coco2d-iphone

# The Director(导演)

CCDirector类,简称Director(导演),是cocos2d游戏引擎的核心。 Director是一个单例:它保存着 cocos2d的全局配置设定,同时管理着cocos2d的场景。Director的主要用处如下:

1. 访问和改变场景

2. 访问cocos2d的配置细节

3. 访问视图(OpenGL,UIView,UIWindow)

4. 暂停,恢复和结束游戏

5. 在UIKit和OpenGL之间转换坐标

1.主程序启动,显示第一个场景的方法:(void) runWithScene:(Scene\*)scene;

2. 挂起当前当前正在运行的场景并压栈到代执行场景队列。将传入场景设置为当前执行场景:(void) pushScene:(Scene\*) scene;

3. 执行代执行场景队列中的最后一个场景,当前场景被释放:(void) popScene;

当代执行队列中没有代执行场景时,系统自劢退出,调用 end 方法。

4. 直接用一个场景取代当前执行场景,释放当前场景:(void) replaceScene:(Scene\*) scene;是经常实用的函数

5.结束场景运行:-(void) end;

 6.暂停场景运行:-(void) pause;画面还存在,时间任务停止。

 7.恢复场景运行:-(void) resume;

# 场景（CCScene）

－主要为了作为容器对象，没有附加任何特殊功能**,**基本上可以看成是层Layer对象的一个容器。

展示类场景:播放视频戒简单的在图像上输出文字,来实现游戏的开场介绍、 胜利、失败提示、帮助简介。

选项类场景:主菜单、设置游戏参数等。

 游戏场景:返是游戏的主要内容,除了返个场景以外的其他类场景基本上都是通用架构实现的。

每个场景都是通过不同的局(Layer)的叠加和组合协作来实现不同的功能的。因此，通常生个场景都是有一个或者几个层组成的。 用 Sence 来作为场景切换 ，基本上可以看成是局-Layer 对象的一个容器

第一个创建场景的地方是在AppDelegate中aplicationDidFinishLaunching方法

结束处。你在那里用Director的runWithScene方法开始运行第一个场景:

// 用以下代码运行第一个场景

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

在其它情况下,用replaceScene方法来替换已有的场景:

// 用replaceScene来替换所有以后需要变化的场景

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

用以下代码在任意一个地方显示“设置场景”: [[CCDirector sharedDirector] pushScene:[Settings scene]];

如果你身处“设置场景”,但又想关闭“设置场景”时,你可以调用popScene。 这样你会回到之前还保留在内存里的场景: [[CCDirector sharedDirector] popScene];

场景初始化一个过渡场景

CCFadeTransition\* tran = [CCFadeTransition transitionWithDuration:1

scene:[HelloWorld scene] withColor:ccWHITE];

// 使用过渡场景对象而不是

HelloWorld [[CCDirector sharedDirector] replaceScene:tran];

你可以把CCTransitionScene与replaceScene和pushScene结合使用,但是你不 能将过渡效果和popScene一起使用。

有很多种过渡效果可以使用,大多是和方向有关的,比如从哪个地方开始过渡 到哪个地方过渡结束。以下是目前可以使用的过渡效果和描述:

1. CCFadeTransition: 淡入淡出到一个指定的颜色,然后回来。

2. CCFadeTRTransition (还有另外三个变化): 瓦片(tiles)反转过来揭示场景。

3. CCJumpZoomTransition: 场景跳动着变小,新场景则跳动着变大。

4. CCMoveInLTransition (还有另外三个变化): 场景移出,同时新的场景从左边,右边,上方或者下方移入。

5. CCOrientedTransitionScene (还有另外六个变化): 这种过渡效果会将整个场景翻转过来。

6. CCPageTurnTransition: 翻动书页的过渡效果。

7. CCRotoZoomTransition: 当前场景旋转变小,新的场景旋转变大。

8. CCShrinkGrowTransition: 当前场景缩小,新的场景在其之上变大。

注:我先在这里警告一下:在游戏里不是每个过渡效果都很有用,即使它们看

起来很好看。玩家们最关心的是过渡的速度。即使3秒钟他们都会觉得长。我设

置过渡效果的时间不会超过一秒,或者干脆完全不用。

你绝对要避免在转换场景是随机选择过渡效果。玩家们不关心这些。而作为开

发者,你可能对于过渡效果太兴奋了。如果你不清楚该为哪个场景转换使用哪

个过渡效果,那就不要用。换句话说,可以使用并不代表你一定要用。

9. CCSlideInLTransition (还有另外三个变化): 新的场景从左边,右边,上方或者下方滑 入。

10. CCSplitColsTransition (还有另外一个变化): 将当前场景切成竖条,上下移动揭示新场 景。

11. CCTurnOffTilesTransition:将当前场景分成方块,用分成方块的新场景随机的替换当前 场景分出的方块。

# 层（CCLayer）

－为了处理输入问题

功能:接收iPone上的屏幕触摸touch操作输入、接收动力感知输入

直接提供了以下三个层:

ColorLayer颜色层(可以通过setContentSize设置层大小,改变颜色块的尺寸) 　　Menu菜单层(实例必须是MenueItem类或者子类的实例)

MultiplexLayer复合层 可以包含多个层的复合层。

为了让不同的局可以组合产生统一的效果,这些层基本上都是透明戒者半透明的。 文本(Label)、链接(HTMLLabel)、精灵(Sprite)、地图等等。其中,精灵是重点

主要功能就是接收iPhone上的屏幕触摸(touch)操作输入和动力感知(Accelerometer)输入层是我们写游戏的重点，我们大约99%以上的时间是在层上实现我们的游戏内容。 　 一个游戏主菜单画面是由3个层叠加实现的:Background layer、Animation layer、Menu layer 　 为了让不同的层可以组合产生统一的效果，这些层基本上都是透明或者半透明的，层的叠加有顺序的，如同事件响应机制。 　　Cocoa2d从技术实现角度提供一些公用层:处理菜单的菜单层(Menu)，处理颜色的颜色层(ColorLayer)等。 　　每一层又可以包含很多各式各样的内容要素：文本(Label) 链接(HTMLLabel) 精灵(Sprite) 地图等。

Layer 的主要功能在亍:

1)接收 iPhone 上的屏幕触摸(touch)操作输入。

2) 接收劢力感知(Accelerometer)输入。

CCScene和CCLayer还是存在一些显著的差异。CCScene仅仅是CCNode的继承类，而CCLayer则在CCNode的基础上添加了对多点触摸和加速计的支持

接收触摸事件

CCLayer类是用来接收触摸输入的。不过你要首先启用这个功能才可以使用它。 你通过设置isTouchEnabled为YES来让层接收触摸事件: self.isTouchEnabled = YES;

很多情况下,你可能想知道触摸是在哪里开始的。因为触摸事件由Cocoa Touch API接收,所以触摸的位置必须被转换为OpenGL的坐标。以下是一个用来转换坐 标的方法: -(CGPoint) locationFromTouches:(NSSet \*)touches

{

UITouch \*touch = [touches anyObject];

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

return [[CCDirector sharedDirector] convertToGL:touchLocation];

}

接收加速计事件

和触摸输入一样,加速计必须在启用以后才能接收加速计事件:

self.isAccelerometerEnabled = YES;

同样的,层里面要加入一个特定的方法来接收加速计事件:

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

{

CCLOG(@"acceleration: x:%f / y:%f / z:%f", acceleration.x, acceleration.y, acceleration.z);

}

# 定位点揭秘

每个节点都有一个定位点,但是只有当此节点拥有贴图时,这个定位点才有用。 默认情况下,anchorPoint属性设置为(0.5,0.5)或者贴图尺寸的一半。它是 一个抽象的因素,一个乘数,而不是一个特定的像素尺寸。

和你想的恰恰相反,定位点和节点的位置没有关系。虽然当你改变anchorPoint 属性的时候,你看到精灵在屏幕上的位置发生了变化。但那是错觉,因为节点 的位置并没有改变;改变的是精灵里贴图的位置!

anchorPoint定义的是贴图相对于节点位置的偏移。你可以通过把贴图的宽和高 乘以定位点来得到贴图的偏移值。顺便提一下,有一个只读的 anchorPointPixels属性可以得到贴图的像素偏移值,所以你不需要自己计算。

如果设置anchorPoint为(0,0)的话,你实际上是把贴图的左下角同节点的位 置对齐了。以下代码会把精灵图片完美地同屏幕左下角对齐:

# 贴图大小

我要特别提一下贴图大小。目前可用于iOS设备的贴图尺寸必须符合“2的n次方” 规定,所以贴图的宽和高必须是2,4,8,16,32,64,128,256,512,1024。 在第三代设备上可以达到2048像素。贴图不一定是正方形的,所以8x1024像素 的贴图完全没有问题。

