HelloWorld画面。

本节将构建一个垂直射击游戏。由于模拟器的默认方向是横向（Landscape），因此需要把设备方向发成垂直方向的，同时清除模板内容，为后面的工作做准备。

1. 打开AppDelegate.m，找到如代码清单2-1所示的代码。
2. 打开AppDelegate.m的代码

// Supported orientations: Landscape. Customize it for your own needs

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation

{

return UIInterfaceOrientationIsLandscape(interfaceOrientation);

}

找到以下语句：

return UIInterfaceOrientationIsLandscape(interfaceOrientation)

改成以下语句就可以：

return UIInterfaceOrientationIsPortrait(interfaceOrientation)

这时编译运行结果如图2-1所示。



1. 编译运行结果
2. 清除这个HelloWorld场景，同时去掉GameCenter的Achievement和Leaderboard。
3. 打开HelloWorldLayer.h，使用代码清单2-2所示代码替换。
4. 替换HelloWorldLayer.h代码

// When you import this file, you import all the Cocos2D classes

#import "Cocos2D.h"

// HelloWorldLayer

@interface HelloWorldLayer : CCLayer

{

}

// returns a CCScene that contains the HelloWorldLayer as the only child

+(CCScene \*) scene;

@end

这里主要去掉GameKit头文件，同时移除GKAchievementViewControllerDelegate和GKLeaderboardViewControllerDelegate两个代理。

1. 打开HelloWorldLayer.m，把init方法用代码清单2-3所示代码替换掉。
2. 替换init方法代码

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

-(id) init

{

// always call "super" init

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

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

}

return self;

}

1. 把GameKit相关的代理方法删除，找到代码清单2-4所示代码直接删除。
2. GameKit相关代理方法的待删除代码

#pragma mark GameKit delegate

-(void) achievementViewControllerDidFinish:(GKAchievementViewController \*)viewController

{

AppController \*app = (AppController\*) [[UIApplication sharedApplication] delegate];

[[app navController] dismissModalViewControllerAnimated:YES];

}

-(void) leaderboardViewControllerDidFinish:(GKLeaderboardViewController \*)viewController

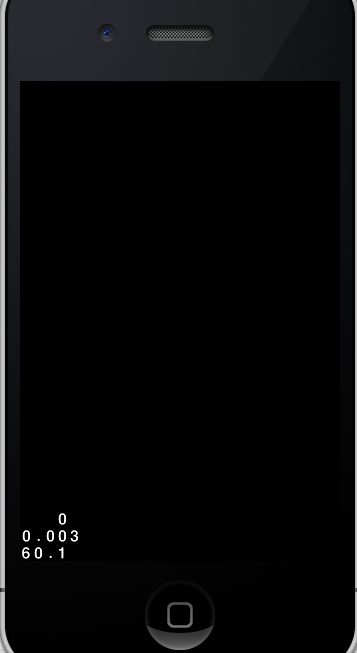
{

AppController \*app = (AppController\*) [[UIApplication sharedApplication] delegate];

[[app navController] dismissModalViewControllerAnimated:YES];

}

编译运行得到如图2-2所示结果。



1. 再次编译运行结果

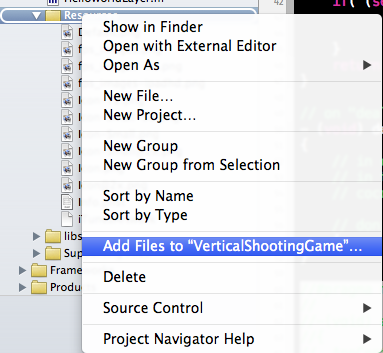
好，空空如也，没关系，这正是下一个游戏的伟大起点。

## 构建游戏场景

在构建游戏场景之前，需要将制作该游戏所需要资源文件包含进来。

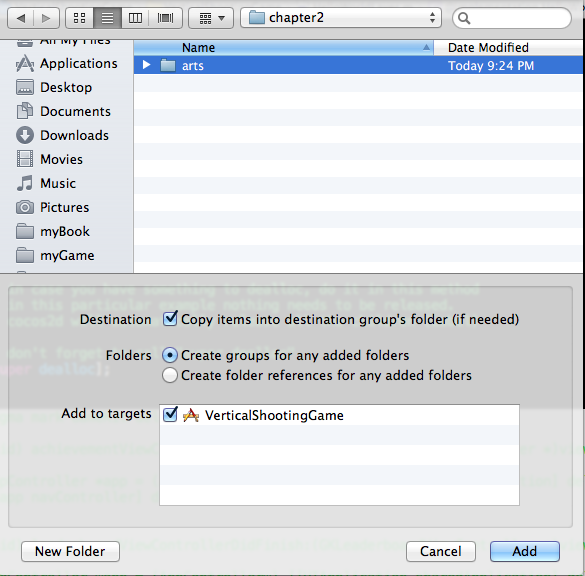
### 添加资源目录

找到本书的随书源代码，打开chapter2/resource/arts目录，把需要的图片和声音资源都添加到项目中。右键单击Resource，选择“Add files to VerticalShootingGame”，如图2-3所示。



