个人资料

房燕良

+ 加关注 发私信

博客专家

访问： 125162次
积分： 1743
等级： BLOG 4
排名： 第12304名

原创： 27篇 转载： 0篇
译文： 1篇 评论： 160条

原 《炉石传说》架构设计赏析(2)：Scene管理

分类： Unity3D 2014-09-13 11:58 7041人阅读 评论(6) 收藏 举报

unity3d 游戏

欢迎来的我的酒馆，快来火炉旁暖暖你的靴子。哈哈，我们继续欣赏炉石的代码。欢迎转载，请注明作者【燕良@游戏开发】及原文地址：http://blog.csdn.net/neil3d/article/details/39231541

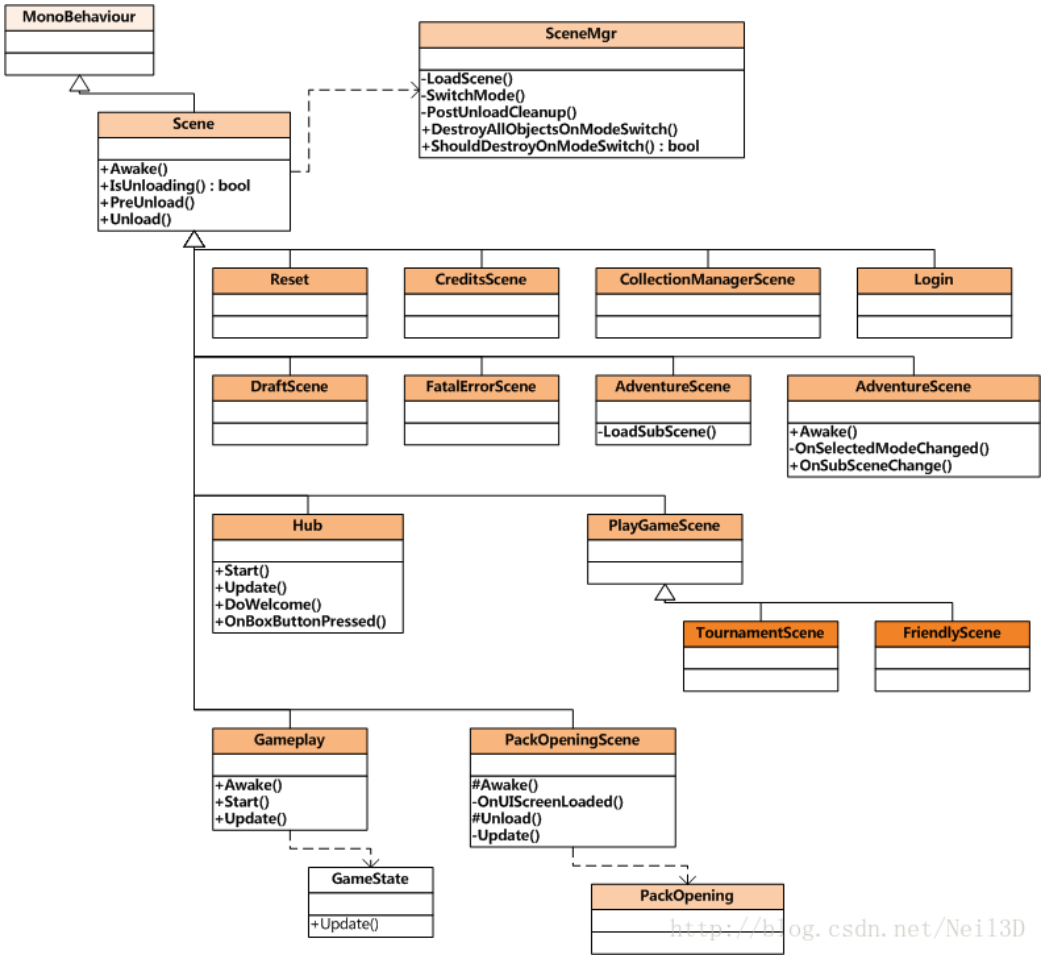
上篇文章我们分析到Scen cene的加载工作，今天我们主要分析一下炉石这款游戏中一共有哪些Scene，他们各自负责什么的逻辑、UI的处理方式。

在正式开始之前，我来对 cene切换再做一些补充分析。前文中我们看到SceneMgr是调用了“ Application.LoadLe sync(this.sceneName);”，那内存中的东西岂不是越搞越多吗？我们再仔细看一下SceneMgr:Switch 数，它是一个Coroutine，他主要进行了下面这几个步骤的操作：

