# 第6章 Cocos2D中的事件处理机制

游戏被称为第九艺术，和绘画、雕塑、工艺、建筑、音乐、舞蹈、文学、戏剧、电影一样可登艺术的大雅之堂。而游戏与传统艺术领域的最大差别在于其极强的交互性。只有具备优秀交互设计的应用或游戏才能让用户获得好的使用体验。因此，一款应用或游戏是否简洁、易上手和精心设计显得至关重要。

本章将学习Cocos2D中和用户交互密切相关的事件处理（Event Handling），即用户输入（User Input）处理。在目前的版本中，iOS设备支持的用户交互事件包括多点触摸、重力感应和陀螺仪。此外，最新的iPhone 4S提供了全所未有的Siri智能语音识别功能，但遗憾的是当前苹果并没有完全开放Siri语音识别和语义识别的API，仅在iOS 5.1 中提供了用于语音识别的少量API。在Cocos2D中，目前使用最多的用户输入是多点触摸（Multi-Touch）和重力感应（Accelerometer）。

## iOS中的交互事件处理

在iOS中，通常将用户输入事件作为对象传递给应用或游戏，事件可能有多种形式：触摸事件（Touch Event）、设备动作事件（Motion Event），以及用于控制多媒体播放的事件（又称为远端控制，即Remote Control，源自耳机或其他外设）。对于不同的用户输入事件，iOS提供了不同的处理机制。

本节将对iOS中的交互事件处理加以详细介绍。

### 什么是用户输入事件

在学习iOS的事件处理机制前，需要了解iOS中用户输入事件的基本类型及其特点。

图6-1是一些基本类型的事件输入。

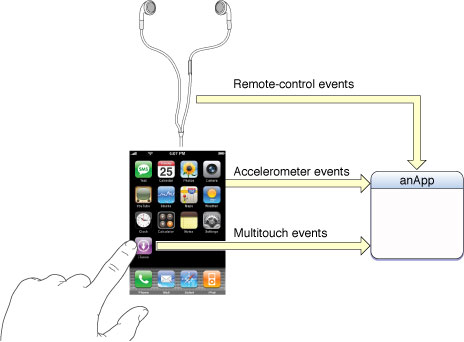


图6-1 iOS中的用户输入事件类型

事实上，任何一部iPhone、iPad或iPod touch设备中都内置了多个硬件元件，可以产生不同类型的用户输入数据。比如多点触摸机制允许用户直接操作视图，包括虚拟键盘；三向加速计（重力感应）可以测量三个不同空间维度的加速度；陀螺仪可以测量三个不同空间维度的旋转速率。GPS（全球定位系统）和罗盘可以提供地址和方位的详细信息。每个硬件子系统在接收到用户输入事件，比如触摸、设备的移动和地理位置的变化时，都会产生相应的源数据，并将这些数据传递给系统对应的框架。比如，分别使用UIKit和Core Motion框架来负责触摸事件和设备移动事件的接管和分发。这些框架将会对源数据进行包装，然后作为事件分发给当前的应用或游戏，从而用于处理。

本章重点学习多点触摸和重力感应的事件处理机制。对于其他类型的事件处理机制，感兴趣的读者可以仔细研读苹果官方提供的iOS事件处理指南（[http://developer.apple.com/library/ ios/#documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/Introduction/Introduction.html](http://developer.apple.com/library/ios/#documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/Introduction/Introduction.html)）。

### 多点触摸事件处理机制

在iOS中，多点触摸事件由UIKit框架来负责事件的分发。除了多点触摸事件，UIKit框架还可以识别设备摇晃事件和远程控制事件。

从严格的定义上来讲，事件其实是一个对象，它代表某个被设备硬件检测到并传递给iOS的用户输入行为，比如用户用手指触摸屏幕或者用手摇晃屏幕。许多事件都是UIKit框架中UIEvent类的实例对象。一个UIEvent对象可能封装了和用户事件相关的状态，比如相关联的触摸。此外，该对象还记录了事件发生的时刻。当某个用户行为发生时，比如当用户的手指触摸屏幕或在屏幕表面移动时，操作系统会持续发送事件对象给当前应用，从而便于处理。

注意 在编写代码时，千万不要retain某个UIEvent事件。如果需要保存某个事件的当前状态以便后续使用，必须采用适当的方法来复制和储存事件的状态（比如使用某个实例变量或词典对象）。

在iOS系统中，基本上所有和UI相关的控件，UIView和UIViewController等都直接或间接继承自UIResponder。事件的分发正是沿着所谓的响应链（Responder Chain）进行的。一个典型的iOS响应链如图6-2所示。

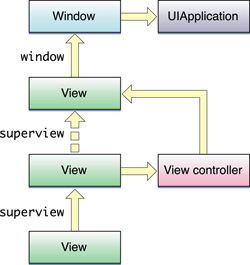


图6-2典型的iOS响应链

将一个事件分发给某个对象进行处理的过程是沿着特殊路径进行的。当用户触摸设备的屏幕时，iOS首先会识别触摸点的集合，然后使用UIEvent对象将这些数据包装，并放置在当前应用的事件列表中。接下来，UIApplication的单例对象会从队列的顶部获取该事件，并进行分发以便处理。通常来说，UIApplication单例对象会将事件分发给应用的主窗口（当前关注用户输入事件的窗口），以及代表将该事件发送给某初始对象进行处理的窗口对象。

对于多点触摸事件，窗口对象会使用点击测试（hit-test）和响应链来确定接收触摸事件的视图。对于点击测试，窗口对象会对视图层次中最顶部的视图调用hitTest:withEvent:方法，而该方法会对视图层次中每个返回YES的视图递归调用pointInside:withEvent方法，并沿着视图层次进行，直到发现包含了触摸点的子视图。该视图将成为hit-test视图，并确定为first responder。如果hit-test视图不能处理当前事件，则会按照响应链回溯，直到系统发现可以处理该事件的视图。在其生命周期内，触摸对象和hit-test视图都会紧密联系在一起，甚至当触摸对象随后移出了视图也是如此。

