# 使用coocs2dx制作横版冒险游戏初体验

首先我们需要下载cocos2dx的引擎，附上链接<http://www.cocos2d-x.org/download>。下载后解压备用，然后配置cocos2dx Android开发环境。需准备好1、Python环境 2、Ant

3、SDK 4、NDK，附上配置攻略链接：

<https://blog.csdn.net/Marine_snow/article/details/71403702>。配置好后打开cmd，输入cocos new demo -p com.sduhyd.cocos2dx -l cpp -d 存放地址。回车生成cocos2dx的HelloWorld。

虽是Android程序，但因为代码是C++，所以一般在VS上编写代码。打开demo文件夹。cocos2d是工程文件，暂时无需改动。Classes与Resources是整个工程共享的资源，在不同平台运行时都要用到这两个文件夹下的文件。proj.android,proj.ios\_mac等显然就是跨平台所需的文件了。

现在打开proj.win32。这里讲个坑（猜测），在VS里如果没有配置cpp的生成路径，cpp会默认更新在此路径下，不会更新在Classes下。生成HelloWorld时，在Classes下是默认有四个关于AppDelegate与HelloWorldScene的文件。前者管理了整个cocos2dx程序的生命周期，后者是一个空的场景，就是程序打开的第一个界面。第一次运行后会在此路径下生成一模一样的四个文件。如果你在VS中修改了以上的代码，运行后Classes下的代码会发生相应改变，而proj.win32下的那四个文件却不会发生改变。然后其它的所有cpp都会默认更新到proj.win32路径下，不会更新到Classes下，所以这就很混乱。因为后期在Android平台上运行时，我们不可能去proj.win32下寻找cpp，目标必须是Classes下的文件（但是那个目录下又只有四个文件）。于是我傻乎乎地复制、粘贴、覆盖，然后我就gg了。（所以说代码修改后迅速上传到github保存很重要，说不定哪天电脑嗝屁就凉了）

首先介绍下AppDelegate，1、void AppDelegate::initGLContextAttrs()配置OpenGL啥的不太懂默认就行，2、static int register\_all\_packages()也不知道是啥，默认，3、然后是三个生命周期函数：启动、进入后台、从后台恢复。目前我只调整启动函数里的代码，里面默认有设置分辨率、设置帧率、打开第一个场景的入口。在这里我对player进行了状态的初始化。Player是玩家类，里面的成员用于1、判断玩家的状态 2、计算玩家每一帧的具体物理信息 3、计算某一帧某一个位置玩家自身的碰撞矩形。

然后是HelloWorldScene, Scene\* HelloWorld::createScene()、static void problemLoading(const char\* filename)无须赘述。重点是bool HelloWorld::init()，在里面编写相关代码就好。这个界面很简单就一些标签，参考<http://www.cocos2d-x.org/docs/cocos2d-x/zh/ui_components/>。

接着点击start进入下一个场景FirstScene，直接看init()。加载音乐不赘述。我们需要先生成一个GameLayer图层，实际上游戏中的元素大部分是放在图层上的，然后图层再放在场景上。目前来说场景只是一个比较抽象的东西。我们载入事先做好的瓦片地图，将其放入图层。再将玩家放在瓦片地图上，这样瓦片与玩家才能进行碰撞检测。其实瓦片地图自身也有很多层，有的层我们想让它触发碰撞条件，比如大地、墙体等，有的图层只是装饰，比如天空，云彩等。所以每一层都要提取出来分层处理。接着编写触摸监听器和点击事件，同样监听器要放入图层中。最后是由场景调用scheduleUpdate()，顾名思义，按照时间表更新。个人猜测应该是根据帧数决定更新画面的频率。具体的逻辑我们需要重写一个update()方法。

我们看void FirstScene::update(float delta)，浮点数是一帧所花费的时间。场景更新一个很重要的地方就是player的状态更新，所以我们这里调用了player的update()方法，这个方法的目的我们之前提到过了也就是第二点。里面的变量全是向量，然后再搭配上cocos2dx原有的计算函数进行计算得出结果。角色状态更新后判断是否有触发游戏结束的条件，如果触发则跳出，无需进行接下来的碰撞检测。游戏结束的函数后面讲。重点是接下来的check系列方法。

进入void FirstScene::checkForAndResolveCollisions(Player\* player)。首先调用的函数是getSurroundingTilesAtPosition，传入玩家下一帧的坐标和瓦片地图碰撞层。将坐标转化为瓦片地图坐标。通过这个坐标获得八个瓦片的信息，信息就是每个瓦片的顶点坐标以及长宽。这里用自定义的TileRect封装，也就是八个瓦片的碰撞矩形。然后将这些TileRect对象存入数组，并返回。然后通过for循环以之前设定好的优先顺序依次取出八个碰撞矩形。接下来就是进行判断以及选择是否修正位置，这部分内容结合代码注释看比较好。我解释下为啥要加两个判断语句if (intersection.size.height > 0)和width。这里又是cocos2d的一个坑。起因是这个函数intersectsRect，即使交叉碰撞矩形是一条线，甚至是一个点，它返回的也是true。而实际上那不是矩形，也就是说根本没有发生碰撞。当时出的问题就是玩家出场落地后，前进一丢丢后就卡住不动了，仿佛前方有障碍物，实际上只是玩家与地面右侧的瓦片有了一个点的交集，系统误判是撞墙了。这个坑就是这样。如果得到正常的交叉碰撞矩形，则立刻修正desiredPosition(下一帧的位置)和velocity(速度)，相当于预判了下一帧的位置，如果要穿墙了，则把它拉回来，每一帧重复此动作，就能实现玩家站在实体上，遇到障碍物停止的效果。后面有四行打印tgid的无需理会，是我当初在控制台打印用的，没有影响。

到这里重点内容讲完了另一个check方法就是检测是否撞陷阱了，跟第一个check一样的原理，还是简化版。

最后是void FirstScene::gameOver(bool isWon)，这个是处理游戏结果的函数，输赢都是调用这个，具体逻辑取决于传入的布尔值。弹出”You Win!”标签或者”You Died!”标签。这里加个“华丽的特效”，也是全场唯一的特效，标签从屏幕底部升上来....emmmmmm然后加个replay标签按钮用于返回第一个界面，游戏结束。