# 第4章 Cocos2D中的动作、特效与动画

第3章对Cocos2D几个核心类进行了更深入的介绍，这些核心类是Cocos2D的基础，后续所学习的都会在此基础之上进行。现在可以正式学习Cocos2D中更加实用，也更加有意思的部分。

本章示例项目基于第3章代码，引入各种不同类型Action，使读者对Cocos2D内置的节点动作有更进一步的认识。此外，本章还将学习Cocos2D提供的一些特效，以及如何利用代码实现游戏角色的动画效果。

## Cocos2D中的节点动作

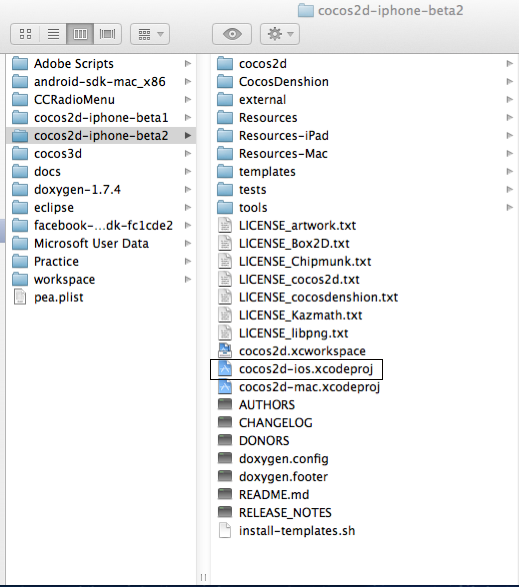
任何一个游戏中的角色都不会在原地一动不动，使用动作可以让玩家更好地产生游戏中的参与感，并沉浸其中。可以把节点动作想象成给所有CCNode节点对象下达的命令。这些动作可以修改节点对象的各种属性，如position（位置）、rotation（旋转）、scale（比例）等。如果这些属性是经过一段时间之后修改的，属于CCActionInterval（区间动作），否则属于CCActionInstant（瞬时动作）。

区间动作提供一些有意思的特性，比如可以随着时间改变而加速，而且所有的相对动作（动作名称后面加By）和部分绝对动作（动作名称后面加To）都有自己的反转动作。

无论区间动作还是瞬时动作，都属于CCFiniteTimeAction（有限时间动作），Cocos2D还支持CCRepeatForever（无限重复动作）、CCFollow（跟随节点的动作），以及CCSpeed（用于更改节点动作的速度）。关于各类节点动作的继承关系，可以参考官方文档http://www.cocos2d-iphone.org/api-ref/2.0.0/interface\_c\_c\_action.html。

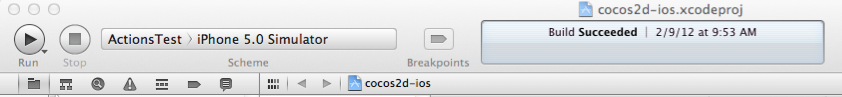
### 运行ActionsTest测试

在Cocos2D的安装目录下找到cocos2d-ios.xcodeproj项目，双击该文件以使用Xcode打开该项目，如图4-1所示。



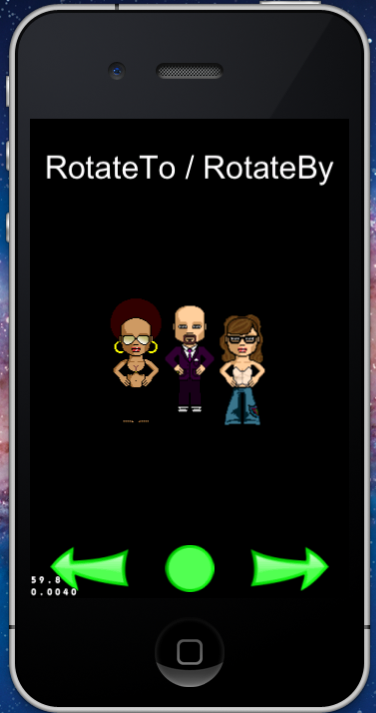
1. 在Cocos2D安装目录下找到cocos2d-ios.xcodeproj

在Xcode中将Target设置为ActionsTest，然后单击“Run”按钮查看Cocos2D中提供的动作效果，如图4-2所示。



1. 运行ActionsTest测试

此时，在模拟器或设备中将展示模板中所提供的各种动作及动作组合，单击或触碰屏幕下角的绿色箭头，切换查看不同动作效果，如图4-3所示。

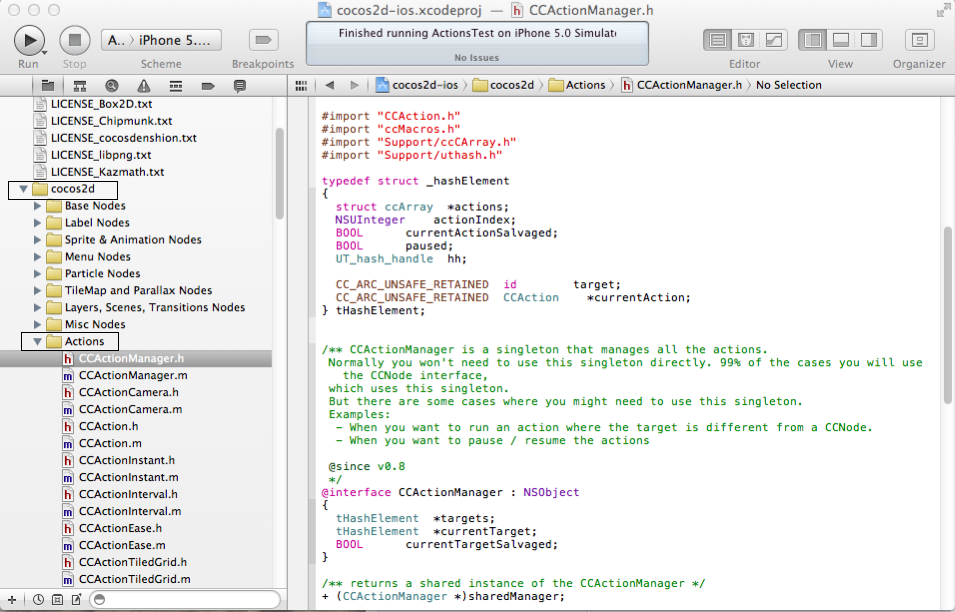


1. 模拟器中显示的动作效果

运行ActionsTest示例，可以对Cocos2D中所支持的节点动作效果有一个直观的认识，建议读者认真研读每个Action的用法。

### Cocos2D中的动作类

在学习具体的动作前，首先需要对Cocos2D提供的动作类有一个整体的认识。在Xcode中打开cocos2d-ios.xcodeproj项目，在左侧找到cocos2d目录，单击下三角图标展开，找到Actions目录，单击下三角图标展开，就可以看到Cocos2D中和动作相关的类，如图4-4所示。



1. 在Xcode中查看与动作相关的类

可以看到，Cocos2D中和动作相关的类很多，限于篇幅，这里只对几个重要的类加以简单的介绍。

##### CCActionManager

管理所有节点动作的对象。

CCActionManager在v2.0 之前还属于单例，现在属于CCDirector的实例变量。通常情况下，开发者不会在代码中直接用到该对象。但在某些特殊情况下会用到它，比如对某个非CCNode节点执行动作时。此外，也使用CCActionManager暂停/继续执行所有的动作。

看一个CCActionManager的应用实例。首先在Xcode中打开HelloWorldLayer.m文件，添加一个onEnter方法，同时把添加开始游戏菜单的代码从init方法移至onEnter方法中。如代码清单4-1所示。

代码清单4-1 添加一个onEnter方法

-(void) onEnter{

[super onEnter];

CGSize winSize = [CCDirector sharedDirector].winSize;

[CCMenuItemFont setFontSize:20];

[CCMenuItemFont setFontName:@"Arial"];

CCMenuItemFont \*startItem = [CCMenuItemFont itemWithString:@"开始游戏" block:^(id sender)

{

\_isGameStarted = YES;

CCMenuItem \*item = (CCMenuItemFont\*)sender;

item.visible = NO;

//6.spawn enemy after 1.0 sec

[self performSelector:@selector(spawnEnemy)

withObject:nil

afterDelay:1.0f];

//7.enable accelerometer

self.isAccelerometerEnabled = YES;

//9.enable touch

self.isTouchEnabled = YES;

}];

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

\_startGameMenu = [CCMenu menuWithItems:startItem, nil];

\_startGameMenu.position = CGPointZero;

[self addChild:\_startGameMenu];

//1.add moveBy action to startGameMenu

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

[\_startGameMenu runAction:moveBy];

//2.pause \_startGameMenu action

[[[CCDirector sharedDirector] actionManager] pauseTarget:\_startGameMenu];

//3.resume \_startGameMenu action after 1 sec

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

}

这里把菜单初始化在屏幕下方非显示区域内，然后定义一个moveBy动作，让此菜单在一秒之后移动到屏幕的中点位置。但是我们并不希望这个动作立即执行，而是过1秒之后再触发，这时CCActionManager就派上用场了。如代码清单4-1里标号为2的代码片断所示，调用actionManager的pauseTarget方法暂停菜单的动作。然后触发一个selector，每隔1秒调用一次resumeStartMenuAction方法。

resumeStartMenuAction方法的实现如代码清单4-2所示。

代码清单4-2 resumeStartMenuAction方法的实现

-(void) resumeStartMenuAction:(ccTime)dt{

[self unschedule:\_cmd];

[[[CCDirector sharedDirector] actionManager] resumeTarget:\_startGameMenu];

}

这段代码很简单，首先取消定时回调的selector，然后调用CCActionManager的resumeTarget方法以恢复动作的运行。此时编译并运行，当加载完资源以后，需要等待1秒钟然后才触发开始菜单的moveBy动作。这里需要注意，pauseTarget方法的调用一定要放在onEnter函数里，不能放在init方法里。同时，也不能放在[super onEnter]之前。读者可以自行去验证一下。

##### CCAction

该类属于抽象类，几乎所有和动作相关的类都继承自该类。

##### CCFiniteTimeAction

有限时间动作，包含CCActionInstant（瞬时动作）和CCActionInterval（区间动作）。而CCActionInstant和CCActionInterval中又包含了多个不同的动作。

##### CCRepeatForever

无限重复动作。

##### CCFollow

跟随节点的动作。此动作配合CCParallaxNode可以实现视差滚动背景效果。如果大家运行ActionTest中关于CCFollow的例子会发现，其实是移动Camera的一种替代方案。而且，Cocos2D的作者也推荐以此取代直接操作Camera。

##### CCSpeed

更改节点动作的速度。

##### CCOrbitCamera

继承自CCActionCamera，使用球坐标系围绕屏幕中心旋转摄像机的视角。

## 动作的分类及使用

接下来将对各种动作进行详细的介绍。

### 基本动作

首先学习简单动作。简单动作可以更改节点的基本属性，如位置等。为了让读者更好地了解每种动作的实际效果，除了参考Cocos2D官方模板的ActionTest.m文件，还可以参考本章的示例项目BasicActions。

接下来看如何使用简单动作。让精灵对象在2秒内从原来的位置移动到新的位置，如代码清单4-3所示。

代码清单4-3 使用CCMoveTo动作示例

// 定义一个CCAction动作

CCAction \*moveAction = [CCMoveTo actionWithDuration:2.0 position:ccp(size.width/2, 100+size.height/2)];

// 让精灵对象执行该动作

[plane runAction:moveAction];

这行代码虽然简单，我们还是具体解释一下，对于这个精灵来说究竟发生了什么。

首先，所有CCNode节点都支持runAction方法（参考2.1.2节）,而runAction方法使用CCAction对象作为参数代表该节点要执行的动作。

示例中使用CCMoveTo定义一个moveAction动作，然后让精灵对象调用runAction方法执行该动作。当然，正如之前所提到的，CCMoveTo对象的创建是使用类方法实现的。该方法包含两个参数，分别是动作的持续时间，以及在该时间段结束时精灵对象要移动到的位置。

另外，如果开发者足够熟练，也可以直接使用一行代码替代上面的内容，如下所示：

//使用以下代码可以实现同样的效果

[plane runAction:[CCMoveTo actionWithDuration:5.0 position:ccp(size.width/2, 100+size.height/2)]];

Cocos2D中提供了大量的基本动作，而且在每个新的版本中还会添加新动作。以下是开发者经常使用的基本动作。

#### 和位置相关的基本动作

##### CCMoveTo

让节点对象在一定时间内运动到屏幕上的某一点。如下所示：

[plane runAction:[CCMoveTo actionWithDuration:5.0 position:ccp(size.width/2, 100+size.height/2)]];