以上的理论基础大家只需稍微了解一下，下面将重点说明多点触摸事件的处理机制。

为了处理多点触摸事件，首先需要创建一个responder类的子类，该子类可能是：

* 定制的视图（UIView的子类）
* UIViewController的子类
* UIKit视图或控制类的子类（如UIImageView或UISlider）
* UIApplication或UIWindow的子类（很少见）

为了处理多点触摸事件，该视图必须是可见的（不能是透明的或隐藏的），同时userInteractionEnabled属性必须设置为YES（默认情况下会设置为YES）。

为了接收并处理多点触摸事件，响应对象必须实现以下方法中的至少一个：

* - (void)touchesBegan:(NSSet \*)touches withEvent:(UIEvent \*)event;
* - (void)touchesMoved:(NSSet \*)touches withEvent:(UIEvent \*)event;
* - (void)touchesEnded:(NSSet \*)touches withEvent:(UIEvent \*)event;
* - (void)touchesCancelled:(NSSet \*)touches withEvent:(UIEvent \*)event;

以上的每种方法都带有两个参数，其中之一是UITouch对象的集合，代表指定时间段内新增或变化的触摸对象。而第二个参数则是代表了该特定事件的UIEvent对象。

对于UITouch对象，通常会用到以下的方法或属性：

1. **locationInView**：该方法会返回触摸点在指定视图坐标系中的当前位置，返回值是CGPoint类型。该方法也是在Cocos2D中要经常用到的。
2. **previousLocationInView**：该方法会返回触摸点在指定视图坐标系中的上一个位置，返回值是CGPoint类型。
3. **tapCount**：该属性代表当前触摸事件中手指触摸某点的次数，类型是NSUInteger。如果要判断在某指定时间段内用户是单击，双击或是多次触碰视图中的某点，可以考虑使用该属性。
4. **timestamp**：该属性代表触摸事件发生时的时间点，类型是NSTimeInterval。
5. **view**：该属性代表触摸事件初始发生的视图，类型是UIView \*view。
6. **window**：该属性代表触摸事件初始发生的窗口，类型是UIWindow \*window。
7. **phase**：该属性代表触摸的阶段类型，即触摸事件是开始、移动、结束或是取消，类型是UITouchPhase。
8. **gestureRecognizers**：该属性代表接收触摸对象的手势识别。

而对于UIEvent，通常会用到以下的方法或属性：

1. **allTouches**：该方法会返回所有的触摸对象，即和该触摸事件相关的所有触摸点。
2. **touchesForGestureRecognizer**：该方法会返回被发送到特定手势识别的触摸对象。
3. **touchesForView**：返回一个UITouch对象集合，代表和当前触摸事件相关的特定视图中的触摸点。
4. **touchesForWindow**：返回一个UITouch对象集合，代表和当前触摸事件相关的指定窗口中的触摸点。

判断在当前视图中有双触事件发生的方法如代码清单6-1所示。

代码清单6-1 判断在当前视图中有双触事件发生

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

}

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

}

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

for (UITouch \*touch in touches) {

if (touch.tapCount >= 2) {

[self.superview bringSubviewToFront:self];

}

}

}

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

}

简而言之，在接收到用户输入事件后，iOS系统会沿着响应链找到first responder，即响应对象；然后由响应对象实现指定方法中的至少一个，从而实现用户输入事件与应用或游戏的互动。对于开发者来说，最重要的是实现响应对象四个方法中的至少一个。

### 重力感应事件处理机制

重力感应事件又称为加速计事件，属于运动事件的一种。当用户通过某种方式移动设备，比如摇晃或倾斜设备时，就会产生运动事件。所有运动事件都基于设备的加速计或陀螺仪。这里要关心的是基于加速计产生的重力感应事件。

在应用或游戏中，可以使用UIAccelerometer单例对象接收加速运动的数据。使用UIAccelerometer单例对象，可以设置信息更新间隔，并通过定制的delegate代理接收重力感应事件。最小信息更新间隔可以设置为10ms，或者说每秒更新数据100次，实际上大多数的应用或游戏都不需要如此小的间隔时间。一旦分配好代理对象，加速计就会开始传输数据。此时，所设置的代理对象会以指定的更新频率来获取数据。

常用的更新间隔如表6-1所示。

表6-1 事件更新间隔

|  |  |
| --- | --- |
| 事件频率 | 用途 |
| 10~20 | 用于判断设备的当前方向 |
| 30~60 | 适合用于游戏或其他需要实时用户输入的加速计数据的应用 |
| 70~100 | 适合需要检测高频设备运动的应用，如检测用户快速敲击或摇晃设备 |

代码清单6-2展示了配置加速计的基本步骤。

代码清单6-2 配置加速计的基本步骤

#define kAccelerometerFrequency 50.0//Hz

-(void)configureAccelerometer{

UIAccelerometer\* theAccelerometer = [UIAccelerometer sharedAccelerometer];

theAccelerometer.updateInterval = 1/ kAccelerometerFrequency;

theAccelerometer.delegate = self;

//代理对象的事件将立即开始

}

在这个例子中，更新频率设置为50Hz，即更新间隔为20ms。其中myDelegateObject就是所定义的定制代理对象，该对象必须支持UIAccelerometerDelegate协议。

代码清单6-3中，accelerometer单例对象通过accelerometer:didAccelerate方法向代理对象发送数据。可以使用该方法以任何方式处理所获取的加速计数据。

代码清单6-3通过accelerometer:didAccelerate方法向代理对象发送数据

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

UIAccelerationValue x,y,z;

x = acceleration.x;

y = acceleration.y;

z = acceleration.z;

//使用自己的方法来处理这些数据

}

需要注意的是，即便设备处于完全静止的状态，加速计所返回的数据仍会有轻微的波动。此外，通常不会直接使用这些数据，而会先通过某些方法来校准和过滤。

加速计的三个轴向数据如图6-3所示。

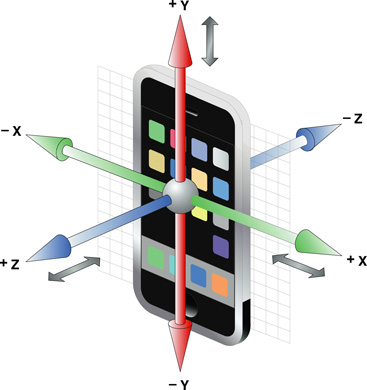


图6-3加速计的三个轴向数据图

提示 如果要停止重力感应事件的发送，只需将UIAccelerometer单例对象设置为nil即可。

当使用加速计数据来检测设备的当前方向时，需要从原始数据中筛选出仅受重力影响的部分。当然，可以直接使用苹果推荐的低通滤波算法来实现这一点。具体的代码如代码清单6-4所示。