1. 添加资源文件

浏览chapter/resource/arts目录，确保选中“Copy items into destination group’s folder(if needed)”，然后单击“Add”，如图2-4所示。



1. 选中“ Copy items into destination group’s folder(if needed)”后单击“Add”

### 添加游戏背景

一个漆黑的游戏背景上玩游戏，肯定会使人觉得乏味。接下来，给游戏添加一个静态背景图片。iPhone屏幕大小是480\*320，这里设置竖直放置，图片长度是480，宽度为320。

打开HelloWorldLayer.m，找到init方法，在if判断里添加代码清单2-5所示代码。

1. 在if判断里添加代码

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

//1.get screen's size

CGSize winSize = [[CCDirector sharedDirector] winSize];

//2.add background

CCSprite \*bgSprite = [CCSprite spriteWithFile:@" background\_1.jpg"];

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

[self addChild:bgSprite z:0];

}

按照代码注释中的标号解释这段代码的作用。

1. 通过CCDirector对象获得当前设备的屏幕大小。

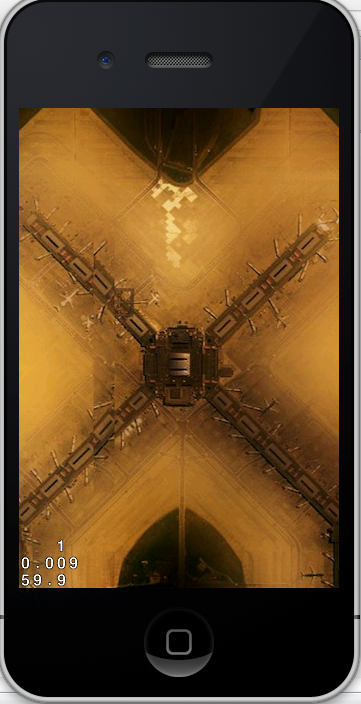
屏幕大小以point为单位，支持Retina显示的iPhone 4以及不支持Retina显示的iPhone 3GS屏幕都为（320，480）。想获得实际像素大小，也可以调用与之相应的像素级API winSizeInPixels。

1. 通过sprite的spriteWithFile方法，从background\_1.jpg实例化一个精灵，然后通过之前获得的屏幕大小计算出背景精灵图片放置的坐标位置。

因为精灵的坐标定位是相对于它的锚点（anchorPoint）得来，而sprite的anchorPoint为图片的中心点。所以，这里取屏幕的中点作为精灵的坐标点。最后，把该背景精灵图片作为当前层的孩子添加进去，Cocos2D框架会自动处理背景图片的渲染。

提示 这里背景图片格式为JPG，而不是大家所常见的PNG图片，因为背景图片一般不需要有透明像素，JPG图片和PNG图片最大的区别就是JPG图片没有Alpha通道，因而能够减少背景图片的大小，并且渲染的速度也会更快一些。

编译并运行，结果如图2-5所示。



1. 添加背景图片后编译运行结果

### 添加玩家飞机

好了，是时候添加玩家操控的飞机了。

打开HelloWorldLayer.m文件，在@implementation HelloWorldLayer的上面加上下列枚举定义，如代码清单2-6所示。

1. 在@implementation HelloWorldLayer的上面加上枚举定义

enum {

kTagPalyer = 1,

};

在init的代码片断2后面添加代码清单2-7所示代码。

1. 在init的代码片断2后面添加代码

//3.add player's plane

CCSprite \*playerSprite = [CCSprite spriteWithFile:@"hero\_1.png"];

playerSprite.position = CGPointMake(winSize.width / 2, playerSprite.contentSize.height/2 + 20);

[self addChild:playerSprite z:4 tag:kTagPalyer];

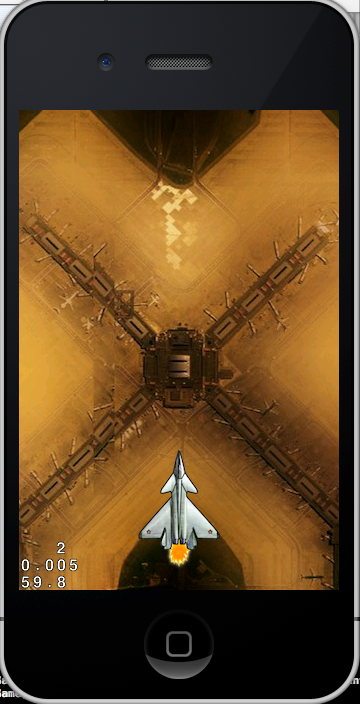
接下来，向大家解释代码片断3。

首先，从hero\_1.png文件初始化一个player精灵，通过屏幕大小与精灵纹理大小的运算，计算放置player精灵的坐标。

注意 这里使用的是相对坐标，而不是绝对坐标，这样写有助于代码的可适应性，比如移植到android平台后，多种不同分辨率的设备，如果使用绝对坐标定位就会导致界面布局混乱。这里的CGPointMake等价于ccp宏。