在你制作贴图的时候你要考虑到上述尺寸要求,比如在为精灵准备图片时。让 我们马上来看看最坏情况下会发生什么事情:假设你的图片尺寸是260x260,用 的是32位颜色。在内存里,贴图本来只占279KB左右的空间,但是现在却使用了 整整1MB。

这几乎是原尺寸四倍的内存占用,这是因为iOS设备要求任何贴图的尺寸必须符 合“2的n次方”规定。260x260像素的贴图到了iOS设备中以后,系统会自动生 成一张与260x260尺寸最相近的符合“2的n次方”规定的图片(一张512x512像 素的图片),以便于把原贴图放进这个符合规定的“容器”中。而这张512x512 像素的图片占用了1MB的内存空间。

为了解决这个问题,你唯一能够做的是确保任何制作的图片尺寸符合“2的n次 方”规定。260x260像素的图片其实应该做成256x256像素。这样就不会浪费这 么多的内存。如果你有设计师为你工作,你要确保她按照要求制作。

# 使用动作(Actions)

节点可以运行动作。我会在以后多讲一些动作相关的知识。现在你只要知道动 作可以让节点移动,旋转和缩放,还可以让节点做一些其它的事情。

# 重复动作

你可以让动作或者一系列动作重复运行到永远。你可以通过这个特性生成循环 动画。以下代码会让一个节点永远旋转下去,就像一个永远旋转的轮子:

CCRotateBy\* rotateBy = [CCRotateBy actionWithDuration:2 angle:360];

CCRepeatForever\* repeat = [CCRepeatForever actionWithAction:rotateBy];

[myNode runAction:repeat];

# 舒缓动作

CCEaseAction类让cocos2d的动作更加有用。“舒缓动作”允许你改变在一段时 间内发生的动作效果。例如,如果你在节点上应用CCMoveTo动作,此节点在整 个移动过程中将会保持同一个速度。而如果你使用CCEaseAction的话,你就可 以让节点慢慢启动,然后加速向目标移动,或者反过来(快速启动,慢慢减速 到达目标)。或者你也可以让节点移动到超过目的地一些,然后再反弹回来。 “舒缓动作”可以帮助你创造出通常很费时间才能做况下才能看到舒缓动作的效果:

// 我想让myNode在3秒钟之内移动到100,200坐标点

CCMoveTo\* move = [CCMoveTo actionWithDuration:3 position:CGPointMake(100, 200)];

// 节点应该慢慢启动,然后在移动过程中减速

CCEaseInOut\* ease = [CCEaseInOut actionWithAction:move rate:4];

[myNode runAction:ease];

cocos2d实现了以下CCEaseAction类:

1. CCEaseBackIn, CCEaseBackInOut, CCEaseBackOut

2. CCEaseBounceIn, CCEaseBounceInOut, CCEaseBounceOut

3. CCEaseElasticIn, CCEaseElasticInOut, CCEaseElasticOut

4. CCEaseExponentialIn, CCEaseExponentialInOut, CCEaseExponentialOut

5. CCEaseIn, CCEaseInOut, CCEaseOut

6. CCEaseSineIn, CCEaseSineInOut, CCEaseSineOut

# 动作序列

以下代码演示了如何让一个标签的颜色从红色变为蓝色,最后变为绿色:

CCTintTo\* tint1 = [CCTintTo actionWithDuration:4 red:255 green:0 blue:0];

CCTintTo\* tint2 = [CCTintTo actionWithDuration:4 red:0 green:0 blue:255];

CCTintTo\* tint3 = [CCTintTo actionWithDuration:4 red:0 green:255 blue:0];

CCSequence\* sequence = [CCSequence actions:tint1, tint2, tint3, nil];

[label runAction:sequence];

你也可以将动作序列与CCRepeatForever动作结合使用:

CCSequence\* sequence = [CCSequence actions:tint1, tint2, tint3, nil];

CCRepeatForever\* repeat = [CCRepeatForever actionWithAction:sequence];

[label runAction:repeat];

一个挺有用的设置和停止选择器预定的方法。很多时候你需要在设置好的 预定方法里面停止调用某个指定的方法,同时因为参数和方法名可能发生变化, 你又不想重复相同的方法名和参数,这时你可以用以下的方法设置(预定的控 制器只会运行一次):

-(void) scheduleUpdates {

[self schedule:@selector(tenMinutesElapsed:) interval:600];

} -(void) tenMinutesElapsed:(ccTime)delta {

// 用\_cmd关键词停止当前方法的预定 [self unschedule:\_cmd];

}

# \_cmd关键词是当前方法的缩写