代码清单6-4 低通滤波算法

#define kFilteringFactor 0.1

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

// 使用低通滤波算法来筛选出每个轴向受重力影响的成分

UIAccelerationValue accelX, accelY,accelZ;

accelX = (acceleration.x \* kFilteringFactor) +(accelX \* (1.0 - kFilteringFactor));

accelY = (acceleration.y \* kFilteringFactor) +(accelY \* (1.0 - kFilteringFactor));

accelZ = (acceleration.z \* kFilteringFactor) +(accelZ \* (1.0 - kFilteringFactor));

//使用过滤后的数据

}

当使用加速计数据来检测设备的实时运动时，就需要把突然的运动变化数据从受重力影响的部分中独立出来，此时需要使用高通滤波算法。具体的代码如代码清单6-5所示。

代码清单6-5高通滤波算法

#define kFilteringFactor 0.1

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

//将低通数据从当前数值中过滤掉，以获取简化后的高通数据

UIAccelerationValue accelX,accelY,accelZ;

accelX = acceleration.x - ((acceleration.x \* kFilteringFactor) + (accelX \* (1.0 - kFilteringFactor)));

accelY = acceleration.y - ((acceleration.y \* kFilteringFactor) + (accelY \* (1.0 - kFilteringFactor)));

accelZ = acceleration.z - ((acceleration.z \* kFilteringFactor) + (accelZ \* (1.0 - kFilteringFactor)));

//使用过滤后的加速计数据

}

除了使用UIAccelerometer类，iOS还提供了Core Motion框架来处理重力感应事件，即使用CMAcceleration。但在Cocos2D中，通常使用UIAccelerometer类即可。

## Cocos2D中的触摸事件处理

了解了iOS的触摸事件处理机制，就可以学习如何在Cocos2D中对触摸事件进行处理。

1. 如何处理触摸事件

第3章曾提到CCLayer的主要作用是接收用户输入事件，因此Cocos2D的触摸事件处理机制重点在于如何将用户输入事件发送给CCLayer对象。

在Cocos2D中，这一工作是由CCTouchDispatcher核心类实现。整个触摸事件处理机制涉及CCTouchDelegateProtocol.h、CCTouchDispatcher类和CCTouchHandler类，以及负责处理OpenGL的CCGLView（v2.0 版本前的名称是EAGLView）类。

Cocos2D使用以下三个步骤实现用户触摸输入的响应。

1. 接管：从iOS的标准UIView中获取触摸输入事件。
2. 分发：按照预设逻辑分发给代理对象。
3. 处理：使用代理对象响应用户触摸输入事件。

对于开发者来说，无需了解Cocos2D是如何从iOS标准视图获取触摸输入事件的，但需要大概了解事件是如何分发和处理的。

Cocos2D支持两种不同的触摸事件分发和处理方式，并由两种不同类型的代理协议来实现，而这两种代理协议都在CCTouchDelegateProtocol.h中定义。在Xcode中切换到该文件，并查看其中的详细内容，如图6-4所示。

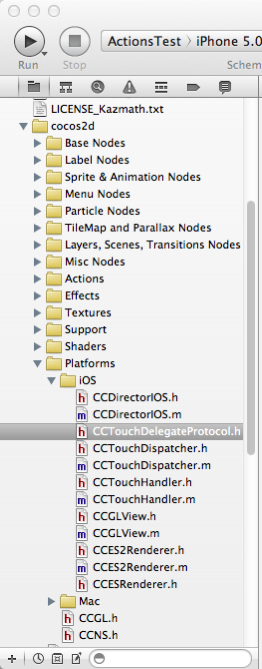


图6-4 在Xcode中查看CCTouchDelegateProtocol.h的内容

#### 标准代理对象处理协议

首先是Standard Touch Delegate（标准代理对象处理协议），其定义如代码清单6-6所示。

代码清单6-6 Standard Touch Delegate的定义

@protocol CCStandardTouchDelegate <NSObject>

@optional

- (void)ccTouchesBegan:(NSSet \*)touches withEvent:(UIEvent \*)event;

- (void)ccTouchesMoved:(NSSet \*)touches withEvent:(UIEvent \*)event;

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

- (void)ccTouchesCancelled:(NSSet \*)touches withEvent:(UIEvent \*)event;

@end

如果仔细比较会发现，以上方法和标准iOS触摸事件处理方法完全相同（除了在每个方法名称的前面添加cc，即代表属于Cocos2D中的方法）。通过以上方法，可以获取所有事件和触摸点，然后需要自行指定需要使用的触摸事件。

为了在CCLayer中获取这些事件，只需在程序中设置isTouchEnabled = YES，如下所示：

self.isTouchEnabled = YES;

标准代理对象处理协议没有进一步的处理要求，外部传入的NSSet，event参数会直接传递给协议的实现对象。

#### 目标代理对象处理协议

除了标准代理对象处理协议，还有一种CCTargetedTouchDelegate（目标代理对象处理协议），其定义如代码清单6-7所示。

代码清单6-7 CCTargetedTouchDelegate定义

@protocol CCTargetedTouchDelegate <NSObject>

/\*\* Return YES to claim the touch.

