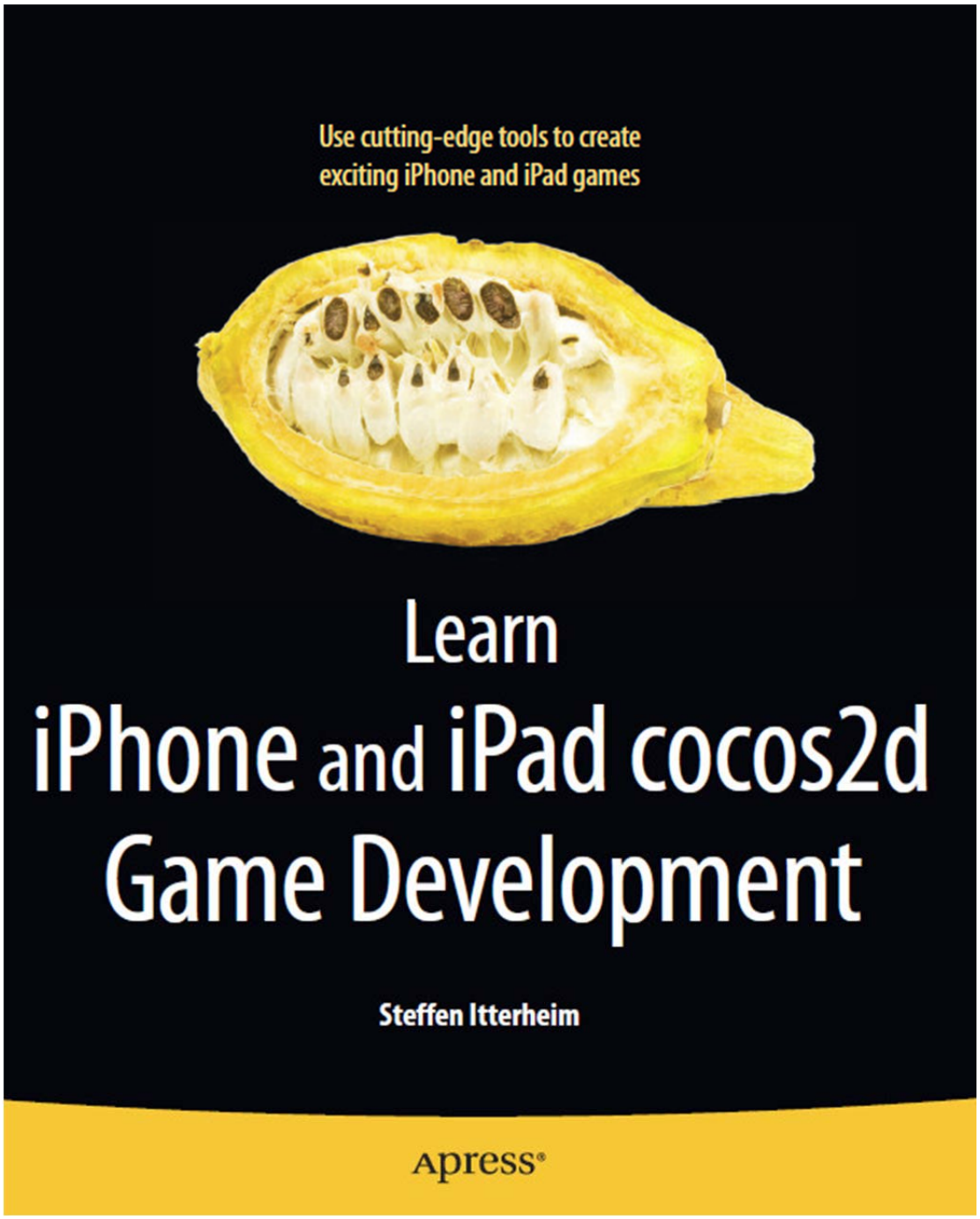

译者：杨栋
邮箱：yangdongmy@gmail.com

第四章
你的第一个游戏

本章将会制作你的第一个游戏。它不会为你带来任何奖项，但是你将在这里学到如何让基本的cocos2d元素一起工作。我将一步步为你讲解制作的过程，所以你也会学习到一些Xcode使用方面的知识。

我们将要制作的游戏和目前很出名的Doodle Jump（涂鸦跳跃）游戏相反，比较恰当的名字应该是Doodle Drop（涂鸦降落）。玩家的目标是通过晃动iOS设备来移动精灵，以躲开从空中飞下的蜘蛛怪物。**图4-1**是游戏最终完成后的效果图：

图4-1.最终完成的DoodleDrop游戏

项目设置步骤

现在打开Xcode，我将一步步引导你。在Xcode里，选择File ➤ New Project…，然后选择如**图4-2**所示的cocos2d程序模板。当被问及新项目的名称时，输入DoodleDrop，然后选择一个地方保存项目文件。Xcode会自动生成一个叫DoodleDrop的子文件夹，这样你就不用手动生成了。

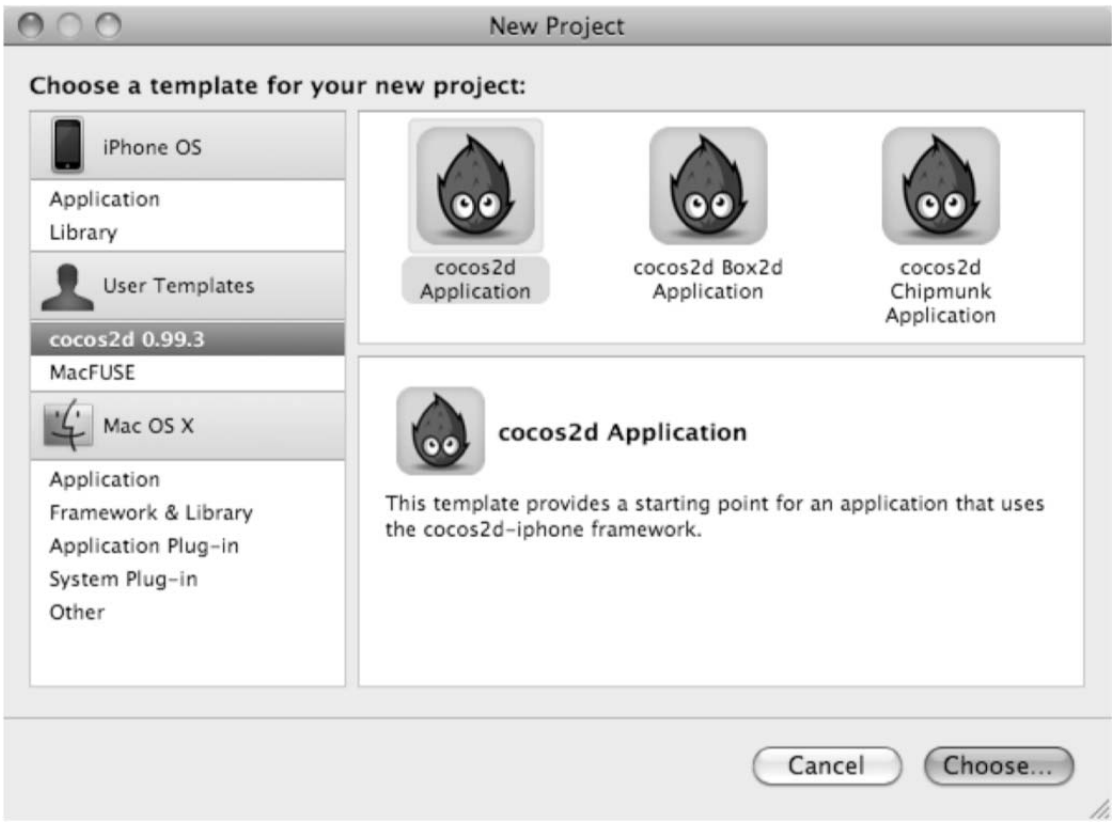

图4-2. 生成基于cocos2d程序模板的项目

Xcode会为你呈现如**图4-3**所示的项目视图。我已经在Xcode里展开了Classes和Resources这两个组，你将在那里添加源代码（Classes）和资源文件（Recources）。任何不是源代码的文件都可以被认为是资源，比如图片，声音文件，文字文件，或者plist文件，等等。我不要求你把相关的文件非常仔细地分类，但是如果你把类似的文件放在一起的话，下次你就能更容易地找到它们。

