coco2d常用:

1.一个CCLabel对象用于显示文字:

CCLabel\* label = [CCLabel labelWithString:@"text" fontName:@"AppleGothic" fontSize:32]; [self addChild:label];

2.

右对齐label.anchorPoint = CGPointMake(1, 0.5f);

左对齐 label.anchorPoint = CGPointMake(0, 0.5f);

置顶放置 label.anchorPoint = CGPointMake(0.5f, 1);

放置在底部 label.anchorPoint = CGPointMake(0.5f, 0);

3.获取精灵图片的高和宽:

float imageHeight = [player texture].contentSize.height;

4.得到主角精灵和蜘蛛精灵之间的距离

float actualDistance = ccpDistance(player.position, spider.position);

5. \_cmd关键词是当前方法的缩写

6. 添加音频

#import "GameScene.h"

#import "SimpleAudioEngine.h"

以下代码展示了如何使用SimpleAudioEngine播放音乐和音频:

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

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

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

7 .addChild中的z参数:

决定了节点的绘制顺序。拥有最小z值的节点会首先被绘制; 拥有最大z值的节点最后一个被绘制。如果多个节点拥有相同的z值,他们的绘 制顺序将由他们的添加顺序来决定。当然,这个规则只适用于像sprites那样有 视觉表现的节点。

8.

规则一:永远不要在一个节点的init方法中调用CCDirector的replaceScene方法。

规则二:请遵守规则一。不遵守规则的后果是程序崩溃。Director无法容忍一个节点在初始化的同时进行场景替换。

9. 在过渡效果中使用LoadingScene可以优化内存的使用:因为你使用了一个简单 的过渡场景用于替换当前场景,然后用最终的目标场景替换这个过渡场景。在 这个替换的过程中,cocos2d将会有足够的时间来释放之前场景所占用的内存。 我们得到的实际效果是:不再会有两个复杂场景同时占用着内存的情况了,因 此在场景转换过程中也就减少了出现内存使用高峰的机会。

10. 为了正确地处理各自的触摸输入事件,我们需要用到 TargetedTouchHandlers。通过使用优先级参数,我们可以确保 UserInterfaceLayer在GameLayer之前接收到触摸事件。UserInterfaceLayer利 用isTouchForMe方法来决定它是否需要处理接收到的触摸事件,如果它处理了 触摸事件,ccTouchBegan方法将会返回YES。如果返回了YES,其它层,也就是 其它的触摸事件处理器,将不会再接收到当前事件。

// 将UserInterfaceLayer的触摸事件处理器的优先级设置的比GameLayer高

-(void) registerWithTouchDispatcher

{

[[CCTouchDispatcher sharedDispatcher] addTargetedDelegate:self priority:-1 swallowsTouches:YES];

}

// 检查是否触摸到的地方是在用户界面元素的区域里面

-(bool) isTouchForMe:(CGPoint)touchLocation

{

CCNode\* node = [self getChildByTag:UILayerTagFrameSprite];

return CGRectContainsPoint([node boundingBox], touchLocation);

}

11. CCProgressTimer (进度条)

CCProgressTimer\* timer = [CCProgressTimer progressWithFile:@"Icon.png"];

timer.position = ccp(100,100);

timer.type = kCCProgressTimerTypeHorizontalBarLR;

timer.percentage = 0;

[self addChild:timer z:1 tag:111];

// 进度条需要预约的更新方法来更新自身的状态

[self scheduleUpdate];

-(void)update:(ccTime)delta

{

CCNode\* node = [self getChildByTag:111];

// 更新进度条的时间显示

CCProgressTimer\* timer = (CCProgressTimer\*)node;

timer.percentage += delta \* 10;

if (timer.percentage >= 100)

{

timer.percentage = 0;

}

}

12. 在-(void) draw方法中的代码进行渲染时使用的z-order值是0。而且此方

法会在所有具有z-order为0的节点渲染之前进行渲染。这就意味着任何OpenGL

ES代码渲染得到的图形将会被其它也拥有z-order为0的节点所覆盖。在我们渲

染物体层里的长方形的代码中,我必须将tileMap的z-order值设为-1,这样才

不会将一开始用OpenGL ES代码渲染的长方形覆盖掉。

}