然后，调用CCNode的addChild方法，把player精灵加到当前layer中。这里使用带tag的addChild版本，方便后面使用getChildByTag提取层中的精灵。

编译并运行结果如图2-6所示。



1. 放置player精灵后编译运行结果

### 添加敌机

现在，需要添加一些敌机，希望敌机会从屏幕上方随机出现，然后向下俯冲。为了简单，这些敌机暂时不会发射子弹，随着学习的深入，玩家可以自行添加这个特性。

1. 添加一个CCArray \*\_enemySprites实例变量。

打开HelloWorldLayer.m文件，在init里代码片断3后面初始化此数组，如代码清单2-8所示。

1. 在init里初始化数组

//4.init enemy sprites array

\_enemySprites = [[CCArray alloc] init];

注意，这里的成员变量命名加了下划线作为前缀，它是一种编码风格，主要用于区别成员变量与局部变量。

1. 为了防止忘记释放内存，立刻在dealloc函数里释放掉该数组，如代码清单2-9所示。
2. 在dealloc函数里释放掉数组

[\_enemySprites release];

\_enemySprites = nil;

1. 初始化一系列的敌机精灵，并把这些精灵都添加到数组中。

找到init方法，在代码清单2-8所示代码后添加代码清单2-10所示代码。

1. 找到init方法添加代码

//5.initialize 10 enemy sprites & add them to \_enemySprites array for future useage

const int NUM\_OF\_ENEMIES = 10;

for (int i=0; i < NUM\_OF\_ENEMIES; ++i) {

CCSprite \*enemySprite = [CCSprite spriteWithFile:@"enemy1.png"];

enemySprite.position = ccp(0,winSize.height + enemySprite.contentSize.height + 10);

enemySprite.visible = NO;

[self addChild:enemySprite z:4];

[\_enemySprites addObject:enemySprite];

}

这段代码和前面的类似，唯一区别就是在一个循环里初始10个sprite，并且把sprite初始可见性设置为NO，最后把这些sprite都添加到\_enemySprites数组中。

如果现在运行项目，与上一次编译的结果没有区别，因为新初始化的敌机精灵是不可见的，同时它们的位置是在设备屏幕上方之外。现在，需要一些代码，希望能够隔一段时间有一架敌机随机从上方俯冲下来。

1. 打开init方法，在代码清单2-10后添加代码清单2-11所示代码。
2. init方法继续添加代码

//6.spawn enemy after 1.0 sec

[self performSelector:@selector(spawnEnemy)

withObject:nil

afterDelay:1.0f];

该方法作用是：等init方法调用完之后，隔1.0秒会调用spawnEnemy方法。这种特性非常有用，因为有时候经常需要在触发某个事件之后，不马上响应，而希望隔一段时间再做处理。

1. 定义一些私有方法。

其实在Objective-C世界里不存在真正意义的私有方法，有时为了不让API暴露给客户端程序员，一些类的内部使用的函数和变量需要放在私有扩展里，而不是直接声明在头文件中。

找到HelloWorldLayer.m的@implementation HelloWorldLayer部分，紧跟上面添加代码清单2-12所示代码。

1. @implementation HelloWorldLayer部分后添加代码

@interface HelloWorldLayer()

-(void) spawnEnemy;

-(CCSprite\*) getAvailableEnemySprite;

@end

这里定义一个匿名的Category，把一些私有方法直接扔到这里就可以。

1. 实现这两个方法。

找到@end，紧接在这之前添加代码清单2-13所示代码。

1. @end之前添加代码

#pragma mark - private methods

-(void) spawnEnemy{

//1.

CGSize winSize = [CCDirector sharedDirector].winSize;

CCSprite \*enemySprite = [self getAvailableEnemySprite];

//2.

float durationTime = arc4random() % 4 + 1;

id moveBy = [CCMoveBy actionWithDuration:durationTime

position:ccp(0,-enemySprite.position.y-enemySprite.contentSize.height)];

id callback = [CCCallBlockN actionWithBlock:^(id sender)

{

CCSprite \*sp = (CCSprite\*)sender;

sp.visible = NO;

sp.position = ccp(0,winSize.height + sp.contentSize.height + 10);

CCLOG(@"reset enemy plane!");

}];

id action = [CCSequence actions:moveBy,callback, nil];

enemySprite.visible = YES;

enemySprite.position = ccp( arc4random() % (int)(winSize.width - enemySprite.contentSize.width) + enemySprite.contentSize.width/2 , enemySprite.position.y);

CCLOG(@"enemySprite x = %f, y = %f",enemySprite.position.x, enemySprite.position.y);

[enemySprite runAction:action];

//3.

