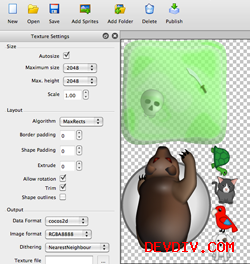
**在cocos2d里面如何使用Texture Packer和像素格式来优化spritesheet by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

前言：这篇文章是我翻译的第一篇关于cocos2d的文章，我在翻译的时候尽量按原文意思来，但难免会加入自己一些理解进去。还有这篇文章所采用的xcode版本是3.2.5，用xcode4.0的朋友可能实现起来有点出入，我会在文章的最后给出解决办法。

文章原来出处：<http://www.raywenderlich.com/2361/how-to-create-and-optimize-sprite-sheets-in-cocos2d-with-texture-packer-and-pixel-formats>

在cocos2d里面，为了使你的游戏获得最佳性能，你需要把许多小的sprite图片组合到一张大图里面，这张大图就叫做sprite sheet。

如果你使用cocos2d已经有一段时间了的话，你可能已经使用过了一款叫做[Zwoptex](http://zwoptexapp.com/)的工具来帮你生成sprite sheet。Zwoptex是一个非常棒的工具–我在我的很多程序里面使用它，而且确实帮我节省了很多时间。

然后，这里有一款新的工具，叫做Texture Packer.它类似于Zwoptex，也能创建sprite sheet，但是它还有一些非常方便的、很神奇的特性。

这篇文章将以一种教程的形式，讲述如何在cocos2d游戏开发使用Texture　Packer，同时，你还将学习到如何使用像素格式（pixel formats）、Texture Packer 如何智能地让你的游戏加载速度更快，运行更流畅，而且还能够在游戏界面看起来很不错的前提下使用尽可能少的内存。

澄清：我在发博客之前就从Texture Packer这个工具的作者手中拿到了license key。我当时并没有保证我会发一篇博文来回报他，\_但是，在我使用这个工具一段时间之后，它确实为我的应用程序减少了很多需要加载的内存，因此我爱上了这款工具。所以，我想让你们都了解它。

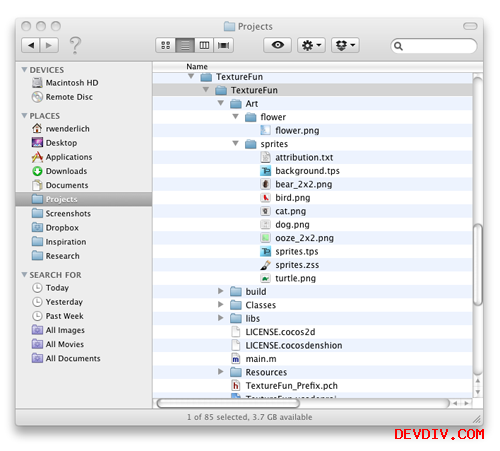
这篇教程是为那些熟悉cocos2d的人写的。如果你对cocos2d完全陌生的话，你应该从[“怎样做一个简单的iphone应用程序”](http://www.raywenderlich.com/1797/how-to-create-a-simple-iphone-app-tutorial-part-1)系列和[其它一些cocos2d的教程](http://www.raywenderlich.com/tag/cocos2d)开始。（目前这些链接依然是e文，但随着我翻译进度的前进，这些也会相应的更新。）

**开始**

首先，确保你安装了最新版本的cocos2d（在写作这篇文章的时候，版本号是 [Cocos2D v0.99.5-rc1](http://www.cocos2d-iphone.org/forum/topic/11150)，在翻译这篇文章的时候，已经是 [cocos2d-iphone-1.0.0-beta.tar.gz](http://cocos2d-iphone.googlecode.com/files/cocos2d-iphone-1.0.0-beta.tar.gz)了）。获得最新版本非常重要，因为新版本里面增加了对一些新的图片格式的支持，而这恰恰是这篇文章后面要用到的。

一旦你安装完之后，你就可以启动XCode，然后使用cocos2d应用程序模板来新建一个工程并把它命名为TextureFun。

下一步，你需要一些图片来制作sprite　sheets。你可以下载这些我收集来的[样例图片](http://www.cnblogs.com/andyque/admin/sample%20artwork)，然后解压缩，并把整个解压缩后的目录拖到你的TextureFun工程的一个子文件夹下面，如下图所示：



好，现在你拥有一个工程模板和一些样例图片以，是时候使用TexturePacker来制作spritesheet了！

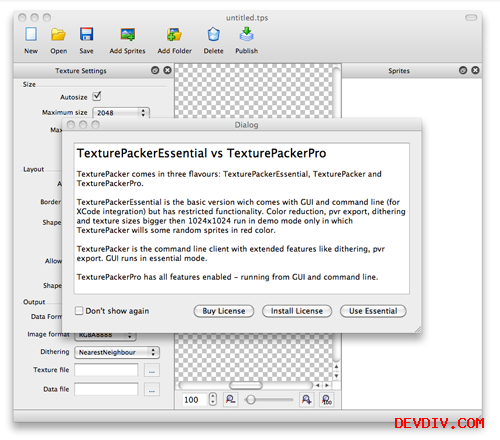
**使用TexturePacker来创建spritesheet**

你需要做的第一件事情就是下载 [Texture Packer](http://texturepacker.com/download/)的免费版本。请注意，你并不需要购买任何东西，对于这篇教程来说你只需要免费版本就足够了。

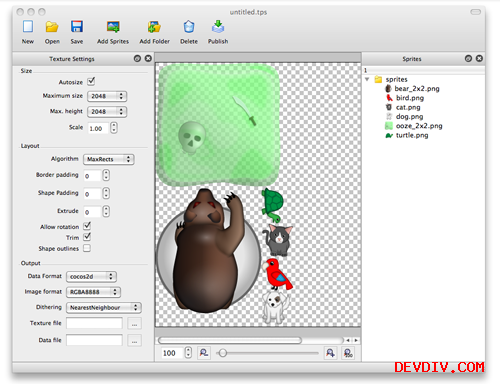
当你下载完之后双击，然后点击“TexturePacker.mpkg”，接下来就会弹出一个窗口，然后按照提示一步步安装在你的mac上。



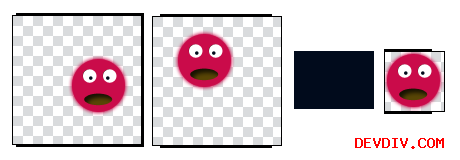
在你完成安装之后，在你的Application文件夹下面找到它并运行。当你看到第一个提示窗口出现的时候，选择“ Use Essentia”（免费版本）继续。



现在，点击工具栏上的“Add　Folder”按钮，然后选择TextureFun\Art\sprites文件夹。Texture　Packer将会加载图片并且智能地把这些图片布局在spritesheet中，如下图所示：



另一方面，你能够看到所有导入到纹理集（Texture　Atlas）中的图片，当你选中某一个的时候会看到有一个边框–另一个非常方便的功能！你也能够把鼠标停留在精灵（sprite）上面，看它是否创建走样（alias）（走样（alias）指那些被裁剪（cropping）之后的图像看起来实际上是一样的）。如下图所示：



顺便说一下通过点击“Add　Folder”按钮来添加图片的一些注意事项。首先，当你像这样通过增加文件夹的方式来增加精灵后，Texture　Packer并不是对每一个精灵增加一个索引（reference），而是对整个文件夹增加了一个索引。这意味着，当你以后向这个文件夹中增加精灵之后，下一次你运行Texture　Packer的时候，它会重新根据文件夹下面的所有的精灵来创建spritesheet–多么方便啊！

同时，你也不需要非得把所有的精灵都放在同一个根文件夹下面，你可以按照自己的方式把这些精灵放在不同的子目录下（比如sprites\animals，sprites\monsters），之后当你从cocos2d中引用它们的时候只需指定相对路径即可。

最后，需要注意的是，你可以包含多个精灵文件夹–这也是一个非常方便的功能，尤其是在你的游戏非常大的时候，你可以为每一个关卡制作一个spritesheet。

好了，现在让我们来看一下软件左边的一些选项。通过这些选项，你能够配置spritesheet的大小、布局和输出格式。首先，让我们来快速浏览一遍控制大小和布局的选项：

* **Autosize** （默认） – 这个选项会为你的spritesheet挑选最小的２的指数倍的大小。这是一个非常方便的特性，因为它可以省去你自己去计算spritesheet大小的时间。
* **Min/max size** 让你为你的spritesheet指定一个最大值。如果你想设置特定大小的spritesheet的时候，这也是一个非常方便的特性。（因为对于特定的设备来说，你可不想超过设备能够支持的最大限制，比如2代touch最大支持texture大小为1024\*1024）
* **Scale** 让你可以保存一个比原始图片尺寸要大一点、或者小一点的spritesheet。比如，如果你想在spritesheet中加载“2x”的图片（也即为Retina-display设备或者ipad创建的）。但是你同时也想为不支持高清显示的iphone和touch制作spritesheet，这时候只需要设置scale为０.５就可以了。也就是说，只需要美工提供高清显示的图片，用这个软件可以自己为你生成高清和普清的图片。
* **Algorithm** TexturePacker里面目前唯一支持的算法就是MaxRects，即按精灵尺寸大小排列，但是这个算法效果非常好，因此你不用管它。
* **Border/shape padding** 即在spritesheet里面，设置精灵与精灵之间的间隔。如果你在你的游戏当中看到精灵的旁边有一些“杂图”的时候，你就可以增加这个精灵之间的间隔。
* **Extrude** 精灵边界的重复像素个数. 这个与间隔是相对应的–如果你在你的精灵的边边上看到一些透明的小点点，你就可以通过把这个值设设置大一点。
* **Trim** 通过移除精灵四周的透明区域使之更好地放在spritesheet中去。不要担心，这些透明的区域仅仅是为了使spritesheet里面的精灵紧凑一点。–当你从cocos2d里面去读取这些精灵的时候，这些透明区域仍然在寻里。（因为，有些情况下，你可能需要这些信息来确定精灵的位置）
* **Shape outlines** 把这个选项打开，那么就能看到精灵的边边。这在调试的时候非常有用。

对于spritesheet来说，上面提到的各个选项的默认值，你一个也不需要改变–因为它们本来就已经很好了。然后，在输出部分，你需要改变一些设置。但是在讲到那个之前，让我们先谈一谈cocos2d中的像素格式。

**cocos2d和像素格式**

在cocos2d里面，理解像素格式非常重要。因为，像素格式会影响在你的游戏中加载一张图片到底需要多少内存。因为游戏通常要加载大量的图片资源，所以你要尽可能充分利用移动设备上面非常少的可用物理内存。

默认情况下面，当你在cocos2d里面加载一张图片的时候，对于每一个像素点使用４个byte来表示–１个byte（８位）代表red，另外３个byte分别代表green、blue和alpha透明通道。这个就简称RGBA8888。

因此，如果你使用默认的像素格式来加载图片的话，你可以通过下面的公式来计算出将要消耗多少内存来加载：

图像宽度（width）×图像高度（height）×每一个像素的位数（bytes　per　pixel）　=　内存大小

此时，如果你有一张５１２×５１２的图片，那么当你使用默认的像素格式去加载它的话，那么将耗费

５１２×５１２×４=１MB（好多啊！）

这里，我们以Iphone３G为例。它总共只有１２８兆内存，但是系统就要占掉一大半，还有其它一些程序也要使用一些内存，实际可用的内存更少。对于单独一张spritesheet来说那确实足够了。可是想像一下你有许许多多的spritesheet，而且游戏里面经常需要大量的spritesheet！

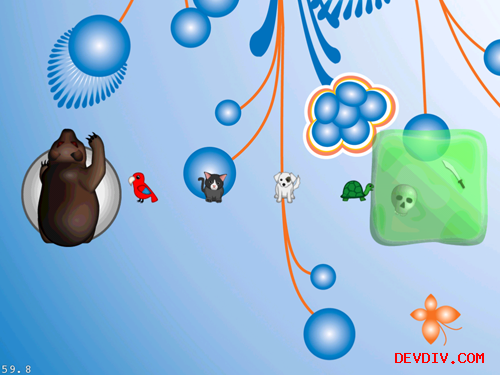
这里就需要让像素格式来帮忙了。你可以为图片的每个像素点指定更小的字节来保存图片。（比如每个像素点２个字节，即每个像素点１６位），这种方式就能够在图片质量和内存消耗之间取得一个很好的平衡点。

通常，你是在你的游戏看起来还ok的提前下，尽可能少地使用内存。背景图片就非常适用用８位或者１６位来存储，而精灵则一般要用１６位或者３２位。对于更多可选的像素格式和适用的场合，你可以参考Riq（cocos2d的作者）的一篇文章： [understanding pixel format guide](http://www.cocos2d-iphone.org/archives/61).（理解像素格式向导）

顺便说一下，如果你注意看窗口的右下角，你会看到Texture　Packer会基于你当前选择的像素格式计算出这张spritesheet所消耗的内存大小，因此你不必手动计算了。：）

**像素格式和抖动**

很多时候，当你使用较小的像素格式来加载图片的时候，你会发现图片的质量也在相应的降低。这时你会看到图像存在许多颜色的梯度变化。这里有一个例子，展示了当你使用像素格式RBGA4444去显示一张图片的时候会是什么样子：

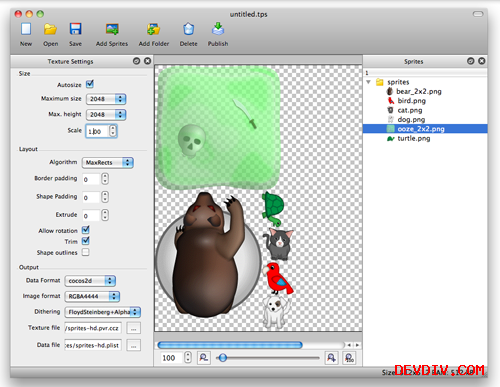


看到没有，图像上面有许许多多的“条条”和颜色梯度变化，特别是熊和绿色的框框那里。

这时，你可能想重新设计你的图片来确保使用更少的梯度，或者使用大一点的像素格式。但是，在这里，TexturePacker实现了另外一个杀手锏功能–图像抖动。

当你使用TexturePacker来保存spritesheet的时候，你可以指定目标像素格式为RGBA4444，然后选择“dithering　method”。这个默认选项会修改一些你的图像的颜色，但是当有梯度变化或者其它一些会带来问题的颜色以后，图像看起来就会非常糟糕。

继续，我们为spritesheet选择RBGA4444格式，然后改变抖动选项为“FloydSteinberg＋Alpha”。Texture　Packer将会在动态修改你的图片，而且马上显示出效果来。和上面的图片相比，是不是好看多了？



现在让我们保存这个spritesheet。点击Texture　file旁边的“…”，在弹出的对话框中选择TextureFun\Resouces目录，然后命名为“sprites-hd.pvr.ccz”。然后，TexturePack会自动为我们在Data　file那里生成相应的plist文件路径。并且会命名为“sprites-hd.plist”，这个名字是根据前面你提要的名字来命名的。

“但是，等一下！”，你可能会说，“为什么是pvr.ccz？！”。好吧，我很高兴你会这样问。。。

**PVRs和压缩**

PVR图像是专门为ios设备上面的PowerVR图形芯片指定的图像容器。它们在ios设备上非常好用，因为可以直接加载到显卡上面，而不需要经过中间的转化。

PVR图像也可以包含许多种不同像素格式的图像数据。之前，cocos2d仅仅支持一些用  sdk指定的 [texturetool app](http://developer.apple.com/library/ios/#documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/TextureTool/TextureTool.html)来创建的[图片格式](http://www.cocos2d-iphone.org/forum/topic/5984)，不过后来cocos2d已经[扩展了许多格式](http://www.cocos2d-iphone.org/forum/topic/8073)了。

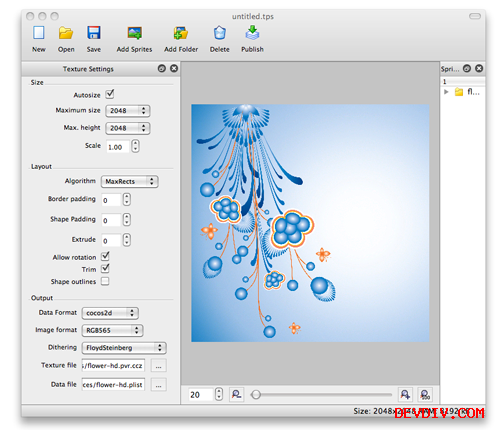
而且，最近cocos2d更新到了可以支持压缩了的pvr图像格式pvr.ccz。使用这种图片格式的好处有两点：一、可以使你的应用程序更小，因为图片是压缩过了的。二、你的游戏能够启动地更快。

总而言之，对于spritesheet来说，你可能通过指定１６位的像素格式来减少内存消耗，同时保存为pvr.ccz格式来使程序加载速度更快。最后，点击“Publish”按钮，你的spritesheet和属性列表文件就生成好了。Teture　Packer会提示你，一些精灵将会创建成红色（因为你使用的是免费版本）。

**优化背景图片**

现在，让我们也来加载并优化一下我们的背景图片。点击new创建一个新的Texture　Packer窗口，然后点击“Add　Folder”，并且选择“TextureFun\Art\flower”文件夹。

把图片格式改成RBG565（对于大的图片来说，你可能需要更好的质量），然后改变抖动方法为“FloydSteinberg”（为什么不是FloydSteinberg＋Alpha呢？因为像素格式是RBG565，没有了Alpha通道）。然后指定保存texture　file的路径为“TextureFun\Resouces\flower-hd.pvr.ccz”。最后，点击“Publish”，关闭警告信息，这时你的屏幕看起来会是下面这样：



**在cocos2d里面使用spritesheet**

现在回到我们的项目，右键点击Resources，然后选择“Add\Existing　Files…”。选择flower-hd.plist， lower-hd.pvr.ccz, sprites-hd.plist, and sprites-hd.pvr.ccz。同时，确保没有选中“ Copy items into destination group’s folder (if needed)”,然后单击完成。

下一步，打开HelloWorldScene.m，并且用下面的代码替换掉你的init方法里的内容：

-(id) init

{

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

CGSize winSize = [CCDirector sharedDirector].winSize;

[CCTexture2D setDefaultAlphaPixelFormat:kCCTexture2DPixelFormat\_RGB565]; CCSprite \* background = [CCSprite spriteWithFile:@"flower-hd.pvr.ccz"]; background.anchorPoint = ccp(0,0); [self addChild:background];

// More coming here soon… } return self; }

你需要做的第一件事情就是加载背景图片。首先，你告诉cocos2d使用RBG565的像素格式（你正在为你的背景图片的每个像素使用８位），然后调用spriteWithFile从磁盘上加载pvr.ccz格式的图片。注意，这里你并不需要把它当作一个spritesheet（比如，加载plist文件），因为这里“spritesheet”就只有一张图片。

注意，其实你在加载pvr.ccz格式的文件的时候并不需要指定像素格式，因为这个文件格式本身就包含了这样一些信息。但是，我们还是显示地在这里指　定了像素格式，因为如果你加载png格式的图片的话，（png格式图片总是保存为每个像素３２位，尽管你可能会使用不同的像素格式把它加载到内存里）。

下面，让我们在“more　coming　here　soon”注释的地方添加下面的代码：

[CCTexture2D setDefaultAlphaPixelFormat:kCCTexture2DPixelFormat\_RGBA4444];

CCSpriteBatchNode \*spritesBgNode;

spritesBgNode = [CCSpriteBatchNode batchNodeWithFile:@"sprites-hd.pvr.ccz"];

[self addChild:spritesBgNode];

[[CCSpriteFrameCache sharedSpriteFrameCache] addSpriteFramesWithFile:@”sprites-hd.plist”];

这里把像素格式设置为RBGA4444（你为主精灵所使用的每个像素点１６位的像素格式），然后为这个spritesheet创建一个batch　node。你也需要加载plist文件，把每一个精灵对应的帧（frame）加载到精灵帧缓冲区（sprite　frame　cache）中。

最后，紧接着上面加入下面的代码：

NSArray \*images = [NSArray arrayWithObjects:@"bear\_2x2.png", @"bird.png", @"cat.png", @"dog.png", @"turtle.png", @"ooze\_2x2.png", nil];

for(int i = 0; i < images.count; ++i) {

NSString \*image = [images objectAtIndex:i];

float offsetFraction = ((float)(i+1))/(images.count+1);

CGPoint spriteOffset = ccp(winSize.width\*offsetFraction, winSize.height/2);

CCSprite \*sprite = [CCSprite spriteWithSpriteFrameName:image];

sprite.position = spriteOffset;

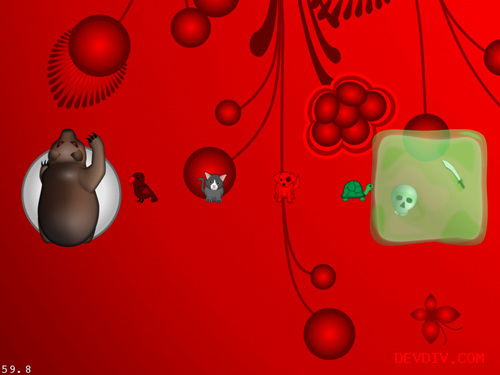
[spritesBgNode addChild:sprite];

}

[CCTexture2D setDefaultAlphaPixelFormat:kCCTexture2DPixelFormat\_Default];

这个循环遍历spritesheet中的所有的图片，并把他们合理地旋转在屏幕上面。

如果你使用iPad来编译并运行你的代码的话，你会得到下面的运行结果：



这个不就是你想实现的效果吗–记住，这里的红色仅仅因为你使用的是免费版本。（ps：正式版本也不贵，１００多块人民币可以搞定）

但是真正让人着迷的是那些你看不到的东西。

在背后，你的应用程序加载速度会明显比以前快很多。而且使用更少的内存，更让人心动的是，它看起来还是那么棒！而这些功能使用Texture　Packer都可以很容易的完成。

**不相信我？**

当我写这篇文章的时候，我做了一系列的简单的测试，从最好的情形到最坏的情形，来测试到底我的程序是如何运转的。下面是我得出的一些结论：

* **做最原始的事情。使用默认像素格式单个单个精灵地加载，不使用任何spritesheet。大约花费了０.７３秒钟加载，消耗大约２６兆内存。而且当你添加更多的精灵进去的时候，游戏就开始卡了。**
* **使用默认的像素格式，并且使用spritesheet：前进了一大步。这样会使游戏性能更好，同时也会减少内存消耗（因为你会把所有的精灵加载到一张大小的spritesheet的，而opengl使用纹理的大小都是２的指数幂，如果你一张精灵的大小是３２０×２００的话，那么opengl会创建５１２×５１２的纹理来加载精灵，这样就有很多空白的地方。实际上就是浪费了内存。）**
* **使用不含抖动的Zwoptex并保存为png格式，同时减少像素格式：这样可以大幅地减少内存消耗，大约只需要１５兆左右）。但是却增加了程序的启动时间，上升到大约１秒钟。我认为可能是由于不得不改变颜色缓冲的缘故吧。另外，图像显示的效果并没有在“像素格式和抖动”一节中的截屏效果那么好。**
* **使用通过Texture　Packer创建的抖动过的spritesheet并且保存为pvr.ccz格式：这在启动时间和显示效果上都前进了一大步！（启动时间大约只有０.３１秒左右，内存也只需要大约１７兆左右，我认为这可能是由于**[**内存泄漏**](http://www.cocos2d-iphone.org/forum/topic/11214#post-63675)**，这个问题在现在的版本中已经解决了。**

好了，如果你按照上面所讲的最佳实践来做的话，我想你在大部分情况下都会做得非常好。：）如果你想看看我写的测试程序，也想拿来跑一跑的话，[可以点此下载](http://www.raywenderlich.com/downloads/MemoryTest2.zip)。

**Texture Packer 和XCode集成**

当使用Texture　Packer的时候，你可以像这里介绍的一样使用GUI工具，但是你还可以把它集成到Xcode构建过程中去。这样你每一次编译的时候，它都会自动地（如果没有更改，就不会更新）为了更新spritesheet。

如果你过去用cocos2d写过游戏的话，你肯定明白一遍又一遍地重新生成你的spritesheet是那么的烦人！虽然每次可能都只需要几秒钟的时间，但是老是这样重复地做这样的事，确实很烦。

因此，让我们更方便地构建我们的工程吧–这里只需要花几秒钟时间，但是却可以为你以后节省大量的时间。右键点击“Resources”，选择“Add\New　File…“，然后选择　Mac　OS　X\Other\Shell Script，然后选择下一点。并命名为PackTextures.sh，单击完成。

然后使用下面的代码替换掉PackTextures.sh里面的内容：

#!/bin/sh

TP=”/usr/local/bin/TexturePacker”

if [ "${ACTION}" = "clean" ] then echo ”cleaning…”

rm resources/sprites-hd.pvr.ccz rm resources/sprites-hd.plist

rm resources/flower-hd.pvr.ccz rm resources/flower-hd.plist

# …. # add all files to be removed in clean phase # …. else echo ”building…”

# create hd assets ${TP} –smart-update \ –format cocos2d \ –data resources/sprites-hd.plist \ –sheet resources/sprites-hd.pvr.ccz \ –dither-fs-alpha \ –opt RGBA4444 \ Art/sprites/\*.png

${TP} –smart-update \ –format cocos2d \ –data resources/flower-hd.plist \ –sheet resources/flower-hd.pvr.ccz \ –dither-fs-alpha \ –opt RGB565 \ Art/flower/\*.png

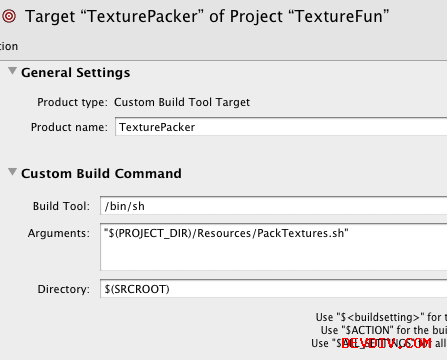
# …. # add other sheets to create here # …. fi exit 0

所有Texture　Packer　GUI界面能够做的事情，命令行工具也能做。如果你在命令行里面输入“TexturePacker”，你将会看到一系列它能够接收的参数说明。

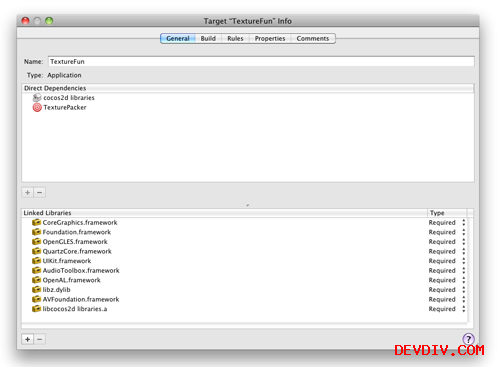
这个脚本仅仅通过运行TexturePacker来从你的Art目录下读取精灵文件并创建spritesheet–就像你之前用GUI工具所做的一样。你可以通过查看TexturePacker命令行工具帮助来获得更多有关每个参数具体的用法。

接下来，你需要让你的工程在编译的时间能够运行这个脚本。右键点击Targets，选择“Add\New　Target…”，然后选择“External　Target”（不是Shell　Script　Target），然后点击下一步，重命名这个Target为TexturePacker，最后点击Finish。

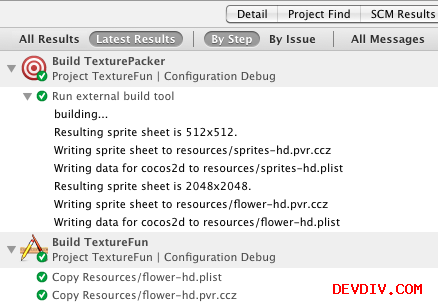
然后在你新建的target上面双击，然后把里面的内容设置成下图所示：



最后一步就是把这个target设置成你的程序的一个依赖。在TextureFun　target上面双击，然后选择General　tab，再在Direct　Dependencies窗口下面选择＋按钮。然后从列表中选择TexturePacker，最后单击Add　Target。如图下所示：



编译并运行你的程序，你将会从你的构建结果信息中看到Texture　Packer的一些输出信息，由此来判断是否一切运转良好。



如果你看到这些输出信息的话，那么意味着如果你想要增加一些新的文件到spritesheet中，你只需要把这些新的文件拖到指定的文件下面（这里是Art目录），然后重新编译一下，那么就会自动生成新的spritesheet。相反，如果你要移除一些精灵图片，再重新编译一下，也ok。是不是非常方便？

**Texture Packer vs. Zwoptex**

首先，让我们再说一遍，我是Zwoptex的超级粉丝。我认为Robert已经做了一件非常好的事情来把这些东西整合到一起，而且我老实说，如果没有这个工具，cocos2d不会走到今天这一步！

然后，当我们来比较Texture　Packer和Zwoptex的特性的时候，Texture　Packer似乎包含了Zwoptex９０％的功能。但是，Texture　Packer有三个杀手锏级别的功能是Zwoptex所没有的。

* **抖动，抖动，抖动.** 噢，我是多么喜欢抖动啊！在过去，有时我想使用比较低的像素格式，但是我不能，因为它看起来效果很不好。但是Texture　Packer内置的抖动功能使得图片看起来还是那么棒，尽管此时的图片质量很低。
* **pvr.ccz 支持.** 我喜欢这个特性. 它确实能够使游戏启动得更快, 而且会使你的可执行程序更小. 现在我不用花很长时间来上传和下载我以前写的应用程序了，我可以很快的下载并更新。
* **命令行工具支持.** 一旦你花一点时间把Texture　Packer集成到你的Xcode中去，你将会热爱生活。它是如此地方便，特别是在整个开发过程中，美工对图片改来改去的时候。

尽管Zwoptex非常棒, 但是它目前为此还是没有提供我这里列举的这些功能特性。

Texture Packer 有一点点贵 ($17.95 vs. Zwoptex’s $14.95), 但是，我认为多花这点钱值得。而且就像  [Steffen Itterheim](http://www.learn-cocos2d.com/tag/gui/)所说, ２个工具都有所长，都做了一件了不起的工作。

**接下来该怎么做呢？**

[这里](http://www.raywenderlich.com/downloads/TextureFun.zip)有上面的教程中所使用的示例代码。

你是一个Texture　Packer粉丝　还是一个Zwoptex粉丝呢？不妨在下面一起聊聊吧！或者你有其它一些很好的策略来高效地在cocos2d里面加载纹理的话，请让我知道。：）

ps：使用xcode4的朋友，主要需要注意的就是新增加的Target的参数的一些设置。因为xcode４生成的项目文件夹下并不是直接包含了所有的项目文件，而是一个工程文件和另外一个和项目名字一模一样的文件夹，然后所有的项目有关的源文件和资源文件都放在那个目录下，所以，我们增加的Target里的设置信息应该改成：

“$(PROJECT\_DIR)/TextureFun/Resources/PackTexture.sh”

$(SRCROOT)/TextureFun

其实可能需要注意的就是自动生成普清spritesheet，选中AutoSD选项就行了。

但是PackTexture.sh脚本里要相应增加

${TP} –smart-update \ –format cocos2d \ –data resources/sprites-hd.plist \ –sheet resources/sprites-hd.pvr.ccz \ –dither-fs-alpha \ –opt RGBA4444 \

–auto-sd \ #只要增加这一行就可以自动生成普清的图片了 Art/sprites/\*.png

还有一些软件的设置选项，大家可以参照软件自带的帮助文档。

如果大家在实践的过程中，遇到一些问题，可以留言。

**如何使用cocos2d来做一个简单的iphone游戏教程（第一部分） by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/352/how-to-make-a-simple-iphone-game-with-cocos2d-tutorial>

**游戏截图：**



cocos2d是一个非常强大的开源库，它可以为你开发iphone上面的游戏节省大量的时间。目前支持精灵（sprite）、动画、物理引擎、声音引擎以及许许多多非常酷的图像效果等等。

我也是刚开始学习cocos2d，目前已经有许多非常好的教程来教你如何使用cocos2d了。但是，那些都不是我想要的。我真正想要的是一个非常简单，但是可以跑起来的游戏。这个游戏包括怎么使用动画、碰撞检测和播放声音，这就够了，并不需要使用太多高级的特性。最终，我自己做了一个这样的游戏，因此，基于我的一些经验写了这篇教程，希望会对初学者有所帮助。

这篇教程将会从头至尾、一步一步地教你如何使用cocos2d来制作一个简单的iphone游戏。你可以按照教程一步步来，或者干脆直接跳到文章的最后，下载样例程序。没错！游戏里面有忍者。

**下载并安装cocos2d**

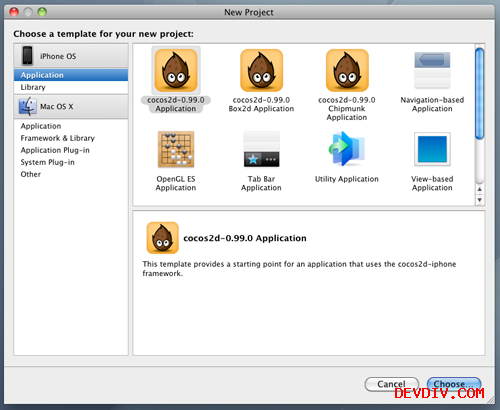
你可以从下面的链接来下载cocos2d的最新版本（以前是google，现在采用git了） [the Cocos2D Google Code page](http://code.google.com/p/cocos2d-iphone/).

在你下载完代码以后，你可能想安装非常有用的项目模板。打开终端窗口，然后cd到你的cocos2d解压缩目录下面，输入./install\_template.sh

请注意，你可以传递一些参数给这个安装脚本，比如你把xcode安装在了一个非标准的目录下面。（或者你有多个版本的sdk安装在你的机器上面）

**Hello, Cocos2D!**

让我们从最简单的HelloWorld项目开始吧！启动xcode，点file->new project，然后选择cocos2d　Application　template来创建一个新的cocos2d项目并把它命名为“Cocos2dSimpleGame”



继续，点编译并运行这个工程模板，如何一切都ok的话，那么你会看到下面的内容：



cocos2d是按照“场景”（scene）的概念组织的，对一个游戏来说，就好像某个关卡或者屏幕之类的。比如，你可能需要一个场景来为你的游戏建立初使化菜单界面，另外一个场景当作玩游戏的主要界面，还有一个游戏结束的时候的界面。在一个场景里面，你可以有许多“层”（layer）（这个和photoshop有点类似）。每一个层又可以包含一些结点，比如精灵、标签、菜单等。而且一个结点也可以包含其它的结点。（比如，一个精灵可以包含一个子精灵）

如果你看一下样例工程，你会看到只有一个场景HelloWorldScene–我们接下来将会在这个场景里面实现我们的游戏逻辑。继续打开它，你会在init方法里面看到，它把一个带有“Hello　World”字样的标签加到了当前场景中。我们将把这些代码去掉，并且放一个精灵在上面。

**增加一个精灵**

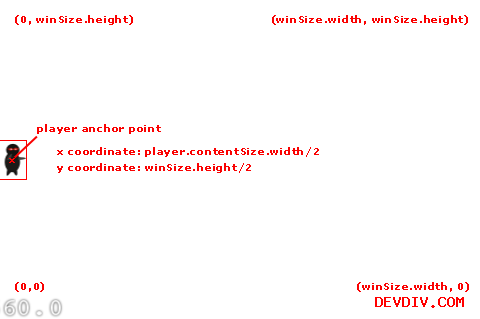
在我们增加一个精灵之前，我们需要一些图片。你可以自己去创建一些，或者使用我可爱的妻子为这个项目所创建的图片: [a Player image](http://www.raywenderlich.com/downloads/Player.png), [a Projectile image](http://www.raywenderlich.com/downloads/Projectile.png), and [a Target image](http://www.raywenderlich.com/downloads/Target.png).

一旦你获得了这些图片，你就可以把它们用鼠标拖到xcode的resource文件夹中去，并且保证“ Copy items into destination group’s folder (if needed)”勾上了。现在，我们拥有自己的图片了，我们先得计算一下，应该把player放在哪里。请注意，cocos2d坐标系的原点在屏幕的左下角，即（０，０）位置在左下角，这与一般的窗口坐标系统在左上角有所区别。x方向往右是正，y方向往上是正。由于项目是landscape（横版）模式，这就意味着右上角的坐标是（４８０，３２０）。

还有需要注意的是，当我们设置一个对象的位置的时候，这个位置是相对于所加精灵的中心点来加的。因此，如果我们想让我们的player精灵与屏幕的左边界水平对齐，并且垂直居中的话。

那么对于x轴：我们需要设置为［player sprite’s width]/2。y坐标设置为[window height]/2。

下面有一张图，可以更加清楚一些：



好，让他开始射击吧！打开Class文件夹并点击HelloWorldScene.m，然后把init方法用下面的代码替换掉：

-(id) init

{

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

CGSize winSize = [[CCDirector sharedDirector] winSize];

CCSprite \*player = [CCSprite spriteWithFile:@"Player.png"

rect:CGRectMake(0, 0, 27, 40)];

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

[self addChild:player];

}

return self;

}

编译并运行，你的精灵这时候应该出现在正确的位置上了。但是，这个前景默认是黑的。对于这张忍者图片来说，白色的背景可能看起来更好一些。在cocos2d里面，有一种非常简单的方式来改变层的背景颜色，那就是使用CCColoredLayer类。好，跟我来吧！点击HelloWorldScene.h，然后把HelloWorld接口声明改成下面这样：

@interface HelloWorld : CCColorLayer

然后点击HelloWorldScene.m文件，在init方法里面做一些小小的修改，以便使我们能把层的背景颜色改成白的：

if( (self=[super initWithColor:ccc4(255,255,255,255)] )) {

编译并运行，这时你将看到你的精灵是在一个白色的背景上面了。哈哈，我们的忍者看起来整装待发呢！



**移动目标**

接下来，我们想增加一些目标怪物来与我们的忍者战斗。为了使事情变得更加有趣，我想让这些目标可以移动–实际上这也并不是很难！因此，让我们先在屏幕的右边靠外一点点创建一些目标，然后设置一个action，并使之从右边移动到左边。

接下来，紧接着上面init函数，添加下面代码：

-(void)addTarget {CCSprite \*target = [CCSprite spriteWithFile:@"Target.png"

rect:CGRectMake(0, 0, 27, 40)];// Determine where to spawn the target along the Y axis

CGSize winSize = [[CCDirector sharedDirector] winSize];

int minY = target.contentSize.height/2;

int maxY = winSize.height - target.contentSize.height/2;

int rangeY = maxY - minY;

int actualY = (arc4random() % rangeY) + minY;// Create the target slightly off-screen along the right edge,

// and along a random position along the Y axis as calculated above

target.position = ccp(winSize.width + (target.contentSize.width/2), actualY);

[self addChild:target];// Determine speed of the target

int minDuration = 2.0;

int maxDuration = 4.0;

int rangeDuration = maxDuration - minDuration;

int actualDuration = (arc4random() % rangeDuration) + minDuration;// Create the actions

id actionMove = [CCMoveTo actionWithDuration:actualDuration

position:ccp(-target.contentSize.width/2, actualY)];

id actionMoveDone = [CCCallFuncN actionWithTarget:self

selector:@selector(spriteMoveFinished:)];

[target runAction:[CCSequence actions:actionMove, actionMoveDone, nil]];

}

在这里我将以一种非常啰嗦的形式来介绍，目的是方便大家理解。第一部分需要解释的是我们之前已经讨论过了的：我们做一些简单的计算来决定把对象放在什么位置，然后设置对象的position，然后并把它加在场景上面，就和加载player精灵一样。

这里增加的新的元素就是actions。cocos2d里面提供了许多非常方便的内置的action，你可以使用这样action来让你的精灵动起来。比如move　action，jump　action，fade　action，animation　action（就是播放图片序列）等等。这里，我们对目标对象使用了３种类型的action：

* **CCMoveTo:** 我们使用CCMoveTo　action让目标从屏幕右边一直往左移动，直到移出屏幕。注意，这里我们可以指定这个过程要花费多长时间。这里使用了变化的时间间隔２-４秒。
* **CCCallFuncN:** 它可以让你为某个执行此action的对象指定一个回调函数。我们指定的回调函数是：spriteMoveFinished—目前并没有，到后面会具体给了来。
* **CCSequence:** 它允许我们把一系列的action组成一个action序列，并且这些acton可以按顺序执行。一次执行完所有的action。在上面的例子中，我们让对象首先执行CcMoveTo，等CCMoveTo完成后，马上就会执行CCCallFuncN　action。

接下来, 为CCCallFuncN　action增加一个回调函数。你可以在addTarget前面增加下面的代码：

-(void)spriteMoveFinished:(id)sender {

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

[self removeChild:sprite cleanup:YES];

}

这个函数的目的是当精灵飞出屏幕之后，需要移除出当前的scene。这个非常重要，这样的话我们就不会因为屏幕外面积累太多没有用到的精灵而造成内存泄漏。注意，其实还有其它更好的方式来解决这个问题，比如使用一组可以重用的精灵等。不过，对于初学者来说，我在这篇教程里，尽量简单化。

在我们继续之前，还有最后一件事没做。我们需要调用这个方法来创建我们的目标怪物。而且，为了使事情变得更加有趣，我们会随着时间连续不断地发射一些怪物出来。我们可以使用cocos2d的定时scheduler，并指定一个回调函数来完成此功能。一秒钟调用一次回调函数就可以了。因此，在init函数返回之前，我们再加入下面的代码：

[self schedule:@selector(gameLogic:) interval:1.0];

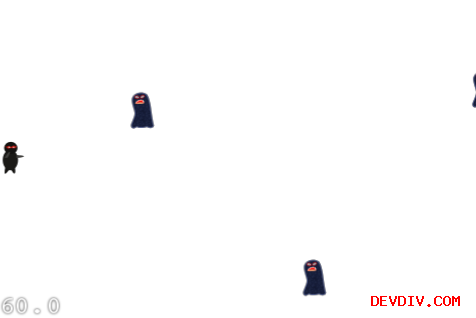
然后简单的实现一下这个回调函数，如下：

-(void)gameLogic:(ccTime)dt {

[self addTarget];

}

就这么多！现在编译再运行一下工作，你可以看到怪物在屏幕上面happy地移动了！

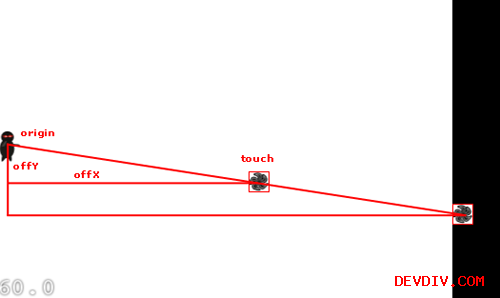


**发射飞盘**

在这里，我们的忍者需要有一些行动了–因此让我们增加一些射击吧！这里有许许多多实现射击的方式，但是在这个游戏里面，我们想让用户触摸一下屏幕，然后飞盘就会从player开始，沿着你触摸的位置发射出来。

我们使用CCMoveTo　action来实现这个功能。但是，为了使用这个功能，我们必须首先来做一些数学题。这是因为，CCMoveTo需要我们为飞盘指定目的地。但是我们又不能使用触摸点，因为触摸点仅仅代表飞盘飞的方向。我们实际上想让子弹超过触摸点，然后飞出屏幕之外去。

下面这张图解释了这个问题：



因此，就像你看到的，在触摸点和player之间有一个小的三角形，由origin点，offx和offy组成。我们只需要画一个更大的三角形，同时使用一样的比率就行了。然后我们就可以根据比例算出飞盘飞出屏幕的位置。

好了，让我们看看代码怎么写。首先我们需要让layer能接收touch事件。在你的init方法面添加下面一行代码：

self.isTouchEnabled = YES;

由于我们激活了layer的touch，因此我们能够接收到touch事件的回调。这里，我们实现ccTouchesEnded方法，这是在用户完成一次touch之后调用的，代码如下：

- (void)ccTouchesEnded:(NSSet \*)touches withEvent:(UIEvent \*)event {// Choose one of the touches to work with

UITouch \*touch = [touches anyObject];

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

location = [[CCDirector sharedDirector] convertToGL:location];// Set up initial location of projectile

CGSize winSize = [[CCDirector sharedDirector] winSize];

CCSprite \*projectile = [CCSprite spriteWithFile:@"Projectile.png"

rect:CGRectMake(0, 0, 20, 20)];

projectile.position = ccp(20, winSize.height/2);// Determine offset of location to projectile

int offX = location.x - projectile.position.x;

int offY = location.y - projectile.position.y;// Bail out if we are shooting down or backwards

if (offX <= 0) return;// Ok to add now – we’ve double checked position

[self addChild:projectile];

// Determine where we wish to shoot the projectile to int realX = winSize.width + (projectile.contentSize.width/2); float ratio = (float) offY / (float) offX; int realY = (realX \* ratio) + projectile.position.y; CGPoint realDest = ccp(realX, realY);

// Determine the length of how far we’re shooting int offRealX = realX - projectile.position.x; int offRealY = realY - projectile.position.y; float length = sqrtf((offRealX\*offRealX)+(offRealY\*offRealY)); float velocity = 480/1; // 480pixels/1sec float realMoveDuration = length/velocity;

// Move projectile to actual endpoint [projectile runAction:[CCSequence actions: [CCMoveTo actionWithDuration:realMoveDuration position:realDest], [CCCallFuncN actionWithTarget:self selector:@selector(spriteMoveFinished:)], nil]];

}

在第一部分，我们选择一个touch来处理，获得它在当前view中的位置，然后调用convertToGL把坐标转换成我们当前层的坐标系中去。这个非常重要，因为我们使用的是landscape模式。

接下来，我们加载飞盘精灵并且设置它的初始位置。然后，我们计算出它需要飞往何处，使用player和touch之间的向量并且根据前面描述的算法计算出来。

注意，这个算法并不完美。我们强迫子弹飞出屏幕x轴的外边–即使在它已经飞出屏幕y轴的外边界了。这里有许多方向来解决这个问题，比如检查飞出屏幕的最短距离，或者使用一个游戏回调函数来检查一个飞盘是否飞出，飞出就移出场景。但是，在这里，我们尽量保持简单。

最后一件事情就是，决定飞盘移动的时间。我们想让子弹以常量速度飞行，不管飞行方向如何。因此，我们不得不再做一点点数学。我们能够使用 [Pythagorean Theorem](http://en.wikipedia.org/wiki/Pythagorean_theorem)来计算我们移动了多久。记得几何学中，三角形的斜边=两个直角边的平方和再开根号。

一旦我们得到了距离，我们就可以通过除了速度来得到时间。因为速度=距离/时间。换句话说　时间=距离/速度。

余下的部分就和设置我们target一样了。编译并运行，现在忍者可以射击侵犯的敌人了！



**碰撞检测**

现在，我们可以看到飞镖到处乱飞了！但是，我们的忍者真正想做的，是能够放倒一些怪物。好吧，让我们增加一些代码来检测什么时候我们的飞镖与怪物相撞了。

在cocos2d里面，有许多方法可以解决这个问题，包括使用cocos2d内置的开源物理引擎box２d和chipmunk。然而，为了使事情变得简单一点，在这里我们自己实现了一个简单的碰撞检测。

为了实现这个，我们首先需要当前场景中存在的飞镖和怪物。在HelloWorldScene类里面增加下面的声明：

NSMutableArray \*\_targets;

NSMutableArray \*\_projectiles;

然后在init方法里面初使化这些数组：

\_targets = [[NSMutableArray alloc] init];

\_projectiles = [[NSMutableArray alloc] init];

我们还需要在dealloc函数里面做一些清理工作，防止内存泄漏：

[\_targets release];

\_targets = nil;

[\_projectiles release];

\_projectiles = nil;

现在，我们修改addTarget方法，把一个新的target加到targets数组里面，并且为这个target设置一个tag，以便将来使用：

target.tag = 1;

[\_targets addObject:target];

然后，修改ccTouchesEnded方法，同样的，把新增加的projectile加到projectiles数组里面，并为之设置一个tag供后面使用：

projectile.tag = 2;

[\_projectiles addObject:projectile];

最后，修改你的spriteMoveFinished方法，基于tag标签来从正确的数组中移除相应的sprite。

if (sprite.tag == 1) { // target

[\_targets removeObject:sprite];

} else if (sprite.tag == 2) { // projectile

[\_projectiles removeObject:sprite];

}

编译并运行程序，确保一切都ok。目前来说，应该没有什么可见的差别。但是，接下来我们就会去实现真正的碰撞检测了。

现在，在HelloWorldScene里面增加如下方法：

- (void)update:(ccTime)dt {NSMutableArray \*projectilesToDelete = [[NSMutableArray alloc] init];

for (CCSprite \*projectile in \_projectiles) {

CGRect projectileRect = CGRectMake(

projectile.position.x - (projectile.contentSize.width/2),

projectile.position.y - (projectile.contentSize.height/2),

projectile.contentSize.width,

projectile.contentSize.height);NSMutableArray \*targetsToDelete = [[NSMutableArray alloc] init];

for (CCSprite \*target in \_targets) {

CGRect targetRect = CGRectMake(

target.position.x - (target.contentSize.width/2),

target.position.y - (target.contentSize.height/2),

target.contentSize.width,

target.contentSize.height);if (CGRectIntersectsRect(projectileRect, targetRect)) {

[targetsToDelete addObject:target];

}

}for (CCSprite \*target in targetsToDelete) {

[\_targets removeObject:target];

[self removeChild:target cleanup:YES];

}if (targetsToDelete.count > 0) {

[projectilesToDelete addObject:projectile];

}

[targetsToDelete release];

}

for (CCSprite \*projectile in projectilesToDelete) { [\_projectiles removeObject:projectile]; [self removeChild:projectile cleanup:YES]; } [projectilesToDelete release]; }

上面的代码应该非常清楚。我们仅仅通过遍历projectiles和targets数组，为每个projectile和target创建边界矩形，然后使用CGRectIntersectsRect来检测碰撞。如果发现有碰撞了，我们就从场景中移除精灵，同时也把它移除出数组。注意，我们不得不添加一个toDelete数组，因为我们不能在遍历一个数组的时候去删除数组中的对象。当然，还有许多方式可以实现类似的逻辑，我只不过挑选了简单的方法。

在你真正完成之前，还差最后一件事情。在你的init方法里面调用下面的函数：

[self schedule:@selector(update:)];

编译并运行，现在，当你的飞镖和怪物相碰的时候，他们都会消失啦！

**完成触摸事件**

我们离制作一个可以玩的游戏（但是非常简单）的目标已经越来越近了。我们仅仅需要增加一些音效和背景音乐（试想哪个游戏没有声音呢！），再增加一点点简单的逻辑就更好了。

如果你之前看我的博文[《关于iphone上面的音效编程》](http://www.raywenderlich.com/204/audio-101-for-iphone-developers-file-and-data-formats)的话，你将会非常高兴，因为使用cocos2d向游戏里音效和背景音乐实在是太简单了。

首先，把一些背景音乐和音效拖到工程的resource文件夹中。你可以使用 [cool background music I made](http://www.raywenderlich.com/downloads/background-music-aac.caf.zip) 或者我的 [awesome pew-pew sound effect](http://www.raywenderlich.com/downloads/pew-pew-lei.caf.zip),或者自制一些。

然后，在HelloWorldScene.m文件里导入下面的头文件：

#import ”SimpleAudioEngine.h”

在你的init方法里加载背景音乐：

[[SimpleAudioEngine sharedEngine] playBackgroundMusic:@”background-music-aac.caf”];

然后，在你的ccTouchesEnded方法里面添加音效代码：

[[SimpleAudioEngine sharedEngine] playEffect:@”pew-pew-lei.caf”];

现在，让我们创建一个新的场景，来作为“You　Win”或者“You　Lose”的标志。右击Classes文件夹，然后选择File/New　File并选择Objective-c　class。同时，确保NSObject基类被选中。点击下一步，然后输入GameOverScene作为文件名，同时确保“Also　create　GameOverScene.h”复选框打上勾。

然后把GameOverScene.h里面的文件替换成下面的代码：

#import ”cocos2d.h”@interface GameOverLayer : CCColorLayer {

CCLabelTTF \*\_label;

}

@property (nonatomic, retain) CCLabelTTF \*label;

@end@interface GameOverScene : CCScene {

GameOverLayer \*\_layer;

}

@property (nonatomic, retain) GameOverLayer \*layer;

@end

接下来替换掉GameOverScene.m文件里的内容：

#import ”GameOverScene.h”

#import ”HelloWorldScene.h”@implementation GameOverScene

@synthesize layer = \_layer;- (id)init {if ((self = [super init])) {

self.layer = [GameOverLayer node];

[self addChild:\_layer];

}

return self;

}- (void)dealloc {

[\_layer release];

\_layer = nil;

[super dealloc];

}@end

@implementation GameOverLayer @synthesize label = \_label;

-(id) init { if( (self=[super initWithColor:ccc4(255,255,255,255)] )) {

CGSize winSize = [[CCDirector sharedDirector] winSize]; self.label = [CCLabelTTF labelWithString:@"" fontName:@"Arial" fontSize:32]; \_label.color = ccc3(0,0,0); \_label.position = ccp(winSize.width/2, winSize.height/2); [self addChild:\_label];

[self runAction:[CCSequence actions: [CCDelayTime actionWithDuration:3], [CCCallFunc actionWithTarget:self selector:@selector(gameOverDone)], nil]];

} return self; }

- (void)gameOverDone {

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

}

- (void)dealloc { [\_label release]; \_label = nil; [super dealloc]; }

@end

注意，这里有两个不同的对象：场景和层。场景可以包含任意数量的层，但是此例中只有一个层。这个层只是在屏幕的中间放置了一个label，然后运行了一个action。这个action的作用就是，等待３秒钟，然后调用一个回调函数切换回HelloWorld场景。

最后，让我们增加一些基本的游戏逻辑。首先，让我们来追踪player销毁的飞镖projectiles。接下来，在HelloWorld类里面增加一个成员变量，如下所示：

int \_projectilesDestroyed;

在HelloWorldScene.m里面，导入我们的GameOverScene类：

#import ”GameOverScene.h”

在update方法里，增加（销毁的projectile）计数，同时检测游戏胜利的条件。并在targetsToDelete循环里，紧接着removeChild：target的地方添加如下代码：

\_projectilesDestroyed++;

if (\_projectilesDestroyed > 30) {

GameOverScene \*gameOverScene = [GameOverScene node];

[gameOverScene.layer.label setString:@"You Win!"];

[[CCDirector sharedDirector] replaceScene:gameOverScene];

}

最后，让我们这样设计，只要有一个怪物穿过了屏幕左边，你就输了。修改spriteMoveFinished方法，通过在tag==１里面、removeChild：sprite后面添加下面的代码：

GameOverScene \*gameOverScene = [GameOverScene node];

[gameOverScene.layer.label setString:@"You Lose :["];

[[CCDirector sharedDirector] replaceScene:gameOverScene];

继续，编译并运行程序。现在，你的游戏可以实现胜利或者失败的场景了！：）

**获得源代码**

旁边是本教程使用的完整的源代码：[simple Cocos2D iPhone game](http://www.raywenderlich.com/downloads/Cocos2DSimpleGame.zip) 。

**何去何从？**

这个项目对于一个cocos2d的初学者来说非常有帮助，而且你还可以自己往项目里面添加更多新的特性。或许你可以尝试一下，添加一个提示框，提示当前你已经打中了多少个怪物了。或者你可以增加一些很酷的动画，比如怪物被击中后不是直接消失，而是用一段动画来模拟死去。（可以参考cocs2d　TestBed里面的ActionsTest，EffectsTest和EffectsAdvancedTest）。或者你还可以增加更多的图片和声音资源，或者更多的游戏逻辑。心情发挥吧！

如果你想继续这一系列的教程，可以期待一个我的下一篇翻译： [如何增加一个旋转的炮塔](http://www.cnblogs.com/andyque/articles/1997820.html)：）

如果你想了解更多关于cocos2d的内容，你也可以期待我的下面的一系列的翻译教程：

[在cocos2d里面如何制作按钮：简单按钮、单选按钮和开关按钮](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%9C%A8cocos2d%E9%87%8C%E9%9D%A2%E5%A6%82%E4%BD%95%E5%88%B6%E4%BD%9C%E6%8C%89%E9%92%AE%EF%BC%9A%E7%AE%80%E5%8D%95%E6%8C%89%E9%92%AE%E3%80%81%E5%8D%95%E9%80%89%E6%8C%89.html),

[Box2D入门教程](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%9C%A8cocos2d%E9%87%8C%E9%9D%A2%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8%E7%89%A9%E7%90%86%E5%BC%95%E6%93%8Ebox2d%EF%BC%9A%E5%BC%B9%E7%90%83.html)，

[如何使用box2d来制作一个简单的Breakout游戏](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E5%92%8Cbox2d%E6%9D%A5%E5%88%B6%E4%BD%9C%E4%B8%80%E4%B8%AAbreakout%E6%B8%B8%E6%88%8F%EF%BC%9A%E7%AC%AC%E4%B8%80%E9%83%A8%E5%88%86.html)

好了，翻译完了！如果你们有什么问题，或者很好的建议，或者学习cocos2d过程中的一些心得和体会，都拿出来，大家一起分享吧！

**如何使用cocos2d开发一个简单的iphone游戏：旋转炮塔。（第二部分） by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/692/rotating-turrets>

《[怎样使用cocos2d来开发一个简单的iphone游戏](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E6%9D%A5%E5%81%9A%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84iphone%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B1.html)》这个帖子太火了，你们当中的许多人都想要一些后续的教程！特别是，有些人问到我如何旋转炮塔来改变射击的方向。许多游戏都有这个功能，包括我最喜欢的一款游戏—-塔防！

因此，在这个教程中，我将会详细地讲解如何实现这个功能，即如何把旋转炮塔的功能添加到一个游戏当中去。在这里，特别要感谢Jason和Robert，是他们建议我来写这篇教程。

**准备工作**

如果你看完并实践了上一个教程，你可以继续使用那个工程。如果没有的话，那么下载[这个链接](http://www.raywenderlich.com/downloads/Cocos2DSimpleGame.zip)的代码吧。

接下来，下载新的 [player sprite](http://www.raywenderlich.com/downloads/Player2.png) 和 [projectile sprite](http://www.raywenderlich.com/downloads/Projectile2.png)图片，然后把它们加到工程里面，在这之前，先从工程里删除旧的Player.png和Projectile.png图片。然后，修改代码，把每个sprite添加进去。如下所示：

// In the init method

CCSprite \*player = [CCSprite spriteWithFile:@"Player2.png"];

// In the ccTouchesEnded method

CCSprite \*projectile = [CCSprite spriteWithFile:@"Projectile2.png"];

注意，这一次我们并没有指定精灵的宽度和高度，而是让cocos2d替我们来处理这些事情。

编译并运行你的工程，如果一切顺利的话，你将会看到一个炮塔正在发射子弹。然后，这并不是很好，因为炮塔在射击的时候并没有面朝那个方向。因此，接下来让我们来解决这个问题。



