Bombing Adventure

Development Diary

刘洋

Preparation: cocos2d-x 3.0

Visual Studio 2017

Qt Creator

Git

Day1(可作废): Objective: create new game scene based on project HelloCpp.

Rewrite Class: HelloWorld → MainScene /* These two class are inherited from cocos2d::Layer. */

Rewrite Functions: (1) virtual bool init() /* Initialize the instance */

(2) static __TYPE__ *create() → static MainScene *create1()

/* Create a new layer */

Explanation: CREATE_FUNC 是 static __TYPE__* create()函数的宏。对于 Scene、Layer、Sprite 这些类，都可以使用 create()创建并初始化一个对象，并且返回该类型的指针。

Errors occur: (1) Cannot use member function without object

Solution: declare those member function as static

(2) Followed by (1): Multiple Definition of … (functions).

Solution: ?

Day2 (2018/4/29): Objective: solve the day1 error.

1. 今日任务：重写 HelloWorld 类（增加菜单 menu）。定义其他两个场景。

(1) 理解 HelloWorld 类：两种源码写法。

第一种继承自 Layer 类：class HelloWorld : public Layer, 成员函数里的 CREATE_FUNC(HelloWorld) 调用的是 Layer 类的 create()。HelloWorld::scene()方法是关键，先调用 Scene::create()创建空场景，然后调用 HelloWorld::create()创建一个 layer。函数栈调用 HelloWorld::init()，对此 layer 进行初始化。最

后，addChild 将这个 layer 加入 scene，返回场景的指针。

第二种继承自 Scene 类（也就是 github 上的那个版本）：class HelloWorld : public Scene，此时的 HelloWorld::scene()就直接返回 HelloWorld.create()了（因为 create()直接创造一个场景）。调用 create()时，函数栈 push 进 HelloWorld.init()，对生成的场景进行初始化。

这两种方法的区别是：第一种生成的 scene 只有一个包含 layer（因为 HelloWorld 继承自 Layer），而第二种生成的 scene 可以在 init()里随意定义，例如添加多层 layers/menues/sprites…所以推荐使用继承自 Scene 的 HelloWorld，因为场景最好由多层 layers 组成（便于 node 控制）。

（2）给 HelloWorld 改名：replace all “HelloWorld” with “BombingAdventure”.

（3）重写 BombingAdventure::init().因为现在只有一个 helloCPP 的初始场景。

因为 BombingAdventure 是游戏初始场景，所以最主要的就是做个菜单，里面有 start、help、close 这些选项。下面是菜单的写法。

在 resources 目录下添加 start.png (如图)、rules.png 等。

然后创建菜单项（用 MenuItemImage::create()方法），接受图片文件名（两个，后一个是 selected）和一个方法参数（反正就是告诉你，点了这个 Item 之后会发生什么）一共三个参数。一共创建了 startItem、rulesItem 和 closeItem（自带）这三个菜单项。

之后，用 Menu::create(item1, item2, ...)初始化一个 menu 对象。所以就创建了一个由上述三个 menu item 组成的 menu。然后用 addChild 把 menu 加进场景里。

接下来定义 menuItem 的功能（也就是 create()时候接受的第三个参数）。以 ruleItem 为例，此函数名为 BombingAdventure::menuRulesScene()，接受一个 Object* sender 对象（应该就是鼠标事件吧）。函数内容为，获得*director 单例 getInstance()，然后用->调用 pushScene()方法。pushScene()接受一个*scene 参数，在这个例子里是一个表达式 RulesScene::scene()，也就是等下要写的 RulesScene 类对象初始化方法。

pushScene 这个函数的具体功能就是在场景栈里 push 一个新的场景（原来的先挂起来）。类似的还有 popScene()、replaceScene()等等。

（4）定义下一个场景类。

在 BombingAdventure 主界面中提供了退出、开始游戏、规则说明三个 botton。所以下一步要写 GameScene 和 RulesScene，此处以 RulesScene 为例。

Header: 直接复制粘贴 BombingAdventure.h，然后替换一下类名。注意，由于在 RulesScene 场景里只设置一个“返回主菜单”的 menu item，所以要把 BombingAdventure 的 menu 方法去掉，换成 menuPopBack()这个自定义方法。

Source: scene()函数相同。init()里，把原来的 menu items 都去除，用 Sprite::create()创建一个图片式的规则说明（如图）。加一个返回主菜单按钮。然后写 menuPopBack()方法：调用*Director 的 popScene()方法，pop 掉现在运行的 RulesScene，返回到 BombingAdventure 场景。

2. day2 Problem: 这里出现了几个折磨了我三小时的 errors: multiple definition of…，检查头文件保护符问题，函数也是遵守 header 内声明，source 内定义的原则。后来发现是.pro 里多次加入了 header files，问题解决。鉴于后来开发不使用 Qt Creator(或 qmake)，所以这个问题可以忽略不计。

3. 已完成：菜单创建，场景创建。如图。（请忽略 backgrond&sprite…）

4. 难点： 坐标系问题：如何适配不同终端的屏幕尺寸、分辨率等，在不同设备上保持游戏界面相同。

   多平台、cmake

5. Day3 目标： Sprite 相关: Animate the sprite.(可能吧。。)

Day3(2018/5/1): Objective: 在 VS 上通过编译。上传到 github。

1. 把 Bombing Adventure 的仓库 clone 到本地，然后重写 HelloWorldScene、AppDelegate，新建 GameScene.cpp/.h、RulesScene.cpp/.h 文件。
   按下“调试”按钮。

   Problem 1 出现：提示找不到指定的文件。

   Problem 1 解决过程：查看 VS 输出窗口-生成，发现报错 cannot find source file: GameScene.cpp/ RulesScene.cpp: no such a file… 打开 proj.win32 目录下的 vsxproj.filters 文件，发现未找到的两个 cpp 文件还没有设置路径。设置：../Classes/{name}.cpp (name 就是那两个文件的名)（原来的没有指定到上一级目录下的 classes 文件夹，所以编译器直接就在 proj.win32 这个目录里找）。再次编译仍报错。百度之后，得知原因是没有把新建的那四个文件加入项目里。在 VS 里右键 src 文件夹，选择添加-现有项，拖进去。找不到文件的问题解决。

   Problem 2 出现：提示大量 LNK2001 和 LNK2019 错误。一共有 217 个无法解析的外部符号。

   Problem 2 解决过程：明显是链接问题。详细查看了一下没链接到的外部符号，发现全部来自外部依赖项(比如 MenuItemImage.h 这些属于 cocos 引擎的文件)。查询资料后，点击项目-属性-链接器，将“链接库依赖项”的“否”改成“是”（感觉这个选项默认就是“是”，可能之前手贱误操作了）。确定后，点击绿色三角形符号，编译成功，生成 BombingAdventure.exe 并出现游戏界面。

2. 已完成：把之前在 Qt Creator 上做出的界面在 VS 里实现，所使用的仓库是 fork 自 Bombing Adventure 小组。

3. 待解决：上传到 github。

4. 难点：新代码能否在 mac OS + Clion 上运行。