@since v0.8

\*/

- (BOOL)ccTouchBegan:(UITouch \*)touch withEvent:(UIEvent \*)event;

@optional

// touch updates:

- (void)ccTouchMoved:(UITouch \*)touch withEvent:(UIEvent \*)event;

- (void)ccTouchEnded:(UITouch \*)touch withEvent:(UIEvent \*)event;

- (void)ccTouchCancelled:(UITouch \*)touch withEvent:(UIEvent \*)event;

@end

目标代理对象处理协议和标准代理对象处理协议的区别在于：

* 目标代理对象处理协议只提供单一触摸点，而非触摸点集合。因此这里使用ccTouch而非ccTouches。
* ccTouchBegan方法必须实现，并返回一个Bool值。

因此，对每个可用触摸事件都会触发ccTouchBegan方法，而对于触摸事件需要返回YES。只有在ccTouchBegan方法中声明的触摸点才会传递给ccTouchMoved、ccTouchEnded和ccTouchCancelled（可选方法）。

为了接收这些事件，还必须使用全局dispatcher注册为一个目标代理对象。在CCLayer中，需要使用如下方式来覆盖，如代码清单6-8所示。

代码清单6-8 覆盖目标代理对象

-(void)registerWithTouchDispatcher{

[[director touchDispatcher]addTargetedDelegate:self priority:0 swallowsTouches:YES];

}

当然，在此之前也需要在文件的顶部添加以下一行代码：

#import "CCTouchDispatcher.h"

那么，在实际开发中，应该使用哪种代理协议呢？

通常情况下，目标代理对象协议使用起来更方便，因为无需手动分离NSSet集合，也无需反复检查是否该事件是程序所需的。所以，在一些简单的应用或游戏中，使用目标代理对象协议更方便。

但如果需要在应用或游戏中用到多点触摸（比如缩放或旋转），那么使用标准代理对象协议就是必须的。

注意 在程序中只能选择使用其中的一种代理协议。

当然，为了获取多点触摸事件，必须在程序中使用代码激活。具体方法是在AppDelegate.m的applicationDidFinishLaunching方法中添加一行代码如下所示：

[glView setMultipleTouchEnabled:YES];

#### 其他类的处理方法

最后还有一个问题，以上处理机制是针对CCLayer及其子类（比如CCMenu），因此只需设置isTouchEnabled属性和registerWithTouchDispatcher方法即可。对于其他类该如何实现呢？

* 该类必须实现某种代理协议，即CCStandardTouchDelegate或CCTargetedTouchDelegate协议。
* 该类必须使用CCTouchDispatcher的对象在onEnter方法中将其自身注册为代理，并在onExit方法中取消注册。
* 从v2.0 开始，CCTouchDispatcher和CCEventDispatcher不再支持作为单例使用，而是作为CCDirector类的实例变量使用。

对于标准代理对象协议，需要添加的方法如代码清单6-9所示。

代码清单6-9 标准代理对象协议添加的方法

-(void)onEnter{

[[director touchDispatcher]addStandardDelegate:self priority:0];

[super onEnter];

}

-(void)onExit{

[[director touchDispatcher]removeDelegate:self];

[super onExit];

}

而对于目标代理对象协议，需要添加的方法如代码清单6-10所示。

代码清单6-10目标代理对象协议需要添加的方法

-(void)onEnter{

CCDirector \*director = [CCDirector sharedDirector];

[[director touchDispatcher] addTargetedDelegate:self priority:0 swallowsTouches:YES]; [super onEnter];

}

-(void)onExit{

CCDirector \*director = [CCDirector sharedDirector];

[[director touchDispatcher] removeDelegate:self];

[super onExit];

}

以上就是Cocos2D中处理用户触摸事件的实现机制。如果从理论的角度来看，似乎非常复杂，甚至令人望而生畏。但在实际的使用中，会发现其实比想象的要容易！

1. 模板中自带的TouchesTest项目

上一节详细介绍了Cocos2D对触摸事件的接管、分发和处理机制。很多读者可能被如此复杂的理论吓到了。其实对于开发者来说，需要做的事情并不多，只需实现所指定的方法，就可以轻松地处理用户输入方法。

Cocos2D模板自带一个TouchesTest项目，在Xcode中打开cocos2d-ios.xcodeproj，并选择TouchesTest，如图6-5所示。

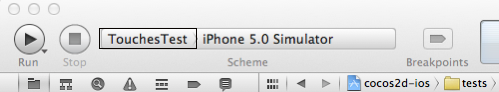


图6-5在Xcode中选择TouchesTest

此时单击左侧的Run按钮，可以在模拟器中看到一个简单的乒乓球小游戏。用鼠标拖动屏幕中的白色球拍，可以控制乒乓球的移动方向。如果在设备中测试，则可以直接用手指触摸进行操控。如图6-6所示。