**旋转并射击**

在我们旋转炮塔之前，首先，我们需要保存Player精灵的引用，以便后面旋转它的时候使用。打开HelloWorldScene.h，然后修改类文件并包含以下成员变量：

CCSprite \*\_player;

然后修改init方法中的代码，把Player对象加入到层（layer）中。代码如下：

\_player = [[CCSprite spriteWithFile:@"Player2.png"] retain];

\_player.position = ccp(\_player.contentSize.width/2, winSize.height/2);

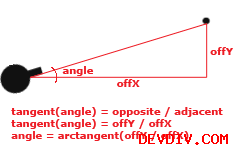
[self addChild:\_player];

最后，我们在dealloc函数里面添加一些清除代码。（这是个好习惯，初使化后就做相应的清理操作，防止忘记。）

[\_player release];

\_player = nil;

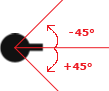
好了，现在让我们取出player对象的引用并且旋转它吧！为了旋转它，我们首先需要计算出旋转的角度。为了解决这个问题，想想我们在高中时候学过的三角代数吧。还记得sin　cos　tan吗？为了便于理解，下面使用一张图来解释一下：tan　=　对面/邻边。



如上所示，我们想要旋转的角度是arctangent（angle），即对offY/offX求arctangent运算。

然而，这里还有两件事情，我们需要放在心上。首先，当我们计算actangent（offY/offX）的时候，这个结果是弧度，但是cocos2d使用的却是角度。还好，cocosd2d提供了一个非常方便的宏，可以使得角度和弧度之间方便转化。

第二点，我们假定上面的图中angle的偏转是正２０度，但是，cocos2d里面顺时针方向为正（而不是上图所示的逆时针为正）。让我们看到下面这个图：



因此，为了得到正确的方向，我们把运算结果乘以一个-１就可以了。比如，如果我们把上面那幅图片里的角度乘以-１的话，我们就得够得到-２０度，这个角度其实就是逆时针方向的２０度。（感觉老外说话好啰嗦啊，聪明的读者恐怕早就明白了吧！：）

好了，讲得够多了！让我们来写一点代码吧。在ccTouchesEnded里面加入以下代码，添加位置在你的projectile　runAction之前。

// Determine angle to face

float angleRadians = atanf((float)offRealY / (float)offRealX);

float angleDegrees = CC\_RADIANS\_TO\_DEGREES(angleRadians);

float cocosAngle = -1 \* angleDegrees;

\_player.rotation = cocosAngle;

编译并运行工程，现在我们的炮塔在射击的时候可以改变方向了。

**旋转之后再射击**

目前来说还不错，但是有一点点怪。因为，这个炮塔好像突然一下跳到一个方向射击，有点不够流畅。我们可以解决这个问题，但是在这之前，我们需要重构一下代码。

首先，打开HelloWorldScene.h，然后在你的类里添加如下成员变量：

CCSprite \*\_nextProjectile;

然后，修改你的ccTouchesEnded方法，并且添加一个新的方法，叫做finishShoot，如下所示：

- (void)ccTouchesEnded:(NSSet \*)touches withEvent:(UIEvent \*)event {if (\_nextProjectile != nil) return;// Choose one of the touches to work with

UITouch \*touch = [touches anyObject];

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

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

// Set up initial location of projectile CGSize winSize = [[CCDirector sharedDirector] winSize]; \_nextProjectile = [[CCSprite spriteWithFile:@"Projectile2.png"] retain]; \_nextProjectile.position = ccp(20, winSize.height/2);

// Determine offset of location to projectile int offX = location.x - \_nextProjectile.position.x; int offY = location.y - \_nextProjectile.position.y;

// Bail out if we are shooting down or backwards if (offX <= 0) return;

// Play a sound! [[SimpleAudioEngine sharedEngine] playEffect:@”pew-pew-lei.caf”];

// Determine where we wish to shoot the projectile to int realX = winSize.width + (\_nextProjectile.contentSize.width/2); float ratio = (float) offY / (float) offX; int realY = (realX \* ratio) + \_nextProjectile.position.y; CGPoint realDest = ccp(realX, realY);

// Determine the length of how far we’re shooting int offRealX = realX - \_nextProjectile.position.x; int offRealY = realY - \_nextProjectile.position.y; float length = sqrtf((offRealX\*offRealX)+(offRealY\*offRealY)); float velocity = 480/1; // 480pixels/1sec float realMoveDuration = length/velocity;

// Determine angle to face float angleRadians = atanf((float)offRealY / (float)offRealX); float angleDegrees = CC\_RADIANS\_TO\_DEGREES(angleRadians); float cocosAngle = -1 \* angleDegrees; float rotateSpeed = 0.5 / M\_PI; // Would take 0.5 seconds to rotate 0.5 radians, or half a circle float rotateDuration = fabs(angleRadians \* rotateSpeed); [\_player runAction:[CCSequence actions: [CCRotateTo actionWithDuration:rotateDuration angle:cocosAngle], [CCCallFunc actionWithTarget:self selector:@selector(finishShoot)], nil]];

// Move projectile to actual endpoint [\_nextProjectile runAction:[CCSequence actions: [CCMoveTo actionWithDuration:realMoveDuration position:realDest], [CCCallFuncN actionWithTarget:self selector:@selector(spriteMoveFinished:)], nil]];

// Add to projectiles array \_nextProjectile.tag = 2;

}

- (void)finishShoot {

// Ok to add now – we’ve finished rotation! [self addChild:\_nextProjectile]; [\_projectiles addObject:\_nextProjectile];

// Release [\_nextProjectile release]; \_nextProjectile = nil;

}

这看上去好像有许多代码，但是，实际上我们改动的并不多–大部分只是做一些小小的重构。下面是我们所修改的内容的一个列表：

１.在函数开头检查nextProjectile的值是否为nil。这意味着我们当前的touch事件正发生在射击过程之中。也就是说，炮塔已经发射出一个子弹了。

２.之前，我们使用一个projectile的局部变量，并把它加入到了当前的场景中。在这个版本中，我们增加了一个nextProjectile的成员变量，但是并没有马上加到当前场景中。因为后要还要使用。

３.定义炮塔旋转的角度，半秒钟旋转半个圆。记住，一个圆有２　PI个弧度。

４.计算旋转特定的角度需要多长时间，这里是拿弧度乘以速度。

５.接下来，我们使用一个sequence　action来旋转我们的炮塔。最后，调用一个函数，把projectile加入到当前场景当中去。

好，大功告成！编译并运行工程，现在炮塔可以旋转，并且很流畅地射击了！

**接下来做什么？**

首先，这里有[《怎样使用cocos2d来开发简单的iphone游戏》](http://www.raywenderlich.com/downloads/Cocos2DSimpleGame2.zip)目前为止的完整代码。

接下来，在这个系列的教程中，我们教大家[如何添加更猛的怪物和更多的关卡！](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E6%9D%A5%E5%88%B6%E4%BD%9C%E7%AE%80%E5%8D%95%E7%9A%84iphone%E6%B8%B8%E6%88%8F%EF%BC%9A%E6%9B%B4%E7%8C%9B%E7%9A%84%E6%80%AA.html)

或者期待我接下来翻译的cocos2d和box２d方面的教程吧！

**如何使用cocos2d来制作简单的iphone游戏：更猛的怪物和更多的关卡。（第三部分。完！） by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/782/harder-monsters-and-more-levels>

目前为止，我正在翻译的游戏《[如何如何使用cocos2d来制作一个简单的iphone](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E6%9D%A5%E5%81%9A%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84iphone%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B1.html)》游戏非常cool。我们有一个可以旋转的炮塔，有怪物可以射杀，还有很棒的音效。

但是，我们的炮塔觉得这太简单了。这些怪物只要开一枪就挂了，而且现在只有一个关卡！它还没有热身呢！

在这个教程里，我将会扩展我们的工程，并增加一些不同种类和难度的怪物，然后实现多个关卡。



**更猛的怪物**

为了好玩，让我们创建两种不同类型的怪物：一种不怎么经打，但是移动速度很快，还有一种很能抗（坦克级别），但是移动速度很慢！为了使玩家可以区分这两种不同类型的怪物，下载修改的[怪物图片](http://www.raywenderlich.com/downloads/Target2.png)并把它们添加到工程里。同时，下载[我制作的爆炸音效](http://www.raywenderlich.com/downloads/explosion.caf.zip)，也把它们添加到工程中去。

好了，让我们来创建Monster类。这里有许多方法来为Monster类建模，但是，我们选择最简单的方式，即把Monster类当作CCSprite的一个子类。同时，我们会创建两个Monster类的子类：一个为我们的虚弱快速怪创建，另一个为我们的强悍缓慢怪创建。

选择File\New　File，再选择Cocoa　Touch　Class\Objective-C　class，确保Subclass　of　NSObject被选中。单击下一步，取名为Monster.m，然后确保“Also　create　Monster.h”复选上。

接下来，把Monster.h中的代码替换成下面的：

#import ”cocos2d.h”

@interface Monster : CCSprite { int \_curHp; int \_minMoveDuration; int \_maxMoveDuration; }

@property (nonatomic, assign) int hp; @property (nonatomic, assign) int minMoveDuration; @property (nonatomic, assign) int maxMoveDuration;

@end

@interface WeakAndFastMonster : Monster { } +(id)monster; @end

@interface StrongAndSlowMonster : Monster { } +(id)monster; @end

这里非常直白：我们从CCSprite派生一个Monster类，然后增加了一些成员变量来记录monster的状态。然后，我们又从Monster类派生出两个不同的monster子类。

现在，打开Monster.m并添加下面的代码：

#import ”Monster.h”

@implementation Monster

@synthesize hp = \_curHp; @synthesize minMoveDuration = \_minMoveDuration; @synthesize maxMoveDuration = \_maxMoveDuration;

@end

@implementation WeakAndFastMonster

+ (id)monster {

WeakAndFastMonster \*monster = nil; if ((monster = [[[super alloc] initWithFile:@”Target.png”] autorelease])) { monster.hp = 1; monster.minMoveDuration = 3; monster.maxMoveDuration = 5; } return monster;

}

@end

@implementation StrongAndSlowMonster

+ (id)monster {

StrongAndSlowMonster \*monster = nil; if ((monster = [[[super alloc] initWithFile:@”Target2.png”] autorelease])) { monster.hp = 3; monster.minMoveDuration = 6; monster.maxMoveDuration = 12; } return monster;

}

@end

这里代码很简单的，只有我们为每个类添加的一个静态方法，用来返回这个类的实例。然后初使化了默认的HP和移动所需要的时间。

现在，让我们把新创建的Monster类集成到之前的代码中去！首先在HelloWorldScene.m中导入文件：

#import ”Monster.h”

然后，修改addTarget方法来构造我们新创建的类的实例，而不是直接创建精灵（sprite）。替换spriteWithFile那一行，如下所示：

//CCSprite \*target = [CCSprite spriteWithFile:@"Target.png" rect:CGRectMake(0, 0, 27, 40)];

Monster \*target = nil;

if ((arc4random() % 2) == 0) {

target = [WeakAndFastMonster monster];

} else {

target = [StrongAndSlowMonster monster];

}

这里将会有５０％的机率来出现不同类型的monster。当然，我们把怪物的speed定义移到了类当中，因此，我们需要修改min/max移动间隔，把它改成下面的样子：

int minDuration = target.minMoveDuration; //2.0;

int maxDuration = target.maxMoveDuration; //4.0;

最后，在updateMethod里面做一些修改。首先，在targetsToDelete的声明之前，添加一个boolean值。

BOOL monsterHit = FALSE;

然后，在CGRectIntesectsRect里面，不是马上把对象添加到targetsToDelete里面，而是改成下面的：

//[targetsToDelete addObject:target];

monsterHit = TRUE;

Monster \*monster = (Monster \*)target;

monster.hp–;

if (monster.hp <= 0) {

[targetsToDelete addObject:target];

}

break;

这里，我们不是马上杀死怪物，而是减少它的HP，而且只有当它的生命值小于０的时候，才kill它。注意，如果projectile击中一个怪物的话 我们就跳出循环，这意味着一个飞盘射击一次只能打一个怪物。

最后，我们把projectilesToDelete测试改成下面所示：

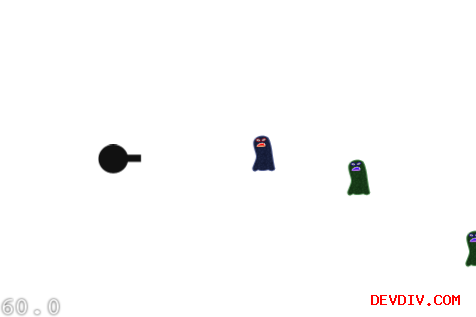
if (monsterHit) {

[projectilesToDelete addObject:projectile];

[[SimpleAudioEngine sharedEngine] playEffect:@”explosion.caf”];

}

编译并运行代码，如果一切顺利，那么你将会看到两种不同类型的怪物在屏幕上飞过—这使得我们的炮塔的生活更加富有挑战了！



**多个关卡**

为了使游戏支持多个关卡，首先我们需要重构。这个重构的工作非常简单，但是在这个项目里，有许多工作要做。如果把所有的内容都放在这个帖子上，那将会是一篇又长又乏味的帖子。

相反，我会从一个更高的角度来谈谈我做了什么，并且提供一个功能完整的样例工程。

抽象出一个Level类。目前，HelloWorldScene类里面把“level”的概念硬编码进去了，比如发射哪种类型的monster，发射频率如何等等。因此，我们的第一步就是要把这些信息提取出来，放到一个Level类里面。这样，在HelloWorldScene里面我们就可以为不同的关卡重用相同的逻辑。

重用场景。目前，我们每一次转换场景（scene）的时候都是重新创建了一个新的场景类。这里有一个缺点就是效率问题。每一次在场景对象的init方法里加载资源，这会影响游戏frame。

因为我们是一个简单的游戏，我们需要做的就是，每一个scene创建一个实例，并且提供一个reset方法来清除任何老的状态（比如上一关中的飞盘或者怪物）。

使用应用程序委托来当做跳板。目前，我们并没有任何全局的状态，比如：我们在哪一个关卡或者当前关卡的设置是什么。每一个场景仅仅是硬编码它需要跳转的下一个场景是谁。

我们将会修改这些内容，使用App　Delegate来存储指向一些全局状态（比如关卡信息）的指针。因为，所有的场景（scene）都可以很方便地得到delegate对象。我们也会在App　Delegate类里面放置一些方法，用来实现不同场景之间的切换的集中控制。并且减少场景之间的相互依赖。

好了，上面就是我所做的主要的重构内容—更多细节可查看[样例工程](http://www.raywenderlich.com/downloads/Cocos2DSimpleGame3.zip)。记住，这只是实现功能的方式之一，如果你有其它更好的组织场景和游戏对象的方法，请在这里分享出来吧！

不管怎么说，下载代码，运行看看吧。我们有一个非常不错的游戏了—-一个旋转的炮塔，成千上万的不同类型的敌人，多个关卡，win/lose场景，当然，还有很棒的音效！

**总结**

老样子，这里你可以下载到本系列教程目前为止[完整的源代码](http://www.raywenderlich.com/downloads/Cocos2DSimpleGame3.zip)。

现在，你知道如何制作一个简单的游戏了，为什么不更深入一点呢？学一学如[何使用cocos2d来制作基于tile的游戏](http://www.cnblogs.com/andyque/archive/2011/04/11/2012852.html)？毕竟，谁不喜欢忍者吃西瓜呢？

我希望你能喜欢这个系列的教程，希望这些教程能对你做项目有所帮助。

ps：接下来，我们尝试翻译一些有关box2d方面的教程，敬请期待！

**在cocos2d里面如何制作按钮：简单按钮、单选按钮和开关按钮 by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/414/how-to-create-buttons-in-cocos2d-simple-radio-and-toggle>

程序截图：



当你在使用cocos2d制作一个游戏的时候，你很可能会发现，你需要的第一个东西就是“按钮”。（比如游戏开始时的菜单选择界面等）这个教程将会一步步地教你如何使用cocos2d来创建按钮。刚开始创建简单的按钮，然后再介绍开关按钮和单选按钮。这篇教程假设你已经阅读了[《如何使用cocos2d来制作简单的iphone游戏》](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E6%9D%A5%E5%81%9A%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84iphone%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B1.html)这一系列的教程，或者具备同等相关经验。

当我第一次想要在cocos2d里面添加一个按钮的时候，我是这样想的：创建一个精灵（sprite）来代表按钮，然后检测这个按钮什么时候被按下去。当然，这样做肯定是可行的。但是，在cocos2d里面，还有更简单的方法–通过使用cocos2d的菜单系统。

在cocos2d的菜单系统里面，包含一个menu，在menu里面又包含一系列的menuitems。Menu　items可以是文本或者图片，而且菜单系统里面还包含了一些非常有用的逻辑，比如：排列菜单项（menu　item），高亮显示被按下去的菜单项，开关菜单项等等。好了，让我们实践一下，看看用cocos2d的方式如何创建一个简单的按钮！

**创建一个简单的按钮**

打开Xcode，使用cocos2d　Application　template创建一个新的工程并命名为：CCButtons。接下来，你需要一些按钮的图片–你可以自己创建，或者下载一些[我已经做好的图片](http://www.raywenderlich.com/downloads/Buttons.zip)。好了，现在你有图片了，把它们拖到resource文件夹下面，同时确保选中“ Copy items into destination group’s folder (if needed)”。

打开Classes分组下面的HelloWorldscene.h文件，然后在HelloWorld类里面添加一个成员变量，后面将会用到它：

CCLabelTTF \*\_label;

然后，为了防止忘记内存清理操作，打开HelloWorldScene.m并在dealloc方法中添加一些清理的代码：

[\_label release];

\_label = nil;

好了，接下来就是重点了。同样，在HelloWorldScene.m文件中，用下面的代码替换掉init方法：

-(id) init

{

if( (self=[super init] )) {CGSize winSize = [[CCDirector sharedDirector] winSize];

// Create a label for display purposes \_label = [[CCLabelTTF labelWithString:@"Last button: None" dimensions:CGSizeMake(320, 50) alignment:UITextAlignmentCenter fontName:@"Arial" fontSize:32.0] retain]; \_label.position = ccp(winSize.width/2, winSize.height-(\_label.contentSize.height/2)); [self addChild:\_label];

// Standard method to create a button CCMenuItem \*starMenuItem = [CCMenuItemImage itemFromNormalImage:@"ButtonStar.png" selectedImage:@"ButtonStarSel.png" target:self selector:@selector(starButtonTapped:)]; starMenuItem.position = ccp(60, 60); CCMenu \*starMenu = [CCMenu menuWithItems:starMenuItem, nil]; starMenu.position = CGPointZero; [self addChild:starMenu];

} return self; }

首先，为了调试方便，我们创建了一个label。这个看起来很熟悉对不对？—我们在上一篇教程中有提到过。然而，这一次我们使用了一个新的构造函数，它可以让我们指定label的大小和文字对齐方式。在这里，我把label的大小设置和窗口大小一样宽，而且文本需要居中对齐。这是一种广为人知的技术了，特别是当你想实现一些左对齐或者右对齐的文本的时候。

接下来的代码是创建按钮。首先使用类CCMenuItemImage来创建一个菜单项，并为这个按钮指定一张被选中的图片和没有被选中的图片。（也就是单击时被显示的图片和没有被单击时被显示的图片）当创建完菜单项之后，我们为按钮的点击事件指定了一个回调函数（这个函数后面会给出代码）。最后一步，就是创建一个菜单来包含这个按钮（或者一系列的按钮，以nil结尾）。

注意，我们在CGPointZero（原点）的位置创建了按钮。这里实际上指定了菜单的中心点的位置。然后，我们指定菜单项的位置相对于菜单的位置偏移（６０，６０）–这样的话，在屏幕上面显示的时候，菜单项就会显示在（６０，６０）的位置了。（因为菜单项的position是相对于菜单的中心点来的，把菜单的中心点设置为（０，０），与屏幕坐标原点重合后，可以方便为每个菜单项指定坐标点，因为这时候，只要按钮实际屏幕出现的位置设置菜单项的坐标点就行了）。

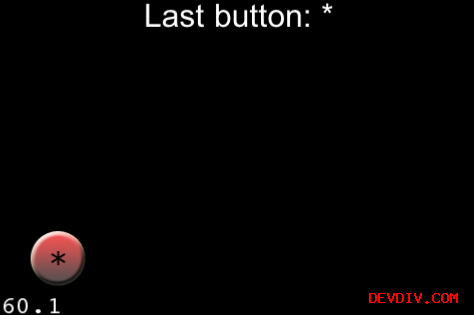
好了，还有一些代码需要补充。在init方法后面，添加我们的按钮回调函数：

- (void)starButtonTapped:(id)sender {

[\_label setString:@"Last button: \*"];

}

编译并运行，你会看到如下的运行结果：



**开关按钮**

另外一种在iphone游戏里面常用的按钮类型就是–开关按钮。这种类型的按钮一次只有一个图片显示出来，当你单击它的时候，它就会切换到另外一张图片。这个可以用来制作一个控制面板的可见性的控制器，这样可以最大限度地利用iphone上面有限的屏幕大小。

非常幸运的是，cocos2d里面就内置了一种特殊的menu　item叫做CCMenuItemToggle，它可以使事情变得更加简单。让我们来体验一下吧！首先，在HelloWorldScene.h里面添加２个成员变量：

CCMenuItem \*\_plusItem;

CCMenuItem \*\_minusItem;

然后在dealloc方法里面添加下面的清理代码：

[\_plusItem release];

\_plusItem = nil;

[\_minusItem release];

\_minusItem = nil;

然后，在你为场景添加的StartMenu后面，再添加下面的代码：

\_plusItem = [[CCMenuItemImage itemFromNormalImage:@"ButtonPlus.png"

selectedImage:@"ButtonPlusSel.png" target:nil selector:nil] retain];

\_minusItem = [[CCMenuItemImage itemFromNormalImage:@"ButtonMinus.png"

selectedImage:@"ButtonMinusSel.png" target:nil selector:nil] retain];

CCMenuItemToggle \*toggleItem = [CCMenuItemToggle itemWithTarget:self

selector:@selector(plusMinusButtonTapped:) items:\_plusItem, \_minusItem, nil];

CCMenu \*toggleMenu = [CCMenu menuWithItems:toggleItem, nil];

toggleMenu.position = ccp(60, 120);

[self addChild:toggleMenu];

首先，  就像我们前一个例子中一样，创建两个CCMenuItemImage。这里有一点不同–我把它们都添加到了CCMenuItemToggle里面。这个类会管理当前应该显示的菜单项，同时会在开关元素之间进行一些切换。

注意，当创建CCMenuItemImage的时候，我把回调函数设置成了nil，但是我为CCMenuItemToggle类设置了回调函数。这样，就会使得代码意途更加清晰：当CCMenuItemImage在CCMenuItemToggle中的时候，在CCMenuItemImage上的任何selector都不会被调用，而只有 CCMenuItemToggle的selector会被调用。当然，我们可以很容易地在回调函数里面区分，到底哪个菜单项是可见的。

接下来，让我们看看如何实现回调函数吧！在init方法后面添加下面的代码：

- (void)plusMinusButtonTapped:(id)sender {

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

if (toggleItem.selectedItem == \_plusItem) {

[\_label setString:@"Visible button: +"];

} else if (toggleItem.selectedItem == \_minusItem) {

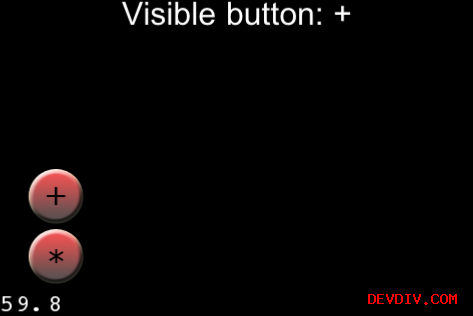
[\_label setString:@"Visible button: -"];

}

}

因此，正如你所见，CCMenuItemToggle里有一个selectedItem属性，它可以告诉我们它的哪一个子菜单项当前可见（注意，当前可见的不等于被单击的）

好了，让我们运行一下吧！你会看到如下结果：



**单选按钮**

第三种常用的按钮类型就是单选按钮（radio　button）.我在做一个游戏的时候，发现自己需要一些单选按钮，但是，cocos2d的源代码里面并没有任何有关单选按钮的实现。因此，我们自己实现一个单选按钮。然后，当我在写这篇教程的时候，我发现另外两个人也写了一些[《在cocos2d里面如何支持单选按钮》](http://code.google.com/p/cocos2d-iphone/issues/detail?id=381)的文章–这意味着，在不久的将来，你将会在cocos2d的源文件里面看到有关单选按钮的实现。

但是，目前cocos2d里面还是没有，因此，在这期间，你可以免费地使用我上面提到的一些实现。这篇教程使用的是我自己写的单选按钮的实现。首先，下载[CCRadioMenu.h和CCRadioMenu.m](http://www.raywenderlich.com/downloads/CCRadioMenu.zip)，然后把它们拖到你的Classes分组下面（确保复选“”）。然后在HelloWorldScene.m的顶部添加下面代码：

#import ”CCRadioMenu.h”

然后，在init方法后面，紧跟你添加开关按钮的代码，添加下面代码：

CCMenuItem \*menuItem1 = [CCMenuItemImage itemFromNormalImage:@"Button1.png"

selectedImage:@"Button1Sel.png" target:self selector:@selector(button1Tapped:)];

CCMenuItem \*menuItem2 = [CCMenuItemImage itemFromNormalImage:@"Button2.png"

selectedImage:@"Button2Sel.png" target:self selector:@selector(button2Tapped:)];

CCMenuItem \*menuItem3 = [CCMenuItemImage itemFromNormalImage:@"Button3.png"

selectedImage:@"Button3Sel.png" target:self selector:@selector(button3Tapped:)];

CCRadioMenu \*radioMenu =

[CCRadioMenu menuWithItems:menuItem1, menuItem2, menuItem3, nil];

radioMenu.position = ccp(120, 180);

[radioMenu alignItemsHorizontally];

radioMenu.selectedItem = menuItem1;

[menuItem1 selected];

[self addChild:radioMenu];

首先，像之前一样，创建CCMenuItemImage，但是我们不是把它加到CCMenu类中，而是把它们加到CCRadioMenu类中。这个类确保一次只有一个菜单项被选中。这里，我们设置默认情况下，第一个菜单项被选中。

这里有一个新的知识点：我们利用cocos2d里面的自己布局功能，调用menu的alignItemsHorizontally来水平对齐menu中的所有菜单项。注意，菜单项是相对于菜单的中心点来布局的。因此，我们不再需要把菜单的中心点设置为（０，０）了–取而代之的是，我们需要把菜单往中间靠右挪动一些，这样我们就可以让菜单项都完整地显示出来。

最后一件事情–像之前一样添加回调函数：

- (void)button1Tapped:(id)sender {

[\_label setString:@"Last button: 1"];

}- (void)button2Tapped:(id)sender {

[\_label setString:@"Last button: 2"];

}

- (void)button3Tapped:(id)sender { [\_label setString:@"Last button: 3"]; }

编译并运行，你将会看到下面的结果：



**背后的原理**

如果你看一看菜单系统是如何实现的，你会注意到所有的菜单项都是CCNode的子类，但是Menu是CCLayer的子类。根据[cocos2d最佳实践](http://www.cocos2d-iphone.org/wiki/doku.php/ko:prog_guide:best_practices?s%5B%5D=layers)，你不应该创建非常大的层次结构，你应该让层次结构尽可能的小。

因此，这意味着，你可能需要把尽可能多的菜单项放到一个菜单里面。因为，CCLayer是从CCNode派生出来的，它也可以run　action。就是说，你可以对menu　run　action。

**总结**

这里有这个教程的完整的[全部源代码](http://www.raywenderlich.com/downloads/Cocos2DButtons.zip)。希望这个教程对你有帮助，如果你在使用cocos2d的button的时候有什么好的意见或好点子，请跟我分享！

**在cocos2d里面如何使用物理引擎box2d：弹球 by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/457/intro-to-box2d-with-cocos2d-tutorial-bouncing-balls>

程序截图：



这个教程的目的就是让你们熟悉在cocos2d里面如何使用box2d，所采用的例子就是制作一个简单的应用，里面有一个篮球，你可以通过旋转你的iPhone来改变重力的方向，同时篮球碰到屏幕边界可以反弹。

这个教程假设你已经学过前面的教程[《如何使用cocos2d来制作一个简单的iphone游戏》](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E6%9D%A5%E5%81%9A%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84iphone%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B1.html)，或者有同等相关经验也可以。

好了，让我们开始学习Box2d物理引擎吧！

**创建一个空的工程**

打开Xcode，选择 cocos2d-0.99.1 Box2d Application template来创建一个新的工程，并且命名为Box2D.如果你直接编译并且运行的话，你将会看到一个很酷的例子，里面展示了Box2d的许多内容。然后，这个教程的目的，我们将从0开始，创建一个篮球反弹的应用，这样我们就可以更好地理解那个范例的具体原理。

因此，让我们把HelloWorld模板里面的内容都删除掉，因为我们要从0开始。把HelloWorldScene.h里面的内容替换成下面的代码：

#import ”cocos2d.h”@interface HelloWorld : CCLayer {

}+ (id) scene;@end

同时修改HelloWorldScene.mm文件：（为什么后缀是.mm，因为box2d是c++写的，而objective-c++的实现文件必须是.mm后缀，否则你编译会出n个错误！）

#import ”HelloWorldScene.h”@implementation HelloWorld+ (id)scene {CCScene \*scene = [CCScene node];

HelloWorld \*layer = [HelloWorld node];

[scene addChild:layer];

return scene;

}

- (id)init {

if ((self=[super init])) { } return self; }

@end

最后一步—-验证一下，你的Classes分组下面的所有文件（比如HelloWorldScene）是以.mm文件结尾的，如果是.m，那么请改成.mm，否则等下使用Box2d的时候，编译器会报出一大堆莫名其妙的错误！

如果你编译并运行，你应该看到一个黑色的屏幕。好了，现在让我们开始创建Box2d场景吧。

**Box2D世界相关理论**

在我们开始之前，让我们先交待一下Box2D具体是如何运作的。

你需要做的第一件事情就是，当使用cocos2d来为box2d创建一个world对象的时候。这个world对象管理物理仿真中的所有对象。

一旦我们已经创建了这个world对象，接下来需要往里面加入一些body对象。body对象可以随意移动，可以是怪物或者飞镖什么的，只要是参与碰撞的游戏对象都要为之创建一个相应的body对象。当然，也可以创建一些静态的body对象，用来表示游戏中的台阶或者墙壁等不可以移动的物体。

为了创建一个body对象，你需要做很多事情–首先，创建一个body定义结构，然后是body对象，再指定一个shap，再是fixture定义，然后再创建一个fixture对象。下面会一个一个解释刚刚这些东西。

* 你首先创建一个body定义结构体，用以指定body的初始属性，比如位置或者速度。
* 一旦创建好body结构体后，你就可以调用world对象来创建一个body对象了。
* 然后，你为body对象定义一个shape，用以指定你想要仿真的物体的几何形状。
* 接着创建一个fixture定义，同时设置之前创建好的shape为fixture的一个属性，并且设置其它的属性，比如质量或者摩擦力。
* 最后，你可以使用body对象来创建fixture对象，通过传入一个fixture的定义结构就可以了。
* 请注意，你可以往单个body对象里面添加很多个fixture对象。这个功能在你创建特别复杂的对象的时候非常有用。比如自行车，你可能要创建2个轮子，车身等等，这些fixture可以用关节连接起来。

只要你把所有需要创建的body对象都创建好之后，box2d接下来就会接管工作，并且高效地进行物理仿真—只要你周期性地调用world对象的step函数就可以了。

但是，请注意，box2d仅仅是更新它内部模型对象的位置–如果你想让cocos2d里面的sprite的位置也更新，并且和物理仿真中的位置相同的话，那么你也需要周期性地更新精灵的位置。

好了，现在有一些基本的了解了，还是先看看代码吧！

**Box2d World 实战**

好了，下载我制作的[篮球图片](http://d1xzuxjlafny7l.cloudfront.net/downloads/Ball.jpg)，并且把它添加到工程里去吧。下载完后，直接拖到Resources文件夹下，同时确保 “Copy items into destination group’s folder (if needed)” 被复选中。

接下来，在HelloWorldScene.mm文件顶部添加下面的代码：

#define PTM\_RATIO 32.0

这里定义了一个“像素/米”的比率。当你在cocos2d里面指定一个body在哪个位置时，你使用的单位要是米。但是，我们之前使用的都是像素作为单位，那样的话，位置就会不正确。根据[Box2d参考手册](http://www.box2d.org/manual.html)，Box2d在处理大小在0.1到10个单元的对象的时候做了一些优化。这里的0.1米大概就是一个杯子那么大，10的话，大概就是一个箱子的大小。

因此，我们并不直接传递像素，因为一个很小的对象很有60×60个像素，那已经大大超过了box2d优化时所限定的大小。因此，如果我们有一个64像素的对象，我们可以把它除以PTM\_RATIO，得到2米—这个长度，box2d刚好可以很好地用来做物理仿真。

好了，现在来点有意思的东西。在HelloWorldScene.h文件顶部添加下列代码：

#import ”Box2D.h”

同时在HelloWorld类中添加以下成员变量：

b2World \*\_world;

b2Body \*\_body;

CCSprite \*\_ball;

然后，在HelloWorldScene.mm的init方法中加入下面的代码：

CGSize winSize = [CCDirector sharedDirector].winSize;// Create sprite and add it to the layer

\_ball = [CCSprite spriteWithFile:@"Ball.jpg" rect:CGRectMake(0, 0, 52, 52)];

\_ball.position = ccp(100, 100);

[self addChild:\_ball];// Create a world

b2Vec2 gravity = b2Vec2(0.0f, -30.0f);

bool doSleep = true;

\_world = new b2World(gravity, doSleep);// Create edges around the entire screen

b2BodyDef groundBodyDef;

groundBodyDef.position.Set(0,0);

b2Body \*groundBody = \_world->CreateBody(&groundBodyDef);

b2PolygonShape groundBox;

b2FixtureDef boxShapeDef;

boxShapeDef.shape = &groundBox;

groundBox.SetAsEdge(b2Vec2(0,0), b2Vec2(winSize.width/PTM\_RATIO, 0));

groundBody->CreateFixture(&boxShapeDef);

groundBox.SetAsEdge(b2Vec2(0,0), b2Vec2(0, winSize.height/PTM\_RATIO));

groundBody->CreateFixture(&boxShapeDef);

groundBox.SetAsEdge(b2Vec2(0, winSize.height/PTM\_RATIO),

b2Vec2(winSize.width/PTM\_RATIO, winSize.height/PTM\_RATIO));

groundBody->CreateFixture(&boxShapeDef);

groundBox.SetAsEdge(b2Vec2(winSize.width/PTM\_RATIO,

winSize.height/PTM\_RATIO), b2Vec2(winSize.width/PTM\_RATIO, 0));

groundBody->CreateFixture(&boxShapeDef);

// Create ball body and shape b2BodyDef ballBodyDef; ballBodyDef.type = b2\_dynamicBody; ballBodyDef.position.Set(100/PTM\_RATIO, 100/PTM\_RATIO); ballBodyDef.userData = \_ball; \_body = \_world->CreateBody(&ballBodyDef);

b2CircleShape circle; circle.m\_radius = 26.0/PTM\_RATIO;

b2FixtureDef ballShapeDef; ballShapeDef.shape = &circle; ballShapeDef.density = 1.0f; ballShapeDef.friction = 0.2f; ballShapeDef.restitution = 0.8f; \_body->CreateFixture(&ballShapeDef);

[self schedule:@selector(tick:)];

呃，这里有很多陌生的代码。我们一点点来解释一下。下面，我会一段段地重复上面的代码，那样可以解释地更加清楚一些。

CGSize winSize = [CCDirector sharedDirector].winSize;// Create sprite and add it to the layer

\_ball = [CCSprite spriteWithFile:@"Ball.jpg" rect:CGRectMake(0, 0, 52, 52)];

\_ball.position = ccp(100, 100);

[self addChild:\_ball];

首先，我们往屏幕中间加入一个精灵。如果你看了前面的教程的话，这里应该没有什么问题。

// Create a world

b2Vec2 gravity = b2Vec2(0.0f, -30.0f);

bool doSleep = true;

\_world = new b2World(gravity, doSleep);

接下来，我们创建了world对象。当我们创建这个对象的时候，需要指定一个初始的重力向量。这里，我们设置y轴方向为-30，因此，所有的body都会往屏幕下面下落。同时，我们还指定了一个值，用以指明对象不参与碰撞时，是否可以“休眠”。一个休眠的对象将不会花费处理时间，直到它与其实对象发生碰撞的时候才会“醒”过来。

b2BodyDef groundBodyDef;

groundBodyDef.position.Set(0,0);

b2Body \*groundBody = \_world->CreateBody(&groundBodyDef);

b2PolygonShape groundBox;

b2FixtureDef boxShapeDef;

boxShapeDef.shape = &groundBox;

groundBox.SetAsEdge(b2Vec2(0,0), b2Vec2(winSize.width/PTM\_RATIO, 0));

groundBody->CreateFixture(&boxShapeDef);

groundBox.SetAsEdge(b2Vec2(0,0), b2Vec2(0, winSize.height/PTM\_RATIO));

groundBody->CreateFixture(&boxShapeDef);

groundBox.SetAsEdge(b2Vec2(0, winSize.height/PTM\_RATIO),

b2Vec2(winSize.width/PTM\_RATIO, winSize.height/PTM\_RATIO));

groundBody->CreateFixture(&boxShapeDef);

groundBox.SetAsEdge(b2Vec2(winSize.width/PTM\_RATIO,

winSize.height/PTM\_RATIO), b2Vec2(winSize.width/PTM\_RATIO, 0));

groundBody->CreateFixture(&boxShapeDef);

接下来，我们为整个屏幕创建了一圈不可见的边。具体的步骤如下：

* 首先创建一个body定义结构体，并且指定它应该放在左下角。
* 然后，使用world对象来创建body对象。（注意，这里一定要使用world对象来创建，不能直接new，因为world对象会做一些内存管理操作。）
* 接着，为屏幕的每一个边界创建一个多边形shape。这些“shape”仅仅是一些线段。注意，我们把像素转换成了“meter”。通过除以之前定义的比率来实现的。
* 再创建一个fixture定义，指定shape为polygon shape。
* 再使用body对象来为每一个shape创建一个fixture对象。
* 注意：一个body对象可以包含许许多多的fixture对象。

// Create ball body and shape

b2BodyDef ballBodyDef;

ballBodyDef.type = b2\_dynamicBody;

ballBodyDef.position.Set(100/PTM\_RATIO, 100/PTM\_RATIO);

ballBodyDef.userData = \_ball;

\_body = \_world->CreateBody(&ballBodyDef);b2CircleShape circle;

circle.m\_radius = 26.0/PTM\_RATIO;b2FixtureDef ballShapeDef;

ballShapeDef.shape = &circle;

ballShapeDef.density = 1.0f;

ballShapeDef.friction = 0.2f;

ballShapeDef.restitution = 0.8f;

\_body->CreateFixture(&ballShapeDef);

接下来，我们创建篮球的body。这个步骤和之前创建地面的body差不多，但是有下面一些差别需要注意一下：

* 我们指定body的类型为dynamic body。默认值是static body，那意味着那个body不能被移动也不会参与仿真。很明显，我们想让篮球参与仿真。
* 设置body的user data属性为篮球精灵。你可以设置任何东西，但是，你设置成精灵会很方便，特别是当两个body碰撞的时候，你可以通过这个参数把精灵对象取出来，然后做一些逻辑处理。
* 这里使用了一个不同的shape类型–circle shape。
* 在这里，我们需要为这个fixture指定一些参数，因此，我们没有使用便捷方法来创建fixture。后面我们会讲到这些参数的具体意义。

[self schedule:@selector(tick:)];

最后一件事情就是调度一个tikck方法，这个方法默认是0.1秒回调一次。注意，这并不是最好的处理方式—最好的方式应该是让tick方法有固定的频率（比如每秒60次）。然后，这个教程我们就先这样了。

因此，让我们来实现tick方法。在init方法之后加入下面的代码：

- (void)tick:(ccTime) dt {\_world->Step(dt, 10, 10);

for(b2Body \*b = \_world->GetBodyList(); b; b=b->GetNext()) {

if (b->GetUserData() != NULL) {

CCSprite \*ballData = (CCSprite \*)b->GetUserData();

ballData.position = ccp(b->GetPosition().x \* PTM\_RATIO,

b->GetPosition().y \* PTM\_RATIO);

ballData.rotation = -1 \* CC\_RADIANS\_TO\_DEGREES(b->GetAngle());

}

}}

第一件事情就是调用world对象的step方法，这样它就可以进行物理仿真了。这里的两个参数分别是“速度迭代次数”和“位置迭代次数”–你应该设置他们的范围在8-10之间。（译者：这里的数字越小，精度越小，但是效率更高。数字越大，仿真越精确，但同时耗时更多。8一般是个折中，如果学过数值分析，应该知道迭代步数的具体作用）。

接下来，我们要使我们的精灵匹配物理仿真。因此，我们遍历world对象里面的所有body，然后看body的user data属性是否为空，如果不为空，就可以强制转换成精灵对象。接下来，就可以根据body的位置来更新精灵的位置了。

最后一件事—清理内存！因此，在文件的末尾加入下面的代码：

- (void)dealloc {

delete \_world;

\_body = NULL;

\_world = NULL;

[super dealloc];

}

编译并运行，你应该可以看到球会往下掉，并且会从屏幕底部往上面弹起来。



**关于仿真的一些注意事项**

前面我们说后面会讨论density，friction和restitution参数的意义。

* **Density** 就是单位体积的质量（密度）。因此，一个对象的密度越大，那么它就有更多的质量，当然就会越难以移动.
* **Friction 就是摩擦力。它的范围是0-1.0， 0意味着没有摩擦，1代表最大摩擦，几乎移不动的摩擦。**
* **Restitution** 回复力。它的范围也是0到1.0. 0意味着对象碰撞之后不会反弹，1意味着是完全弹性碰撞，会以同样的速度反弹。

建议多去改一改这些参数，看看具体会给小球带来什么影响。一定要去试哦！

**完成加速计控制**

如果我们可以通过倾斜屏幕让球朝着屏幕的某个方向运行，那将会很棒。首先，我们需要在init方法里面加入下面的代码：

self.isAccelerometerEnabled = YES;

然后，在文件的某个位置加入下面的方法：

- (void)accelerometer:(UIAccelerometer \*)accelerometer didAccelerate:(UIAcceleration \*)acceleration {// Landscape left values

b2Vec2 gravity(-acceleration.y \* 15, acceleration.x \*15);

\_world->SetGravity(gravity);}

这里就是设置加速计的向量乘以某个数，然后再设置为world对象的重力向量。编译并运行（最好编译到设备上面，只有设备上面才有加速计），看看效果吧！

**何去何从？**

这里是[完整源代码](http://d1xzuxjlafny7l.cloudfront.net/downloads/Box2D.zip)。

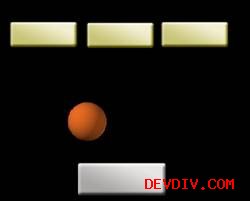
如果你想学习更多有关box2d相关的内容，请看下一篇教程[《在cocos2d里面如何使用box2d制作一个Breakout游戏：第一部分》](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E5%92%8Cbox2d%E6%9D%A5%E5%88%B6%E4%BD%9C%E4%B8%80%E4%B8%AAbreakout%E6%B8%B8%E6%88%8F%EF%BC%9A%E7%AC%AC%E4%B8%80%E9%83%A8%E5%88%86.html)。

**如何使用cocos2d和box2d来制作一个Breakout游戏：第一部分 by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/475/how-to-create-a-simple-breakout-game-with-box2d-and-cocos2d-tutorial-part-12>

程序截图：



box2d是一个非常强大的物理引擎库，同时它与cocos2d结合非常适合在iphone上面做游戏开发。著名的angry birds，tiny wings都是用box2d写的。你可以用它做好多事情，当然，最好的学习方法就是使用它来创建一个简单的游戏。

在这个教程中，我们将一步一步创建一个简单的breakout游戏，完成碰撞检测，篮球反弹物理效果，通过touch拖动paddle（就是上图的白色矩形），以及胜利/失败的场景。

如果你还不了解cocos2d和box2d，你可能先要读一读[《如何使用cocos2d制作一个简单的iphone游戏](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E6%9D%A5%E5%81%9A%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84iphone%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B1.html)》以及[《在cocos2d里面如何使用box2d物理引擎：弹球》](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%9C%A8cocos2d%E9%87%8C%E9%9D%A2%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8%E7%89%A9%E7%90%86%E5%BC%95%E6%93%8Ebox2d%EF%BC%9A%E5%BC%B9%E7%90%83.html)这些教程。

好了，是时候制作breakout了！

**一个永远反弹的球**

首先，打开Xcode，选择 cocos2d-0.99.1 Box2d Application template创建一个工程，命名为“ Box2DBreakout”. 删除掉模板代码，因此你会有一个空的工程来重新开始—具体步骤可以参照 [《在cocos2d里面如何使用box2d物理引擎：弹球》](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%9C%A8cocos2d%E9%87%8C%E9%9D%A2%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8%E7%89%A9%E7%90%86%E5%BC%95%E6%93%8Ebox2d%EF%BC%9A%E5%BC%B9%E7%90%83.html)这个教程。

一旦你有了一个很好的干净的工程后，接下来，在HelloWorldScene.h中导入下面的头文件：

#import ”Box2D.h”

同时在HelloWorld类中添加以下成员变量：

b2World \*\_world;

b2Body \*\_groundBody;

b2Fixture \*\_bottomFixture;

b2Fixture \*\_ballFixture;

然后在HelloWorldScene.mm文件顶部定义比率：

#define PTM\_RATIO 32

这个比率我们在上一个教程中已经讨论过了，这里就不再啰嗦了。

然后，在init方法中加入下列代码：

CGSize winSize = [CCDirector sharedDirector].winSize;// Create a world

b2Vec2 gravity = b2Vec2(0.0f, 0.0f);

bool doSleep = true;

\_world = new b2World(gravity, doSleep);

// Create edges around the entire screen b2BodyDef groundBodyDef; groundBodyDef.position.Set(0,0); \_groundBody = \_world->CreateBody(&groundBodyDef); b2PolygonShape groundBox; b2FixtureDef groundBoxDef; groundBoxDef.shape = &groundBox; groundBox.SetAsEdge(b2Vec2(0,0), b2Vec2(winSize.width/PTM\_RATIO, 0)); \_bottomFixture = \_groundBody->CreateFixture(&groundBoxDef); groundBox.SetAsEdge(b2Vec2(0,0), b2Vec2(0, winSize.height/PTM\_RATIO)); \_groundBody->CreateFixture(&groundBoxDef); groundBox.SetAsEdge(b2Vec2(0, winSize.height/PTM\_RATIO), b2Vec2(winSize.width/PTM\_RATIO, winSize.height/PTM\_RATIO)); \_groundBody->CreateFixture(&groundBoxDef); groundBox.SetAsEdge(b2Vec2(winSize.width/PTM\_RATIO, winSize.height/PTM\_RATIO), b2Vec2(winSize.width/PTM\_RATIO, 0)); \_groundBody->CreateFixture(&groundBoxDef);

好，这个代码和我们上一个教程中，为整个屏幕创建一个盒子边界差不多。然后，这一次，我们把重力设置为0，因为，在我们的breakout游戏中，我们并不需要重力！注意，我们存储了底部的fixture的一个指针，以方便后面使用（在后面的教程中，我们将用来追踪什么时候篮球与顶部相碰撞了）。

现在，下载我制作的[篮球图片](http://d1xzuxjlafny7l.cloudfront.net/downloads/Ball.jpg)，并且拖到Resources文件夹中，确保 “Copy items into destination group’s folder (if needed)” 被复选中。

让我们往场景里面添加一个精灵吧。紧接着上面的代码，加入下面的代码片段：

// Create sprite and add it to the layer

CCSprite \*ball = [CCSprite spriteWithFile:@"Ball.jpg"

rect:CGRectMake(0, 0, 52, 52)];

ball.position = ccp(100, 100);

ball.tag = 1;

[self addChild:ball];

这里没什么疑问，我们已经做过好多次类似的事情了。注意，我们为篮球设置了一个tag标识，后面你会看到，这个tag标记有什么用。

接下来，为shape创建一个body：

// Create ball body

b2BodyDef ballBodyDef;

ballBodyDef.type = b2\_dynamicBody;

ballBodyDef.position.Set(100/PTM\_RATIO, 100/PTM\_RATIO);

ballBodyDef.userData = ball;

b2Body \* ballBody = \_world->CreateBody(&ballBodyDef);// Create circle shape

b2CircleShape circle;

circle.m\_radius = 26.0/PTM\_RATIO;

// Create shape definition and add to body b2FixtureDef ballShapeDef; ballShapeDef.shape = &circle; ballShapeDef.density = 1.0f; ballShapeDef.friction = 0.f; ballShapeDef.restitution = 1.0f; \_ballFixture = ballBody->CreateFixture(&ballShapeDef);

这个看起来和上一篇教程中的也很像。再巩固一下吧，为了创建一个body对象，我们先要创建一个body定义结构，然后再创建body，接着是shape，再指定fixture结构，最后是创建fixture对象。

注意，我们设置这些参数有一点点不一样了：我们把回复力（restitution）设置为1.0，这意味着，我们的球在碰撞的时候，将会是完全弹性碰撞。

同时，我们也保存了球的fixture，原因和我们为什么保存屏幕底部的fixture是一样的，后面你就会看到了。

更新：注意，我们也把球的摩擦力设置为0.这样可以防止球在碰撞的时候，由于摩擦损失能量，导致来回碰撞的过程中会有一点点偏差。

**好了，是时候做一些完全不同的事了！紧接上面的代码：**

b2Vec2 force = b2Vec2(10, 10);

ballBody->ApplyLinearImpulse(force, ballBodyDef.position);

这里往球上面施加了一个冲力（impulse），这样可以让它初始化的时候朝一个特定的方向运动。

最后一件事情，就是在init方法中，增加一个tick调度方法：

[self schedule:@selector(tick:)];

下面是tick方法的实现：

- (void)tick:(ccTime) dt {

\_world->Step(dt, 10, 10);

for(b2Body \*b = \_world->GetBodyList(); b; b=b->GetNext()) {

if (b->GetUserData() != NULL) {

CCSprite \*sprite = (CCSprite \*)b->GetUserData();

sprite.position = ccp(b->GetPosition().x \* PTM\_RATIO,

b->GetPosition().y \* PTM\_RATIO);

sprite.rotation = -1 \* CC\_RADIANS\_TO\_DEGREES(b->GetAngle());

}

}}

当然，这里也和上一个教程中的一样，没有什么特别的。

最后一件事我们永远不忘记！那就是清理：

- (void)dealloc {delete \_world;

\_groundBody = NULL;

[super dealloc];

}

好了，让我们试一下吧。编译并运行工程，你将会看到一个球无限地在屏幕里面来回弹！—-很酷吧！



**增加 Paddle**

如果没有一个paddle的话，那么就不可能称其为一个breakout游戏。下载[我制作的paddle图片](http://d1xzuxjlafny7l.cloudfront.net/downloads/Paddle.jpg)，然后把它拖到Resources文件夹中，同时确保 “Copy items into destination group’s folder (if needed)” 被复选上。

然后在HelloWorldScene.h文件中往HelloWorld类中添加下列成员变量：

b2Body \*\_paddleBody;

b2Fixture \*\_paddleFixture;

然后，在init方法中构建paddle body：

// Create paddle and add it to the layer

CCSprite \*paddle = [CCSprite spriteWithFile:@"Paddle.jpg"];

paddle.position = ccp(winSize.width/2, 50);

[self addChild:paddle];// Create paddle body

b2BodyDef paddleBodyDef;

paddleBodyDef.type = b2\_dynamicBody;

paddleBodyDef.position.Set(winSize.width/2/PTM\_RATIO, 50/PTM\_RATIO);

paddleBodyDef.userData = paddle;

\_paddleBody = \_world->CreateBody(&paddleBodyDef);

// Create paddle shape b2PolygonShape paddleShape; paddleShape.SetAsBox(paddle.contentSize.width/PTM\_RATIO/2, paddle.contentSize.height/PTM\_RATIO/2);

// Create shape definition and add to body b2FixtureDef paddleShapeDef; paddleShapeDef.shape = &paddleShape; paddleShapeDef.density = 10.0f; paddleShapeDef.friction = 0.4f; paddleShapeDef.restitution = 0.1f; \_paddleFixture = \_paddleBody->CreateFixture(&paddleShapeDef);

我不想花太多的时间解释上面的内容了。因为，和之前的创建篮球的body的过程差不太多。这里只给出不同的地方：

* 当你创建CCSprite的时候，你并不需要指定精灵的大小。如果你传递一个文件名给它，它会自动计算出大小。
* 注意，这里不是使用circle shape了。这一次，我们使用polygon shape。我们使用一个辅助方法来创建shape，当然，其形状是个盒子。
* 我们使用了SetAsBox方法来指定shape相对于body的位置，这个方法在构建复杂的对象的时候比较有用。这里，我们只是让shape在body中间。
* 我把paddle的密度设置得比球要大得多，同时调节了一下其它的参数。（这些参数要靠试，按照真实的高中物理知识去计算，可能得不到）
* 同时，我们存储paddleBody和paddleFixture的引用，为了方便后面使用。

如果你编译并运行的话，你将会看到屏幕中间有一个paddle，而且球碰到它将会反弹。



然后，这还不是很有趣，因为我们还不能移动paddle！

**移动Paddle**

移动paddle需要touch事件，所以先在init方法中允许touch事件：

self.isTouchEnabled = YES;

然后，在HelloWorld类中添加下面的成员变量：

b2MouseJoint \*\_mouseJoint;

现在，让我们实现touch方法！首先是ccTouchesBegan：

- (void)ccTouchesBegan:(NSSet \*)touches withEvent:(UIEvent \*)event {if (\_mouseJoint != NULL) return;

UITouch \*myTouch = [touches anyObject]; CGPoint location = [myTouch locationInView:[myTouch view]]; location = [[CCDirector sharedDirector] convertToGL:location]; b2Vec2 locationWorld = b2Vec2(location.x/PTM\_RATIO, location.y/PTM\_RATIO);

if (\_paddleFixture->TestPoint(locationWorld)) { b2MouseJointDef md; md.bodyA = \_groundBody; md.bodyB = \_paddleBody; md.target = locationWorld; md.collideConnected = true; md.maxForce = 1000.0f \* \_paddleBody->GetMass();

\_mouseJoint = (b2MouseJoint \*)\_world->CreateJoint(&md); \_paddleBody->SetAwake(true); }

}

呃，好多新知识！让我们一点一点来讨论。

首先，我们把touch坐标转换成coocs2d坐标（convertToGL）然后，再转换成Box2d坐标（locationWorld）。

然后，我们使用paddle fixture的一个方法来测试这个touch点是否在fixture内部。

如果是的话，我们就创建一个所谓的”鼠标关节“。在Box2d里面，一个鼠标关节用来让一个body朝着一个指定的点移动—在这里个例子中，就是用户点的方向。

当你创建一个mouse joint后，你赋值给它两个body。第一个没有被使用，通常都是设置成ground body。第二个，就是你想让它移动的body，在这个例子中就是paddle。

接下来，你指定移动的终点—这个例子中就是用户点击的位置。

然后，你告诉box2d，但bodyA和bodyB碰撞的时候，把它当成是碰撞，而不是忽略它。这个很重要！因为，我之前没有设置它为ture，结果不行！因此，当我们用鼠标拖动这个paddle的时候，它并不会与屏幕的边界相碰撞，而且有时候，我的paddle直接就飞出屏幕之外了。这个非常非常奇怪，不过我现在知道是为什么了。因为没有设置bodyA和bodyB是可碰撞的。

你然后指定移动body的最大的力是多少。如果你减少这个数值的话，paddle body响应鼠标移动时就会慢一些。但是，我们想让paddle快速地响应鼠标的变化。

最后，我们把这个关节加入到world中，同时，保存这个指针，因为后面有用。同时，我们还要把body设置成苏醒的（awake）。之所以要这么做，是因为如果body在睡觉的话，那么它就不会响应鼠标的移动！

好了，接下来，让我们添加ccTouchesMoved方法：

-(void)ccTouchesMoved:(NSSet \*)touches withEvent:(UIEvent \*)event {if (\_mouseJoint == NULL) return;

UITouch \*myTouch = [touches anyObject]; CGPoint location = [myTouch locationInView:[myTouch view]]; location = [[CCDirector sharedDirector] convertToGL:location]; b2Vec2 locationWorld = b2Vec2(location.x/PTM\_RATIO, location.y/PTM\_RATIO);

\_mouseJoint->SetTarget(locationWorld);

}

这个方法的开头部分和ccTouchesBegan差不多—我们把touch坐标转换成Box2d坐标。唯一的区别就是，我们更新了鼠标关节的目标位置（也就是我们想让paddle移动的位置的）。

接下来，我们添加ccTouchesCacelled和ccTouchesEnded方法：

-(void)ccTouchesCancelled:(NSSet \*)touches withEvent:(UIEvent \*)event {if (\_mouseJoint) {

\_world->DestroyJoint(\_mouseJoint);

\_mouseJoint = NULL;

}

}

- (void)ccTouchesEnded:(NSSet \*)touches withEvent:(UIEvent \*)event { if (\_mouseJoint) { \_world->DestroyJoint(\_mouseJoint); \_mouseJoint = NULL; } }

我们在这些方法中做的只有一件事，就是在我们移动完paddle或者取消移动之后销毁mouse joint。

编译并运行，你现在可以用鼠标移动paddle了，同时可以让它与篮球相互碰撞了！



很好。。。不过，等一下，这还不是一个breakout！我们不可以把paddle移动到任何位置，我们只能在屏幕底部左右来回移动它！

**限制Paddle的移动**

我们可以很容易地限制paddle的移动，只需要添加另外一个关节，叫做prismatic joint。这个关节会限制一个body的移动沿着一根指定的轴。

因此，我们可以使用这种方法来限制paddle相对于地面移动，也就是说只能沿着x轴移动。

让我们看看相关代码。往init方法中加入下列代码：

// Restrict paddle along the x axis

b2PrismaticJointDef jointDef;

b2Vec2 worldAxis(1.0f, 0.0f);

jointDef.collideConnected = true;

jointDef.Initialize(\_paddleBody, \_groundBody,

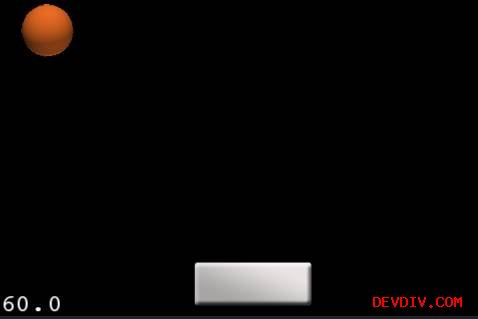
\_paddleBody->GetWorldCenter(), worldAxis);