[self performSelector:\_cmd withObject:nil afterDelay:arc4random()%3 + 1];

}

-(CCSprite\*) getAvailableEnemySprite{

CCSprite \*result = nil;

CCARRAY\_FOREACH(\_enemySprites, result)

{

if (!result.visible) {

break;

}

}

return result;

}

按照代码注释的序号逐步讲解spawnEnemy方法：

1）通过CCDirector的winSize获得设备的大小（以point为单位）；通过调用自定义的getAvailableEnemySprite方法从\_enemySprites数组中获取一个还没有在飞行的飞机。

2）通过arc4random函数获取一个随机时间，以此时间建立一个CCMoveBy的action，同时计算出此action要移动的偏移量。接着，使用Block建立了一个回调action，最后通过CCSequence action把这两个action串在一起，把需要俯冲的敌机的可见性设置为yes。同时，为了视觉效果，在runAction之前，先设置敌机出现的坐标，改为随机位置出现。在moveBy的动作结束以后，精灵的可见性又变成NO，同时坐标也要发回初始化时的坐标位置。这一切都通过一种神奇函数式编程方式block做到。

3）隔1~4秒调用spawnEnemy方法，此处\_cmd参数就是spawnEnemy方法本身。这样，随机地每隔一段时间，就会有一架敌机以不同速度往下俯冲。

编译并运行，你应该会得到如图2-6所示效果！

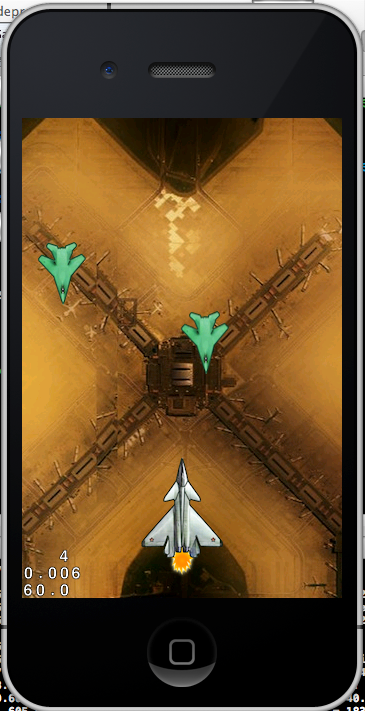


图2-7 敌机俯冲效果图

## 玩家交互

### 添加加速计移动

在Cocos2D世界里添加加速计支持非常简单，只需在当前运行的CCLayer的init方法返回之前，添加代码清单2-14所示代码。

1. 添加加速计支持

//7.enable accelerometer

self.isAccelerometerEnabled = YES;

为了让玩家能够使用加速计非常平滑地控制玩家飞机的飞行，这里提供一种通用的方法。因为游戏是垂直射击游戏，所以只关心设备垂直放置时，Home按键左右摆动方向的加速计值即可。

1. 打开HelloWorldLayer.h文件，添加一个CGPoint \_playerVelocity实例变量。
2. 打开HelloWorldLayer.m文件，在私有分类里添加-(void) updatePlayerPosition:(ccTime)dt方法，如图2-8所示。

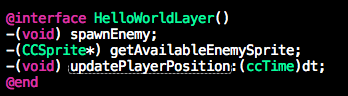


图2-8 在私有分类里添加-(void) updatePlayerPosition:(ccTime)dt方法

1. 在init方法返回之后，紧跟代码片断7添加函数调用,如代码清单2-15所示。
2. 紧跟代码片断7添加函数调用

//8.game main loop

[self scheduleUpdate];

此方法是本游戏的主循环，任何游戏都包含一个游戏主循环。此主循环类用来更新游戏世界的状态、碰撞处理逻辑和玩家状态等。这里主要更新玩家的位置信息。

1. 定义一个update函数，调用更新玩家位置信息的方法，如代码清单2-16所示。
2. 定义update函数

-(void) update:(ccTime)dt{

[self updatePlayerPosition:dt];

}

在讨论updatePlayerPositoin方法之前，先看看如何操作加速计，同时保存加速计的数据。在update方法之后，添加代码清单2-17所示代码。

1. 操作加速计

#pragma mark - accelerometer callback

-(void) accelerometer:(UIAccelerometer \*)accelerometer didAccelerate:(UIAcceleration \*)acceleration{

float deceleration = 0.4f;

float sensitivity = 6.0f;

float maxVelocity = 100;

playerVelocity.x = playerVelocity.x \* deceleration + acceleration.x \* sensitivity;

if (playerVelocity.x > maxVelocity) {

playerVelocity.x = maxVelocity;

}else if(playerVelocity.x < -maxVelocity){

playerVelocity.x = -maxVelocity;

}

}

这里主要做两件事：

* 通过“滤波”方式，从加速计x轴方向取得数据，计算得到一个“合理”的速度值。
* 限制飞机飞行的最大速度。

注意 如果想获取其他方向的值，相应地计算acceleration的其他两个分量的值就可以了。

最后，看看实际更新玩家飞机位置信息的方法实现，如代码清单2-18所示。

1. 更新玩家飞机位置信息的方法实现

-(void) updatePlayerPosition:(ccTime)dt{

CCSprite \*playerSprite = (CCSprite\*)[self getChildByTag:kTagPalyer];