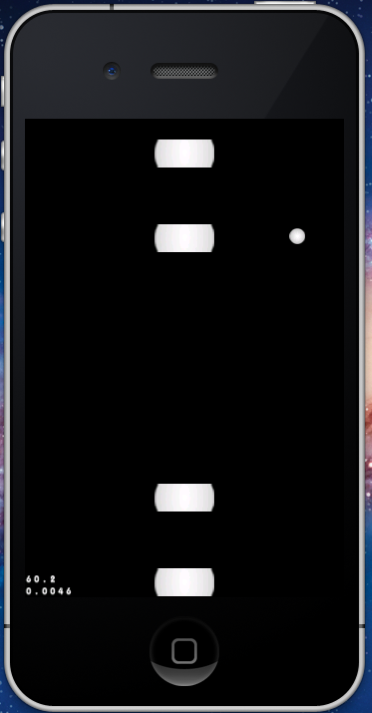


图6-6在模拟器中使用鼠标拖动球拍

要了解TouchesTest是如何实现，可以在Xcode中查看该项目的相关代码，如图6-7所示。

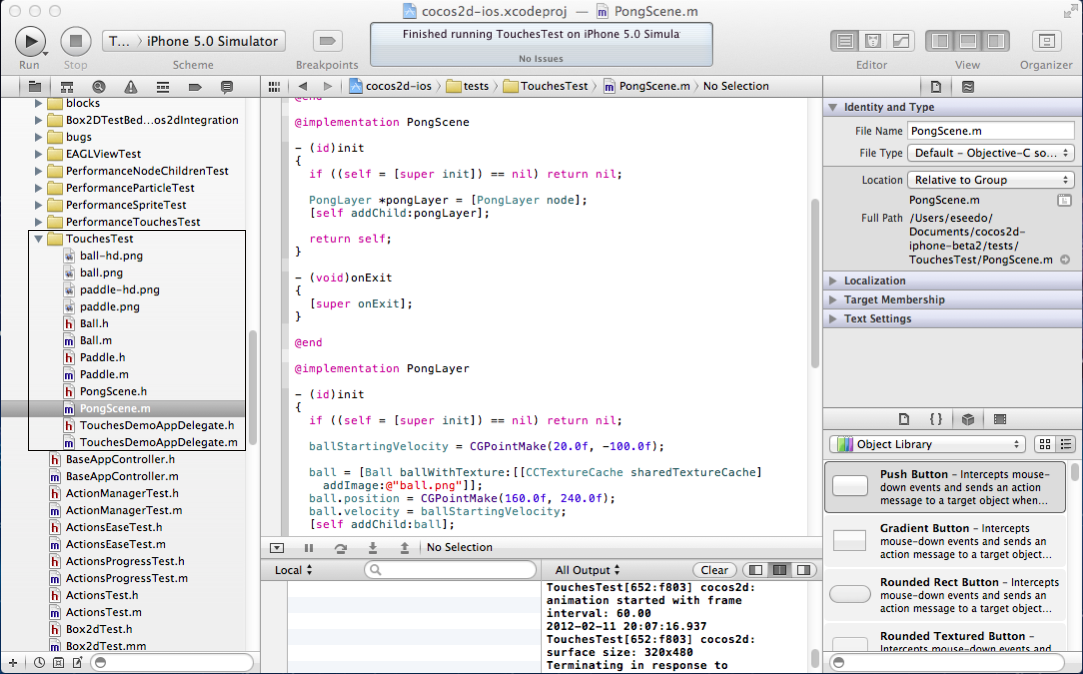


图6-7在Xcode中查看TouchesTest的项目代码

1. 实战：随手指移动的小球

在实际开发中要掌握如何处理用户输入事件，可以仔细研究TouchesTest项目的相关代码，本节提供一个示例项目TouchEvents详细说明。步骤如下：

1. 在Xcode中创建一个新的项目。

选择iOS下的cocos2d-cocos2d标准模板，并将项目命名为TouchEvents。

1. 加载小球图片。

从资源文件夹（chapter6/resource/arts/TouchEvents）中找到ball.png，并将其拖动到Xcode的Resources中，确保选中“Copy items into destination group’s folder(if needed)”。

1. 创建小球精灵的实例变量。

在Xcode中切换到HelloWorldLayer.h，并在类的声明部分添加以下代码：

@interface HelloWorldLayer : CCLayer<GKAchievementViewControllerDelegate, GKLeaderboardViewControllerDelegate>

{

CCSprite \*ball;

}

1. 删除模板中init方法的默认代码。

在Xcode中切换到HelloWorldLayer.m，并在init方法中注释或删除模板中自带的代码。

1. 在init方法中添加新的代码，从而在屏幕中添加精灵对象。如代码清单6-11所示。
2. 在屏幕中添加精灵对象

ball = [CCSprite spriteWithFile:@"ball.png"];

CGSize size = [CCDirector sharedDirector].winSize;

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

[self addChild:ball];

self.isTouchEnabled = YES;

1. 添加使用目标代理协议的相关方法，如代码清单6-12所示。

代码清单6-12添加使用目标代理协议的相关方法

-(void)onEnter

{

CCDirector \*director = [CCDirector sharedDirector];

[[director touchDispatcher] addTargetedDelegate:self priority:0 swallowsTouches:YES];

[super onEnter];

}

- (void)onExit

{

CCDirector \*director = [CCDirector sharedDirector];

[[director touchDispatcher] removeDelegate:self];

[super onExit];

}

-(BOOL)ccTouchBegan:(UITouch \*)touch withEvent:(UIEvent \*)event{

return YES;

}

-(void)ccTouchEnded:(UITouch \*)touch withEvent:(UIEvent \*)event{

//此处将添加代码

}

以上添加了使用目标代理协议处理触摸事件的相关方法，以便让Cocos2D可以通过实现目标代理协议来处理触摸事件。将当前层设置为可以接受触摸事件。

因为在这个游戏中不需要识别多点触摸动作，所以选择了目标代理对象协议。因此要添加onEnter方法、onExit方法和ccTouchBegan方法（必须）。

1. 在ccTouchEnded方法中添加代码以控制小球的运动。

以下对该方法进行详细的解释。

1. 判断触摸点的位置。如下所示：

CGPoint touchLocation = [touch locationInView:[touch view]];

touchLocation = [[CCDirector sharedDirector]convertToGL:touchLocation];

touchLocation = [self convertToNodeSpace:touchLocation];

在以上代码中，第一行用于从触摸点所在的视图（UIView）中的位置。第二行代码则将使用CCDirector单例对象把触摸点在视图中的位置转换为Cocos2D中所使用的OpenGL坐标。而第三行代码则将坐标最终转换成当前节点空间中的坐标。

1. 设置小球的运动速度。

预计小球将在3秒内移动屏幕的宽度（480像素）。如下所示：

float velocity = 480.0/3.0;

1. 计算小球在x轴和y轴的行进距离。

要计算出小球在x轴和y轴分别需要移动多少距离，可以使用ccpSub方法计算出触摸点和小球所在位置的x与y坐标差值。如下所示：

CGPoint moveDifference = ccpSub(touchLocation, panda.position);

1. 计算出小球的实际移动距离。

即沿着直角三角形斜边的行进距离，在Cocos2D中，可以使用ccpLength方法来计算出这一距离。如下所示：

float moveDistance = ccpLength(moveDifference);

1. 计算小球运动的时长。

用距离除以速度得到运动的时长。如下所示：

float moveDuration = moveDistance / velocity;

1. 在需要的情况下让小球转向。

这里添加了一行条件判断语句。如果触摸点和小球所在位置的x坐标差小于0，说明小球在朝左边移动，就不需要翻转，反之就需要掉头！如下所示：

if(moveDifference.x < 0){

ball.flipX = NO;

}else{

ball.flipX = YES;

}

1. 让小球朝触摸点移动。

在代码中定义一个移动动作，并让小球执行这个动作。如下所示：

id moveAction = [CCMoveTo actionWithDuration:moveDuration position:touchLocation];

[ball runAction:moveAction];