使用该动作，精灵对象将在5秒内从当前位置移动到屏幕中心位置靠上100像素的地方。

##### CCMoveBy

让节点对象在一定时间内从原位置相对移动一定的像素值。如下所示：

[plane runAction:[CCMoveBy actionWithDuration:5.0 position:ccp(size.width,0)]];

使用该动作，精灵对象将在5秒内从原位置向右移动相当于屏幕宽度的像素，其高度保持不变。

##### CCJumpTo

让节点对象在一定时间内沿抛物线移动到屏幕的某一特定位置。如下所示：

[ball runAction:[CCJumpTo actionWithDuration:5.0 position:ccp(size.width-ball.contentSize.width, size.height/3) height:50 jumps:3]];

使用该动作，精灵对象将在5秒内从原位置以3次跳跃动作移动到屏幕的某一位置。读者可以通过自行修改height和jump参数，看看精灵的动作会发生什么变化。

##### CCJumpBy

让节点对象在一定时间内沿抛物线移动到屏幕的某一特定位置，和CCJumpTo的区别在于节点对象的位移是相对的。如下所示：

[ball runAction:[CCJumpBy actionWithDuration:5.0 position:ccp(200,100) height:50 jumps:5]];

使用该动作，精灵对象将在5秒内从原位置以5次跳跃动作移动到屏幕的某一位置，且该位置相对于初始位置的偏移是（200，100）。

##### CCBezierTo

让节点对象在一定时间内沿3次贝塞尔曲线移动到屏幕的目标位置。如下所示：

ccBezierConfig c = {ccp(300, 200),ccp(50,50),ccp(-50,-50)};

[ball runAction:[CCBezierTo actionWithDuration:3.0 bezier:c]];

以上代码使用ccBezierConfig定义一个结构体，其中第一个坐标是目标位置，第二和第三个坐标分别是贝塞尔曲线的两个控制点。关于贝塞尔曲线这里不做详细的解释，可以自行修改目标位置和控制点的坐标，看看精灵的动作会发生什么变化。

##### CCBezierBy

让节点对象在一定时间内沿3次贝塞尔曲线移动到屏幕的目标位置。如下所示：

ccBezierConfig c = {ccp(200, 100),ccp(50,50),ccp(-50,-50)};

[ball runAction:[CCBezierBy actionWithDuration:3.0 bezier:c]];

以上代码使用ccBezierConfig定义一个结构体，其中第一个坐标是目标位置相对原位置的偏移，第二和第三个坐标分别是贝塞尔曲线的两个控制点。

##### CCPlace

将节点对象直接放置在所需的位置。

对于直接设置CCNode的position属性有什么好处呢？主要作用就是可以结合CCSequence动作组成动作链。在完成一个动作之后，如果想设置该对象的positon就可以用CCPlace。如下所示：

[ball runAction:[CCPlace actionWithPosition:ccp(200,200)]];

#### 和大小相关的基本动作

##### CCScaleTo

节点对象在一定时间内放大到某个特定的比例大小。例如：

精灵对象在2秒内放大到原来的2倍大小：

[ball runAction:[CCScaleTo actionWithDuration:2.0 scale:2]];

精灵对象的高度在2秒内放大到原来的2倍：

[ball runAction:[CCScaleTo actionWithDuration:2.0 scaleX:1 scaleY:2]];

##### CCScaleBy

节点对象在一定时间内缩小到某个特定的比例大小。例如：

精灵对象在2秒内缩小到原来的0.5倍：

[ball runAction:[CCScaleBy actionWithDuration:2.0 scale:0.5]];

精灵对象的宽度和高度都在2秒内缩小到原来的0.5倍：

[ball runAction:[CCScaleBy actionWithDuration:2.0 scaleX:0.5 scaleY:0.5]];

#### 和旋转相关的基本动作

##### CCRotateTo

设定节点对象的角度属性，让节点对象在一定时间内旋转到某个特定的角度。例如精灵对象在2秒内会旋转到指定的角度：

[ball runAction:[CCRotateTo actionWithDuration:2.0 angle:270]];

需要注意的是，精灵对象的旋转方向是根据角度大小决定的，如果在180度以内会顺时针旋转，反之则逆时针旋转。所以使用刚才的代码，精灵对象会逆时针旋转。

##### CCRotateBy

更改节点对象的角度属性，让节点对象在一定时间内旋转到某个特定的角度。

例如精灵对象在2秒内会旋转到指定的角度：

[ball runAction:[CCRotateBy actionWithDuration:2.0 angle:100]];

实际运行代码会发现，在默认情况下，CCRotateTo和CCRotateBy的效果似乎没有区别。实际上，当节点对象的角度属性并发处于默认值时，两种动作就会显示出不同的效果。

#### 和显示相关的基本动作

##### CCShow

立即显示节点对象。如下所示：

[ball runAction:[CCShow action]];

##### CCHide

立即隐藏节点对象。如下所示：

[ball runAction:[CCHide action]];

##### CCToggleVisibility

切换节点对象的可视属性。如下所示：

[ball runAction:[CCToggleVisibility action]];

注意 一般情况下，上述三个动作需要配合其他动作产生特定的显示效果。

##### CCBlink

让节点对象在一定时间内闪动某个特定的次数。例如让精灵对象在5秒内闪动10次：

[ball runAction:[CCBlink actionWithDuration:5.0 blinks:10]];

前面示例中，当敌机击中玩家飞机时，也使用了CCBlink。

#### 和透明度相关的基本动作

##### CCFadeIn

让节点对象在一定时间内淡入（将其透明度从0调整为255）。例如让精灵对象在3秒内淡入：

[ball runAction:[CCFadeIn actionWithDuration:3.0]];

##### CCFadeOut

让节点对象在一定时间内淡出（将其透明度从255调整为0）。例如让精灵对象在3秒内淡出：

[ball runAction:[CCFadeOut actionWithDuration:3.0]];

##### CCFadeTo

在一定时间内修改节点对象的透明度到某一特定的数值。如下所示：

[ball runAction:[CCFadeTo actionWithDuration:3.0 opacity:64]];

#### 和色彩相关的基本动作

##### CCTintTo

在一定时间内将节点着色到某一特定的RGB色彩值。如下所示：

[ball runAction:[CCTintTo actionWithDuration:5.0 red:200 green:150 blue:100]];

##### CCTintBy

在一定时间内根据某一特定的RGB色彩值将节点着色。如下所示：

[ball runAction:[CCTintBy actionWithDuration:5.0 red:200 green:150 blue:100]];

#### 和翻转相关的基本动作

##### CCFlipX

让节点对象沿水平方向翻转。如下所示：

[plane runAction:[CCFlipX actionWithFlipX:YES]];

##### CCFlipY

让节点对象沿垂直方向翻转。如下所示：

[plane runAction:[CCFlipY actionWithFlipY:YES]];

以上就是目前版本（v2.0）可用的所有基本动作。

### 组合动作

在实际开发游戏的过程中，很多时候会将多个基本动作组合使用，让游戏效果更加灵活，更加吸引用户。想更好地了解每种动作的实际效果，可以参考Cocos2D官方模板的ActionTest.m文件，以及本章示例项目CompositionActions。下面通过一个实例看看如何使用组合动作。

例如让精灵对象淡出的同时比例变为原来的一半：

// 定义一个CCAction动作

CCAction \*action = [CCSpawn actions:[CCFadeOut actionWithDuration:5.0], [CCScaleTo actionWithDuration:5.0 scale:0.5], nil];

// 让精灵对象执行该动作

[mySprite runAction:action];

这个示例中使用组合动作中的CCSpawn。使用该动作，可以让节点对象同时执行若干个动作，比如该例中的CCFadeOut和CCScaleTo。

在初始化该动作时，需要传递进所有需要执行的动作。该动作列表可以包含多个动作，甚至可以包含其他的组合动作。

Cocos2D中支持4种不同的组合动作，使用这些组合动作可以将不同的基本动作组合在一起。实际上，所有能想象的效果都可以用基本动作和组合动作实现。这些动作如下。

#### CCSpawn

使用该组合动作可以让节点同时执行多个动作。例如，让精灵对象在2秒内淡入并放大到原大小的2 倍：

CCAction \*action = [CCSpawn actions:[CCFadeIn actionWithDuration:2],[CCScaleTo actionWithDuration:2 scale:2], nil];

[mySprite runAction:action];

除了这种标准形式，CCSpawn还支持该方法的一种变化形式。如下所示：

NSArray \*actions = [NSArray arrayWithObjects:[CCFadeIn actionWithDuration:2],[CCScaleTo actionWithDuration:2 scale:2], nil];

CCAction \*action =[CCSpawn actionsWithArray:actions];

[mySprite runAction:action];

以上代码首先定义一个NSArray对象，并使用arrayWithObjects方法和两个基本动作初始化该对象。接着使用CCSpawn的actionsWithArray方法定义一个动作。最后让精灵对象执行该动作。CCSpawn组合动作的时间间隔是数组里时间最长的Action时间间隔。

提示 当CCSpawn中包含的基本动作比较多时，建议采用这一方法。

接下来看如何使用CCSpawn。在之前的射击游戏中，玩家击中敌机时，会直接把enemySprite的visible属性设置为NO。这里使用CCSpawn组合动作，结合CCScale、CCRotate、CCHide、CCFadeOut这4个动作让敌机以一种比较炫的方式退场。

1. 找到碰撞检测代码，在collisionDetection函数里修改enemy.visible = NO为下列代码：

id ac1 = [CCScaleTo actionWithDuration:1.0 scale:1.2];

id ac2 = [CCRotateBy actionWithDuration:1.0 angle:720];

id ac3 = [CCFadeOut actionWithDuration:1.0];

id ac4 = [CCHide action];

id ac5 = [CCSequence actions:ac3,ac4, nil];

id action = [CCSpawn actions:ac1,ac2,ac5, nil];

[enemy stopAllActions];

[enemy runAction:action];

注意 这里把[enemy stopAllActions]的位置提前了，之前放在if碰撞检测判断最后。具体可以参考示例代码。同时提前引用了CCSequence类，主要是把一系列Action串起来，组合成一个按时序运行的Action。

读者可能会有疑问，为什么用了CCFadeOut了，还需要CCHide动作呢？因为，CCFadeOut只是把CCNode的透明度从255改到0，而visible属性还是没变。这时编译运行会发现有个bug。那就是，敌机在消失的时候，是不能再做碰撞检测的，也不应该加分。

1. 修改碰撞检测的代码。

只有当敌机的opacity为255时才做碰撞检测。同时需要修改getAvailableEnemySprite方法，如代码清单4-4所示。

代码清单4-4 修改getAvailableEnemySprite方法

-(CCSprite\*) getAvailableEnemySprite{

CCSprite \*result = nil;

CCARRAY\_FOREACH(\_enemySprites, result)

{

if (!result.visible) {

result.opacity = 255;

break;

}

}

return result;

}

1. 最后在判断敌机碰撞检测时添加对透明度的判断语句。示例代码：

if (enemy.visible && enemy.opacity == 255) {

//下面是省略掉了碰撞检测的代码

#### CCSequence

使用该组合动作可以按顺序执行多个动作。例如精灵对象在2秒内淡入，然后在2秒内放大到原来的2倍：

CCAction \*action = [CCSequence actions:[CCFadeIn actionWithDuration:2],[CCScaleTo actionWithDuration:2 scale:2], nil];

[mySprite runAction:action];

和CCSpawn一样，用以下代码也可以实现相同的效果：

NSArray \*actions = [NSArray arrayWithObjects:[CCFadeIn actionWithDuration:2],[CCScaleTo actionWithDuration:2 scale:2], nil];

CCAction \*action = [CCSequence actionsWithArray:actions];

[mySprite runAction:action];

和CCSpawn一样，CCSequence的动作序列中也可以包含其他的组合动作。在实际的开发过程中，会频繁使用CCSequence组合动作。

#### CCRepeat

使用该组合动作让节点在限定的时间内重复执行某个动作。如下所示：

id action = [CCSequence actions:[CCFadeIn actionWithDuration:2],[CCBlink actionWithDuration:1 blinks:3], nil];

[mySprite runAction:[CCRepeat actionWithAction:action times:3]];

上面的代码略显复杂。首先用id类型定义一个动作，而不是之前默认的CCAction，。因为CCRepeat支持的动作必须是CCFiniteTimeAction，即一种特殊的CCAction（其他3个CCAction的子类是CCFollow、CCRepeatForever和CCSpeed）。所以实际上这行代码等同于以下代码：