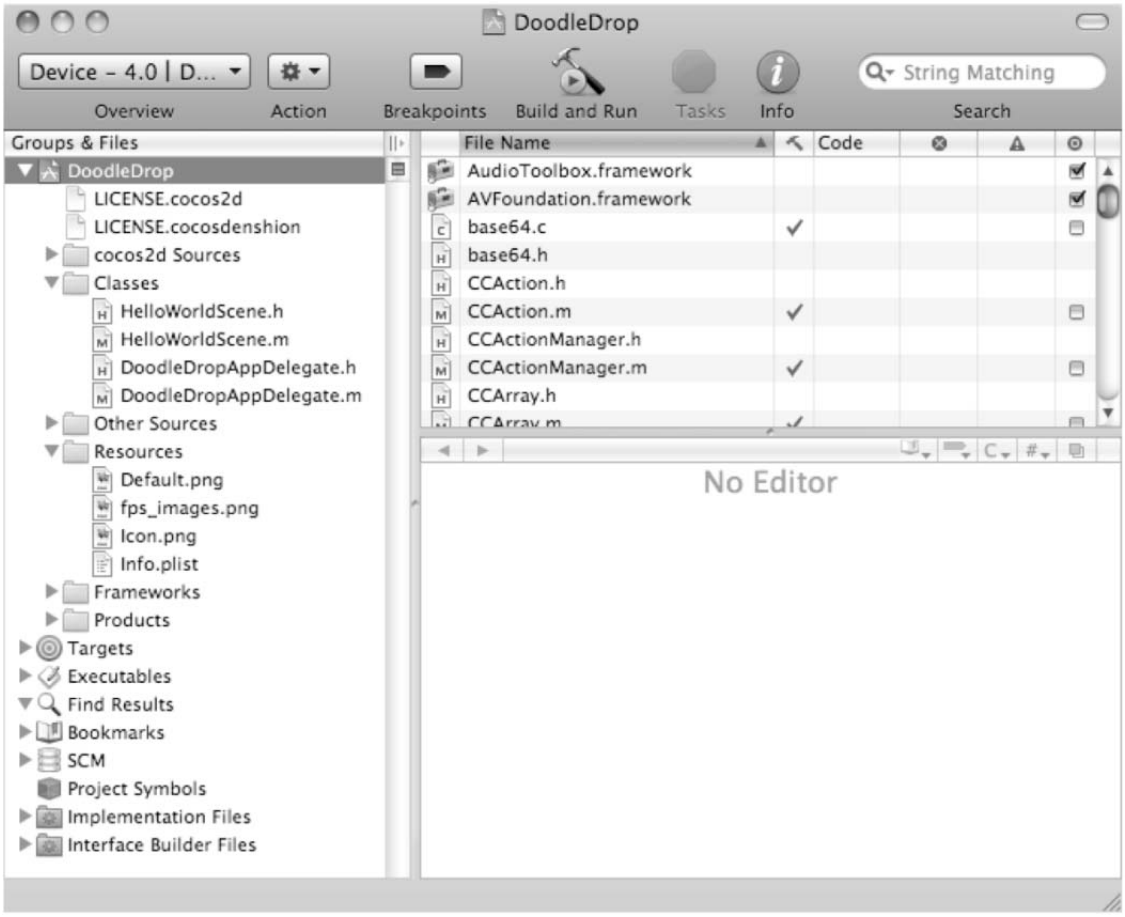

图 4-3. 当前的DoodleDrop项目是基于HelloWorld的cocos2d项目模板的。你要确保将程序和资源文件添加到相应的Classes和Resources这两个组中，让项目组织有序。

下一步你要作出一个决定:你是使用已有的HelloWorldScene作为起点，之后再把名字改成现在的项目名称呢？还是创建自己的场景呢？我选择后者。因为你迟早需要添加新的场景，所以还不如现在就学习如何从头创建一个新的场景。

请确定已选择Classes组，然后在顶部菜单选择File ➤ New File… 或者右键点击Classes，在弹出菜单上选择Add ➤ New File…，打开如**图4-4**所示的New File对话框。因为cocos2d自带了很多重要节点的类模板，不使用它们就太浪费了！在User Templates下，选择cocos2d 0.99.4（或者你当前系统安装的版本），然后选择CCNode类，请确定下方选择框中选择的是“Subclass of CCLayer”。

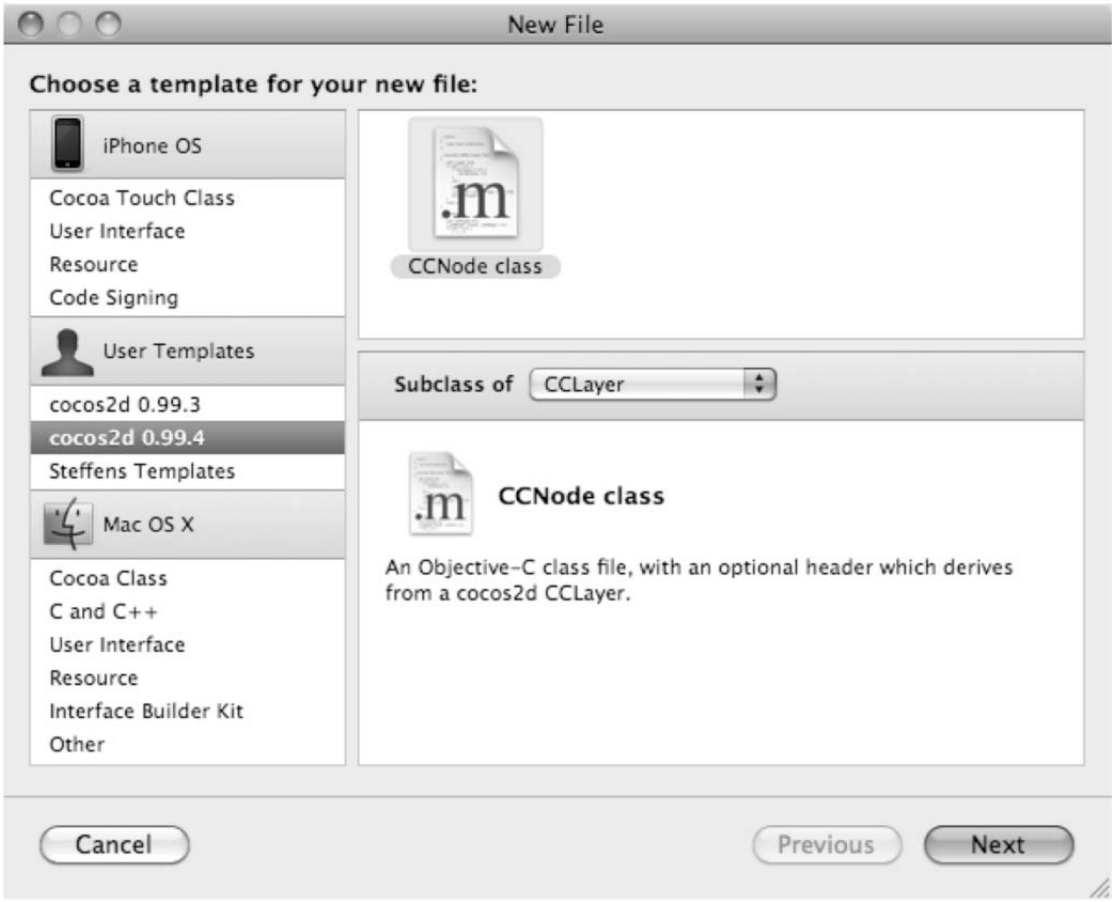

图4-4. 添加新CCNode继承类的最好方式是通过使用cocos2d提供的类模板。在我们的例子中，因为我们要创建一个新的场景，所以选择的CCNode类是CCLayer的子类（subclass）。

如**图4-5**所示，New File对话框打开。我倾向于使用功能来命名类。我在这里使用了GameScene.m来命名，因为这里是进行DoodleDrop游戏的场景。不要忘记勾选 Also create “GameScene.h”和DoodleDrop Target之前的复选框。目标（Targets）是Xcode用来决定要生成哪些可执行文件的方式。比如，iPad版本的游戏通常是作为一个分开的目标（Target）来生成的。我们的例子只有一个目标，但是如果你创建了一个iPad目标的话，你要确保iPad的高分辨率图片不会被错误地加到iPhone或iPod Touch目标中去。

注：不检查目标设置的话可能会带来很多问题，从编译错误到“无法找到文件”错误，或者由于文件没有被添加到正确的目标里面，导致游戏运行时崩溃。或者由于你把文件放进了错误的目标里而浪费了空间，比如你把给iPad和iPhone4使用的高分辨率图片添加到了普通iPhone或iPod Touch的目标里面。