\_world->CreateJoint(&jointDef);

第一件事情就是指定一个沿着x轴的向量。然后，我们需要指定collideConnected为true，因此，我们的球才能够正确的反弹，而不是飞到屏幕之外去。

然后，初始化关节，指定paddle和ground两个body，再使用world对象来创建关节！

编译并运行，你现在只能沿关x轴方向移动paddle了，这正是我们想要的，不是吧？



**完成touch事件**

现在，你玩一下，可能你会发现，有时候球反弹地特别快，有时候又比较慢。这取决于你是如何控制paddle与球相碰撞的。

更新：我第一次尝试去修正这个bug的时候，我通过直接调整球的速度，使用SetLinearVelocity方法。然后，Steve Oldmeadow也指出，这非常不好！它会破坏物理仿真，最好的方法是通过调用SetLinearDamping方法，间接影响速度。因此，现在这个教程就是这个做的。（damping就是阻尼的意思）

接下来，在tick方法中添加下列代码，具体位置是在获得user data之后：

if (sprite.tag == 1) {

static int maxSpeed = 10;b2Vec2 velocity = b->GetLinearVelocity();

float32 speed = velocity.Length();

if (speed > maxSpeed) { b->SetLinearDamping(0.5); } else if (speed < maxSpeed) { b->SetLinearDamping(0.0); }

}

这里，我们判断sprite的tag，看是否是球的tag。如果是的话，我们就检查它的速度，如果太大的话，就设置它的阻尼为0.5，这样可以让它慢下来。

如果你编译并运行的话，你将会看到一个球以非常适中的速度在屏幕四周来回反弹。

**给我源代码！**

这里是本教程的[完整源代码](http://d1xzuxjlafny7l.cloudfront.net/downloads/Box2DBreakout.zip)。这只是一部分，第二部分的教程会包含一个完整的breakout的源码。

**接下来呢？**

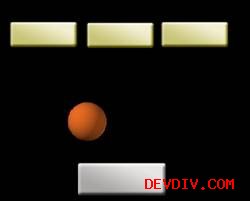
目前为止，我们已经有一个篮球在屏幕四周来回反弹了，同时还有一个paddle可以用鼠标来控制其移动。在下个教程中，我们将创建一些方块，当球碰到它们的时候，方块就会消失。当然，还有游戏胜利和失败的逻辑！

**如何使用cocos2d和box2d来制作一个Breakout游戏：第二部分（完） by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/505/how-to-create-a-simple-breakout-game-with-box2d-and-cocos2d-tutorial-part-22>

程序截图：



这是《如何使用cocos2d和box2d制作一个简单的breakout游戏》的第二部分，也是最后一部分教程。如果你还没有读过第一部分，请先阅读[第一个教程](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E5%92%8Cbox2d%E6%9D%A5%E5%88%B6%E4%BD%9C%E4%B8%80%E4%B8%AAbreakout%E6%B8%B8%E6%88%8F%EF%BC%9A%E7%AC%AC%E4%B8%80%E9%83%A8%E5%88%86.html)！

在上一个教程中，我们创建了一个屏幕盒子，球可以在里面弹跳，同时，我们可以用手指拖着paddle移动。这部分教程中，我们将添加一些游戏逻辑，当篮球碰到屏幕底部的时候，就Gameover。

**Box2D 和碰撞检测**

在Box2D里面，当一个fixture和另一个fixture相互碰撞的时候，我们怎么知道呢？这就需要用到碰撞侦听器了（contact listener)。一个碰撞侦听器是一个C++对象，它继承至box2d的b2ContactListner类，并且要设置给world对象。这样，当有两个对象发生相互碰撞的时候，world对象就会回调contact listener对象的方法，这样我们就可以在那些方法里面做相应的碰撞处理了。

如何使用contact listener呢？根据[BOX2D用户手册](http://www.box2d.org/manual.html)，在一个仿真周期内，你不能执行任何修改游戏物理的操作。因为，在那期间，我们可能需要做一些额外的处理（比如，当两个对象碰撞的时候销毁另一个对象）。因此， 我们需要保存碰撞的引用，这样后面就可以使用它。

另外一点值得注意的是，我们不能存储传递给contact listener的碰撞点的引用，因为，这些点被BOX2D所重用。因此，我们不得不存储这些点的拷贝。

好了，说得够多了，让我们亲手实践一下吧！

**当我们碰到屏幕底部的时候**

注意，在这部分里面，我们将使用一些C++代码和STL（标准模板库）。如果你还不熟悉C++或者是STL，不用太担心–你只需要复制粘贴代码就OK了，不过建议还是学习一下C++和STL，因为大部分游戏都是用C++做成的。

好。点开你的Classes文件夹，点击File\New File），选择左边的“Cocoa Touch Class”，再选择“Objective-C class”，确保“Subclass of NSObject”被选中，再点Next。把它命名为MyContactListener，然后点Finish。

右键点MyContactListener.m，并把它改成MyContactListener.mm。这是因为，我们现在要创建一个C++类，而我们使用C++的时候就需要把文件后缀改成.mm。

接下用下面的代码替换掉 MyContactListener.h里面的内容：

#import ”Box2D.h”

#import <vector>

#import <algorithm>

struct MyContact { b2Fixture \*fixtureA; b2Fixture \*fixtureB; bool operator==(const MyContact& other) const { return (fixtureA == other.fixtureA) && (fixtureB == other.fixtureB); } };

class MyContactListener : public b2ContactListener {

public: std::vector<MyContact>\_contacts;

MyContactListener(); ~MyContactListener();

virtual void BeginContact(b2Contact\* contact); virtual void EndContact(b2Contact\* contact); virtual void PreSolve(b2Contact\* contact, const b2Manifold\* oldManifold); virtual void PostSolve(b2Contact\* contact, const b2ContactImpulse\* impulse);

};

这里，我们定义了一个数据结构，当碰撞通知到达的时候，用来保存碰撞点信息。再说一遍，我们需要存储其拷贝，因为它们会被重用，所以不能保存指针。注意，我们这里一定要重载=号，因为，我们将使用find算法来查找vector中的一个特定的元素，而这个标准算法find必须要求其查找的元素是可比较相等的。

在我们申明完contact listener类之后，我们只需要声明一些我们感兴趣的方法来实现就可以了。这里的vector用来缓存碰撞点信息。

现在，用下面的内容替换掉MyContactListener.mm：

#import ”MyContactListener.h”

MyContactListener::MyContactListener() : \_contacts() { }

MyContactListener::~MyContactListener() { }

void MyContactListener::BeginContact(b2Contact\* contact) { // We need to copy out the data because the b2Contact passed in // is reused. MyContact myContact = { contact->GetFixtureA(), contact->GetFixtureB() }; \_contacts.push\_back(myContact); }

void MyContactListener::EndContact(b2Contact\* contact) { MyContact myContact = { contact->GetFixtureA(), contact->GetFixtureB() }; std::vector<MyContact>::iterator pos; pos = std::find(\_contacts.begin(), \_contacts.end(), myContact); if (pos != \_contacts.end()) { \_contacts.erase(pos); } }

void MyContactListener::PreSolve(b2Contact\* contact, const b2Manifold\* oldManifold) { }

void MyContactListener::PostSolve(b2Contact\* contact, const b2ContactImpulse\* impulse) { }

我们在构造函数中初使化vector。然后，我们只需要实现BeginContact和EndContact方法就可以了。这两个方法，一个是碰撞开始的时候world对象会回调，另一个就是碰撞结束的时候被回调。在BeginContact方法中，我们复制了刚刚发生碰撞的fixture的一份拷贝，并把它存储在vector中。在EndContact方法中，我们检查一下碰撞点是否在我们的vector中，如果在的话，就移除它！

好了，现在可以使用它吧。打开HelloWorldScene.h，然后做下面的修改：

// Add to top of file

#import ”MyContactListener.h”

// Add inside @interface MyContactListener \*\_contactListener;

然后在init方法中增加下列代码：

// Create contact listener

\_contactListener = new MyContactListener();

\_world->SetContactListener(\_contactListener);

这里，我们创建了contact listener对象，然后调用world对象把它设置为world的contact listener。

接下来，再我们忘记之前先做一些清理内存的操作：

delete \_contactListener;

最后，在tick方法底部添加下列代码：

std::vector<MyContact>::iterator pos;

for(pos = \_contactListener->\_contacts.begin();

pos != \_contactListener->\_contacts.end(); ++pos) {

MyContact contact = \*pos;

if ((contact.fixtureA == \_bottomFixture && contact.fixtureB == \_ballFixture) || (contact.fixtureA == \_ballFixture && contact.fixtureB == \_bottomFixture)) { NSLog(@”Ball hit bottom!”); } }

这里遍历所有缓存的碰撞点，然后看看是否有一个碰撞点，它的两个碰撞体分别是篮球和屏幕底部。目前为止，我们只是使用NSLog来打印一个消息，因为我们只想测试这样是否可行。

因此，在debug模式下编译并运行，你会发现，不管什么时候，当球和底部有碰撞的时候，你会看到控制台输出一句话“Ball hit bottom”!

**添加Game Over场景**

增加[GameOverScene.h and GameOverScene.mm files](http://d1xzuxjlafny7l.cloudfront.net/downloads/GameOverScene.zip)两个文件，它们可以在[《如何使用cocos2d制作一个简单的iphone游戏教程》](http://www.cnblogs.com/andyque/archive/2011/03/22/1990716.html)中找到。注意，你必须把GameOverScene.m改成GameOverScene.mm，因为我们要在里面使用一些C++，如果你不改的话，那么编译就会报错。

然后，在HelloWorldScene.mm文件中加入下列代码：

#import ”GameOverScene.h”

然后，把NSLog语句替换成下列代码：

GameOverScene \*gameOverScene = [GameOverScene node];

[gameOverScene.layer.label setString:@"You Lose :["];

[[CCDirector sharedDirector] replaceScene:gameOverScene];

好了，我们已经实现得差不多了。但是，如果你游戏你永远不能赢，那有什么意思呢？

**增加一些方块**

下载我制作的[方块图片](http://d1xzuxjlafny7l.cloudfront.net/downloads/Block.jpg)，然后把它拖到Resources文件夹下面，同时确保“Copy items into destination group’s folder (if needed)”被复选中。

然后往init方法中添加下列代码：

for(int i = 0; i < 4; i++) {

static int padding=20;

// Create block and add it to the layer CCSprite \*block = [CCSprite spriteWithFile:@"Block.jpg"]; int xOffset = padding+block.contentSize.width/2+ ((block.contentSize.width+padding)\*i); block.position = ccp(xOffset, 250); block.tag = 2; [self addChild:block];

// Create block body b2BodyDef blockBodyDef; blockBodyDef.type = b2\_dynamicBody; blockBodyDef.position.Set(xOffset/PTM\_RATIO, 250/PTM\_RATIO); blockBodyDef.userData = block; b2Body \*blockBody = \_world->CreateBody(&blockBodyDef);

// Create block shape b2PolygonShape blockShape; blockShape.SetAsBox(block.contentSize.width/PTM\_RATIO/2, block.contentSize.height/PTM\_RATIO/2);

// Create shape definition and add to body b2FixtureDef blockShapeDef; blockShapeDef.shape = &blockShape; blockShapeDef.density = 10.0; blockShapeDef.friction = 0.0; blockShapeDef.restitution = 0.1f; blockBody->CreateFixture(&blockShapeDef);

}

现在，你应该可以很好地理解上面的代码了。就像之前我们为paddle创建一个body类似，这里，我们每一次也会一个方块创建一个body。注意，我们把方块精灵对象的tag设置为2，这样将来可以用到。

编译并运行，你应该可以看到篮球和方块之间有碰撞了。

**销毁方块**

为了使breakout游戏是一个真实的游戏，当篮球和方块有交集的时候，我们需要销毁这些方块。我们已经添加了一些代码来追踪碰撞，因此，我们对tick方法做一改动。

具体改动方式如下：

std::vector<b2Body \*>toDestroy;

std::vector<MyContact>::iterator pos;

for(pos = \_contactListener->\_contacts.begin();

pos != \_contactListener->\_contacts.end(); ++pos) {

MyContact contact = \*pos;

if ((contact.fixtureA == \_bottomFixture && contact.fixtureB == \_ballFixture) || (contact.fixtureA == \_ballFixture && contact.fixtureB == \_bottomFixture)) { GameOverScene \*gameOverScene = [GameOverScene node]; [gameOverScene.layer.label setString:@"You Lose :["]; [[CCDirector sharedDirector] replaceScene:gameOverScene]; }

b2Body \*bodyA = contact.fixtureA->GetBody(); b2Body \*bodyB = contact.fixtureB->GetBody(); if (bodyA->GetUserData() != NULL && bodyB->GetUserData() != NULL) { CCSprite \*spriteA = (CCSprite \*) bodyA->GetUserData(); CCSprite \*spriteB = (CCSprite \*) bodyB->GetUserData();

// Sprite A = ball, Sprite B = Block if (spriteA.tag == 1 && spriteB.tag == 2) { if (std::find(toDestroy.begin(), toDestroy.end(), bodyB) == toDestroy.end()) { toDestroy.push\_back(bodyB); } } // Sprite B = block, Sprite A = ball else if (spriteA.tag == 2 && spriteB.tag == 1) { if (std::find(toDestroy.begin(), toDestroy.end(), bodyA) == toDestroy.end()) { toDestroy.push\_back(bodyA); } } } }

std::vector<b2Body \*>::iterator pos2; for(pos2 = toDestroy.begin(); pos2 != toDestroy.end(); ++pos2) { b2Body \*body = \*pos2; if (body->GetUserData() != NULL) { CCSprite \*sprite = (CCSprite \*) body->GetUserData(); [self removeChild:sprite cleanup:YES]; } \_world->DestroyBody(body); }

好，让我们解释一下。我们又一次遍历所有的碰撞点，但是，这一次在我们测试完篮球和屏幕底部相撞的时候，我们将检查碰撞点。我们可以通过fixture对象的GetBody方法来找对象。

接着，我们基于精灵的tag，看看到底是哪个在发生碰撞。如果一个精灵与一个body相交的话，我们就需要添加一系列的对象来删除。

注意，我们把该销毁的全部加入到一个指定的销毁点去。但是，也需要注意的是，即使还没好，我也要等几分钟把它加进去。

最后，遍历我们想要删除的body列表。

编译并运行，现在你可以销毁bricks了！

**加入游戏胜利条件**

接下来，我们需要添加一些逻辑，让用户能够取得游戏胜利。修改你的tick方法的开头部分，像下面一样：

- (void)tick:(ccTime) dt {

bool blockFound = false; \_world->Step(dt, 10, 10); for(b2Body \*b = \_world->GetBodyList(); b; b=b->GetNext()) { if (b->GetUserData() != NULL) { CCSprite \*sprite = (CCSprite \*)b->GetUserData(); if (sprite.tag == 2) { blockFound = true; } //…

我们需要做的，仅仅是遍历一下场景中的所有对象，看看是否还有一个方块—-如果我们确实找到了一个，那么就把blockFound变量设置为true，否则就设置为false.

然后，在这个函数的末尾添加下面的代码：

if (!blockFound) {

GameOverScene \*gameOverScene = [GameOverScene node];

[gameOverScene.layer.label setString:@"You Win!"];

[[CCDirector sharedDirector] replaceScene:gameOverScene];

}

这里，如果方块都消失了，我们就会显示一个游戏结束的场景。编译并运行，看看，你的游戏现在有胜利终止条件了！



**完成touch事件**

这个游戏非常酷，但是，毫无疑问，我们需要音乐！你可以下载我[制作的一些背景音乐](http://d1xzuxjlafny7l.cloudfront.net/downloads/background-music-aac.caf.zip)，还有[好听的blip声音](http://d1xzuxjlafny7l.cloudfront.net/downloads/blip.caf.zip)。和之前一样，把它拖到你的resources文件夹下。

顺便提一下，我制作这些声音效果使用一个非常不错的程序，叫做[cfxr](http://thirdcog.eu/apps/cfxr).不管怎么说，你旦你加入进去之后，把HeloWorldScene.mm中加入下面的代码：

#import ”SimpleAudioEngine.h”

接着，在init方法中加入下列代码：

[[SimpleAudioEngine sharedEngine] playBackgroundMusic:@”background-music-aac.caf”];

最后，在tick方法的末尾添加下面的代码：

if (toDestroy.size() > 0) {

[[SimpleAudioEngine sharedEngine] playEffect:@”blip.caf”];

}

恩，终于完成了！你现在拥有一个使用Box2d物理引擎制作的breakout游戏了1

And there you have it – your own simple breakout game with Box2D physics!

**给我代码！**

这里是本系列教程的[完整源代码](http://d1xzuxjlafny7l.cloudfront.net/downloads/Box2DBreakout2.zip)。

**何去何从？**

很明显，这是一个非常简单的beakout游戏，但是，你还可以在此教程的基础上实现更多。我可以添加一些逻辑，比如打击一个白色块就计一分，或者有些块需要打中很多下才消失。或者你也可以添加新的不同类型的block，并且让paddle可以发射出激光等等。你可以充分发挥想象。

如果大家有什么问题可以提出来，大家一起讨论解决！如果有什么新的发现的，也欢迎贴出代码来，大家一起学习下！

PS:译者水平有限，如果有翻译的不好，或者翻译错误的，希望大家在看的时候给我指出来。谢谢！

**如何在cocos2d里面使用动画和spritesheet by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/1271/how-to-use-animations-and-sprite-sheets-in-cocos2d>

教程截图：



在这个博客中，我收到了大量的读者来信说，你能不能写一个关于如何在cocos2d里面使用动画和spritesheet的教程。这篇教程就应运而生了！

在这个教程里，我将向大家展示如何用cocos2d来制作一只熊在走路的动画。同时，我会使用spritesheet来使动画运行效率更高，还有如何让用户鼠标点击决定熊的行走方向，以及怎样基于熊当前行进的方向改变熊的面朝方向。

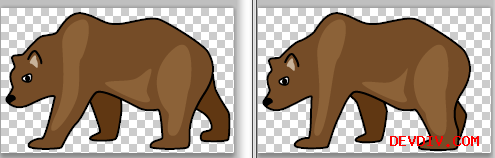
如果你对cocos2d完全陌生的话，你可能需要先阅读[《怎么使用cocos2d来制作简单的iphone游戏》](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E6%9D%A5%E5%81%9A%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84iphone%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B1.html)这一系列的教程，但是也不一定！（如果说你已经有相关经验就另当别论了）

**Getting Started**

让我们首先创建一个工程骨架–使用cocos2d工程模板创建一个新的项目并取名为AnimBear.

接下来，下载一些由我的老婆制作的[熊行走的图片](http://www.raywenderlich.com/downloads/BearImages.zip)。（老婆会美工多好啊！）

当你解压之后，看看那些图片—它们仅仅是一张张单个的熊在行走的动画帧。但是，当你把它们连续地放映，就会看到一只熊在移动。



现在，把这些图片加到工程里面，然后基于这些单个的图片来创建动画。然后，在cocos2d里面，还有另一种更加高效的方式来创建动画–那就是使用spritesheet。（也叫精灵表单）

**精灵表单和熊**

如果你从来没有使用过spritesheet，你可以把它看作是一张巨大的图片，你可以把许许多多的sprite放进去。与spritesheet对应的，还有一个plist文件，这个文件指定了每个独立的sprite在这张“大图”里面的位置和大小，当你在代码之间需要使用这个sprite的时候，就可以很方面地使用plist文件中的这些信息来获取sprite。

为什么这会提高效率呢？因为cocos2d对它进行了优化！如果你使用spritesheet来获取sprite，那么当场景中有许多sprite的时候，如果这些sprite共享一个spritesheet，那么cocos2d就会使用一次OpenGL　ES调用来渲染这些sprite。但是，如果是单个的sprite的话，那么就会有N次OpenGL　ES　call，这个代价是相当昂贵的。

简而言之–使用spritesheet会更快，尤其是当你有很多的sprite的时候！（使用spritesheet还可以减少游戏占用的内存大小，具体参考我翻译的文章[《在cocos2d里面如何使用TexturePacker和像素格式来优化spritesheet》](http://www.cnblogs.com/andyque/articles/1988097.html)）

由于要使用spritesheet，你当然可以手工用图片编辑器来创建，然后创建一个plist指定每一个sprite在spritesheet里面的位置和大小。然后，那样将会是一个非常傻比的行为，因为Robert　Payne已经开发出了一个非常好用的工具，叫做Zwoptex，它可以帮助我们自动生成这一切！

**Zwoptex To Victory!**

如果你还没有这个工具，那么可以从 [zwoptexapp.com](http://zwoptexapp.com/)上面下载。它有一个免费的Flash版本和一个收费的安装版，但是最近我使用的是可安装的版本。

安装完这个工具之后，选择File\New，然后你将会看到一个空白窗口。打开你先前下载的熊的图片，并把它们拖到这个窗口里面。

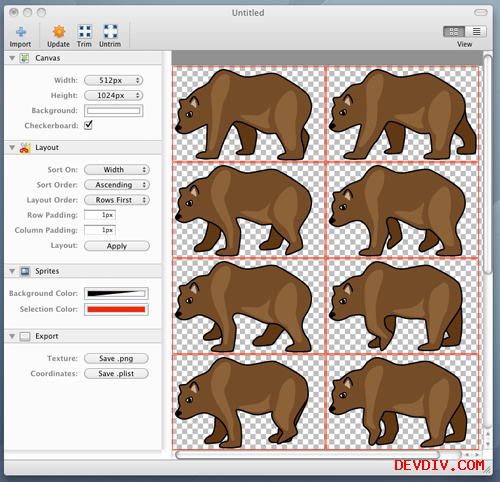
你会看到，所有的熊的图片都层叠在一起。我们需要将他们摊开放在spritesheet上面，因此在Layout部分点击“Apply”来排序。

当你这样做以后，你会注意到，默认的画布（５１２×５１２）太小了，不足以把所有的熊图片装下。所以，还会有一些图片层叠在一起。因此，我们在Canvas部分把画布改成５１２×１０２４，然后在Layout部分点击“Apply”来重新排序它们。

我们马上要完成了–但是，请注意，有些熊的图片比其它宽一些。如果你看一下原图，你会发现和原图尺寸不一样了–这是因为，Zwoptex在默认情况下会把图片周围的透明区域剪裁掉。

对于这些图片，它们并不是我们最终需要的，因为对于动画来说，这些图片的位置信息已经错乱了（由于透明区域的裁剪）。还好，这非常容易解决–在工具栏上选择”Untrim“，然后再点”Apply“。

这时，你的窗口可能和下图类似：



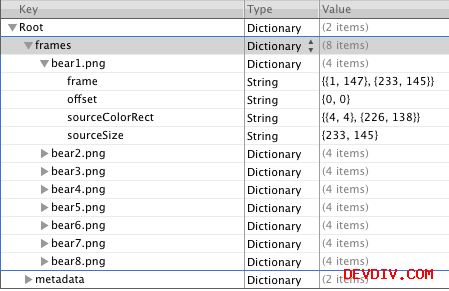
就这么多，让我们保存spritesheet图片和定义，这样我们就可以在程序中使用它们啦.

点击Export部分的”Save.png“，把这个文件取名为”AnimBear.png“并保存到你的resources文件夹下面。然后点击”Save.plist“，命名为”AnimBear.plist“，同样保存到你的resources文件夹下面。

更新：当我们在Zwoptex里面点击保存的时候，确保选择”Cocos2d“作为导出格式，否则你的plist文件就不能正确在cocos2d里面使用！谢谢Muhammad在评论部分给我指出来了！

**现在，让我们回到XCode，然后把刚刚这两个文件加进去。或击点击Resources文件夹，选择”Add\Existing　Files。。。“，选择AnimBear.png和AnimBear.plist文件，然后点增加。**

好了，让我们打开AnimBear.plist文件，看看Zwoptex到底为我们做了些什么事。你将会看到它仅仅是一个包含两个section的属性文件–两个部分分别为frames和metadata。在frames部分，包含了一系列的对spritesheet中每个图片的描述信息，这些描述信息里面包含了图片在spritesheet中的位置、大小和名字等信息。很cool，不是吗？



但是，如果你能让这只熊动起来，那将会更酷！下面就跟着我，一步步地让熊动起来！

**简单的动画**

首先，让我们把熊放在屏幕中间，然后循环播放所有的动画帧，这样看起来熊就在永远的移动，这里仅仅是先让代码可以跑起来。

因此，让我们在HelloWorldScene.h里面增加一些属性吧，在那个文件中做以下修改：

// Add inside the HelloWorld interface

CCSprite \*\_bear;

CCAction \*\_walkAction;

CCAction \*\_moveAction;

BOOL \_moving;// Add after the HelloWorld interface

@property (nonatomic, retain) CCSprite \*bear;

@property (nonatomic, retain) CCAction \*walkAction;

@property (nonatomic, retain) CCAction \*moveAction;

实际上，我们并不是马上需要所有的这些属性，但是，我们把它们先定义在这里，这样，等下我们就不用回过头来再改代码了。

现在，有趣的部分来了！打开HelloWorldScene.m，然后作如下改动：

// At the top, under @implementation

@synthesize bear = \_bear;

@synthesize moveAction = \_moveAction;

@synthesize walkAction = \_walkAction;// In dealloc

self.bear = nil;

self.walkAction = nil;

// Replace the init method with the following -(id) init { if((self = [super init])) { // Add the stuff from below! } return self; }

为了获得动画效果，我们有５个步骤需要做。接下，将会一个步骤一个步骤给大家讲解。把下面的一些代码片断按顺序增加到你的init的Add　the　stuff　from　below注释后面。

**1) 缓冲sprite帧和纹理**

**[[CCSpriteFrameCache sharedSpriteFrameCache] addSpriteFramesWithFile:**

**@”AnimBear.plist”];**

首先，调用CCSpriteFrameCache的addSpriteFramesWithFile方法，然后把Zwoptex生成的plist文件当作参数传进去。这个方法做了以下几件事：

* 寻找工程目录下面和输入的参数名字一样，但是后缀是.png的图片文件。然后把这个文件加入到共享的CCTextureCache中。（这我们这个例子中，就是加载AnimBear.png）
* 解析plist文件，追踪所有的sprite在spritesheet中的位置，内部使用CCSpriteFrame对象来追踪这些信息。

**2) 创建一个精灵批处理结点**

**CCSpriteBatchNode \*spriteSheet = [CCSpriteBatchNode**

**batchNodeWithFile:@"AnimBear.png"];**

**[self addChild:spriteSheet];**

接下来，创建CCSpriteBatchNode对象，把spritesheet当作参数传进去。spritesheet在cocos2d中的工作原理如下：

* 你创建一个CCSpriteBatchNode对象，通过传递一个包含所有sprite的spritesheet的名字作为参数，并把它加入到当前场景之中。
* 接下来，你从spritesheet中创建的任何sprite，你应该把它当作CCSpriteBatchNode的一个孩子加进去。只要sprite包含在spritesheet中，那么就没问题，否则会出错。
* CCSpriteBatchNode可以智能地遍历它的所有的孩子结点，并通过一次OpenGL　ES　call来渲染这些孩子，而不是以前每个sprite都需要一个OpenGL　call，这样渲染速度就会更快。

注意：CCSpriteBatchNode以前叫做CCSpriteSheet，你可能会在一起比较老的代码里面看见它。

**3) 收集帧列表**

**NSMutableArray \*walkAnimFrames = [NSMutableArray array];**

**for(int i = 1; i <= 8; ++i) {**

**[walkAnimFrames addObject:**

**[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:**

**[NSString stringWithFormat:@"bear%d.png", i]]];**

**}**

为了创建一系列的动画帧，我们简单地遍历我们的图片名字（它们是按照Bear1.png–>Bear8.png的方式命名的），然后使用共享的CCSpriteFrameCache来获得每一个动画帧。记住，它们已经在缓存里了，因为我们前面调用了addSpriteFramesWithFile方法。

**4) 创建动画对象**

**CCAnimation \*walkAnim = [CCAnimation**

**animationWithFrames:walkAnimFrames delay:0.1f];**

接下来，我们通过传入sprite帧列表来创建一个CCAnimation对象，并且指定动画播放的速度。我们使用０.１来指定每个动画帧之间的时间间隔。

**5) 创建sprite并且让它run动画action**

**CGSize winSize = [CCDirector sharedDirector].winSize;**

**self.bear = [CCSprite spriteWithSpriteFrameName:@"bear1.png"];**

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

**self.walkAction = [CCRepeatForever actionWithAction:**

**[CCAnimate actionWithAnimation:walkAnim restoreOriginalFrame:NO]];**

**[\_bear runAction:\_walkAction];**

**[spriteSheet addChild:\_bear];**

**我们首先通过spriteframe来创建一个sprite，并把它放在屏幕中间。然后，生成CCAnimationAction，并赋值给场景的walkAction属性，最后让熊来运行这个action。**

最后，我们把熊加个场景中–把它当作spritesheet的孩子加到spritesheet中去。注意，如果在这里我们没有把它加到spritsheet中，而是加到当前层里面的话。那么我们将得不到spritesheet为我们带来的性能提升！！！

完成了！

就这么多！编译并运行，你将会看到一只熊欢快地在屏幕上面走动！



**基于熊的移动方向改变熊的朝向**

一切看起来好极了–除了我们并不想让熊自己独自一个人走之外，那太危险了！如果我们能够通过点击屏幕就可以想让熊往哪走，它就会往哪走的话，那就太棒了.

因此，在HelloWorldScene.m文件中做如下修改：

// Comment out the runAction method in the init method:

//[\_bear runAction:\_walkAction];// And add this to the init method

self.isTouchEnabled = YES;

// Add these new methods -(void) registerWithTouchDispatcher { [[CCTouchDispatcher sharedDispatcher] addTargetedDelegate:self priority:0 swallowsTouches:YES]; }

-(BOOL) ccTouchBegan:(UITouch \*)touch withEvent:(UIEvent \*)event { return YES; }

-(void) ccTouchEnded:(UITouch \*)touch withEvent:(UIEvent \*)event { // Stuff from below! }

-(void)bearMoveEnded { [\_bear stopAction:\_walkAction]; \_moving = FALSE; }

开始之前，我们先把init方法中的运行行走action的代码注释掉，因为我们并不想让熊自己动，直到我们发出指令之后，它才能动！

我也设置了层能够接收touch事件，然后实现了registerWithTouchDispatcher和ccTouchBegan方法。如果你对使用这个方法的好处感到好奇的话（为什么不使用ccTouchesBegan呢？），你可以查看[《如何在cocos2d里面制作基于Tile地图的游戏教程》](http://www.cnblogs.com/andyque/archive/2011/04/11/2012852.html)。（当前是英文，以后会更新）

当bearMoveEnded方法被调用的时候，我们想让熊停止任何正在运行的动画，并且设置标记为不再移动。

看到ccTouchEnded方法，那里就是待会要实现功能的地方。那儿有许多东西要实现，因此，让我把它们分解成一些小片断，一步步向众位看官道来：

**1) 计算touch坐标点**

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

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

**touchLocation = [self convertToNodeSpace:touchLocation];**

这里没什么新东西–我们仅仅是把touch点转换成我们要使用的局部坐标系点。

**2) 设置熊移动速度**

**float bearVelocity = 480.0/3.0;**

这里，我们设置了熊的移动速度。我假设熊要花３秒钟时间才能从iphone屏幕（４８０个像素宽）的一头移动到另一头。因此，简单地用４８０个像素除以３秒。

**3) 计算x轴和y轴的移动量**

**CGPoint moveDifference = ccpSub(touchLocation, \_bear.position);**

接下来，我们需要计算出熊相当于x轴和y轴移动了多远。我们简单地使用touch坐标减去熊当前的坐标。这里使用了cocos2d的一个帮助函数ccpSub来实现这个功能。

**4) 计算实际移动的距离**

**float distanceToMove = ccpLength(moveDifference);**

我们需要计算出熊实际移动的距离（欧几里德距离）。cocos2d里面也提供了一个帮助函数来做这个事情，这个函数就是ccpLength，用来求一个向量的长度。

**5) 计算移动需要花费的时间**

**float moveDuration = distanceToMove / bearVelocity;**

最后，我们需要计算出熊要花费多长时间来走完这段路程，只需要拿距离除以速率就可以了。

**6) 按照需要翻转动画**

**if (moveDifference.x < 0) {**

**\_bear.flipX = NO;**

**} else {**

**\_bear.flipX = YES;**

**}**

接下来，我们通过判断移动的差值，如果小于０，那么就不需要翻转动画，否则，就需要翻转。因为我们的原画里面，熊就是往左移动的，因此，当熊往左移动时，我们不需要翻转动画，而往右移动的时候，只需要翻转动画。

我们的第一直觉可能是用图片编辑器重新创建另一套朝向不同的熊的动画序列图，然后使用它们。但是，cocos2d里面有一种更容易的方式（也更高效）–我们仅仅翻转已经存在的图片就行了。

这种方式可行，实现上，我们只是设置了运行动画的sprite的flip属性，但是它会使所有相关的动画帧也相应地翻转。在这个例子中，当熊往右行走的时候，我们就设置熊的flipX为Yes。

**7) 运行合适的action**

**[\_bear stopAction:\_moveAction];**if (!\_moving) {

[\_bear runAction:\_walkAction];

}

self.moveAction = [CCSequence actions: [CCMoveTo actionWithDuration:moveDuration position:touchLocation], [CCCallFunc actionWithTarget:self selector:@selector(bearMoveEnded)], nil];

[\_bear runAction:\_moveAction]; \_moving = TRUE;

接下来，我们停止任何正在运行的action。（因为我们将要覆盖任何已经存在的命令，让它移动到其它地方去！）。当然，如果我们没有移动，我们也需要停止任何动画action。（防止意外情况）。如果我们已经在移动了，那么我们当然需要停止，因为这样就不会影响后面运行的action。（这段话有些绕口，大家仔细体会，就是说，我们在让一个sprite运行一个atcion之前，最好先让它停止任何已经在运行的action。）

最后，我们创建移动action，指定移动的位置，花费的时间，并且指定一个回调函数，这个函数会在熊移动到指定位置之后被调用。我们也需要记录，我们移到那个点了！

完成啦！

写了好多代码啊—但是值得，不是吗？编译并运行，然后点击屏幕，你将会看到一只熊在屏幕上面移动。



**何去何从?**

这里有这个教程的[完整源代码](http://www.raywenderlich.com/downloads/AnimBear.zip)。

现在，你应该知道如何在项目里面使用spritesheet了吧。你可以在你的项目中创建自己的动画，然后看看你到底能做些什么有趣的事情！just　do　it！

**如何使用cocos2d制作基于tile地图的游戏教程：第一部分 by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/1163/how-to-make-a-tile-based-game-with-cocos2d>

教程截图：



在这个２部分的教程中，我将会教大家如何使用cocos2d来做一个基于tile地图的游戏，当然还有Tiled地图编辑器。（我们小时候玩的小霸王小学机里面的游戏，大部分都是基于tile地图的游戏，如坦克大战、冒险岛、吞食天地等）我们将会创建一个忍者在沙漠中找西瓜吃的小游戏。-\_-

在第一部分教程中，我将教大家如何使用Tile来创建地图，怎样把地图加到游戏中，怎么让地图跟随玩家滚动，以及怎样使用对象层。

在第二部分教程中，我将介绍如何在地图中创建可碰撞的区域，如何使用tile属性，如何制作可拾取的物体和动态修改地图，还有确保忍者不要吃撑了！

如果你还没有准备好的话，你可能需要先从[《如何使用cocos2d来制作简单的iphone游戏》](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E6%9D%A5%E5%81%9A%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84iphone%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B1.html)系列教程开始学起，因为我们这个教程使用了大量的基本概念，而这些概念都可以从上面的教程中获取。

好了，让我们玩一玩tile地图吧！

**创建工程骨架**

让我们首先创建整个工程的骨架，这样可以确保今后我们需要的文件都包含进来了，并且能够跑起来。

因此，启动XCode，点击“File\New Project…”，选择cocos2d　Application　template，并且把工程命名为TileGame。

接下来，下载[游戏资源文件](http://www.raywenderlich.com/downloads/TileGameResources.zip)。这个资源文件包里包含了以下内容：

* 玩家sprite。这个图片和[《如何使用cocos2d来制作简单的iphone游戏》](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E6%9D%A5%E5%81%9A%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84iphone%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B1.html)差不多。
* 我使用[cxfr](http://thirdcog.eu/apps/cfxr)这个工具制作的一些音效。
* 我使用Garage　Band制作的一些背景音乐。（查看这篇[博文](http://www.raywenderlich.com/233/audio-101-for-iphone-developers-converting-and-recording)获得更多的信息）
* 我们将会使用的tile集合–它实际上会和tile地图编辑器一块儿使用，但是，我想把它放在这里，余下的事情会变得更容易。
* 一些额外的“特殊”的tile，我将会在后面加以说明。

一旦你获得了这些资源，解压并把它拖到你的工程的“Resources”分组下面。确保复选中“Copy　items into destination group’s folder (if needed)”，引用类型为“Relative　to　Project”，然后点击增加。

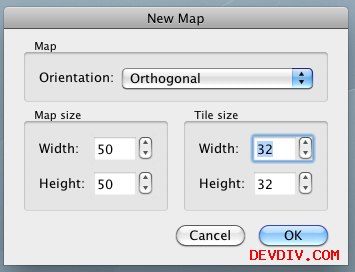
如果一切顺利，所有的文件应该都在你的工程里了。是时候制作我们的地图了！

**使用Tile来制作地图**

cocos2d支持使用[开源的Tile地图编辑器](http://dl.dbank.com/c0basyzwse)创建的TMX格式的地图。

（作者给出的网址现在打不开了，这是我在另一个地方找到的。我把它放到我的网盘里了，并且做了一个链接。如果有人下载不了，请留言。这个tile地图编辑器是java版的，其实还有一个at版的，但是java版的功能强大一些。但是，大家请注意，作者使用的是qt版本的，所以界面会有一些不一致，但这并不影响程序的使用。）

下载完之后，直接双击运行。点击File\New，然后会出现以下对话框：

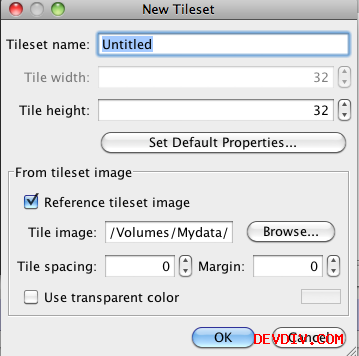


在 orientation部分，你可以选择Orthogonal（参考：  [Legend of Zelda](http://www.zelda.com/universe/game/zelda/)）或者Isometric（参考：  [Disgaea](http://disgaea.us/psp/)）。我们这里将选择Orthogonal。

接下来，设置地图的大小。记住，这个大小是以tile为单位的，而不是以像素为单位。我们将创建一个尽量小的地图，因此选择５０×５０.

最后，你指定每个tile的宽度和高度。你这里选择的宽度和高度要根据你的实际的tile图片的尺寸来做。这个教程使用的样例tile的尺寸是３２×３２，所以在上面的选项中选择３２×３２.

接下来，我们把制作地图所需要的tile集合导入进来。点击菜单栏上面的“TileSets”菜单，“New　Tileset…”，然后会出现下面的窗口：



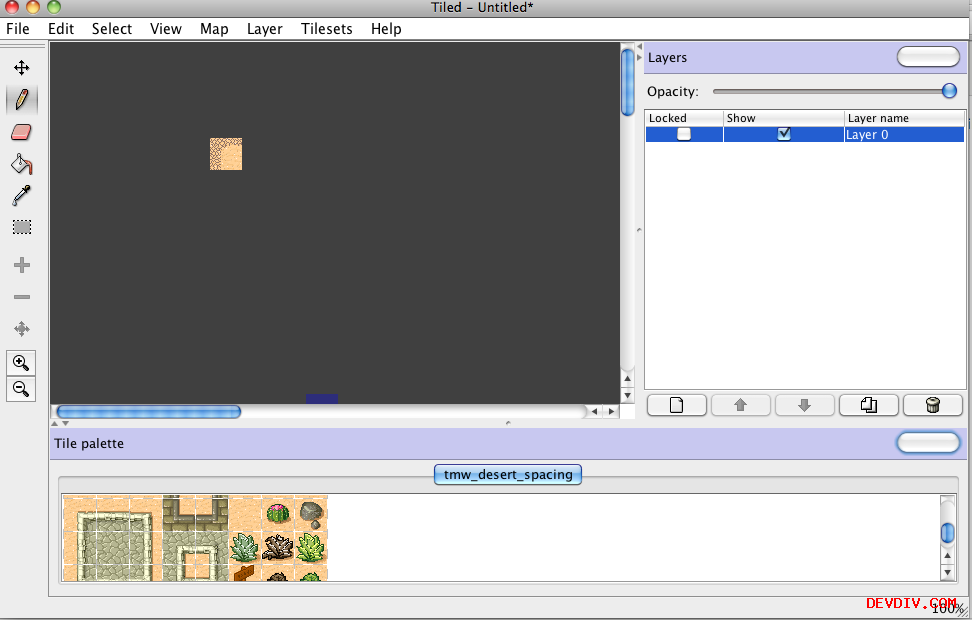
为了获得图片，点击Browse按钮，然后定位到你的TestGame文件夹，选择 tmw\_desert\_spacing.png文件，然后加到工程中去。它会基于文件名自动填充Name。然后把TileSet name命名为“tmw\_desert\_spacing.png”.同时，设置下面的Tile spacing和Margin都为1.

你可以保留宽度和高度为３２×３２，因为tile的实际大小也是这么多。至于margin和spacing，我还没找到任何好的文档解释如何设置这两个值，下面是我的个人看法：

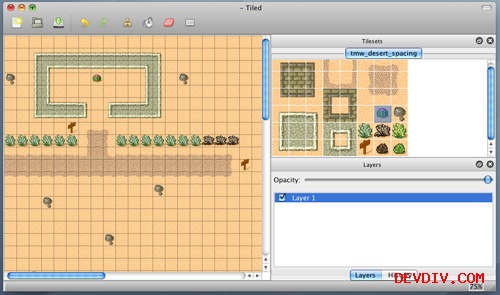
* **Margin就是当前的tile计算自身的像素的时候，它需要减去多少个像素（宽度和高度都包含在内）。（类比word、css的margin）**
* **Spacing** 就是相邻两个tile之间的间隔（同时考虑宽度和高度）（类比word、css的spacing）

如果你看看 tmw\_desert\_spacing.png，你将会看见每一个tile都有一个像素的空白边界围绕着，这意味着我们需要把margin和spacing设置为１.

一旦你选择ok，你将会看到Tilesets窗口中显示了一些tiles。现在，你可以制作地图了！点击工具栏上的“Stamp”按钮。点击Tile palette中的tile map，然后选择一个tile，然后再在地图上的任意位置单击，你就会看到你选中的tile出现在点中的地方了。



因此，继续制作地图吧—充分发挥你的聪明才智！确保增加至少一对建筑物在地图上，因为后面我们需要一些东西来做碰撞。



记住一些方便的快捷方式：

* 你可以在Tileset拾取器中拖出一个方框，一次选取多个tile。
* 你可以使用工具栏上的paint按钮来基于一个基准tile绘制整个地图。
* 你可以使用“View\Zoom　In…”和“View\Zoom out…”来放大和缩小地图。

一旦你完成了地图的绘制工作，在Layers选项卡的层上面双击（现在可以说是“Layer１”），然后重命名为“Background”。然后点击“File\Save”并且保存文件到你的工程的资源文件夹中，并且命名为“TileMap.tmx”。

后面我们将会使用这个tmx来做一些有趣的事情，好了，让我们把地图加载到游戏中去吧！

**把tile地图添加到cocos2d的场景中**

首先，第一件事情，右键点击Resources，选择“ Add\Existing Files…”，然后添加TileMap.tmx文件。

打开HelloWorldScene.h，然后添加一些成员变量，并且申明一声属性：

// Inside the HelloWorld class declaration

CCTMXTiledMap \*\_tileMap;

CCTMXLayer \*\_background;// After the class declaration

@property (nonatomic, retain) CCTMXTiledMap \*tileMap;

@property (nonatomic, retain) CCTMXLayer \*background;

然后在HelloWorldScene.m文件中做如下修改：

// Right after the implementation section

@synthesize tileMap = \_tileMap;

@synthesize background = \_background;// In dealloc

self.tileMap = nil;

self.background = nil;

// Replace the init method with the following -(id) init { if( (self=[super init] )) {

self.tileMap = [CCTMXTiledMap tiledMapWithTMXFile:@"TileMap.tmx"]; self.background = [\_tileMap layerNamed:@"Background"];

[self addChild:\_tileMap z:-1];

} return self; }

这里，我们调用CCTMXTiledMap类的一些方法，把我们刚刚创建的地图文件加载进去。

一些简明的CCTMXTiledMap的背景知识。它是一个CCNode，你可以设置它的位置和比例等。这个地图的孩子是一些层，而且提供了一个帮助函数可以让你通过层的名字得到层对象–我们上面就是通过这种方面获得地图背景的。每一个层都是一个CCSpriteSheet的子类，这里考虑了性能的原因–但是这也意味着每一个层只能有一个tile集。

因此，我们这里做的所有这些，就是指向一个tile地图，然后保存背景层的引用，并且把tile地图加到HelloWorld层中。

好了，就这么多！编译并运行工程，你将会看到地图的左下角出现在模拟器中。



还不错！但是，这还不是一个游戏！我们还需要三个东西：a）游戏主角，b）主角初使位置和c）能够移动视图，这样就好像是第一视角了。

好了，接下来让我们来解决这些问题。

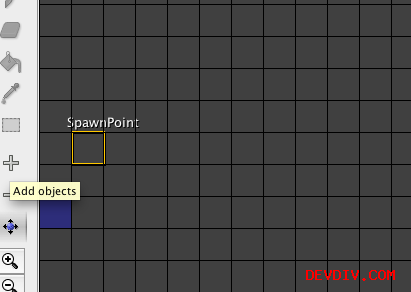
**tiled对象层和设置tile地图位置**

tiled支持两类层–tile层（就是我们目前使用的层），还有对象层。

对象层允许你在地图上圈出一些区域，来指定一些事件的发生。比如，你可能想制作一个区域，在那里怪物将会跳出来，或者是一个区域，只要进入就会死掉。这我们这个例子中，我们将创建一个区域来显示我们的游戏主角。

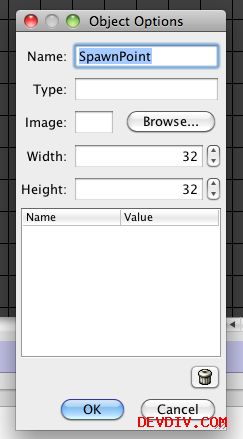
因此，找到Tiled的菜单，点击”  Layer\Add Object Group…”，命名为“Objects”，然后选择Ok。如果你绘制了地图，你将会注意到，它并没有绘制一个tile，而是画了一个很难看的灰色矩形，这个矩形我们之后可以扩展，使之能够包含多个tiles或者移动它。

我们只想要选择一个tile来让主角显示。因此，在你的地图上选择一个tile。这个区域的大小实际上并没有关系，因为我们仅仅使用x、y坐标。

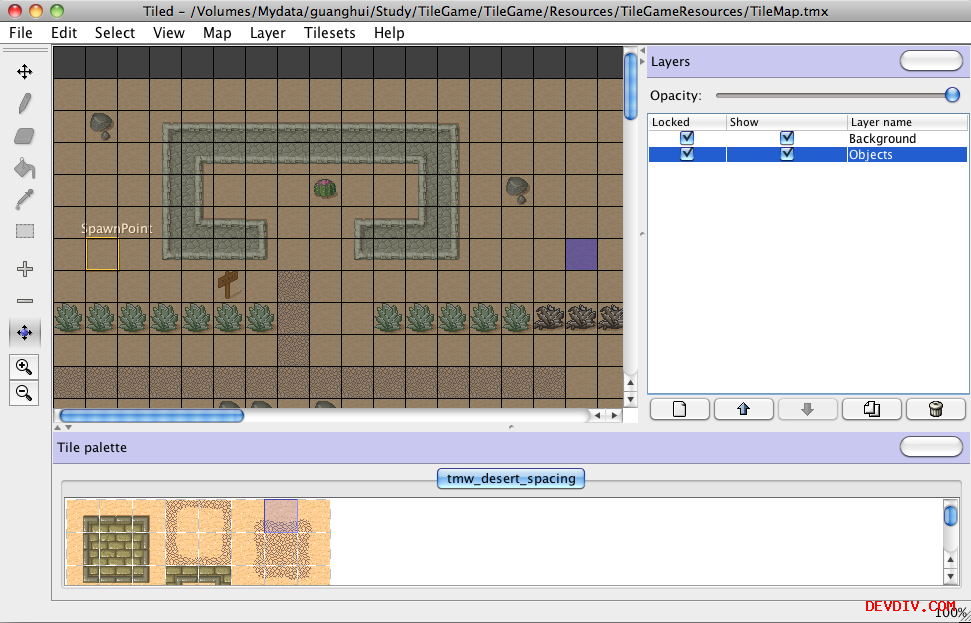


然后，上面的黄色对象上面点右键， 取名为“SpawnPoint”，然后选择Ok：

（添加对象方法很简单，看到左边的＋号和-号了吗？一个是增加对象，一个是删除对象。点中加号以后，用鼠标可以拖出一个矩阵区域。）



（下面给出一些技巧。如何把一个对象准确放置到Background的空白区域，只需要调整背景的opacity就可以了）



假设，你可以做一些奇思妙想。你把对象的类型命名为cocos2d里面的class的名字，然后它会为你创建那个类型（比如CCSprite），但是，我在cocos2d自带的源代码里面代不到这样的例子。更新：来自GeekAndDad.com的Tyler提供了之前版本的cocos2d里面使用这种“妙想”的方法，但是，由于背景有白色，所以被移除了。

不管怎么说–我们仅仅把这个类型设置为空就行了，最后cocos２d会为我们创建NSMutableDictionary，我们可以从中获得对象的各种属性，包含x，y坐标。

保存地图，然后返回XCode。在HelloWorldScene.h中做如下修改：

// Inside the HelloWorld class declaration

CCSprite \*\_player;// After the class declaration

@property (nonatomic, retain) CCSprite \*player;

同样，修改HelloWorldScene.m，代码如下：

// Right after the implementation section

@synthesize player = \_player;// In dealloc

self.player = nil;

// Inside the init method, after setting self.background CCTMXObjectGroup \*objects = [\_tileMap objectGroupNamed:@"Objects"]; NSAssert(objects != nil, @”‘Objects’ object group not found”); NSMutableDictionary \*spawnPoint = [objects objectNamed:@"SpawnPoint"]; NSAssert(spawnPoint != nil, @”SpawnPoint object not found”); int x = [[spawnPoint valueForKey:@"x"] intValue]; int y = [[spawnPoint valueForKey:@"y"] intValue];

self.player = [CCSprite spriteWithFile:@"Player.png"]; \_player.position = ccp(x, y); [self addChild:\_player];

[self setViewpointCenter:\_player.position];

好了，让我们先歇会儿，来解释一下对象层和对象组。首先，注意你通过CCTMXTiledMap对象的objectGroupNamed方法来获得对象层（而不是layerNamed方法）。它返回一个特殊的CCTMXObjectGroup对象。

我们然后调用CCTMXObjectGroup类的objectNamed方法来获得一个NSMutableDictionary，这个字典包含了关于对象的大量信息，包括x和y坐标值，宽度和高度。在这个例子中，我们只关心x和y坐标，因此，我们提取出这两个信息，并且设置player的位置。

最后，我想设置这个视图为玩家所在的位置。因此，添加下面一个新方法到文件中：

-(void)setViewpointCenter:(CGPoint) position {CGSize winSize = [[CCDirector sharedDirector] winSize];

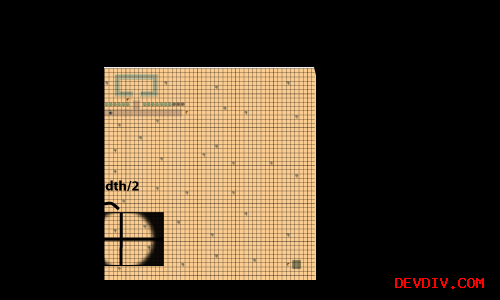
int x = MAX(position.x, winSize.width / 2); int y = MAX(position.y, winSize.height / 2); x = MIN(x, (\_tileMap.mapSize.width \* \_tileMap.tileSize.width) - winSize.width / 2); y = MIN(y, (\_tileMap.mapSize.height \* \_tileMap.tileSize.height) - winSize.height/2); CGPoint actualPosition = ccp(x, y);

CGPoint centerOfView = ccp(winSize.width/2, winSize.height/2); CGPoint viewPoint = ccpSub(centerOfView, actualPosition); self.position = viewPoint;

}

好了，让我解释一下。假设这个函数是设置camera的中心。我们允许用户传入地图上任何x、y坐标值–但是如果你仔细想一下，有些东西我们并不想让它显示出来–比如，我们不想让屏幕超过地图的边界（那些区域仅仅是一个空白区域！）

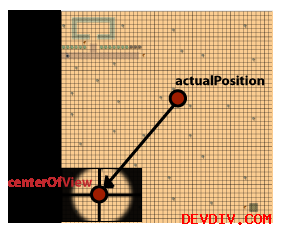
比如，看看下面这幅图：



看一下，什么时候camera的中心会小于winSize.width/2或者winSize.height/2，部分视图将会在屏幕之外？类似的，我们需要检查上面的界限区间，也和我们这里的情形一样。

因此，我们把这个函数看作是设置camera的视角中心点。然而。。。那不完全是我们想要的。在cocos2d里面和一种方式可以操作一个CCNode的camera，但是那会使事情变得更困难：那种方法是[移动整个层](http://www.cocos2d-iphone.org/forum/topic/2229#post-14529)。

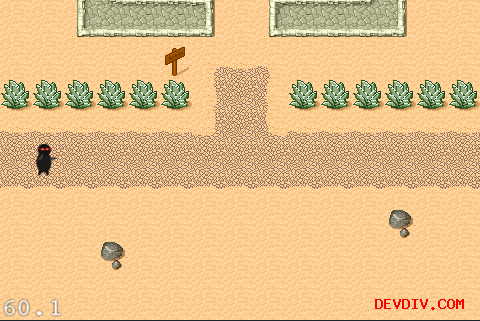
看看下面的图：



想像一个大的地图，我们查看从０到winSize.height/width的坐标。我们的视图的中心点是centerOfView，而且我们知道我们要把这个中心设置到哪里（actualPositon）。因此，为了使实际的位置和视图中心相吻合，我们只需要把地图往左下角移动即可！

这个可以通过使实际的位置减去视图的中心位置来实现，然后设置HelloWorld层到那个点。

唉！太多理论了–让我们看点实际的吧！编译并运行项目，如果一切顺利，你将会看到忍者在场景当中，然而视角也移过来了。



**使忍者移动**

我们已经有一个好的开端了，但是我们的忍者只是站在那儿不动！这可不像真正的忍者！

让我们使忍者动起来吧，只需要让忍者移动到用户点击的地方就行了。在HelloWorldScene.m中增加以下代码：

// Inside init method

self.isTouchEnabled = YES;-(void) registerWithTouchDispatcher

{

[[CCTouchDispatcher sharedDispatcher] addTargetedDelegate:self

priority:0 swallowsTouches:YES];

}

-(BOOL) ccTouchBegan:(UITouch \*)touch withEvent:(UIEvent \*)event { return YES; }

-(void)setPlayerPosition:(CGPoint)position { \_player.position = position; }

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

CGPoint touchLocation = [touch locationInView: [touch view]]; touchLocation = [[CCDirector sharedDirector] convertToGL: touchLocation]; touchLocation = [self convertToNodeSpace:touchLocation];

CGPoint playerPos = \_player.position; CGPoint diff = ccpSub(touchLocation, playerPos); if (abs(diff.x) > abs(diff.y)) { if (diff.x > 0) { playerPos.x += \_tileMap.tileSize.width; } else { playerPos.x -= \_tileMap.tileSize.width; } } else { if (diff.y > 0) { playerPos.y += \_tileMap.tileSize.height; } else { playerPos.y -= \_tileMap.tileSize.height; } }

if (playerPos.x <= (\_tileMap.mapSize.width \* \_tileMap.tileSize.width) && playerPos.y <= (\_tileMap.mapSize.height \* \_tileMap.tileSize.height) && playerPos.y >= 0 && playerPos.x >= 0 ) { [self setPlayerPosition:playerPos]; }

[self setViewpointCenter:\_player.position];

}

首先，在init方法中设置层能够接收touch事件。如果我们覆盖registerWithTouchDispatcher方法，来使这个层能够处理目标touch事件。这样会导致ccTouchBegan和ccTouchEnded方法被调用（注意是单数形式，而不是复数形式的ccTouchesBegan和ccTouchesEnded方法）

你可能会问，为什么我要讲这个，因为我们在 [《如何使用cocos2d来制作简单的iphone游戏》](http://www.cnblogs.com/andyque/articles/1990716.html)里面使用的是ccTouchesBegan和ccTouchesEnded方法。那两个方法可以，在这个教程里用两种方法都可以。但是，我想向大家介绍一个新方法，因为它有两个优点：

* “你不需要处理NSSet，划分UITouch集合并调度的工作全部由cocos2d框架来完成。每一次方法调用，你只获得了一个UITouch。“
* “你可以在ccTouchBegan中返回yes，这样当前的层就可以接收touch事件回调。而且，只有当你返回yes的时候，才会响应move/ended/cancelled回调. 这个就使你从一些复杂的多触摸判断中解放出来了。

不管怎么说，在我们的ccTouchEnded里面，我们转换屏幕touch坐标为局部view坐标，然后再转换成GL的坐标。这两个步骤，在新的cocos2d版本中，只需要一步完成，即调用 [self convertToNodeSpace:touchLocation].就可以了。

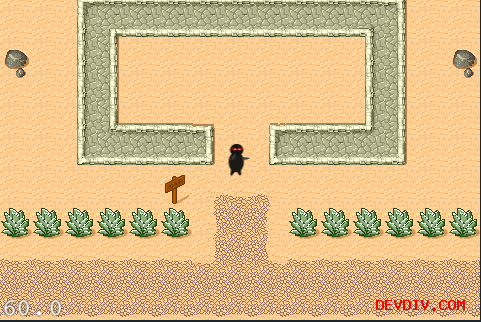
这是因为，touch位置只是告诉我们屏幕视口的坐标（比如１００，１００）。但是，我们我们滚动了地图，这个位置实际可能对应地图的（８００，８００）。因此，调用这个方法基于我们当前层的位置来决定touch的偏移。