CCFiniteTimeAction \*action = [CCSequence actions:[CCFadeIn actionWithDuration:2],[CCBlink actionWithDuration:1 blinks:3], nil];

该动作是一个序列动作，首先在2秒内淡入，然后在1秒内闪动3次。

接着，使用CCRepeat让精灵对象重复执行该动作3次。

#### CCRepeatForever

使用该组合动作让节点反复执行某个动作。例如精灵对象每2秒向右移动100像素，并持续移动：

id action = [CCMoveBy actionWithDuration:2 position:ccp(100,0)];

[mySprite runAction:[CCRepeatForever actionWithAction:action]];

注意 CCSequence的序列中不能包含此组合动作。

### Ease动作

CCActionEase是区间动作的一个子类，属于特殊的组合动作。使用Ease动作可以修改节点对象内部动作的时间。

实际上，使用Ease动作修改的是内部动作的速度，而非动作执行的时间。所以，一个1秒完成的动作仍然在1秒内完成，但内部动作的速度却可以大不相同。使用Ease类型的动作可以获得一些类似物理仿真的效果。

举例而言，CCMoveTo动作可以让某个精灵对象在5秒内从某一点移动到另一点。动作执行时，精灵会以一种线性的方式朝目标移动，也就是说匀速运动。

而实际的物体很少匀速前进的。因此，使用一个Ease动作（比如EaseInCubic动作），让精灵对象缓慢启动，并逐渐加速。

感兴趣的读者可以参考Cocos2D官方模板的ActionTest.m文件，以及本章的示例项目EaseActions。

Cocos2D中提供了很多Ease动作，本节将学习如何使用其中的某些动作。

例如让精灵在2秒内以CCEaseBounceOut效果移动到指定位置，然后放大到原大小的2倍，如下所示：

id easeAction = [CCEaseBounceOut actionWithAction:[CCMoveTo actionWithDuration:2 position:ccp(100+size.width/2,size.height/2)]];

id action = [CCSequence actions:easeAction,[CCScaleTo actionWithDuration:2 scale:2], nil];

[mySprite runAction:action];

以上代码首先定义一个CCEaseBounceOut动作，然后定义一个CCSequence组合动作。该组合动作会先执行刚才定义的Ease动作，然后执行一个缩放动作。最后让精灵对象执行该组合动作。

Cocos2D中支持的基本Ease动作如下：

* CCEaseBounce、CCEaseBounceIn、 CCEaseBounceInOut、CCEaseBounceOut
* CCEaseElastic、CCEaseElasticIn、CCEaseElasticInOut、CCEaseElasticOut
* CCEaseRateAction 、CCEaseIn、CCEaseInOut、CCEaseOut
* CCEaseExponentialIn、CCEaseExponentialInOut、CCEaseExponentialOut
* CCEaseSineIn、CCEaseSineInOut、CCEaseSineOut

其中CCEaseBounce、CCEaseElastic、CCEaseRateAction都是抽象类，具体实现3种动作。

关于每种动作的具体效果，可以使用模板中的ActionsTest测试查看。

接下来，结合之前的代码示例，用几个随机Ease Action修饰敌机向下俯冲的动作。在HelloWorldLayer.m中找到spawnEnemy方法的代码片断2，修改成代码清单4-5所示代码。

代码清单4-5 修饰敌机向下俯冲的动作代码片断2

//2.

float durationTime = arc4random() % 4 + 1;

id moveBy = [CCMoveBy actionWithDuration:durationTime

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

int choice = arc4random() % 4 + 1;

id easedMoveBy = nil;

switch (choice) {

case 1:

easedMoveBy = [CCEaseIn actionWithAction:moveBy];

break;

case 2:

easedMoveBy = [CCEaseExponentialIn actionWithAction:moveBy];

break;

case 3:

easedMoveBy = [CCEaseBounceIn actionWithAction:moveBy];

break;

case 4:

easedMoveBy = [CCEaseBackIn actionWithAction:moveBy];

break;

default:

break;

}

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

{

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

sp.visible = NO;

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

CCLOG(@"reset enemy plane!");

}];

CCSequence \*action = [CCSequence actions:easedMoveBy,callback, nil];

编译并运行，感受一下Ease动作的强大吧。怎么样，现在敌机往下俯冲不再是均速运动，而是变速运动，更有游戏的刺激体验了。

### 延迟动作

程序中有时不需要立即执行某个动作，而希望先延迟一段时间再执行。这时可以用到Cocos2D中提供的一种特殊动作CCDelayTime（延迟动作）。

CCDelayTime的使用很简单，例如：

id action = [CCSequence actions:[CCDelayTime actionWithDuration:3],[CCMoveTo actionWithDuration:2 position:ccp(100+size.width/2,size.height/2)],nil];

[mySprite runAction:action];

以上代码定义一个序列组合动作，首先延迟3秒，然后让精灵对象执行CCMoveTo动作。

该部分内容的代码请参考本书本章的示例项目DelayActions。

### 方法回调动作

Cocos2D中提供一些特殊动作，让节点对象执行这些动作时，不会直接修改节点的属性，而调用某个特定方法。方法回调动作包括CCCallFunc、CCCallFuncN、CCCallFuncND和CCCallFuncO。

提示 N代表Node，即节点；D代表Data，即数据；O代表Object，即对象。

这4个动作所做的事情基本相同，即从该动作调用一个方法（以selector选择器的形式）。区别在于：

* CCCallFunc最简单，只调用所需要的方法。
* CCCallFuncN在调用所需要方法的同时会传递一个参数，指定要调用动作的节点对象。
* CCCallFuncND在CCCallFuncN基础上再传递另一个参数，通常是用户自定义的数据。
* CCCallFuncO在CCCallFunc基础上传递一个参数对象。

通常情况下，可以在序列动作的结尾执行此类动作，以便调用某个特定的方法。

为了理解这些动作的实际作用，还是通过实例说明。参考本书示例项目CallbackActions。

1. 在Xcode创建一个新项目。

选择iOS-cocos2d模板，创建并命名为CallbackActions。

1. 加载资源图片。

将准备好的资源图片（chapter4/resource/arts/CallbackActions）拖动到Xcode的Resources中，记住，必须选中“Copy items into destination group’s folder(if needed)”。

1. 添加实例变量的定义。

切换到HelloWorldLayer.h，在类声明中添加实例变量的定义，如代码清单4-6所示。

代码清单4-6 在类声明中添加实例变量的定义

@interface HelloWorldLayer : CCLayer<GKAchievementViewControllerDelegate, GKLeaderboardViewControllerDelegate>

{

CCSprite \*ball;

CCSprite \*apple;

CCSprite \*plane;

CCSprite \*orange;

CCSprite \*tomato;

}

1. 添加几个方法。

切换到HelloWorldLayer.m，先在init方法前添加以下几个方法，如代码清单4-7所示。

代码清单4-7 在init方法前添加以下几个方法

-(void)moveSprite{

[ball1 runAction:[CCMoveTo actionWithDuration:2 position:ccp(50,200)]];

}

-(void)removeSprite:(id)sender{

CCNode \*node =(CCNode \*)sender;

[self removeChild:node cleanup:YES];

}

-(void)tintSprite:(id)sender data:(void\*)data{

CCNode \*node = (CCNode \*)sender;

[node runAction:[CCTintBy actionWithDuration:(NSInteger)data red:255 green:0 blue:255]];

}

-(void)rotateSprite:(id)sender object:(id)object {

CCNode \*node = (CCNode\*)object;

[node runAction:[CCSequence actions:[CCFadeIn actionWithDuration:2.0f],

[CCRotateBy actionWithDuration:3.0 angle:180],nil]];

}

其中所添加方法的作用如下：

* moveSprite：精灵对象在2秒内移动到（50，200）。
* removeSprite：在场景中清除传入的节点，并停止该节点的所有动作。
* tintSprite:data：精灵对象在传入参数（data）指定的时间内着色到RGB（255，0，255）。
* rotateSprite:object:方法：指定的精灵对象在2秒内淡入，然后在3秒内旋转180度。

1. 清空init方法。如代码清单4-8所示。

代码清单4-8 把init方法清空后的代码

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

-(id) init

{

// always call "super" init

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

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

//删除这中间的所有代码

｝

return self;

}

1. 创建5个精灵对象。

在注释的代码之后添加以下代码创建5个精灵对象，并将其添加为当前层的子节点如代码清单4-9所示。

代码清单4-9 创建5个精灵对象

// 创建第一个精灵，并将其添加为当前层的子节点

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

apple.position = ccp(50,100);

apple.opacity = 0;

[self addChild:apple];

// 创建第二个精灵，并将其添加为当前层的子节点

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

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

plane.opacity = 0;

[self addChild: plane];

// 创建第三个精灵，并将其添加为当前层的子节点

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

ball.position = ccp(250,100);

ball.opacity = 0;

[self addChild:ball];

// 创建第四个精灵，并将其添加为当前层的子节点

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

orange.position = ccp(250,300);

orange.opacity = 0;

[self addChild:orange];

// 创建第五个精灵，并将其添加为当前层的子节点

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

tomato.position = ccp(250,200);

tomato.opacity = 0;

[self addChild:tomato];

1. 让第一个精灵对象执行序列组合动作。

紧接着上面的代码添加以下代码，让第一个精灵对象执行序列组合动作。其中最后一个动作调用了removeSprite方法，注意冒号不能省略，如代码清单4-10所示。

代码清单4-10第一个精灵对象执行序列组合动作

[apple runAction:[CCSequence actions:[CCDelayTime actionWithDuration:1],

[CCSpawn actions:

[CCFadeIn actionWithDuration:1],

[CCScaleTo actionWithDuration:1 scale:1.5],

nil],

[CCDelayTime actionWithDuration:1],

[CCCallFunc actionWithTarget:self selector:@selector(moveSprite)],

nil]];

1. 让第二个精灵对象执行序列组合动作。

添加以下代码，让第二个精灵对象执行序列组合动作。其中最后一个动作调用了removeSprite:方法，并将当前对象传递给指定的方法，注意冒号不能省略，如代码清单4-11所示。

代码清单4-11 第二个精灵对象执行序列组合动作

[plane runAction:[CCSequence actions:[CCDelayTime actionWithDuration:1],

[CCSpawn actions:

[CCFadeIn actionWithDuration:1],

[CCScaleTo actionWithDuration:1 scale:1.5],

nil],

[CCDelayTime actionWithDuration:1],

[CCCallFuncN actionWithTarget:self selector:@selector(removeSprite:)],

nil]];

1. 让第三个精灵对象执行序列组合动作。

添加以下代码，让第三个精灵对象执行序列组合动作。其中最后一个动作调用了tintSprite:data:方法，并将当前对象和数据传递给指定的方法，注意冒号不能省略，如代码清单4-12所示。

代码清单4-12第三个精灵对象执行序列组合动作

[ball runAction:[CCSequence actions:[CCDelayTime actionWithDuration:1],

[CCSpawn actions:

[CCFadeIn actionWithDuration:1],

[CCScaleTo actionWithDuration:1 scale:1.5],

nil],

[CCDelayTime actionWithDuration:1],

[CCCallFuncND actionWithTarget:self selector:@selector(tintSprite:data:)data:(void \*)2],

nil]];

1. 让第四个精灵对象执行序列组合动作。

添加以下代码，让第四个精灵对象执行序列组合动作，其中最后一个动作调用了rotateSprite:object:方法，并将指定对象（第五个精灵对象）传递给指定的方法，注意冒号不能省略，如代码清单4-13所示。

代码清单4-13让第四个精灵对象执行序列组合动作

[orange runAction:[CCSequence actions:[CCDelayTime actionWithDuration:1],

[CCSpawn actions:

[CCFadeIn actionWithDuration:1],

[CCScaleTo actionWithDuration:1 scale:1.5],

nil],

[CCDelayTime actionWithDuration:1],

[CCCallFuncO actionWithTarget:self selector:@selector(rotateSprite:object:) object:tomato],

nil]];

1. 编译并运行项目。

编译并运行项目，会看到5个精灵对象各自执行不同的组合动作：

其中第一个精灵对象先延迟1秒，同时在1秒内淡入并放大到原大小的1.5倍，再延迟1秒，接着调用moveSprite方法，让小球在2秒内移动到（50，200）这点。

第二个精灵对象会先延迟1秒，同时在1秒内淡入并放大到原大小的1.5倍，再延迟1秒，接着调用removeSprite:方法，从屏幕中清除这个小球并停止它的所有动作。