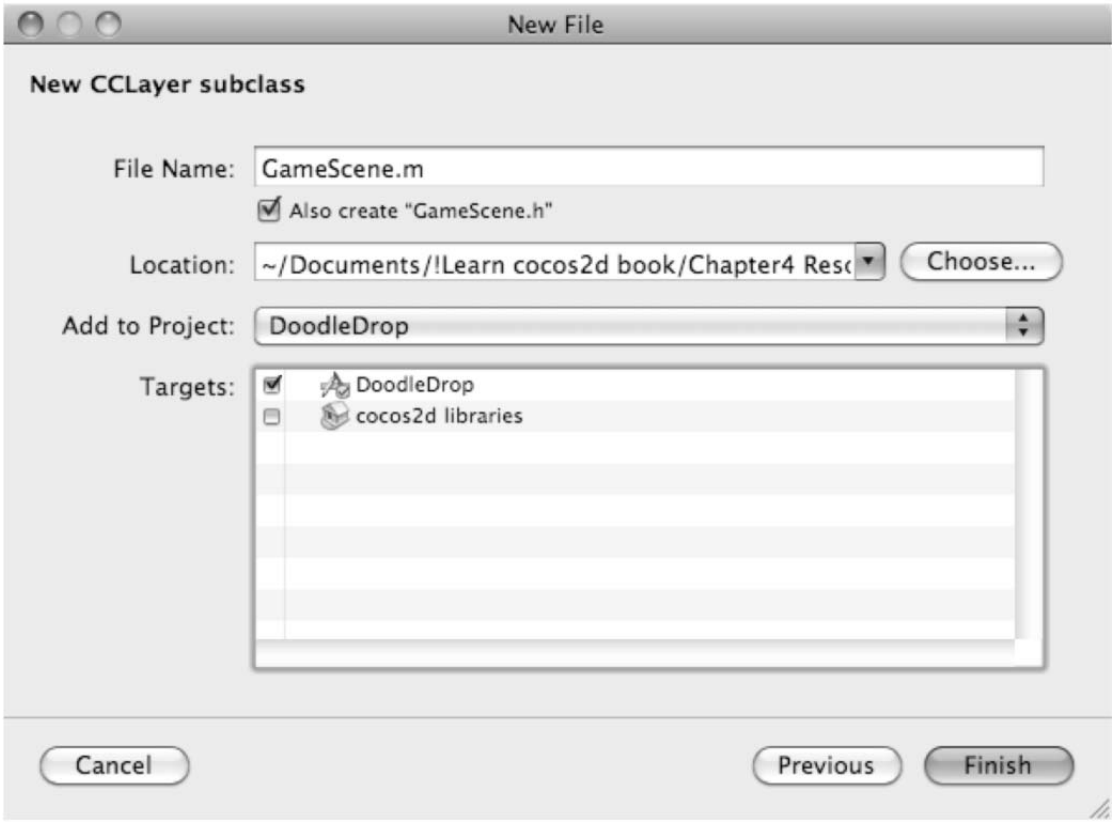

图4-5. 给新场景命名，并且确保它被添加到正确的目标（target）中。

目前，我们的GameScene类是空的，为了将它设置为一个场景，我们要做的第一件事是在里面添加+(id)scene方法。我们在这里添加的代码和第三章的基本上一样，只是层的类名称改变而已。几乎在任何一个类里面你都需要-(id)init和-(void)dealloc这两个方法，所以我们现在就加进去。我也把在第三章里介绍过的日志声明加了进去。**列表4-1**是完成的GameScene.h，**列表4-2**是完成的GameScene.m：

列表 4－1. GameScene.h和场景方法

```
#import <Foundation/Foundation.h>
#import "cocos2d.h"
@interface GameScene : CCLayer
{
}
+(id) scene;
@end
```

列表 4－2. GameScene.m和场景方法，外加一些标准方法，包括日志记录

```
#import "GameScene.h"
@implementation GameScene
+(id) scene
{
	CCScene *scene = [CCScene node];
	CCLayer* layer = [GameScene node];
```

```
        [scene addChild:layer];
        return scene;
}
-(id) init
{
        if ((self = [super init]))
        {
                CCLOG(@"%@: %@", NSStringFromSelector(_cmd), self);
        }
        return self;
}
-(void) dealloc
{
        CCLOG(@"%@: %@", NSStringFromSelector(_cmd), self);
        // 不要忘记调用 [super dealloc]
        [super dealloc];
}
@end
```

现在你可以放心地删除HelloWorldScene类了。选择HelloWorldScene.h和HelloWorldScene.m两个文件，在顶部菜单选择Edit ➤ Delete，或者右键点击选中的文件，在弹出菜单中选择Delete，然后在删除弹出对话框中，选择Also Move to Trash选项，它会把文件从硬盘上删除。删除了HelloWorldScene类以后，你必须把DoodleDropAppDelegate.m里对HelloWorldScene的引用改为引用GameScene。**列表4-3**粗体显示了需要作出的修改。我也把设备方向改为Portrait（纵向）模式，因为这个方向最适合这个游戏的设计。

列表4-3. 把DoodleDropAppDelegate.m里对HelloWorldScene的引用改为引用GameScene

```
// 用以下代码替换 #import “HelloWorldScene.h”
#import "GameScene.h"
- (void) applicationDidFinishLaunching:(UIApplication*)application
{
        // 设置纵向模式
        [director setDeviceOrientation:kCCDeviceOrientationPortrait];
        // 用GameScene替换HelloWorld
        [[CCDirector sharedDirector] runWithScene: [GameScene scene]];
}
```

编译代码然后运行，你应该会得到...一个空白的场景。成功！如果与遇到任何的问题，将你的代码和本书自带的DoodleDrop01源代码做个比较，这会有助于找到问题所在。

添加主角精灵（Player Sprite）

接下去是添加主角精灵，你将会用加速计来控制精灵的动作。要添加用于显示

主角的图片，在Xcode里选择Resources组，然后选择Project ➤ Add to Project…，或者右键点击Resources组，在弹出菜单中选择Add ➤ Existing Files…，打开文件选择对话框。主角的图片在本书提供的DoodleDrop项目的Resources文件夹里。你也可以选用自己的图片，只要它的尺寸是64x64像素即可。

注：iOS游戏首选的图片格式是PNG，Portable Network Graphics。它虽然是种压缩格式，但是与JPG不同的是，PNG是无损压缩，它将源图片的所有像素都保留了。你也可以把图片存为无压缩JPG格式，但是一般情况下同样的图片用PNG格式比无压缩JPG格式要小一些。不过，这个区别只影响应用程序的大小，它和内存的使用无关。

如**图4-6**所示，Xcode会问你要把图片存到哪里，以什么方式存储。请确保在Add To Targets选择区域中，勾选所有要用到此图片的目标前的复选框。在我们的游戏里，只有一个目标：DoodleDrop，所以我们可以直接使用默认设置。不过，当你开始制作比较大的项目时，你最好花点时间研究一下这些设置。这是Xcode的基础知识。苹果的Xcode文档描述了Xcode是如何管理项目文件的：
http://developer.apple.com/mac/library/documentation/DeveloperTools/Conceptual/XcodeProjectManagement/130-Files_in_Projects/project_files.html

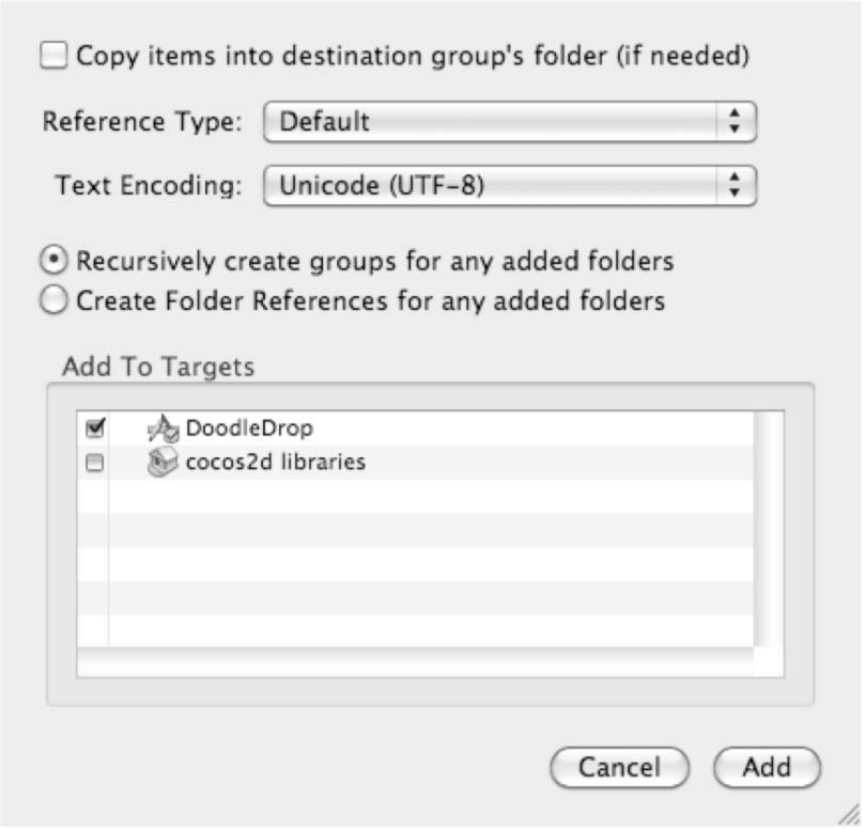

图4-6. 当你添加资源文件时，你将看到这个对话框。大多数情况下，你应该使用默认设置。

现在我们将把主角精灵添加到游戏场景中。我把主角精灵作为一个 CCSprite 成员变量添加到GameScene类中。目前为止我们的游戏还很简单，所以把所有东西都放到同一个类里不会有什么问题。一般来说，我不建议这样做。在以后的

几个项目里，为了培养你良好的代码设计习惯，我会为各个游戏组件创建各自的类。

列表4-4：演示如何在GameScene的头文件中添加 CCSprite 成员变量

```
#import <Foundation/Foundation.h>
#import "cocos2d.h"
@interface GameScene : CCLayer
{
        CCSprite* player;
}
+(id) scene;
@end
```

在**列表4-5**代码中，我添加了用于初始化精灵的 init 方法。然后将精灵赋值给成员变量。最后将它放置在屏幕下方的中央处。在这里我也启用了加速计功能。

列表4-5. 启用加速计功能，生成和放置主角精灵

```
-(id) init
{
        if ((self = [super init]))
        {
            CCLOG(@"%@: %@", NSStringFromSelector(_cmd), self);
            self.isAccelerometerEnabled = YES;
            player = [CCSprite spriteWithFile:@"alien.png"];
            [self addChild:player z:0 tag:1];
            CGSize screenSize = [[CCDirector sharedDirector] winSize];
            float imageHeight = [player texture].contentSize.height;
            player.position = CGPointMake(screenSize.width / 2, imageHeight / 2);
        }
        return self;
}
```