接下来，计算出touch点和player的位置之差。我们必须基于touch位置选择一个方向，因此，首先，我们需要计算出是上下移动还是左右移动。然后，我们比较正负值，决定具体的方向。

相应的，我们再调整player的位置，并且设置player的位置为视口的中心位置，这个在上一节中已经用到了。

更新：注意，我们不得不添加一个安全检查，来确保我们的player不会移到地图之外！这一点，是Geek&Dad指出来的，谢谢你！

**编译并运行！你现在可以点击鼠标，想让尽者移到哪，它就移到哪儿！**



**何去何从？**

这只是这个教程的一部分。此时，你应该了解一些创建tile地图的基础了，而且知道如何把它导入到游戏当中。

这里有我们目前为止用的[完整源代码](http://www.raywenderlich.com/downloads/TileGamePart1.zip)。

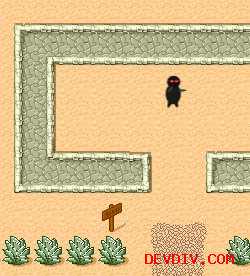
接下，期待[第二部分教程吧](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E7%A2%B0%E6%92%9E%E6%A3%80%E6%B5%8B%E5%92%8C%E6%94%B6%E9%9B%86%E7%89%A9%E5%93%81%EF%BC%9A%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E5%88%B6%E4%BD%9C%E5%9F%BA%E4%BA%8Etile.html)！在那里，我将教大家如何在地图中添加碰撞检测，如果使我们的忍者沿着墙壁快乐的奔跑！

**碰撞检测和收集物品：如何使用cocos2d制作基于tiled地图的游戏：第二部分 by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/1186/collisions-and-collectables-how-to-make-a-tile-based-game-with-cocos2d-part-2>

程序截图：



这篇教程是《如何使用cocos2d制作基于tiled地图的游戏》的第二部分。在上一个教程中，我们创建了一个简单的基于tiled地图的游戏，里面有一个忍者在沙漠里寻找可口的西瓜！

在[第一部分教程](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E5%88%B6%E4%BD%9C%E5%9F%BA%E4%BA%8Etile%E5%9C%B0%E5%9B%BE%E7%9A%84%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B%EF%BC%9A%E7%AC%AC%E4%B8%80.html)中，我们介绍了如何基于tiled创建地图，怎样把地图增加到游戏中去，以及如何滚动地图来跟随主角移动、还有如何使用对象层。

在这部分教程中，我们将会介绍如何在地图中制作可以碰撞的区域，如何使用tile属性，如果收集游戏物品并且动态地修改地图、如何确保你的忍者不会吃得太饱！

因此，让我们继续我们上篇教程所学并且让它更像一个真实的游戏吧！

**tiled地图和碰撞检测**

你可能已经注意到了，目前我们的忍者可以毫无阻拦地穿过墙壁和障碍物。他是一个忍者，但是即使是真正的忍者，他也没这么厉害啊！

因此，我们需要找到一种方法，通过把一些tile标记成“可碰撞的”，这样的话，我们就可以防止玩家穿过那些点的位置。有很多方法可以做得到（包括使用对象层），但是，我想向你们展示一种新的技术。我认为它更高效，并且也是一次好的学习锻炼–使用一个元层（meta　layer）和层属性。

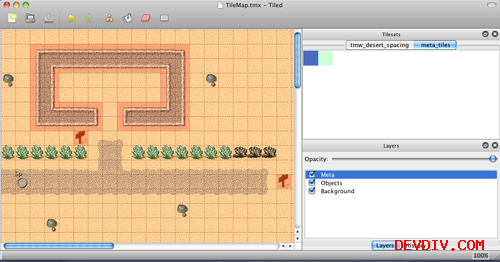
让我们开始动手吧！再一次启动Tiled软件，点击“Layer\Add　tile　Lyaer…”，并且命名为“Meta”，然后选择OK。我们将在这个层里面加入一些假的tile代表一些“特殊tile”。

因此，现在我们需要增加我们的特殊tile。点击“Map\New　tileset…”，在你的Resources文件夹下面找到mate\_tiles.png，然后选择打开。设置Margin和Spacing都为１并点击OK。

这时，你可以在Tilesets区域看到一个新的标签。打开它，而且你会看到２个tile：一个红色的和一个绿色的。

这些tile并没有什么特殊的东西–我只是制作了一个简单的图片，里面包含了一个红色的和一个绿色的半透明tile。接下来，我们把红色的tile当作是“可碰撞的”（后面我们会用到绿色的），然后，合适地绘制我们的场景。

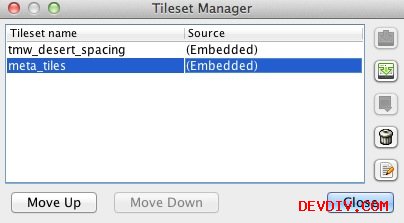
\_因此，确保Meta层被选中，选择stamp工具，选择红色的tile，然后把任何你不想让忍者通过的地图都涂一遍。当你做完的时候，应该看起来像下面的图示一样：



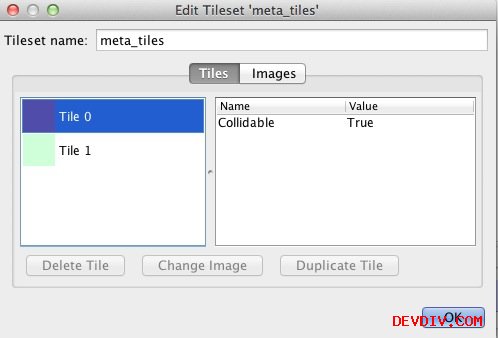
接下来，我们可以设置tile的属性，这样的话，我们在代码中就可以识别这个tile是“可以碰撞的（穿不过去的）”。在Tileset里面的红色tile上在，右击，选择“Properties…“。增加一个新的属性，叫做”Collidable“，并且设置成”Ture“：

（由于版本的关系，我这里补充我上传的Tiled编辑器（java版本）如何设置属性！！！）

首先，选择TileSets–>TileSetManager，并选中meta\_tiles，出现如下所示图：



然后点击右下角的“Edit”按钮（就是垃圾回收站下面那个图标），接下来会出现下图所示：（接着就选中红色tile和绿色tile，然后添加Collidable属性并设置为True就ok啦）



保存map，并返回Xcode。在HelloWorldScene.h中做如下改动：

// Inside the HelloWorld class declaration

CCTMXLayer \*\_meta;

// After the class declaration @property (nonatomic, retain) CCTMXLayer \*meta;

同时修改HelloWorldScene.m文件如下：

// Right after the implementation section

@synthesize meta = \_meta;

// In dealloc self.meta = nil;

// In init, right after loading background self.meta = [\_tileMap layerNamed:@"Meta"]; \_meta.visible = NO;

// Add new method - (CGPoint)tileCoordForPosition:(CGPoint)position { int x = position.x / \_tileMap.tileSize.width; int y = ((\_tileMap.mapSize.height \* \_tileMap.tileSize.height) - position.y) / \_tileMap.tileSize.height; return ccp(x, y); }

好了，让我们先停一会儿。像之前一样，我会meta层声明了一个成员变量，而且从tile　map中加载了一个引用。注意，我们把这个字当作是不可见的，因为我们并不想看见这些对象，它们的存在只是为了说明，那个区域是可以碰撞的。

接下来，我们增加一个新的帮助方法，这个方法可以帮助我们把x，y坐标转换成”tile坐标“。每一个tile都有一个坐标，从左上角的（０，０）开始，到右下角的（４９，４９）。（本例中，地图的大小是４９×４９）



上面的截屏是java版本的tiled界面。能否显示tile的坐标，我不确定这个功能在QT版本的tiled中是否存在。不管怎么说，我们将要使用的一些功能会使用tile坐标，而不是x，y坐标。因此，我们需要一种方式，将x，y坐标转换成tile坐标。这正是那个函数所需要做的。

获得x坐标非常容易–我们只需要让它除以一个tile的宽度就可以了。为了得到y坐标，我们不得不翻转一些东西，因为，在cocos2d里面（０，０）是在左下角的，而不是在左上角。

接下来，把setPlayerPosition替换成以下内容：

CGPoint tileCoord = [self tileCoordForPosition:position];

int tileGid = [\_meta tileGIDAt:tileCoord];

if (tileGid) {

NSDictionary \*properties = [\_tileMap propertiesForGID:tileGid];

if (properties) {

NSString \*collision = [properties valueForKey:@"Collidable"];

if (collision && [collision compare:@"True"] == NSOrderedSame) {

return;

}

}

}

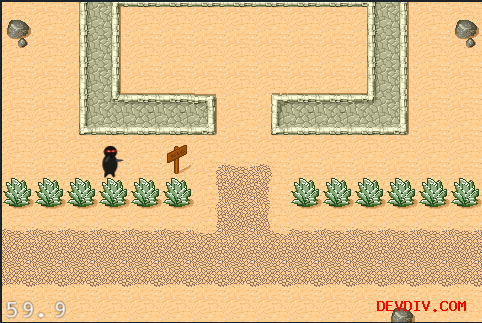
\_player.position = position;

在这里，我们把玩家的x，y坐标转换成tile坐标。然后，我们使用meta层中的tileGIDAt方法来获取指定位置点的GID号。

对了，什么是GID呢？GID代表”全球唯一标志符“（我个人意见）。但是，在这个例子中，我认为它只是我们使用的tile的一种标识，它可以是我们想要移动的红色区域。

当我们使用GID来查找指定tile的属性的时候。它返回一个属性字典，因此，我们可以遍历一下，看是否有”可碰撞的“物体被设置成”true“，或者是gij仅仅就是那样。编译并运行工程，因此还没有设置玩家的位置。

就这么多！编译并运行程序，它将会向你展示，现在你不能够通过那些红色的tile组成的地方了吧：

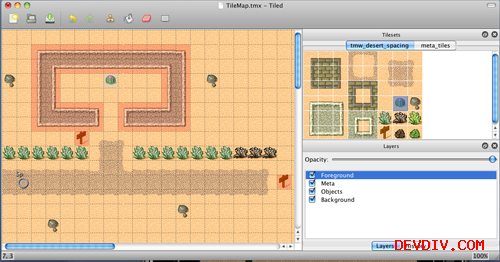


**动态修改Tiled　Map**

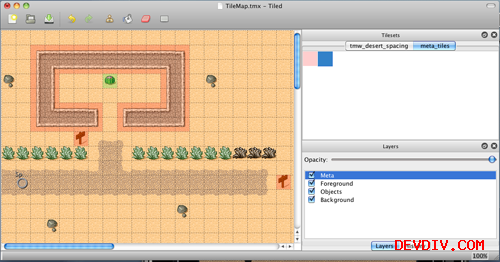
目前为此，我们的忍者已经有一个比较有意思的冒险啦，但是，这个世界有一点点无趣。而且简单无任务事可做！加上，我们的忍者看起来比较贪吃，而且背景将会随着玩家移动而移动。因此，让我们创建一些东西让忍者来玩吧！

为了使之可行，我将不得不创建一个前景层，这样做可以让用户收集东西。那样的话，我们仅仅从前景层中删除不用的tile（当tile被玩角拾取的时候），这个过程中，背景将会随之移动。

因此，打开Tiled，选择”Layer\Add Tile Layer…“，把这个层命名为”Foreground“，然后选择OK。确保前景层被选择，而且增加一对可以拾取的物品在游戏中。我喜欢放置一些向西瓜或者别的什么东西。



现在，我们需要把这些tile标记成可以拾取的，类似的，参照我们是如何把tile标志成可以碰撞的。选择Meta层，转换到Meta\_tiles。现在，我们需要使这些tile可以拾取，点击”Layer\Move　Layer　Up“来确保你的meta层是在最顶层，并且保持绿色可见的。



接下来，我们需要为tile增加属性，这样把它标记成可拾取的。点键点击Tilesets选项卡里的绿色的tile，然后点“Properties…”，再增加一个新的属性，命名为“Collectable”，值设置为“True”。

保存地图，然后返回到Xcode。在HelloWorldScene.h中做如下修改：

// Inside the HelloWorld class declaration

CCTMXLayer \*\_foreground;

// After the class declaration @property (nonatomic, retain) CCTMXLayer \*foreground;

同时，相应地修改HelloWorldScene.m：

// Right after the implementation section

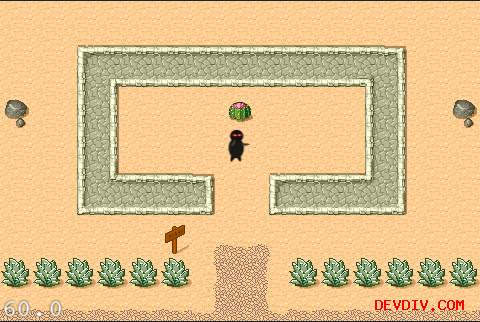
@synthesize foreground = \_foreground;

// In dealloc self.foreground = nil;

// In init, right after loading background self.foreground = [\_tileMap layerNamed:@"Foreground"];

// Add to setPlayerPosition, right after the if clause with the return in it NSString \*collectable = [properties valueForKey:@"Collectable"]; if (collectable && [collectable compare:@"True"] == NSOrderedSame) { [\_meta removeTileAt:tileCoord]; [\_foreground removeTileAt:tileCoord]; }

这里是一个常用的方法，用来保存前景层的句柄。不同之处在于，我们测试玩家正朝之移动的tile是否含有“Collectable”属性。如果有，我们就使用removeTileAt方法来把tile从mata层和前景层中移除掉。编译并运行工程，现在你的忍者可以尝尝西瓜的滋味啦！



**创建一个计分器**

我们忍者非常高兴地吃西瓜啦，但是，作为一个游戏玩家，我们想知道自己到底吃了多少个西瓜。你懂的，我们并不想让他吃得太胖。

通常的做法是，我们在层上面添加一个label。但是，等一下：我们在不停地移动这个层，那样的话，label就会看不到了，怎么办？

这是一个非常好的机会，如果在一个场景中使用多个层–这正是我们现在面临的难题。我们将保留HelloWorld层，但是，我们会再增加一个HelloWorldHud层来显示我们的label。（Hud意味着Heads　up　display，大家可以google一下，游戏中常用的技术）

当然，这两个层之间需要一种方式联系起来–Hud层应该知道什么时候忍者吃了一个西瓜。有许许多多的方式可以使２个不同的层相互通信，但是，我只介绍最简单的。我们在HelloWorld层里面保存一个HelloWorldHud层的句柄，这样的话，当忍者吃了一个西瓜就可以调用Hud层的一个方法来进行通知。

因此，在HelloWorldScene.h里面增加下面的代码：

// Before HelloWorld class declaration

@interface HelloWorldHud : CCLayer

{

CCLabel \*label;

}

- (void)numCollectedChanged:(int)numCollected; @end

// Inside HelloWorld class declaration int \_numCollected; HelloWorldHud \*\_hud;

// After the class declaration @property (nonatomic, assign) int numCollected; @property (nonatomic, retain) HelloWorldHud \*hud;

同样的，修改HelloWorldScene.m文件：

// At top of file

@implementation HelloWorldHud

-(id) init { if ((self = [super init])) { CGSize winSize = [[CCDirector sharedDirector] winSize]; label = [CCLabel labelWithString:@"0" dimensions:CGSizeMake(50, 20) alignment:UITextAlignmentRight fontName:@"Verdana-Bold" fontSize:18.0]; label.color = ccc3(0,0,0); int margin = 10; label.position = ccp(winSize.width - (label.contentSize.width/2) - margin, label.contentSize.height/2 + margin); [self addChild:label]; } return self; }

- (void)numCollectedChanged:(int)numCollected { [label setString:[NSString stringWithFormat:@"%d", numCollected]]; }

@end

// Right after the HelloWorld implementation section @synthesize numCollected = \_numCollected; @synthesize hud = \_hud;

// In dealloc self.hud = nil;

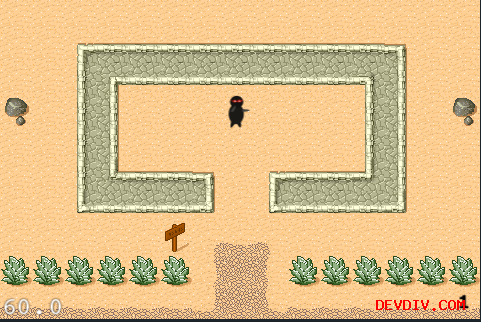
// Add to the +(id) scene method, right before the return HelloWorldHud \*hud = [HelloWorldHud node]; [scene addChild: hud];

layer.hud = hud;

// Add inside setPlayerPosition, in the case where a tile is collectable self.numCollected++; [\_hud numCollectedChanged:\_numCollected];

一切很明了。我们的第二个层从CCLayer派生，只是在它的右下角加了一个label。我们修改scene把第二个层也添加进去，然后传递一个Hud类的引用给HelloWorld层。然后修改HelloWorldLayer层，当计数器改变的时候，就调用Hud类的方法，这样就可以相应地更新Hud类了。

编译并运行，如果一切ok，你将会在屏幕右下角看到统计忍者吃西瓜的Label。



**来点音效和音乐**

如果没有很cool的音效和背景音乐的话，这就不能算作是一个完整的游戏教程了。

增加音效和音乐非常简单，只需在HelloWolrdScene.m作如下修改：

// At top of file

#import ”SimpleAudioEngine.h”

// At top of init for HelloWorld layer [[SimpleAudioEngine sharedEngine] preloadEffect:@”pickup.caf”]; [[SimpleAudioEngine sharedEngine] preloadEffect:@”hit.caf”]; [[SimpleAudioEngine sharedEngine] preloadEffect:@”move.caf”]; [[SimpleAudioEngine sharedEngine] playBackgroundMusic:@”TileMap.caf”];

// In case for collidable tile [[SimpleAudioEngine sharedEngine] playEffect:@”hit.caf”];

// In case of collectable tile [[SimpleAudioEngine sharedEngine] playEffect:@”pickup.caf”];

// Right before setting player position [[SimpleAudioEngine sharedEngine] playEffect:@”move.caf”];

现在，我们的忍者可以开怀大吃了！

**何去何从？**

这个系列的教程，就此完结了。距离上次翻译时间长了点。通过这个教程的学习，你对cocos2d里面的tiled　map的使用，应该有一个非常好的理解了。这里有这个教程的[完整源代码。](http://www.raywenderlich.com/downloads/TileGame.zip)

接下来，我会接着翻译下一篇，是原作者的一个朋友写的，这个系列教程的终结版：[《加入敌人和战斗：如果使用cocos2d制作基于tiled的地图：第三部分》。](http://www.cnblogs.com/andyque/archive/2011/05/07/2039481.html)

如果你看了这个教程，有什么好的意见或建议，可以自由发言，谢谢！

**加入敌人和战斗：如果使用cocos2d制作基于tiled地图的游戏：第三部分 by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

　　免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://geekanddad.wordpress.com/2010/06/22/enemies-and-combat-how-to-make-a-tile-based-game-with-cocos2d-part-3/>

程序截图：



　　这篇教程是Ray　Wenderlich的《如何使用cocos2d制作基于tiled地图的游戏》系列教程的后续。如果你还没有看过前面两部分的教程，可以在我的博客上找到另外两篇我翻译Ray的教程。

　　在第二部分教程中，Ray教大家如何在地图中制作可碰撞的区域，如何使用tile属性，如何制作可以拾取的物品以及如何动态修改地图、如何使用“Heads　up　display”来显示分数。

　　在这个教程中，我们将加入敌人，这样的话，你的忍者就可以向它们扔飞镖啦，同时还增加了胜利和失败的游戏逻辑。但是，首先，你得下载一些相关的[资源文件](http://geekanddad.com/downloads/cocos2d-iphone-tutorial/Added_Resources.zip)。

　　这个zip文件里面包含以下内容：

　　　　１.一个敌人精灵

　　　　２.一个忍者飞镖，从Ray的《如何使用cocos2d制作一个简单的iphone游戏》中拿过来的。

　　　　３.两张按钮的图片，在教程的后面有使用。

　　在继续学习之前，不要忘了把这些资源加入到你的工程中。

**增加敌人**

　　到第二部分教程结束的时候，游戏已经很酷了，但是它还不是一个完整的游戏。你的忍者可以轻而易举地四处游荡，想吃就吃。但是，什么时候玩家会胜利或者失败呢。我们不妨想象一下，有２个敌人在追杀你的忍者，那么这个游戏会显得更加有趣。

**敌人出现的位置点**

　　好了，回到Tiled软件（此教程使用java版），然后打开你的Tile地图（TileMap.tmx）。

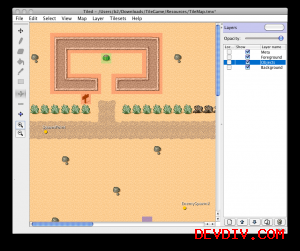
　　往对象层中加入一个对象，在player附近就行，但是不要太近，否则敌人一出现玩家就Game　over了。这个位置将成为敌人出现的位置点，把它命名为“EnemySpawn1”。

　　对象组（对象层中的所有对象组成一个对象组）中的对象被存储在一个NSMutableDictionary中，同时使用对象名字作为key。这意味着每一个位置点必须有一个唯一的名字。尽管我们可以遍历所有的key来比较哪个是以“EnemySpawn”开头，但是这样做效率很低下。相反，我们采用的做法是，使用一个属性来表示，每个给定的对象代表一个敌人出现的位置点。



　　给这个对象一个属性“Enemy”，同时赋一个值１.如果你想在这个教程的基础上扩展，并且增加其它的不同类型的敌人，你可以使用这些敌人的属性值来表示不同类型的敌人。

　　现在，制作６-１０个这种敌人出现位置点对象，相应的它们离player的距离也要有一些不同。为每一个对象定义一个“Enemy”属性，并且赋值为１.保存这张地图并且回到Xcode。



**开始创建敌人**

　　好了，现在我们将把敌人实际显示到地图上来。首先在HelloWorldScene.m中添加如下代码：

//in the HelloWorld class

-(void)addEnemyAtX:(int)x y:(int)y {

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

enemy.position = ccp(x, y);

[self addChild:enemy];

}

// in the init method – after creating the player // iterate through objects, finding all enemy spawn points // create an enemy for each one NSMutableDictionary \* spawnPoint;

for (spawnPoint in [objects objects]) { if ([[spawnPoint valueForKey:@"Enemy"] intValue] == 1){ x = [[spawnPoint valueForKey:@"x"] intValue]; y = [[spawnPoint valueForKey:@"y"] intValue]; [self addEnemyAtX:x y:y]; } }

　　第一个循环遍历对象列表，判断它是否是一个敌人出现的位置点。如果是，则获得它的x和y坐标值，然后调用addEnemyAtX:Y方法把它们加入到合适的地方去。

　　这个addEnemyAtX：Y方法非常直白，它仅仅是在传入的X，Y坐标值处创建一个敌人精灵。

　　如果你编译并运行，你会看到这些敌人出现在你之前在Tiled工具中设定的位置处，很酷吧！



　　但是，这里有一个问题，这些敌人很傻瓜，它们并不会追杀你的忍者。

**使它们移动**

　　因此，现在我们将添加一些代码，使这些敌人会追着我们的player跑。因为，player肯定会移动，我们必须动态地改变敌人的运动方向。为了实现这个目的，我们让敌人每次移动１０个像素，然后在下一次移动之前，先调整它们的方向。在HelloWorldScene.m中加入如下代码：

// callback. starts another iteration of enemy movement.

- (void) enemyMoveFinished:(id)sender {

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

[self animateEnemy: enemy]; }

// a method to move the enemy 10 pixels toward the player - (void) animateEnemy:(CCSprite\*)enemy { // speed of the enemy ccTime actualDuration = 0.3;

// Create the actions id actionMove = [CCMoveBy actionWithDuration:actualDuration position:ccpMult(ccpNormalize(ccpSub(\_player.position,enemy.position)), 10)]; id actionMoveDone = [CCCallFuncN actionWithTarget:self selector:@selector(enemyMoveFinished:)]; [enemy runAction: [CCSequence actions:actionMove, actionMoveDone, nil]]; }

// add this at the end of addEnemyAtX:y: // Use our animation method and // start the enemy moving toward the player [self animateEnemy:enemy];

　　animateEnemy：方法创建两个action。第一个action使之朝敌人移动１０个像素，时间为０.３秒。你可以改变这个时间使之移动得更快或者更慢。第二个action将会调用enemyMoveFinished：方法。我们使用CCSequence　action来把它们组合起来，这样的话，当敌人停止移动的时候就立马可以执行enemyMoveFinished:方法就可以被调用了。在addEnemyAtX:Y:方法里面，我们调用animateEnemy:方法来使敌人朝着玩家（player）移动。（其实这里是个递归的调用，每次移动10个像素，然后又调用enemyMoveFinished:方法）



　　很简洁！但是，但是，如果敌人每次移动的时候面部都对着player那样是不是更逼真呢？只需要在animateEnemy:方法中加入下列语句即可：

//immediately before creating the actions in animateEnemy

//rotate to face the player

CGPoint diff = ccpSub(\_player.position,enemy.position);

float angleRadians = atanf((float)diff.y / (float)diff.x);

float angleDegrees = CC\_RADIANS\_TO\_DEGREES(angleRadians);

float cocosAngle = -1 \* angleDegrees;

if (diff.x < 0) {

cocosAngle += 180;

}

enemy.rotation = cocosAngle

　　这个代码计算每次玩家相对于敌人的角度，然后旋转敌人来使之面朝玩家。



**忍者飞镖**

　　已经很不错了，但是玩家是一个忍者啊！他应该要能够保护他自己！

　　我们将向游戏中添加模式（modes）。模式并不是实现这个功能的最好方式，但是，它比其他的方法要简单，而且这个方法在模拟器下也能运行（因为并不需要多点触摸）。因为这些优点，所以这个教程里面，我们使用这种方法。首先将会建立UI，这样的话玩家可以方便地在“移动模式”和“掷飞镖”模式之间进行切换。我们将增加一个按钮来使用这个功能的转换。（即从移动模式转到掷飞镖模式）。

　　现在，我们将增加一些属性，使两个层之间可以更好的通信。在HelloWorldScene.h里面增加如下代码：

// at the top of the file add a forward declaration for HelloWorld,

// because our two layers need to reference each other

@class HelloWorld;

// inside the HelloWorldHud class declaration HelloWorld \*\_gameLayer;

// After the class declaration @property (nonatomic, assign) HelloWorld \*gameLayer;

// Inside the HelloWorld class declaration int \_mode;

// After the class declaration @property (nonatomic, assign) int mode;

　　同时修改HelloWorldScene.m文件

// At the top of the HelloWorldHud implementation

@synthesize gameLayer = \_gameLayer;

// At the top of the HelloWorld implementation @synthesize mode = \_mode;

// in HelloWorld’s init method \_mode = 0;

// in HelloWorld’s scene method // after layer.hud = hud hud.gameLayer = layer;

　　如果想知道在cocos2d里面如何使用按钮，可以参照我翻译的另外一篇教程[《在cocos2d里面如何制作按钮：简单按钮、单选按钮和开关按钮》。](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%9C%A8cocos2d%E9%87%8C%E9%9D%A2%E5%A6%82%E4%BD%95%E5%88%B6%E4%BD%9C%E6%8C%89%E9%92%AE%EF%BC%9A%E7%AE%80%E5%8D%95%E6%8C%89%E9%92%AE%E3%80%81%E5%8D%95%E9%80%89%E6%8C%89.html)

　　在HelloWorldScene.m中添加下面的代码，这段代码定义了一个按钮。

Add the folowing code, which defines a button, to HelloWorldScene.m:

// in HelloWorldHud’s init method // define the button CCMenuItem \*on; CCMenuItem \*off;

on = [[CCMenuItemImage itemFromNormalImage:@"projectile-button-on.png" selectedImage:@"projectile-button-on.png" target:nil selector:nil] retain]; off = [[CCMenuItemImage itemFromNormalImage:@"projectile-button-off.png" selectedImage:@"projectile-button-off.png" target:nil selector:nil] retain];

CCMenuItemToggle \*toggleItem = [CCMenuItemToggle itemWithTarget:self selector:@selector(projectileButtonTapped:) items:off, on, nil]; CCMenu \*toggleMenu = [CCMenu menuWithItems:toggleItem, nil]; toggleMenu.position = ccp(100, 32); [self addChild:toggleMenu];

// in HelloWorldHud //callback for the button //mode 0 = moving mode //mode 1 = ninja star throwing mode - (void)projectileButtonTapped:(id)sender { if (\_gameLayer.mode == 1) { \_gameLayer.mode = 0; } else { \_gameLayer.mode = 1; } }

　　编译并运行。这时会在左下角出现一个按钮，并且你可以打开或者关闭之。但是这并不会对游戏造成任何影响。我们的下一步就是增加飞镖的发射。



**发射飞镖**

　　接下来，我们将添加一些代码来检查玩家当前处于哪种模式下面，并且在用户点击屏幕的时候影响不同的事件。如果是移动模式则移动玩家，如果是射击模式，则掷飞镖。在ccTouchEnded:withEvent:方法里面增加下面代码：

if (\_mode == 0) {

// old contents of ccTouchEnded:withEvent:

} else {

// code to throw ninja stars will go here

}

　　这样可以使得移动模式下，玩家只能移动。下一步就是要添加代码使忍者能够发射飞镖。在else部分增加，在增加之前，先在HelloWorld.m中添加一些清理代码：

- (void) projectileMoveFinished:(id)sender {

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

[self removeChild:sprite cleanup:YES];

}

好了，看到上面的else部分的注释了吗：

// code to throw ninja stars will go here

在上面的注释后面添加下面的代码：

// Find where the touch is

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

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

touchLocation = [self convertToNodeSpace:touchLocation];

// Create a projectile and put it at the player’s location CCSprite \*projectile = [CCSprite spriteWithFile:@"Projectile.png"]; projectile.position = \_player.position; [self addChild:projectile];

// Determine where we wish to shoot the projectile to int realX;

// Are we shooting to the left or right? CGPoint diff = ccpSub(touchLocation, \_player.position); if (diff.x > 0) { realX = (\_tileMap.mapSize.width \* \_tileMap.tileSize.width) + (projectile.contentSize.width/2); } else { realX = -(\_tileMap.mapSize.width \* \_tileMap.tileSize.width) - (projectile.contentSize.width/2); } float ratio = (float) diff.y / (float) diff.x; int realY = ((realX - projectile.position.x) \* ratio) + projectile.position.y; CGPoint realDest = ccp(realX, realY);

// Determine the length of how far we’re shooting int offRealX = realX - projectile.position.x; int offRealY = realY - projectile.position.y; float length = sqrtf((offRealX\*offRealX) + (offRealY\*offRealY)); float velocity = 480/1; // 480pixels/1sec float realMoveDuration = length/velocity;

// Move projectile to actual endpoint id actionMoveDone = [CCCallFuncN actionWithTarget:self selector:@selector(projectileMoveFinished:)]; [projectile runAction: [CCSequence actionOne: [CCMoveTo actionWithDuration: realMoveDuration position: realDest] two: actionMoveDone]];

　　这段代码会在用户点击屏幕的方向发射飞镖。对于这段代码的完整的细节，可以查看我翻译的另一个文章[《如何使用cocos2d来做一个简单的iphone游戏教程（第一部分）》。](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E6%9D%A5%E5%81%9A%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84iphone%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B1.html)当然，查看原作者的文章后面的注释会更加清楚明白一些。

　　projectileMoveFinished:方法会在飞镖移动到屏幕之外的时候移除。这个方法非常关键。一旦我们开始做碰撞检测的时候，我们将要循环遍历所有的飞镖。如果我们不移除飞出屏幕范围之外的飞镖的话，这个存储飞镖的列表将会越来越大，而且游戏将会越来越慢。编译并运行工程，现在，你的忍者可以向敌人投掷飞镖了。



**碰撞检测**

　　接下来，就是当飞镖击中敌人的时候，要把敌人销毁。在HelloWorldClass类中增加以下变量（在HelloWorldScene.h文件中）：

NSMutableArray \*\_enemies;

NSMutableArray \*\_projectiles;

　　然后初使化\_projectiles数组：

// at the end of the launch projectiles section of ccTouchEnded:withEvent:

[\_projectiles addObject:projectile];

// at the end of projectileMoveFinished: [\_projectiles removeObject:sprite];

　　然后在addEnemyAtX:y方法的结尾添加如下代码：

[\_enemies addObject:enemy];

　　接着，在HelloWorld类中添加如下代码：

- (void)testCollisions:(ccTime)dt {

NSMutableArray \*projectilesToDelete = [[NSMutableArray alloc] init];

// iterate through projectiles for (CCSprite \*projectile in \_projectiles) { CGRect projectileRect = CGRectMake( projectile.position.x - (projectile.contentSize.width/2), projectile.position.y - (projectile.contentSize.height/2), projectile.contentSize.width, projectile.contentSize.height);

NSMutableArray \*targetsToDelete = [[NSMutableArray alloc] init];

// iterate through enemies, see if any intersect with current projectile for (CCSprite \*target in \_enemies) { CGRect targetRect = CGRectMake( target.position.x - (target.contentSize.width/2), target.position.y - (target.contentSize.height/2), target.contentSize.width, target.contentSize.height);

if (CGRectIntersectsRect(projectileRect, targetRect)) { [targetsToDelete addObject:target]; } }

// delete all hit enemies for (CCSprite \*target in targetsToDelete) { [\_enemies removeObject:target]; [self removeChild:target cleanup:YES]; }

if (targetsToDelete.count > 0) { // add the projectile to the list of ones to remove [projectilesToDelete addObject:projectile]; } [targetsToDelete release]; }

// remove all the projectiles that hit. for (CCSprite \*projectile in projectilesToDelete) { [\_projectiles removeObject:projectile]; [self removeChild:projectile cleanup:YES]; } [projectilesToDelete release]; }

　　最后，初始化敌人来飞镖数组，并且调度testCollisions：方法，把这些代码加在HelloWorld类的init方法中。

// you need to put these initializations before you add the enemies,

// because addEnemyAtX:y: uses these arrays.

\_enemies = [[NSMutableArray alloc] init];

\_projectiles = [[NSMutableArray alloc] init];

[self schedule:@selector(testCollisions:)];

　　上面的所有的代码，关于具体是如何工作的，可以在我的博客上查找[《如何使用COCOS2D制作一个简单的iphone游戏》](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E6%9D%A5%E5%81%9A%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84iphone%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B1.html)教程。当然，原作者的文章注释部分的讨论更加清晰，所以我翻译的教程，也希望大家多讨论啊。代码尽量自己用手敲进去，不要为了省事，alt+c,alt+v，这样不好，真的！

　　好了，现在可以用飞镖打敌人，而且打中之后它们会消失。现在让我们添加一些逻辑，使得游戏可以胜利或者失败吧!

**胜利和失败**

**The Game Over Scene**

　　好了，让我们创建一个新的场景，来作为我们的“You Win”或者“You Lose”指示器吧。在Xcode中，选择Classes文件夹，然后点击File\New File，再选择Objective-c类，确保NSObject是基类被选中。点击下一步，然后输入文件名GameOverScene，并且确保“Also create GameOverScene.h”复选中。

　　然后用下面的代码替换掉模板生成代码：

#import ”cocos2d.h”

@interface GameOverLayer : CCColorLayer { CCLabel \*\_label; } @property (nonatomic, retain) CCLabel \*label; @end

@interface GameOverScene : CCScene { GameOverLayer \*\_layer; } @property (nonatomic, retain) GameOverLayer \*layer; @end

　　相应地修改GameOverScene.m文件：

#import ”GameOverScene.h”

#import ”HelloWorldScene.h”

@implementation GameOverScene @synthesize layer = \_layer;

- (id)init {

if ((self = [super init])) { self.layer = [GameOverLayer node]; [self addChild:\_layer]; } return self; }

- (void)dealloc { [\_layer release]; \_layer = nil; [super dealloc]; }

@end

@implementation GameOverLayer @synthesize label = \_label;

-(id) init { if( (self=[super initWithColor:ccc4(255,255,255,255)] )) {

CGSize winSize = [[CCDirector sharedDirector] winSize]; self.label = [CCLabel labelWithString:@"" fontName:@"Arial" fontSize:32]; \_label.color = ccc3(0,0,0); \_label.position = ccp(winSize.width/2, winSize.height/2); [self addChild:\_label];

[self runAction:[CCSequence actions: [CCDelayTime actionWithDuration:3], [CCCallFunc actionWithTarget:self selector:@selector(gameOverDone)], nil]];

} return self; }

- (void)gameOverDone {

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

}

- (void)dealloc { [\_label release]; \_label = nil; [super dealloc]; }

@end

　　GameOverLayer仅仅只是在屏幕中间旋转一个label,然后调度一个transition隔3秒后回到HelloWorld场景中。

**胜利场景**

　　现在，让我们添加一些代码，使得玩家吃完所有的西瓜的时候，游戏会结束。在HelloWorld类的setPlayerPositoin:方法中添加以下代码，（位于HelloWorldScene.m中，就是update代码后面：）

// put the number of melons on your map in place of the ’2′

if (\_numCollected == 2) {

[self win];

}

　　然后，在HelloWorld类中创建win方法：

- (void) win {

GameOverScene \*gameOverScene = [GameOverScene node];

[gameOverScene.layer.label setString:@"You Win!"];

[[CCDirector sharedDirector] replaceScene:gameOverScene];

}

　　不要忘了包含头文件：

#import ”GameOverScene.h”

　　编译并运行，当你吃完所有的西瓜后，就会出现如下画面：



**失败场景**

　　就这个教程而言，我们的玩家只要有一个敌人碰到他，游戏是结束了。在HelloWorld类的testCollision方法中添加以列循环：

for (CCSprite \*target in \_enemies) {

CGRect targetRect = CGRectMake(

target.position.x - (target.contentSize.width/2),

target.position.y - (target.contentSize.height/2),

target.contentSize.width,

target.contentSize.height );

if (CGRectContainsPoint(targetRect, \_player.position)) { [self lose]; } }

　　这个循环遍历所有的敌人，只要有一个敌人精灵的图片所在的矩形和玩家接触到了，那么游戏就失败了。接下，再创建lose方法：

- (void) lose {

GameOverScene \*gameOverScene = [GameOverScene node];

[gameOverScene.layer.label setString:@"You Lose!"];

[[CCDirector sharedDirector] replaceScene:gameOverScene];

}

　　编译并运行，一旦有一个敌人碰到你，你就会看到下面的场景：



**完整源代码**

　　这里有这个教程的[完整源代码](http://geekanddad.com/downloads/cocos2d-iphone-tutorial/TileGame.zip)。谢谢你们有耐心看到这里。

**接下来怎么做？**

建议：

* 增加多个关卡
* 增加不同类型的敌人
* 在Hud层中显示血条和玩家生命
* 制作更多的道具，比如加血的，武器等等
* 一个菜单系统，可以选择关卡，关闭音效，等等

使用更好的用户界面，来使游戏画面更加精美，投掷飞镖更加潇洒。

**在cocos2d里面如何拖拽精灵 by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文出处：<http://www.raywenderlich.com/2343/how-to-drag-and-drop-sprites-with-cocos2d>

PS:非常感谢aom7610给我拟的这份免责申明，以后我会继续努力翻译，把这件事情坚持下去。

程序截图：



我收到许多读者来信说，能不能写一个教程，关于如何在cocos2d里面使用touch事件来拖拽精灵（sprite）。既然你们这么要求，我就满足你们啦！

在这个教程中，你将学到下列内容：

* 使用touch事件拖拽精灵的基本方法
* 如何通过touch事件来滚动视图本身
* 如何方便地计算坐标
* 如何通过识别手势来实现一些更cool的效果

为了使事件变得有趣，你将要移动一些非常可爱的动画图片，它是[我可爱的妻子](http://www.vmwstudios.com/)创作的，背景则是由 [gwebstock](http://www.gwebstock.net/2010/11/06/blue-shotting-stars-vector-background/)创建。

这个教程假设你已经有一些基本的cocos2d的知识，同时已经安装了一份cocos2d的版本。如果你对cocos2d还不熟悉，你可能需要先学习一下译者翻译的[《如何使用cocos2d来制作一个简单的iphone游戏：第一部分》](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E6%9D%A5%E5%81%9A%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84iphone%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B1.html)

好了，不多说，准备好键盘，开始吧！

**Getting Started**

在实现touch事件之前，首先你需要创建一个基本的cocos2d场景来显示背景和这些动物精灵。

打开XCode，点击File\New Project，选择 User Templates\cocos2d X.X.X\cocos2d Application，再点击“Choose…”。把工程命名为“DragDrop”并点击Save。

接下来，继续，下载你需要的[图片](http://www.raywenderlich.com/downloads/DragDropImages.zip)。下载完后，解压，然后把这些图片拖到Resources分组下面。确保“ Copy items into destination group’s folder (if needed)”被选中，然后点击Add。

在你把图片导入到工程之后，在Xcode中展开Classes分组，然后选择HelloWorld.h。在@interface申明处，像下面所示，申明３个实例变量：

CCSprite \* background;

CCSprite \* selSprite;

NSMutableArray \* movableSprites;

你将使用这些变量才追踪你的背景图片、当前选中的精灵以及一个在处理touch事件时需要移动的精灵的数组。

现在，回到HelloWorldScene.m，找到init方法，把它替换成下面的代码：

-(id) init {

if((self = [super init])) {

CGSize winSize = [CCDirector sharedDirector].winSize;

[CCTexture2D setDefaultAlphaPixelFormat:kCCTexture2DPixelFormat\_RGB565]; background = [CCSprite spriteWithFile:@"blue-shooting-stars.png"]; background.anchorPoint = ccp(0,0); [self addChild:background]; [CCTexture2D setDefaultAlphaPixelFormat:kCCTexture2DPixelFormat\_Default];

movableSprites = [[NSMutableArray alloc] init]; NSArray \*images = [NSArray arrayWithObjects:@"bird.png", @"cat.png", @"dog.png", @"turtle.png", nil]; for(int i = 0; i < images.count; ++i) { NSString \*image = [images objectAtIndex:i]; CCSprite \*sprite = [CCSprite spriteWithFile:image]; float offsetFraction = ((float)(i+1))/(images.count+1); sprite.position = ccp(winSize.width\*offsetFraction, winSize.height/2); [self addChild:sprite]; [movableSprites addObject:sprite]; } } return self; }

这里有一些新的知识点需要引入，让我们一步步来学习吧！

**加载背景**

这个方法的第一部分加载了一张本场景的背景图片（blue-shooting-stars.png）。注意，这里把图片的锚点（anchor　point）设置成图片的左下角（０，０）点。

在cocos2d里面，当你设置一个精灵的位置的时候，实际上，你设置的是这个精灵的锚点的位置。默认情况下，图片的锚点就是图片的中点。因此，通过把精灵锚点设置成左下角，当你设置精灵位置的时候，实际上你就是指定了精灵的中心位置在左下角。

这个方法并没有设置背景的位置，因此背景的位置默认情况下是（０，０）。因此，图片的实际位置就是在（０，０）。（因此设置精灵位置是相对于锚点来的，锚点在左下角，因此图片的左下角就位于屏幕的左下角）。因此，这个图片有８００个像素宽，那么超过的部分就在屏幕之外了。

另外需要注意的一点就是，在加载图片之前，转换了一下像素格式。在默认情况下，cocos2d里面加载图片，它们是作为３２位的图片加载进来的。这意味着每个像素占４个字节的内存空间。当你需要非常高质量的显示效果时，非常好！但是，有时候需要折中一下，因为以前的设备内存很有限，如果全部使用３２的像素格式来加载图片的话，会造成内存消耗过多。

当你加载大的图片的时候（比如背景图片），最佳实践是使用１６位的像素格式来加载–也就是牺牲一点质量来减少内存开销。cocos2d里面有[很多不同的像素格式](http://www.cocos2d-iphone.org/archives/61)–这个教程中，我们选择１６位的像素格式，RGB565，因为背景一般不需要透明效果。（少了Alpha通道，RGBA就是有Alpha通道）

**加载图片**

init方法的另外一部分，就是循环遍历一个图片数组，然后创建精灵并且计算精灵放置的坐标。这些精灵会一字排开，显示在屏幕上。同时把这些精灵的引用保存在movableSprites数组里面，这个数组后面会使用到。

最后，我们需要一些清理内存的操作。找到dealloc方法，然后添加下列代码：

[movableSprites release];

movableSprites = nil;

就这么多！编译并运行，你将看到一排非常可爱的小动物，在等待你touch呢！

**基于touch事件选取精灵**

现在，我们将编写一些代码基于用户的touch事件来决定哪一个精灵被选到了。

第一步，就是激活你的HelloWorldLayer层，让它能够接收touch事件。在init方法的最后添加下列代码：

[[CCTouchDispatcher sharedDispatcher] addTargetedDelegate:self priority:0 swallowsTouches:YES];

注意，这是一种新的方式来激活层的touch事件–老的方式就是，设置层的isTouchEnabled属性为Yes，然后实现ccTouchesBegan方法。如果你非常关心，这个新的方法和旧的方法的优缺点的话，可以参考译者翻译的[《如何使用cocos2d制作基于tiled地图的游戏教程》](http://www.cnblogs.com/andyque/archive/2011/04/11/2012852.html)。

接下来，在HelloWorldScene.m的底部添加一些新的方法：

- (void)selectSpriteForTouch:(CGPoint)touchLocation {

CCSprite \* newSprite = nil;

for (CCSprite \*sprite in movableSprites) {

if (CGRectContainsPoint(sprite.boundingBox, touchLocation)) {

newSprite = sprite;

break;

}

}

if (newSprite != selSprite) {

[selSprite stopAllActions];

[selSprite runAction:[CCRotateTo actionWithDuration:0.1 angle:0]];

CCRotateTo \* rotLeft = [CCRotateBy actionWithDuration:0.1 angle:-4.0];

CCRotateTo \* rotCenter = [CCRotateBy actionWithDuration:0.1 angle:0.0];

CCRotateTo \* rotRight = [CCRotateBy actionWithDuration:0.1 angle:4.0];

CCSequence \* rotSeq = [CCSequence actions:rotLeft, rotCenter, rotRight, rotCenter, nil];

[newSprite runAction:[CCRepeatForever actionWithAction:rotSeq]];

selSprite = newSprite;

}

}

- (BOOL)ccTouchBegan:(UITouch \*)touch withEvent:(UIEvent \*)event { CGPoint touchLocation = [self convertTouchToNodeSpace:touch]; [self selectSpriteForTouch:touchLocation]; return TRUE; }

第一个方法（selectSpriteForTouch）是一个帮助方法，这个方法遍历movableSprites数组中的所有精灵，查找第一个精灵位置与touch点位置相交的精灵。

注意，CCNode有一个辅助属性叫做boundingBox，它返回精灵的边界矩形。这比你自己手动计算精灵的边界矩形要好多了。因为，第一，它更简单;第二，它考虑了精灵的位置坐标变换。（比如锚点变了，要执行相应的矩阵变换，具体可以参考源代码，这些就不再细说了。）

如果找到一个匹配的精灵，那么就让这个精灵执行一些动画，这样用户就知道哪个精灵被选中了。如果动画还没执行完，又选中另一个精灵了，那么就中断前一个精灵的动画。这里的动画效果，使用了一系列的CCAction来实现的。

最后，ccTouchBegan方法基于用户的touch事件调用上面的方法。注意，这里把touch坐标点从UIView的坐标系转换成了结点坐标系。为了实现这个目的，通过调用CCNode的一个辅助函数，convertTouchToNodeSpace。这个方法做了以下三件事：

* 计算touch视图（也就是屏幕）的touch点位置（使用locaitonInView方法）
* 转换touch坐标点为OpenGL坐标点（使用convertToGL方法
* 转换OpenGL坐标系为指定结点的坐标系（使用convertToNodeSpace方法）

这是一个非常常用的转换过程，所以提供这个方法可以节约很多时间。

编译并运行代码，并且用手触摸这些动物。当你点中一个精灵的时候，它就会以一种非常可爱的方式旋转，表明它被你选中啦！

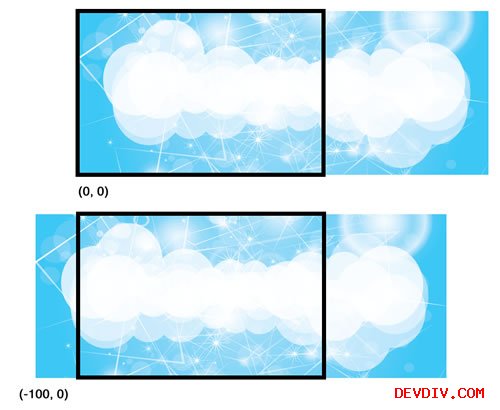


**基于touch事件移动精灵和背景层**

是时候让小动物移动了！基本的思想就是实现ccTouchMoved回调函数，然后计算本次touch点到上一次touch点之间的距离。如果一个动物被选中，将按照计算出来的touch偏移量来移动它。如果动物没有被选中，那就移动整个层，因此用户可以从左至右滚动层。

在编写代码之前，让我们花点时间来讨论一下，如何在cocos2d里面滚动一个层。

首先，看到下面两张图片：



如你所见，你设置背景锚点在左下角（０，０），前景其它部分就在屏幕之外。黑色框框表示当前可见的区域，也就是屏幕范围。

因此，如果你将图片往右边滚动１００个像素，你可以把整个cocos2d的层往左移动１００个像素，如第二张图所示。

同时，你要确保不会移得太多。比如，你不能够把层往右移动，因为那样左边就是空白的了。

现在，你了解了一些背景信息，让我们看看代码怎么写吧！在文件的最后添加下列新的方法：

- (CGPoint)boundLayerPos:(CGPoint)newPos {

CGSize winSize = [CCDirector sharedDirector].winSize;

CGPoint retval = newPos;

retval.x = MIN(retval.x, 0);

retval.x = MAX(retval.x, -background.contentSize.width+winSize.width);

retval.y = self.position.y;

return retval;

}

- (void)panForTranslation:(CGPoint)translation { if (selSprite) { CGPoint newPos = ccpAdd(selSprite.position, translation); selSprite.position = newPos; } else { CGPoint newPos = ccpAdd(self.position, translation); self.position = [self boundLayerPos:newPos]; } }

- (void)ccTouchMoved:(UITouch \*)touch withEvent:(UIEvent \*)event { CGPoint touchLocation = [self convertTouchToNodeSpace:touch];

CGPoint oldTouchLocation = [touch previousLocationInView:touch.view]; oldTouchLocation = [[CCDirector sharedDirector] convertToGL:oldTouchLocation]; oldTouchLocation = [self convertToNodeSpace:oldTouchLocation];

CGPoint translation = ccpSub(touchLocation, oldTouchLocation); [self panForTranslation:translation]; }

第一个方法（boundLayerPos），用来确保你在滚动层的时候不会超出背景图片的边界。你传递一个目标点坐标，然后相应地修改x值，保证不会超出边界。如果你不是很理解的话，可以拿出纸和笔，结合上面给出的图片，自己动手画一画。

接来的方法（panForTranslation）基于传入的目标点位置移动精灵（如果有精灵被选中就移动之，否则移动整个层）。具体的做法就是设置精灵或者层的位置。

最后一个方法（ccTouchMoved）是一个回调函数，它在你拖动屏幕上的手指时调用。像之前一样，把touch坐标转换成局部node坐标。因为没有一个辅助方法可以把前一个层的touch坐标转换成node坐标，因此，我们需要手工地调用那３个方法来执行这个操作。

然后，计算touch偏移量，通过把当前的点坐标减去上一个点坐标，然后调用panForTranslation方法。

编译并运行，现在，你可以通过拖拽来移动精灵和层啦！



**如何在cocos2d里面识别手势**

这里还有另一种方式来实现刚刚的touch处理–通过使用手势识别！

手势识别是新版本的ios　sdk增加的内容（iOS　SDK　３.２引入）。说实话，这个功能太棒了！

首先，你不再不需写一大堆代码来检测“点击（tap）、双击（double　tap）、滑过（swipe）、平移（pan），挤压（pinch）“的区别了。你只需要创建一个手势识别对象，然后把它加到view里面。当有相应的手势发生的时候，会给你一个回调通知。

它非常容易使用，在cocos2d里面使用也非常方便。让我们看看它是如何工作的。

首先，回到init方法，把注册touch的调用注释掉，因为你将使用一种新的方法：

//[[CCTouchDispatcher sharedDispatcher] addTargetedDelegate:self priority:0 swallowsTouches:YES];

然后，回到DragDropAppDelegate.m，找到applicationDidFinishLaunching方法，把最后一行替换掉：

CCScene \*scene = [HelloWorld scene];

HelloWorld \*layer = (HelloWorld \*) [scene.children objectAtIndex:0];

UIPanGestureRecognizer \*gestureRecognizer = [[[UIPanGestureRecognizer alloc] initWithTarget:layer action:@selector(handlePanFrom:)] autorelease];

[viewController.view addGestureRecognizer:gestureRecognizer];

[[CCDirector sharedDirector] runWithScene:scene];

这段代码获得HelloWorld层的引用（它是HelloWorldScene的唯一孩子节点），然后，创建一个UIPanGestureRecognizer对象。注意，为了创建手势对象，你必须初使化它，并且指定回调函数–在本例中就是handlePanFrom方法。

创建完手势识别对象后，你需要把它加入到OpenGL视图中（viewController.view）。

接下来，在HelloWorldScene.m底部加入下面的代码：

- (void)handlePanFrom:(UIPanGestureRecognizer \*)recognizer {

if (recognizer.state == UIGestureRecognizerStateBegan) {

CGPoint touchLocation = [recognizer locationInView:recognizer.view]; touchLocation = [[CCDirector sharedDirector] convertToGL:touchLocation]; touchLocation = [self convertToNodeSpace:touchLocation]; [self selectSpriteForTouch:touchLocation];

} else if (recognizer.state == UIGestureRecognizerStateChanged) {

CGPoint translation = [recognizer translationInView:recognizer.view]; translation = ccp(translation.x, -translation.y); [self panForTranslation:translation]; [recognizer setTranslation:CGPointZero inView:recognizer.view];

} else if (recognizer.state == UIGestureRecognizerStateEnded) {

if (!selSprite) { float scrollDuration = 0.2; CGPoint velocity = [recognizer velocityInView:recognizer.view]; CGPoint newPos = ccpAdd(self.position, ccpMult(velocity, scrollDuration)); newPos = [self boundLayerPos:newPos];

[self stopAllActions]; CCMoveTo \*moveTo = [CCMoveTo actionWithDuration:scrollDuration position:newPos]; [self runAction:[CCEaseOut actionWithAction:moveTo rate:1]]; }

} }

这个回调函数在pan手势开始、变更（用户连续拖拽）和结束的时候被触发。通过判别不同的状态来处理不同的行为。

当手势开始的时候，把touch坐标转换成node坐标，然后调用之前写好的selectSpriteForTouch方法。

当手势变更的时候，需要计算出手势移动偏移量。这里使用手势对象的方法translationInView来获得连续的偏移量。注意，这里需要把y值设置成负值。因为UIKit的坐标和OpenGL坐标系统不一样。（UIKit　左上角是（０，０），而OpenGL左下角的坐标是（０，０）。

之后把recognizer设置为０，因为这个移动是连续的，而我们只需要得到离散的偏移量。

当手势结束的时候，这里有一些新的代码。另一个非常cool的事情就是，你可以通过UIPanGestureRocognizer对象可以得到Pan移动的速度。你可以使用这个速度来滑动层，这个效果就像你使用tableview一样。

因此，这个部分包含一些代码，基于移动的速度来计算移动点。然后运行CCMoveTo　action（使用CCEaseOut效果会更好）。

编译并运行代码，你可以通过手势识别来滑动层啦！



**何去何从?**

这里有本教程的[完整源代码](http://www.raywenderlich.com/downloads/DragDrop.zip)。

到目前为止，你应该知道如何在cocos2d里面使用touch事件来移动精灵了。同时，你也了解了在cocos2d里面如何使用手势识别对象。

现在，你可以尝试扩展本范例，可以使用不同的手势识别对象，比如挤压（pinch）和旋转（rotate）手势识别对象。

如果大家看教程有什么不明白的地方，都可以拿出来，大家一起讨论，共同提高！

**如何使用cocos2d来制作一个打地鼠的游戏：第一部分 by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/2560/how-to-create-a-mole-whacking-game-with-cocos2d-part-1>

教程截图：



在[iOS Programming 101](http://www.raywenderlich.com/2424/iphon-101-for-baltimore-developers)课程中，我的一个学生提议，让我写一个教程，关于如何使用cocos2d来制作一个打地鼠的游戏。

我认为这是一个非常好的主意，原因有三：

* 目前，我的博客上有许多关于cocos2d的教程了，但是，却没有一个教程把制作一个游戏相关的所有细节都放在一起来开发一个游戏。因此，这个教程应运而生了。
* 这是一个非常好的机会来介绍一个新的话题：如何制作一个游戏，让它同时在iphone、ipad和支持高清（retina）显示的iphone上面运行。
* 当然，最重要的原因是–打地鼠本来就很有趣！

这个教程建立在下列教程的基础之上：

* [如何使用cocos2d来制作一个简单的iphone游戏](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E6%9D%A5%E5%81%9A%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84iphone%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B1.html)
* [如何在cocos2d里面使用动画和spritesheet](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E5%9C%A8cocos2d%E9%87%8C%E9%9D%A2%E4%BD%BF%E7%94%A8%E5%8A%A8%E7%94%BB%E5%92%8Cspritesheet.html)
* [在cocos2d里面如何使用Texture Packer和像素格式来优化spritesheet](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%9C%A8cocos2d%E9%87%8C%E9%9D%A2%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8texture-packer%E5%92%8C%E5%83%8F%E7%B4%A0%E6%A0%BC%E5%BC%8F%E6%9D%A5%E4%BC%98%E5%8C%96spritesheet.html)
* [在cocos2d里面如何拖拽精灵](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%9C%A8cocos2d%E9%87%8C%E9%9D%A2%E5%A6%82%E4%BD%95%E6%8B%96%E6%8B%BD%E7%B2%BE%E7%81%B5.html)

如果你还没有阅读上面这些教程，我强烈建议你先读一读。

这是一个两部分的教程。在第一部分中，我们将会创建基本的游戏框架–有一些可爱的地鼠从洞里面钻出来。我们将会花费大量的时候来讨论如何组织图片资源和坐标计算，以便于你的游戏能够同时运行在iPhone３GS、ipad和高清显示的iphone（其实是就是iphone４）上面，同时还要保证游戏性能！

**规划图片资源: 总览**

因为，我们想让这个应用能够同时运行在iphone、retina-display　iphone和ipad上面，所以，我们需要花点时间来仔细考虑一下，如何制作图片，主要是图片的尺寸！

为了理解到底多大的尺寸合适，首先，我们要先讨论下面三个主题：

* Retina Display and UIKit
* Retina Display and Cocos2D
* iPad, iPhone, and Aspect Ratios

**Retina Display and UIKit**

普通的iphone和支持高清显示的iphone之间的差别就是支持高清显示的iphone能够显示２倍的像素。因此（在landscape模式下），普通的iphone显示４８０×３２０个像素，而支持高清的iphone则可以显示９６０×６４０个像素。

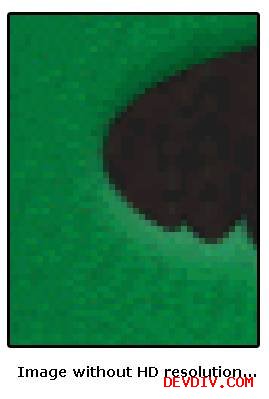


“但是，请等一下”，你可能会问，“如果显示２倍数量的像素，那么不会破坏那些已经写好的假设屏幕大小只有４８０×３２０的应用程序吗？”　没错，特别是当你使用UIKit编程的时候，通常都是硬编码frame的大小。但是，如果你在UIKit里面指定frame大小的时候，实际的单位是点，而不是像素。

在一个普通的iphone上面（不支持retina　display），一个点就等于一个像素。但是，在支持retina的iphone上面，一个点等于２个像素。因此，当你指定（１０，１０），在普通的iphone上面也是（１０，１０），而在支持retina的iphone上面，则是（２０，２０），因此，那会是同样的相对偏移量。（因为，像素加倍了，相对于图片来说，相对偏移量相等）。

当你使用apple的控件或者Core　Graphics，Apple已经写了一些代码来确保在Retina显示的时候没有问题。（因为你写代码指定大小都是point为单位）

唯一的需要注意的就是使用图片的时候。比如你在一个iphone程序中使用了一张２００×２００的图片，如果你什么也不做，那么retina显示会把这张图的尺寸放大２倍–那样会很难看，如下图所示，会模糊。



因此，你所需要做的就是为所有的图片提供两种版本，一种普清的，一种高清的。如果在高清图片名字后面增加一个@2x后缀（比如sample@2x.png），那么不管什么时候，你只要调用［UIImage imageNamed:…]或者一些更简单的api，当程序编译到支持高清设备的时候，就会自动加载后缀为@2x的图片。

因此，制作一个支持Retina显示的UIKit程序是非常容易的，只需要为每一张普清图片同时提供一个高清版本（２倍像素），并且指定为@2x后缀即可。

那么cocos2d里面该如何做呢？

**Retina Display and Cocos2D**

告诉你们一个好消息–最新版本的cocos2d完全支持retina显示，而且使用这个功能只需要下列１-２-３步：

* 调用CCDirector的函数enableRetinaDisplay来开启retina显示支持。如果你使用cocos2d模板，那么只需要把app　delegate里面对这句函数调用的注释去掉即可！
* 往你的程序中添加２倍大小的精灵图片，但是，这里命名不是增加@2x后缀，而是使用-hd（如：sample-hd.png)。当你加载精灵的时候，使用不带-hd后缀的图片名字，当cocos2d被部署到支持retina显示的设备上时，就会自动加载高清的图片。
* 当你在cocos2d里面设置精灵坐标的时候，使用的是点，而不是像素。注意：有些api使用的是像素，这些方法中都包含pixel字样，比如position和positionInPixel。InPixels;

因此，你只需要让美工提供给你高清格式的图片就行了，你可以很容易地使用软件把这些高清图片变得普清图片。比如TexturePacker就可以做这样的事。

你可能会奇怪，为什么需要准备两种不同格式的图片–为什么不每次都加载高清的图片，然后当需要普清图片的时候，用程序来缩放？好吧，因为把纹理（texture）加载到内存里面是非常消耗内存的操作，如果某个程序被安装在不支持高清显示的设备上面（比如IPOD2），那么你就会占用两倍大小的内存，当然是相比于只加载普清的图片的程序。（因为高清图片像素个数多一倍，所以必须消耗的内存要大。正确的做法就是前面说的，提供两套图片。）

不用担心，你没有必要在photoshop里面不停地缩放图片的尺寸。TexturePaker里面有一个非常好的功能，可以很容易地通过高清图片得到对应的普清图片，这种方法在后面的教程中会讲到。

**iPad, iPhone, and Aspect Ratio**

好，现在处理iphone上面的高清和普清显示非常简单了（准备两套图片），但是，iPad呢？该怎么做呢？

呃，要想编写一个游戏，让它同时能够在iphone和ipad上面运行似乎是一件非常烦人的事–因为设备的纵横比（aspect ratio)不一样！

iphone支持480\*320或者960\*640—因此是1.5的纵横比。然后，iPad是768\*1024的–纵横比等于1.33.