1. 编译运行项目。

现在可以使用鼠标（模拟器）或手指触摸（设备）控制小球的运动，如图6-8所示。

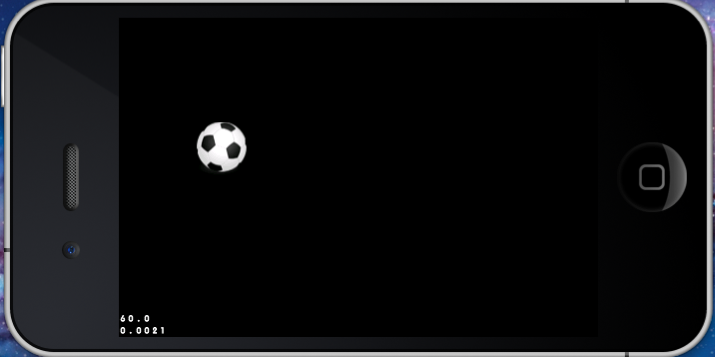


图6-8使用鼠标或手指触摸控制小球的运动

## Cocos2D中的重力感应事件

在Cocos2D中，处理重力感应事件的方式和iOS中的标准处理方式没有任何区别。因此，在详细了解了iOS中的重力感应事件处理机制后，就很容易理解在Cocos2D中如何处理重力感应事件。

本节提供一个示例项目，帮助开发者学习如何在实际项目中处理重力感应事件。

1. 如何处理重力感应事件

在Cocos2D中，处理重力感应事件的方式和iOS中的标准处理方法是相同的。读者可以参考6.1.3节的内容。

而对于开发者来说，在实际的应用或游戏开发中，只需完成以下两个步骤就可以。

1. 在init方法中添加一行代码。如下所示：

self.isAccelerometerEnabled = YES;

1. 实现accelerometer方法。如下所示：

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

}

1. 实战：受加速计控制的小球

为了让读者更好地了解在应用或游戏中响应加速计事件，本节通过AccelEvents实例详细说明。

1. 创建一个新项目。

打开Xcode，选择iOS下的cocos2d-cocos2d标准模板，并将项目命名为AccelEvents。

1. 加载图片文件。

从准备好的资源文件夹（chapter6/resource/arts/AccelEvents）中找到ball.png，并将其拖动到Xcode的Resources中，确保选中“Copy items into destination group’s folder(if needed)”。

1. 创建小球精灵。

切换到HelloWorldLayer.h文件，并在类的声明部分添加以下代码：

@interface HelloWorldLayer : CCLayer <GKAchievementViewControllerDelegate, GKLeaderboardViewControllerDelegate>

{

CCSprite \*ball;

}

1. 让当前层对加速计事件进行响应。

切换到HelloWorldLayer.m文件，对代码做出以下修改。

1. 在init方法中删除原有的代码，如代码清单6-13所示。

代码清单6-13 在init方法中删除原有的代码

-(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. 在init方法的if循环中添加代码，以创建精灵对象，并设置当前层对加速计事件进行响应。如代码清单6-14所示。

代码清单6-14在init方法中创建精灵对象

ball =[CCSprite spriteWithFile:@"ball.png"];

CGSize size = [CCDirector sharedDirector].winSize;

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

[self addChild:ball];

self.isAccelerometerEnabled = YES;

以上步骤创建了一个小球精灵，并将其添加为当前层的子节点。同时还设置了让当前层对加速计事件进行响应。

1. 添加方法使用加速计来控制小球的运动。

这里采用最简单的方式来更新小球的位置。如代码清单6-15所示。

代码清单6-15添加方法以使用加速计来控制小球的运动

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

CGPoint pos = ball.position;

pos.x += acceleration.x \*10;

pos.y += acceleration.y \*10;

ball.position = pos;

//此处将添加更多代码

}

需要注意，这里不能直接对ball.position.x赋值。因为在Objective-C中，ball.position实际上调用了[ball position]方法，而该方法会返回一个临时CGPoint变量。如果对临时CGPoint变量的x进行赋值，该变量会被抛弃，因而赋值没有任何作用。所以必须创建一个新CGPoint变量，修改其x值，并将该变量赋值给ball.position（调用[ball setPosition:]）。

提示 测试和重力感应相关的项目时，必须在设备上进行，因为模拟器不支持重力感应事件。

编译运行项目，会发现加速计控制很不灵敏，而且移动也非常不顺畅。为此，需要增加一些代码。

1. 过滤加速度传入的数值。

切换到HelloWorldLayer.h文件，在类的声明中添加代码如代码清单6-16所示。

代码清单6-16在类的声明中添加代码

@interface HelloWorldLayer : CCLayer<GKAchievementViewControllerDelegate, GKLeaderboardViewControllerDelegate>

{

CCSprite \*ball;

float velocityX;

float velocityY;

}

1. 修改accelerometer方法以过滤加速计传入的数值。

找到accelerometer方法，注释掉之前的代码，并修改其实现方法如代码清单6-17所示。

代码清单6-17修改accelerometer的实现方法以过滤相关数值

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

//最简单的使用加速计数据的方法，但不灵敏

// CGPoint pos = ball.position;

// pos.x += acceleration.x \*10;

// pos.y += acceleration.y \*10;

// ball.position = pos;

//此处将添加更多代码

#define kFilteringFactor 0.1

#define kRestAccelY -0.6

#define kMaxDiffX 0.2

#define kMaxDiffY 0.2

#define kMaxPointsPerSecX (winSize.width\*0.5)

#define kMaxPointsPerSecY (winSize.height\*0.5)

UIAccelerationValue rollingX,rollingY,rollingZ;

rollingX = (acceleration.x \* kFilteringFactor) +(rollingX \*(1.0 - kFilteringFactor));

rollingY = (acceleration.y \* kFilteringFactor) +(rollingY \*(1.0 - kFilteringFactor));

// rollingZ = (acceleration.z \* kFilteringFactor) +(rollingZ \*(1.0 - kFilteringFactor));

float accelX = acceleration.x - rollingX;

float accelY = acceleration.y - rollingY;

// float accelZ = acceleration.z - rollingZ;