主角精灵（player）作为子节点被添加到场景中。它的tag值是1，以后，这个tag值将被用于寻找主角精灵，并且用此tag同其它精灵分开。你会注意到我没有保留（retain）主角精灵。因为我们将主角精灵作为子节点添加到了场景中，cocos2d会自动将其保留，也因为主角精灵需要一直存在于场景中，所以我们不需要手动保留它。不保留对象而由另一个类或对象来管理它的内存使用，叫作“保持弱引用”。

注：请注意，iOS设备上的文件名是区分大小写的。如果你加载Alien.png或者ALIEN.PNG，在模拟器上运行程序不会出现问题，但是在iOS设备上就不行了，因为我们使用的图片名是全小写的alien.png。所有的文件名都使用小写是个很好的习惯。

我们把主角精灵X轴的值设为屏幕宽度的一半，这样精灵就会被放置在水平方向的中央位置。纵向上我要让主角精灵的贴图同屏幕底部对齐。因为精灵的贴图是和精灵节点的位置对齐的，所以如果将精灵在纵向上放置于(0，0)点的话，

精灵贴图下半部分就会处于屏幕以下。我们当然不希望发生这样的事情；我们需要把精灵的位置向上移动贴图高度的一半大小，让精灵贴图全部显示出来。

通过调用[playertexture].contentSize.height，我们可以得到精灵的贴图大小（contentSize）。那么什么是“精灵的贴图大小”呢？在第三章，我提到过iOS上的贴图尺寸必须符合“2的n次方”的规定。但是实际的图片尺寸可能小于所要求的贴图尺寸。例如，如果图片大小是100x100像素，iOS上的贴图尺寸其实是128x128像素。contentSize属性返回的是图片的实际尺寸100x100像素。在绝大多数情况下，你需要的是图片的实际大小尺寸，而非贴图尺寸。

通过将精灵的Y轴值设置为图片高度的一半，主角精灵的贴图就可以和屏幕底部对齐了。

注：任何时候你都应该避免使用固定的位置。如果你只是简单的把主角精灵的位置设为(160, 32)的话，你做了两个其实应该避免的假设。第一个假设：你假设屏幕尺寸是320像素，但是实际上并不是每个iOS设备屏幕都是这个尺寸。第二个假设：你假设图片的高度是64像素，但是这个高度可能会改变。一旦你开始做出类似的假设，你会习惯于在整个项目里做出这样的假设。

我所用的代码虽然会多一些，但是从长远来说是很有好处的。你可以在不同的设备上部署这些代码，你也可以使用不同尺寸的图片文件。你再也不需要改变这部分代码。程序员最耗时的工作是改变那些基于假设的代码。

想像一下，项目开始三个月以后。你的游戏中有很多图片和对象，如果你想创建一个iPad版本的话，你必须改变所有用到固定尺寸的地方，包括那些用到固定图片尺寸的地方。如果你要创建一个iPhone4的视网膜屏版本的话，你还需要再做一遍同样的事情。在那以后，你将会有3个不同的Xcode项目需要维护。最终会导致“复制和粘贴”所造成的混乱，那就太糟糕了！

加速计输入

现在是最后一步，完成以后我们就可以晃动iPhone让主角精灵移动了。就如我在第三章演示的那样，你必须在接收加速计输入的层添加加速计方法。这里我将acceleration.x参数乘以10，然后把得到的数值和主角精灵的位置相加，乘以10 的目的是为了加快主角精灵的移动速度。

```
-(void) accelerometer:(UIAccelerometer *)accelerometer
didAccelerate:(UIAcceleration *)acceleration
{
        CGPoint pos = player.position;
        pos.x += acceleration.x * 10;
        player.position = pos;
}
```

注意到上述代码中奇怪的地方了吗？本来可能一行代码就足够的地方，我却用了三行。

```
// ERROR: lvalue required as left operand of assignment
player.position.x += acceleration.x * 10;
```

不过，不像其它编程语言如Java，C++和C#，上面的代码不能够改变Objective-C的属性。

这个问题与Objective-C属性的工作方式和C语言给变量赋值的方式有关。

player.position.x实际上调用的是位置的获取方法(getter method)：[player position]。这个方法会获取当前主角精灵的临时位置信息，上述一行代码实际上是在尝试着改变这个临时CGPoint中成员变量x的值。不过这个临时的CGPoint是要被丢弃的。在这种情况下，精灵位置的设置方法(setter method)：[player setPosition]根本不会被调用。你必须直接赋值给player.position这个属性，这里使用的值是一个新的CGPoint。在使用Objective-C的时候，你必须习惯于这个规则，而唯一的办法是改变你从Java，C++或C#里带来的编程习惯。

第一次测试运行

现在你的项目应该具备与DoodleDrop02项目一样的代码了。试着运行一下。因为在模拟器下无法使用加速计，所以请务必在真实iOS设备上运行你的代码，测试一下加速计的运行状况。

如果你的Xcode中还没有安装Development Provisioning Profiles的话，你将会得到一个CodeSign出错信息。要想在真实iOS设备上运行你的代码，必须首先做Code Signing。请参照以下链接了解如何生成和安装必要的Development Provisioning Profiles：
(https://developer.apple.com/iphone/manage/provisioningprofiles/howto.action).
注：要访问上述链接，你必须拥有一个付费的苹果开发者帐号，99美元一年。

主角精灵的速率

你可能注意到加速计的反应有些慢，动作不是很流畅。那是因为主角精灵没有做到真正的加速和减速。让我们修正一下这个问题。修改过的代码在DoodleDrop03项目文件夹下。

实现加速和减速的方式不是直接去改变主角精灵的位置信息，而是使用一个单独的CCPoint变量作为速度矢量。

每一次系统接收到一个加速计事件，速度变量就会累计从加速计那里得到的输入数值。当然，这就意味着我们必须给这个速度变量设置一个最大值，以防主角精灵在减速上花费太多时间。然后，在每一帧里面，不管系统是否接收到加速计事件，都将速度变量和当前主角精灵的位置相加。

注：为什么不使用动作(actions)直接移动主角精灵呢？当一个物体需要经常改变它的速度或方向时（每秒钟好几次），选择动作来移动它就不是个好选择了。因为动作是设计用于存在时间相对较长的物体的，所以频繁的生成动作会增加系统分配和释放内存的开销，降低程序运行效率。

更糟糕的是，如果你不为动作的运行留出时间的话，动作根本就不会起作用。这也是为什么在每一帧里都用新的动作去替换旧的动作时，新动作不会产生任何作用的原因。因为新的动作根本没有时间去替换旧的动作。很多cocos2d开发者碰到过这个看起来很奇怪的问题。

比如，你把当前的动作都停止，然后在下一帧里添加MoveBy动作，这个MoveBy动作就完全不会起作用！因为MoveBy动作只会在再下一帧里改变物体的位置。但是到再下一帧的时候，你已经停止了所有的当前动作，同时添加了另一个MoveBy动作。这样重复的动作是不会移动物体的。

让我们看一下所作的代码修改。在以下代码里，playerVelocity变量被添加到头文件中：

```
@interface GameScene : CCLayer
{
	CCSprite* player;
	CGPoint playerVelocity;
}
```

如果你不理解为什么我使用了CGPoint而不是float，那是因为我们也可能在将来让主角精灵上下移动，所以用CGPoint可以让代码的扩展性更好。

列表4-6展示了处理加速计事件的代码，我用速度变量来间接更新主角精灵的位置而不是直接更新主角精灵。以下代码使用了三个新的“设计参数”：**减速度速率**，**加速计敏感度**和**最大速度**。这三个数值没有最佳值可用：你需要做些调整以找到符合你的游戏设计的最佳设定，这也是为什么它们被称为“设计参数”的原因。

减速是通过降低当前速度值来实现的。在降低速度值以后，要将新的加速度值和加速计敏感度值相乘所得的值与降低后的速度值相加。**减速度速率**越小，主角精灵就能越快的改变蜘蛛的方向。**加速计敏感度**的值越大，主角精灵对加速计的输入就越敏感。因为这三个参数一起影响着同一个值，所以在调整的时候要注意每次只调整一个值。

列表4-6. GameScene的头文件设置了playerVelocity变量

```
-(void) accelerometer:(UIAccelerometer *)accelerometer
didAccelerate:(UIAcceleration *)acceleration
{
	// 控制减速的速率（值越低=可以更快的改变方向）
	float deceleration = 0.4f;
	//加速计敏感度的值越大，主角精灵对加速计的输入就越敏感
	float sensitivity = 6.0f;
```

```
        // 最大速度值
        float maxVelocity = 100;

        // 基于当前加速计的加速度调整速度
        playerVelocity.x = playerVelocity.x * deceleration + acceleration.x *
sensitivity;

        // 我们必须在两个方向上都限制主角精灵的最大速度值
        if (playerVelocity.x > maxVelocity)
        {
                playerVelocity.x = maxVelocity;
        }
        else if (playerVelocity.x < - maxVelocity)
        {
                playerVelocity.x = - maxVelocity;
        }
}
```

现在，playerVelocity的值会发生变化了，但是我们如何使用它来影响主角精灵的位置呢？通过在GameScene的init方法中预约更新方法来实现。将以下代码添加到GameScene的init方法中：

```
// 设置在每一帧都运行的预约方法：-(void) update:(ccTime)delta
[self scheduleUpdate];
```