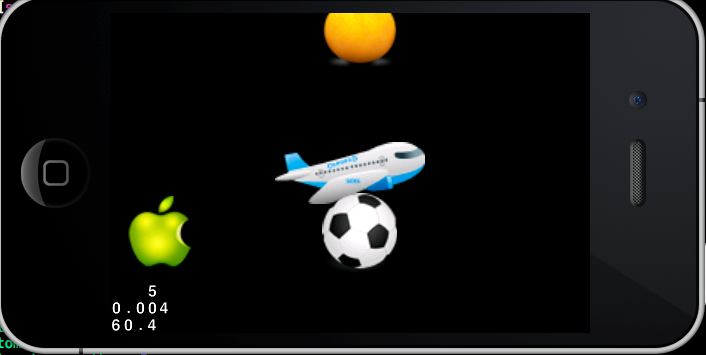
第三个精灵对象会延迟1秒，同时在1秒内淡入并放大到原大小的1.5倍，再延迟1秒，接着调用tintSprite:data:方法，在2秒内让小球着色到RGB（255，0，255）。

注意 需要特别说明，CCCallFuncND动作所传递的data参数并没有特别指定的类型，所以在实际使用的过程中可以根据需要选择不同的参数，比如上例中的数字。

第四个精灵对象会延迟1秒，同时在1秒内淡入并放大到原大小的1.5倍，再延迟1秒，接着调用rotateSprite:object:方法。

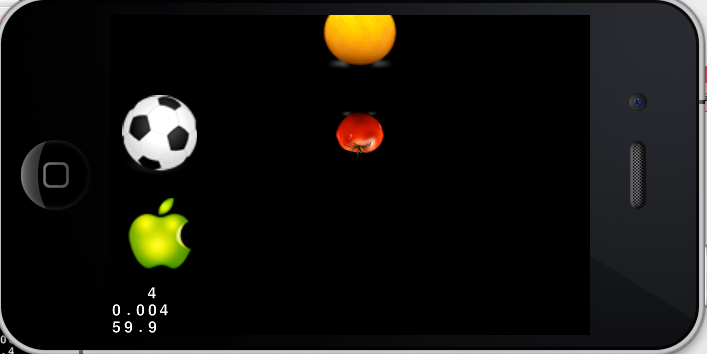
注意 这里传递的参数对象是第5个精灵对象，即西红柿。此时，第五个精灵对象会在2秒内淡入，然后在3秒内旋转180度。

图4-5是CallbackActions项目运行开始时的效果。



1. 项目开始效果图

图4-6是CallbackActions项目运行结束时的效果。



1. 项目结束效果图

### 块语句调用动作

从iOS 4.0开始支持块语句（Block），使复杂的代码结构得以简化。Cocos2D当前版本提供一些特殊动作，让节点对象执行这些动作时，不会直接修改节点的属性，而是调用某个块语句。块语句调用动作包括CCCallBlock、CCCallBlockN和CCCallBlockO。

这三个动作的作用分别等同于4.2.5中的3个方法回调动作，其中CCCallBlock对应CCCallFunc，而CCCallBlockN对应CCCallFuncN，CCCallBlockO则对应CCCallFuncO。

通常情况下，在序列动作的结尾执行此类动作以便调用某个特定的块语句。

为了理解这些动作的实际作用，还是通过实例说明。可参考本书例项目BlockActions。

1. 在Xcode创建一个新项目。

选择iOS-cocos2d模板，创建并命名为BlockActions。

1. 加载资源图片。

将准备好的资源图片（chapter4/resource/arts/BlockActions）拖动到Xcode的Resources中。记住，必须选中“Copy items into destination group’s folder(if needed)”。

1. 添加实例变量的定义。

切换到HelloWorldLayer.h中，在类声明中添加实例变量的定义，如代码清单4-14所示。

代码清单4-14添加实例变量的定义

@interface HelloWorldLayer : CCLayer<GKAchievementViewControllerDelegate, GKLeaderboardViewControllerDelegate>

{

CCSprite \*ball;

CCSprite \*apple;

CCSprite \*plane;

CCSprite \*orange;

}

1. 清空init方法。

切换到HelloWorldLayer.m，把init方法清空，如代码清单4-15所示。

代码清单4-15 把init方法清空

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

-(id) init

{

// always call "super" init

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

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

//清空init方法

}

return self;

}

1. 创建4个精灵对象。

在清空事的init方法中创建4个精灵对象，并将其添加为当前层的子节点，添加代码如代码清单4-16所示。

代码清单4-16创建4个精灵对象

// 创建第一个精灵，并将其添加为当前层的子节点

CGSize size = [[CCDirector sharedDirector] winSize];

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

apple.position = ccp(50,100);

apple.opacity = 0;

[self addChild:apple];

// 创建第二个精灵，并将其添加为当前层的子节点

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

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

plane.opacity = 0;

[self addChild: plane];

// 创建第三个精灵，并将其添加为当前层的子节点

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

ball.position = ccp(250,100);

ball.opacity = 0;

[self addChild:ball];

// 创建第四个精灵，并将其添加为当前层的子节点

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

orange.position = ccp(250,300);

orange.opacity = 0;

[self addChild:orange];

1. 让第一个精灵对象执行序列组合动作。

紧接着上面代码添加以下代码，让第一个精灵对象执行序列组合动作。最后一个动作调用CCCallBlock块语句，其作用等同于调用moveSprite方法，如代码清单4-17所示。

代码清单4-17让第一个精灵对象执行序列组合动作

[apple runAction:[CCSequence actions:[CCDelayTime actionWithDuration:1],

[CCSpawn actions:

[CCFadeIn actionWithDuration:1],

[CCScaleTo actionWithDuration:1 scale:1.5],

nil],

[CCDelayTime actionWithDuration:1],

[CCCallBlock actionWithBlock:^{

[apple runAction:[CCMoveTo actionWithDuration:2 position:ccp(50,200)]];

}],

nil]];

1. 让第二个精灵对象执行序列组合动作。

添加以下代码，让第二个精灵对象执行序列组合动作。最后一个动作调用CCCallBlockN块语句，其作用等同于调用removeSprite:方法，并将当前对象传递给指定的块语句，如代码清单4-18所示。

代码清单4-18让第二个精灵对象执行序列组合动作

[plane runAction:[CCSequence actions:[CCDelayTime actionWithDuration:1],

[CCSpawn actions:

[CCFadeIn actionWithDuration:1],

[CCScaleTo actionWithDuration:1 scale:1.5],

nil],

[CCDelayTime actionWithDuration:1],

[CCCallBlockN actionWithBlock:^(CCNode \*node) {

CCNode \*currentNode =node;

[self removeChild:currentNode cleanup:YES];

}],

nil]];

1. 让第三个精灵对象执行序列组合动作。

添加以下代码，让第三个精灵对象执行序列组合动作。最后一个动作调用CCCallBlockO块语句，其作用等同调用rotateSprite:object:方法，并将指定的对象传递给块语句，如代码清单4-19所示。

代码清单4-19让第三个精灵对象执行序列组合动作

[ball runAction:[CCSequence actions:[CCDelayTime actionWithDuration:1],

[CCSpawn actions:

[CCFadeIn actionWithDuration:1],

[CCScaleTo actionWithDuration:1 scale:1.5],

nil],

[CCDelayTime actionWithDuration:1],

[CCCallBlockO actionWithBlock:^(id object) {

CCNode \*node = (CCNode\*)object;

[node runAction:[CCSequence actions:[CCFadeIn actionWithDuration:2.0f],

[CCRotateBy actionWithDuration:3.0 angle:180],nil]];

} object:orange],

nil]];

1. 编译并运行项目。

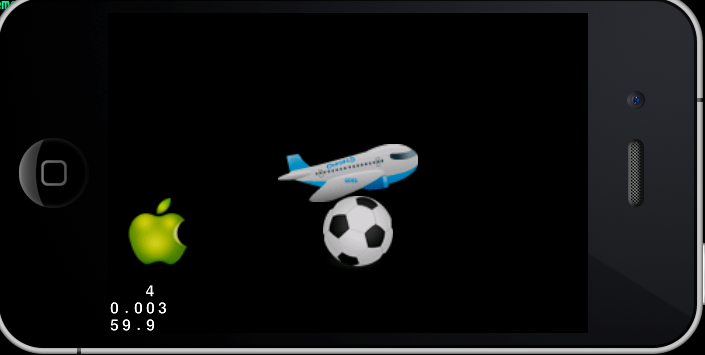
编译并运行项目，会看到4个精灵对象各自执行不同的组合动作，且和上节中的动作效果相同。

第一个精灵对象先延迟1秒，同时在1秒内淡入并放大到原大小的1.5倍，再延迟1秒，接着调用moveSprite方法，让小球在2秒内移动到（50，200）这一点。

第二个精灵对象会先延迟1秒，同时在1秒内淡入并放大到原大小的1.5倍，再延迟1秒，接着调用removeSprite:方法，从屏幕中清除这个小球并停止它的所有动作。

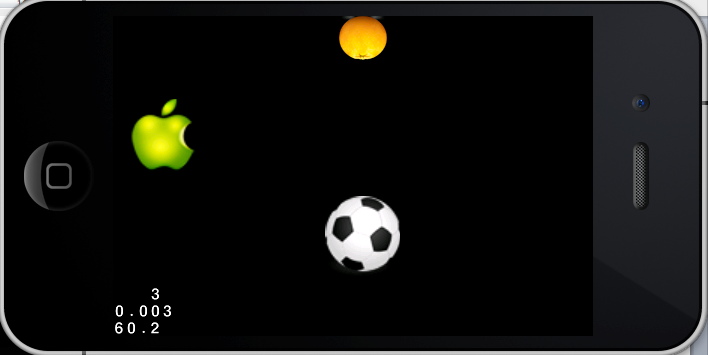
第三个精灵对象会延迟1秒，同时在1秒内淡入并放大到原大小的1.5倍，再延迟1秒，接着调用rotateSprite:object:方法。但是注意这里传递的参数对象是第四个精灵对象，即橙子。此时第四个精灵对象会在2秒内淡入，然后在3秒内旋转180度。

图4-7是CallbackActions项目运行开始的效果。



1. 开始的效果

图4-8是CallbackActions项目运行结束的效果。



1. 结束时的效果

### 反转动作

除了以上的动作，Cocos2D中还提供一种反转动作CCReverseTime。所谓反转动作，其实就是让节点对象反向执行某个动作或组合动作。如下所示：

id action = [CCMoveBy actionWithDuration:2 position:ccp(100,0)];

id reverseAction = [CCReverseTime actionWithAction:action];

[ball runAction:[CCSequence actions:action,reverseAction, nil]];

上面的代码中，首先创建一个CCMoveBy动作，紧接着又创建了它的反动作；最后让精灵对象执行一个序列动作，即先执行CCMoveBy动作，然后执行它的反动作。

注意 不是所有的动作都支持反动作，通常后面加To的动作不支持反动作，后面加By的动作则支持反动作。

此外，还可以使用以下方法创建反转动作：

id reverseAction2 = [action reverse];

注意 仅CCActionInterval类型的动作才可使用reverse方法。

该部分内容的代码可参考本章的示例项目ReverseAction。

### 特殊动作

CCSpeed可以修改一个Action的速度，如果speed小于1，那么此Action的运行时间会变长，反之亦然。注意该动作不能放置在CCSequence里，因为它不是一个CCIntervalAction。

CCActionTween动作可以修改对象的任何属性。

比如，如果想把一个结点的width属性从200变成300。可以编写以下代码：

id modifyWidth = [CCActionTween actionWithDuration:2 key:@"width" from:200 to:300];

[target runAction:modifyWidth];

下面使用CCActionTween实现，示例代码：

// scaleA和scaleB两个动作是等价的

id scaleA = [CCScaleTo actionWithDuration:2 scale:3];

id scaleB = [CCActionTween actionWithDuration:2 key:@"scale" from:1 to:3];

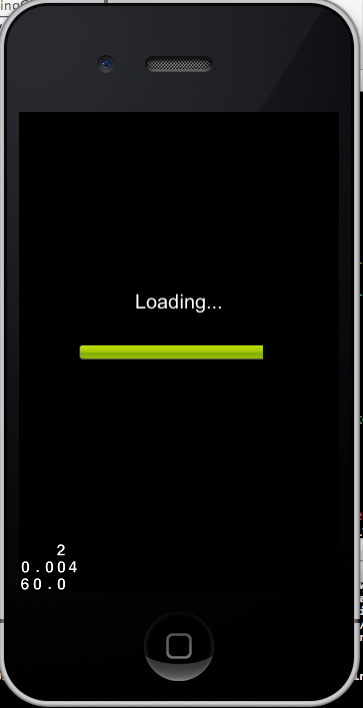
## 垂直射击游戏：加载进度条与滚动背景

本节运用CCProgressTo和CCProgressFromTo加载进度条；运用特殊动作CCParallaxNode和CCFollow添加连续滚动的背景。

### 使用CCProgressTo和CCProgressFromTo动作添加进度条

CCProgressTo和CCProgressFromTo这两个动作都继承自CCActionInterval，作用是以渐近方式显示图片。比如程序加载时的进度条，用这两个Action制作就非常方便。