CGPoint pos = playerSprite.position;

pos.x += playerVelocity.x;

CGSize screenSize = [CCDirector sharedDirector].winSize;

float imageWidthHavled = playerSprite.texture.contentSize.width \* 0.5f;

float leftBoderLimit = imageWidthHavled;

float rightBoderLimit = screenSize.width - imageWidthHavled;

if (pos.x < leftBoderLimit) {

pos.x = leftBoderLimit;

playerVelocity = CGPointZero;

}else if(pos.x > rightBoderLimit){

pos.x = rightBoderLimit;

playerVelocity = CGPointZero;

}

playerSprite.position = pos;

}

这段代码根据之前计算出来的加速计的速度值，加上玩家飞机的x轴方向的速度，同时限制玩家飞机只能在屏幕范围之内飞行。

好了，这时候编译并运行，就可以通过加速计来控制飞机的移动了！

### 添加子弹并射击

目前来说，一切看起来还不错。但是，假如不能开枪打死这些敌机，那么游戏会非常乏味。话不多说，给玩家飞机添加子弹射击功能吧！为了玩家操作的方便性，希望在游戏开始时，只要轻轻单击屏幕就可以自动发射子弹了。之后，玩家只需要专心控制飞机的飞行就可以了。当然，子弹是无限的。

1. 在HelloWorldLayer.h中添加两个成员变量，如代码清单2-19所示。
2. 在HelloWorldLayer.h中添加两个成员变量

BOOL \_isTouchToShoot;

CCSprite \*\_bulletSprite;

1. 在HelloWorldLayer()的@end之前添加一个新方法，如代码清单2-20所示。
2. 在HelloWorldLayer()的@end之前添加新方法

-(void) bulletFinishedMoving:(id)sender;

1. 为了能接收玩家的触摸消息，在init方法返回之前添加代码清单2-21所示代码。
2. 在init方法返回之前添加代码

//9.enable touch

self.isTouchEnabled = YES;

\_isTouchToShoot = NO;

//10.init bullets

\_bulletSprite = [CCSprite spriteWithFile:@"bullet1.png"];

\_bulletSprite.visible = NO;

[self addChild:\_bulletSprite z:4];

其他self.isTouchEnabled设置为YES之后，就可以在CCLayer里处理触摸消息了。

1. 框架自动调用ccTouchesEnded方法，实现代码如代码清单2-22所示。
2. ccTouchesEnded方法的实现。

-(void) ccTouchesEnded:(NSSet \*)touches withEvent:(UIEvent \*)event{

CCLOG(@"touch!");

\_isTouchToShoot = YES;

}

在init方法最后初始化子弹精灵。

注意 这里没有采用敌机那种方式，用数组保存一系列精灵图片。因为这里的设计子弹消失了，才可以发射下一颗子弹，这样可以给游戏添加一些难度和趣味性。

1. 在update方法底部添加发射子弹的方法。

[self updatePlayerShooting:dt]方法的作用就是让玩家飞机不断地发射子弹。具体实现如代码清单2-23所示。

1. 在update方法底部添加[self updatePlayerShooting:dt]方法

-(void) updatePlayerShooting:(ccTime)dt{

if (\_bulletSprite.visible || !\_isTouchToShoot) {

return;

}

CCSprite \*playerSprite = (CCSprite\*)[self getChildByTag:kTagPalyer];

CGPoint pos = playerSprite.position;

CGPoint bulletPos = CGPointMake(pos.x,

pos.y + playerSprite.contentSize.height/ 2 + \_bulletSprite.contentSize.height);

\_bulletSprite.position = bulletPos;

\_bulletSprite.visible = YES;

CGSize winSize = [CCDirector sharedDirector].winSize;

id moveBy = [CCMoveBy actionWithDuration:1.0 position:ccp(0,winSize.height - bulletPos.y)];

id callback = [CCCallFuncN actionWithTarget:self selector:@selector(bulletFinishedMoving:)];

id ac = [CCSequence actions:moveBy,callback, nil];

[\_bulletSprite runAction:ac];

}

-(void) bulletFinishedMoving:(id)sender{

\_bulletSprite.visible = NO;

}

这段代码很简单，主要判断子弹当前是否可见。如果可见，或者玩家还没有触摸屏幕，则直接返回。接下来，就是计算子弹的起始坐标，然后让子弹运行一个action往上飞行就ok了。等子弹到达屏幕顶端时，再把子弹的可见性重新设置为NO，这样下一颗子弹又可以从玩家飞机上空发射出来了。