如**列表4-7**所示，你需要添加 -(void)update: (cc Time)delta 方法。这个预约的更新方法每一帧都会被调用，我们就在这里用速度值来影响主角精灵的位置。这样无论加速计输入事件发生的频率有多大，主角精灵都能很平滑流畅的移动。

列表4-7. 用当前速度更新主角精灵的位置

```
-(void) update:(ccTime)delta
{
        // 用playerVelocity持续增加主角精灵的位置信息
        CGPoint pos = player.position;
        pos.x += playerVelocity.x;

        // 如果主角精灵移动到了屏幕以外的话，它应该被停止
        CGSize screenSize = [[CCDirector sharedDirector] winSize];
        float imageWidthHalved = [player texture].contentSize.width * 0.5f;
        float leftBorderLimit = imageWidthHalved;
        float rightBorderLimit = screenSize.width - imageWidthHalved;

        // 以防主角精灵移动到屏幕以外
        if (pos.x < leftBorderLimit)
        {
                pos.x = leftBorderLimit;
                playerVelocity = CGPointZero;
```

```
        }
        else if (pos.x > rightBorderLimit)
        {
                pos.x = rightBorderLimit;
                playerVelocity = CGPointZero;
        }

        // 将更新过的位置信息赋值给主角精灵
        player.position = pos;
}
```

边界测试可以防止主角精灵离开屏幕。我们需要将精灵贴图的contentSize考虑进来，因为精灵的位置在精灵贴图的中央，但是我们不想让贴图的任何一边移动到屏幕外面。所以我们计算得到了imageWidthHalved值，并用它来检查当前的精灵位置是不是落在左右边界里面。上述代码可能有些啰嗦，但是这样比以前更容易理解。这就是所有与加速计处理逻辑相关的代码。

注：在计算imageWidthHalved时，我们将contentSize乘以0.5,而不是用它除以2。这是一个有意的选择，因为除法可以用乘法来代替以得到同样的计算结果。

因为上述更新方法在每一帧都会被调用，所以所有代码必须在每一帧的时间里以最快的速度运行。因为iOS设备使用的ARM CPU不支持直接在硬件上做除法，乘法一般会快一些。虽然在我们的例子里效果并不明显，但是养成这个习惯对我们很有好处。

添加障碍

如果我们的游戏没有什么东西可以让主角避让的话，那就太无聊了。DoodleDrop04项目中包含了一个令人憎恶的东西：六条腿的大蜘蛛。谁不想避开它们呢？

和主角精灵一样，你需要把spider.png添加到Resources组中。然后在GameScene.h的interface中添加三个新的成员变量。第一个是CCArray* spiders，它的类引用展示在**列表4-9**中。另外两个变量是spiderMoveDuration和numSpidersMoved，它们被用于**列表4-12**所示的代码中。

```
@interface GameScene : CCLayer
{
        CCSprite* player;
        CGPoint playerVelocity;
        CCArray* spiders;
        float spiderMoveDuration;
        int numSpidersMoved;
}
```

我也在GameScene的init方法中添加了对initSpiders方法的调用，紧跟scheduleUpdate方法之后：

```
-(id) init
{
	if ((self = [super init]))
	{
		…
		[self scheduleUpdate];
		[self initSpiders];
	}
	return self;
}
```

在GameScene类中添加了上述代码之后，我们创建如**列表4-8**所示的initSpiders方法，此方法用于生成蜘蛛精灵。

列表4-8：为了方便使用，用一个CCArray将蜘蛛精灵们放在一起

```
-(void) initSpiders
{
	CGSize screenSize = [[CCDirector sharedDirector] winSize];

	// 利用一个临时的蜘蛛精灵以得到蜘蛛图片的大小
	CCSprite* tempSpider = [CCSprite spriteWithFile:@"spider.png"];
	float imageWidth = [tempSpider texture].contentSize.width;

	// 计算可以在同一行的水平方向上同时显示的蜘蛛个数
	int numSpiders = screenSize.width / imageWidth;

	// 用alloc初始化蜘蛛数组
	spiders = [[CCArray alloc] initWithCapacity:numSpiders];
	for (int i = 0; i < numSpiders; i++)
	{
		CCSprite* spider = [CCSprite spriteWithFile:@"spider.png"];
		[self addChild:spider z:0 tag:2];
		// 将蜘蛛精灵添加到数组
		[spiders addObject:spider];
	}

	// 调用方法以排列蜘蛛
	[self resetSpiders];
}
```

上述代码有一些需要注意的地方。我生成了一个临时的CCSprite（tempSpider）以得到精灵图片的宽度。在之后的代码中，此宽度用于计算在同一行的水平方向上同时显示的蜘蛛个数。得到图片大小最简单的方法就是生成一个临时

CCSprite精灵。你会注意到我没有把tempSpider作为子节点添加到其它节点中。这意味着tempSpider的内存会被自动释放。

和上述做法不同，用于放置蜘蛛精灵的数组必须使用alloc来生成；否则数组的内存会被释放，之后访问精灵数组的话就会导致程序因为EXC_BAD_ACCESS错误而崩溃。并且因为是我在控制精灵数组的内存，我必须在dealloc方法中释放蜘蛛精灵数组，如下所示：

```
-(void) dealloc
{
        CCLOG(@"%@: %@", NSStringFromSelector(_cmd), self);

        // 因为蜘蛛精灵数组由[CCArray alloc]生成，此数组必须在这里被释放
        [spiders release];
        spiders = nil;


        // 永远不要忘记调用 [super dealloc]!
        [super dealloc];
}
```

在撰写本书时，CCArray类还没有相关的文档，但是cocos2d完全支持它。你可以在Xcode项目中的cocos2d/Support组找到CCArray的类文件。cocos2d引擎内部使用了CCArray。它和苹果官方的NSMutableArray类似，但是运行效率更高。CCArray类实现了NSArray和NSMutableArray的子集，并且添加一些从NSArray初始化到CCArray的新方法。CCArray通过将数组中的最后一个对象(nil)赋值给删除的位置，实现了fastRemoveObject和fastRemoveObjectAtIndex两个方法。避免了复制部份数组的内存。这让删除数组中的元素变的更快，不过这也意味着CCArray中的对象会改变位置。如果你的程序依赖于指定的对象排列次序，你就不应该使用fastRemoveObject这类方法。在**列表4-9**中，你可以看到CCArray的类引用 - 它没有实现所有的NSArray和NSMutableArray方法：

列表4-9. CCArray的类引用

```
+ (id) array;
+ (id) arrayWithCapacity:(NSUInteger)capacity;
+ (id) arrayWithArray:(CCArray*)otherArray;
+ (id) arrayWithNSArray:(NSArray*)otherArray;
- (id) initWithCapacity:(NSUInteger)capacity;
- (id) initWithArray:(CCArray*)otherArray;
- (id) initWithNSArray:(NSArray*)otherArray;
- (NSUInteger) count;
- (NSUInteger) capacity;
- (NSUInteger) indexOfObject:(id)object;
- (id) objectAtIndex:(NSUInteger)index;
- (id) lastObject;
- (BOOL) containsObject:(id)object;
```

```
#pragma mark Adding Objects
- (void) addObject:(id)object;
- (void) addObjectsFromArray:(CCArray*)otherArray;
- (void) addObjectsFromNSArray:(NSArray*)otherArray;
- (void) insertObject:(id)object atIndex:(NSUInteger)index;
#pragma mark Removing Objects
- (void) removeLastObject;
- (void) removeObject:(id)object;
- (void) removeObjectAtIndex:(NSUInteger)index;
- (void) removeObjectsInArray:(CCArray*)otherArray;
- (void) removeAllObjects;
- (void) fastRemoveObject:(id)object;
- (void) fastRemoveObjectAtIndex:(NSUInteger)index;
- (void) makeObjectsPerformSelector:(SEL)aSelector;
- (void) makeObjectsPerformSelector:(SEL)aSelector withObject:(id)object;
- (NSArray*) getNSArray;
```

在**列表4-8**的结束处，我调用了[self resetSpiders]方法；**列表4-10**展示了这个方法。将精灵初始化和放置精灵分开是因为游戏最终是要结束的。结束以后游戏需要重新设置。最有效率的做法是将所有游戏对象移到它们的初始位置。但是，一旦你的游戏变的更加复杂，这个做法就行不通了。最后，最简单的方法是将整个场景重新加载，代价是玩家需要等待场景被重新加载。

列表4-10. 重设蜘蛛精灵们的位置

```
-(void) resetSpiders
{
	CGSize screenSize = [[CCDirector sharedDirector] winSize];

	// 生成一个临时蜘蛛对象，得到它的图片宽度
	CCSprite* tempSpider = [spiders lastObject];
	CGSize size = [tempSpider texture].contentSize;

	int numSpiders = [spiders count];
	for (int i = 0; i < numSpiders; i++)
	{
		// 将生成的蜘蛛放在屏幕外制定的位置
		CCSprite* spider = [spiders objectAtIndex:i];
		spider.position = CGPointMake(size.width * i + size.width * 0.5f, screenSize.height +
	size.height);
		[spider stopAllActions];
	}

	// 将预约的方法取消掉。如果没有方法之前没有被预约过的话，这行代码不会产生任何作用
	[self unschedule:@selector(spidersUpdate:)];
```

```
        // 预约蜘蛛的更新方法，用指定的时间间隔进行调用
        [self schedule:@selector(spidersUpdate:) interval:0.7f];
}
```