在之前垂直射击游戏的Loading界面基础上，显示一个进度条，进度条全部显示完成代表加载完毕。当制作完进度条之后的效果如图4-9所示。



1. 进度条效果
2. 打开LoadingScreen.h文件，添加几个成员变量。示例代码：

CCProgressTimer \*progress; //此成员变量用来显示进度条

float progressInterval; //此变量用来计算每次进度条的更新量

1. 打开LoadingScreen.m，在init方法的计算winCenter的语句之后添加代码，如示例代码：

progress = [CCProgressTimer progressWithSprite:[CCSprite spriteWithFile:@"progressbar.png"]];

[progress setPercentage:0.0f];

progress.scale = 0.5f;

progress.midpoint = ccp(0,0.5);

progress.barChangeRate = ccp(1,0);

progress.type = kCCProgressTimerTypeBar;

[progress setPosition:winCenter];

[self addChild:progress];

这里使用progressbar.png图片初始化一个精灵，再从此精灵初始化一个CCProgressTimer对象。首先设置percentage为0，这表示并未加载任何资源，表现在画面上就是什么也看不见。由于图片大小关系，这里把scale设置成0.5，即缩小一半。

接下来是最重要的3个参数。

* Midpoint：表示进度条动画的起始位置，默认是在图片的中点，如果想显示从左右到右的一个动画效果，必须把midPoint改成（0，y）。
* barChangeRate：表示是否改变水平或者垂直方向的比例。如果设置为1则表示会改变，如果设置为0则表示不改变。因为我们想制作一个从左至右的水平进度条，所以midPoint应该是（0，0.5），因为x方向需要改变，而y方向不需要，所以barChangeRate设置为（1，0）。
* type：设置为 kCCProgressTimerTypeBar。

这几个参数的具体用途，可以参考ActionProgressTest，其中详细演示了几种不同的progress的用法。自己动手改改参数看看效果，相信很快就掌握了。

1. 更新progressUpdate方法，如代码清单4-20所示。

代码清单4-20 更新后的progressUpdate方法

-(void) progressUpdate

{

if (--assetCount)

{

[progress setPercentage:(100.0f - (progressInterval \* assetCount))];

}

else {

CCProgressFromTo \*ac = [CCProgressFromTo actionWithDuration:0.5 from:progress.percentage to:100];

CCCallBlock \*callbak = [CCCallBlock actionWithBlock:^(){

[self loadingComplete];

CCLOG(@"All done loading assets.");

}];

id action = [CCSequence actions:ac,callbak, nil];

[progress runAction:action];

}

}

这段代码首先根据每次加载资源时手动设置progress的percentage。这种做法可以实时地反映出具体的资源加载情况。最后，当资源加载达到99%时，运行一个CCProgressFromTo动作，把进度条更新到100%，同时调用loadingComplete函数把场景切换到游戏场景。

至此，游戏资源加载部分就全部完成了。

注意 本书提供的这种方法是Cocos2D社区里分享的，具有很高的可重用性。而且采用PLIST配置文件定义预先加载的资源内容，进一步增强代码的可复用性。

### 使用CCFollow和CCParallaxNode动作添加滚动背景

之前提到过CCFollow和CCParallaxNode动作可以模拟移动Camera效果，本节给垂直射击游戏添加连续滚动背景。

1. 打开HelloWorldLayer.h文件，添加2个新的成员变量：

CCParallaxNode \*\_backgroundNode;

int \_totalSeconds; //游戏进行时长

1. 打开HelloWorldLayer.m，在分类里声明一个私有方法：

-(void) updateBackground:(ccTime)dt;

1. 在init函数的最后添加下列代码：

//16.添加连续滚动背景

\_backgroundNode = [CCParallaxNode node];

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

CGPoint ratio = ccp(1.0,0.5);

CCSprite \*bgSprite1 = [CCSprite spriteWithSpriteFrameName:@"background\_1.jpg"];

[[bgSprite1 texture] setAliasTexParameters];

bgSprite1.anchorPoint = ccp(0,0);

[\_backgroundNode addChild:bgSprite1 z:1 parallaxRatio:ratio positionOffset:ccp(0,0)];

CCSprite \*bgSprite2 = [CCSprite spriteWithSpriteFrameName:@"background\_2.jpg"];

[[bgSprite2 texture] setAliasTexParameters];

bgSprite2.anchorPoint = ccp(0,0);

[\_backgroundNode addChild:bgSprite2 z:1 parallaxRatio:ratio positionOffset:ccp(0,winSize.height - offset)];

这段代码具体内容如下：

1. 首先初始化一个CCParallaxNode，把它加到当前层中。
2. 初始化两个背景精灵，并且更改精灵的锚点为（0，0）。
3. 调用CCParallaxNode的addChild方法，把两个精灵都添加进去。

这里的ratio指添加进去的sprite，在CCParallaxNode移动时，它们的移动速度与CCParallaxNode的比率。假如CCParallaxNode以每秒10像素移动，如果ratio是（1，2），那么bgSprite1和bgSprite2的x方向的速率也是10像素每秒，但是y方向上的速率却是20像素每秒。而setAliasTexParameters的调用主要是解决拼接的地图在连接滚动时容易形成黑色的缝隙。同时，在设置positionOffset时，有意在第二个背景与第一个背景拼接处减去1像素，这样可以消除地图拼接的缝隙。

1. 找到onEnter方法，在“开始游戏”菜单项的Block函数最后添加下列代码：

//8.添加开始连续滚动背景的代码

const int MAX\_LEVEL\_WIDTH = 320;

const int MAX\_LEVEL\_HEIGHT = 480 \* 100;

CCSprite \*hiddenPlayerSprite = [CCSprite spriteWithSpriteFrameName:@"hero.png"];

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

[self addChild:hiddenPlayerSprite z:-4 tag:1024];

\_totalSeconds = 60;

id move = [CCMoveBy actionWithDuration:\_totalSeconds position:ccp(0,MAX\_LEVEL\_HEIGHT)];

[hiddenPlayerSprite runAction:move];

//让背景开始滚动

[\_backgroundNode runAction:[CCFollow actionWithTarget:hiddenPlayerSprite

worldBoundary:CGRectMake(0, 0, MAX\_LEVEL\_WIDTH, MAX\_LEVEL\_HEIGHT)]];

这里运用了一个技巧。因为想实现背景的连续滚动，而CCFollow动作需要背景能够跟随（Follow）一个往上不断前进的CCNode，这样才会产生连续不断地向下滚动的效果。这里采用的方式是添加一个隐藏的玩家，让它在游戏时间范围内（\_totalSeconds）运行最大的关卡距离（MAX\_LEVEL\_HEIGHT）。最后，让背景ParallaxNode结点运行一个CCFollow动作，同时指定跟随的目标结点和边界范围。

1. 添加两个预先制作好的CCParallaxNode扩展类。

找到chapter4/resource/classes/目录，添加两个预先制作好的CCParallaxNode扩展类。把CCParallaxNode-Extras.h和CCParallaxNode-Extras.m都添加到项目中。然后在update函数里添加之前更新背景函数的调用。最终的update函数看起来如下所示：

-(void) update:(ccTime)dt{

if (!\_isGameStarted) {

return;

}

[self updatePlayerPosition:dt];

[self updatePlayerShooting:dt];

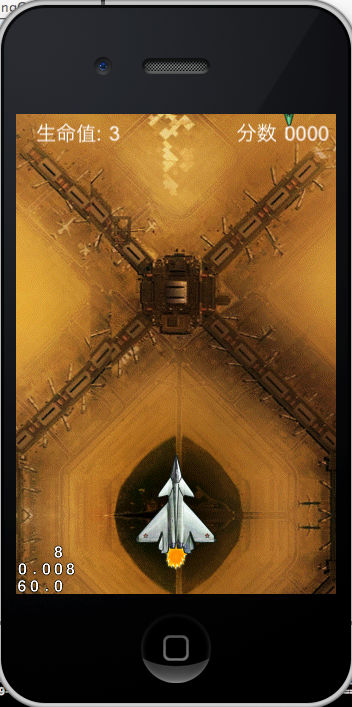
[self collisionDetection:dt];

[self updateHUD:dt];

[self updateBackground:dt];

}

这时编译并运行，可以看到一个连续往下滚动的游戏背景啦，效果如图4-10所示。



1. 连续往下滚动的游戏背景效果

注意 此时尾部的火焰是静态的，4.7.1节会介绍如何添加尾部喷气的动态效果。

细心的读者马上就会发现，当游戏进行到40秒时，隐藏的sprite停止移动，这时滚动背景也失效了。当然，既然是游戏时间到了，那肯定是要给玩家提示Game Over的。

1. 最后再添加一个consumedTime变量记录游戏逝去的时间。
2. 打开HelloWorldLayer.h，添加下面代码：

ccTime \_consumedTime;//已消耗的游戏时长

1. 打开HelloWorldLayer.m，并在init函数的最后添加下列代码：

//17.init consumedTime

\_consumedTime = 0.0f;

最后，修改一下update函数，修改如下：

-(void) update:(ccTime)dt{

if (!\_isGameStarted) {

return;

}

\_consumedTime += dt;

if (\_consumedTime >= \_totalSeconds) {

//game over

[self gameOver];

}

//后面的代码省略

}

这里的GameOver代码跟玩家3条生命全部失去时的代码一样，根据DRY（Don’t Repeat Yourself）原则，把这段代码封装成一个gameOver方法。具体实现如下：

-(void) gameOver{

[\_gameEndLabel setString:@"游戏失败!"];

\_gameEndLabel.visible = YES;

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

[\_gameEndLabel runAction:scaleTo];

[self unscheduleUpdate];

[\_backgroundNode stopAllActions];

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

}

相应地，需要把游戏玩家生命值为0的代码逻辑也替换，如下所示：

if (\_totalLives <= 0) {

[self gameOver];

}

1. 编译并运行。

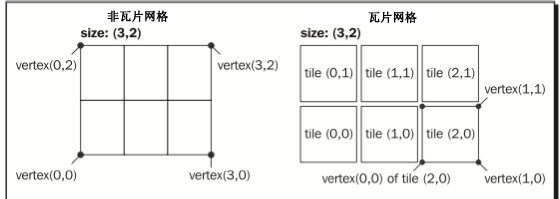
这时该游戏就更加具有挑战性啦。如果60秒之内没有达到1000分则gameOver。呵呵，如果觉得1000分不算什么，那么改成10000试试吧。

## Cocos2D中的特效

在Cocos2D中，有一类非常特殊的动作，称为特效。特效会调整节点的网格属性。网格是节点的一种新属性，使用网格可以将节点划分为更小的方块或瓦片，通过移动组成每个方块的顶点调整节点的属性。

### Cocos2D中的网格

在Cocos2D中提供了两类网格，分别是瓦片网格和非瓦片网格。其区别在于，非瓦片网格是由顶点组成的，而瓦片网格则是由瓦片组成的，而每个瓦片都有自己的独立顶点，具体如图4-11所示。



1. 瓦片网格和非瓦片网格

尽管网格只有2维（行和列），但每个顶点都有3维（x，y和z），这样就可以通过调整网格产生仿3D的效果。

注意 虽然使用特效动作可以制作非常炫的效果，但为此付出的代价就是游戏性能的下降。使用特效动作有时会让游戏的帧数急剧下降，所以使用时一定要谨慎。

当然，网格的大小也可以被修改。通常来说，网格越大，特效越酷，但付出的代价就是性能会下降得更多；网格越小，性能越好，但特效可能没有那么突出。

### 特效的种类及使用

Cocos2D中提供了20余种特效，为了方便学习，可以参考Cocos2D官方模板项目的ActionsTest，或随书附带的示例项目EffectsAction。

这里先实际看其中的一种特效。让精灵对象产生类似波动的效果，如以下代码所示：

id effect = [CCWaves actionWithWaves:10 amplitude:10 horizontal:YES vertical:YES grid:ccg(10,10) duration:5];