编译并运行，然后轻轻触摸屏幕，是不是有真实的射击游戏的感觉了呢？

## 碰撞检测

很快就会发现，射出的子弹和敌机都是“擦肩而过”，毫无威力，这显然不合逻辑。这里介绍Cocos2D里常用的碰撞检测方法，用于检测游戏对象之间的碰撞。虽然这种碰撞比较简单，检测精度也不是很高，但是对于一般游戏来说已经足够。

在介绍实际的碰撞检测方法之前，需要一个辅助方法，根据精灵当前的位置算出它的“可碰撞区域”，如代码清单2-24所示。

1. 辅助方法的实现

-(CGRect) rectOfSprite:(CCSprite\*)sprite{

return CGRectMake(sprite.position.x - sprite.contentSize.width / 2,

sprite.position.y - sprite.contentSize.height /2,

sprite.contentSize.width, sprite.contentSize.height);

}

此方法由精灵所在位置和精灵的实际大小计算出一个矩形区域代表精灵的可碰撞区域。

具体的碰撞检测代码如代码清单2-25所示。

1. 碰撞检测

-(void) collisionDetection:(ccTime)dt{

CCSprite \*enemy;

CGRect bulletRect = [self rectOfSprite:\_bulletSprite];

CCARRAY\_FOREACH(\_enemySprites, enemy)

{

if (enemy.visible) {

//1.bullet & enemy collision detection

CGRect enemyRect = [self rectOfSprite:enemy];

if (\_bulletSprite.visible && CGRectIntersectsRect(enemyRect, bulletRect)) {

enemy.visible = NO;

\_bulletSprite.visible = NO;

[\_bulletSprite stopAllActions];

[enemy stopAllActions];

CCLOG(@"collision bullet");

break;

}

//2.enemy & player collision detection

CCSprite \*playerSprite = (CCSprite\*)[self getChildByTag:kTagPalyer];

CGRect playRect = [self rectOfSprite:playerSprite];

if (playerSprite.visible &&

playerSprite.numberOfRunningActions == 0

&& CGRectIntersectsRect(enemyRect, playRect)) {

enemy.visible = NO;

id blink = [CCBlink actionWithDuration:2.0 blinks:4];

[playerSprite stopAllActions];

[playerSprite runAction:blink];

CCLOG(@"collision player");

break;

}

}

}

}

因为在这个游戏中只需要关心子弹与敌机的碰撞、敌机与玩家的碰撞就行了。所以，首先计算出子弹的碰撞区域，然后遍历所有的敌机，然后逐个判断敌机与子弹以及敌机与玩家之间的碰撞。

注意 这里用CGRectIntersectsRect方法检测碰撞，此方法判断两个矩形是否有相交。如果形状不规则，或者原始精灵图片的透明像素过多的情况，这种检测方法会显得非常不准确。后面学习物理引擎时，将介绍一种更加精确的碰撞检测方法。

这里需要注意两点：

* 检测对象必须在屏幕可见范围之内；
* 检测到碰撞以后需要停止action，再做相应的碰撞处理逻辑，比如让敌机消失、计分之类的。

当然，由于此处敌机采用复用的方式，击中以后并不是删除这个对象，而只是把可见性设置为NO，然后放回数组里，下次又可从里再取出来，再次向玩家发起进攻。

编译并运行，体验杀死敌机的快感吧！

## 游戏音效

游戏如果跟无声电影一样，那么肯定会少了很多乐趣。其实要在Cocos2D的游戏世界里添加声音是非常简单的。

打开HelloWorldLayer.m，然后包含#import "SimpleAudioEngine.h"头文件。在init函数返回之前添加代码清单2-26所示代码。

1. 在init函数返回之前添加代码

1．[[SimpleAudioEngine sharedEngine] preloadEffect:@"bullet.mp3"];

2．[[SimpleAudioEngine sharedEngine] playBackgroundMusic:@"game\_music.mp3" loop:YES];

3.[[SimpleAudioEngine sharedEngine] setBackgroundMusicVolume:0.5];

1. 预加载子弹发射时的音效，在发射子弹时，直接可以播放该音效而不至于有一点卡。
2. 播放背景音乐，同时第二个参数是指定背景音乐重复循环播放。
3. 设置背景音量。1表示最大，0.5表示一半。

最后，在updatePlayerShooting方法返回之前，添加播放子弹音效，如代码清单2-27所示。

1. 播放子弹音效

[[SimpleAudioEngine sharedEngine] playEffect:@"bullet.mp3"];

好了，编译并运行，听听精彩的游戏世界吧。

## 最后的点缀

有大师曾经说过，游戏永远没有真正意义上的完结，因为总是可以不断地完善它，给它赋予更多的玩法和元素，使之更充实和完美。本章目的是学习使用Cocos2D制作一个简单的游戏，而制作游戏追求完美的心境是永恒不变的。接下来将进一步完善此游戏。

### 添加计分和玩家生命值

为了刺激玩家玩游戏，此游戏还需要加上计分规则以及玩家生命值的设定。简单起见，每杀死一个敌人计100分，玩家共有3条生命，如果玩家分数超过1000分，即击杀10个敌人就可以胜利过关；反之，如果失去3条生命，则Game Over！