这意味着，如果你有一张图片，能够完全覆盖iPad的背景（768\*1024），如果你也想在iPhone里面继续使用它。那恐怕不会匹配那么好。比如，你把它缩小，让宽度等于iphone的宽度（通过乘以0.9375）：你会得到720\*960，因此，还是会有一部分图像不能显示！



这让事情变得有点恼火了，因为，不只是背景图片的问题，因为纵横比的不同，原先设置节点使用的坐标就不能够再使用了。

目前有一些方法可以解决这个问题，下面列举了一些我看过的、听过的和使用过的（如果你们有更好的方法，可以留言）：

* 考虑一个“可玩区域”，这个区域的大小等于640\*960，同时把它放置在iPad的屏幕中间。这会使得游戏区域外面有一些空白–但是，你还可以放置一张位置不动、颜色相同的背景图片，那么玩家在玩游戏的时候就很难发现了。这样会使你很容易地在不同的设备之间使用相同的坐标值，并且还可以重用图片资源。这个方法也是我们这个教程里所使用的方法。
* 你可以使你的iPad的程序的纵横比与iPhone的纵横比一样，在iPad屏幕的边界留出42个像素，把“主要内容”放置在685\*1024的范围中。如果，你可以把你的应用中的内容放置在684\*1024的“盒子”中，那么你就可以为每一个设备使用缩放图片了。
* 你可以为iPhone、iPad和Retina display的iPhone提供不同的图片和坐标值。这样的灵活性是最大的，但是，会产生更大的二进制程序文件，并且不同的设备上面的对象位置都需要重新计算。

另一个原因，cocos2d目前对iPad的支持并不是很友好。只是可以自动加载后缀为“-hd”的图片，转换坐标的工作还是得由你来做。

**规划图片资源：结论**

好，基于上面的讨论，下面是这篇教程的图片资源规划：

* 图片被设计成一个960\*640的“可玩区域”大小，在高清的iphone上面就是全屏的，在iPad上面就会居中显示。
* 高清图片可以通过TexturePacker转换成普清的，并且生成相应的spritesheet。
* 高清的图片将以“-hd”作为后缀，而普清的图片则没有。cocos2d会根据当前是否支持高清显示而加载相应的图片。
* 背景是一个特例，因为它总是需要全屏。背景将会做成1024\*768的大小（因为iPad屏幕的大小是1024\*768）。当这张图片用作iPhone上面的时候，会有一部分图片会超过屏幕边界，但是，在这个教程中，并没有关系。（后面将会看到，背景颜色很单一)
* iPad版本版本将会包含一些代码，用来加载“-HD”后缀的图片，转换坐标并设置到“可玩区域”内，使用合适的字体大小等。

首先，下载这篇教程的[图片资源](http://d1xzuxjlafny7l.cloudfront.net/downloads/WhackAMoleArt.zip)，解压，然后看看里面有什么：

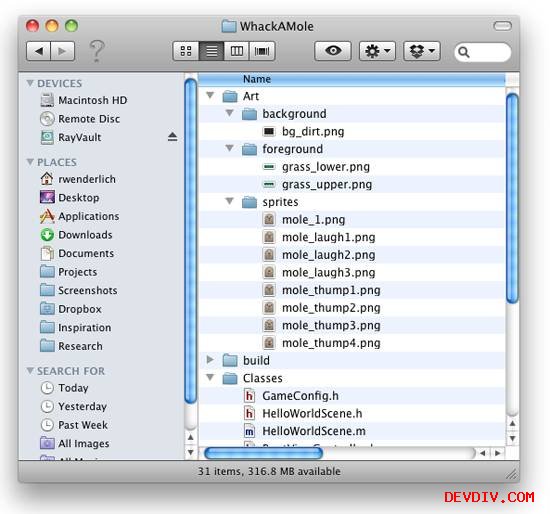
* 在“foreground”文件夹中，前景大小为1024\*768（也就是iPad的大小），但是，它实际上被划分为两个部分：上半部和下半部。因为我们将在这两个部分当中放置三个“地鼠洞“，这样就会看起来，地鼠真的是从地底下钻出来的。
* 在”background“文件夹中，iPad的纵横比是1.33，所以一半是512\*384.这是因为，背景实际上是看不到的，只有通过3个地鼠洞才可以看到背景。这么小的范围，完全没有必要加载一张1024\*1024的图片。相反，我们加载一张小图片，然后放大。
* 在”sprites”文件夹中，所有的精灵都被设置成可以放到960\*640的“可玩区域”内。注意，还有一个地鼠图片，二个地鼠动画（一个笑的动画，一个被打中时的动画）。

好了，有足够的背景信息了–是时候开始动手了！、

**Getting Started**

打开XCode，选择“File\New Project…”,然后选择“User Templates\cocos2d\cocos2d Application”,再点击”Choose…“。把工程命名为WhackAMole，单击Save。

接下来，把你刚刚下载下来的图片资源文件夹全部拷到工程目录下面。它的位置应该是Classes文件夹的兄弟，如下图所示：



接下来，确保你已经安装了TexturePacker。如果你还不知道怎么使用它，可以查看译者翻译的[另一篇教程](http://www.cnblogs.com/andyque/archive/2011/03/18/1988097.html)。

你可以打开TexturePacker工具，并为这个项目创建所有的sprite sheet。你也可以使用TexturePacker的命令行工具，它可以和XCode集成，并且处理起来更加高效！

右击Resources，选择”Add\New File…“，选择Mac OS X\Other\Shell Script，再点Next。取名为PackTextures.sh，然后点Finish。

然后把PackTextures.sh里面的内容替换成下面的代码：

#!/bin/sh

TP=”/usr/local/bin/TexturePacker”

if [ "${ACTION}" = "clean" ]

then

echo ”cleaning…”

rm resources/background\*

rm resources/foreground\*

rm resources/sprites\*

else

echo ”building…”

${TP} –smart-update \

–format cocos2d \

–data resources/background-hd.plist \

–sheet resources/background-hd.pvr.ccz \

–dither-fs \

–opt RGB565 \

Art/background/\*.jpg

${TP} –smart-update \

–format cocos2d \

–data resources/background.plist \

–sheet resources/background.pvr.ccz \

–dither-fs \

–scale 0.5 \

–opt RGB565 \

Art/background/\*.jpg

${TP} –smart-update \

–format cocos2d \

–data resources/foreground-hd.plist \

–sheet resources/foreground-hd.pvr.ccz \

–dither-fs-alpha \

–opt RGBA4444 \

Art/foreground/\*.jpg

${TP} –smart-update \

–format cocos2d \

–data resources/foreground.plist \

–sheet resources/foreground.pvr.ccz \

–dither-fs-alpha \

–scale 0.5 \

–opt RGBA4444 \

Art/foreground/\*.jpg

${TP} –smart-update \

–format cocos2d \

–data resources/sprites-hd.plist \

–sheet resources/sprites-hd.pvr.ccz \

–dither-fs-alpha \

–opt RGBA4444 \

Art/sprites/\*.jpg

${TP} –smart-update \

–format cocos2d \

–data resources/sprites.plist \

–sheet resources/sprites.pvr.ccz \

–dither-fs-alpha \

–scale 0.5 \

–opt RGBA4444 \

Art/sprites/\*.jpg

fi

exit 0

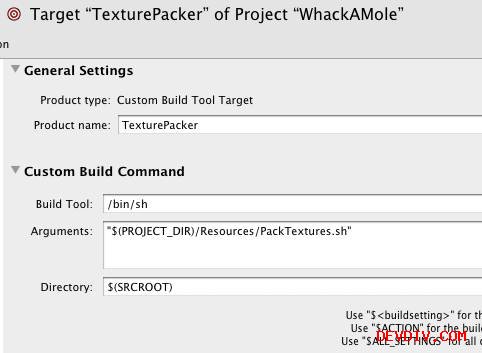
这个脚本运行TexturePacker来为背景图片、前景图片和精灵图片创建相应的精灵表单（sprite sheet）–同时包含高清的和普清的。

注意，每一个图片都被保存为pvr.cca格式，因为，这是一种磁盘占用空间小，并且内存消耗也少的图片格式。当然，像素格式和抖动选项也被设置了，这是一个折中，综合考虑内存消耗和图片显示质量。

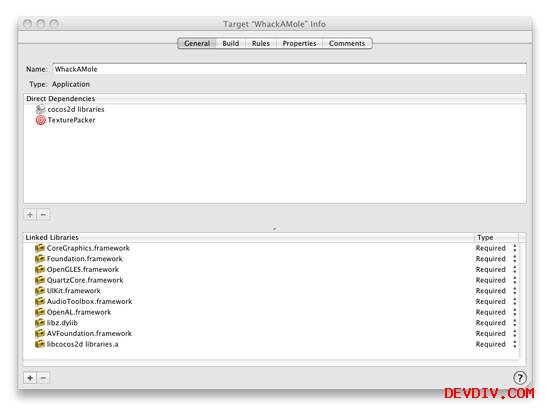
如果你对TexturePacker的这些选项的功能不是很清楚，你可以打开终端（Terminal)，然后输入TexturePacker -help，那么就会有完整的选项功能列表。

接下来，你需要做一些设置，在你的工程每次编译的时候，可以运行这个shell脚本。右键点击Targets，选择“Add\New Target…”,再选择”External Target”（注意不要选择Shell Script Target!)，然后点击Next。把这个Target命名为TexturePacker并点击Finish。

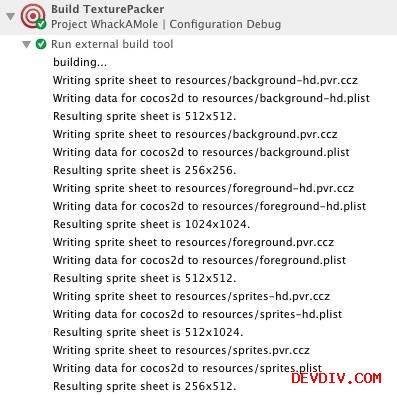
然后右键点击刚刚建立的这个Target，修改它的设置，如下图所示：



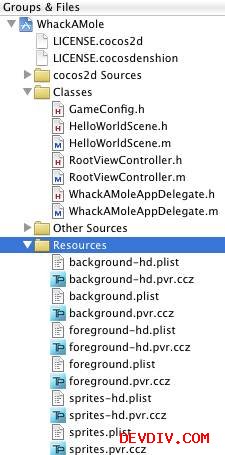
最后一步，就是为你的程序设置target依赖。双击项目Target，然后找到General标签，点下面的+号，再从弹出的列表中选择TexturePacker Target，并点击Add Target。



编译你的工程，如果输出结果如下图所示，那么就证明你的配置OK。（注意，上面的这些全是针对XCode3所设置的，XCode4大同小异，只是界面不一样了，原理不变）



接下来，把刚刚生成的精灵表单和相应的属性列表增加到项目中。右击Resources，选择“Add\Existing Files…”，然后选择background,foreground和sprites文件（总共有12个文件），再点Add.如果做完后，那么看起来会是下图这个样子:



如果你愿意，你可以双击任何一个.pvr.ccz文件，打开看看，里面到底有什么。你就会发现TexturePacker有多么方便了！

**设置背景**

接下来，HelloWorldScene.m文件并找到init方法。删掉自动生成的那4行创建一个label的代码。同时，用下面的代码替换掉：

// Determine names of sprite sheets and plists to load

NSString \*bgSheet = @”background.pvr.ccz”;

NSString \*bgPlist = @”background.plist”;

NSString \*fgSheet = @”foreground.pvr.ccz”;

NSString \*fgPlist = @”foreground.plist”;

NSString \*sSheet = @”sprites.pvr.ccz”;

NSString \*sPlist = @”sprites.plist”;

if (UI\_USER\_INTERFACE\_IDIOM() == UIUserInterfaceIdiomPad) {

bgSheet = @”background-hd.pvr.ccz”;

bgPlist = @”background-hd.plist”;

fgSheet = @”foreground-hd.pvr.ccz”;

fgPlist = @”foreground-hd.plist”;

sSheet = @”sprites-hd.pvr.ccz”;

sPlist = @”sprites-hd.plist”;

}// Load background and foreground

[[CCSpriteFrameCache sharedSpriteFrameCache] addSpriteFramesWithFile:bgPlist];

[[CCSpriteFrameCache sharedSpriteFrameCache] addSpriteFramesWithFile:fgPlist];

// Add background CGSize winSize = [CCDirector sharedDirector].winSize; CCSprite \*dirt = [CCSprite spriteWithSpriteFrameName:@"bg\_dirt.jpg"]; dirt.scale = 2.0; dirt.position = ccp(winSize.width/2, winSize.height/2); [self addChild:dirt z:-2];

// Add foreground CCSprite \*lower = [CCSprite spriteWithSpriteFrameName:@"grass\_lower.jpg"]; lower.anchorPoint = ccp(0.5, 1); lower.position = ccp(winSize.width/2, winSize.height/2); [self addChild:lower z:1];

CCSprite \*upper = [CCSprite spriteWithSpriteFrameName:@"grass\_upper.jpg"]; upper.anchorPoint = ccp(0.5, 0); upper.position = ccp(winSize.width/2, winSize.height/2); [self addChild:upper z:-1];

// Add more here later…

好，让我们一个一个来讲解，因为，这里面有许多新的东西。

1.决定加载的精灵表单和plist文件的名字。这个部分，把TexturePacker生成的精灵表单和plist文件，按照某种方式加载进来了。注意，在iPhone上面，cocos2d会自动地加载“-HD”和不带“HD”的版本，具体取决于是否激活Retina 显示。然后，在iPad上面，它并不会加载”-hd”版本，所以你必须手动告诉它加载，因此，上面我们判断，如果是iPad，那么就加载带”-hd”版本的资源。

2.加载背景和前景。接下来就是把背景、前景还有精灵都加载到精灵帧缓冲（sprite frame cache）中，以便后面可以使用。注意，目前精灵并没有加载到CCSpriteBatchNode中。

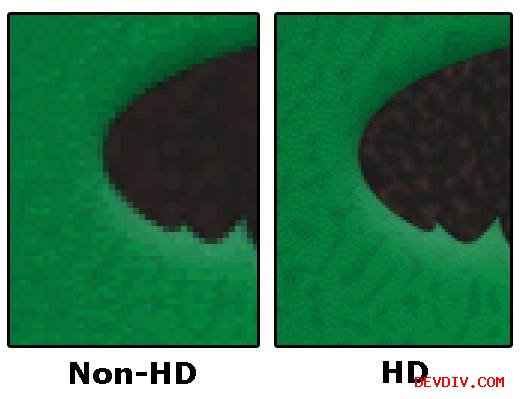
3.添加背景。背景图片作为当前层的一个孩子被加进去。（Z=-2的话，就可以使任何对象都在它上面，除非其它对象的Z小于-2）。接下来，把这张图片放大两倍，因为，之前，我们为了节约内存，只使用了一半大小的图片。之后，再把图片放置到屏幕中间。

4.前加前景。前景是当作两个部分加上去的。这里用了一种非常简单的设置精灵图片坐标点的方式，通过设置精灵的锚点为图片的中点，或者底部，或者顶部。通过这种方式，假如你要把一张图片的左下角设置为屏幕的左下角，你只需要先把图片的锚点设置为（0，0），然后设置position为0，0就好了。非常方便！而且，这个坐标值，在所有的设备上都通用。注意，背景部分超出屏幕之外了，但是，这并没有关系，因为用户看不到。还要注意的是，图片增加的时候，使用了不同的Z值，最后面的图片使用最小的Z值。

编译并运行，你会看到如下的输出。你可以通过模拟器测试，也可以通过真机测试，看看效果到底如何^^。



如果你尝试在支持高清的设备上运行，你会注意到，图像有模糊，似乎还是使用普清的资源。



这是因为，我们还有一步没有做完:还需要调用CCDirector的enableRetinaDisplay方法能激活高清显示。

为了实现这个目的，打开WhackAMoleAppDelegate.m，然后在applicationDidFinishLaunching方法里面，删除那三句注释：

if( ! [director enableRetinaDisplay:YES] )

CCLOG(@”Retina Display Not supported”);

编译并运行，现在，再尝试使用支持高清的设备吧，这时就会加载高清图片资源了。

**放置地鼠**

对于这个游戏来说，你将会在场景中添加3只地鼠–每个洞一个地鼠。地鼠会躲在草的下面，但是，会突然从某个洞里冒出来，这样你就可以打它了。

首先，让我们为每个洞的下面添加一只地鼠。我们会先让它们在所有的图片上面，这样，可以确保位置对了，然后通过修改Z值，就能够把它们隐藏到草地后面去了。

打开HelloWorldScene.h，然后增加一个数组来保存当前关卡的地鼠，如下所示：

// Inside @interface HelloWorld

NSMutableArray \*moles;

通过保存地鼠的数组，我们可以很容易地遍历每一只地鼠。

接下来，在init方法中添加一些代码来放置地鼠（添加位置位于 “Add more here later…”注释后面），如下所示：

// Load sprites

CCSpriteBatchNode \*spriteNode = [CCSpriteBatchNode batchNodeWithFile:sSheet];

[self addChild:spriteNode z:999];

[[CCSpriteFrameCache sharedSpriteFrameCache] addSpriteFramesWithFile:sPlist];moles = [[NSMutableArray alloc] init];

CCSprite \*mole1 = [CCSprite spriteWithSpriteFrameName:@"mole\_1.jpg"]; mole1.position = [self convertPoint:ccp(85, 85)]; [spriteNode addChild:mole1]; [moles addObject:mole1];

CCSprite \*mole2 = [CCSprite spriteWithSpriteFrameName:@"mole\_1.jpg"]; mole2.position = [self convertPoint:ccp(240, 85)]; [spriteNode addChild:mole2]; [moles addObject:mole2];

CCSprite \*mole3 = [CCSprite spriteWithSpriteFrameName:@"mole\_1.jpg"]; mole3.position = [self convertPoint:ccp(395, 85)]; [spriteNode addChild:mole3]; [moles addObject:mole3];

首先，为精灵创建一个CCSpriteBatchNode，这样就可以更加高效地渲染这三只地鼠了。（每次只需要一个OpenGL call),并把batchNode当作当前层的一个孩子加进去。注意，Z值被临时设置成999，因此，地鼠就会显示在所有的图片之前，那样我们就可以判断地鼠位置是否正确。

然后从精灵帧缓冲中加载所有的精灵。接着，为每一个地鼠创建一个精灵，把它们放置在场景中，同时添加到数组里面。注意，每个地鼠的坐标是在一个480\*320的“可玩区域”内（iphone屏幕的大小）。对于iPad来说，这些点需要做一些转换，因此，我们定义了一个辅助函数convertPoint。接下来，在Init方法的上面加入下列代码：

- (CGPoint)convertPoint:(CGPoint)point {

if (UI\_USER\_INTERFACE\_IDIOM() == UIUserInterfaceIdiomPad) {

return ccp(32 + point.x\*2, 64 + point.y\*2);

} else {

return point;

}

}

这个方法把“可玩区域”内的一个点转换成合适的iPad屏幕上面的点。记住：

* 我们在iPad上面使用的是高清的图像，所有的像素点都要加倍。
* 我们把960\*640的区域居中放在1024\*768的iPad屏幕上，因此，左右两边要留下32个像素的margin，上下两边要留下64个像素的margin。

因此， 这个方法只是做了一些简单的数学计算，使得坐标在iPad上面不会出问题。

还有一件事—在忘记之前，我们需要做一些内存清理操作，找到dealloc方法，添加下列代码：

[moles release];

moles = nil;

编译并运行，你将看到3只地鼠放置在正确的位置上了！你因此同时测试一下iPhone,Retina iPhone和iPad，确保每个设备上看起来都没问题。



**弹出地鼠**

现在，我们确定地鼠的位置没有问题了，接下来，让我们添加一些代码，让他们能够从地下钻出来吧！

第一件事，把Z值，从999改成0，这样的话，地鼠就会在地底下了。

改完之后，在init方法的后面添加下面的代码：

[self schedule:@selector(tryPopMoles:) interval:0.5];

如果之前你没有见过这样的用法，你可以调用其它的方法，比如schedelUpdate。但是，这个例子中，我们只想让地鼠每隔0.5秒钻出一个。

接下来，增加tryPopMoles方法：

- (void)tryPopMoles:(ccTime)dt {

for (CCSprite \*mole in moles) {

if (arc4random() % 3 == 0) {

if (mole.numberOfRunningActions == 0) {

[self popMole:mole];

}

}

}

}

这个方法会每隔0.5秒调用一次，每一次它都会循环判断每一个地鼠，让它有1/3的机会从洞里面钻出来。但是，如果它已经钻出来，也就是运行的action的个数不等于0的话，那么也不会再钻了。

最后，实现popMole：方法：

- (void) popMole:(CCSprite \*)mole {

CCMoveBy \*moveUp = [CCMoveBy actionWithDuration:0.2 position:ccp(0, mole.contentSize.height)]; // 1

CCEaseInOut \*easeMoveUp = [CCEaseInOut actionWithAction:moveUp rate:3.0]; // 2

CCAction \*easeMoveDown = [easeMoveUp reverse]; // 3

CCDelayTime \*delay = [CCDelayTime actionWithDuration:0.5]; // 4[mole runAction:[CCSequence actions:easeMoveUp, delay, easeMoveDown, nil]]; // 5

}

这个代码，使用cocos2d的action来制作弹出洞的效果，先弹出来，然后暂停半秒，再钻回去。让我们一行一行地解释：

1. 创建一个action，让地鼠沿着Y轴钻洞来。因为，之前我们放置地鼠的位置是正确的，所以看起来，就像是从洞下面钻出来一样。
2. 为了使得移动更加真实，上面的action用一个CCEaseInOut action包装起来了。这会得钻洞的动作在开始和结束的时候速度比较慢，看起来，就好像地鼠在加速和减速一样。
3. 创建一个action，使地鼠能够钻回来。这里通过调用action的reverse方法，能得到相反的action.
4. 创建一个action，在地鼠钻出来以后，能够暂停1秒。
5. 现在，action可以运行了。通过用CCSequence把这3个action包装起来，那样它们就可以按顺序执行了。

就这么多！编译并运行代码，你将会看到3只地鼠交替地钻出洞来，cool吧？



**何去何从?**

本教程[完整源代码](http://d1xzuxjlafny7l.cloudfront.net/downloads/WhackAMole1.zip)。

期待这个教程的后续吧，在那个教程里，我们会添加很酷的动画，同时会添加一些游戏逻辑，使得你可以打地鼠并且得分。当然，还有非常好听的音乐。

**如何使用cocos2d制作一个打地鼠的游戏：（第二部分。完） by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/2593/how-to-create-a-mole-whacking-game-with-cocos2d-part-2>

教程截图：



这篇文章是《如何使用cocos2d来制作一个打地鼠的游戏》的第二部分。打地鼠系列教程，里面用到的很多概念和方法是从这个博客的其它教程中拿来的，但是，同时，本系列教程还引入了一些新的概念。

在第一部分教程中，我们创建了一个游戏的基本框架–让可爱地地鼠从洞里面钻出来。我们花费了大量时间来讨论如何规划图片资源及其坐标，以便可以开发出一个游戏，让它同时能够在iPhone、iPad和Retina display的设备上运行–并且要保证尽可能地高效率！

在这篇教程中，我们将会增加一些很酷的动画效果，比如地鼠大笑和被打中时的动画。同时，会增加一些游戏逻辑，以便你能够打击地鼠并且获得相应的分数，当然，还会添加一些非常好听的音乐和音效。

如果你还没有上一个教程的工程，可以从[这里](http://d1xzuxjlafny7l.cloudfront.net/downloads/WhackAMole1.zip)下载一份工程拷贝。

**定义动画：实用性**

为了使游戏变得更有趣，我们将给地鼠增加两个动画。首先，当它从洞里钻出来的时候，它会笑一下（那笑声你绝对会忍不住想打它！）。然后，如果你打中它了，那么你会看到地鼠被打中时的面部表情。

但是，在我们开始之前，先讨论一下代码中如何组织动画。

回想我们之前的教程[《如何在cocos2d里面使用spritesheet和动画》](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E5%9C%A8cocos2d%E9%87%8C%E9%9D%A2%E4%BD%BF%E7%94%A8%E5%8A%A8%E7%94%BB%E5%92%8Cspritesheet.html)，其中，在创建动画过程中，有一个步骤是，创建一系列的精灵帧（sprite frames）。因此，对于你的动画效果中的每一张不同的图片，你必须为之增加精灵帧，如下所示：

[animFrames addObject:

[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@”myImage.jpg”]];

我们的地鼠笑的动画将会是下面的一些图片序列：  mole\_laugh1.jpg, mole\_laugh2.jpg mole\_laugh3.jpg, mole\_laugh2.jpg, mole\_laugh3.jpg, mole\_laugh1.jpg.

因此，我们可以硬编码来建立动画，如下所示：

[animFrames addObject:

[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@”mole\_laugh1.jpg”]];

[animFrames addObject:

[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@”mole\_laugh2.jpg”]];

[animFrames addObject:

[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@”mole\_laugh3.jpg”]];

[animFrames addObject:

[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@”mole\_laugh2.jpg”]];

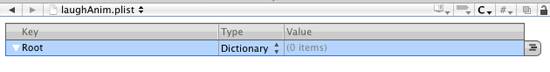
// And so on…

但是，那会使我们的代码变得非常难看。为了使代码变得更简洁，我们不是直接在代码里面定义这些图片，取而代之的是，我们把它们的名字都存到一个plist文件中。

**Property List文件**

如果你之前没有使用过plist文件，你需要知道，其实就是一种xml格式的文件。只不过是，它的后缀不一样，同时，它能够被Xcode直接识别，并且可以方便地存储数组，字典、字符串和数字等等。创建这种类型的文件非常方便，当然使用也一样很方便。

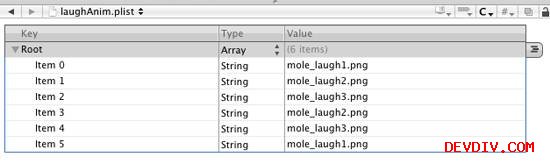
让我们看一下，它在Xcode里面是什么样子。右键点击Resources，选择“Add\New File…”，再选择 “Mac OS X\Resource\Property List”，再点击“Next”。把这个文件命名为“laughAnim.plist”，然后点Finish。这时，属性列表文件laughAnim.plist文件的结构应该如下图所示：



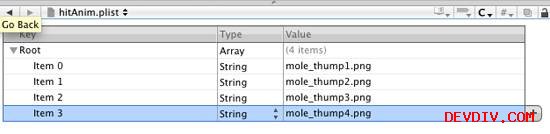
每一个属性列表文件（plist文件）有一个根元素。这个根元素要么是一个数组或者一个字典。这里的plist文件将会包含一个组成地鼠笑的动画的一系列图片名字的数组。因此，点击根元素的第二列，（当前默认是Dictionary），我们把它更改成Array。

接下来，点击最右边的小按钮（有三行的那个按钮）–这样会往数组里面增加一个新的实体。默认情况下，这个新添加的实体的类型是String–那也正是我们想要的数据类型。把Item0的名称改成“mole\_laugh1.jpg”。

然后再点＋号来添加更多的行，重复这个过程，最终的结果如下图所示：



接下来，重复上述的过程来创建地鼠被打中时的动画属性列表文件。按上面所说的，创建一个hitAnim.plist文件，然后把它建立成下图所示的结构：



现在，是时候添加一些代码来加载这些动画了。打开HelloWorldScene.h文件，然后为每一个动画定义一个成员变量，如下所示：

// Inside @interface HelloWorld

CCAnimation \*laughAnim;

CCAnimation \*hitAnim;

这样做的目的主要是重用，因为可以在init函数里面初使化好这些动画效果，那么在其它的地方就直接可以使用这些动画效果了。（这里需要记住的一点是，游戏里面的任何对象都要事先分配好，在玩家玩游戏的过程中，只需要按照某种规则把它们拿出来即可）。

接下来，基于先前创建的plist文件来创建CCAnimation，如下所示：

- (CCAnimation \*)animationFromPlist:(NSString \*)animPlist delay:(float)delay {

NSString \*plistPath = [[NSBundle mainBundle] pathForResource:animPlist ofType:@”plist”]; // 1 NSArray \*animImages = [NSArray arrayWithContentsOfFile:plistPath]; // 2 NSMutableArray \*animFrames = [NSMutableArray array]; // 3 for(NSString \*animImage in animImages) { // 4 [animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:animImage]]; // 5 } return [CCAnimation animationWithFrames:animFrames delay:delay]; // 6

}

理解这个方法非常重要，所以让我们一行一行代码来看。

* 属性列表文件包含在工程当中，因此，它也在应用程序的“main bundle”中。这个方法会返回main bundle中的文件的完整路径，也就是我们需要读取的plist文件的完整路径。
* 为了读取一个plist文件，我们调用NSArray的arrayWithContentsOfFile方法，然后把plist文件的完整路径传递给它。这样就会把plist文件中的内容初使化成一个数组。（本例中，这个数组就是一系列图片名字的数组）。注意，这个方法可行，是因为我们把根元素设置成了NSArray。如果我们把它设置成NSDictionary的话，那么我们就要用一个NSDictionary去初使化它。具体的方法就是调用 [NSDictionary dictionaryWithContentsOfFile...] 。
* 创建一个空的数组来存储这些动画帧。
* 从plist文件中循环遍历每一张图片并把它存到一个数组中去。
* 为每一张图片创建一个精灵帧，同时把它加到 animFrames数组里面去。
* 基于一个精灵帧数组，返回一个CCAnimation对象。

接下来，在init方法的末尾为每一个动画调用这个辅助函数来创建相应的动画：

laughAnim = [self animationFromPlist:@"laughAnim" delay:0.1];

hitAnim = [self animationFromPlist:@"hitAnim" delay:0.02];

[[CCAnimationCache sharedAnimationCache] addAnimation:laughAnim name:@”laughAnim”];

[[CCAnimationCache sharedAnimationCache] addAnimation:hitAnim name:@”hitAnim”];

注意，在存储动画对象的引用之后，我们把它们加入到了动画缓存中（animation cache）。这个非常重要，因为我们可以在其他地方很容易地使用引用。（对于laughAnim和hitAnim，不用retain就可以使用了。因为，加入到动画缓存中的时候，CCAnimationCache已经帮你ratain了）。这样做还有一个好处就是，你可以通过CCAnimationCache来获得你想要的动画对象引用，只需要提供动画的名字即可，因为它内部实现是采用的字典。）

最后一步–让我们来使用动画（先只使用笑的动画）。修改popMole方法，如下所示：

- (void) popMole:(CCSprite \*)mole {

CCMoveBy \*moveUp = [CCMoveBy actionWithDuration:0.2 position:ccp(0, mole.contentSize.height)];

CCEaseInOut \*easeMoveUp = [CCEaseInOut actionWithAction:moveUp rate:3.0];

CCAction \*easeMoveDown = [easeMoveUp reverse];

CCAnimate \*laugh = [CCAnimate actionWithAnimation:laughAnim restoreOriginalFrame:YES];

[mole runAction:[CCSequence actions:easeMoveUp, laugh, easeMoveDown, nil]]; }

这里唯一的差别就是，在钻出来和钻回去的action中间，我们不是延迟几秒，取而代之的是播放地鼠笑的动画。CCAnimate action使用之前已经创建好的laughAnim，同时设置restoreOriginalFrame为yes。这样的话，当动画结束的时候，它会回到播放动画之前的面貌。

编译并运行代码，现在，当地鼠从洞里钻出来的时候，它会朝着你大笑！是不是想打它？有木有！



是时候让这些地鼠的笑容消失了，让我们开始添加打击逻辑吧！

**增加游戏逻辑**

现在我们将往游戏中添加一些玩法逻辑。主要就是记录有多少个地鼠钻出来过，还有就是通过打地鼠，你能得到多少分。你会尝试尽可能多地获得分数。

因此，我们将保存分数，并且显示给用户看。当地鼠钻回去的时候，我们也要告诉用户。

所以，再打开HelloWorldScene.h文件，添加下面一些实例变量到HelloWord层中：

CCLabelTTF \*label;

int score;

int totalSpawns;

BOOL gameOver;

这里保存了一个分数label，当前的分数值，总共钻出来的地鼠数目，以及游戏是否结束。

接下来，在你的init方法的结尾添加下列初始化代码：

self.isTouchEnabled = YES;

float margin = 10; label = [CCLabelTTF labelWithString:@"Score: 0" fontName:@"Verdana" fontSize:[self convertFontSize:14.0]]; label.anchorPoint = ccp(1, 0); label.position = ccp(winSize.width - margin, margin); [self addChild:label z:10];

首先，设置层能够接收到touch事件，因为你想检查用户击打屏幕的消息。然后创建一个label来显示分数。注意，这里把label的锚点设置成右上角，那样可以非常方便地把它放置在屏幕的右上方。

你也要注意到，我们并不是直接传递字体大小，而是通过一个辅助函数来决定字体的大小。这是因为，在iPad上面，字体应该大一些，因为它的屏幕大一些。所以要实现一个convertFontSize方法，如下所示：

- (float)convertFontSize:(float)fontSize {

if (UI\_USER\_INTERFACE\_IDIOM() == UIUserInterfaceIdiomPad) {

return fontSize \* 2;

} else {

return fontSize;

}

}

这个非常简单–在iPad上运行的时候，字体大小加倍，否则，就不变。

接下一，我们将添加touch检测代码，来检测用户是否击中一个地鼠。但是，在这之前，我们需要添加一个标记，标记地鼠是否可以击打。因为地鼠应该只有在它朝着你笑的时候才能够被击打，而在它笑完钻回去的时候，你是不能够击打它的。

我们可以创建CCSprite的一个子类来做这个事，但是，因为我们只需要存储一点点信息，所以，我们只需要使用CCSprite的userData属性即可。因此，添加两个辅助方法，并且修改popMole方法，如下所示：

- (void)setTappable:(id)sender {

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

[mole setUserData:TRUE];

}

- (void)unsetTappable:(id)sender { CCSprite \*mole = (CCSprite \*)sender; [mole setUserData:FALSE]; }

- (void) popMole:(CCSprite \*)mole {

if (totalSpawns > 50) return; totalSpawns++;

[mole setDisplayFrame:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@”mole\_1.jpg”]];

// Pop mole CCMoveBy \*moveUp = [CCMoveBy actionWithDuration:0.2 position:ccp(0, mole.contentSize.height)]; CCCallFunc \*setTappable = [CCCallFuncN actionWithTarget:self selector:@selector(setTappable:)]; CCEaseInOut \*easeMoveUp = [CCEaseInOut actionWithAction:moveUp rate:3.0]; CCAnimate \*laugh = [CCAnimate actionWithAnimation:laughAnim restoreOriginalFrame:YES]; CCCallFunc \*unsetTappable = [CCCallFuncN actionWithTarget:self selector:@selector(unsetTappable:)]; CCAction \*easeMoveDown = [easeMoveUp reverse];

[mole runAction:[CCSequence actions:easeMoveUp, setTappable, laugh, unsetTappable, easeMoveDown, nil]];

}

popMole方法做了如下一些变动：

* 在地鼠大笑之前，它通过运行一个CCCallFunc action来调用一个方法setTappable。这个方法会把精灵的userData属性设置成True，表明当前地鼠是可以被击打的。
* 类似的，在地鼠笑完之后，同样运行一个CCCallFunc action来调用unsetTappable方法，把是否可击打的标记又设置回去。
* 只要超过50个地鼠从洞里钻出来后，这个方法就返回，因此，这个游戏的限制就是只出现50个地鼠。
* 在这个方法的开始部分，还把精灵的显示帧设置成初使图片（“mole.jpg”），因为，如果地鼠上一次被打中了，它下次再钻出来的时候，还会显示被打中。所以需要在它每次从洞里钻出来的时候，设置它的显示帧为初使图片。
* 好了，现在，这个精灵有一个userData标记，可以表明当前它是否可以被击打了。我们接下来，添加下面的击打检测代码：

-(void) registerWithTouchDispatcher

{

[[CCTouchDispatcher sharedDispatcher] addTargetedDelegate:self priority:kCCMenuTouchPriority swallowsTouches:NO];

}

-(BOOL) ccTouchBegan:(UITouch \*)touch withEvent:(UIEvent \*)event { CGPoint touchLocation = [self convertTouchToNodeSpace:touch]; for (CCSprite \*mole in moles) { if (mole.userData == FALSE) continue; if (CGRectContainsPoint(mole.boundingBox, touchLocation)) {

mole.userData = FALSE; score+= 10;

[mole stopAllActions]; CCAnimate \*hit = [CCAnimate actionWithAnimation:hitAnim restoreOriginalFrame:NO]; CCMoveBy \*moveDown = [CCMoveBy actionWithDuration:0.2 position:ccp(0, -mole.contentSize.height)]; CCEaseInOut \*easeMoveDown = [CCEaseInOut actionWithAction:moveDown rate:3.0]; [mole runAction:[CCSequence actions:hit, easeMoveDown, nil]]; } } return TRUE; }

这个registerWithTouchDispatcher方法会使得每一个touch事件到来的时候，都会先调用ccTouchBegan方法，如果ccTouchBegan返回yes，则有touch事件，否则没有。对于更多的细节信息，请查照第一篇教程[《如何使用cocos2d来制作一个基于tiled地图的游戏》](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E5%88%B6%E4%BD%9C%E5%9F%BA%E4%BA%8Etile%E5%9C%B0%E5%9B%BE%E7%9A%84%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B%EF%BC%9A%E7%AC%AC%E4%B8%80.html)。

ccTouchBegan方法把touch坐标转换成相对于层的本地坐标，然后循环遍历每一个地鼠。如果地鼠不可以击打（它的userData属性是false），那么就直接看下一个地鼠。否则的话，就使用CGRectContainPoint来检测touch点是否在地鼠的精灵边框之内。

如果地鼠被击中了，就把它设置成不可击打的，同时增加分数。并且停止所有正在运行的action，然后播放“被打中”的动画，并且立马把地鼠缩回洞里去。

最后一步–添加一些代码来更新分数label以及检查关卡是否完成。

if (gameOver) return;

[label setString:[NSString stringWithFormat:@"Score: %d", score]];

if (totalSpawns >= 50) {

CGSize winSize = [CCDirector sharedDirector].winSize;

CCLabelTTF \*goLabel = [CCLabelTTF labelWithString:@"Level Complete!" fontName:@"Verdana" fontSize:[self convertFontSize:48.0]]; goLabel.position = ccp(winSize.width/2, winSize.height/2); goLabel.scale = 0.1; [self addChild:goLabel z:10]; [goLabel runAction:[CCScaleTo actionWithDuration:0.5 scale:1.0]];

gameOver = true; return;

}

就这么多了！编译并运行，你现在可以尽情打地鼠赚分啦！你能得多少分呢？



**免费的音效**

和之前一样，让我们添加一些非常酷的音效。下载[这些音效](http://d1xzuxjlafny7l.cloudfront.net/downloads/WhackAMoleSounds.zip)（它们是用Garage Band和Audacity制作的，这两个在iPad上面有），解压之，并把它们拖到Resource文件夹下面。同时，确保  “Copy items into destination group’s folder”被选中，再点Add。

然后，修改HelloWorldScene.m：

// Add to top of file

#import ”SimpleAudioEngine.h”

// Add at the bottom of your init method [[SimpleAudioEngine sharedEngine] preloadEffect:@”laugh.caf”]; [[SimpleAudioEngine sharedEngine] preloadEffect:@”ow.caf”]; [[SimpleAudioEngine sharedEngine] playBackgroundMusic:@”whack.caf” loop:YES];

// Add at bottom of setTappable [[SimpleAudioEngine sharedEngine] playEffect:@”laugh.caf”];

// Add inside ccTouchBegan, inside the CGRectContainsPoint case [[SimpleAudioEngine sharedEngine] playEffect:@”ow.caf”];

编译并运行，心情享受打地鼠的乐趣吧！

**何去何从**

这里有本教程的[完整源代码](http://d1xzuxjlafny7l.cloudfront.net/downloads/WhackAMole2.zip)。

这个系列的教程到此基本上就结束了，为什么不往工程里添加更多的东西呢？我确定你可以往这个游戏添加一些更加好玩的元素。

如果你们有什么好的想法，或者好的建议，可以在下面留言。

译者的话：希望对大家有帮助。

**如何使用box2d来做碰撞检测（且仅用来做碰撞检测） by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/606/how-to-use-box2d-for-just-collision-detection-with-cocos2d-iphone>

程序截图：



当你使用cocos2d来制作一个游戏的时候，有时，你可能想使用cocos2d的action来移动游戏中的对象，而不是直接使用Box2d物理引擎来做。然而，这并不是说你不能使用box2d提供的强大的碰撞检测功能！

这个教程的目的，就是一步一步地向你展示如何仅使用box2d来做碰撞检测—没有物理效果。我们将创建一个简单的demo，里面有一辆车在屏幕上奔驰，当它撞到一只猫后就会放声大笑。是的，我明白，这样做太残忍了。

在这篇教程中，我们会引入一些新的、有趣的概念，比如使用sprite sheets，利用Zwoptex工具来制作它。还会涉及box2d的debug drawing和VertexHelper这个工具。

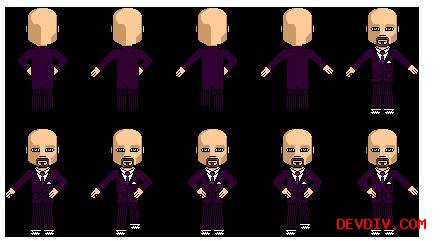
这个简单假设你已经阅读过前面的一系列的cocos2d和box2d的教程了（如果没有，最好把前面的教程先看一遍），或者你有相关经验也可以。

还有，在我忘记之前–特别要感谢Kim在评论中建议我写一篇这样的教程。

**Sprites and Sprite Sheets**

在我们开始之前，需要简单地介绍一下sprites和spritesheets。

目前为止，当我们使用cocos2d的时候，我们都是直接使用CCSprite类，然后传递一个精灵图片给它。但是，根据[cocos2d最佳实践](http://www.cocos2d-iphone.org/wiki/doku.php/prog_guide:best_practices)，如果使用spritesheet来做的话，那样会极大地提高效率。



上图就是一个spritesheet的样例，这个图在cocos2d的样例代码中可以找到。简言之，spritesheet就是一张大的图片，它能够被裁成许多小的子图片。为了指定spritesheet中的每个子图片的位置，你需要为每个图片指定一个坐标，通常是一个矩形。比如，下面的代码展示了，如何把spritesheet中前四张图片抠出来：

CCSpriteSheet \*sheet = [CCSpriteSheet

spriteSheetWithFile:@"grossini\_dance\_atlas.jpg" capacity:1];

CCSprite \*sprite1 = [CCSprite spriteWithTexture:sheet.texture rect:

CGRectMake(85\*0, 121\*1, 85, 121)];

CCSprite \*sprite2 = [CCSprite spriteWithTexture:sheet.texture rect:

CGRectMake(85\*1, 121\*1, 85, 121)];

CCSprite \*sprite3 = [CCSprite spriteWithTexture:sheet.texture rect:

CGRectMake(85\*2, 121\*1, 85, 121)];

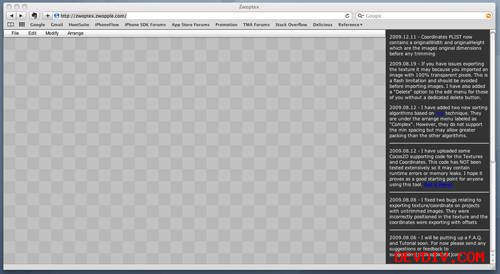
CCSprite \*sprite4 = [CCSprite spriteWithTexture:sheet.texture rect:

CGRectMake(85\*3, 121\*1, 85, 121)];

你可以会说，写一大堆硬编码的坐标太麻烦了，太烦人了！幸运的是，Robert Payne已经写好了一个非常方便地web工具，叫做Zwoptex，它可以创建精灵表单（spritesheet），同时会导出每个子图片的坐标，这样你在cocos2d里面使用这些子图片就会非常方便了。

**制作精灵表单**

在我们开始之前，你需要一张图片。你可以下载我老婆制作的[车](http://d1xzuxjlafny7l.cloudfront.net/downloads/car.jpg)和[猫](http://d1xzuxjlafny7l.cloudfront.net/downloads/cat.jpg)的图片，或者使用你自己的。接下来，打开浏览器，加载[Zwoptex主页](http://zwoptex.zwopple.com/)。你将会看到下面的屏幕：



一旦加载完毕后，点击Zwoptex的File菜单，然后点Import Images。选择你刚刚下载的车和猫的图片，然后点click，这时，你的图片应该出现会互相重叠在一起。拖动其中一张，这样会看得更清楚一些。

注意，这些图片已经被自动地把图片周围的空白部分给去掉了。这并不是我们想要的（后面你会知道为什么），因此，用鼠标把这两张图片都框住，然后选择菜单Modify\Untrim Selected Images。

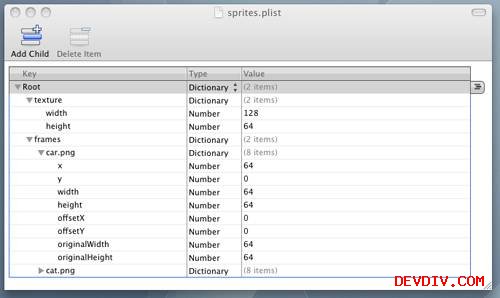
现在，我们的图片看起来非常好了。点击Arrange\Complex by Height（no spacing)，然后它们会排列地更加整齐。

最后，让我们把画布（canvas）的大小调整为合适的大小。点击Modify\Canvas Width，并把它设置为128px。同样的，点击Modify\Canvas Height，然后把它设置为64px。你的屏幕最后看起来应该是下面这个样子。



最后，是时候导出sprite sheet和相应的坐标了！点击File\Export Texture，然后保存sprite sheet为“sprites.jpg”。然后点击File\Export Coordinates“并且保存为”sprites.plist“。注意，这里必须把spritesheet和坐标文件的名字取成相同的，因为，spritesheet为假设它的坐标文件为相应名字的plist文件。

接下来，打开sprites.plist。你会看到Zwopte已经自动地帮你把原图中每一个子图片的坐标计算出来了，并且存储成了一个plist文件。我们接下来就可以使用这些坐标，而不用手工去输入它们了！



**从Sprite sheet中添加我们的精灵**

好，是时候写一些代码了！

打开Xcode并创建一个新的工程，选择 cocos2d-0.99.1 Box2d Application template，取名为Box2DCollision。然后把自带的样例代码全部删除，你可能从[弹球的教程](http://www.cnblogs.com/andyque/archive/2011/05/27/2059453.html)中找到具体的做法。

当然，我们要在HelloWorldScene.mm的顶部加入下面一行代码：

#define PTM\_RATIO 32.0

如果你不明白我在这里讲的是些什么，你可能需要看看[弹球的教程](http://www.cnblogs.com/andyque/archive/2011/05/27/2059453.html)来获取更多的信息。

接下来，让我们把sprite sheet和坐标plist文件都加入到工程中去。把sprites.jpg和sprites.plist文件都拖到Resouces文件夹中，同时确保  “Copy items into destination group’s folder (if needed)”被复选中。

然后，在HelloWorldScene.h文件的HelloWorld类中，添加下面的成员变量：

CCSpriteSheet \*\_spriteSheet;

现在，让我们修改HelloWorldScene.mm中的init方法来加载我们的spritesheet和plist文件。具体修改如下：

- (id)init {

if ((self=[super init])) {

// Create our sprite sheet and frame cache \_spriteSheet = [[CCSpriteSheet spriteSheetWithFile:@"sprites.jpg" capacity:2] retain]; [[CCSpriteFrameCache sharedSpriteFrameCache] addSpriteFramesWithFile:@”sprites.plist”]; [self addChild:\_spriteSheet];

[self spawnCar]; [self schedule:@selector(secondUpdate:) interval:1.0];

} return self;

}

第一件事情就是创建一个spritesheet对象，它是一个可以用来高效地绘制它的所有的CCSprite结点的对象。很明显，这些CCSprite必须共享相同的纹理（texture）。当我们把车子和猫加入到场景中的时候，我们需要将它们当作spritesheet的孩子加进去。

然后，我们使用CCSpriteFrameCache类来加载坐标属性列表文件。这个函数会自动地查找一个与之同名的图片（即sprites.jpg).这也就是前面说的，为什么要把”sprites.jpg“和”sprites.plist”取成相同名字的原因。

在这之后，我们调用一个函数在场景中显示一辆车。同时，还设置一个计时器，每隔一秒调用一次secondUpdate函数。

接下来，让我们实现spawnCar方法。我们的做法是让车子永远地在屏幕中间做路径为三角形的运动。在init函数的上面添加下面函数代码：

- (void)spawnCar {

CCSprite \*car = [CCSprite spriteWithSpriteFrameName:@"car.jpg"]; car.position = ccp(100, 100); car.tag = 2;

[car runAction:[CCRepeatForever actionWithAction: [CCSequence actions: [CCMoveTo actionWithDuration:1.0 position:ccp(300,100)], [CCMoveTo actionWithDuration:1.0 position:ccp(200,200)], [CCMoveTo actionWithDuration:1.0 position:ccp(100,100)], nil]]];

[\_spriteSheet addChild:car];

}

注意，这里创建sprite的时候，不是使用spriteWithFile，而是使用spriteWithSpriteFrameName。这里指的是使用spritesheet纹理中代表Car的图片来创建精灵。

还有一点需要注意，我们不是把Car作为HelloWorld层的函数添加进去，而是把Car作为Spritesheet的孙子添加进去的。

这个函数的后面部分你应该比较熟悉了。因此，让我们添加一些猫吧！在上面的spawnCar方法后面添加下面的方法：

- (void)spawnCat {

CGSize winSize = [CCDirector sharedDirector].winSize;

CCSprite \*cat = [CCSprite spriteWithSpriteFrameName:@"cat.jpg"];

int minY = cat.contentSize.height/2; int maxY = winSize.height - (cat.contentSize.height/2); int rangeY = maxY - minY; int actualY = arc4random() % rangeY;

int startX = winSize.width + (cat.contentSize.width/2); int endX = -(cat.contentSize.width/2);

CGPoint startPos = ccp(startX, actualY); CGPoint endPos = ccp(endX, actualY);

cat.position = startPos; cat.tag = 1;

[cat runAction:[CCSequence actions: [CCMoveTo actionWithDuration:1.0 position:endPos], [CCCallFuncN actionWithTarget:self selector:@selector(spriteDone:)], nil]];

[\_spriteSheet addChild:cat];

}

- (void)spriteDone:(id)sender {

CCSprite \*sprite = (CCSprite \*)sender; [\_spriteSheet removeChild:sprite cleanup:YES];

}

- (void)secondUpdate:(ccTime)dt {

[self spawnCat];

}

你应该对上面的代码线路熟悉了，如果不熟悉，建议看相关的教程后再继续。编译并运行代码，如果一切ok，你将会看到一辆车在屏幕上来回动，同时有一只猫从右至左穿过屏幕。接下来，我们将添加一些碰撞检测的代码。



**为这些精灵创建Box2d的body**

接下来的一步就是为每个精灵创建一个body，这样box2d就能知道它们的位置了，这样的话，当碰撞发生的时候，我们就可以被告知了。下面所做的事情和之前的教程做法差不多。

然后，这一次，我们不是更新box2d的body，然后再更新sprite。这里，我们是先更新sprite（使用action或者别的），然后再更新box2d的body。

因此，让我们首先创建world。打开HelloWorldScene.h，并在文件顶部添加下面的代码：

#import ”Box2D.h”

然后在HelloWorld类中添加下面的成员变量：

b2World \*\_world;

然后在HeloWorldScene.mm的init方法中加入下列代码：

b2Vec2 gravity = b2Vec2(0.0f, 0.0f);

bool doSleep = false;

\_world = new b2World(gravity, doSleep);

注意，这里有两件事情非常重要！首先，我们把重力向量设置成（0，0）。因为我们并不想让这些对象自动运动，而是人为控制其运动。其次，我们告诉box2d不能让这些对象sleep。这一点非常重要，因为我们是人为地移动对象，所以必须设置。

然后，在spawnCat方法上面添加下列代码：

- (void)addBoxBodyForSprite:(CCSprite \*)sprite {

b2BodyDef spriteBodyDef; spriteBodyDef.type = b2\_dynamicBody; spriteBodyDef.position.Set(sprite.position.x/PTM\_RATIO, sprite.position.y/PTM\_RATIO); spriteBodyDef.userData = sprite; b2Body \*spriteBody = \_world->CreateBody(&spriteBodyDef);

b2PolygonShape spriteShape; spriteShape.SetAsBox(sprite.contentSize.width/PTM\_RATIO/2, sprite.contentSize.height/PTM\_RATIO/2); b2FixtureDef spriteShapeDef; spriteShapeDef.shape = &spriteShape; spriteShapeDef.density = 10.0; spriteShapeDef.isSensor = true; spriteBody->CreateFixture(&spriteShapeDef);

}

这段代码对你来说应该比较熟悉了—它和我们在breakout教程中是一样的。然后，这里有一点区别，我们把shape定义的isSensor成员设置成了true。

根据[Box参考手册](http://www.box2d.org/manual.html)，如果你想让对象之间有碰撞检测但是又不想让它们有碰撞反应，那么你就需要把isSensor设置成true。这正是我们想要的效果！

接下来，在spawnCat的最后一行addChild之前添加下列代码：

[self addBoxBodyForSprite:cat];

同样的，你需要在spawnCar的相应位置添加下列代码：

[self addBoxBodyForSprite:car];

当我们的sprites被销毁的时候，我们需要销毁Box2d的body。因此，把你的spriteDone方法改写成下面的形式：

- (void)spriteDone:(id)sender {

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

b2Body \*spriteBody = NULL; for(b2Body \*b = \_world->GetBodyList(); b; b=b->GetNext()) { if (b->GetUserData() != NULL) { CCSprite \*curSprite = (CCSprite \*)b->GetUserData(); if (sprite == curSprite) { spriteBody = b; break; } } } if (spriteBody != NULL) { \_world->DestroyBody(spriteBody); }

[\_spriteSheet removeChild:sprite cleanup:YES];

}

现在，最重要的一部分，我们需要根据sprite的位置改变来更新box2d的body。因此，在init方法里面添加下面一行代码：

[self schedule:@selector(tick:)];