在上述代码中我再次使用了一个临时蜘蛛精灵以得到图片尺寸。这次我没有生成新的蜘蛛精灵，因为已经存在一个蜘蛛精灵的数组。而且因为所有蜘蛛精灵使用的是同一个图片，所以我可以使用任意一个蜘蛛精灵。因此我直接使用了数组的最后一个蜘蛛精灵。

然后通过修改每个蜘蛛的位置信息，我们将所有蜘蛛横向排列开来。在x坐标轴上我们多加了半个图片的宽度（因为精灵贴图是居中放置在精灵节点的位置上的）。至于高度的设定，每一个蜘蛛被放置在屏幕外离开屏幕顶部一个蜘蛛图片高度的地方。这个高度是任意设置的。只要图片在屏幕之外不可见就可以了。因为在蜘蛛位置重置的时候，有的蜘蛛有可能还在移动，所以我在这里停止了所有的动作。

技巧：如果不是绝对必要，不要在for或者其它循环里的条件判断中调用方法，这样可以节省几个CPU循环。在我们的例子中，我使用了一个名为 numSpiders 的变量，用于存放[spiders count]的计算结果。然后在for循环的条件判断中使用 numSpiders 而不是[spiders count]。因为spiders数组并没有在for循环里面被修改，所以此数组的元素个数在循环过程中是不会改变的。这也是为什么我可以将此计算结果保存在变量中，而不至于在for循环的条件判断中反复调用[spiders count]。

我预约了spidersUpdate:这个方法，让它每0.7秒运行一次，也就是说每0.7秒屏幕上方将有一个蜘蛛降落下来。但是在预约之前，我通过使用：[self unschedule:@selector(spidersUpdate:)]，来确保spidersUpdate:这个方法还没有被预约过。如果不这样做的话，你在调用resetSpiders方法的同时，spidersUpdate:可能还是会运行，这样的话实际上是将spiderUpdate:这个方法运行了两次，也就是让蜘蛛降落的频率增加了一倍。**列表4-11**中的spidersUpdate:方法会挑选一个已存在的蜘蛛，检查一下它是不是处于闲置状态，如果发现蜘蛛处于闲置状态，就使用一系列的动作让它从天而降：

列表4-11. spidersUpdate:方法会让蜘蛛经常从天而降

```
-(void) spidersUpdate:(ccTime)delta
{
        // 尝试着寻找一个目前闲置不动的蜘蛛
        for (int i = 0; i < 10; i++)
        {
                int randomSpiderIndex = CCRANDOM_0_1() * [spiders count];
                CCSprite* spider = [spiders objectAtIndex:randomSpiderIndex];

                // 如果蜘蛛不动的话，应该没有任何正在运行的动作
```

```
            if ([spider numberOfRunningActions] == 0)
            {
                // 以下是控制蜘蛛运动的动作序列
                [self runSpiderMoveSequence:spider];

                // 每次只能有一只蜘蛛会开始移动
                break;
            }
        }
}
```

你可能觉得奇怪，为什么我一定要经过10次循环才拿到一个随机的蜘蛛呢？因为我并不知道与随机生成的索引相关的蜘蛛是否闲置不动，所以我想确保找到一个目前闲置不动的蜘蛛。如果经过10次尝试（10这个数字是任意的，也可能是20）还找不到的话，那么我就会跳过目前的更新，等待下一次更新。

我也可以使用do/while循环一直寻找闲置不动的蜘蛛。但是，这取决于我们的“设计参数”，比如新蜘蛛被投放的频率，有可能所有的蜘蛛都在同一时间内移动；如果那样的话，游戏就会被锁定，因为系统在无止境的寻找闲置不动的蜘蛛。此外，即使游戏不能够在几秒钟内投放新的蜘蛛，也没有什么关系。不过，如果你查看一下DoodleDrop04项目的代码，你会看到我添加了一个日志功能，用于打印出发现一只闲置蜘蛛所需要尝试的次数。

因为蜘蛛们只会执行移动序列动作，所以我可以通过检查蜘蛛是否在运行任何动作来确定它是不是闲置不动。我们接下来讨论**列表4-12**中与之相关的runSpiderMoveSequence方法：

列表4-12. 蜘蛛的移动是由动作序列来 控制的

```
-(void) runSpiderMoveSequence:(CCSprite*)spider
{
    // 随着时间慢慢增加蜘蛛的移动速度
    numSpidersMoved++;

    if (numSpidersMoved % 8 == 0 && spiderMoveDuration > 2.0f)
    {
        spiderMoveDuration -= 0.1f;
    }

    // 用于控制蜘蛛移动的动作序列
    CGPoint belowScreenPosition = CGPointMake(spider.position.x,
                                              -[spider texture].contentSize.height);
```

```
        CCMoveTo* move = [CCMoveTo actionWithDuration:spiderMoveDuration
                                             position:belowScreenPosition];

        CCCallFuncN* call = [CCCallFuncN actionWithTarget:self
                                             selector:@selector(spiderBelowScreen:)];

        CCSequence* sequence = [CCSequence actions:move, call, nil];

        [spider runAction:sequence];
}
```

RunSpiderMoveSequence方法的作用是跟踪已被放下的蜘蛛数量。每次到第八个蜘蛛时，spiderMoveDuration的值就会被减少，从而提高所有蜘蛛的移动速度。%这个符号叫作“余数运算子”（Modulus Operator），用于得到运用除法以后得到的余数。比如，如果numSpidersMoved可以用8除尽，那么“余数运算子”的计算结果就应该是0。

这里用到的动作序列只有一个CCMoveTo动作和一个CCCallFuncN动作。你可以改进蜘蛛的行为，比如让它往下移动一点，等个几秒钟，然后一直移动到底部，就像真的邪恶的蜘蛛通常会做的那样。我将把具体的做法留给你去发挥。我选择CCCallFuncN的目的是给spiderBelowScreen方法传递蜘蛛精灵作为它的sender变量。这样的话，当某只蜘蛛到达屏幕底部时，我就可以直接引用那个蜘蛛，不需要再去到处找了。**列表4-13**的代码会在蜘蛛移动到超出屏幕底部以后，通过重新设置蜘蛛位置让它回到刚好超出屏幕顶部的位置：

列表4-13. 重新设置蜘蛛位置，这样它就可以再次从天而降。

```
-(void) spiderBelowScreen:(id)sender
{
        // 确保传进来的sender参数是我们需要的类
        NSAssert([sender isKindOfClass:[CCSprite class]], @"sender is not a CCSprite!");
        CCSprite* spider = (CCSprite*)sender;

        // 将蜘蛛移到刚好超出屏幕顶部的位置
        CGPoint pos = spider.position;
        CGSize screenSize = [[CCDirector sharedDirector] winSize];
        pos.y = screenSize.height + [spider texture].contentSize.height;
        spider.position = pos;
}
```

注：作为一名防守型的程序员，因为我假设了sender将会是一个CCSprite，但是传进来的参数可能不是这个类，所以我使用了NSAssert来确定sender是一个CCSprite。

实际上，我在第一次写上述代码的时候，使用了CCCallFunc，而不是CCCallFuncN。使用CCCallFunc的结果是sender为nil，因为CCCallFunc不会将sender参数传进来。因为sender是nil，isKindOfClass将不会被调用，导致返回值为nil，所以NSAssert报告了这个问题。由于NSAssert的提示，我轻松地找到了问题所在。

一旦确认了sender是一个CCSprite，我就可以将它转换成一个CCSprite，然后用它来调整精灵的位置。现在你应该已经熟悉调整位置的过程了。

一切顺利。你可以试着玩一下游戏，然后你就会发现少了些什么东西。**提示：**请看以下标题。

碰撞测试

你可能惊奇地发现，原来碰撞测试可以如**列表4-14**代码所示的那样简单。当然，以下代码只是测试了主角精灵和所有蜘蛛精灵之间的距离，这类碰撞测试只做径向测试（radial check）。不过对于我们的游戏，这样的测试足够了。我把对[self checkForCollision]的调用放到了-(void) update: (ccTime)delta方法的结束处。

列表4-14. 简单的径向碰撞测试可以满足我们的要求

```
-(void) checkForCollision
{
        // 假设：主角精灵和蜘蛛精灵使用的图片都是正方形的
        float playerImageSize = [player texture].contentSize.width;
        float spiderImageSize = [[spiders lastObject] texture].contentSize.width;
        float playerCollisionRadius = playerImageSize * 0.4f;
        float spiderCollisionRadius = spiderImageSize * 0.4f;

        // 这里的碰撞距离和图片形状大约一致
        float maxCollisionDistance = playerCollisionRadius + spiderCollisionRadius;

        int numSpiders = [spiders count];
        for (int i = 0; i < numSpiders; i++)
        {
                CCSprite* spider = [spiders objectAtIndex:i];
                if ([spider numberOfRunningActions] == 0)
                {
                        // 这只蜘蛛没有移动，所以我们可以跳过对它的碰撞测试
                        continue;
                }

                // 得到主角精灵和蜘蛛精灵之间的距离
                float actualDistance = ccpDistance(player.position, spider.position);
```

```
                // 检查是否两个物体已经碰撞
                if (actualDistance < maxCollisionDistance)
                {
                        // 游戏没有被结束掉，这里只是重新设定了蜘蛛的位置
                        [self resetSpiders];
                }
        }
}
```