打开HelloWorldLayer.h文件，在头文件中添加两个实例变量，如代码清单2-28所示。

1. 在头文件中添加两个实例变量

CCLabelTTF \*\_lifeLabel;

CCLabelTTF \*\_scoreLabel;

在init函数返回之前添加代码清单2-29所示代码。

1. 在init函数返回之前添加代码

//12.init player lives & score

CCLabelTTF \*lifeIndicator = [CCLabelTTF labelWithString:@"生命值:" fontName:@"Arial" fontSize:20];

lifeIndicator.anchorPoint = ccp(0.0,0.5);

lifeIndicator.position = ccp(20,winSize.height - 20);

[self addChild:lifeIndicator z:10];

\_lifeLabel = [CCLabelTTF labelWithString:@"3" fontName:@"Arial" fontSize:20];

\_lifeLabel.position = ccpAdd(lifeIndicator.position, ccp(lifeIndicator.contentSize.width+10,0));

[self addChild:\_lifeLabel z:10];

CCLabelTTF \*scoreIndicator = [CCLabelTTF labelWithString:@"分数：" fontName:@"Arial" fontSize:20];

scoreIndicator.anchorPoint = ccp(0.0,0.5f);

scoreIndicator.position = ccp(winSize.width - 100,winSize.height - 20);

[self addChild:scoreIndicator z:10];

\_scoreLabel = [CCLabelTTF labelWithString:@"00" fontName:@"Arial" fontSize:20];

\_scoreLabel.position = ccpAdd(scoreIndicator.position, ccp(scoreIndicator.contentSize.width+ 10,0));

[self addChild:\_scoreLabel z:10];

这段代码的主要作用就是在游戏界面上添加计分和玩家生命值的显示。

坐标计算技巧：这里通过修改label的anchorPoint定位label，同时使用ccpAdd宏计算两个点的和。

编译并运行代码，屏幕输出如图2-9所示。

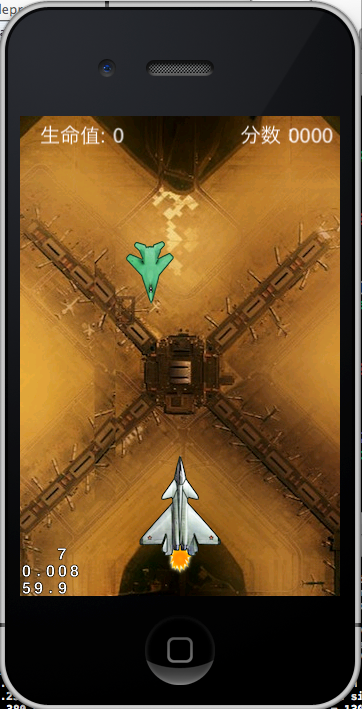


图2-9 添加计分和玩家生命值

按照先前设定的规则，击中一个敌机加100分，被敌机撞到减去1生命值。

1. 打开HelloWorldLayer.h，并往其中添加两个实例变量，如代码清单2-30所示。
2. 添加两个实例变量

int \_totalLives;

int \_totalScore;

1. 在init函数返回之前添加代码进行初始化，如代码清单2-31所示。
2. 在init函数返回之前添加代码

//13.init lives & score variable

\_totalLives = 3;

\_totalScore = 0;

1. 定义一个updateHUD的方法，根据\_totalLives和\_totalScore这两个变量实时更新相应的label方法。该方法实现如代码清单2-32所示。
2. 实时更新相应的label方法

-(void) updateHUD:(ccTime)dt{

[\_lifeLabel setString:[NSString stringWithFormat:@"%2d",\_totalLives]];

[\_scoreLabel setString:[NSString stringWithFormat:@"%04d",\_totalScore]];

}

提示 游戏开发初学者对HUD可能有点陌生，这里有必要解释。HUD就是Head-Up Display的简写，是游戏开发中一个非常重要的概念。比如Mini地图、分数、血条、时间进度条和魔兽世界里的技能条等，这些都是HUD。主要用来给玩家提供一些辅助信息，让玩家可更加关心游戏玩法，而不用去管低层的数据。

1. 更新游戏主循环，添加updateHUD方法调用。如代码清单2-33所示。
2. 游戏主循环

-(void) update:(ccTime)dt{

[self updatePlayerPosition:dt];

[self updatePlayerShooting:dt];

[self collisionDetection:dt];

[self updateHUD:dt];

}

1. 在碰撞检测的逻辑里添加相应的计分和损失生命值的逻辑计算，如代码清单2-34所示。
2. 添加计分和损失生命值的逻辑计算

-(void) collisionDetection:(ccTime)dt{

CCSprite \*enemy;

CGRect bulletRect = [self rectOfSprite:\_bulletSprite];

CCARRAY\_FOREACH(\_enemySprites, enemy)