[mySprite runAction:effect];

以上代码创建一个CCWaves类型的特效动作，其中参数包括波动的数量、波幅、波动的方向（是否会在水平或垂直方向波动）、网格的大小以及特效的持续时间。这里注意到其中用ccg创建网格的大小。使用ccg可以用指定大小创建一个CCGrideBase对象（网格对象）。

除了CCWaves，Cocos2D中还提供了一系列特效动作，下面分别进行介绍。

##### CCWaves

通过更改节点对象的网格属性，让节点对象产生类似波动的效果。示例代码：

id effect = [CCWaves actionWithWaves:10 amplitude:50 horizontal:YES vertical:YES grid:ccg(10,10) duration:5];

[mySprite runAction:effect];

##### CCWaves3D

通过更改节点对象的网格属性，让节点对象产生类似三维波动的效果。如以下代码所示：

id effectWaves3D = [CCWaves3D actionWithWaves:10 amplitude:40 grid:ccg(12,12) duration:5];

[mySprite runAction:effectWaves3D];

##### CCFlipX3D

通过更改节点对象的网格属性，让节点对象沿着水平方向产生三维翻转效果。如以下代码所示：

id effectFlipX3D = [CCFlipX3D actionWithSize:ccg(1,1) duration:2];

id effectFlipX3Dback =[effectFlipX3D reverse];

[mySprite runAction:[CCSequence actions:effectFlipX3D,[CCDelayTime actionWithDuration:2], effectFlipX3Dback,nil]];

##### CCFlipY3D

通过更改节点对象的网格属性，让节点对象沿着垂直方向产生三维翻转效果。如以下代码所示：

id effectFlipY3D = [CCFlipY3D actionWithDuration:2];

id effectFlipY3Dback = [effectFlipY3D reverse];

[mySprite runAction:[CCSequence actions:effectFlipY3D,[CCDelayTime actionWithDuration:2], effectFlipY3Dback, nil]];

##### CCLens3D

通过更改节点对象的网格属性，让节点对象产生类似镜面三维效果。如以下代码所示：

id effectLens3D = [CCLens3D actionWithPosition:ccp(size.width/2,size.height/2) radius:240 grid:ccg(15,10) duration:3];

[mySprite runAction: effectLens3D];

##### CCRipple3D

通过更改节点对象的网格属性，让节点对象产生类似水面涟漪的三维波动效果。如以下代码所示：

id effectRipple3D = [CCRipple3D actionWithPosition:ccp(size.width/2,size.height/2) radius:240 waves:4 amplitude:160 grid:ccg(32,24) duration:5];

[mySprite runAction:effectRipple3D];

##### CCShaky3D

通过更改节点对象的网格属性，让节点对象产生类似三维摇晃效果。如以下代码所示：

id effectShaky3D = [CCShaky3D actionWithRange:5 shakeZ:YES grid:ccg(10,10) duration:3];

[mySprite runAction:effectShaky3D];

##### CCLiquid

通过更改节点对象的网格属性，让节点对象产生类似液体流动的效果。如以下代码所示：

id effectLiquid = [CCLiquid actionWithWaves:10 amplitude:22 grid:ccg(16,10) duration:5];

[ball runAction:effectLiquid];

##### CCTwirl

通过更改节点对象的网格属性，让节点对象产生漩涡效果。如以下代码所示：

id effectTwirl = [CCTwirl actionWithPosition:ccp(size.width/2, size.height/2) twirls:1 amplitude:2.5f grid:ccg(12,8) duration:5];

[ball runAction:effectTwirl];

##### CCShatteredTiles3D

通过更改节点对象的网格属性，让节点对象及其瓦片产生类似粉碎的三维效果。如以下代码所示：

id effectShatteredTiles3D = [CCShatteredTiles3D actionWithRange:5 shatterZ:YES grid:ccg(16,12) duration:5];

[mySprite runAction:effectShatteredTiles3D];

##### CCShakyTiles3D

通过更改节点对象的网格属性，让节点对象及其瓦片产生类似晃动的三维效果。如以下代码所示：

id effectShakyTiles3D = [CCShakyTiles3D actionWithRange:5 shakeZ:YES grid:ccg(16,12) duration:5];

[mySprite runAction:effectShakyTiles3D];

##### CCShuffleTiles

通过更改节点对象的网格属性，让节点对象及其瓦片产生类似洗牌的效果。如以下代码所示：

id effectShuffleTiles = [CCShuffleTiles actionWithSeed:25 grid:ccg(16,12) duration:3];

id shuffleBack = [effectShuffleTiles reverse];

id delay = [CCDelayTime actionWithDuration:2];

[mySprite runAction:[CCSequence actions:effectShuffleTiles,delay,shuffleBack, nil]];

##### CCFadeOutTRTiles

通过更改节点对象的网格属性，让节点对象及其瓦片产生从左下角到右上角淡出的效果。如以下代码所示：

id effectFadeOutTRTiles = [CCFadeOutTRTiles actionWithSize:ccg(16,12) duration:5];

id fadeouttrBack = [effectFadeOutTRTiles reverse];

id trDelay = [CCDelayTime actionWithDuration:2];

[mySprite runAction:[CCSequence actions:effectFadeOutTRTiles,trDelay,fadeouttrBack, nil]];

##### CCFadeOutBLTiles

通过更改节点对象的网格属性，让节点对象及其瓦片产生从右上角到左下角淡出的效果。如以下代码所示：

id effectFadeOutBLTiles = [CCFadeOutBLTiles actionWithSize:ccg(16,12) duration:5];

id fadeoutblBack = [effectFadeOutBLTiles reverse];

id blDelay = [CCDelayTime actionWithDuration:2];

[mySprite runAction:[CCSequence actions:effectFadeOutBLTiles,blDelay,fadeoutblBack, nil]];

##### CCFadeOutUpTiles

通过更改节点对象的网格属性，让节点对象及其瓦片产生自下而上淡出的效果。如以下代码所示：

id effectFadeOutUpTiles = [CCFadeOutUpTiles actionWithSize:ccg(10,10) duration:5];

id fadeoutupBack = [effectFadeOutUpTiles reverse];

id upDelay = [CCDelayTime actionWithDuration:2];

[mySprite runAction:[CCSequence actions:effectFadeOutUpTiles,upDelay,fadeoutupBack, nil]];

##### CCFadeOutDownTiles

通过更改节点对象的网格属性，让节点对象及其瓦片产生自上而下淡出的效果。如以下代码所示：

id effectFadeOutDownTiles = [CCFadeOutDownTiles actionWithSize:ccg(10,10) duration:5];

id fadeoutdownBack = [effectFadeOutDownTiles reverse];

id downDelay = [CCDelayTime actionWithDuration:2];

[mySprite runAction:[CCSequence actions:effectFadeOutDownTiles,downDelay,fadeoutdownBack, nil]];

##### CCTurnOffTiles

通过更改节点对象的网格属性，产生瓦片逐渐消失的效果。如以下代码所示：

id effectTurnOffTiles = [CCTurnOffTiles actionWithSeed:5 grid:ccg(10,10) duration:5];

id turnoffBack = [effectTurnOffTiles reverse];

id turnoffDelay = [CCDelayTime actionWithDuration:2];

[mySprite runAction:[CCSequence actions:effectTurnOffTiles,turnoffDelay,turnoffBack, nil]];

##### CCWavesTiles3D

通过更改节点对象的网格属性，让节点对象及其瓦片产生类似三维波浪的效果。如以下代码所示：

id effectWavesTiles3D = [CCWavesTiles3D actionWithWaves:10 amplitude:120 grid:ccg(15,12) duration:5];

[ball runAction:effectWavesTiles3D];

##### CCJumpTiles3D

通过更改节点对象的网格属性，瓦片产生跳动的三维效果。如以下代码所示：

id effectJumpTiles3D = [CCJumpTiles3D actionWithJumps:10 amplitude:30 grid:ccg(12,12) duration:5];

[mySprite runAction:effectJumpTiles3D];

##### CCSplitRows

通过更改节点对象的网格属性，让节点对象产生分行分割并消失的效果。如以下代码所示：

id effectSplitRows = [CCSplitRows actionWithRows:10 duration:5];

[mySprite runAction:effectSplitRows];

##### CCSplitCols

通过更改节点对象的网格属性，让节点对象产生分列分割并消失的效果。如以下代码所示：

id effectSplitCols = [CCSplitCols actionWithCols:10 duration:5];

[mySprite runAction:effectSplitCols];

##### CCPageTurn3D

通过更改节点对象的网格属性，让节点对象产生三维翻页的效果。如以下代码所示：

id effectPageTurn3D = [CCPageTurn3D actionWithSize:ccg(15,12) duration:5];

[mySprite runAction:effectPageTurn3D];

### 让节点对象返回最初状态

前面的示例中看到，每次特效动作结束后，精灵对象不会自动返回到最初状态。为了弥补这一点缺陷，可以在特效之后执行另一个动作修复。如以下代码所示：

id effect = [CCSequence actions:[CCWaves actionWithWaves:10 amplitude:30 horizontal:YES vertical:YES grid:ccg(10,10) duration:5], [CCStopGrid action], nil];

[mySprite runAction:effect];

以上代码定义一个序列组合动作，首先让节点对象产生类似波浪的效果，使用CCStopGrid方法停止动作。

CCStopGrid方法的作用是让网格的每个顶点返回到最初位置，让图像恢复到执行特效之前的原貌。

## Cocos2D中的动画

在Cocos2D中，让精灵对象快速运行一系列图片可以产生动画效果。比如，在游戏场景中有一只可爱的小猪，如何让它走到其他地方呢？虽然用简单的移动动作也可以实现这个目标，但看上去非常不真实。通常希望达到的效果是，产生一种让小猪爬动的动画，而不是瞬间飞移过去，这里就要用到动画。

在Cocos2D中实现动画其实很简单，只需从精灵表单（CCSpriteBatchNode）中获取一系列图片，按照特定的顺序排列，最后在精灵对象上执行特定的动作。

### Cocos2D中与动画相关的类

为了学习如何在Cocos2D中实现动画效果，除了在第3章中曾经接触过的核心类，还需要了解CCAnimate、CCAnimation和CCAnimationCache这三个类。

##### CCAnimate

一种特殊的动作，也称为动画动作。

##### CCAnimation

用于在CCSprite精灵对象上执行动画的对象。该对象中包含CCSpriteFrame（精灵帧）对象，以及各个精灵帧之间的延迟。

通常情况下，使用CCAnimate执行某个动画。如下所示：

CCAnimation \*anim =[CCAnimation animationWithFrames:animFrames delay:0.2f];

[mySprite runAction:[CCAnimate actionWithAnimation:anim]];

以上代码首先使用精灵帧创建一个CCAnimation对象，然后通过CCAnimate动作让精灵对象执行该动画。

##### CCAnimationCache

是一个单例，用来缓存CCAnimation动画。如果在游戏中创建多个动画，就可以使用CCAnimationCache缓存管理这些动画。如下所示：

[[CCAnimationCache sharedAnimationCache]addAnimation:anim name:@"AnimationName"];

CCAnimation \*myAnimation = [[CCAnimationCache sharedAnimationCache] animationByName:@AnimaitonName"];

以上代码首先在CCAnimationCache缓存中添加anim动画，并指定一个名称。使用该动画时，只需要使用animationByName方法从缓存中读取该动画即可。

### 在Cocos2D中实现动画效果

用一个实例说明在Cocos2D中如何使用动画。参考本书示例项目AnimationActions。

打开Xcode，使用Cocos2D模板创建一个项目，并命名为AnimationActions。下载准备好的图片资源（chapter4/resource/arts/AnimationActions）。

首先尝试直接把这些图片添加到项目之中，并利用这些单独的图片创建动画。如代码清单4-21所示。

代码清单4-21利用这些单独的图片创建动画

// 简单的动画实现方式

// 创建精灵对象并将其添加为当前层的子节点

CGSize size = [CCDirector sharedDirector].winSize;

CCSprite \*mySprite = [CCSprite spriteWithFile:@"pandawalk1.png"];

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

[self addChild:mySprite];

// 创建CCAnimation动画，指定动画帧的内容

CCAnimation \*anim = [CCAnimation animation];

[anim addFrameWithFilename:@"pandawalk1.png"];

[anim addFrameWithFilename:@"pandawalk2.png"];

[anim addFrameWithFilename:@"pandawalk3.png"];