主角精灵和蜘蛛精灵的图片尺寸用于决定碰撞半径，得到的粗略估计值对这个游戏来说已经足够。如果你查看DoodleDrop05项目，你会发现我添加了一个调试用的画图方法，此方法会将每个精灵的碰撞半径渲染出来。

我循环测试所有蜘蛛，但是忽略那些不活动的蜘蛛，因为它们肯定不会在碰撞半径之内。蜘蛛和主角精灵之间的距离通过ccpDistance方法来计算。这是另一个没有文档说明但是cocos2d完全支持的方法。你可以在cocos2d的Xcode项目中的cocos2d/Support组，找到CGPointExtension文件，里面有很多有用的数学函数。

计算得到的距离用于和主角精灵与蜘蛛的碰撞半径之和进行比较。如果距离小于碰撞半径之和，那么可以判定碰撞发生。这里，我没有让游戏结束，只是重新设定蜘蛛的位置而已。

得分标签

我们的游戏需要某种得分的机制。我决定添加一个简单的时间间隔计数器作为得分。首先我在GameScene类的init方法里添加得分的显示标签：

```
scoreLabel = [CCLabel labelWithString:@"0" fontName:@"Arial" fontSize:48];
scoreLabel.position = CGPointMake(screenSize.width / 2, screenSize.height);

// 调整标签的定位点(anchorPoint)的Y轴位置，让其与屏幕顶部对齐
scoreLabel.anchorPoint = CGPointMake(0.5f, 1.0f);

// 将标签的z值设定为-1，这样它就会显示在所有其它物体的下面一层
[self addChild:scoreLabel z:-1];
```

我有意选择了CCLabel对象，因为大多数刚开始使用cocos2d的程序员一般都会选择它作为标签。不过，我将很快让你看到标签不好的一面。把以下代码添加到更新方法的任意一个地方，这样得分标签的数字就会随着每次更新方法的调用像秒表一样显示更新：

```
// 每秒更新一次得分标签
totalTime += delta;
int currentTime = (int)totalTime;
```

```
if (score < currentTime)
{
        score = currentTime;
        [scoreLabel setString:[NSString stringWithFormat:@"%i", score]];
}
```

udate:方法的delta参数被持续地加到totalTime成员变量里，它被用于跟踪时间的变化。因为totalTime是一个浮点数变量，我直接将它赋值给一个整数变量，这样实际上是移除了浮点数的小数点后面部分。这样我们就可以用得到的数值与currentTime作比较了。如果当前得分小于currentTime，currentTime就变成了当前得分，得分标签（CCLabel）的数字就会被更新。

这里，标签不好的一面就显露出来了。在上一章，我提到更新CCLabel的文字速度会很慢，因为上面的文字实际上是张贴图，它需要通过iOS的字体渲染方法来生成，同时还要兼顾分配一个新的贴图和释放旧的贴图，这些都很花时间。如果你在iOS设备上玩现在这个版本的话，你会注意到每次得分标签上的文字发生变化的时候，蜘蛛都会稍微跳一下。游戏运行不再流畅。

如果你想看到更明显的效果，就把if条件判断去掉，这样[scoreLabel setString:…]就会每一帧都被调用。在我的iPhone 3GS上，帧率会从之前始终如一的60帧每秒突然下降到30帧每秒，这是由于CCLabel每一帧都在更新文字所造成的！

不过CCLabel只有在经常改变文字时才会让游戏变慢。如果你一次性生成它的文字，从此再无文字变化，它的运行速度和其它相同尺寸的CCSprite是一样的。

介绍CCBitmapFontAtlas和Hiero

CCBitmapFontAtlas类可以让标签文字更新速度加快。在DoodleDrop07项目中，我用CCBitmapFontAtlas替换了CCLabel。你只要在头文件里将scoreLabel变量的声明从CCLabel改为CCBitmapFontAtlas，然后把init方法中的调用改成以下代码即可：

```
scoreLabel = [CCBitmapFontAtlas bitmapFontAtlasWithString:@"0" fntFile:@"bitmapfont.fnt"];
```

注：Bitmap字体对游戏来说是个非常好的选择，因为它的速度很快。不过它有一个很大的缺点：任何bitmap的字体大小都是固定的。如果你需要相同的字体，但是需要字体大一些的话，你可以放大CCBitmapFontAtlas，但是付出的代价是牺牲图片质量。另外一个选择是生成一个单独的字体文件。但是这会使用更多的内存，因为即使只是改变字体大小，每个bitmap字体都会带有自己的贴图。

不过实际上没有那么简单。因为你需要添加bitmapfont.fnt和配套的bitmapfont.png两个文件，两个文件都要放在项目的Resources文件夹中。这两个文件一般需要你自己创建。

既然迟早要自己创建bitmap字体文件，我们就来讨论一下如何创建这些文件。我们使用的工具叫Hiero，开发者是Kevin James Glass。它是一个免费的Java Web程序，你可以在这里找到它：http://slick.cokeandcode.com/demos/hiero.jnlp

因为Hiero是个免费的Java Web程序，所以它没有安全证书，它会要求你相信它。不过从另一方面来说，因为已经有很多开发者成功地使用了这个工具，所以它应该是值得信赖的。如果你希望使用另外的程序，可以试一下BMFont。不过BMFont是个Windows程序，它需要在Windows平台上运行。这也是为什么在Mac开发者社区里没有得到广泛应用的原因。你可以在这里下载BMFont：http://www.angelcode.com/products/bmfont/

图4-7显示了我用于创建bitmapfont.fnt的配置信息。你可以在DoodleDrop07项目的Resources文件中打开bitmapfont.hiero文件，看到相同的配置信息。基本的操作过程很简单。你在列表中选择一个字体，在Sample Text输入框中输入需要的文字，然后保存文件。因为Hiero是个Java Web程序，所以你在通常显示菜单的地方不会看到它的菜单。它的菜单在程序窗口的左上角，就像Windows程序一样。而且还有个奇怪的地方是，在你保存文件时Hiero会忘记添加文件扩展名。所幸只有两种扩展名，一个是.hiero，此类文件包含了当前编辑器的设置信息。另一个是.fnt文件，这是cocos2d的CCBitmapFontAtlas类可以读取的文件类型。

剩下的设置完全是可选的。我决定添加渐变和阴影的效果，让字体拥有更多色彩和3D的感觉。从Hiero窗口右上角的Effects列表，你可以选择需要的效果。然后点击Add按钮，一个新的效果将被添加到下面的列表中。你可以在那里调整所有效果的参数。调整的结果将会显示在Rendering区域。如果你不喜欢某个效果，点击效果名称右边的X按钮可以移除它。

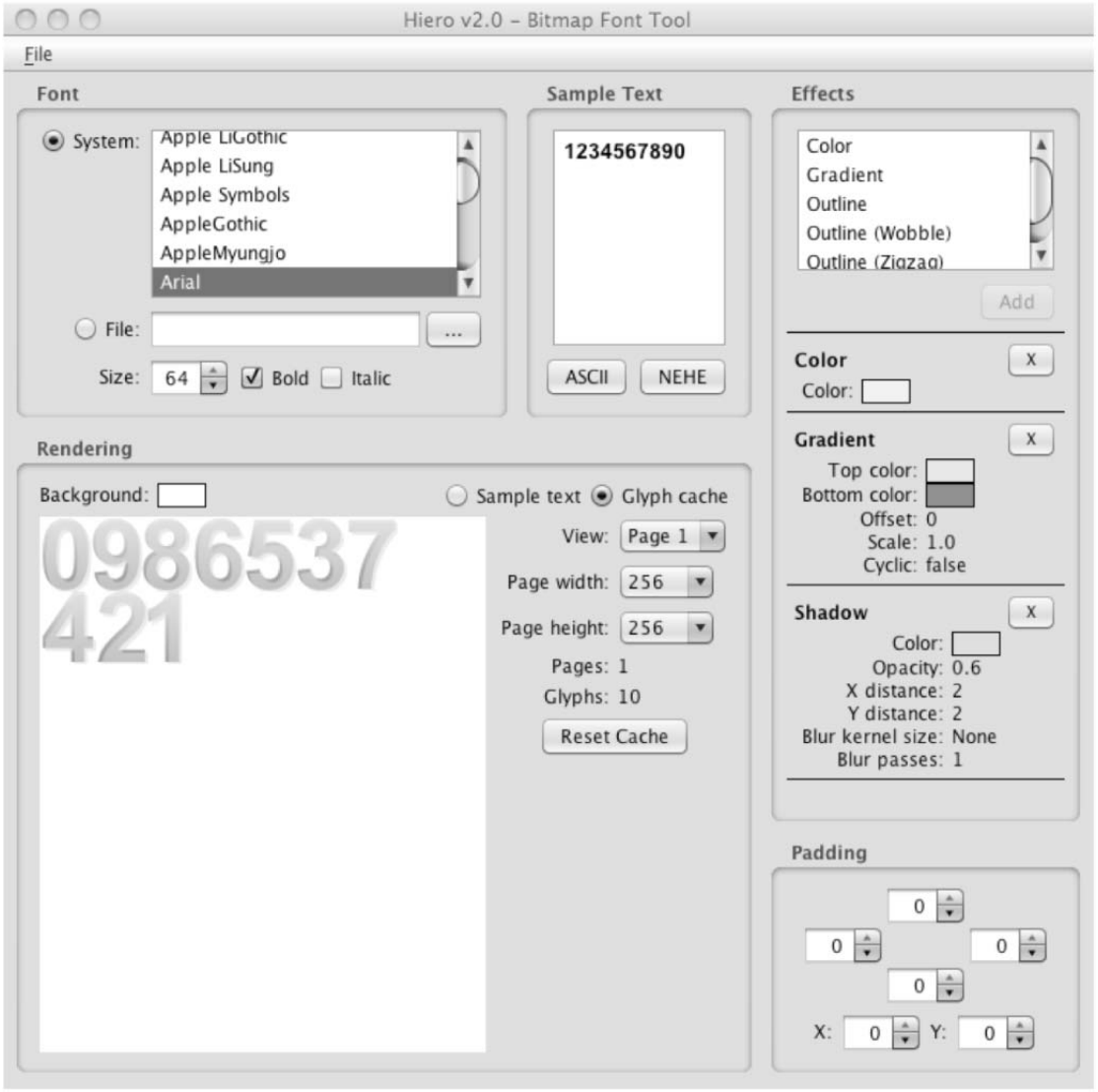

图4-7. Hiero Bitmap Font Tool可以根据TrueType字体生成bitmap字体。它生成的.fnt文件可以直接为cocos2d的CCBitmapFontAtlas类所用。