CGSize winSize = [CCDirector sharedDirector].winSize;

float accelFractionX = accelX / kMaxDiffX;

float pointsPerSecX = kMaxPointsPerSecX \* accelFractionX;

float accelDiffY = accelY - kRestAccelY;

float accelFractionY = accelDiffY /kMaxDiffY;

float pointsPerSecY = kMaxPointsPerSecY \* accelFractionY;

velocityX = pointsPerSecX;

velocityY = pointsPerSecY;

}

以上代码中，绝大多数内容来自苹果官方的示例代码。其作用是过滤加速度传入的数值。如果看不明白也没关系，只需要知道这段代码的作用是让加速计事件的响应更加平滑。至于具体的数值设置则无需了解太多，在今后的开发中，只需直接复制重用这段代码。在以上方法的最后部分，将获取的小球运动速度赋值给velocityX（横向移动速度）和velocityY（纵向移动速度）。

1. 修改update更新方法。

仅有速度还不够，还需要在update更新方法中添加代码如代码清单6-18所示。

代码清单6-18在update更新方法中添加代码

-(void)update:(ccTime)delta{

CGSize winSize = [CCDirector sharedDirector].winSize;

//设置小球在横向和纵向的移动边界

float maxY = winSize.height -ball.contentSize.height/2;

float minY = ball.contentSize.height/2;

float maxX = winSize.width - ball.contentSize.width/2;

float minX = ball.contentSize.width/2;

//根据加速度数据计算出小球的最终y坐标，注意使用了一个简单的算法来确保小球不会超出边界

float newY = ball.position.代码清单6 + (velocityY \*delta);

newY = MIN(MAX(newY, minY),maxY);

//根据加速度数据计算出小球的最终x坐标，注意，使用了一个简单的算法来确保小球不会超出边界

float newX = ball.position.x + (velocityX \*delta);

newX = MIN(MAX(newX, minX),maxX);

//更新小球的位置

ball.position = ccp(newX,newY);

}

以上代码的作用很简单，根据小球每秒沿x轴和y轴的运动距离，以及上次更新至今的delta时间来设置小球的最终位置。

1. 编译运行项目。

现在，可以通过倾斜设备来控制小球在屏幕中滚动，如图6-9所示。



图6-9设备中的小球受重力影响在屏幕中滚动

注意 对于支持重力感应事件的项目，请一定要在设备上进行测试，通常在模拟器上测试是没有任何反应的。

## 垂直射击游戏：暂停游戏

学习了Cocos2D的触摸事件处理和重力感应事件处理，此时回过头看前面几章的例子，对于控制飞机飞行和触发飞机发射子弹的代码片断，理解起来就容易多了。可以更改控制飞机飞行的方式，不使用重力感应，采用类似iFlight方式，用手指按住飞机然后在屏幕上面滑动来控制飞机飞行。

本节介绍如何为游戏制作一个暂时界面，玩家按下暂停按钮之后，游戏逻辑和游戏动画会暂停，玩家此时只能单击暂停画面上面的“重新开始”或者“继续游戏”按钮，暂停界面下层的事件要屏蔽掉。

### PauseLayer类的实现

首先，右键单击VerticalShootingGame分组，选择File→New→File，选择cocos2d v2.x模板，新建一个PauseLayer类，选择CCLayer作为超类，然后单击Create。

用下面的代码替换PauseLayer.h头文件，如代码清单6-19所示。

代码清单6-19替换PauseLayer.h头文件代码

#import <Foundation/Foundation.h>

#import "cocos2d.h"

//1.

@class PauseLayer;

//2.

@protocol PauseLayerProtocol <NSObject>

@required

-(void) didRecieveResumeEvent:(PauseLayer\*)layer;

-(void) didRecieveRestartEvent:(PauseLayer\*)layer;

@end

//3.

@interface PauseLayer : CCLayerColor<CCTargetedTouchDelegate> {

id<PauseLayerProtocol> delegate;

}

@property(nonatomic,assign)id<PauseLayerProtocol> delegate;

@end

//4.

@interface CustomMenu : CCMenu{

}

@end

按照代码注释中的序号逐一说明：

1. 前置声明。

当两个类定义在同一个文件中，先定义的类（PauseLayerProtocol）要使用后定义的类（PauseLayer）时，就需要使用@class做相应的前置声明。

1. 定义暂停层协议。

这个协议（protocol）就是Java里的接口，C++里的含有纯虚函数的抽象类。它的作用就是定义暂停层与游戏层的交互接口，是两个层之间相互交换数据的一种方式。其中定义两个方法，一个是“继续游戏”按钮按下去时要调用的，还有一个是“重新开始”按钮被按下去时调用的。

1. 定义暂停层（PauseLayer）类。

此类实现了标准触摸事件代理协议，同时在内部定义了暂停层协议（PauseLayerProtocol）代理。之后的代码示例中，会展示此代理的用法。

1. 定义一个定制的菜单类。

为什么需要一个这样的类呢。首先touch事件分发是有优先级的，由touchDispatcher在添加分发代理对象时的参数priority决定。priority越低，优先级越高。而系统自带菜单的优先级是由kCCMenuHandlerPriority决定的，它是一个枚举常量，值是-128。而暂停层需要屏幕下面游戏层的touch事件，包括菜单事件。那么暂停层的priority必须小于kCCMenuHandlerPriority。这里使用kCCMenuHandlerPriority-1，而暂停层上需要一个按钮。此时，按钮的prority是kCCMenuHandlerPriority，显示也被屏蔽了。所以需要定制一个菜单，把它的优先级设置为小于kCCMenuHandlerPriority-1.后面大家会看到具体的做法。

PauseLayer.m类的实现如代码清单6-20所示。

代码清单6-20 PauseLayer.m类的实现代码

#import "PauseLayer.h"

@interface PauseLayer()

-(void) doRestart;

-(void) doResume;

@end

@implementation PauseLayer

@synthesize delegate;

-(id) init{

if ((self = [super initWithColor:ccc4(128, 128, 128, 128) width:320 height:480])) {

CCMenuItem \*pauseItem = [CCMenuItemFont itemWithString:@"重新开始"

target:self

selector:@selector(doRestart)];

pauseItem.position = ccp(160,200);