// 创建CCAnimate动画动作，并让精灵对象执行。通过指定restoreOriginalFrame为YES，可以让精灵对象在动画执行完成后恢复到最初的状态。

id animAction = [CCAnimate actionWithDuration:0.5f animation:anim restoreOriginalFrame:YES];

id repeatanimAction = [CCRepeatForever actionWithAction:animAction];

[mySprite runAction:repeatanimAction];

以上代码首先使用3张图片创建一个简单的动画，并让精灵对象反复执行该动画。

这种方式对于简单动画是很有效的。效果如图4-12所示。



1. 简单动画效果

## 使用Zwoptex工具实现动画效果

为了提高游戏的效能，通常需要使用更高效的方式——精灵表单。

知识链接 精灵表单

第3章曾经对精灵表单做过简单的介绍，这里需要复习相关的内容。关于精灵表单，可以将其想象成一张大图，其中的内容由多个精灵的小图片拼接而成。精灵表单会提供一个文件，在其中指定单个精灵的边界，从而让用户在需要时从表单中提取单个精灵的图片。

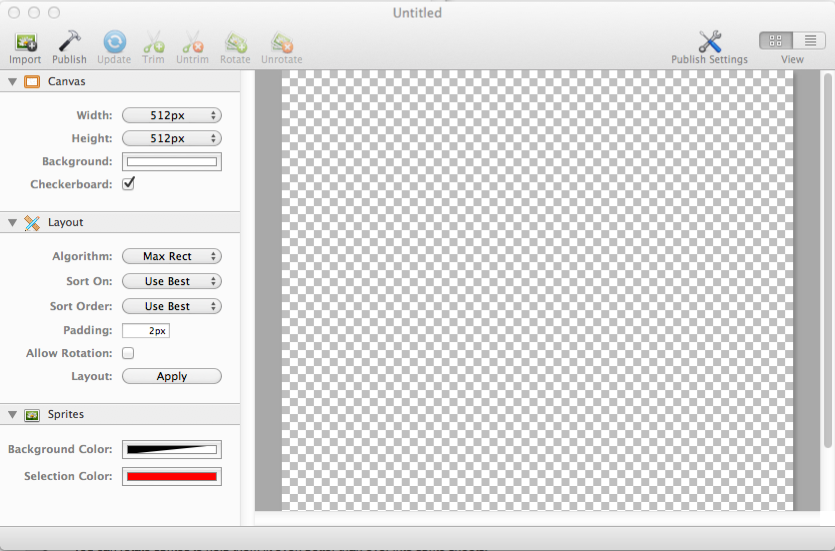
Cocos2D专门针对精灵表单做过优化。如果使用单独的精灵，那么每次都需要调用OpenGL ES绘图命令，当游戏中用到很多精灵时，速度会大大降低。而如果使用精灵表单，无论表单上有多少精灵，只需要调用一次绘图命令即可。简而言之，使用精灵表单会大大提升游戏性能。关于游戏性能提升的具体方法，本书高级篇会进行更为详尽的介绍。

该如何实际创建精灵表单呢？当然，不怕麻烦，可以使用图像编辑器手动创建一张包含所有精灵的大图，然后手动创建一个PLIST文件指定每个精灵的边界和其他属性。但是这样做太耗费时间和精力。

古人云，工欲善其事，必先利其器。幸运的是，Zwoptex工具可以帮我们轻松解决这一问题。下面逐步介绍使用Zwoptex工具实现动画效果的过程。首先要下载工具。

### 安装Zwoptex工具

Zwoptex下载地址：http://zwoptexapp.com/。下载安装后，单击File→New，会打开画面如图4-13所示。

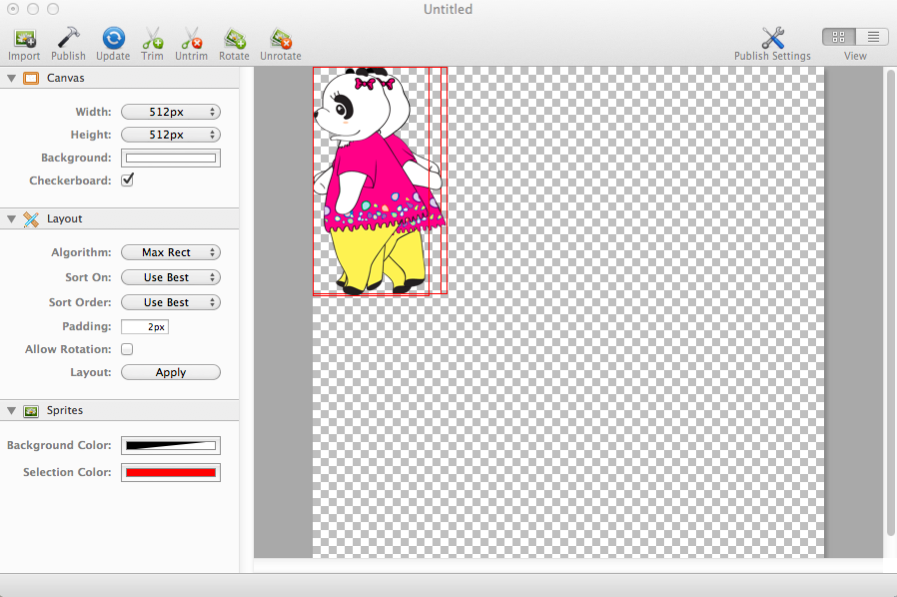


1. Zwoptex主界面

### 制作精灵表单

1. 加载图片。

打开下载的资源文件夹（/chapter4/resource/arts/AnimationActions），把所有图片拖动到空白处。如图4-14所示。

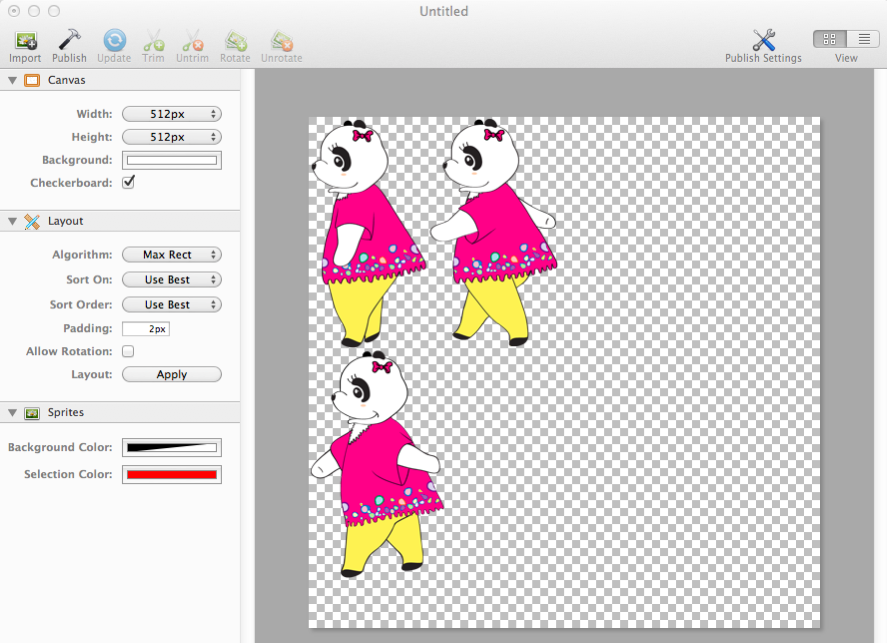


1. 添加图片后Zwoptex界面效果

现在所有图片是叠放在一起的，需要让它们在表单中散开，所以要单击左侧Layout下面的Apply按钮。

注意 如果默认的512px\*512px的画布放不下这些图片，可以根据需要更改Canvas的Width或者Height,然后再按下Apply按钮，就OK了。而这里就采用默认的画布大小。

如果不同图片的宽度不太一致，因为Zwoptex会自动裁边以去除图片周围的空白。在这种情况下只需单击工具栏上的Untrim按钮，然后单击Apply按钮即可，此时的画面如图4-15所示。



1. 添加图片并单击Apply后的画面
2. 保存项目。

单击File→Save以保存该项目，并取一个名字AnimPanda。

1. 修改目录。

单击工具栏上的Publish→Publish Settings，把Texture File Path和Coordinates File Path的目录都修改到项目中的Resources下，把Coordinates Format改为Cocos2D，然后保存。

1. 生成目标文件。

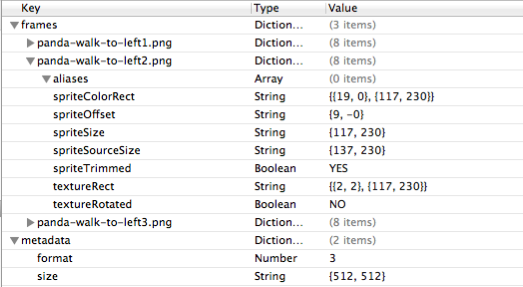
单击Publish按钮生成所需的PLIST和PNG文件。其中，PLIST文件对应Coordinates Format，PNG文件对应Texture File Path。

1. 将目标文件加载到项目中。

从Resources目录中将AnimPanda.plist和AnimPanda.png文件拖到项目的Resources中。确保选中“copy items into destination group’s folder(if needed)”。

1. 查看AnimPanda.plist文件。

查看生成的AnimPanda.plist文件，具体内容如图4-16所示。



1. AnimPanda.plist的组成

AnimPanda.plist文件由两部分组成，分别是frames和metadata。其中，frames部分是每个精灵图片的入口，其属性值用于从精灵表单中获取精灵。

以上就是制作精灵表单的具体方法，接下来将演示如何制作精灵动画。

### 制作精灵动画

首先把熊猫精灵放在游戏场景的中间，让它在原地永无止尽地踏步。

在Xcode中切换到HelloWorldLayer.m，找到init方法，更改代码如代码清单4-22所示。

代码清单4-22 清空init方法

-(id) init

{

// always call "super" init

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

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

//以下的内容会在此处添加代码，以生成动画效果。

}

return self;

}

在Cocos2D中为了实现动画，需要分5步完成，下面将逐一说明。在init方法注释代码的后面需要分别添加以下代码。

1. 将精灵帧纹理添加到精灵帧缓存中。

使用PLIST文件将精灵帧纹理添加到精灵帧缓存中，如下所示：

[[CCSpriteFrameCache sharedSpriteFrameCache] addSpriteFramesWithFile:@"AnimPanda.plist"];

以上代码首先创建一个CCSpriteFrameCache（关于精灵帧缓存详见本书3.6.4节）对象，然后用它调用addSpriteFramsWithFile方法，并向该方法传递一个参数，即AnimPanda.plist（刚刚使用Zwoptex创建的精灵表单文件之一）。

addSpriteFramesWithFile完成了以下事情：

1. 查找和所传递过来的PLIST文件名称相同，但是以png结尾的文件（纹理图，本例是AnimPanda.png文件），将其加载入CCTextureCache（纹理缓存）。
2. 解析PLIST文件内容，记录精灵表单中所有精灵的位置，使用内部CCSpriteFrame对象记录这些信息。
3. 创建一个精灵表单。

创建CCSpriteBatchNode（详见本书第3章相关内容）对象，如下所示：

CCSpriteBatchNode \*batchNode = [CCSpriteBatchNode batchNodeWithFile:@"AnimPanda.png"];

[self addChild:batchNode];

在将精灵帧纹理添加到精灵帧缓存中后，这里创建一个CCSpriteBatchNode对象，并将精灵表单的图像文件传递给该对象。

在Cocos2D中，精灵表单的工作方式如下：

1. 创建一个精灵表单对象，将包含所有精灵的图像文件传递给它（PNG或PVR），然后把它添加为当前层的子节点。
2. 当需要从精灵表单中创建单一精灵对象时，应将其添加为精灵表单的子节点，而不能直接添加为当前层的子节点，这一点需要切记！
3. 使用精灵表单后，程序会调用一次OpenGL ES命令绘制所有的精灵，而不是每次绘制精灵都需要单独调用一次OpenGL ES命令。

注意 CCSpriteBatchNode在Cocos2D旧版本中对应的是CCSpriteSheet。

1. 创建图片帧列表。

为了创建图片帧列表，需要遍历所有图片（panda1.png、panda2.png、panda3.png），然后从精灵帧缓存中获取与图片名称相对应的精灵帧。如下所示：

NSMutableArray \*animFrames = [NSMutableArray array];

for(int i=1; i<3;i++){

[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache]spriteFrameByName:[NSString stringWithFormat:@"pandawalk%d.png",i]]];

}

注意 因为之前调用过addSpriteFramesWithFile方法，所以精灵帧应存在于缓存之中。

1. 创建动画对象。

创建CCAnimation对象并传入图片帧列表，同时指定动画的播放速度。如下所示：

CCAnimation \*walkAnim = [CCAnimation animationWithFrames:walkAnimFrames delay:0.1f];