左下方的Rendering区域有几个设置可用于裁剪Hiero生成的图片文件。如果你只需要几个字母，可以将Page width（页面宽度）和Page height（页面高度）设置到最小的尺寸，在我们的例子中是256像素。设置完后点击Reset Cache按钮应用新的设置。这会生成小一些的图片文件，可以节省内存。对于更复杂的字体，Hiero会生成好几个图片文件。CCBitmapFontAtlas不会对此有任何问题，但是不要忘记把所有Hiero生成的.png图片添加到Xcode的Resources组里。

我只选择了数字用于生成需要的bitmap字体。因为Hiero生成的每个字体文件至少是256x256像素，所以你只能省下一小部分内存。其实在Sample Text区域你还可以输入更多的字母，因为还有很多空间。

如果你想学习更多关于Hiero的用法，你可以查看我网站上的教程：
http://www.learn-cocos2d.com/knowledge-base/tutorialbitmap-fonts-hiero/

注：如果你用CCBitmapFontAtlas显示的文字不存在于.fnt文件中，它们将不会

被显示。例如，使用[label setString:@“Hello, World!”]来设置标签文字，如果你的bitmap字体里只有小写字母，那么上述代码显示的文字将会是“ello orld”。

添加音频

我在这个游戏里添加了一些音频。在DoodleDrop07项目的Resources文件夹中，你可以找到两个音频文件：blues.mp3和aliensfx.caf。在cocos2d里播放音频最好也是最简单的方式是使用SimpleAudioEngine。音频支持不是cocos2d自带的功能；它来自CocosDenshion，和物理引擎一样是cocos2d的第三方插件。出于这个原因，你必须在使用音频功能的地方包含相应的CocosDenshion头文件，如下所示：

```
#import "GameScene.h"
#import "SimpleAudioEngine.h"
```

以下代码展示了如何使用SimpleAudioEngine播放音乐和音频：

```
[[SimpleAudioEngine sharedEngine] playBackgroundMusic:@"blues.mp3" loop:YES];
[[SimpleAudioEngine sharedEngine] playEffect:@"alien-sfx.caf"];
```

如果播放音乐，MP3是最好的选择。你只能一次播放一个MP3背景音乐。从技术上来说，有可能同时播放两个或两个以上的MP3文件，但是只有一个MP3文件可以通过硬件来解码。这样就会导致使用CPU来替其它MP3解码。对于游戏来说，这会给系统造成很大的压力。所以绝大多数情况下不应该同时播放多个MP3文件。

如果播放音效文件的话，我建议使用CAF格式。这是我唯一使用过的能够得到好效果的格式。

如果你想快速转换音频文件格式，同时改变一些基本的音频设置比如采样率（sampling rate），我建议你使用SoundConverter。它由Steve Dekorte开发。500KB以下的音频文件转换是免费的，如果你想没有任何限制的使用，你也只需要支付15美元。你可以通过以下地址下载SoundConverter：

http://dekorte.com/projects/shareware/SoundConverter/

注:如果你碰到不能在iOS程序里正常播放的音频文件，试着重新保存一下。因为每个音频软件都会生成同一个格式，但稍微有些区别的文件。一般通过重新保存，你可以解决在iOS设备里无法播放的问题。

移植到iPad

如果游戏中所有坐标尺寸都考虑到了设备的屏幕大小的话，当你把游戏移植到iPad上时，它会自动放大相应的尺寸。不过，如果你使用的都是固定尺寸的话，你的工作量就大了。

你可以很容易的将iPhone项目移植到iPad上。在Groups&Files面板选择你需要

移植的目标（target），然后选择Project ➤ Upgrade Current Target for iPad...，如图4-8所示的对话框会弹出：

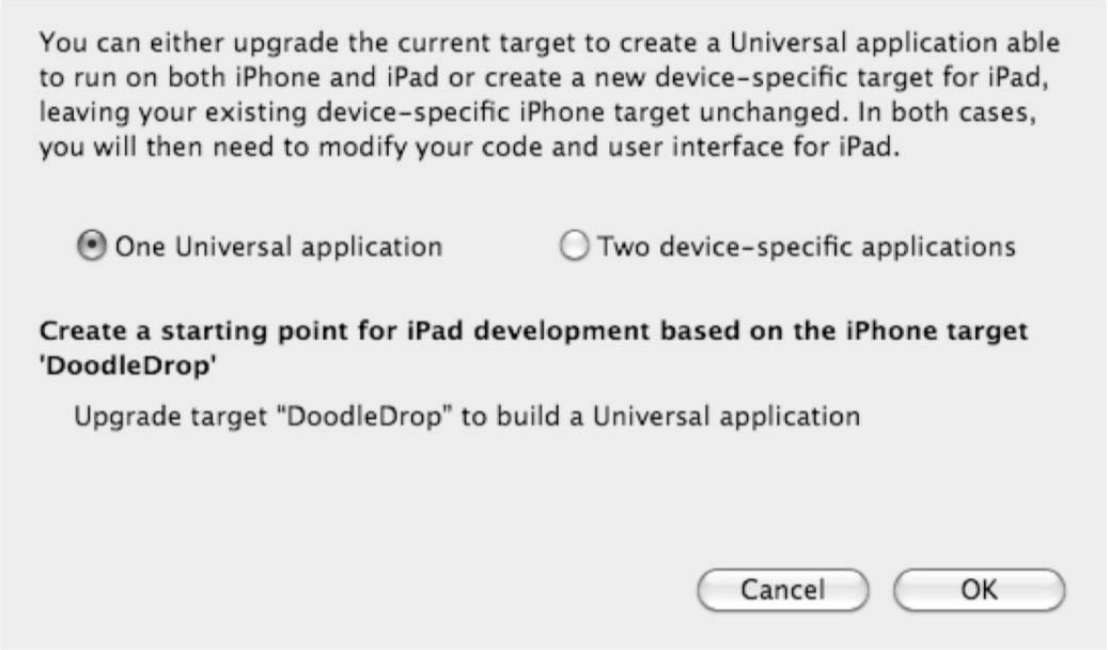

图4-8. 移植到iPad对话框给了你两个选择。简单点说，Universal Application可以给你节省点时间，但是你必须将iPhone和iPad所用到的素材放到同一个目标里面，从而增加了程序文件的下载尺寸。

对于一个简单的游戏来说，默认的Universal application是个不错的选择。虽然会增加程序文件的大小，但是程序在iPhone和iPad上都可以运行。

另外一个选择允许你把iPhone和iPad的素材分开放置，但是你需要维护两个应用程序，而且你必须分开提交到App Store。更重要的是，如果一个iTunes用户同时需要iPhone和iPad版本的话，他将不得不购买两次。

我选择了One Universal application选项。在iOS设备上，程序会自动检查连接的是哪一种设备，然后运行相应的版本。如果你想在iPad模拟器上运行，如图4-9，选择iPad Simulator3.2作为Active Executable：

图4-9. 要在iPad模拟器上运行universal app，将iPad Simulator 3.2设为Active Executable

结语

我希望你在制作第一个游戏的过程中感到快乐。我们在制作过程中学到了很多东西。

到目前为止，你学习了如何创建游戏层的类，并且学习了如何使用精灵。你也学到了如何使用加速计来控制主角精灵的移动，添加速度变量让主角精灵加速和减速，让它的动作看上去更加真实。

我也介绍了没有文档支持的CCArray类，这是cocos2d用来替换NSMutableArray的类。当你需要保存一组数据的时候，你应该首选CCArray。另一个没有文档支持的类是CGPointExtensions，它的一个距离测试方法被用于简单的径向碰撞测试。

最后的甜点是标签，bitmap字体和用于生成bitmap字体文件的Hiero Bitmap Font Tool。外加一点音频编程。

还剩下什么没有做呢？你可以完整地读一遍本章的源代码。在提供的源代码里我增加了一些游戏设置，开始菜单和游戏结束信息。

对于DoodleDrop项目，我还要提及一件事：所有代码都在一个类里面。对于一个小项目来说，这没有什么问题。但是随着你在代码里实现更多的功能，你的代码会变得越来越凌乱。你需要在代码里创建一些结构，来更好的组织代码。在下一章里我会教你cocos2d编程的最佳实践，如何组织代码，还有如何让处于不同类里的对象互相传递信息。