然后，把tick方法写成下面的形式：

- (void)tick:(ccTime)dt {

\_world->Step(dt, 10, 10); for(b2Body \*b = \_world->GetBodyList(); b; b=b->GetNext()) { if (b->GetUserData() != NULL) { CCSprite \*sprite = (CCSprite \*)b->GetUserData();

b2Vec2 b2Position = b2Vec2(sprite.position.x/PTM\_RATIO, sprite.position.y/PTM\_RATIO); float32 b2Angle = -1 \* CC\_DEGREES\_TO\_RADIANS(sprite.rotation);

b->SetTransform(b2Position, b2Angle); } }

}

这个方法和我们之前在breakout中写的tick方法差不多，唯一的差别就是，我们现在是基于cocos2d的精灵位置来更新box2d的body位置。

编译并运行工程，可能看起来和之前并没有什么差别。你可能会想，到底我们刚刚写了这么多代码有用没啊！好，接下来我就会展示给你看—激活debug drawing！

**激活 Box2D 的Debug Drawing**

因为，你是使用box2d的模板建的项目，所以里面已经包含了一个GLES-Render.h和GLES-Render.mm文件，如果想要激活debug drawing，有这两个文件就足够了！

接下来，让我们在HelloWorldScene.h的顶部包含下面的头文件：

#import ”GLES-Render.h”

然后，在HelloWorld类中添加以下成员变量：

GLESDebugDraw \*\_debugDraw;

接下来，在init方法中添加下面的代码：

// Enable debug draw

\_debugDraw = new GLESDebugDraw( PTM\_RATIO );

\_world->SetDebugDraw(\_debugDraw);

uint32 flags = 0; flags += b2DebugDraw::e\_shapeBit; \_debugDraw->SetFlags(flags);

这段代码是创建一个GLESDebug类，并且把它注册到world对象里面。我们传递一个flag来指定我们需要显示的仿真细节—这里我们只想显示box2d的shape。对于你可以设置的所有的flag标志，可以参考b2WorldCallbacks.h文件。

接下来，我们需要添加一个draw方法。在init方法后面添加draw方法：

-(void) draw

{

glDisable(GL\_TEXTURE\_2D);

glDisableClientState(GL\_COLOR\_ARRAY);

glDisableClientState(GL\_TEXTURE\_COORD\_ARRAY);

\_world->DrawDebugData();

glEnable(GL\_TEXTURE\_2D); glEnableClientState(GL\_COLOR\_ARRAY); glEnableClientState(GL\_TEXTURE\_COORD\_ARRAY); }

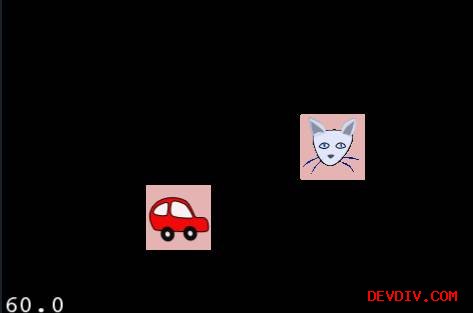
老实说，我现在仍然对opengl陌生，因此，我并不是很清楚这里的opengl代码做了些什么事。然后，这是激活debug draw的标准代码，它可以工作！：）

最后需要注意的一点，在dealloc函数中添加清理代码：

- (void)dealloc {

delete \_world; delete \_debugDraw; [super dealloc]; }

现在，编译并运行工程，你将会看到所有的box2d shape周围都有一个粉红色的矩形区域。如果一切ok的话，你会看到这些粉红色的shape会跟着sprite运动。



**碰撞检测**

现在，是时候压死几只猫了！

像之前的breakout游戏一样，我们将往world对象里面注册一个contact listener对象。你可以下载我为这个教程制作的[contact listener代码](http://d1xzuxjlafny7l.cloudfront.net/downloads/MyContactListener.zip)，然后把 MyContactListener.h 和 MyContactListener.mm两个文件添加到工程中去：

同时，你还可以下载我为本教程制作的[美妙的音效](http://d1xzuxjlafny7l.cloudfront.net/downloads/hahaha.caf.zip)。

回到代码。在HelloWorldScene.h中添加下面的代码：

#import ”MyContactListener.h”

#import ”SimpleAudioEngine.h”

然后，在HelloWord类中添加下面成员变量：

MyContactListener \*\_contactListener;

然后，在init方法中加入下面的代码：

// Create contact listener

\_contactListener = new MyContactListener();

\_world->SetContactListener(\_contactListener);

// Preload effect [[SimpleAudioEngine sharedEngine] preloadEffect:@”hahaha.caf”];

最后，在tick方法的底部加入下面的代码：

std::vector<b2Body \*>toDestroy;

std::vector<MyContact>::iterator pos;

for(pos = \_contactListener->\_contacts.begin();

pos != \_contactListener->\_contacts.end(); ++pos) {

MyContact contact = \*pos;

b2Body \*bodyA = contact.fixtureA->GetBody(); b2Body \*bodyB = contact.fixtureB->GetBody(); if (bodyA->GetUserData() != NULL && bodyB->GetUserData() != NULL) { CCSprite \*spriteA = (CCSprite \*) bodyA->GetUserData(); CCSprite \*spriteB = (CCSprite \*) bodyB->GetUserData();

if (spriteA.tag == 1 && spriteB.tag == 2) { toDestroy.push\_back(bodyA); } else if (spriteA.tag == 2 && spriteB.tag == 1) { toDestroy.push\_back(bodyB); } } }

std::vector<b2Body \*>::iterator pos2; for(pos2 = toDestroy.begin(); pos2 != toDestroy.end(); ++pos2) { b2Body \*body = \*pos2; if (body->GetUserData() != NULL) { CCSprite \*sprite = (CCSprite \*) body->GetUserData(); [\_spriteSheet removeChild:sprite cleanup:YES]; } \_world->DestroyBody(body); }

if (toDestroy.size() > 0) { [[SimpleAudioEngine sharedEngine] playEffect:@”hahaha.caf”]; }

这段代码和我们之前的breakout教程中的代码基本上差不多，因此你应该比较熟悉了。

还有一件事别忘了，往dealloc方法里面加入清理代码！这个灰常重要！

delete \_contactListener;

[\_spriteSheet release];

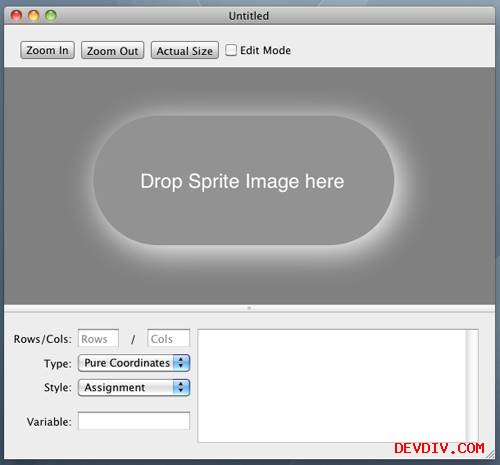
再运行一下工程看看！

**调整shape的边界**

你可能已经注意到了，我们的box2d shape的边界并不是和sprite的边界完全吻合。对于有些碰撞检测要求不是特别精确的游戏，这也许够了，但是，有些游戏确不行！我们需要严格地定义shape的边界和精灵的边界重合在一起。

在Box2d里面，你可以通过指定shape的顶点来指定shape的形状。然后，硬编码这些顶点数据会很耗时，而且容易出错。幸运的是， [Johannes Fahrenkrug](http://twitter.com/jfahrenkrug)已经开发出了一个非常方便的工具，叫做 [VertexHelper](http://blog.springenwerk.com/2010/02/introducing-vertexhelper-for-box2d.html)，它可以用来非常方便地定义顶点，而且可以导出box2d所需要的格式的数据。

好了，先去下载VertexHelper吧。他是一个Mac应用程序，同时包含了源代码，因此，你只需要打开  VertexHelper.xcodeproj，然后编译并运行就可以了。当运行工程的时候，你会看到下面的屏幕输出：（为了方便起见，你可以把编译好的工程放到Application文件夹下面，以后就直接打开就可以了）



继续，把sprites.jpg拖到VertexHelper中，拖的时候，放置在 “Drop Sprite Image here”标签上面。在Rows/Cols部分，把相应的数字设置成1和2.VertexHelper会自动地把图片划分成两个部分。

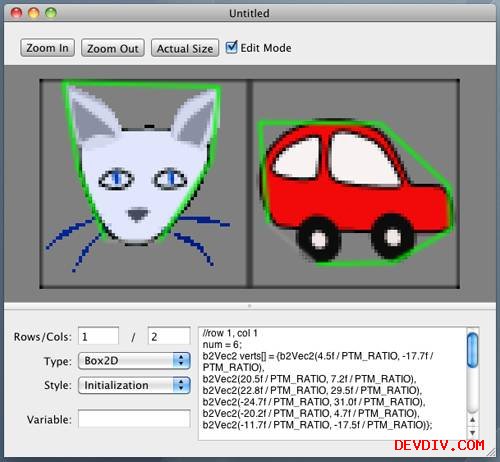
接下来，把”Edit Mode”复选框打上勾，紧接着，你需要按照逆时针方向在精灵的四周定义一些顶点。注意，box2d将自动地把最后一个顶点与第一个顶点连接起来，因此，我们不需要连接它们。

另外一件非常重要的事情是，当你定义顶点的时候，你需要确保定义的多边形是凸多边形。这意味着多边形的内部没有一个角大于180度。或者说是，多边形内任何两个顶点的连线都在多边形的内部。如果你对这个定义不是很清楚的话，建议百度一下凸多边形和凹多边形。当然，也可以看看下面这个js制作的[demo](http://www.mathopenref.com/polygonconvex.html)。

最后，注意box2d定义了b2\_maxPolygonVertices，它限制了每一个shape最多可以定义的顶点的个数，默认值是8.当然，你可以在b2Settings.h中把这个值改掉。但是，这个教程中，我们只需要8个顶点就够了。

（这个过程最好用一个视频来演示一下，不过翻不了墙，也看不了了。不过自己多摸索一下，这个工具还是很容易使用的）

一旦你做完之后，在下拉列表中选择Box2d，Style选择“Initialization”。然后把右边生成的代码copy下来并粘贴到工程中去。



好，让我们把工程中的shape定义换一下。打开HelloWorldScene.mm，然后修改 addBoxBodyForSprite方法。首先注释掉一些代码，如下图所示：

/\*spriteShape.SetAsBox(sprite.contentSize.width/PTM\_RATIO/2,

sprite.contentSize.height/PTM\_RATIO/2);\*/

if (sprite.tag == 1) {

// Uncomment this and replace the number with the number of vertices

// for the cat that you defined in VertexHelper

//int num = 6;

//b2Vec2 verts[] = {b2Vec2(4.5f / PTM\_RATIO, -17.7f / PTM\_RATIO),

//b2Vec2(20.5f / PTM\_RATIO, 7.2f / PTM\_RATIO),

//b2Vec2(22.8f / PTM\_RATIO, 29.5f / PTM\_RATIO),

//b2Vec2(-24.7f / PTM\_RATIO, 31.0f / PTM\_RATIO),

//b2Vec2(-20.2f / PTM\_RATIO, 4.7f / PTM\_RATIO),

//b2Vec2(-11.7f / PTM\_RATIO, -17.5f / PTM\_RATIO)};

// Then add this

//spriteShape.Set(verts, num);

} else {

// Do the same thing as the above, but use the car data this time

}

再多啰嗦一句：在选择Style的时候一定要选择“Initialization”，因为b2PolygonShape:Set方法对于“Initialization”会自动计算shape的[法向量](http://en.wikipedia.org/wiki/Surface_normal)和[质心](http://en.wikipedia.org/wiki/Centroid)。

一旦做完之后，编译并运行工程。这时，你会看到shape的边界基本上和sprite的边界吻合在一起了，当然碰撞检测就会更加真实了！



恩，你已经看到了使用box2d来做碰撞检测的强大了！

发挥想象，去做更多好玩的物理效果游戏吧！

**总结！**

这里有本教程的[完整源代码](http://d1xzuxjlafny7l.cloudfront.net/downloads/Box2DCollision.zip)。

注意，这里用来做碰撞检测的代码，仅仅是box2d里面实现碰撞检测的一种方式。Lam在cocos2d论坛里面提出了另外一种方法，使用b2CollidePolygons来做碰撞检测，详情请参考[这里](http://www.cocos2d-iphone.org/forum/topic/2692)。如果你只想做碰撞检测，你可能想看看lam的实现。

对于其它开发者全体在他们的项目里面使用cocos2d和box2d，我非常之感兴趣！你在你的项目中使用box2d吗？或者仅仅用它来做碰撞检测，或者根本不使用它？或者如果你使用box2d来做碰撞检测，你是怎么做的呢？

欢迎大家踊跃发言，分享自己的经验和看法，谢谢！

**如何使用cocos2d制作一个太空射击游戏 by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/3611/how-to-make-a-space-shooter-iphone-game>

程序截图：



在这个教程里面，你将会学习到如何为iPhone开发一个太空射击游戏！

你可以使用加速计（重力感应）来控制飞船的移动，并且可以点击屏幕来发射激光武器。

如果你对于如何制作iphone游戏完全陌生的话，这个教程可以帮助你！你将会学习到，如何从头至尾构建一个完整的游戏，不需要任何的经验！

假如你对cocos2d编程完全陌生的话，那么你可能需要先学习一下[相关的教程了](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8cocos2d%E6%9D%A5%E5%81%9A%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84iphone%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B1.html)。

这个教程对于中级开发者来说也非常好，因为它覆盖了一些比较高级的主题，比如视差滚动（parallax scrolling），预分配CCNode，加速计移动以及粒子系统的使用。

话不多说，直入主题！

**安装cocos2d**

为了制作这个游戏，你需要成为  [iOS developer program](http://developer.apple.com/programs/ios/)的一员（这样的话，你的程序就能够安装到你的iPhone上面去，不过听说网上有人越狱也可以安装，知道的朋友麻烦给个链接，谢谢！）同时，需要安装[Xcode](http://developer.apple.com/xcode/)和[cocos2d](http://www.cocos2d-iphone.org/download)框架。

如果你之前已经安装过cocos2d了，那接下来这部分就不要看了。（如果想使用新的cocos2d版本，只需要把之前安装的目录下面的模板文件全部删除，再按照下面的指令重新安装即可）。如果你重来没有安装过cocos2d的话，那么只需要按照下面的指令序列，一步步地安装到你的mac上面就行了。

* 首先下载[Cocos2D](http://www.cocos2d-iphone.org/download) .确定获得最新版本–作者写这篇文章的时候是1.0.0-rc2版本，目前是1.0.0-rc3版本。虽然不是稳定版本，但是，没关系，其实很稳定啦！：）
* 双击下载下来的文件，并且解压缩到一个安全的位置。
* 打开Terminal (Applications\Utilities\Terminal), 然后使用cd命令定位到刚刚解压缩的cocos2d文件夹下面去。然后运行./install-templates.sh来安装xcode模板，如下面所示：

$ cd Downloads

$ cd cocos2d-iphone-1.0.0-rc2

$ ./install-templates.sh -f -u

如果一切顺利的话，你应该会看到终端里面一系列的输出语句：“ Installing xxx template”。

然后重新启动Xcode，祝贺你，您已成功安装cocos2d了！

**Hello, Cocos2D!**

让我们首先创建一个“Hello World”cocos2d工程。

打开Xcode，选择 File\New\New Project，然后选 iOS\cocos2d template，接下来点Next，并且把工程命名为SpaceGame，再点Next并选择一个文件夹作为你的工程的保存路径，最后点Create。

编译并运行工程，你将会看到一个“Hello World”出现在屏幕的正中间。



**添加相关资源文件**

为了做这样一个iphone游戏，你将需要一些跟太空主题相关的图片资源和声音资源。

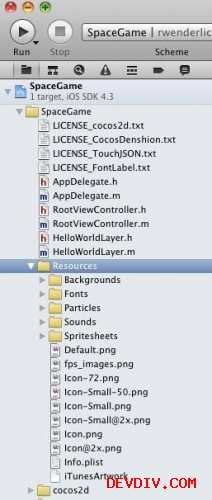
你可以直接下载我老婆制作的[太空游戏资源](http://d1xzuxjlafny7l.cloudfront.net/downloads/SpaceGameResources.zip)。

因此，请直接下载吧，并且把它解压到你的硬盘的某个目录下面去。

一旦你解压完这些资源以后，把Backgrounds，Fonts，Particles，Sounds和Spritesheets文件夹拖到Resouces分组下面去。（基本上，除了Classes文件夹以外，其它所有的文件压都拖到Resource目录下面去）

确保 “Copy items into destination group’s folder (if needed)”被复选中，然后点击Finish。

当你做完这些事之后，你的工程的分组看起来会是下图所示的样子：

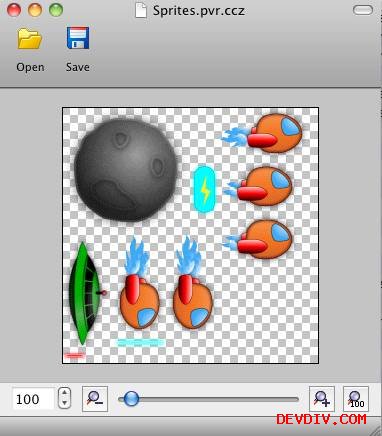


如果你很好奇，你可以随便看看你刚刚向工程里面添加进去了一些什么东西。下面是完整的内容列表：

* **Backgrounds**: 一些背景图片，你等下会使用它们来制作一个滚动背景。里面包含星系，太阳，和空间异常（它移动速度比较慢），还有一组空间尘埃图片（它们会出现在背景前面，而且会移动地稍微快一点）
* **Fonts**: 使用 [Glyph Designer](http://glyphdesigner.71squared.com/)制作的位图字体，我们将使用这些字体来在游戏中显示文字。
* **Particles**: 使用  [Particle Designer](http://particledesigner.71squared.com/)制作的一些特殊的粒子效果。 在这里，我们用来创建星星飞动的效果。
* **Sounds**: 一些与太空相关的背景音乐和音效。使用 [Garage Band](http://www.apple.com/ilife/garageband/) 和 [cxfr](http://thirdcog.eu/apps/cfxr)制作的。
* **Spritesheets**: 一张格式为pvr.ccz的大图片，里面包含了游戏中将要用到的许多小图片，比如陨石，太空船等。这个文件使用 [Texture Packer](http://www.shareit.com/product.html?cart=1&productid=300421849&backlink=http%3A%2F%2Fwww.raywenderlich.com&affiliateid=200142990)制作的—如果你想使用pvr.ccz文件格式的话，你可能就需要使用这个工具。当然pvr.ccz格式的优点就是文件小，加载速度快。

如果你还没安装上面任何一款工具的话，也不用担心！对于这个教程来说，你完全不需要他们，你可以使用我已经制作好的这些资源就够了。以后，如果有条件，你可以再去试试上面提到的工具。

下面就是Sprites.pvr.ccz文件，它看起来如下图所示：



你可能会奇怪，为什么要把所有的这些图片都弄成这样一张大图呢？因为，首先，它可以帮助节省内存，同时还可以提高性能。

接下来，让我们开始coding吧！：）

**添加一个太空船**

首先，让我们在屏幕上添加一艘太空船吧！

打开HelloWorldLayer.h文件，然后在@interface里面添加两个实例变量：

CCSpriteBatchNode \*\_batchNode;

CCSprite \*\_ship;

第一个变量 (\_batchNode)是必须的，因为我们将把所有的图片存储在一张图片里面，然后使用这个BatchNode就可以仅使用一次opengl调用来做所有的绘图操作。

第二个变量（\_ship)代表屏幕上的太空飞船。

接下来，打开HelloWorldLayer.m，并且把init方法改成下面的样子：

-(id) init

{

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

\_batchNode = [CCSpriteBatchNode batchNodeWithFile:@"Sprites.pvr.ccz"]; // 1 [self addChild:\_batchNode]; // 2 [[CCSpriteFrameCache sharedSpriteFrameCache] addSpriteFramesWithFile:@”Sprites.plist”]; // 3

\_ship = [CCSprite spriteWithSpriteFrameName:@"SpaceFlier\_sm\_1.png"]; // 4 CGSize winSize = [CCDirector sharedDirector].winSize; // 5 \_ship.position = ccp(winSize.width \* 0.1, winSize.height \* 0.5); // 6 [\_batchNode addChild:\_ship z:1]; // 7 } return self; }

让我们一句一句地解释上面的代码：

* 使用一张大的图片创建一个CCSpriteBatchNode对象来批处理所有的对象的描绘操作。接收的参数是Sprites.pvr.ccz。
* 把CCSpriteBatchNode添加到当前层里面去，这样就可以绘制它的所有的孩子对象。
* 加载Sprites.plist文件，它里面包含了这张大图里面的所有的小图的位置坐标信息。这样，你以后可以非常方便地使用 spriteWithSpriteFrameName来提取一张张小图片来初使化一些精灵。
* 使用 SpaceFlier\_sm\_1.png图片来创建一个精灵，注意这张图片是大图里的一个子图。
* 使用CCDirector来获得屏幕的大小—我们接下来会用到这个大小。
* 设置飞船的位置在屏幕宽度的10％，高度的50处。注意，飞船的中心点位置默认是飞船的中心。
* 把ship当作batchNode的一个孩子添加进去，这样的话，这些精灵就会被批处理显示出来。

编译并运行工程，你将会看到你的飞船图片出现在屏幕上面啦！



**添加视差滚动**

我们已经有一个很酷的飞船在屏幕上了，但是，它看起来就好像坐在那里一样，毫无生气！我们可以通过往里面添加视差滚动背景来解决这个问题。

但是，等一下，到底什么是视差滚动了？

视差滚动，简单来说，就是“移动背景中的一些图片比其它图片慢一点点”，打个比方，一个背景中的物体有远有近，近的背景移动地快（比如地面），远的背景移动地慢（比如天空），这样子就会形成景深不一样的视差效果出来。

想要在cocos2d里面使用视差滚动效果非常简单。你只需要做3步就ok了：

* 创建一个CCParallaxNode，然后把它加到层中去。
* 创建你想要滚动的元素，然后通过调用CCParallaxNode的 addChild:parallaxRatio:positionOffset方法把这些元素添加进去。
* 移动CCParallaxNode来滚动背景。这样的话，CCParallaxNode就会根据parallaxRatio的不同，或快或慢地移动它里面添加的元素了。

让我们看看这个过程具体是怎样的。打开HelloWorldLayer.h，然后在@interface里面加入下面代码：

CCParallaxNode \*\_backgroundNode;

CCSprite \*\_spacedust1;

CCSprite \*\_spacedust2;

CCSprite \*\_planetsunrise;

CCSprite \*\_galaxy;

CCSprite \*\_spacialanomaly;

CCSprite \*\_spacialanomaly2;

然后，转到HelloWorldLayer.m文件，在init方法的底部加入下面的代码：

// 1) Create the CCParallaxNode

\_backgroundNode = [CCParallaxNode node];

[self addChild:\_backgroundNode z:-1];

// 2) Create the sprites we’ll add to the CCParallaxNode \_spacedust1 = [CCSprite spriteWithFile:@"bg\_front\_spacedust.png"]; \_spacedust2 = [CCSprite spriteWithFile:@"bg\_front\_spacedust.png"]; \_planetsunrise = [CCSprite spriteWithFile:@"bg\_planetsunrise.png"]; \_galaxy = [CCSprite spriteWithFile:@"bg\_galaxy.png"]; \_spacialanomaly = [CCSprite spriteWithFile:@"bg\_spacialanomaly.png"]; \_spacialanomaly2 = [CCSprite spriteWithFile:@"bg\_spacialanomaly2.png"];

// 3) Determine relative movement speeds for space dust and background CGPoint dustSpeed = ccp(0.1, 0.1); CGPoint bgSpeed = ccp(0.05, 0.05);

// 4) Add children to CCParallaxNode [\_backgroundNode addChild:\_spacedust1 z:0 parallaxRatio:dustSpeed positionOffset:ccp(0,winSize.height/2)]; [\_backgroundNode addChild:\_spacedust2 z:0 parallaxRatio:dustSpeed positionOffset:ccp(\_spacedust1.contentSize.width,winSize.height/2)]; [\_backgroundNode addChild:\_galaxy z:-1 parallaxRatio:bgSpeed positionOffset:ccp(0,winSize.height \* 0.7)]; [\_backgroundNode addChild:\_planetsunrise z:-1 parallaxRatio:bgSpeed positionOffset:ccp(600,winSize.height \* 0)]; [\_backgroundNode addChild:\_spacialanomaly z:-1 parallaxRatio:bgSpeed positionOffset:ccp(900,winSize.height \* 0.3)]; [\_backgroundNode addChild:\_spacialanomaly2 z:-1 parallaxRatio:bgSpeed positionOffset:ccp(1500,winSize.height \* 0.9)];

编译并运行工程，你将会看到飞船背景了。



然而，这还不是很有趣，因为还没有任何东西在动！

为了移动太空尘埃和相关背景层，你只需要移动一样东西就可以了，那就是parallaxNode。对于移动parallax node的每一个y值，灰尘就会移动0.1y值，同时背景会移动0.05y值。

为了移动parallax节点，你只需要飞一帧更新一下它的位置就可以了。打开HelloWorldLayer.m文件，加入下列的代码：（添加位置注意看注释）

// Add to end of init method

[self scheduleUpdate];

// Add new update method - (void)update:(ccTime)dt {

CGPoint backgroundScrollVel = ccp(-1000, 0); \_backgroundNode.position = ccpAdd(\_backgroundNode.position, ccpMult(backgroundScrollVel, dt));

}

编译并运行工程，你会看到，使用parallax来做视差滚动效果真是太简洁了！



然后，运行几秒钟之后，你会发现一个问题：背景滚动完之后没有了！我们只得到了一个黑色的屏幕！那真是太糟糕了！好，接下来，看看我是怎么解决的吧！

**连续地滚动**

我们想要背景保持无限地连续滚动效果。我们的做法就是，当背景移出屏幕的左边的时候，就马上把它移动到屏幕右边的适当的位置上去。

这里有一个小小的问题，目前CCParallaxNode并不支持直接修改它的每个孩子的offset。你不能够简单地更新它的孩子的坐标点，因为CCParallaxNode每次更新的时候会覆盖那些改变。

不过没关系，我已经制作了一个CCParallaxNode的分类，它可以用来解决这个问题，这个Category可以在[项目的资源文件](http://d1xzuxjlafny7l.cloudfront.net/downloads/SpaceGameResources.zip)下面的Classes文件夹中找到。把 CCParallaxNode-Extras.h 和 CCParallaxNode-Extras.m拖到工程中去，同时确保 “Copy items into destination group’s folder”被复选中，然后点击Finish。

然后，在HelloWorldLayer.m中做下列改变来实现连续滚动的效果：

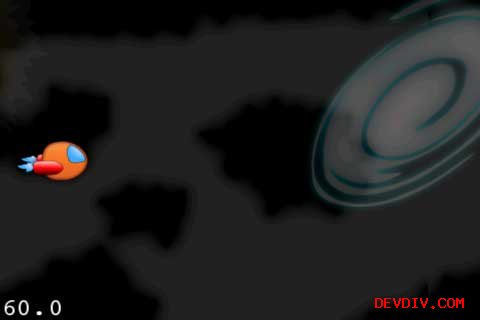
// Add to top of file

#import ”CCParallaxNode-Extras.h”

// Add at end of your update method NSArray \*spaceDusts = [NSArray arrayWithObjects:\_spacedust1, \_spacedust2, nil]; for (CCSprite \*spaceDust in spaceDusts) { if ([\_backgroundNode convertToWorldSpace:spaceDust.position].x < -spaceDust.contentSize.width) { [\_backgroundNode incrementOffset:ccp(2\*spaceDust.contentSize.width,0) forChild:spaceDust]; } }

NSArray \*backgrounds = [NSArray arrayWithObjects:\_planetsunrise, \_galaxy, \_spacialanomaly, \_spacialanomaly2, nil]; for (CCSprite \*background in backgrounds) { if ([\_backgroundNode convertToWorldSpace:background.position].x < -background.contentSize.width) { [\_backgroundNode incrementOffset:ccp(2000,0) forChild:background]; } }

编译并运行工程，这时你可以看到有一个无限连续滚动的背景了！



**添加星星**

没有哪一个太空射击游戏是没有星星在旁边飞的！

同样的，我们也可以创建相应的星星图片，并且把它当作paralla节点的一个孩子添加进去。但是，这里我们不想这么做，因为星星是一个非常好的介绍粒子系统的例子。

粒子系统能够让你使用同样的精灵创建大量的小对象，并且非常高效。cocos2d给了你很多配置粒子系统的参数， [Particle Designer](http://particledesigner.71squared.com/)这个工具可以使这些参数的配置可视化。

但是，这个教程里面，我们不会涉及如何调整粒子系统的各项参数，我们使用已经做好的星星的粒子效果。只需要在init方法中把它简单地添加进来即可：

NSArray \*starsArray = [NSArray arrayWithObjects:@"Stars1.plist", @"Stars2.plist", @"Stars3.plist", nil];

for(NSString \*stars in starsArray) {

CCParticleSystemQuad \*starsEffect = [CCParticleSystemQuad particleWithFile:stars];

[self addChild:starsEffect z:1];

}

通过把粒子系统加到层中去，接着它会自动运行起来。编译并运行代码，你现在可以看到好多星星在屏幕上飞过了！：）



**使用加速计来移动飞船**

到目前为止，还是很好，但是，如果我们不能控制飞船的移动的话，那就不是一个完整的游戏！

这里，我们将采用加速计来移动太空飞船。当用户沿着x轴方向倾斜设备的时候，飞船就会上下移动。

这个功能实际上非常容易实现。首先，在HelloWorldLayer.h里面的@interface里面添加一个成员变量，用来记录飞船沿着y轴方向每秒移动的点的个数。（这个点不一定等于一个实际的像素点，如果是Retina的设备，一个点=2个像素）

float \_shipPointsPerSecY;

然后，在HelloWorldLayer.m中做如下修改：

// 1) Add to bottom of init

self.isAccelerometerEnabled = YES;

// 2) Add new method - (void)accelerometer:(UIAccelerometer \*)accelerometer didAccelerate:(UIAcceleration \*)acceleration {

#define kFilteringFactor 0.1 #define kRestAccelX -0.6 #define kShipMaxPointsPerSec (winSize.height\*0.5) #define kMaxDiffX 0.2

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 accelDiff = accelX - kRestAccelX; float accelFraction = accelDiff / kMaxDiffX; float pointsPerSec = kShipMaxPointsPerSec \* accelFraction;

\_shipPointsPerSecY = pointsPerSec;

}

// 4) Add to bottom of update CGSize winSize = [CCDirector sharedDirector].winSize; float maxY = winSize.height - \_ship.contentSize.height/2; float minY = \_ship.contentSize.height/2;

float newY = \_ship.position.y + (\_shipPointsPerSecY \* dt); newY = MIN(MAX(newY, minY), maxY); \_ship.position = ccp(\_ship.position.x, newY);

让我们一点点剖析一下这段代码:

* 添加这行代码的作用是让当前的层可以接收到加速计移动事件，当有事件发生的时候，会回调 accelerometer:didAcccelerate这个方法。
* 这个方法的前面一部分是直接从Apple的样例代码中copy过来的，所做的事情其实是某种意义上的“滤波”。其实你也不用理解它的原理，基本上就是为了让飞船的移动更加平滑。如果你实在是对此非常感兴趣，可以查看[这里](http://en.wikipedia.org/wiki/High-pass_filter)来获得一些信息。不管怎么说，在运行完这个滤波之后，我们可以测试下看看，到底有哪些改进。通过实现证明，这种方法确实感觉不错！
* 基于每秒沿y轴方向移动的点数，来更新飞船的位置。同时要注意边界值的判断。

编译并运行工程（一定要编译到真机上面，否则模拟器是没有加速计效果的，你不可能抱着电脑摇吧：））。这时，上下晃动真机，你可以操作你的飞船移动了！



**添加陨石**

这个游戏目前看起来还不错，但是，危险和激情在哪里呢？让我们往场景中添加一些陨石吧！

我们接下来将要采纳的方法其实是非常普遍的做法，我们在右边屏幕之后创建一些陨石，然后利用cocos2d的action把它移动到屏幕的左边去。

我们可以在每次需要一个陨石的时候，马上创建一个对象，但是，分配内存的操作是非常慢的！所以你最好不要这样做！因此，我们可以预先分配好一堆陨石对象，这样当需要一个陨石对象的时候，就从中抓取一个就ok了。

好，我们看看具体该怎么做吧。打开HelloWorldLayer.h，然后往类中添加下列成员变量：

CCArray \*\_asteroids;

int \_nextAsteroid;

double \_nextAsteroidSpawn;

接下来在HelloWorldLayer.m里面做如下修改：

// Add to top of file

#define kNumAsteroids 15

// Add to bottom of init \_asteroids = [[CCArray alloc] initWithCapacity:kNumAsteroids]; for(int i = 0; i < kNumAsteroids; ++i) { CCSprite \*asteroid = [CCSprite spriteWithSpriteFrameName:@"asteroid.png"]; asteroid.visible = NO; [\_batchNode addChild:asteroid]; [\_asteroids addObject:asteroid]; }

// Add new method, above update loop - (float)randomValueBetween:(float)low andValue:(float)high { return (((float) arc4random() / 0xFFFFFFFFu) \* (high - low)) + low; }

// Add to bottom of update loop double curTime = CACurrentMediaTime(); if (curTime > \_nextAsteroidSpawn) {

float randSecs = [self randomValueBetween:0.20 andValue:1.0]; \_nextAsteroidSpawn = randSecs + curTime;

float randY = [self randomValueBetween:0.0 andValue:winSize.height]; float randDuration = [self randomValueBetween:2.0 andValue:10.0];

CCSprite \*asteroid = [\_asteroids objectAtIndex:\_nextAsteroid]; \_nextAsteroid++; if (\_nextAsteroid >= \_asteroids.count) \_nextAsteroid = 0;

[asteroid stopAllActions]; asteroid.position = ccp(winSize.width+asteroid.contentSize.width/2, randY); asteroid.visible = YES; [asteroid runAction:[CCSequence actions: [CCMoveBy actionWithDuration:randDuration position:ccp(-winSize.width-asteroid.contentSize.width, 0)], [CCCallFuncN actionWithTarget:self selector:@selector(setInvisible:)], nil]];

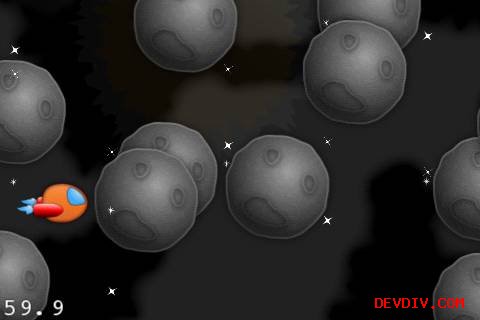
}

// Add new method - (void)setInvisible:(CCNode \*)node { node.visible = NO; }

关于上面的代码，有几点需要说明一下：

* CCArray 和NSArray差不多，但是做了速度方面的优化。所以如果可以的话，尽可能多地使用CCArray。
* 注意，我们添加了15个陨石到batchNode中去了。但是，把它们都设置成了不可见。如果是不可见的，就把它当前是未激活的。
* 我们使用一个实例变量  (\_nextAsteroidSpawn) 来指示下一个陨石出现的时间点。我们会在update循环中一直检测这个变量的值。
* 如果你对cocos2d的action还很陌生的话，其实action就是一种很简单的，可以让精灵在一段时间做一些事情的对象。比如，你可以让精灵在指定的一段时间内，让它旋转、缩放、移动等等。这里，我们执行了2个action的序列：一个从右边移动到左边的action，还有一个是当前一个action结束时，调用另一个函数把陨石设置为不可见的action。这两个action的顺序是固定的。

编译并运行代码，这时候你可以看见一些陨石在屏幕上飞过啦！



**发射激光**

我不清楚你是怎么想的，当我看到屏幕上有陨石在飞的时候，我的第一感觉就是，把它们干掉！

因此，让我们给飞船添加发射激光武器的功能吧！这段代码和我们之前添加陨石的代码有点类似，因为我们会创建一组可以重用的激光束，同时使用action来移动这些激光束。

主要的区别就是，我们将使用touch事件来发射激光。

首先打开HelloWorldLayer.h，然后在类中添加下面成员变量：

CCArray \*\_shipLasers;

int \_nextShipLaser;

然后在HelloWorldLayer.m做如下修改：

// Add to top of file

#define kNumLasers 5

// Add to bottom of init \_shipLasers = [[CCArray alloc] initWithCapacity:kNumLasers]; for(int i = 0; i < kNumLasers; ++i) { CCSprite \*shipLaser = [CCSprite spriteWithSpriteFrameName:@"laserbeam\_blue.png"]; shipLaser.visible = NO; [\_batchNode addChild:shipLaser]; [\_shipLasers addObject:shipLaser]; }

self.isTouchEnabled = YES;

// Add new method - (void)ccTouchesBegan:(NSSet \*)touches withEvent:(UIEvent \*)event {

CGSize winSize = [CCDirector sharedDirector].winSize;

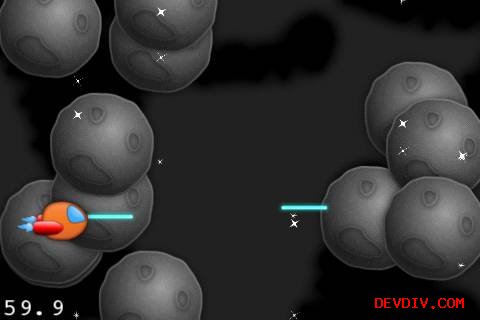
CCSprite \*shipLaser = [\_shipLasers objectAtIndex:\_nextShipLaser]; \_nextShipLaser++; if (\_nextShipLaser >= \_shipLasers.count) \_nextShipLaser = 0;

shipLaser.position = ccpAdd(\_ship.position, ccp(shipLaser.contentSize.width/2, 0)); shipLaser.visible = YES; [shipLaser stopAllActions]; [shipLaser runAction:[CCSequence actions: [CCMoveBy actionWithDuration:0.5 position:ccp(winSize.width, 0)], [CCCallFuncN actionWithTarget:self selector:@selector(setInvisible:)], nil]];

}

这个例子也向你展示了，在cocos2d里面接收touch事件是多么容易啊—仅需要把isTouchEnabled设置为yes就ok了。然后你需要实现ccTouchesBeban（或者ccTouchesMoved，ccTouchesEnded等等）。

编译并运行代码，现在你可以发射激光武器了。



**基本的碰撞检测**

恩，到目前为止，这看起来有点像一个游戏了，但是，还不够完整，因为没有爆炸！

而且我天性不听话，喜欢搞破坏，所以是时候往游戏里面添加一些破坏啦！：）

打开HelloWorldLayer.h，然后添加下面的实例变量：

int \_lives;

然后update方法的最后面添加下面代码：

for (CCSprite \*asteroid in \_asteroids) {

if (!asteroid.visible) continue;

for (CCSprite \*shipLaser in \_shipLasers) { if (!shipLaser.visible) continue;

if (CGRectIntersectsRect(shipLaser.boundingBox, asteroid.boundingBox)) { shipLaser.visible = NO; asteroid.visible = NO; continue; } }

if (CGRectIntersectsRect(\_ship.boundingBox, asteroid.boundingBox)) { asteroid.visible = NO; [\_ship runAction:[CCBlink actionWithDuration:1.0 blinks:9]]; \_lives–; } }

这里使用了最最简单的方式，只是判断两个精灵的边界矩形是否有交集。注意，边界部分可能有透明，而且边界矩形不能反应精灵实例的轮廓，所以最好的做法是使用前面介绍的box2d的方法。不过没关系，这个游戏，我们这样做就可以了。

对于更好的使用box2d的方法，请查看[《如何只使用box2d来做碰撞检测》](http://blog.devdiv.com/%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8box2d%E6%9D%A5%E5%81%9A%E7%A2%B0%E6%92%9E%E6%A3%80%E6%B5%8B%EF%BC%88%E4%B8%94%E4%BB%85%E7%94%A8%E6%9D%A5%E5%81%9A%E7%A2%B0%E6%92%9E%E6%A3%80.html)。

编译并运行代码，现在你会看到有东西爆炸啦！

**胜利/失败条件检测**

我们差不多快做完了—现在只需要往游戏中添加判断游戏胜利或者失败的条件就可以了。

我是这样考虑的，只要玩家存活了30秒，就是胜利;如果被陨石打中了3次，那么就是失败。

因此，在HelloWorldLayer.h中做如下修改：

// Add before @interface

typedef enum {

kEndReasonWin,

kEndReasonLose

} EndReason;

// Add inside @interface double \_gameOverTime; bool \_gameOver;

同时，相应地修改HelloWorldLayer.m：

// Add at end of init

\_lives = 3;

double curTime = CACurrentMediaTime();

\_gameOverTime = curTime + 30.0;

// Add at end of update loop if (\_lives <= 0) { [\_ship stopAllActions]; \_ship.visible = FALSE; [self endScene:kEndReasonLose]; } else if (curTime >= \_gameOverTime) { [self endScene:kEndReasonWin]; }

// Add new methods above update - (void)restartTapped:(id)sender { [[CCDirector sharedDirector] replaceScene:[CCTransitionZoomFlipX transitionWithDuration:0.5 scene:[HelloWorldLayer scene]]]; }

- (void)endScene:(EndReason)endReason {

if (\_gameOver) return; \_gameOver = true;

CGSize winSize = [CCDirector sharedDirector].winSize;

NSString \*message; if (endReason == kEndReasonWin) { message = @”You win!”; } else if (endReason == kEndReasonLose) { message = @”You lose!”; }

CCLabelBMFont \*label; if (UI\_USER\_INTERFACE\_IDIOM() == UIUserInterfaceIdiomPad) { label = [CCLabelBMFont labelWithString:message fntFile:@"Arial-hd.fnt"]; } else { label = [CCLabelBMFont labelWithString:message fntFile:@"Arial.fnt"]; } label.scale = 0.1; label.position = ccp(winSize.width/2, winSize.height \* 0.6); [self addChild:label];

CCLabelBMFont \*restartLabel; if (UI\_USER\_INTERFACE\_IDIOM() == UIUserInterfaceIdiomPad) { restartLabel = [CCLabelBMFont labelWithString:@"Restart" fntFile:@"Arial-hd.fnt"]; } else { restartLabel = [CCLabelBMFont labelWithString:@"Restart" fntFile:@"Arial.fnt"]; }

CCMenuItemLabel \*restartItem = [CCMenuItemLabel itemWithLabel:restartLabel target:self selector:@selector(restartTapped:)]; restartItem.scale = 0.1; restartItem.position = ccp(winSize.width/2, winSize.height \* 0.4);

CCMenu \*menu = [CCMenu menuWithItems:restartItem, nil]; menu.position = CGPointZero; [self addChild:menu];

[restartItem runAction:[CCScaleTo actionWithDuration:0.5 scale:1.0]]; [label runAction:[CCScaleTo actionWithDuration:0.5 scale:1.0]];

}

如果你不理解endScene方法的话，也没关系—那些代码是我过去用来快速地判断游戏胜利或失败的方式。

真正重要的是，你要理解其它部分的代码—在每一个update循环中，你只检测玩家是否胜利或失败，然后相应地调那个方法就可以了。

编译并运行代码，看看你会不会输？



**免费的音乐和音效**

你懂的，我怎么可能不给你们提供一些很棒的音效和音效呢？

之前，你已经把相关的音乐和音效加到工程里面来了，因此，我们只需要在HelloWorldLayer.m中做如下修改即可：

// Add to top of file

#import ”SimpleAudioEngine.h”

// Add to bottom of init [[SimpleAudioEngine sharedEngine] playBackgroundMusic:@”SpaceGame.caf” loop:YES]; [[SimpleAudioEngine sharedEngine] preloadEffect:@”explosion\_large.caf”]; [[SimpleAudioEngine sharedEngine] preloadEffect:@”laser\_ship.caf”];

// Add inside BOTH CGRectIntersectsRect tests [[SimpleAudioEngine sharedEngine] playEffect:@”explosion\_large.caf”];

// Add inside ccTouchBegan [[SimpleAudioEngine sharedEngine] playEffect:@”laser\_ship.caf”];

好了！编译并运行，这就是一个我们从头至尾一步步开发出来的完整的游戏啦！：）



**何去何从?**

这里有本项目的[完整源代码](http://d1xzuxjlafny7l.cloudfront.net/downloads/SpaceGame.zip)。

到目前为止，除了网络编程的教程没有翻译外，其它的cocos2d和box2d相关的教程全部翻译完了。：）以后，随着原作者的博客更新，我也会相应地更新。当然时间上肯定会落后不少，所以我希望大家没事的时候，还是多看看e文版。google，金山词霸等工具都用上，看着看着，就能看懂了。

最后还是那句话，本人水平有限，翻译不准确的地方，请见谅！

**如何使用GameCenter制作一个简单的多人游戏教程：第一部分 by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/3276/how-to-make-a-simple-multiplayer-game-with-game-center-tutorial-part-12>

程序截图：



我正在实验以一种新的方式来撰写教程–通过采纳你们的建议来写教程！

在网站右边，你会发现一个新的区域，在那里，你们可以通过投票的方式来决定下一篇教程是什么。（当然，在原作者网站上面，我这里没有啦：）

在第一次投票中，你们中的好多人说，想让我写一篇关于如何制作一个简单的多人在线游戏教程—现在满足你们的要求!

在这个2部分系列教程中，你将会学习到如何使用cocos2d和game center来做一个简单的2人联机玩的小游戏。

这个游戏非常简单，就是一只狗与一只猫在比赛跑步—你点屏幕点得越快就越容易赢得比赛！

这个教程假设你对于cocos2d的基础知识已经非常熟悉了。如果你对于cocos2d完全陌生的话，你可以需要先看看这个网站里的[其它教程](http://www.cnblogs.com/andyque/category/305876.html)。

注意：为了完整实践本系列教程，你必须注册iOS开发者，这样的话，你才可以激活Game Center。当然，你将至少需要一台物理设备（这样的话，你就可以运行一个程序在模拟器上面，另一个程序在你的设备上面啦）。最后，你将至少需要2个不同的Game Center帐号来测试（不用担心，你可以创建n个免费的帐号，只需要提供不同的邮件地址就ok了）

准备好了吗？出发！

**Getting Started**

这个教程将教你如何向一个简单的游戏里面添加matchmaking和多人在线支持。

因为游戏逻辑并不是本教程的重点，所以，我已经准备好了[一些代码](http://d1xzuxjlafny7l.cloudfront.net/downloads/CatRaceStarter.zip)，只是还没有联网功能。

下载上面链接中的代码，编译并运行，你将会看到下面的游戏截屏：



这个游戏非常简单，而且代码注释良好—你可以直接去研究代码，但是要确保你能够看懂每一行代码。

如果你们对些代码感兴趣的话，我可以再单独写一个教程来教大家如何从头开始构建一个这样的游戏。（如果你愿意，请到原作者的网站上面去投票！）

**激活Game Center:总览**

现在，你已经有了一个非常简单的可以玩的游戏了，但是，它很无聊，因为你老是自己跟自己玩！

如何使用Game Center的话，这个游戏会变得灰常有趣，因为你可以邀请朋友和你一起玩，或者使用matchmaking来随机查找线上的玩家。

但是，在你开始写任何Game Center代码之前，你需要做以下2件事情：

* 创建并设置一个App ID
* 在 iTunes Connect 里面注册你的app

让我们一步步来：

**创建和设置一个App ID**

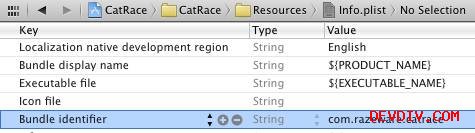
第一步就是创建并设置一个App ID。首先，你需要登录到 [iOS Dev Center](http://developer.apple.com/devcenter/ios/index.action)，然后进到 iOS Provisioning Portal。

在那里，选择App IDs标签，然后为你的应用程序创建一个App ID，和下面的图类似（需要你填写的值可能有差异）。



最重要的部分就是Bundle Identifier–你需要设置它为一个唯一的字符串（因此，你不能使用这个教程里我所使用过的！！！）最佳做法是使用你的域名来避免名字冲突。

一旦完成之后，点击Submit。然后打开Cat Race Xcode工程，选择Resources\Info.plist，然后输入你在iOS Provisioning portal里面输入的任何唯一的字符串，如下所示：（注意，要填写你自己的）



最后一件事，Xcode有时候会出问题，特别是在你更改了bundle identifier之后，因此，为了保证万无一失，你需要做以下3步：

* 删除你的模拟器或者设备上面的Cat Race程序
* 如果模拟器正在运行的话就退出来。
* 点Project\Clean来清理工程

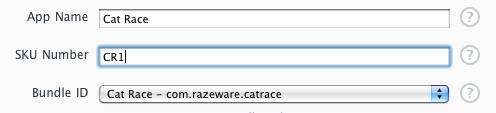
恭喜—现在你已经为你的应用程序创建App ID了，而且接下来会使用到它。下面，你可以通过ITunes Connect注册你的应用程序并激活Game Center。

**在Itunes Connect中注册你的程序**

接下来，登录到  [iTunes Connect](https://itunesconnect.apple.com/WebObjects/iTunesConnect.woa)并为你的应用创建一个新的入口（entry）。

一旦你登录入到iTunes Connect以后，选择Manage Your Applications，然后点左上角的蓝色的“Add New App”按钮。

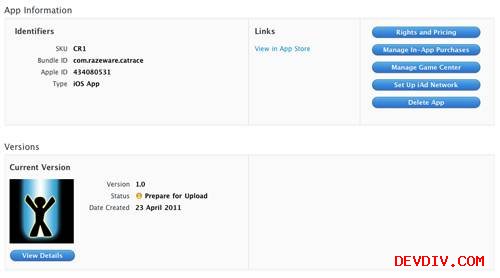
在出现的第一个屏幕中，在App Name中输入Cat Race，SKU Number中输入CRI，然后选择你之前创建的id，类似于下面的截屏：



点continue，并按照提示输入关于你的app的一些基本信息。

不用担心你填写的值对不对，尽管乱填，因为你之后还是可以改的—你只需要往里面添加一些傻瓜式的icon和screenshoot就可以让iTunes Connect很happy了。

当你做玩这些之后，点Save。如果一切ok，你将会到达“Prepare for Upload”阶段，如下图所示：



点右上角的蓝色的“Manage Game Center”按钮，然后点“Enable”按钮，再点“Done”。就这么多，你的app的Game Center功能已经激活了，接下来是时候写一些代码了。

顺便提一下，在“Manager Game Center”部分，你可能注意到了Leaderboards和Achievments等选项。不过，这篇教程中，我们并不会介绍它们，但是，如果你们对此感兴趣的话，你可以在我即将出版的[书](http://www.amazon.com/gp/product/0321735625/ref=as_li_tf_tl?ie=UTF8&tag=raywend-20&linkCode=as2&camp=217145&creative=399349&creativeASIN=0321735625)里找到。

**认证本地的用户：策略**

当你的游戏开始的时候，第一件事你需要做的就是认证本地玩家。

你可以把它看作是“把玩家添加进Game Center”。如果他已经登录了的话，那么会收到“Welcome Back！”消息。否则，它会要求玩家输入用户名和密码。

认证本地用户是非常容易的—你只需要调用 authenticateWithCompletionHandler就可以了。你可以选择性地传入一个block，当用户被认证身份以后就会回调这个block。（block是ios的新特性，ios3.0之前是不能使用的）

但是，这里有个技巧。还有另外一种方式让用户登录和登出。他可以先打开你的app，然后切换到Game Center app，从这里登录或登出，然后回到你的app。

因此，你的app在用户认证状态改变的时候应该得到通知。你可以注册一个“authentication changed” notification。

因此，我们的策略来认证用户的过程如下：

* 创建一个单例类来管理所有与Game Center相关的代码.
* 当单例对象创建的时候，它会注册“authentication changed” notification。
* 游戏将调用单例对象上的一个方法来认证用户。
* 不管什么时候用户被认证（或登出），“authentication changed”回调将会触发。
* 这个回调将会追踪用户当前是否被认证。

现在，你已经知道怎么做啦，让我们直接coding吧！

**认证本地用户：实现**

打开Cat Race项目，点  File\New\New File，然后选择  iOS\Cocoa Touch\Objective-C class，再点Next。选择NSObject作为基类，再点Next，把它命名为GCHelper，然后点Finish。

把GCHelper.h换成下面的形式：

#import <Foundation/Foundation.h>

#import <GameKit/GameKit.h>

@interface GCHelper : NSObject { BOOL gameCenterAvailable; BOOL userAuthenticated; }

@property (assign, readonly) BOOL gameCenterAvailable;

+ (GCHelper \*)sharedInstance; - (void)authenticateLocalUser;

@end

这里导入了GameKit头文件，然后定义了两个bool型的实例变量–一个用来追踪设备是否支持game center，还有一个用来追踪当前用户是否被认证。

我们也需要创建property，这样的话就可以直接查看game center是否可用。同时，还需要定义一个静态方法用来创建单例，还有一个认证本地用户的方法（这个方法会在app启动的时候被调用）

接下来，回到GCHelper.m，然后替换成下面的样子：

@synthesize gameCenterAvailable;

#pragma mark Initialization

static GCHelper \*sharedHelper = nil; + (GCHelper \*) sharedInstance { if (!sharedHelper) { sharedHelper = [[GCHelper alloc] init]; } return sharedHelper; }

这里synthesize gameCenterAvailable属性，然后定义了单例方法的实现。

注意，有很多方式可以实现单例方法，但是，我们这里使用了最简单，我们没有考虑多线程的情况。

接下来，在sharedInstance方法后面加入下列代码：

- (BOOL)isGameCenterAvailable {

// check for presence of GKLocalPlayer API

Class gcClass = (NSClassFromString(@”GKLocalPlayer”));

// check if the device is running iOS 4.1 or later NSString \*reqSysVer = @”4.1″; NSString \*currSysVer = [[UIDevice currentDevice] systemVersion]; BOOL osVersionSupported = ([currSysVer compare:reqSysVer options:NSNumericSearch] != NSOrderedAscending);

return (gcClass && osVersionSupported); }

这个方法是直接从苹果的 [Game Kit Programming Guide](http://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/GameKit_Guide/Introduction/Introduction.html%23//apple_ref/doc/uid/TP40008304-CH1-SW1)中copy过来的。它用来检测当前设备是否支持game center。

在使用game center之前必须要判断其是否可用，这和网络编程一样，没有网的情况一定要判断。同时，这个app只能运行ios4.0及其以后的系统上面。

接下来，在 isGameCenterAvailable方法后面添加下列代码：

- (id)init {

if ((self = [super init])) {

gameCenterAvailable = [self isGameCenterAvailable];

if (gameCenterAvailable) {

NSNotificationCenter \*nc =

[NSNotificationCenter defaultCenter];

[nc addObserver:self

selector:@selector(authenticationChanged)

name:GKPlayerAuthenticationDidChangeNotificationName

object:nil];

}

}

return self;

}

- (void)authenticationChanged {

if ([GKLocalPlayer localPlayer].isAuthenticated && !userAuthenticated) { NSLog(@”Authentication changed: player authenticated.”); userAuthenticated = TRUE; } else if (![GKLocalPlayer localPlayer].isAuthenticated && userAuthenticated) { NSLog(@”Authentication changed: player not authenticated”); userAuthenticated = FALSE; }

}

init方法检测Game Center是否可用，如果可用，则注册  “authentication changed” notification。（这是非常经典的观察者模式）。在尝试认证用户之前，注册这个通告灰常重要，这样，当认证完成的时候，它就会被调用。

这里的 authenticationChanged回调函数是很简单的–它只是简单地判断用户是否被认证，并且相应地更新标记变量。

注意，实际上，这个回调可能会被调用许多次，所以要确保 userAuthenticated和之前的状态不一样，只有当上一次状态改变的时候才更新。（具体理解参考代码）

最后，在 authenticationChanged方法后面添加下面方法：

#pragma mark User functions

- (void)authenticateLocalUser {

if (!gameCenterAvailable) return;

NSLog(@”Authenticating local user…”); if ([GKLocalPlayer localPlayer].authenticated == NO) { [[GKLocalPlayer localPlayer] authenticateWithCompletionHandler:nil]; } else { NSLog(@”Already authenticated!”); } }

这里调用前面提到过的 authenticateWithCompletionHandler来认证用户。注意，这里目前并没有传入一个完整的处理器（传入的是nil）。因为，前面你已经注意了  “authentication changed”notification了，所以这里就没有必要再写一个handler了。

好了–GCHelper现在包含认证用户所需的所有代码了，所以，你是时候使用它们了！回到AppDelegate.，然后做如下更改：

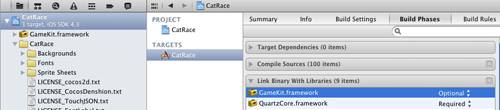
// At the top of the file

#import ”GCHelper.h”

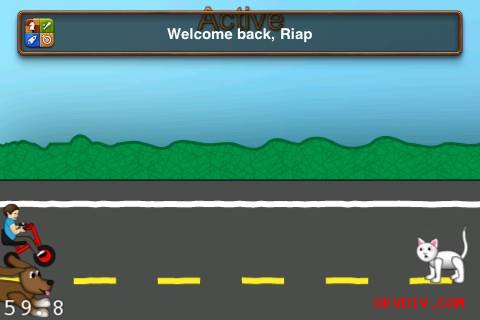
// At the end of applicationDidFinishLaunching, right before // the last line that calls runWithScene: [[GCHelper sharedInstance] authenticateLocalUser];

这里创建GCHelper的单例（它的init方法里面注册了 “authentication changed”通告）并调用 authenticateLocalUser 方法。

差不多快完成了！最后一步就是添加Game Kit framework到你的工程中。我们先点工程文件，然后选择Build Phases 标签，展开 “Link Binary with Libraries”，再点”＋“号来添加相应的framework。选择GameKit.framework并点Add。把type改成Required，然后看起来如下图所示：



就这么多！编译并运行工程，如果你登入Game Center，你将会看到下面的输出：



现在，你已经认证用户了，你可以进而探寻更有趣的部分了，比如找一个人与你共同来玩这个游戏！

**Matchmaker, Matchmaker, 给我找个对手吧！**

这里有两种方法可以通过Game Center来找一些人来一起玩游戏：一种是编程来主动查找，另一种是使用内置的matchmaking接口。

在这篇教程中，我们将使用内置的matchmaking接口。当你想要寻找一个对手的时候，只需要在 GKMatchRequest对象上面设置几个参数，然后再创建并显示一个 GKMatchmakerViewController就ok了。

让我们来看一看这个具体是怎么工作的。首先在GCHelper.h中做一些修改：

// Add to top of file

@protocol GCHelperDelegate

- (void)matchStarted;

- (void)matchEnded;

- (void)match:(GKMatch \*)match didReceiveData:(NSData \*)data

fromPlayer:(NSString \*)playerID;

@end

// Modify @interface line to support protocols as follows @interface GCHelper : NSObject <GKMatchmakerViewControllerDelegate, GKMatchDelegate> {