1. 创建精灵，并运行动画动作。如下所示：

CGSize size = [CCDirector sharedDirector].winSize;

CCSprite \*panda = [CCSprite spriteWithSpriteFrameName:@"pandawalk1.png"];

panda.position = ccp(size.width\*0.8,size.height\*0.4);

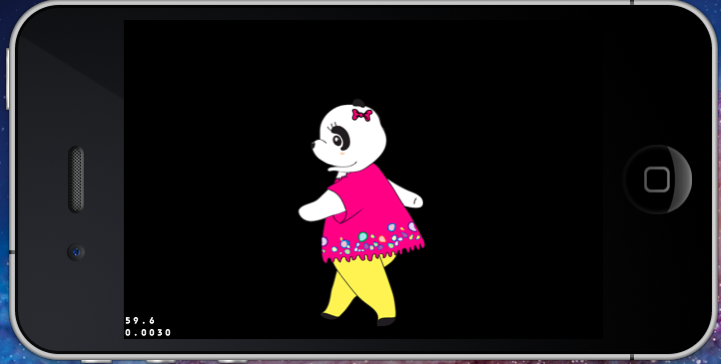
id walkAction = [CCRepeatForever actionWithAction:[CCAnimate actionWithAnimation:walkAnim restoreOriginalFrame:YES]];

[panda runAction:walkAction];

[panda runAction:[CCMoveTo actionWithDuration:6.0f position:ccp(size.width\*0.2,size.height\*0.4)]];

[batchNode addChild:panda];

通过以上5个步骤，轻松创建Cocos2D动画效果。编译运行本书附带的项目AnimationActions，会看到屏幕中的熊猫在6秒内从屏幕右方走到左方，然后停下，且双腿不断摆动，如图4-17所示。



1. AnimationActions项目编译运行后的画面

提示 在项目中既可以直接使用图片帧创建动画，也可以使用精灵表单创建动画。但如果游戏场景中有很多精灵对象，建议采用精灵表单以提高游戏的性能。

## 垂直射击游戏：添加飞行和爆炸动画

把刚刚学习到的动画技术运用到垂直射击游戏当中，这样可以使得该游戏更加丰富和逼真。

### 添加玩家飞机飞行动画

1. 打开HelloWordLayer.h，新添加一个成员变量：

id \_playerFlyAction; //玩家飞机飞行动画

1. 打开HelloWorldLayer.m，在最上面的私有分类中新增一个辅助方法：

-(CCAnimation\*)getAnimationByName:(NSString\*)animName delay:(float)delay animNum:(int)num;

该方法的作用是根据动画帧的名字animName和动画帧的数量num，以及动画帧与帧之间的间隔创建CCAnimation动画。使用该方法创建动画需要注意2点：

* + 动画帧的命名必须带序号，比如xxx1.png、xxx2.png、xxx3.png等。
  + 动画帧的命名必须连续，而且必须从1开始命名。

如果不满足条件的话，则使用此辅助方法创建动画会报错。接下来看看该方法的具体代码实现：

-(CCAnimation\*)getAnimationByName:(NSString \*)animName delay:(float)delay animNum:(int)num{

NSMutableArray \*animFrames = [NSMutableArray arrayWithCapacity:num];

for (int i=1; i<= num; ++i) {

NSString \*frameName = [NSString stringWithFormat:@"%@%d.png",animName,i];

CCSpriteFrame \*frame = [[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:frameName];

[animFrames addObject:frame];

}

CCAnimation \*animation = [CCAnimation animationWithSpriteFrames:animFrames delay:delay];

return animation;

}

这段代码之前学习动画的时候已经详细讲述过了，这里不再赘述。

1. 找到init方法，在代码注释标号为17的代码后面添加下列代码：

//18.init player fly animation

CCAnimation \*playerFlyAnimation = [self getAnimationByName:@"hero\_" delay:0.08 animNum:2];

\_playerFlyAction = [CCRepeatForever actionWithAction:[CCAnimate actionWithAnimation:playerFlyAnimation]];

[\_playerFlyAction retain];

[playerSprite runAction:\_playerFlyAction];

这段代码的作用就是调用之前创建的CCAnimation辅助方法创建玩家飞机飞行动画；然后使用CCRepeatForever和CCAnimate创建一个永久飞行的动画效果；最后调用playerSprite的runAction方法播放动画效果。注意，这里给\_playerFlyAction发送了retain消息，因为之后还需要使用此动画。所以，为了避免内存泄漏，需要在dealloc方法里去释放：

[\_playerFlyAction release];

\_playerFlyAction = nil;

1. 编译并运行，会得到一个尾部不断喷射火焰的飞机啦！如图4-18所示。



1. 尾部喷射火焰的飞机

### 添加飞机爆炸效果

之前敌机被击中以及玩家飞机被敌机碰撞的效果都是通过运行Cocos2D自带的动作表现的，如果能在此基础之上添加飞机爆炸的动画效果，那将会使游戏更加丰富多彩。

1. 打开HelloWorldLayer.h，并新添加下面4个成员变量：

id \_playerBlowupAnimation; //玩家飞机爆炸动画

id \_enemyBlowupAnimation; //敌机爆炸动画

BOOL \_isEnemyCollidable; //敌机是否可以碰撞

BOOL \_isPlayerCollidable; //玩家飞机是否可以碰撞

1. 在init方法中初始化这些变量，如下所示：

//19.init player & enemy 's blow up animation

\_playerBlowupAnimation = [self getAnimationByName:@"plane\_bao\_" delay:0.08 animNum:5];

[\_playerBlowupAnimation retain];

\_enemyBlowupAnimation = [self getAnimationByName:@"plane2\_bao\_" delay:0.08f animNum:5];

[\_enemyBlowupAnimation retain];

//20.init collidable flags

\_isEnemyCollidable = YES;

\_isPlayerCollidable = YES;

这段代码也非常简单，就是初始化玩家飞机爆炸和敌机爆炸的Animation，同时把敌机和玩家飞机是否可以碰撞的标志设置为YES。

1. 为了防止忘记释放内存，找到dealloc方法，新添加下列代码：

[\_playerBlowupAnimation release];

\_playerBlowupAnimation = nil;

[\_enemyBlowupAnimation release];

\_enemyBlowupAnimation = nil;

1. 重构碰撞检测代码。

打开HelloWorldLayer.m文件，找到其中的collisionDetection方法，在添加爆炸效果之前，首先，需要重构代码。

提示 关于重构，有个很形象的比喻，就是“两顶帽子”。一顶帽子是修改代码，即重构。另一顶帽子是添加新功能。任何时候，你的头上应该只有一顶帽子。所以，我们决定先重构代码再添加爆炸效果。

找到下列代码：

CCARRAY\_FOREACH(\_enemySprites, enemy)

{

if (enemy.visible && enmy.opacity == 255) {

//后面代码省略

}

//后面代码省略

}

改成以下形式：

CCARRAY\_FOREACH(\_enemySprites, enemy)

{

if (enemy.visible) && \_isEnemyCollidable {

\_isEnemyCollidable = NO;

//后面代码省略

}

}

同样地，把敌机与玩家飞机碰撞的代码重构成以下形式：

//2.enemy & player collision detection

CCSprite \*playerSprite = [self getPlayerSprite];

// CGRect playRect = [self rectOfSprite:playerSprite];

if (playerSprite.visible && \_isPlayerCollidable

&& CGRectIntersectsRect(enemy.boundingBox, playerSprite.boundingBox)) {

\_isPlayerCollidable = NO;

//后面代码省略

}

这两处重构的主要目的就是使代码结构更清晰，程序鲁棒性更好。

之前判断敌机与玩家飞机的碰撞条件都利用了一些特殊的技巧，比如判断可见度（opacity）是否为255，以及判断玩家飞机的当前运行总动作数（numberOfRunningActions）是否为0。这种判断方式很容易给游戏造成bug，所以，采用状态标志是比较合适的做法。

1. 修改collisionDetection方法中敌机被击中的效果，如代码清单 4-23所示。

代码清单4-23 给敌机销毁添加爆炸效果

id ac1 = [CCScaleTo actionWithDuration:1.0 scale:1.2];

id ac2 = [CCRotateBy actionWithDuration:1.0 angle:720];

id ac3 = [CCFadeOut actionWithDuration:1.0];

id ac4 = [CCHide action];

id blowup = [CCAnimate actionWithAnimation:\_enemyBlowupAnimation restoreOriginalFrame:YES];

id block = ^(){

\_isEnemyCollidable = YES;

};

id ac5 = [CCSequence actions:ac3,ac4,[CCCallBlock actionWithBlock:block], nil];

id action = [CCSpawn actions:ac1,ac2,ac5,blowup, nil];

[enemy stopAllActions];

[enemy runAction:action];

这段代码相比于之前的代码，主要新添了blowup动画动作，同时，在敌机爆炸效果结束之后，会运行一个回调方法，把\_isEnemyCollidable设置为YES，这样敌机就又可以参与碰撞了。

1. 添加玩家飞机销毁时的爆炸效果，如代码清单4-24所示。

代码清单4-24 给玩家飞机销毁添加爆炸效果

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

id blowup = [CCAnimate actionWithAnimation:\_playerBlowupAnimation restoreOriginalFrame:YES];

id action = [CCSequence actions:blowup,blink, [CCCallBlock actionWithBlock:^(){

\_isPlayerCollidable = YES;

//重新运行飞机动画

[playerSprite stopAllActions];

[playerSprite runAction:\_playerFlyAction];

playerSprite.opacity = 255;

playerSprite.visible = YES;

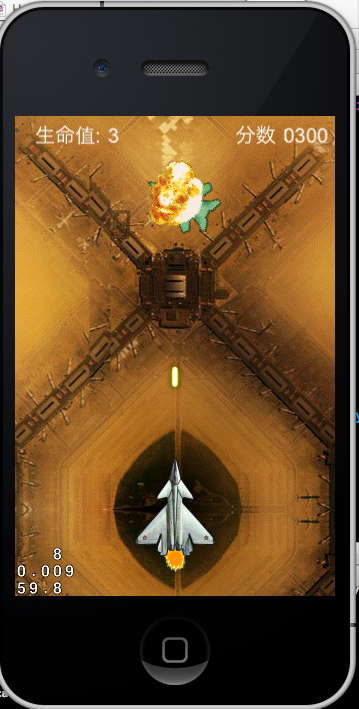
}],nil];

[playerSprite stopAllActions];

[playerSprite runAction:action];

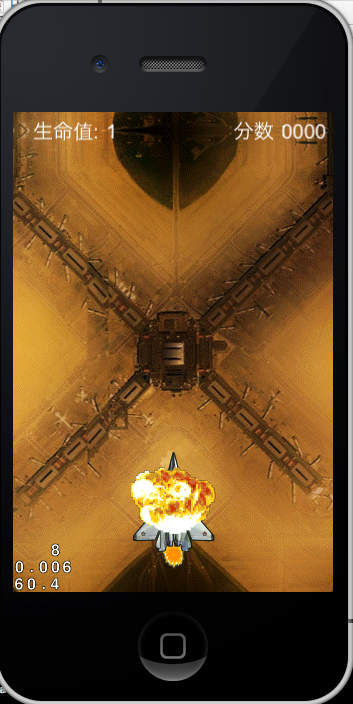
这段代码和之前处理敌机的爆炸效果差不多，只是在玩家飞机“隐身闪烁”之前先展示一个爆炸的效果，等闪烁完成之后，把玩家飞机的属性设置为可见，同时透明度设置为255。最后，由于播放爆炸动画效果的时候，把之前飞行动画效果覆盖掉，所以需要重新播放playerFlyAction飞行动画效果。

1. 编译并运行，此时可以看到敌机爆炸效果如图4-19所示。



1. 敌机爆炸效果

玩家飞机爆炸的效果如图4-20所示。



1. 玩家飞机爆炸效果

## 4.9本章小结

本章重点了解Cocos2D中实用且有趣的动作。在节点对象上使用动作，可以让应用和游戏显得更加生动；此外还学习了Cocos2D提供的多种特效，从而让节点对象产生更炫的效果；最后介绍Cocos2D中动画的实现，并通过一个实例讲解应如何使用Zwoptex工具实现角色的动画效果。同时，通过给第3章的垂直射击游戏添加飞行和爆炸效果，使得游戏的内容更加丰富和逼真了。

第5章将了解Cocos2D中的文本渲染系统，从而为设计完整的应用或游戏打下坚实的基础。