{

if (enemy.visible) {

//1.bullet & enemy collision detection

CGRect enemyRect = [self rectOfSprite:enemy];

if (\_bulletSprite.visible && CGRectIntersectsRect(enemyRect, bulletRect)) {

enemy.visible = NO;

\_bulletSprite.visible = NO;

\_totalScore += 100;

[\_bulletSprite stopAllActions];

[enemy stopAllActions];

CCLOG(@"collision bullet");

break;

}

//2.enemy & player collision detection

CCSprite \*playerSprite = (CCSprite\*)[self getChildByTag:kTagPalyer];

CGRect playRect = [self rectOfSprite:playerSprite];

if (playerSprite.visible &&

playerSprite.numberOfRunningActions == 0

&& CGRectIntersectsRect(enemyRect, playRect)) {

enemy.visible = NO;

\_totalLives -= 1;

id blink = [CCBlink actionWithDuration:2.0 blinks:4];

[playerSprite stopAllActions];

[playerSprite runAction:blink];

CCLOG(@"collision player");

break;

}

}

}

}

### 添加游戏胜利和结束画面

玩家达到1000分或者生命值变为0时，需要呈现游戏胜利或者失败的画面。简单起见，游戏结束时（不管是胜利还是失败）只在中间显示一个Label，2秒后重新开始游戏。

1. 打开HelloWorldLayer.h，添加一个新的实例变量，如代码清单2-35所示。
2. 添加一个新的实例变量

CCLabelTTF \*\_gameEndLabel;

1. 添加一个私有方法如代码清单2-36所示。
2. 添加一个私有方法

-(void) onRestartGame{

[[CCDirector sharedDirector] replaceScene:[HelloWorldLayer scene]];

}

1. 然后修改碰撞处理的代码，如代码清单2-37所示。
2. 修改碰撞处理

-(void) collisionDetection:(ccTime)dt{

CCSprite \*enemy;

CGRect bulletRect = [self rectOfSprite:\_bulletSprite];

CCARRAY\_FOREACH(\_enemySprites, enemy)

{

if (enemy.visible) {

//1.bullet & enemy collision detection

CGRect enemyRect = [self rectOfSprite:enemy];

if (\_bulletSprite.visible && CGRectIntersectsRect(enemyRect, bulletRect)) {

enemy.visible = NO;

\_bulletSprite.visible = NO;

\_totalScore += 100;

if (\_totalScore >= 1000) {

[\_gameEndLabel setString:@"游戏胜利！"];

\_gameEndLabel.visible = YES;

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

[\_gameEndLabel runAction:scaleTo];

[self unscheduleUpdate];

[self performSelector:@selector(onRestartGame) withObject:nil afterDelay:2.0f];

}

[\_bulletSprite stopAllActions];

[enemy stopAllActions];

CCLOG(@"collision bullet");

break;

}

//2.enemy & player collision detection

CCSprite \*playerSprite = (CCSprite\*)[self getChildByTag:kTagPalyer];

CGRect playRect = [self rectOfSprite:playerSprite];

if (playerSprite.visible &&

playerSprite.numberOfRunningActions == 0

&& CGRectIntersectsRect(enemyRect, playRect)) {

enemy.visible = NO;

\_totalLives -= 1;

if (\_totalLives <= 0) {

[\_gameEndLabel setString:@"游戏失败!"];

\_gameEndLabel.visible = YES;

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

[\_gameEndLabel runAction:scaleTo];

[self unscheduleUpdate];

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

}

id blink = [CCBlink actionWithDuration:2.0 blinks:4];

[playerSprite stopAllActions];

[playerSprite runAction:blink];

CCLOG(@"collision player");

break;

}

}

}

}

代码比较简单，就是当条件满足时更改label显示的内容，然后让label运行一个action，以放大的形式呈现。最后停止游戏主循环，隔2秒调用重启游戏的方法。

提示 这里唯一需要注意[self unscheduleUpdate]方法，就是终止update方法，即终止游戏主循环。

编译并运行，玩一玩这个游戏去吧！

当然，还可以给这个游戏添加更多的乐趣，比如每过一关，敌机的数量越来越多，过关的分数要求越来越高等。本章介绍就到此为止，后面章节还会不断地完善此游戏，使之更加丰富多彩。各位在学习的过程中，也可以充分发挥想象力和创造力，不断地给这个游戏注入新的活力。

## 本章小结

本章以一个垂直射击游戏为主线，循序渐近地介绍该游戏的制作过程和方法。在完成本章后，应该对使用Cocos2D制作游戏有一个大致的了解，比如什么是sprite、label、HUD、game loop，以及如何做碰撞检测（collision detection）。

在之后学习本书的过程中，不仅要关注具体的技术实现细节，更应该基本掌握这些核心概念以及核心类，这样在编写新游戏时，才能够独立思考。

下一章开始介绍Cocos2D中的核心类，这些类几乎每一个游戏都要使用到。通过对这些类的进一步学习，有助于理解和掌握本章的示例。