// Add inside @interface UIViewController \*presentingViewController; GKMatch \*match; BOOL matchStarted; id <GCHelperDelegate> delegate;

// Add after @interface @property (retain) UIViewController \*presentingViewController; @property (retain) GKMatch \*match; @property (assign) id <GCHelperDelegate> delegate;

- (void)findMatchWithMinPlayers:(int)minPlayers maxPlayers:(int)maxPlayers viewController:(UIViewController \*)viewController delegate:(id<GCHelperDelegate>)theDelegate;

这里出现了一些新的内容，让我们一点一点来看：

* 你定义了一个协议，叫做 GCHelperDelegate。当match开始，结束，或者从第三方接收到数据的时候就会通知其它对象，当然前提是那个对象要实现该协议。在本例中，cocos2d的layer将会实现此协议。
* 同时，GCHelper对象实现了两个协议。第一个是matchmaker进行玩家查找，不管有没有找到一个新的match，就会通知实现该协议的对象。第二个就是当数据到达或者连接状态改变的时候，Game Center会通知GCHelper对象。
* 创建一个新的实例变量和相应的属性来追踪view controlller对象（这个对象将会用来显示matchmaker用户界面），一个match对象的引用，match是否开始的标记以及一个代理。
* 创建一个新的方法，我们之后的cocos2d layer将会调用这个方法来查找可以一起玩游戏的玩家。

接下来跳转到GCHelper.m文件，然后做如下修改：

// At top of file

@synthesize presentingViewController;

@synthesize match;

@synthesize delegate;

// Add new method, right after authenticateLocalUser - (void)findMatchWithMinPlayers:(int)minPlayers maxPlayers:(int)maxPlayers viewController:(UIViewController \*)viewController delegate:(id<GCHelperDelegate>)theDelegate {

if (!gameCenterAvailable) return;

matchStarted = NO; self.match = nil; self.presentingViewController = viewController; delegate = theDelegate; [presentingViewController dismissModalViewControllerAnimated:NO];

GKMatchRequest \*request = [[[GKMatchRequest alloc] init] autorelease]; request.minPlayers = minPlayers; request.maxPlayers = maxPlayers;

GKMatchmakerViewController \*mmvc = [[[GKMatchmakerViewController alloc] initWithMatchRequest:request] autorelease]; mmvc.matchmakerDelegate = self;

[presentingViewController presentModalViewController:mmvc animated:YES];

}

这个cocos2d layer将要调用的方法的主要功能就是查找一个玩家。如果Game Center不可用的话，那么就什么也不干，直接返回。

它首先初始化match为未开始状态，并把match对象设置为nil。并且，存储视图控制器和代码，以便后面使用。同时，还要销毁前面已经出现的任何模态视图控制器（比如： GKMatchmakerViewController已经显示出来了）。

然后，我们来讲一下重点内容。 GKMatchRequest允许你配置你将要查找的match的类型，比如最小或者最大的玩家数量。这个方法比较灵活，你可以传递任何数量。但是，本游戏只需要设置最小和最大都为2就可以了。

接下来，我们使用给定的 request来创建一个 GKMatchmakerViewController类的实例，同时把代理设置为GCHelper对象，然后把它显示到屏幕上。

这时，GKMatchmakerViewController这个类对象的视图就开始接管工作了。它会允许用户查找一个随机的玩家来一起玩游戏。

接下来，我们需要定义一些代理方法：

#pragma mark GKMatchmakerViewControllerDelegate

// The user has cancelled matchmaking - (void)matchmakerViewControllerWasCancelled:(GKMatchmakerViewController \*)viewController { [presentingViewController dismissModalViewControllerAnimated:YES]; }

// Matchmaking has failed with an error - (void)matchmakerViewController:(GKMatchmakerViewController \*)viewController didFailWithError:(NSError \*)error { [presentingViewController dismissModalViewControllerAnimated:YES]; NSLog(@”Error finding match: %@”, error.localizedDescription); }

// A peer-to-peer match has been found, the game should start - (void)matchmakerViewController:(GKMatchmakerViewController \*)viewController didFindMatch:(GKMatch \*)theMatch { [presentingViewController dismissModalViewControllerAnimated:YES]; self.match = theMatch; match.delegate = self; if (!matchStarted && match.expectedPlayerCount == 0) { NSLog(@”Ready to start match!”); } }

如果用户取消查找match或者查找过程中出现了错误的话，那么我们需要关闭matchmaker 视图。

但是，如果找到一个match的话，我们需要隐藏此对象，并且设置match的delegate为GCHelper对象。这样的话，当有新的数据到达，或者连接状态改变的话，GCHelper对象就会得到通知。

同时，我们也需要检测是否可以开始match了。match对象保存了仍然需要多少个玩家才能完成连接，这个数目由“ expectedPlayerCount”来定。

如果这个变量是0的话，那么所有人都准备好了。当然，现在我们只是用NSLog输出一些语句，看是否执行到此处了。

接下来，添加 GKMatchDelegate回调函数实现：

#pragma mark GKMatchDelegate

// The match received data sent from the player. - (void)match:(GKMatch \*)theMatch didReceiveData:(NSData \*)data fromPlayer:(NSString \*)playerID { if (match != theMatch) return;

[delegate match:theMatch didReceiveData:data fromPlayer:playerID]; }

// The player state changed (eg. connected or disconnected) - (void)match:(GKMatch \*)theMatch player:(NSString \*)playerID didChangeState:(GKPlayerConnectionState)state { if (match != theMatch) return;

switch (state) { case GKPlayerStateConnected: // handle a new player connection. NSLog(@”Player connected!”);

if (!matchStarted && theMatch.expectedPlayerCount == 0) { NSLog(@”Ready to start match!”); }

break; case GKPlayerStateDisconnected: // a player just disconnected. NSLog(@”Player disconnected!”); matchStarted = NO; [delegate matchEnded]; break; } }

// The match was unable to connect with the player due to an error. - (void)match:(GKMatch \*)theMatch connectionWithPlayerFailed:(NSString \*)playerID withError:(NSError \*)error {

if (match != theMatch) return;

NSLog(@”Failed to connect to player with error: %@”, error.localizedDescription); matchStarted = NO; [delegate matchEnded]; }

// The match was unable to be established with any players due to an error. - (void)match:(GKMatch \*)theMatch didFailWithError:(NSError \*)error {

if (match != theMatch) return;

NSLog(@”Match failed with error: %@”, error.localizedDescription); matchStarted = NO; [delegate matchEnded]; }

match:didReceiveData:fromPlayer这个方法是在其他玩家给你发送数据的时候被调用的。这个方法只是简单的把这些数据再转发给它的代理类。（我们这个游戏中，cocos2d layer会实现此代码，所以这个代码是跟游戏需要相关的。

For match:player:didChangState这个方法，是当有玩家接入的时候，你需要检测是否所有的玩家都已经就绪了。同时，当有玩家断开连接的时候，这个方法也会被调用。

最后两个方法是发生错误的时候被调用。任何一种情形，都把match标记为已经结束了，同时通知delegate对象。

好了，我们现在写一些代码来建立一个match吧。首先从HelloWorldLayer中开始，打开HelloWorldLayer.h，并做如下修改：

// Add to top of file

#import ”GCHelper.h”

// Mark @interface as implementing GCHelperDelegate @interface HelloWorldLayer : CCLayer <GCHelperDelegate>

然后，跳转到HelloWorldLayer.m中做如下修改：

// Add to top of file

#import ”AppDelegate.h”

#import ”RootViewController.h”

// Add to bottom of init method, right after setGameState AppDelegate \* delegate = (AppDelegate \*) [UIApplication sharedApplication].delegate; [[GCHelper sharedInstance] findMatchWithMinPlayers:2 maxPlayers:2 viewController:delegate.viewController delegate:self];

// Add new methods to bottom of file #pragma mark GCHelperDelegate

- (void)matchStarted { CCLOG(@”Match started”); }

- (void)matchEnded { CCLOG(@”Match ended”); }

- (void)match:(GKMatch \*)match didReceiveData:(NSData \*)data fromPlayer:(NSString \*)playerID { CCLOG(@”Received data”); }

这里最重要的部分就是init方法。它从AppDelegate那里得到一个RootViewController，因为这个视图控制器将会显示出matchmaker界面。然后创建一个GCHelper对象来查找一个match。

剩下的部分仅仅是一些桩代码，简单的实现了GCHelper协议，同时在里面输出了一些语句。

最后一件事，默认情况下面，cocos2d 模板并没有在AppDelegate里面包含一个RootViewController的属性，因此你必须手动添加一个。跳转到AppDelegate.h文件，并添加下面的代码：

@property (nonatomic, retain) RootViewController \*viewController;

然后跳转到AppDelegate.m，synthesize之：

@synthesize viewController;

就这么多！编译并运行你的程序，现在你将看到matchmaker视图了，它看起来如下图所示：



现在，在另一个设备上运行你的程序。当然，你也可以一个运行在模拟器上面，一个运行在iphone上面。

注意：每一个设备上面需要使用一个不同的game center帐号，否则的话就不能工作。

**在两个设备上都点击“Play Now”，然后，过了一段时间后，matchmaker视图将会消失，接着你将会在控制台输出下面的语句：**

CatRace[16440:207] Authentication changed: player authenticated.

CatRace[16440:207] Player connected!

CatRace[16440:207] Ready to start match!

恭喜你！你已经在两台设备之间完成了一次match了。你正在制作一个网络游戏，知道吗？：）呵呵

**Landscape Orientation and GKMatchmakerViewController**

你可以已经注意到了，默认情况下 GKMatchmakerViewController显示的方向是竖的（portrait）。很明显，这不行，因此，cocos2d模板生成的程序是横版的。

幸运的是，你可以为 GKMatchmakerViewController写一个类别，让它强制只接收横版方向。

让我们实现这个，首先点击 File\New\New File，然后选择  iOS\Cocoa Touch\Objective-C class，再点Next。把NSObject作为基类，点Next，并把这个类取名为 GKMatchmakerViewController-LandscapeOnly.m，最后点击Finish。

把 GKMatchmakerViewController-LandscapeOnly.h 换成下面的代码：

#import <Foundation/Foundation.h>

#import <GameKit/GameKit.h>

@interface GKMatchmakerViewController(LandscapeOnly) - (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation; @end

然后把 GKMatchmakerViewController-LandscapeOnly.m相应的替换成下面的代码：

#import ”GKMatchmakerViewController-LandscapeOnly.h”

@implementation GKMatchmakerViewController (LandscapeOnly)

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation { return ( UIInterfaceOrientationIsLandscape( interfaceOrientation ) ); }

@end

大功告成！编译并运行程序，这时视图控制器显示为横版模式了：



**何去何从?**

这里有本教程的[完整源代码](http://d1xzuxjlafny7l.cloudfront.net/downloads/CatRacePart1.zip)。

在下部分教程中，我们将会涉及到如何在两台设备之间发送数据，同时把cat Vs kid包装成一个非常好玩的游戏！

译者的话：前段时间忙着考试，现在又忙着做项目，不好意思，很长一段时间没有更新了，大家见谅。同时，如果翻译过程中有什么明显的错误，请大家给我指出来，谢谢！

**如何使用GameCenter制作一个简单的多人游戏教程：第二部分 by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/3325/how-to-make-a-simple-multiplayer-game-with-game-center-tutorial-part-22>

教程截图：



这是本系列教程的第二部分，主要内容是关于如何使用Game Center matchmaking来制作一个简单的联机游戏。

在[上篇教程中](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8gamecenter%E5%88%B6%E4%BD%9C%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84%E5%A4%9A%E4%BA%BA%E6%B8%B8%E6%88%8F%E6%95%99%E7%A8%8B%EF%BC%9A.html)，你学会了如何为你的app激活Game Center，还有如何使用内置的 GKMatchmakerViewController来查找玩家。

在这篇教程中，你将学会如何查找玩家别名，如何在游戏里面收发数据以及如何支持玩家邀请功能。

在最后，你将会得到一个完整的，但是非常简单的网络游戏，使用的技术就是cocos2d和Game Center。你可以和你的朋友一起玩！

如果你还没有上个教程的工程的话，你可以先在[这里](http://d1xzuxjlafny7l.cloudfront.net/downloads/CatRacePart1.zip)下载。

**网络代码策略：挑选玩家**

在选择玩家之前，让我们先讨论一下我们将要采纳的网络代码相关策略，同时还有如何挑选玩家的策略。

在这个游戏里面，我们有两个玩家：玩家1（狗）和玩家2（小孩）。问题是，我们怎么决定谁充当狗的角色，谁充当小孩的角色？（呵呵，老外这里有点搞笑了，我也照样翻译了）。

策略就是，我们将会在两个玩家游戏启动的时候生成随机数，并且把这个数发给对方。如果哪方的数大，那么就当玩家1，另外一方自然就是玩家2.

但是，极少数情况下面，可能会生成两个一样的随机数，那样的话，我们只好再重试一遍罗。

不管哪一个玩家是palyer1，他都会获得一些“特权”。首先，player1这一方开始游戏的时候，给另一方发送一个消息。同时，player1还负责检查游戏什么时候结束，并且负责发送消息给另一方，告知”游戏结束了“。

换句话说，”player1“将充当服务器的角色。它是整个游戏中拥有最终发言权的那位！

**网络代码策略：玩家别名（player alias）**

因为我们随机决定哪个玩家是dog，哪个玩家是kid，所以我们需要有一种方式可以知道玩家具体属性哪个游戏角色。

对于这个游戏来说，我们将会在游戏角色的上面显示玩家的别名，这样的话，就可以区别出谁是谁了。

如果你还不知道玩家别名是什么，其实就是你在建立Game Center帐号的时候取的nickname啦。当一个match完成的时候，你并不会自动获得它，你需要手动地调用一个方法，然后Game Center会把名字返回给你。

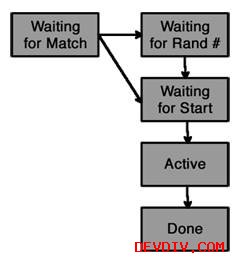
**网络代码策略：游戏状态**

编写网络相关代码的一个挑战就是，程序执行的顺序可能会和你预期的不一样。

比如，一方可能已经完成match的初始化工作了，然后开始发送随机数给另一端，但是，这时候，可能另一端并没有完成match的初始化啊！

因此，如果我们不够仔细的话，我们可以会遇到一些与时间相关的很奇怪的问题。一个比较好的解决方法就是，追踪每一端的游戏的当前状态。

下面一张图可以解释Cat Race这个游戏需要有哪些状态：



让我们一个状态一个状态来看：

* **Waiting for Match**: 游戏当前等待一个match被连接，同时查找对象玩家的别名。如果这两个都完成了的话，再检测是否从另一端收到随机数，然后再跳到等待游戏开始状态。
* **Waiting for Random #**:（＃号是number的意思）游戏里面有一个match和玩家别名，但是仍然需要从另一端那里接收到一个随机数。
* **Waiting for Start**: 游戏等待另一方开始游戏，本例中就是对应等待player2开始游戏。
* **Active**: 游戏正在进行中—还没有发现任何赢的玩家。每一个时刻，当一个玩家移动的时候，他都会给另一方发送一个消息，告诉他我在移动了。
* **Done**: 游戏结束 (player 1 给player2发送一个消息说，游戏结束了). 这时，再没有别的消息会发送了，然后两端就重新开始游戏。

好，现在你头脑中有一个概念了，让我们一步步来实现吧！

**查找玩家别名**

打开GCHelper.h，然后作如下修改：

// Add inside @interface

NSMutableDictionary \*playersDict;// Add after @interface

@property (retain) NSMutableDictionary \*playersDict;

这里定义了一个实例变量和相应的属性。我们使用一个字典类，这样可以方便地基于每个玩家的唯一的id来查找 GKPlayer数据（这里面包含了玩家别名信息）。

然后，打开GCHelper.m文件并作如下修改：

// Add to top of file

@synthesize playersDict;// Add new method after authenticationChanged

- (void)lookupPlayers {

NSLog(@”Looking up %d players…”, match.playerIDs.count); [GKPlayer loadPlayersForIdentifiers:match.playerIDs withCompletionHandler:^(NSArray \*players, NSError \*error) {

if (error != nil) { NSLog(@"Error retrieving player info: %@", error.localizedDescription); matchStarted = NO; [delegate matchEnded]; } else {

// Populate players dict self.playersDict = [NSMutableDictionary dictionaryWithCapacity:players.count]; for (GKPlayer \*player in players) { NSLog(@”Found player: %@”, player.alias); [playersDict setObject:player forKey:player.playerID]; }

// Notify delegate match can begin matchStarted = YES; [delegate matchStarted];

} }];

}

// Add inside matchmakerViewController:didFindMatch, right after @”Ready to start match!”: [self lookupPlayers];

// Add inside match:playerdidChangeState, right after @”Ready to start match!”: [self lookupPlayers];

主要的查找逻辑都在函数 lookupPlayers里面。这个函数在match准备好之后被调用，它会查找在一个match中的所有玩家的信息（除了本地玩家信息），因为我们可以通过 GKLocalPlayer单例类来查找本地玩家。

Game Center会在一个match中，为每一个玩家返回一个GKPlayer对象。为了使得后面使用更方便，我们把每一个GKPlayer对象放到字典里面，使用player id作为key。

最后，把match标记为已经开始了。然后调用游戏的delegate方法来开始整个游戏。

但是，在我们继续讲下去之前，测试一下！把两个设备都编译并运行一下，这一次，你再看一下控制台输出，你将会看到查找玩家及其别名的过程：

CatRace[16918:207] Authentication changed: player authenticated.

CatRace[16918:207] Player connected!

CatRace[16918:207] Ready to start match!

CatRace[16918:207] Looking up 1 players…

CatRace[16918:207] Found player: Vickipsq

CatRace[16918:207] Match started

**添加网络代码**

你已经建立好match了，同时还拥有每个玩家的名字，因此，现在你要开始学习这个项目真正有用的东西了—添加网络代码！

你需要做的第一件事情就是，基于我们前面画的图，定义一些新的游戏状态。打开HelloWorldLayer.h，然后修改GameState结构体，具体如下：

typedef enum {

kGameStateWaitingForMatch = 0,

kGameStateWaitingForRandomNumber,

kGameStateWaitingForStart,

kGameStateActive,

kGameStateDone

} GameState;

你也需要添加一个游戏结束的新的原因—断开连接。因此，修改EndReason枚举类型，如下所示：

typedef enum {

kEndReasonWin,

kEndReasonLose,

kEndReasonDisconnect

} EndReason;

接下来，你需要为每个消息定义相应的结构体类型。因此，在HelloWorldLayer.h里面添加下面的代码：

typedef enum {

kMessageTypeRandomNumber = 0,

kMessageTypeGameBegin,

kMessageTypeMove,

kMessageTypeGameOver

} MessageType;typedef struct {

MessageType messageType;

} Message;

typedef struct { Message message; uint32\_t randomNumber; } MessageRandomNumber;

typedef struct { Message message; } MessageGameBegin;

typedef struct { Message message; } MessageMove;

typedef struct { Message message; BOOL player1Won; } MessageGameOver;

注意，每一个消息开头都有一个消息类型—这样做的目的就是你可以通过比较相应的字段，来判别消息是什么类型的。

最后，在HelloWorldLayer类中添加一些实例变量：

uint32\_t ourRandom;

BOOL receivedRandom;

NSString \*otherPlayerID;

这里将追踪我们为每台设备产生的随机数，是否我们已经从另一端接收到了这个随机数，同时，我们还存储了玩家id的引用。

好了，现在，让我们来实现网络代码吧。打开HelloWorldLayer.m，然后修改matchStartd方法，如下所示：

- (void)matchStarted {

CCLOG(@”Match started”);

if (receivedRandom) {

[self setGameState:kGameStateWaitingForStart];

} else {

[self setGameState:kGameStateWaitingForRandomNumber];

}

[self sendRandomNumber];

[self tryStartGame];

}

因此，当match启动的时候，我们判断一下，是否已经收到另一端的随机数了，然后相应地设置游戏状态。然后我们调用发送随机数函数并开始游戏。

让我们开始实现 sendRandomNumber函数。在HelloWorldLayer.m中作如下修改：

// Add at bottom of init, anbd comment out previous call to setGameState

ourRandom = arc4random();

[self setGameState:kGameStateWaitingForMatch];// Add these new methods to the top of the file

- (void)sendData:(NSData \*)data {

NSError \*error;

BOOL success = [[GCHelper sharedInstance].match sendDataToAllPlayers:data withDataMode:GKMatchSendDataReliable error:&error];

if (!success) {

CCLOG(@”Error sending init packet”);

[self matchEnded];

}

}

- (void)sendRandomNumber {

MessageRandomNumber message; message.message.messageType = kMessageTypeRandomNumber; message.randomNumber = ourRandom; NSData \*data = [NSData dataWithBytes:&message length:sizeof(MessageRandomNumber)]; [self sendData:data]; }

sendRandomNumber创建一个新的 MessageRandomNumber结构体，设置结构体的 randomNumber域，然后把此结构体转换成DSData并发送给另一端。

sendData调用GCHelper的 sendDataToAllPlayers方法来给match对象里的所有玩家发送消息。

接下来，我们实现 tryStartGame方法。在HelloWorldLayer.m作如下修改：

// Add right after sendRandomNumber

- (void)sendGameBegin {MessageGameBegin message;

message.message.messageType = kMessageTypeGameBegin;

NSData \*data = [NSData dataWithBytes:&message length:sizeof(MessageGameBegin)];

[self sendData:data];

}

// Add right after update method - (void)tryStartGame {

if (isPlayer1 && gameState == kGameStateWaitingForStart) { [self setGameState:kGameStateActive]; [self sendGameBegin]; }

}

这个也非常简单—如果玩家1（就是有特权并且充当服务器的一端）和游戏都准备好了，那么就设置游戏状态为active，并且发送一个 MessageGameBegin消息给另一方。

好，现在让我们编写一些代码来处理这些接收到的消息。修改 match:didReceiveData:fromPlayer方法，如下所示：

- (void)match:(GKMatch \*)match didReceiveData:(NSData \*)data fromPlayer:(NSString \*)playerID {// Store away other player ID for later

if (otherPlayerID == nil) {

otherPlayerID = [playerID retain];

}

Message \*message = (Message \*) [data bytes]; if (message->messageType == kMessageTypeRandomNumber) {

MessageRandomNumber \* messageInit = (MessageRandomNumber \*) [data bytes]; CCLOG(@”Received random number: %ud, ours %ud”, messageInit->randomNumber, ourRandom); bool tie = false;

if (messageInit->randomNumber == ourRandom) { CCLOG(@”TIE!”); tie = true; ourRandom = arc4random(); [self sendRandomNumber]; } else if (ourRandom > messageInit->randomNumber) { CCLOG(@”We are player 1″); isPlayer1 = YES; } else { CCLOG(@”We are player 2″); isPlayer1 = NO; }

if (!tie) { receivedRandom = YES; if (gameState == kGameStateWaitingForRandomNumber) { [self setGameState:kGameStateWaitingForStart]; } [self tryStartGame]; }

} else if (message->messageType == kMessageTypeGameBegin) {

[self setGameState:kGameStateActive];

} else if (message->messageType == kMessageTypeMove) {

CCLOG(@”Received move”);

if (isPlayer1) { [player2 moveForward]; } else { [player1 moveForward]; } } else if (message->messageType == kMessageTypeGameOver) {

MessageGameOver \* messageGameOver = (MessageGameOver \*) [data bytes]; CCLOG(@”Received game over with player 1 won: %d”, messageGameOver->player1Won);

if (messageGameOver->player1Won) { [self endScene:kEndReasonLose]; } else { [self endScene:kEndReasonWin]; }

} }

这个方法把接收到的消息转换为我们定义的Message类型，然后我们就可以根据Message结构中的类型来作相应的处理了。

* 对于 **MessageRandomNumber** 这种消息,我们主要基于两个玩家所产生的随机数来进行比较，决定谁是1，谁是2，进而决定谁充当服务器的角色。同时，还有相应的状态改变。
* 对于 **MessageGameBegin** 这种消息,它仅仅是把游戏状态切换到active，这意味着玩家1将会发送一个消息给玩家2.
* 对于 **MessageMove** 这种消息, 它会使另一方向前移动一点点。
* 对于 **MessageGameOver** 这种消息, 它会基于游戏结束时的状态，以一种合适地方式来结束游戏。

好了，现在你差不多完成了大部分游戏逻辑了，只是，还有一些细节部分没有完成。接下来，我们在HelloWorldLayer.m中作如下修改：

// Modify setGameState as follows

// Adds debug labels for extra states

- (void)setGameState:(GameState)state {gameState = state;

if (gameState == kGameStateWaitingForMatch) {

[debugLabel setString:@"Waiting for match"];

} else if (gameState == kGameStateWaitingForRandomNumber) {

[debugLabel setString:@"Waiting for rand #"];

} else if (gameState == kGameStateWaitingForStart) {

[debugLabel setString:@"Waiting for start"];

} else if (gameState == kGameStateActive) {

[debugLabel setString:@"Active"];

} else if (gameState == kGameStateDone) {

[debugLabel setString:@"Done"];

}

}

// Add new methods after sendGameBegin // Adds methods to send move and game over messages - (void)sendMove {

MessageMove message; message.message.messageType = kMessageTypeMove; NSData \*data = [NSData dataWithBytes:&message length:sizeof(MessageMove)]; [self sendData:data];

}

- (void)sendGameOver:(BOOL)player1Won {

MessageGameOver message; message.message.messageType = kMessageTypeGameOver; message.player1Won = player1Won; NSData \*data = [NSData dataWithBytes:&message length:sizeof(MessageGameOver)]; [self sendData:data];

}

// Add to beginning of ccTouchesBegan:withEvent // Sends move message to other side when user taps, but only if game is active if (gameState != kGameStateActive) return; [self sendMove];

// Add to end of endScene: // If the game ends and it’s player 1, sends a message to the other side if (isPlayer1) { if (endReason == kEndReasonWin) { [self sendGameOver:true]; } else if (endReason == kEndReasonLose) { [self sendGameOver:false]; } }

// Add to beginning of update: // Makes it so only player 1 checks for game over conditions if (!isPlayer1) return;

// Add at bottom of matchEnded // Disconnects match and ends level [[GCHelper sharedInstance].match disconnect]; [GCHelper sharedInstance].match = nil; [self endScene:kEndReasonDisconnect];

// Add inside dealloc // Releases variable initialized earlier [otherPlayerID release]; otherPlayerID = nil;

上面的代码其实非常简单（你仔细阅读一下注释就知道它完成了哪些功能），所以，我们在这里就不在啰嗦了。

恩，写了好多代码了，不过可以跑起来！呵呵，编译并运行，两台设备都要，这时你有一个完整的赛跑游戏啦！



**显示玩家的名字**

目前为止，你已经有一个可以玩的游戏了，但是，你并不能分辨游戏里的角色身份，因为他们是随机决定的。

因此，让我们为每一个游戏角色定义玩家别名，这样子就可以区分开来了。打开HelloWorldLayer.h，添加下面的代码：

CCLabelBMFont \*player1Label;

CCLabelBMFont \*player2Label;

然后，打开HelloWorldLayer.m，然后在 tryStartGame方法后面添加一个新的方法，具体如下：

- (void)setupStringsWithOtherPlayerId:(NSString \*)playerID {if (isPlayer1) {

player1Label = [CCLabelBMFont labelWithString:[GKLocalPlayer localPlayer].alias fntFile:@”Arial.fnt”]; [self addChild:player1Label];

GKPlayer \*player = [[GCHelper sharedInstance].playersDict objectForKey:playerID]; player2Label = [CCLabelBMFont labelWithString:player.alias fntFile:@"Arial.fnt"]; [self addChild:player2Label];

} else {

player2Label = [CCLabelBMFont labelWithString:[GKLocalPlayer localPlayer].alias fntFile:@”Arial.fnt”]; [self addChild:player2Label];

GKPlayer \*player = [[GCHelper sharedInstance].playersDict objectForKey:playerID]; player1Label = [CCLabelBMFont labelWithString:player.alias fntFile:@"Arial.fnt"]; [self addChild:player1Label];

}

}

这里创建了两个 CCLabelBMFonts，每个游戏角色一个label。然后获得本地玩家的别名，我们可以使用 GKLocalPlayer来获取。对于另外一个玩家，我们需要查找GCHelper的字典里的GKPlayer对象。

接下来，在HelloWorldLayer.m作如下修改：

// Inside if statement for tryStartGame

[self setupStringsWithOtherPlayerId:otherPlayerID];// Inside if statement for match:didReceiveData, kMessageTypeGameBegin case

[self setupStringsWithOtherPlayerId:otherPlayerID];

// At beginning of update method player1Label.position = player1.position; player2Label.position = player2.position;

基本上，当游戏切找到Active状态的时候，我们肯定已经接收到玩家的名字了，而这时游戏已经快开始了，是时候初始化label了。

在update方法中，每一帧，我们根据玩家的位置来更新label的位置。

你可能会觉得这样实现有一点点奇怪–为什么不把label当作一个精灵的孩子呢？好吧，我们实际上不能这样做，因为，我们的精灵是加到一个batchNode里面的，而batchNode里面添加的孩子只能是ccsprite的子类，并且这个子类的孩子也要是ccsprite的子类）。因此，我们才这样实现的。

在两台设备上编译并运行代码，现在，你可以在每个游戏角色的上面看到玩家的别名了。



**支持邀请功能**

我们已经有一个非常好的，可以跑的游戏了，但是，让我们再添加一些更酷、更好玩的功能吧–支持邀请功能！

你可能已经注意到了，当matchmaker视图控制器出现的时候，它有一个选项，可以让你邀请你的朋友一起来玩。但是，目前为止，这个功能还不能用。因为我们还没有写任何代码，但是，实际上这个功能是非常容易实现的。

打开GCHelper.h，然后作如下修改：（代码添加位置，请注意看注释！）

// Add inside GCHelperDelegate

- (void)inviteReceived;// Add inside @interface

GKInvite \*pendingInvite;

NSArray \*pendingPlayersToInvite;

// Add after @interface @property (retain) GKInvite \*pendingInvite; @property (retain) NSArray \*pendingPlayersToInvite;

这里创建了一些实例变量，并定义了相应的属性。同时往 GCHelperDelegate 协议里面添加了一个新的方法，可以在邀请被接收到的时候被通告。

接下来，回到GCHelper.m，然后作如下修改：

// At top of file

@synthesize pendingInvite;

@synthesize pendingPlayersToInvite;// In authenticationChanged callback, right after userAuthenticated = TRUE

[GKMatchmaker sharedMatchmaker].inviteHandler = ^(GKInvite \*acceptedInvite, NSArray \*playersToInvite) {

NSLog(@”Received invite”); self.pendingInvite = acceptedInvite; self.pendingPlayersToInvite = playersToInvite; [delegate inviteReceived];

};

在Game Center里面，想让邀请功能跑起来的话，你需要提供一个回调函数，当invite被接收到的时候来回调之。你也应该尽可能的在你的游戏启动起来的时候就注册这样一个block函数—推荐的时机，是刚好在用户被认证之后。

对于这个游戏，这个回调函数会保存邀请信息，然后通知delegate对象。而delegate对象会重新切回cocos2d场景，然后通过调用 findMatchWithMinPlayers方法来查找一个match。—我们将会修改这个函数，并且把任何将到到达的邀请信息考虑进去。

因此，修改 findMatchWithMinPlayers方法，具体如下所示：

- (void)findMatchWithMinPlayers:(int)minPlayers maxPlayers:(int)maxPlayers viewController:(UIViewController \*)viewController delegate:(id<GCHelperDelegate>)theDelegate {if (!gameCenterAvailable) return;

matchStarted = NO; self.match = nil; self.presentingViewController = viewController; delegate = theDelegate;

if (pendingInvite != nil) {

[presentingViewController dismissModalViewControllerAnimated:NO]; GKMatchmakerViewController \*mmvc = [[[GKMatchmakerViewController alloc] initWithInvite:pendingInvite] autorelease]; mmvc.matchmakerDelegate = self; [presentingViewController presentModalViewController:mmvc animated:YES];

self.pendingInvite = nil; self.pendingPlayersToInvite = nil;

} else {

[presentingViewController dismissModalViewControllerAnimated:NO]; GKMatchRequest \*request = [[[GKMatchRequest alloc] init] autorelease]; request.minPlayers = minPlayers; request.maxPlayers = maxPlayers; request.playersToInvite = pendingPlayersToInvite;

GKMatchmakerViewController \*mmvc = [[[GKMatchmakerViewController alloc] initWithMatchRequest:request] autorelease]; mmvc.matchmakerDelegate = self;

[presentingViewController presentModalViewController:mmvc animated:YES];

self.pendingInvite = nil; self.pendingPlayersToInvite = nil;

}

}

当创建 GKMatchmakerViewController的时候，这里和之前的做法非常之类似，除了这里使用 pendingInvite和 pendingPlayersToInvite值。

最后，回到HelloWorldLayer.m并实现 inviteReceived方法，如下所示：

- (void)inviteReceived {

[self restartTapped:nil];

}

大功告成！在两台设备上编译并运行工程，当一个设备跑起来后，使用 GKMatchmakerViewController发送一个邀请给另一方。这时另一方将会收到邀请通知，这个通知看起来应该如下图所示：



点击”Accept“，然后在原来的设备上点“Play now”，这时，你又可以和之前一样玩这个游戏了。但是，这次，你的对手是你邀请的朋友呀！：）

**何去何从?**

这里有本教程的[完整源代码](http://d1xzuxjlafny7l.cloudfront.net/downloads/CatRacePart2.zip)。

如果你正想往你的游戏中添加Game Center的功能，你可能想使用Leaderboards和Achievements。如果你对此感兴趣的话，可以买我的[书](http://www.amazon.com/gp/product/0321735625/ref=as_li_tf_tl?ie=UTF8&tag=raywend-20&linkCode=as2&camp=217145&creative=399349&creativeASIN=0321735625)。

译者的话：本人水平有限，翻译不准的地方请不吝提出，谢谢！

**如何使用CCRenderTexture来创建动态纹理 by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/3857/how-to-create-dynamic-textures-with-ccrendertexture>

教程截图：



对于事先制作好的背景图片，我想你已经非常熟知如何把它们添加进游戏里面了。但是，你有没有想过动态地创建背景图片并且修改它们的颜色，梯度（gradients）及实时改变效果呢？

如果你玩过App Store 上面，由[Andreas Illiger](http://www.andreasilliger.com/)制作的[Tiny Wings](http://itunes.apple.com/us/app/tiny-wings/id417817520?mt=8)游戏的话，实际上你已经体验过上面提及的想法了。

在这个系列的教程中，你将会学习到所有这些技术要点：

* 如何用代码实时动态创建纹理
* 如何使用Gimp创建无缝拼接的纹理
* 如何混合阴影和光照来增强现实效果
* 如何创建带状纹理
* 如何制作重复纹理
* 还有更多！

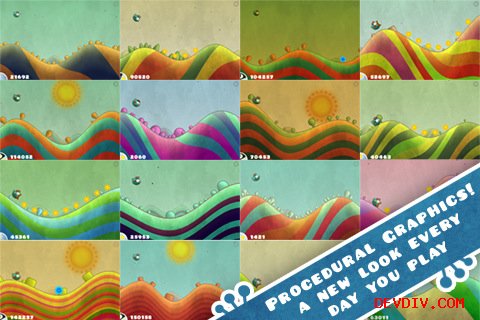
这个教程是基于[Sergey Tikhonov](http://haqu.net/)在cocos2d论坛里面讨论“[如何制作Tiny Wings](http://www.cocos2d-iphone.org/forum/topic/14136)”时所制作的一个样例工程来制作的。

Sergey在demo工程上面做了一件非常好的工作，因此，为了不重复发明轮子，我将把他的代码转换成一个系列的教程，同时还添加了一些很酷的特性。

这个教程假设你对于cocos2d已经非常熟悉了。如果你对cocos2d还很陌生的话，建议你先看看我博客上面的其它[cocos2d 教程](http://blog.devdiv.com/fengliu)。

**使用CCRenderTexture来动态创建纹理**

Tiny Wings有一个很酷的特性就是，每一天都会改变纹理，就如同下面的游戏截屏一下：



但是，你怎么在cocos2d里面来动态创建纹理呢？好吧，其实有一个很酷的类，叫CCRenderTexture，它允许你来动态创建纹理，并且可以在游戏中重用这些纹理。

使用 CCRenderTexture非常简单 – 你只需要做以下5步就行了:

* **创建一个新的CCRenderTexture.** 这里，你可以指定将要创建的纹理的宽度和高度。.
* **调用 CCRenderTexture:begin.** 这个方法会启动OpenGL，并且接下来，任何绘图的命令都会渲染到CCRenderTexture里面去，而不是画到屏幕上。
* **绘制纹理.** 你可以使用原始的OpenGL调用来绘图，或者你也可以使用cocos2d对象里面已经定义好的visit方法。（这个visit方法就会调用一些opengl命令来绘制cocos2d对象）
* **调用 CCRenderTexture:end**. 这个方法会渲染纹理，并且会关闭渲染至CCRenderTexture的通道。
* **从生成的纹理中创建一个sprite.** 你现在可以用CCRenderTexture的sprite.texture属性来轻松创建新的精灵了。

让我们来实践一下具体这个过程是如何工作的，我们先来创建一个简单的单色的纹理。

但是，首先，你需要一个新的工程！因此，打开Xcode，点击File\New\New Project，然后选择 iOS\cocos2d\cocos2d\_box2d template。尽管这个教程并没有使用到box2d，但是，我们后续的教程会用到，所以，我在这里就提前先建立好。

把工程命名为TinySeal，点击Next，然后选择磁盘上的一个文件夹并点击Create。

然后，打开HelloWorldLayer.h，并用以下内容替换之：

#import ”cocos2d.h”@interface HelloWorldLayer : CCLayer

{

CCSprite \* \_background;

}

+(CCScene \*) scene;

@end

这里移除掉了”Hello,World“box2d的代码，并且往里面添加了一个实例变量来追踪我们将要动态创建的背景。

接下来，打开HelloWorldLayer.m，并把它替换成下面的样子：（只需要把box2d相关代码全部删除并且得到一个空的场景就可以了）

#import ”HelloWorldLayer.h”@implementation HelloWorldLayer

+(CCScene \*) scene { CCScene \*scene = [CCScene node]; HelloWorldLayer \*layer = [HelloWorldLayer node]; [scene addChild: layer]; return scene; }

-(id) init { if((self=[super init])) { } return self; } @end

你现在可以编译并运行一下，你将会看到一个黑色的屏幕。

接下来，在init方法的上面添加下面的新方法：

-(CCSprite \*)spriteWithColor:(ccColor4F)bgColor textureSize:(float)textureSize {// 1: Create new CCRenderTexture

CCRenderTexture \*rt = [CCRenderTexture renderTextureWithWidth:textureSize height:textureSize];

// 2: Call CCRenderTexture:begin [rt beginWithClear:bgColor.r g:bgColor.g b:bgColor.b a:bgColor.a];

// 3: Draw into the texture // We’ll add this later

// 4: Call CCRenderTexture:end [rt end];

// 5: Create a new Sprite from the texture return [CCSprite spriteWithTexture:rt.sprite.texture];

}

你可以看到，这里动态创建纹理所采用的五个步骤和我们之前讨论的一模一样。

注意，我们这里不是调用的CCRenderTexture:begin方法，而是调用另外一个较方便的方法beginWithClear:g:b:a:，这个方法可以用给定的颜色来清除纹理的背景，相当于设置画布的颜色。

我们还没有在画布上画任何东西–目前为止，我们仅仅是看看纯色纹理看起来是什么样子.

把上面所讨论的一起打包起来，并且添加一些新的方法，然后把init方法改成下面的样子：

- (ccColor4F)randomBrightColor {while (true) {

float requiredBrightness = 192;

ccColor4B randomColor =

ccc4(arc4random() % 255,

arc4random() % 255,

arc4random() % 255,

255);

if (randomColor.r > requiredBrightness ||

randomColor.g > requiredBrightness ||

randomColor.b > requiredBrightness) {

return ccc4FFromccc4B(randomColor);

}

}

}

- (void)genBackground {

[\_background removeFromParentAndCleanup:YES];

ccColor4F bgColor = [self randomBrightColor]; \_background = [self spriteWithColor:bgColor textureSize:512];

CGSize winSize = [CCDirector sharedDirector].winSize; \_background.position = ccp(winSize.width/2, winSize.height/2); [self addChild:\_background z:-1];

}

-(id) init { if((self=[super init])) { [self genBackground]; self.isTouchEnabled = YES; } return self; }

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

[self genBackground];

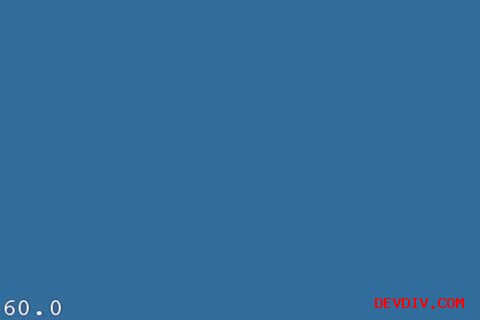
}

randomBrightColor方法是一个辅助方法，用来创建一种随机颜色。注意，这里使用ccc4B（因此，我们能够在0-255的范围内指定R/G/B/A值），同时确保至少有一个颜色分量是大于192的，这样的话，我们就不会得到较暗的颜色。

然后，genBackground调用我们之前写的spriteWithColor方法，同时把它加屏幕中央。

至于init函数，它调用genBackground方法，同时激活touches事件，这样的话，你就可以通过点击屏幕来获得另外的随机背景了。

编译并运行，这样你每一次点击屏幕，你都可以得到一张不同的单色背景图片啦！



**往纹理里面加点噪声**

在Tiny Wings里面，你可能已经注意到了，那纹理并不只是纯粹的单色—而是被一些噪声所修饰了一下，这样，看起来的话就会有阴影和光照。

你可以用代码来动态地生成了一些噪声，但是，如果使用一些预先制作的噪声的话，那样速度会更快一些–这也是我接下来将要在教程里所使用的方法：

一种比较简单的方法，就是使用一个免费的图片处理程序，叫做[Gimp](http://www.cnblogs.cc2/TODO)。接下来这一部分将向您展示如何动手制作这样一个噪声纹理，但是，你得喜欢下面这个人。

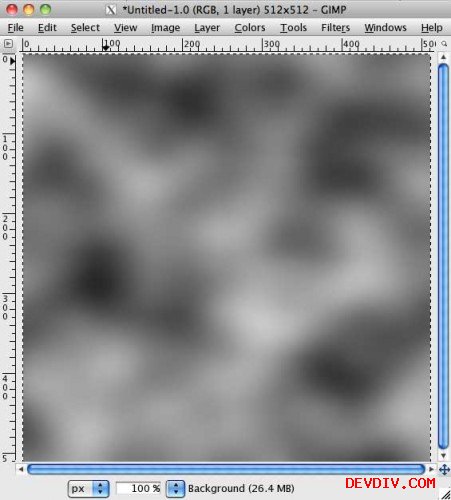


当然，你也可以下载我已经做好了的[噪声图片](http://www.raywenderlich.com/downloads/Noise.png)，并且把它添加到工程里面去，同时可以跳过接下来的部分了：）

如果你想继续，那么先下载Gimp，并且安装好。

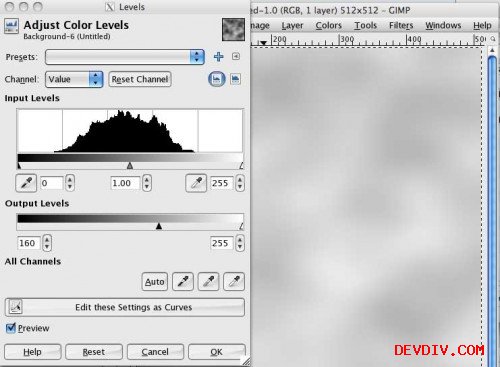
安装好之后，打开Gimp，选择 File\New，然后创建一张新的512\*512的图片。然后选择 Filter\Render\Clouds\Solid Noise，并且可以调节一些参数来看看效果，调好之后就点击Ok吧。

你可以会得到和下图差不多的图片：



我们将使用这张图片去乘以纹理的颜色。因此，噪声图片中白色的部分，纹理中的颜色就会被显示出来，而噪声图片中黑色的地方，纹理图片对应部位的颜色就会被变暗。

上图中可以看出，黑色部分占太多了，这样会影响我们的原图效果。因此，我们要把黑色的部位去掉一些，选择 Colors\Levels，然后”Output Levels“的最左边的滑动条，把它拖到右边去一点，这样我们的噪声图片就会明亮一些啦。



当你做完之后就点击ok，然后，还有最后一步，这个噪声图片纹理要制作成无缝拼接的，因此，这样我们才能制作无限滚动的背景。

Gimp使用得这个功能变得非常简单。只需要点击Filters\Map\Make Seamless就行了！

使用File\Save As来保存你的图片为Noise.png。然后使用Finder打开并拖到TinySeals工程中去。确保“Copy items into destination group’s folder”被复选中，然后点击Finish。

恭喜你，现在你有一张噪声纹理了，位下来，你可以使用它来创建更加真实的纹理效果了！

现在，你知道怎么动态创建纹理了，并且你可以使用Gimp的滤镜来制作一些不同的效果！

**往纹理中添加噪声**

现在，你有一张带噪声的图片了，接下来，让我们把它添加到CCRenderTexture里去吧。

在spriteWithColor:textureSize方法里面，我们在注释中所示的第3步，添加下列代码：

CCSprite \*noise = [CCSprite spriteWithFile:@"Noise.png"];

[noise setBlendFunc:(ccBlendFunc){GL\_DST\_COLOR, GL\_ZERO}];

noise.position = ccp(textureSize/2, textureSize/2);

[noise visit];

这里使用噪声纹理来创建一个精灵，然后把它放置在render texture的中心位置，并且调用visit方法。这个visit方法就是一系列的opengl调用，且这些调用是用来画这个纹理的。

这里只有一个诡异的地方—-就是”setBlendFunc“方法。这玩意儿倒底干吗的呢？

好吧，这第一个常量（GL\_DST\_COLOR）指定如何乘以输入图像/源图像的颜色数据（也就是指噪声图像），第二个参数（GL\_ZERO）指定如何乘以已经存在的图像/目标图像颜色数据（实际上就是之前的单色纹理）。

因此，阐述一下：

* 已经存在的颜色乘以GL\_ZERO，那意味着已经存在的颜色将会被清除掉。
* 噪声纹理颜色乘以GL\_DST\_COLOR。而GL\_DST\_COLOR意味着已经存在的颜色，因此噪声颜色会乘以已经存在的颜色。所以，如果噪声中白色越多，那么对应位置的源图像位置显示的颜色也就越亮。如果噪声颜色中越多黑色，那么源图像的颜色就会变得越暗。
* 上面提及的两种颜色叠加到一起，但是，由于指点一个参数是0，所以，实际上还是看第二个参数的影响。

顺便提一下，我发现这些混合常量非常容易使人迷惑，但是，我们很幸运，这里有[一款免费的在线工具](http://www.andersriggelsen.dk/OpenGL/)，可以用来动态地配置每种常量的效果。

不管怎么说，这就是你所需要的全部东西啦！编译并运行你的代码，现在，你可以看到纹理的一些阴影效果了。注意，这个在模拟器上看起来效果通常不是很好，你可能需要在实际的设备上尝试一下会比较好。



**往纹理中添加梯度**

为了使纹理看起来更好看，让我们从上至下添加一些梯度吧，具体的效果是纹理从上到下，会变得越来越暗。

我们也可以使用Gimp来修改我们的噪声图片，但是，这一次，我们可以在代码中来完成，这样会变得更加动态，并且可以很容易地修改。

基本的想法就是，我们在纹理的顶部画一个黑色的矩形，但是，它将会是上部完全透明的，并且底部完全不透明。这样会使得上面比较亮，而慢慢得往下变暗。

为了实现这个目标，我们需要使用一些opengl命令。如果你对opengl不熟悉的话，也不用担心—我会帮你写好这些代码，并且在高层向你解释具体是如何工作的。在教程的结尾，你会得到一些参考资料，通过它们你可以获得更多的信息。

在spriteWithColor:textureSize里面，在创建噪声精灵之前，添加下列代码：

glDisable(GL\_TEXTURE\_2D);

glDisableClientState(GL\_TEXTURE\_COORD\_ARRAY);float gradientAlpha = 0.7;

CGPoint vertices[4];

ccColor4F colors[4];

int nVertices = 0;

vertices[nVertices] = CGPointMake(0, 0); colors[nVertices++] = (ccColor4F){0, 0, 0, 0 }; vertices[nVertices] = CGPointMake(textureSize, 0); colors[nVertices++] = (ccColor4F){0, 0, 0, 0}; vertices[nVertices] = CGPointMake(0, textureSize); colors[nVertices++] = (ccColor4F){0, 0, 0, gradientAlpha}; vertices[nVertices] = CGPointMake(textureSize, textureSize); colors[nVertices++] = (ccColor4F){0, 0, 0, gradientAlpha};

glVertexPointer(2, GL\_FLOAT, 0, vertices); glColorPointer(4, GL\_FLOAT, 0, colors); glDrawArrays(GL\_TRIANGLE\_STRIP, 0, (GLsizei)nVertices);

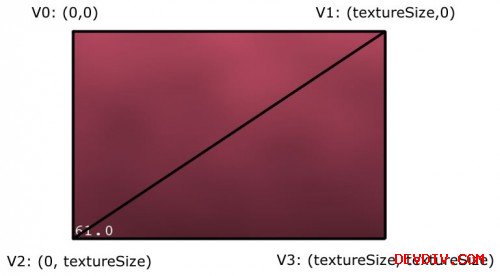
glEnableClientState(GL\_TEXTURE\_COORD\_ARRAY); glEnable(GL\_TEXTURE\_2D);

第一件事情就是禁用GL\_TEXTURE\_2D 和 GL\_TEXTURE\_COORD\_ARRA两个opengl状态，因为，我们接下来只需要绘制颜色—这与纹理无关。

一个比较奇怪的事就是，这里绘制纹理的时候，左上角是坐标原点（0，0）—而不是我们在cocos2d里面使用的，0,0点在左下角。

因此，接下来，我们定义纹理的四个顶点坐标左上、右上、左下、右下–并且同时指定每一个点的颜色值。

你可能会奇怪，为什么顶点会以这样的顺序被定义出来。但是因为，我们将按下图的方式来画两个三角形，当然这两个三角形拼成了一个矩形：



我们的将使用GL\_TRIANGLE\_STRIP来画这些三角形，那意味着，顶点数组中的前三个顶点组成第一个三角形，而第二个三角形就是前面2，3号顶点，再加上下一个新的顶点组成第二个三角形。

因此，第一个三角形是由顶点v0,v1,v2组成，而第二个则由v1,v2,v3组成。

在定义完顶点数组和颜色数组以后，我们把它们传给glVertexPointer opengl调用（每一个顶点用2个浮点数标识）和glColorPointer调用（每个颜色值指定4个浮点数），然后调用glDrawArrays，并且指定 GL\_TRIANGLE\_STRIP为参数来绘制三角形带。

最后，我们重新激活GL\_TEXTURE\_COORD\_ARRAY 和 GL\_TEXTURE\_2D两个状态，这样做的话是保持和之前调用的对称，并且不会破坏opengl状态机。

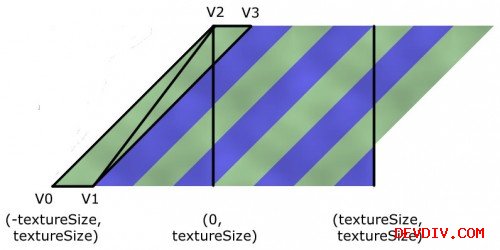
编译并运行代码，然后你会看到一个整齐的带有梯度的纹理啦！



**创建带状纹理**

在我们开始写代码来绘制带状纹理之前，让我们先花上一分钟时间来想一想我们的策略。

我们将从指定纹理为一种颜色开始（如下图的蓝色部分），然后，我们将以对角线的方向画一些三角带子（如下图绿色部分），如图所示：



注意，因为带子是呈对角线的，我们实际上将要在纹理的边界之外就绘制这条带子。（如上图v0,v1,v2所示）

还有一点需要注意，为了获得一个比较好的45度的效果，我们将把V0偏移原点texture size的距离。

好了，让我们来看看实际的代码是什么样子的吧。在init的方法上面添加一个新的方法，如下所示：

-(CCSprite \*)stripedSpriteWithColor1:(ccColor4F)c1 color2:(ccColor4F)c2 textureSize:(float)textureSize stripes:(int)nStripes {// 1: Create new CCRenderTexture

CCRenderTexture \*rt = [CCRenderTexture renderTextureWithWidth:textureSize height:textureSize];

// 2: Call CCRenderTexture:begin [rt beginWithClear:c1.r g:c1.g b:c1.b a:c1.a];

// 3: Draw into the texture

// Layer 1: Stripes glDisable(GL\_TEXTURE\_2D); glDisableClientState(GL\_TEXTURE\_COORD\_ARRAY); glDisableClientState(GL\_COLOR\_ARRAY);

CGPoint vertices[nStripes\*6]; int nVertices = 0; float x1 = -textureSize; float x2; float y1 = textureSize; float y2 = 0; float dx = textureSize / nStripes \* 2; float stripeWidth = dx/2; for (int i=0; i<nStripes; i++) { x2 = x1 + textureSize; vertices[nVertices++] = CGPointMake(x1, y1); vertices[nVertices++] = CGPointMake(x1+stripeWidth, y1); vertices[nVertices++] = CGPointMake(x2, y2); vertices[nVertices++] = vertices[nVertices-2]; vertices[nVertices++] = vertices[nVertices-2]; vertices[nVertices++] = CGPointMake(x2+stripeWidth, y2); x1 += dx; }

glColor4f(c2.r, c2.g, c2.b, c2.a); glVertexPointer(2, GL\_FLOAT, 0, vertices); glDrawArrays(GL\_TRIANGLES, 0, (GLsizei)nVertices);

glEnableClientState(GL\_COLOR\_ARRAY); glEnableClientState(GL\_TEXTURE\_COORD\_ARRAY); glEnable(GL\_TEXTURE\_2D);

// Layer 2: Noise CCSprite \*noise = [CCSprite spriteWithFile:@"Noise.png"]; [noise setBlendFunc:(ccBlendFunc){GL\_DST\_COLOR, GL\_ZERO}]; noise.position = ccp(textureSize/2, textureSize/2); [noise visit];

// 4: Call CCRenderTexture:end [rt end];

// 5: Create a new Sprite from the texture return [CCSprite spriteWithTexture:rt.sprite.texture];

}

这个方法的大部分内容都是重温如何创建一个CCRenderTexture，只有创建带状层的代码是新加的。所以让我们来讨论一下。

它首先创建了一个顶点数组–对于每一条带子，我们需要6个顶点–3个顶点\*2个三角形。我们这里不能使用三角形带，因为，带子不是邻接在一起的。

第一个顶点的坐标是 (-textureSize, textureSize)，上面的图示中也可以看出来，接下来一个顶点就是 (-textureSize+stripWidth, textureSize).第三个顶点是（0，0），而第四个顶点是(stripeWidth, textureSize).这些就是我们画一个带子所需要的全部顶点，而且每一次我们往前递增两个带子的长度，并且继续，直至所有的带子都制作完成。

现在，你可以把genBackground修改成下面的样子了：

- (void)genBackground {[\_background removeFromParentAndCleanup:YES];

ccColor4F bgColor = [self randomBrightColor]; ccColor4F color2 = [self randomBrightColor]; //\_background = [self spriteWithColor:bgColor textureSize:512]; int nStripes = ((arc4random() % 4) + 1) \* 2; \_background = [self stripedSpriteWithColor1:bgColor color2:color2 textureSize:512 stripes:nStripes];

self.scale = 0.5;

CGSize winSize = [CCDirector sharedDirector].winSize; \_background.position = ccp(winSize.width/2, winSize.height/2); [self addChild:\_background];

}

这里调用新的方法，同时修改层的scale为0.5，这样的话就能看到512\*512的纹理的全貌。

编译并运行，现在，你每点击一下屏幕都可以得到一张不同的带状的纹理了！



**重复背景**

对于带状背景和梯度背景，我们都希望它比实际定义的512\*512的区域要大。

一种比较简单的方式就是我们创建多个CCSprites，然后把它们链接在一起。但是，那看起来太疯狂了，因为，我们有更加简单的方法—我们可以设置纹理并让它们重复。

打开HelloWorldLayer.m，并作如下修改：

// Add to genBackground, right BEFORE the call to addChild

ccTexParams tp = {GL\_LINEAR, GL\_LINEAR, GL\_REPEAT, GL\_REPEAT};

[\_background.texture setTexParameters:&tp];// Add to bottom of init

[self scheduleUpdate];

// Add after init - (void)update:(ccTime)dt {

float PIXELS\_PER\_SECOND = 100; static float offset = 0; offset += PIXELS\_PER\_SECOND \* dt;

CGSize textureSize = \_background.textureRect.size; [\_background setTextureRect:CGRectMake(offset, 0, textureSize.width, textureSize.height)];

}

这里最重要的就是纹理参数:

* GL\_LINEAR 是说“当显示纹理时，显示的大小大于或者小于原纹理的尺寸时，使用邻近像素点来插值补点”
* GL\_REPEAT是说“如果你想显示纹理边界之外的像素点时，把它旁边的纹理像素点平铺过去”

同时，调用一个update定时器，更新纹理的x轴坐标，使前不断地前移。这样会使得纹理看起来随着时间不断地往前移动。

编译并运行代码，这时你可以看到一个连续滚动的无限重复的背景纹理了！你也可以用带有梯度的纹理来尝试一下，一样可以跑起来。

**免费的光照**

如果你看一看Tiny Wings的山丘纹理实现，你会发现山的顶部有一些光照，同时从上到下还有一些梯度，并且这一切看起来非常棒！因此， 让我们修改我们前面的带状纹理。

在stripedSpriteWithColor1方法中，紧接着glDrawArrays之后，添加下面代码：

// layer 2: gradient

glEnableClientState(GL\_COLOR\_ARRAY);float gradientAlpha = 0.7;

ccColor4F colors[4];

nVertices = 0;

vertices[nVertices] = CGPointMake(0, 0); colors[nVertices++] = (ccColor4F){0, 0, 0, 0 }; vertices[nVertices] = CGPointMake(textureSize, 0); colors[nVertices++] = (ccColor4F){0, 0, 0, 0}; vertices[nVertices] = CGPointMake(0, textureSize); colors[nVertices++] = (ccColor4F){0, 0, 0, gradientAlpha}; vertices[nVertices] = CGPointMake(textureSize, textureSize); colors[nVertices++] = (ccColor4F){0, 0, 0, gradientAlpha};

glVertexPointer(2, GL\_FLOAT, 0, vertices); glColorPointer(4, GL\_FLOAT, 0, colors); glDrawArrays(GL\_TRIANGLE\_STRIP, 0, (GLsizei)nVertices);