CCMenuItem \*resumeItem = [CCMenuItemFont itemWithString:@"继续游戏"

target:self

selector:@selector(doResume)];

resumeItem.position = ccp(160,300);

CustomMenu \*menu = [CustomMenu menuWithItems:pauseItem,resumeItem, nil];

menu.position = CGPointZero;

[self addChild:menu];

self.isTouchEnabled = YES;

}

return self;

}

-(void) onEnter{

[super onEnter];

[[[CCDirector sharedDirector] touchDispatcher] addTargetedDelegate:self

priority:kCCMenuHandlerPriority - 1

swallowsTouches:YES];

}

-(void) onExit{

[[[CCDirector sharedDirector] touchDispatcher] removeDelegate:self];

[super onExit];

}

-(BOOL) ccTouchBegan:(UITouch \*)touch withEvent:(UIEvent \*)event{

return YES;

}

#pragma mark - menu callback

-(void) doRestart{

if ([delegate respondsToSelector:@selector(didRecieveRestartEvent:)]) {

[self.delegate didRecieveRestartEvent:self];

}

}

-(void) doResume{

if ([delegate respondsToSelector:@selector(didRecieveResumeEvent:)]) {

[self.delegate didRecieveResumeEvent:self];

}

}

@end

上面这段不难理解：

1. 首先是初始化。init方法里调用initWithColor初始化一个灰色的半透明层。
2. 创建一个包含“继续游戏”和“重新开始”字样的菜单。

此处使用CustomMenu，后面会看到CustomMenu的具体实现。因为实现了CCTargetedTouchDelegate协议，所以需要在init方法里开始touch，即调用self.isTouchEnabled = YES方法。

1. 重写onEnter和onExit方法。这里需要注意priority和swallowTouch的参数。
2. 简单让ccTouchBegan返回YES，这一步是必须的。
3. 实现两个菜单项的回调函数。

提示 这里有个技巧，在调用代理类的方法时，首先添加一个respondsToSelector判断，这样可以提供程序的鲁棒性。

### CustomMenu类的实现

CustomMenu的实现如代码清单6-21所示。

代码清单6-21 CustomMenu的实现

@implementation CustomMenu

-(void) registerWithTouchDispatcher{

[[[CCDirector sharedDirector] touchDispatcher] addTargetedDelegate:self priority:kCCMenuHandlerPriority - 2 swallowsTouches:YES];

}

@end

这个类的实现就更简单了，需要重写registerWithTouchDispatcher方法，同时把优先级设置成kCCMenuHandlerPriority – 2就行了。

### 游戏主场景添加暂停层

本节介绍游戏主场景如何添加暂停层，以及它们两个层之间如何交互。

1. 打开HelloWorldLayer.h，添加下列头文件 。

#import "PauseLayer.h"

1. 把HelloWorldLayer的头文件声明改成如下代码 。

@interface HelloWorldLayer : CCLayer<PauseLayerProtocol>

并在@interface最后添加一个实例变量BOOL \_isGamePause。该变量的主要用途就是标记游戏是否处于暂停状态。初始化为NO，当玩家按下暂停键以后就变为YES，如果再按下resume或者restart的话，又重新设置为NO。

1. 打开HelloWorldLayer.m文件，在init方法返回之前添加代码清单6-22所示代码。

代码清单6-22在init方法返回之前添加代码

//18.添加暂停按钮和暂停画面

CCMenuItem \*pauseItem = [CCMenuItemFont itemWithString:@"暂停"

block:^(id sender){

//add pause layer

PauseLayer \*pl = [PauseLayer node];

pl.position = CGPointZero;

[self addChild:pl z:100];

pl.delegate = self;

//pause game logic & animation

[[CCDirector sharedDirector] pause];

\_isGamePause = YES;

}];

pauseItem.position = ccp(winSize.width - 60,winSize.height - 100);

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

menu.position = CGPointZero;

[self addChild:menu z:10];

\_isGamePause = NO;

这段代码的作用就是定义一个“暂停”菜单，当用户按下此菜单项时，会在游戏场景里初始化一个PauseLayer，同时把自身当作PauseLayer的代理类进行赋值。同时调用[[CCDirector sharedDirector] pause]暂停游戏逻辑和动画。

注意 pause方法只暂停动画和schedulers，并不会真正暂停所有的游戏逻辑。比如之前游戏示例中的spawnEnemy方法就不会暂停，所以需要把isGamePause设置为YES，这样在后面的spawnEnemy逻辑中就可以区分暂停和非暂停状态了。

1. 找到spawnEnemy方法，在方法最上方添加如下代码。

if (\_isGamePause) {

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

return;

}

这段代码的作用就是简单地重复循环，但是当游戏真正开始时并不影响游戏性能。

### PauseLayerProtocol代理

PauseLayerProtocol几个代理方法的实现如代码清单6-23所示。

代码清单6-23 PauseLayerProtocol的代理方法

#pragma mark - pauseLayer protocol

-(void) didRecieveResumeEvent:(PauseLayer \*)layer{

[[CCDirector sharedDirector] resume];

\_isGamePause = NO;

[self removeChild:layer cleanup:YES];

}

-(void) didRecieveRestartEvent:(PauseLayer \*)layer{

[[CCDirector sharedDirector] resume];

[self removeChild:layer cleanup:YES];

[self onRestartGame];

}

这两个代理都调用了[self removeChild:layer cleanup:YES]方法清除弹出的暂停层。同时调用CCDirector的一些方法恢复和重新开始游戏。

现在编译并运行，在游戏开始之后按下暂停按钮，输出所如图6-10所示。



图6-10按下“暂停”按钮的画面

## 本章小结

本章首先介绍iOS系统中的各种用户输入事件（用户交互），了解了iOS系统中用户输入事件的处理机制；然后学习Cocos2D对触摸事件的接管、分发和处理机制，使用一个实例帮助读者学习；最后学习iOS系统和Cocos2D对重力感应事件的处理机制，使用一个实例帮助读者认识在实际开发中如何处理重力感应事件。

通过本章学习，读者应该对Cocos2D的事件处理机制有了整体的概念，并且知道如何在实际项目开发中处理用户输入事件。

第7章将带领大家认识如何在Cocos2D中添加和使用音效。