// layer 3: top highlight float borderWidth = textureSize/16; float borderAlpha = 0.3f; nVertices = 0;

vertices[nVertices] = CGPointMake(0, 0); colors[nVertices++] = (ccColor4F){1, 1, 1, borderAlpha}; vertices[nVertices] = CGPointMake(textureSize, 0); colors[nVertices++] = (ccColor4F){1, 1, 1, borderAlpha};

vertices[nVertices] = CGPointMake(0, borderWidth); colors[nVertices++] = (ccColor4F){0, 0, 0, 0}; vertices[nVertices] = CGPointMake(textureSize, borderWidth); colors[nVertices++] = (ccColor4F){0, 0, 0, 0};

glVertexPointer(2, GL\_FLOAT, 0, vertices); glColorPointer(4, GL\_FLOAT, 0, colors); glBlendFunc(GL\_DST\_COLOR, GL\_ONE\_MINUS\_SRC\_ALPHA); glDrawArrays(GL\_TRIANGLE\_STRIP, 0, (GLsizei)nVertices);

这段代码应该又是重新温习一遍啦，但是，第一部分和我们之前创建的梯度背景差不多，而第二部分创建光照的时候，我们从上至下添加，使之有太阳晒着的感觉。

编译并运行代码，现在，你应该发现我们的带状纹理更加逼真了，有梯度变化又有光照效果！



**何去何从?**

这里有本教程的[样例工程](http://www.raywenderlich.com/downloads/TinySeal1.zip)。

如果你愿意的话，你可以尝试着修改一下噪声纹理，看一看你能不能创建出更酷的效果出来？或者，你可以修改一下上面的代码，调节调节参数，看看能不能得到更好的效果。如果你实践了，请你在评论部分分享你的结果。

接下，我们将使用这些动态创建的纹理来制作一个类似Tiny Wings的游戏。请看本系列[下一篇教程](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E5%88%B6%E4%BD%9C%E4%B8%80%E4%B8%AA%E7%B1%BB%E4%BC%BCtiny-wings%E7%9A%84%E6%B8%B8%E6%88%8F%EF%BC%9A%E7%AC%AC%E4%B8%80%E9%83%A8%E5%88%86.html)。

**如何制作一个类似tiny wings的游戏：第二部分（完） by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/3913/how-to-create-a-game-like-tiny-wings-part-2>

教程截图：



这是本系列教程的最后一部分，主要是教大家如何制作一个类似[Tiny Wings](http://itunes.apple.com/us/app/tiny-wings/id417817520?mt=8)的游戏。

在[预备教程](http://www.cnblogs.com/andyque/archive/2011/07/01/2095479.html)中，我们学会了如何创建动态山丘纹理和背景纹理。

在[第一部分](http://blog.devdiv.com/%EF%BC%88%E8%AF%91%EF%BC%89%E5%A6%82%E4%BD%95%E5%88%B6%E4%BD%9C%E4%B8%80%E4%B8%AA%E7%B1%BB%E4%BC%BCtiny-wings%E7%9A%84%E6%B8%B8%E6%88%8F%EF%BC%9A%E7%AC%AC%E4%B8%80%E9%83%A8%E5%88%86.html)教程中，我们学会了如何动态创建游戏里所需要的山丘。

在这篇教程中，也是本系列教程的最后一篇，我们将会学习到更加有意思的部分—如何往游戏里面添加主角，同时使用BOX2D来仿真主角的移动！

再说明一下，这个教程系列是基于[Sergey Tikhonov](http://haqu.net/)所写的一个非常好的[demo project](https://github.com/haqu/tiny-wings)制作的—所以我要特别感谢Sergey！

这个教程假设你对cocos2d和box2d已经很熟悉了。如果你对这两者还很陌生的话，建议你先阅读本博客上翻译的[cocos2d教程](http://www.cnblogs.com/andyque/category/305876.html)和[box2d教程](http://www.cnblogs.com/andyque/category/300608.html)。

**Getting Started**

如果你还没有准备好，可以先下载[上一篇教程](http://www.cnblogs.com/andyque/archive/2011/07/01/2095489.html)中完成的[样例工程](http://www.raywenderlich.com/downloads/TinySeal2.zip)。

接下来，我们将添加一些基本的box2d代码。我们将创建一个box2d world和一些代码来激活debug drawing，同时还会添加一些测试用shape，以此确保BOX2D环境被正确搭建起来！

首先，打开HelloWorldLayer.h，然后作如下修改：

// Add to top of file

#import ”Box2D.h”

#define PTM\_RATIO 32.0// Add inside @interface

b2World \* \_world;

这里包含了box2d的头文件和debug draw的头文件，同时定义一个\_world变量来追踪box2d的world与debug draw类。

同时，我们也声明了一个像素/米的转换率（PTM\_RATIO）为32.回顾一下，这个变量主要作用是在box2d的单位（米）和cocos2d的单位（点）之间做转换。

然后，我们在HelloWorldLayer.mm中添加一下新的方法，添加位置在init方法上面：

- (void)setupWorld {

b2Vec2 gravity = b2Vec2(0.0f, -7.0f);

bool doSleep = true;

\_world = new b2World(gravity, doSleep);

}- (void)createTestBodyAtPostition:(CGPoint)position {

b2BodyDef testBodyDef; testBodyDef.type = b2\_dynamicBody; testBodyDef.position.Set(position.x/PTM\_RATIO, position.y/PTM\_RATIO); b2Body \* testBody = \_world->CreateBody(&testBodyDef);

b2CircleShape testBodyShape; b2FixtureDef testFixtureDef; testBodyShape.m\_radius = 25.0/PTM\_RATIO; testFixtureDef.shape = &testBodyShape; testFixtureDef.density = 1.0; testFixtureDef.friction = 0.2; testFixtureDef.restitution = 0.5; testBody->CreateFixture(&testFixtureDef);

}

如果你对box2d很熟悉的话，上面这个方法只是一个回顾。

setupWorld方法创建一个有重力的world–但是比标准的重力-9.8m/s^2要小一点点。

createTestBodyAtPostition创建一个测试对象—一个25个点大小的圆。我们将使用这个方法来创建一个测试对象，每一次你点击屏幕就会在那个地方产生一个圆，不过这只是测试用，之后会被删除掉。

你现在还没有完成HelloWorldLayer.mm–现在再作一些修改，如下所示：

// Add to the TOP of init

[self setupWorld];// Replace line to create Terrain in init with the following

\_terrain = [[[Terrain alloc] initWithWorld:\_world] autorelease];

// Add to the TOP of update static double UPDATE\_INTERVAL = 1.0f/60.0f; static double MAX\_CYCLES\_PER\_FRAME = 5; static double timeAccumulator = 0;

timeAccumulator += dt; if (timeAccumulator > (MAX\_CYCLES\_PER\_FRAME \* UPDATE\_INTERVAL)) { timeAccumulator = UPDATE\_INTERVAL; }

int32 velocityIterations = 3; int32 positionIterations = 2; while (timeAccumulator >= UPDATE\_INTERVAL) { timeAccumulator -= UPDATE\_INTERVAL; \_world->Step(UPDATE\_INTERVAL, velocityIterations, positionIterations); \_world->ClearForces();

}

// Add to bottom of ccTouchesBegan UITouch \*anyTouch = [touches anyObject]; CGPoint touchLocation = [\_terrain convertTouchToNodeSpace:anyTouch]; [self createTestBodyAtPostition:touchLocation];

第一段代码，我们调用setupWorld方法来创建一个box2d世界。然后使用box2d的world来初始化Terrain类。这样，我们就可以使用这个world来创建山丘的body了。为此，我们将会写一些桩代码（placeholder)。

第二段代码，我们调用\_world->Step方法来运行物理仿真。注意，这里使用的是固定时间步长的实现方式，它比变长时间步长的方式物理仿真效果要更好。对于具体这个是怎么工作的，可以去看看我们的[cocos2d](http://www.amazon.com/gp/product/0321735625/ref=as_li_tf_tl?ie=UTF8&tag=raywend-20&linkCode=as2&camp=217145&creative=399349&creativeASIN=0321735625)书籍中关于box2d的那一章节内容。

最后一段代码是添加到ccTouchesBegan里面，不管什么时候你点击屏幕，就会创建一个box2d的body。再说一下，这样做只是为了测试box2d环境可以run起来了。

注意，我们这里得到的touch坐标是在地形的坐标之内。这是因为，地形将会滚动，而我们想知道地形的位置，而不是屏幕的位置。

接下来，让我们修改一下Terrain.h/m。首先，修改Terrain.h，如下所示：

// Add to top of file

#import ”Box2D.h”

#import ”GLES-Render.h”// Add inside @interface

b2World \*\_world;

b2Body \*\_body;

GLESDebugDraw \* \_debugDraw;

// Add after @interface - (id)initWithWorld:(b2World \*)world;

这里只是包含box2d头文件，然后创建一些实例变量来追踪box2d的world,以及山丘的body,还有支持debug drawing的对象。同时，我们还定义了初始化方法，它接收box2d的world作为参数。

然后在Terrain.m中添加一个新的方法，位置在generateHills上面：

- (void) resetBox2DBody {if(\_body) return;

CGPoint p0 = \_hillKeyPoints[0]; CGPoint p1 = \_hillKeyPoints[kMaxHillKeyPoints-1];

b2BodyDef bd; bd.position.Set(0, 0); \_body = \_world->CreateBody(&bd);

b2PolygonShape shape; b2Vec2 ep1 = b2Vec2(p0.x/PTM\_RATIO, 0); b2Vec2 ep2 = b2Vec2(p1.x/PTM\_RATIO, 0); shape.SetAsEdge(ep1, ep2); \_body->CreateFixture(&shape, 0); }

这里仅仅是一个辅助方法，用来创建山丘的的底部body，代表“地面”。这里只是暂时用这个方法，用来防止随机生成的圆会掉到屏幕之外去。之后，在我们建模好山丘后，我们会再次修改。

目前，我们只是把第一个关键点和最后一个关键点用一条边连接起来。

接下来，在Terrain.m中添加一些代码来调用上面的代码，同时建立起debug drawing：

// Add inside resetHillVertices, right after “prevToKeyPointI = \_toKeyPointI” line:

[self resetBox2DBody];// Add new method above init

- (void)setupDebugDraw {

\_debugDraw = new GLESDebugDraw(PTM\_RATIO\*[[CCDirector sharedDirector] contentScaleFactor]);

\_world->SetDebugDraw(\_debugDraw);

\_debugDraw->SetFlags(b2DebugDraw::e\_shapeBit | b2DebugDraw::e\_jointBit);

}

// Replace init with the following - (id)initWithWorld:(b2World \*)world { if ((self = [super init])) { \_world = world; [self setupDebugDraw]; [self generateHills]; [self resetHillVertices]; } return self; }

// Add at bottom of draw glDisable(GL\_TEXTURE\_2D); glDisableClientState(GL\_COLOR\_ARRAY); glDisableClientState(GL\_TEXTURE\_COORD\_ARRAY);

\_world->DrawDebugData();

glEnable(GL\_TEXTURE\_2D); glEnableClientState(GL\_COLOR\_ARRAY); glEnableClientState(GL\_TEXTURE\_COORD\_ARRAY);

每一次山丘顶点被重置的时候，我们调用resetBox2DBody来创建可见部分山丘的body。目前，这个body是不变的（它只是添加了一条线，当作地面）。但是，接下来，我们将修改这个来建模可见部分的山丘。

setupDebugDraw方法设置了激活box2d 对象debug drawing所需要的一些配置。如果你熟悉box2d的话，那么这个就是回顾啦。

然后，你可能会奇怪，为什么debug draw的代码要放在Terrain.m文件中呢？而不是放在HelloWorldLayer.mm中呢？这是因为，这个游戏中的滚动效果是在Terrain.m中实现的。因此，为了使box2d的坐标系统和屏幕范围内可见部分的坐标系统匹配起来，我们就把debug drawing代码放在Terrain.m中了。

最后一步，如果你现在想要编译的话，可能会出现几百个错误。这是因为Terrain.m导入了Terrain.h文件，而Terrain.h文件又包含了HelloWorldLayer.h文件，而它又导入了Box2D.h头文件。而不管什么时候，只要你在.m文件中使用c++的话，那么就会产生一大堆的错误。

不过还好，解决办法非常简单—只要把Terrain.m改成Terrain.mm就可以了。

编译并运行，现在，你点击一下屏幕，你会看到许多圆形对象掉在屏幕里面拉！



**在box2d里面为山丘定义body边界**

现在，我们只拥有一个box2d的shape代表屏幕的底部边界，但是，我们真正想要的是代表山丘边界的shape。

幸运的是，因为我们拥有所有的线段了，所以添加边界会非常简单！

* 我们有一个山丘顶部所有顶点的数组(borderVertices). 在上一个教程的resetHillVertices方法中，我们生成了这样一个数组。
* 我们有一个方法，不管什么时候顶点为被改变了，它都会被调用，那就是resetBox2DBody.

因此，我们需要修改resetBox2DBody方法，我们要为borderVertices组织中的每一个实体创建一条边，具体方法如下：

- (void) resetBox2DBody {if(\_body) {

\_world->DestroyBody(\_body);

}

b2BodyDef bd; bd.position.Set(0, 0);

\_body = \_world->CreateBody(&bd);

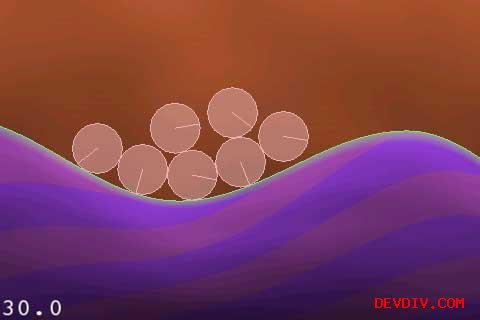
b2PolygonShape shape;

b2Vec2 p1, p2; for (int i=0; i<\_nBorderVertices-1; i++) { p1 = b2Vec2(\_borderVertices[i].x/PTM\_RATIO,\_borderVertices[i].y/PTM\_RATIO); p2 = b2Vec2(\_borderVertices[i+1].x/PTM\_RATIO,\_borderVertices[i+1].y/PTM\_RATIO); shape.SetAsEdge(p1, p2); \_body->CreateFixture(&shape, 0); } }

这个新的实现首先看看是不是存在一个已有的box2d body，如果是的话，就销毁原来的body。

然后，它创建一个新的body，循环遍历border vertices数组里面的所有顶点，这些顶点代表山丘顶部。对于每2个顶点，都将创建一条边来连接它们。

很简单，对不对？编译并运行，现在，你可以看到一个带有斜坡的box2d body了，而且它沿着山丘的纹理边界。



**添加海豹**

我们之前把工程命名为Tiny Seal，可是并没有seal 啊！

接下来，让我们把海豹添加进去！

首先，下载并解压这个工程的[资源文件](http://www.raywenderlich.com/downloads/TinySealResources.zip)，然后把”Sprite sheets“和”Sounds“直接拖到工程里去，对于每一个文件夹，都要确保“Copy items into destination group’s folder”被复选中，然后点击”Finish”。

然后，点击File\New\New File，选择iOS\Cocoa Touch\Objective-C class，再点Next。选择CCSprite作为基类，再点Next，然后把文件命名为Hero.mm（注意，.mm是因为我们将使用到box2d的东西），最后点击Finish.

接着，把Hero.h替换成下面的内容：

#import ”cocos2d.h”

#import ”Box2D.h”#define PTM\_RATIO 32.0

@interface Hero : CCSprite { b2World \*\_world; b2Body \*\_body; BOOL \_awake; }

- (id)initWithWorld:(b2World \*)world; - (void)update;

@end

这个也非常简单—只是导入box2d.h头文件，然后定义一些变量来追踪world和海豹的body.

然后，打开Hero.mm，然后作如下修改：

#import ”Hero.h”@implementation Hero

- (void)createBody {

float radius = 16.0f; CGSize size = [[CCDirector sharedDirector] winSize]; int screenH = size.height;

CGPoint startPosition = ccp(0, screenH/2+radius);

b2BodyDef bd; bd.type = b2\_dynamicBody; bd.linearDamping = 0.1f; bd.fixedRotation = true; bd.position.Set(startPosition.x/PTM\_RATIO, startPosition.y/PTM\_RATIO); \_body = \_world->CreateBody(&bd);

b2CircleShape shape; shape.m\_radius = radius/PTM\_RATIO;

b2FixtureDef fd; fd.shape = &shape; fd.density = 1.0f; fd.restitution = 0.0f; fd.friction = 0.2;

\_body->CreateFixture(&fd);

}

- (id)initWithWorld:(b2World \*)world {

if ((self = [super initWithSpriteFrameName:@"seal1.png"])) { \_world = world; [self createBody]; } return self;

}

- (void)update {

self.position = ccp(\_body->GetPosition().x\*PTM\_RATIO, \_body->GetPosition().y\*PTM\_RATIO); b2Vec2 vel = \_body->GetLinearVelocity(); b2Vec2 weightedVel = vel; float angle = ccpToAngle(ccp(vel.x, vel.y)); if (\_awake) { self.rotation = -1 \* CC\_RADIANS\_TO\_DEGREES(angle); } }

@end

createBody方法为海豹创建了一个圆形的shape。这个方法和之前写过的createTestBodyAtPosition方法几乎没有什么区别，除了圆的大小和海豹图片的大小要匹配（不过实际上要比图片大小小一些，这样子碰撞检测效果会更好）

同时，这里的摩擦系数（friction）设置为0.2（因为海豹是很滑的），同时反弹系数(restitution）设置为0（这样子，当海豹碰撞到山丘的时候就不会反弹起来了）。

同时，我们也设置body的线性阻尼（ linear damping），这样子海豹就会随着时间慢慢减速。同时，设置body的固定旋转为真，这样子，海豹在游戏里面就不会旋转body了。

在initWithWorld方法里面，我们把精灵初始化为一个特定的精灵帧（seal1.png），同时保存一份world的指针，然后调用上面的createBody方法。

这里的update方法基于box2d body的位置来更新海豹精灵的位置，同时基于海豹的body的速度来更新海豹精灵的旋转。

接下来，你需要修改一下Terrain.h和Terrain.mm，因为，我们将要在Terrain.mm中添加一个sprite batch node。

首先，打开Terrain.h，并作以下修改：

// Inside @interface

CCSpriteBatchNode \* \_batchNode;// After @implementation

@property (retain) CCSpriteBatchNode \* batchNode;

然后，打开Terrain.mm，并作如下修改：

// Add to top of file

@synthesize batchNode = \_batchNode;// Add at bottom of init

\_batchNode = [CCSpriteBatchNode batchNodeWithFile:@"TinySeal.png"];

[self addChild:\_batchNode];

[[CCSpriteFrameCache sharedSpriteFrameCache] addSpriteFramesWithFile:@”TinySeal.plist”];

这里只是为TinySeal.png精灵表单创建了一个batch node，然后从TinySeal.plist文件中加载了精灵帧的定义信息到sprite frame cache中。

差不多完成了！接下来，让我们修改HelloWorldLayer.h：

// Add to top of file

#import ”Hero.h”// Add inside @interface

Hero \* \_hero;

同时修改HelloWorldLayer.mm：

// Add to bottom of init

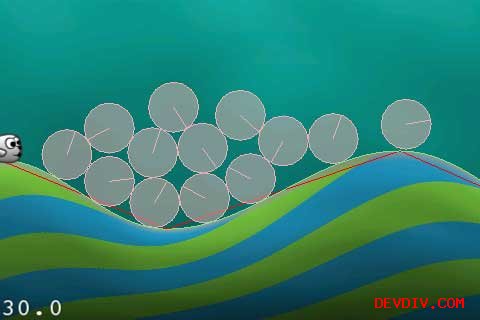
\_hero = [[[Hero alloc] initWithWorld:\_world] autorelease];

[\_terrain.batchNode addChild:\_hero];// In update, comment out the three lines starting with PIXELS\_PER\_SECOND and add the following

[\_hero update];

float offset = \_hero.position.x;

编译并运行，你现在可以看到一只happy的海豹在屏幕左边了！



但是，看起来有起奇怪，它在屏幕之外！如果我们把它往右边挪一下，那样子看起来会更好。

当然，这个改起来很简单！打开Terrain.mm，然后把setOffsetX改成下面的样子：

- (void) setOffsetX:(float)newOffsetX {

CGSize winSize = [CCDirector sharedDirector].winSize;\_offsetX = newOffsetX;

self.position = CGPointMake(winSize.width/8-\_offsetX\*self.scale, 0);

[self resetHillVertices];

}

这里把海豹的位置旋转在屏幕的1/8处，这样子海豹看起来就会往右边一点点了。编译并运行，现在可以看到海豹的全貌啦！



**使海豹移动**

我们离一个完整的游戏越来越近了—我们有一只海豹，我们只需要让它飞起来就可以啦！

我们采取的策略如下：

* 第一次点击屏幕的时候，我们让海豹稍微往右边跳起来一点点，代表开始了！
* 不管什么时候点击屏幕，我们应用一个冲力使海豹往下落。当海豹下山时，会使它的速度变得更快，这样到下一个山头的时候就可以飞起来了。
* 添加一些代码让海豹移动的距离稍微远一点，我们可不想让我们的海豹卡住！

让我们来实现这些策略吧！打开Hero.h，作如下修改：

// Add after @implementation

@property (readonly) BOOL awake;

- (void)wake;

- (void)dive;

- (void)limitVelocity;

然后对Hero.mm作如下修改：

// Add to top of file

@synthesize awake = \_awake;// Add new methods

- (void) wake {

\_awake = YES;

\_body->SetActive(true);

\_body->ApplyLinearImpulse(b2Vec2(1,2), \_body->GetPosition());

}

- (void) dive { \_body->ApplyForce(b2Vec2(5,-50),\_body->GetPosition()); }

- (void) limitVelocity { if (!\_awake) return;

const float minVelocityX = 5; const float minVelocityY = -40; b2Vec2 vel = \_body->GetLinearVelocity(); if (vel.x < minVelocityX) { vel.x = minVelocityX; } if (vel.y < minVelocityY) { vel.y = minVelocityY; } \_body->SetLinearVelocity(vel); }

这个wake方法应用一个冲力（impulse)使得海豹刚开始往右上方飞。

dive方法应用一个比较大的向下的冲力，和一个比较小的向右的力。这个向下的冲力会使得海豹往山丘上撞，这时，山丘的斜坡越大，那么小鸟就飞得越高。（应该是上山的时候，下山相反)

limitVelocity方法确保海豹速度至少在 x轴方向5m/s²，Y轴方向-40m/s²。

基本上要完成了—只需要再修改一下HelloWorldLayer类。首先打开HelloWorldLayer.h，然后添加一个新的实例变量：

BOOL \_tapDown;

同时修改HelloWorldLayer.mm：

// Add at the top of the update method

if (\_tapDown) {

if (!\_hero.awake) {

[\_hero wake];

\_tapDown = NO;

} else {

[\_hero dive];

}

}

[\_hero limitVelocity];// Replace ccTouchesBegan with the following

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

[self genBackground];

\_tapDown = YES;

}

// Add new methods -(void)ccTouchesEnded:(NSSet \*)touches withEvent:(UIEvent \*)event { \_tapDown = NO; }

- (void)ccTouchesCancelled:(NSSet \*)touches withEvent:(UIEvent \*)event { \_tapDown = NO; }

编译并运行，现在你有一只可以飞的海豹啦！



**修正海豹身体的摇晃**

你可能注意到了，当海豹往下飞的时候，身体摇摇晃晃的。

一种方式就是，使用之前的线性速度和现在得到的速度作加权平均。

让我们来实现一下。先打开Hero.h：

// Add to top of file

#define NUM\_PREV\_VELS 5// Add inside @interface

b2Vec2 \_prevVels[NUM\_PREV\_VELS];

int \_nextVel;

然后修改Hero.mm的update方法：

- (void)update {self.position = ccp(\_body->GetPosition().x\*PTM\_RATIO, \_body->GetPosition().y\*PTM\_RATIO);

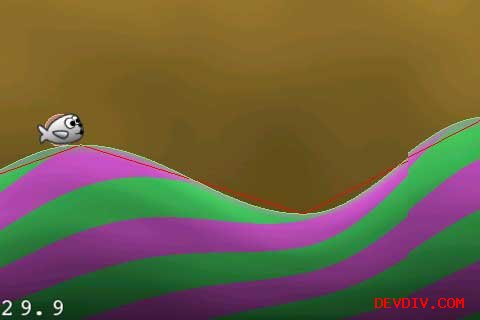
b2Vec2 vel = \_body->GetLinearVelocity();

b2Vec2 weightedVel = vel;

for(int i = 0; i < NUM\_PREV\_VELS; ++i) { weightedVel += \_prevVels[i]; } weightedVel = b2Vec2(weightedVel.x/NUM\_PREV\_VELS, weightedVel.y/NUM\_PREV\_VELS); \_prevVels[\_nextVel++] = vel; if (\_nextVel >= NUM\_PREV\_VELS) \_nextVel = 0;

float angle = ccpToAngle(ccp(weightedVel.x, weightedVel.y)); if (\_awake) { self.rotation = -1 \* CC\_RADIANS\_TO\_DEGREES(angle); } }

这里使用之前的5个线性速度作加权平均，然后使用平均值来修正海豹的旋转。编译并运行，现在你可以看到更加平滑的海豹啦！



**缩小**

Tiny Wings有一个很酷的特性就是，你飞得越高，那么屏幕就会越小。这使得视觉感观更加逼真！

为了实现这个，我们只需要在HelloWorldLayer.mm的update方法里面的[\_hero update]调用之后，再添加下面代码就行了：

CGSize winSize = [CCDirector sharedDirector].winSize;

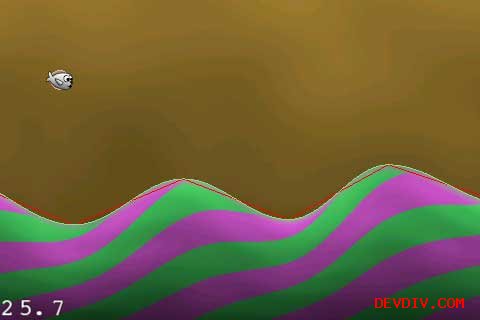
float scale = (winSize.height\*3/4) / \_hero.position.y;

if (scale > 1) scale = 1;

\_terrain.scale = scale;

如果hero在winSize.height\*3/4以下，那么scale就为1.如果它大于winSize.height\*3/4，那么scale就会小于1，就会有缩小的感觉了。

编译并运行，现在看看你能飞多高吧！



**免费的动画和音乐**

你懂的，我不能让你们这些粉丝没有一些免费的动画和音乐可玩。：）

只需要花上几秒钟的时间就可以使游戏变得更有趣！首先，打开Hero.h，并作如下修改：

// Add inside @interface

CCAnimation \*\_normalAnim;

CCAnimate \*\_normalAnimate;// Add after @interface

- (void)nodive;

- (void)runForceAnimation;

- (void)runNormalAnimation;

这里声明我们即将创建的动画，还有一个新方法将在海豹没有diving的时候被调用。

接下来，修改Hero.mm：

// Add new methods

- (void)runNormalAnimation {

if (\_normalAnimate || !\_awake) return;

\_normalAnimate = [CCRepeatForever actionWithAction:[CCAnimate actionWithAnimation:\_normalAnim]];

[self runAction:\_normalAnimate];

}- (void)runForceAnimation {

[\_normalAnimate stop];

\_normalAnimate = nil;

[self setDisplayFrame:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@”seal\_downhill.png”]];

}

- (void)nodive { [self runNormalAnimation]; }

// Add at bottom of initWithWorld: \_normalAnim = [[CCAnimation alloc] init]; [\_normalAnim addFrame:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@”seal1.png”]]; [\_normalAnim addFrame:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@”seal2.png”]]; \_normalAnim.delay = 0.1;

这里为海豹的正常飞行创建了动画效果，同时添加一个方法来播放这个动画。diving动画实际上只是一个精灵帧，因此我们添加了一个辅助方法来完成divng动画的播放。

最后，我们修改一下HelloWorldLayer.mm：

// At top of file

#import ”SimpleAudioEngine.h”// At end of init

[[SimpleAudioEngine sharedEngine] playBackgroundMusic:@”TinySeal.caf” loop:YES];

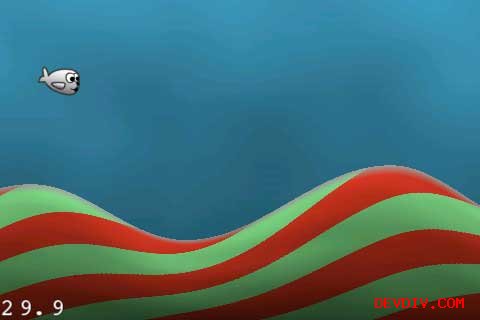
// At start of update, add an else case for the if (\_tapDown): else { [\_hero nodive]; }

// Inside ccTouchesBegan [\_hero runForceAnimation];

// Inside ccTouchesEnded AND ccTouchesCancelled [\_hero runNormalAnimation];

最后，打开Terrian.mm，注释掉draw方法里面的\_world->DrawDebugData。

编译并运行代码，大功造成了！



**何去何从?**

这里有本系列教程的[全部源代码](http://www.raywenderlich.com/downloads/TinySeal3.zip)。

到目前为止，你有一个基本的游戏框架可以玩了。为什么不尝试着完善这个游戏呢？把海豹移动的行为修改得更加逼真、更加平滑一些？或者，你可以添加一些物品和道具，充分发挥你的想象力吧！

如果你扩展了本项目，不妨拿出来分享一下，大家一起学习一下吧！

PS：译者水平有限，翻译不准的地方望不吝指出，谢谢！

**iPhone上面的现实增强（Augmented Reality ）入门教程 by** [**fengliu**](http://www.devdiv.com/blog/?author=140)

免责申明（必读！）：本博客提供的所有教程的翻译原稿均来自于互联网，仅供学习交流之用，切勿进行商业传播。同时，转载时不要移除本申明。如产生任何纠纷，均与本博客所有人、发表该翻译稿之人无任何关系。谢谢合作！

原文链接地址：<http://www.raywenderlich.com/3997/introduction-to-augmented-reality-on-the-iphone>

教程截图：

[](http://blog.devdiv.com/?attachment_id=4001)Create a simple augmented reality game where you can blast some aliens!

在这篇教程中，你将学习到如何为你的iphone和ipod touch制作一个简单的增强现实游戏。

在这个游戏中，你将使用到摄像头，陀螺仪和cocos2d框架。听起来很振奋人心吧？

在写作这篇教程的时候，探索上面提到的一些技术的过程真的是非常有趣。这里有一些数学和转换，不过不要担心—没有什么事情是很难的！

学习这篇教程的时候，你需要一个iPhone4，因为这个教程使用陀螺仪来移动游戏世界视图。

你也需要一些基本的cocos2d方面的知识，当然要安装好cocos2d。如果你对于cocos2d完全陌生的话，你可以先看看本博客上的[其它cocos2d教程](http://blog.devdiv.com/fengliu)。

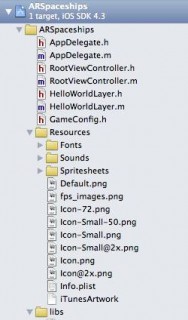
你准备好爆头一些虚拟外星人了吗？跟我来吧！

**Getting Started**

打开Xcode，然后从File菜单中选择New\New Project。然后选择 iOS\cocos2d\cocos2d template，接着点击Next。把工程命名为ARSpaceShips，并点击Next，同时选择一个文件夹位置来保存本项目，最后点Create。

我们将重用Space Shooter游戏的一些资源文件，所以，先下载[它们](http://www.raywenderlich.com/downloads/SpaceGameResources.zip)并解压缩。

下载完后，把Fonts，Sounds和Spritesheet文件夹拖到Resources分组下面。同时确保 Copy items into destination group’s folder被复选中，并且ARSpaceships target也要被选中。然后点击Finish。这时，你的工程看起来应该如下图所示：



我们将在后面使用到这些资源。

**玩一玩摄像头!**

如果你现在运行项目，那简单太无聊了，就是一个黑屏的HelloWorld，这玩意儿谁没看过！让我们往里面添加一些非常好玩的内容吧。选择AppDelegate.h，然后在interface里面添加一个UIView成员变量。

UIView \*overlay;

现在，打开AppDelegate.m文件，找到EAGLView \*glView所在的代码行，把像素格式改成kEAGLColorFormatRGBA8。改完后如下图所示：

EAGLView \*glView = [EAGLView viewWithFrame:[window bounds]

pixelFormat:kEAGLColorFormatRGBA8 depthFormat:0];

如果你不改变像素格式的话，那么摄像头里拍摄出来的图像就显示不出来。因为我们现在正在做一个增强现实的游戏，所以必须这样做！

在 [window addSubview: viewController.view];下面，我们添加了以下代码：

// set the background color of the view

[CCDirector sharedDirector].openGLView.backgroundColor = [UIColor clearColor];

[CCDirector sharedDirector].openGLView.opaque = NO;// set value for glClearColor

glClearColor(0.0, 0.0, 0.0, 0.0);

// prepare the overlay view and add it to the window overlay = [[UIView alloc] initWithFrame:[[UIScreen mainScreen] bounds]]; overlay.opaque = NO; overlay.backgroundColor=[UIColor clearColor]; [window addSubview:overlay];

这里，我们把openGLView的背景颜色清除了，把视图设置为透明的，同时指定了glClearColor，最后，我们创建并添加了一个新的视图，叫做overlay。这个视图后面用来显示camera里的内容。

接下来，在你刚刚添加的代码后面增加以下代码行：

#define CAMERA\_TRANSFORM 1.24299UIImagePickerController \*uip;

@try { uip = [[[UIImagePickerController alloc] init] autorelease]; uip.sourceType = UIImagePickerControllerSourceTypeCamera; uip.showsCameraControls = NO; uip.toolbarHidden = YES; uip.navigationBarHidden = YES; uip.wantsFullScreenLayout = YES; uip.cameraViewTransform = CGAffineTransformScale(uip.cameraViewTransform, CAMERA\_TRANSFORM, CAMERA\_TRANSFORM); } @catch (NSException \* e) { [uip release]; uip = nil; } @finally { if(uip) { [overlay addSubview:[uip view]]; [overlay release]; } }

[window bringSubviewToFront:viewController.view];

首先，我们了一个常量来缩放摄像头。摄像头的比率是4：3，而iphone屏幕的比例是3:4，所以，我们需要缩放一下摄像头，使之与屏幕匹配。

其次，我们创建了一个UIImagePickerController，设置了它的一些属性，具体效果就是没有控件并且缩放了的摄像头。然后，我们把它添加到了overlay视图中。

最后，我们需要把 viewController.view显示到前面来（因为它包含了cocos2d的显示内容）。这样子摄像头拍摄的内容也就会显示到前台来了。

现在，运行一下应用程序。现在，你将看到摄像头里面捕捉的画面是你的Hello World的背景了。



**Shake, Rattle, and Roll…Well at Least Yaw!**

因为我们现在的程序已经可以把现实捕捉并显示出来了，接下来，我们将集中精力来解决本教程中比较难的部分。

首先，我们需要把CoreMotion framework添加到项目中去。点击project文件，然后选择ARSpaceships target，再选择Build Phases标签页，展开 Link Binary With Libraries.。



点击上图中的+号，选择 CoreMotion.framework，然后点击Add按钮。现在，我们就可以在项目中使用陀螺仪啦。

打开HelloWorldLayer.h，然后导入一些头文件：

#include <CoreMotion/CoreMotion.h>

#import <CoreFoundation/CoreFoundation.h>

然后在interface里面添加一些成员变量：

CMMotionManager \*motionManager;

CCLabelTTF \*yawLabel;

CCLabelTTF \*posIn360Label;

同时添加属性声明语句：

@property (nonatomic, retain) CMMotionManager \*motionManager;

现在，重点要来了！打开HelloWorldLayer.m文件，在if ((self=[super init]))语句内部，删除原来的“Hello World”标签语句，同时添加下面的代码来设置一些新的标签.

// add and position the labels

yawLabel = [CCLabelTTF labelWithString:@"Yaw: " fontName:@"Marker Felt" fontSize:12];

posIn360Label = [CCLabelTTF labelWithString:@"360Pos: " fontName:@"Marker Felt" fontSize:12];

yawLabel.position = ccp(50, 240);

posIn360Label.position = ccp(50, 300);

[self addChild: yawLabel];

[self addChild:posIn360Label];

目前为止，并没有什么特别的。只是添加了一些标签，指明了字体和一些文字。标签的位置都是在屏幕的左边。

接下来，你需要设置motion manager，它会启动陀螺仪。

self.motionManager = [[[CMMotionManager alloc] init] autorelease];

motionManager.deviceMotionUpdateInterval = 1.0/60.0;

if (motionManager.isDeviceMotionAvailable) {

[motionManager startDeviceMotionUpdates];

}[self scheduleUpdate];

这里，我们分配并初始化了 motion manager对象。同时，我们还设置了更新间隔为每秒60次。如果设置支持陀螺仪的话，那么就启动更新。最后，我们触发一个update定时器。

不要忘了在.m文件中添加synthesize语句，如下所示：

@synthesize motionManager;

因为我们触发了update定时器，所以我们需要添加一个update方法。同时，在init方法的下面增加下面一个方法：

-(void)update:(ccTime)delta {

CMDeviceMotion \*currentDeviceMotion = motionManager.deviceMotion;

CMAttitude \*currentAttitude = currentDeviceMotion.attitude;// 1: Convert the radians yaw value to degrees then round up/down

float yaw = roundf((float)(CC\_RADIANS\_TO\_DEGREES(currentAttitude.yaw)));

// 2: Convert the degrees value to float and use Math function to round the value [yawLabel setString:[NSString stringWithFormat:@"Yaw: %.0f", yaw]];

// 3: Convert the yaw value to a value in the range of 0 to 360 int positionIn360 = yaw; if (positionIn360 < 0) { positionIn360 = 360 + positionIn360; }

[posIn360Label setString:[NSString stringWithFormat:@"360Pos: %d", positionIn360]];

}

现在，你可以运行应用程序了。你将会看到Yaw和positionIn360的对应标签值在改变。

**那玩意儿究竟如何工作？**

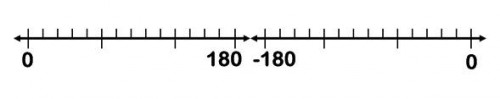
尽管可以跑起来了，但是你可能会问，它究竟是如何工作的呢？让我们花几分钟时间来一步步解释上面的代码是如何工作的。

首先，下载iTunes app store上面的免费程序 [Gyrosocope app](http://itunes.apple.com/us/app/gyroscope/id381953722?mt=8)。安装并运行它，当你移动iphone的时候，就可以看到每个值究竟是怎么变化的。



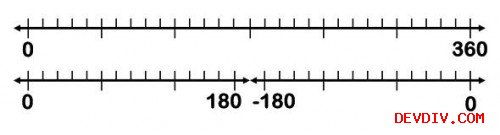
我们关心的值是Yaw。这个值代表往左或往右移动。在Gyroscope程序中，它的单位是度，然而 motion manager获取的值却是弧度。这就是为什么我们需要使用内置的函数CC\_RADIANS\_TO\_DEGREES来把弧度转换成角度的原因啦。

因此，在第一部分中，我们得到了yaw的弧度值，并且把它转换成角度，最后赋值给yaw变量。第二部分，我们只是把yaw的值显示到屏幕上去。如果你运行工程，你会看到yaw值的变化范围是从0~180，然后又从-180回到0.



现在看看第三部分，你可能会奇怪positionIn360的值倒底是什么啊？好吧，这里只是一个手段，目的是使得放置飞碟的过程变得容易。

这里的逻辑其实非常简单。如果yaw值是正的，那么我们什么也不做。如果是负的，那么就减去360.（加上一个负值和减去一个正值是一样的）。最后一行代码只是在屏幕上显示那个值。



如果你还没完全理解，没关系，运行一下程序，看看具体值是怎么变化的吧。

**灯光，摄像机，Action！**

现在，我们为陀螺仪的使用奠定基础了，让我们往里面添加一些太空飞船吧！我们将创建一个新的文件。首先，左键点ARSpaceships工程文件，然后选择New File。接着选 iOS\Cocoa Touch\Objective-C class，然后点击Next。确保NSObject被选中基类，然后点Next。把文件命名为EnemyShip.m，最后点Save。

把 EnemyShip.h里的内容换成下面的代码：

#import ”cocos2d.h”@interface EnemyShip : CCSprite {

int yawPosition;

int timeToLive;

}

@property (readwrite) int yawPosition; @property (readwrite) int timeToLive;

@end

同时修改EnemyShip.m：

#import ”EnemyShip.h”@implementation EnemyShip

@synthesize yawPosition, timeToLive;

-(id)init { self = [super init]; if (self){ yawPosition = 0; timeToLive = 0; } return self; }

@end

现在，回到HelloWorldLayer.h。在顶部导入EnemyShip类的头文件，如下所示：

#import ”EnemyShip.h”

在interface里面声明以下成员变量：

NSMutableArray \*enemySprites;

int enemyCount;

CCSpriteBatchNode \*batchNode;

最后，在interface声明下面，添加enemyCount属性，同时定义一些方法，具体如下图所示：

@property (readwrite) int enemyCount;-(EnemyShip \*)addEnemyShip:(int)shipTag;

-(void)checkEnemyShipPosition:(EnemyShip \*)enemyShip withYaw:(float)yawPosition;

-(void)updateEnemyShipPosition:(int)positionIn360 withEnemy:(EnemyShip \*)enemyShip;

-(void)runStandardPositionCheck:(int)positionIn360 withDiff:(int)difference withEnemy:(EnemyShip \*)enemyShip;

打开 HelloWorldLayer.m文件，同时作以下修改：

// Place after the #import statement

#include <stdlib.h>// Place after the other @synthesize statement

@synthesize enemyCount;

#define kXPositionMultiplier 15

#define kTimeToLive 100

// Add to the bottom of init batchNode = [CCSpriteBatchNode batchNodeWithFile:@"Sprites.pvr.ccz"]; [self addChild:batchNode]; [[CCSpriteFrameCache sharedSpriteFrameCache] addSpriteFramesWithFile:@”Sprites.plist”];

这里，我们加载了spritesheet，它的资源文件在一开始的时候，我们就把它拖到项目中来了。

接下来，我们将添加一个新的方法来创建新的敌人太空飞船。在dealloc方法的上面添加下列方法：

-(EnemyShip \*)addEnemyShip:(int)shipTag {EnemyShip \*enemyShip = [EnemyShip spriteWithSpriteFrameName:@"enemy\_spaceship.png"];

// Set position of the space ship randomly int x = arc4random() % 360; enemyShip.yawPosition = x;

// Set the position of the space ship off the screen, but in the center of the y axis // we will update it in another method [enemyShip setPosition:ccp(5000, 160)];

// Set time to live on the space ship enemyShip.timeToLive = kTimeToLive; enemyShip.visible = true;

[batchNode addChild:enemyShip z:3 tag:shipTag];

return enemyShip; }

这个方法接收一个整数值作为tag，并且返回一个EnemyShip CCSprite。下面一行代码，我们从精灵表单中创建一个EnemyShip精灵。接着，我们使用arc4random方法来得到一个0~360的随机数。最后，我们设置了飞船的位置，并把timeToLive的值设置为100，把飞船添加到batchNode里面，最后返回飞船精灵对象。

在addEnemyShip下面，我们添加一个checkEnemyShipPosition方法：

-(void)checkEnemyShipPosition:(EnemyShip \*)enemyShip withYaw:(float)yawPosition {

// Convert the yaw value to a value in the range of 0 to 360

int positionIn360 = yawPosition;

if (positionIn360 < 0) {

positionIn360 = 360 + positionIn360;

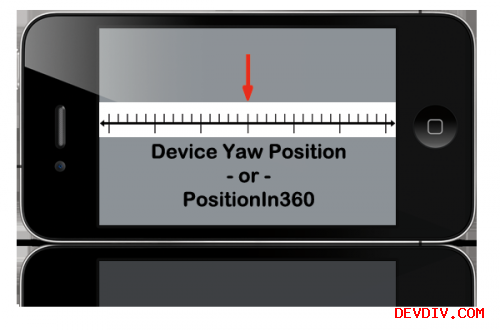
}BOOL checkAlternateRange = false;

// Determine the minimum position for enemy ship int rangeMin = positionIn360 - 23; if (rangeMin < 0) { rangeMin = 360 + rangeMin; checkAlternateRange = true; }

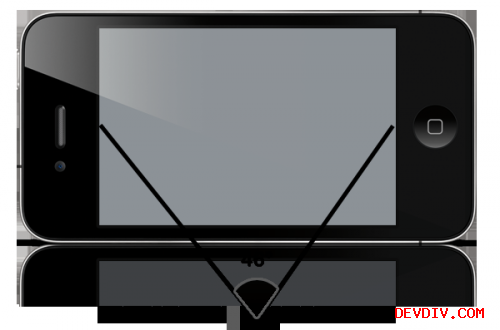
// Determine the maximum position for the enemy ship int rangeMax = positionIn360 + 23; if (rangeMax > 360) { rangeMax = rangeMax - 360; checkAlternateRange = true; }

if (checkAlternateRange) { if ((enemyShip.yawPosition < rangeMax || enemyShip.yawPosition > rangeMin ) || (enemyShip.yawPosition > rangeMin || enemyShip.yawPosition < rangeMax)) { [self updateEnemyShipPosition:positionIn360 withEnemy:enemyShip]; } } else { if (enemyShip.yawPosition > rangeMin && enemyShip.yawPosition < rangeMax) { [self updateEnemyShipPosition:positionIn360 withEnemy:enemyShip]; } } }

这个方法看起来似乎有点让人摸不着头脑，一会最大值，一会儿最小值，一会候选值。不过，不要担心，其实非常简单。首先，我们检查设置的yaw坐标址（positionIn360)，然后把此值限制在0~360之间。



因为，我们有两端范围都是0~360，所以需要检查一下设置的positionIn360具体属于哪一端。我们使用一个任意数23来代表将在屏幕一半处显示的度数。



因为，我们只需要关心变化范围是0~23和337~360的空间了。因为，另一端的线将会包过来。（这里相像成3维空间的一个圆）

最后，如果敌人飞船在屏幕46度的范围之内的话，那么就要更新敌人飞船。checkAlternateRange判断语句只是用来决定什么时候来更新敌人飞船。

如果checkAlternateRange为真的话，那么我们就检查敌船的位置是否落在min和max的范围之内。

positionIn360 = 20

rangeMin = 357

rangeMax = 20

enemyShip.yawPosition = 359

因为我们要考虑线段的两端，我们的min范围比max范围要大一些。现在，我们做一个判断，看敌船的位置是不是大于rangeMin，如果是，则显示在屏幕上。

if语句中的else就更加明了了。他检查敌船的位置是不是大于min且小于max。

多么复杂的一个update方法啊！我们在checkEnemyShipPosition方法下面添加以下代码：

-(void)updateEnemyShipPosition:(int)positionIn360 withEnemy:(EnemyShip \*)enemyShip {

int difference = 0;

if (positionIn360 < 23) {

// Run 1

if (enemyShip.yawPosition > 337) {

difference = (360 - enemyShip.yawPosition) + positionIn360;

int xPosition = 240 + (difference \* kXPositionMultiplier);

[enemyShip setPosition:ccp(xPosition, enemyShip.position.y)];

} else {

// Run Standard Position Check

[self runStandardPositionCheck:positionIn360 withDiff:difference withEnemy:enemyShip];

}

} else if(positionIn360 > 337) {

// Run 2

if (enemyShip.yawPosition < 23) {

difference = enemyShip.yawPosition + (360 - positionIn360);

int xPosition = 240 - (difference \* kXPositionMultiplier);

[enemyShip setPosition:ccp(xPosition, enemyShip.position.y)];

} else {

// Run Standard Position Check

[self runStandardPositionCheck:positionIn360 withDiff:difference withEnemy:enemyShip];

}

} else {

// Run Standard Position Check

[self runStandardPositionCheck:positionIn360 withDiff:difference withEnemy:enemyShip];

}

}

在这个方法中，我们测试，看是否设备的positionIn360是不是在讨论的3个范围内。在第一个测试中，我们检测positionIn360是不理小于23，如果是，我们就看看是不是有一些敌船在线的另一端（大于337）。

第二部分测试，看是否positionIn360大于337.如果是的话，就再检测它是否小于23.

第二部分测试，看敌船是否在23和337之间。如果是，则调用runStandardPositionCheck方法。这个方法的定义如下所示：

-(void)runStandardPositionCheck:(int)positionIn360 withDiff:(int)difference withEnemy:(EnemyShip \*)enemyShip {

if (enemyShip.yawPosition > positionIn360) {

difference = enemyShip.yawPosition - positionIn360;

int xPosition = 240 - (difference \* kXPositionMultiplier);

[enemyShip setPosition:ccp(xPosition, enemyShip.position.y)];

} else {

difference = positionIn360 - enemyShip.yawPosition;

int xPosition = 240 + (difference \* kXPositionMultiplier);

[enemyShip setPosition:ccp(xPosition, enemyShip.position.y)];

}

}

在这个方法中，我们检查看是否enemyShip是否在设备的positionIn360的左边还是右边。当enemyShip的位置小于positionIn360时，它将出现在屏幕的左边。当enemyShip的位置大于positionIn360，那么它将出现在屏幕的右边。

现在，你会说，请等一下！你忘了描述这些变量的作用了！好吧，接下来就解释一下。

如果敌船的yaw坐标值在屏幕的范围之内（从 positionIn360 – 23到 positionIn360 + 23），然后，首先我们计算它位于屏幕的哪一边。如果大于positionIn360，那么就在屏幕的右边，否则就在屏幕的左边。

difference变量的作用是保存设备的positionIn360和敌船的 yaw位置的角度差值。一旦计算出来后，我们就把这个差值乘以一个任意的倍数。这个倍数代表每一度的像素个数。这个里，我们选择15.

基于它位于于当前屏幕的位置，我们将把这个计算出来的值增加240或者减去240。

现在，所有需要的方法已经全部准备就绪啦。

在init方法的底部，添加下面的代码，在屏幕中增加5个飞船：

// Loop through 1 – 5 and add space ships

enemySprites = [[NSMutableArray alloc] init];

for(int i = 0; i < 5; ++i) {

EnemyShip \*enemyShip = [self addEnemyShip:i];

[enemySprites addObject:enemyShip];

enemyCount += 1;

}

因为，我们添加了敌船到屏幕中了，我们确保它们的位置被更新。在udpate方法的底部添加下面方法：

// Loop through array of Space Ships and check the position

for (EnemyShip \*enemyShip in enemySprites) {

[self checkEnemyShipPosition:enemyShip withYaw:yaw];

}

在我们忘记之前，在dealloc里面添加清理代码来清理我们之前创建的enemySpritesArray数组：

[enemySprites release];

现在，编译并运行工程吧！当你旋转设备的时候，你将会看到5个飞船在不同的地方。



**免费的激光和爆炸**

目前为止，这个现实增加的游戏完成的差不多了。不过，还有一个很严重的问题：这里飞船打中后没什么感觉。

很明显，我们并不想这样，所以，让我们添加一些很酷的激光和爆炸效果吧。

在开始之前，让我们移除屏幕上的label–他们只是作为调试时用的。因此，找开 HelloWorldLayer.m中关于labels的代码，并把它们注释掉。完成之后，编译并运行，保证没有错误。

现在，看看有趣的部分—让我们往游戏中添加一些火力吧！首先，我们将添加一个方法用来判断玩家的开火区域是否击中了飞船。在HelloWorldLayer.h文件中，在@end之前声明下列方法：

- (BOOL) circle:(CGPoint) circlePoint withRadius:(float) radius collisionWithCircle:(CGPoint) circlePointTwo collisionCircleRadius:(float) radiusTwo;

打开HelloWorldLayer.m，然后在dealloc方法上面实现上述方法：

- (BOOL) circle:(CGPoint) circlePoint withRadius:(float) radius collisionWithCircle:(CGPoint) circlePointTwo collisionCircleRadius:(float) radiusTwo {

float xdif = circlePoint.x - circlePointTwo.x;

float ydif = circlePoint.y - circlePointTwo.y;float distance = sqrt(xdif\*xdif+ydif\*ydif);

if(distance <= radius+radiusTwo) return YES;

return NO; }

这个方法用来检测是否两个点的半径有交集。输入的参数是敌方飞船位置和屏幕的中心点位置。两个点的半径都设置为50.

首先，我们计算两个点X和Y值的差。接下来计算两点的距离。这个在高中就学过的，叫勾股定理。你可以从[这里](http://www.cliffsnotes.com/study_guide/Distance-Formula.topicArticleId-18851,articleId-18841.html)得到更多的信息。

接下来，我们往屏幕中添加一个区域，用作火力瞄准器。下载[这些资源文件](http://www.raywenderlich.com/downloads/ARSpaceshipsResources.zip)，解压缩，然后把scope.png拖到Resouces文件夹下去。确保“Copy items into destination group’s folder”被复选中，然后点Finish。

打开HelloWorldLayer.m的init方法，然后在 [self scheduleUpdate]方法之前，添加以下代码：

// Add the scope crosshairs

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

scope.position = ccp(240, 160);

[self addChild:scope z:15];// Allow touches with the layer

[self registerWithTouchDispatcher];

如果你现在运行程序，你将看到一个瞄准器出现在屏幕的中间。



非常好，现在让我们添加一些爆炸效果，在玩家点击屏幕的时候就触发。我们将按照添加scope.png的方法一样，来添加Explision.plist。先找到之前下载的[本项目资源文件](http://www.raywenderlich.com/downloads/ARSpaceshipsResources.zip)。把Explosion.plist拖到资源文件夹中，确保“Copy items into destination group’s folder”被复选上，然后点击Finish。

你可能会奇怪这个文件到底是什么？我使用一个很酷的软件制作的，你可能之前也听说过了，叫做 [Particle Designer](http://particledesigner.71squared.com/)，它是由71 Squared的工程师所开发的。我不会在这里讲解如何使用此软件来制作这样的粒子效果文件，但是，实际上这个过程是非常简单的。选择一种粒子效果，然后调节一些参数，最后导出plist文件就可以了。

现在，在dealloc方法之前，添加下列代码：

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

CGPoint location = CGPointMake(240,160);// 1

for (EnemyShip \*enemyShip in enemySprites) {

if (enemyShip.timeToLive > 0) {

// Check to see if yaw position is in range

BOOL wasTouched = [self circle:location withRadius:50 collisionWithCircle:enemyShip.position collisionCircleRadius:50];

if (wasTouched) { enemyShip.timeToLive = 0; enemyShip.visible = false; enemyCount -= 1; } } }

// 2 CCParticleSystemQuad \*particle = [CCParticleSystemQuad particleWithFile:@"Explosion.plist"]; particle.position = ccp(240,160); [self addChild:particle z:20]; particle.autoRemoveOnFinish = YES;

// 3 if (enemyCount == 0) { // Show end game CGSize winSize = [CCDirector sharedDirector].winSize; CCLabelBMFont \*label = [CCLabelBMFont labelWithString:@"You win!" fntFile:@"Arial.fnt"]; label.scale = 2.0; label.position = ccp(winSize.width/2, winSize.height/2); [self addChild:label z:30]; } }

这段代码的第一部分用来做碰撞检测，用来测试是否飞船在瞄准器范围之内。如果其中一个飞船被击中了，我们就飞船的属性来隐藏它，同时把enemyCount计数减1.第二部分代码，往屏幕中心添加了一个爆炸粒子系统。第二部分代码，也是最后一部分代码，它用来检查enemyCount是否为0，如果是的话，就显示一个label，告知玩家游戏结束了。

这个游戏如果就这样的话，有点无聊。所以，让我们往游戏中添加一些基本的AI吧。其实也很简单的，就是随着时间的推移，我们会改变一下飞船的位置。所以，在update方法底部添加下列代码：

// Loop through array of Space Ships and if the timeToLive is zero

// change the yawPosition of the sprite

for (EnemyShip \*enemyShip in enemySprites) {

enemyShip.timeToLive–;

if (enemyShip.timeToLive == 0) {

int x = arc4random() % 360;

[enemyShip setPosition:ccp(5000, 160)];

enemyShip.yawPosition = x;

enemyShip.timeToLive = kTimeToLive;

}

}

这段代码将会遍历所有的enemySprites，然后更新timeToLive属性。然后，检查这个timeToLive属性是否等于0，如果是的话，那么就给飞船的yawPositon设置一个随机值，同时重置timeToLive属性。编译并运行工程吧，现在你想要打中飞船的话就有一些难度了，开火！

**Pump up the Volume!**

游戏如果没有声音的话，那就太没意思了！所以，让我们添加一些音乐吧！

在HellowWorldLayer.m文件顶部包含Simple Audio Engine所需的头文件，具体如下所示：

#import ”SimpleAudioEngine.h”

然后在init方法的最后添加下列代码，记得添加在 if ((self=[super init]))语句内部：

[[SimpleAudioEngine sharedEngine] playBackgroundMusic:@”SpaceGame.caf” loop:YES];

[[SimpleAudioEngine sharedEngine] preloadEffect:@”explosion\_large.caf”];

[[SimpleAudioEngine sharedEngine] preloadEffect:@”laser\_ship.caf”];

这里会加载背景音乐，同时预加载音效。

现在，找到ccTouchesBegan，然后在这个方法的开头添加下列代码：

[[SimpleAudioEngine sharedEngine] playEffect:@”laser\_ship.caf”];

这里会在你点击屏幕的时候播放一个发射激光的音效。

还是在ccTouchesBegan方法里面，打开遍历enemySprites那个循环，然后在 (wasTouched)的if判断句内添加下列代码：

[[SimpleAudioEngine sharedEngine] playEffect:@”explosion\_large.caf”];

当飞船被打中的时候，将会播放爆炸的音效！

编译并运行代码，尽情玩吧！

**何去何从?**

这里有本教程的[完整源代码](http://www.raywenderlich.com/downloads/ARSpaceships.zip)。

如果你想学习更多有关制作增强现实的游戏的话，下面有一些比较好的资源：

* 更多关于摄像头的纵横比的:[1](http://stackoverflow.com/questions/3407986/uiimagepickercontroller-cameraviewtransform-acts-differently-in-ios-4) [2](http://gotoandplay.freeblog.hu/archives/2010/07/06/3Dcompass_augmented_reality_085_-_fullscreen_camera_preview/) [3](http://www.musicalgeometry.com/?p=821)
* <a href=”http://developer.apple.com/library/ios/#documentation/uikit/reference/UIImagePickerController\_Class/UIImagePickerController/UIImagePickerController.html “>Apple’s documentation on UIImagePickerController Class

[Apple’s documentation on Core Motion](http://developer.apple.com/library/ios/#documentation/CoreMotion/Reference/CoreMotion_Reference/_index.html)