1. 调用当前Scene的Scene:PreUnload()函数；
   发送FireScenePreUnloadEvent事件；
   等待直到Unload过程走完（通过检测LoadingScreen的阶段）；
2. 调用Scene:Unload()函数；
   发送FireScenePreUnloadEvent事件；
   调用成员函数PostUnloadCleanup()函数，它调用了两个关键的函数：
   - 首先是成员函数：DestroyAllObjectsOnModeSwitch()，这个函数查找到所有的GameObject（Object.FindObjectsOfType(typeof(GameObject))），然后进行了筛选（通过成员函数ShouldDestroyOnModeSwitch），除了一些全局对象之外（主要是SceneMgr、PegUI、Box、DefLoader），全都删除了(通过调用Object.DestroyImmediate())。
   - 然后调用了：Resources.UnloadUnusedAssets()；
3. 然后是调用前文提到过的成员函数：LoadModeFromModeSwitch()，进行了LoadLevelAdditiveAsync()操作；

综上所述，炉石的Scene切换主要是包含两步：1删除所有非全局对象，卸载未引用的Asset；2加载新的Scene。（我倒是想到另外一个土鳖一点的替代方案：创建一个完全空的scene，调用LoadLevel加载它，那么所有没有设置"DontDestroyOnLoad"的对象就都被删除了。）

除了前文提到的Login，我们可以看到Scene还有很多派生类，详见下图：



这是我猜测的这些类和游戏内容的对应关系，没有太仔细分析，可能有些对应是错误的：

下面我们就挑选一个简单的Scene来分析一下它的内部运作机制，我们来看一下AdventureScene吧。

Adventure相关的Class很多，我们只做一个粗略的分析，只涉及到下面这几个类和接口：

首先我推测，在Hub屏幕中点击中间的【Solo Adventure】（冒险模式）按钮之后，通过我们前文分析的LoadScene流程，加载了一个冒险模式相关的Scene。它里面有一个GameObject绑定了“AdventureScene”这个脚本，我们可以看到AdventureScene:Awake()方法中主要是注册了很多事件的回调。

我们可以看到有一个“AdventrueSubScenes”枚举，它基本上对应了下图中的按钮：

```csharp
public enum AdventureSubScenes
{
    Chooser,
    Practice,
    MissionDeckPicker,
    NormalHeroic,
    ClassChallenge
}
```

接下来还有一个"AdventureSubScene"是处理子场景对应的一些逻辑的。
我们看到有“AdventureChooserTray”这个类：

- 我推测这个类就是用来处理上面这个游戏画面的UI交互操作的；
- 这个类在Awake时，通过调用“CreateAdventureChooserButton()”方法创建了上图中的上部分那几个冒险游戏内容模式相关的按钮；
- 这些按钮都绑定了事件回调：AdventureChooserTray.ButtonModeSelected()；当这些按钮被点击时，主要是调用：
    - AdventureConfig:SetSelectedAdventureMode()，此函数修改内部数据之后触发事件：FireSelectedModeChangeEvent()
    - AdventureChooserTray通过OnSelectedModeChange()响应此事件，这也就是点击上面那几个按钮之后要做的一些处理：包括更新左侧的画面、设置Choose按钮状态等等；
      其中调用了PlayMakerFSM，主要是向其发送事件“Burst”；通过这里，我们可以确定炉石使用了PlayerMaker插件。
    - AdventureScene也通过OnSelectedModeChanged()相应了此事件；
- 它里面还有一个“PlayButton m_ChooseButton”成员变量，并把为它添加了EventListener，用来调用ChangeSubScene()方法。这就和游戏实际的操作对应上来：在上面选择模式，然后点击下面的【Choose】按钮，就进行到下一步的选择了。
- AdventureChooserTray:ChangeSubScene()通过Coroutine的方式调用了AdventureConfig:ChangeSubSceneToSelectedAdventure()，然后调用了AdventureConfig:ChangeSubScene()；它主要触发两个事件：
    - FireSubSceneChangeEvent：AdventureScene通过OnSubSceneChange()函数响应它，主要是调用AdventureScene:LoadSubScene()，内部主要是调用AssetLoader.LoadUIScreen()；
    - FireAdventureModeChangeEvent：AdventureScene通过OnAdventureModeChanged()响应它。

通过上面的分析，我们大致了解了上面这个游戏截图中的操作实现逻辑。
这次的分析算是一次热身，接下来重点分析有两个方面：

- 游戏逻辑的组织，特别是技能的数据、逻辑组织；这可能需要经过多次尝试，慢慢接近；
- 游戏的Asset资源管理、加载机制；

OK，今天的分析就到这里，欢迎大家拍砖。后续分析敬请期待！
顺便来秀一下我的鱼人部队，别看这些1点学的小东西，加在一起还蛮欢乐的！

上一篇 《炉石传说》架构设计赏析(1)：游戏启动流程
下一篇 《炉石传说》架构设计赏析(3)：Gameplay初探

查看评论

4楼 ZeroDeep 2015-08-12 16:28发表

QQ群多少啊

3楼 青蛙果果 2015-02-05 11:10发表

我是c++初级吧，问个可能比较傻的问题吧。
在这个游戏中，自己可以保存的套牌最多有九套，这个从一开始就被很多玩家吐槽了， 希望增加卡槽。 据说开发组表示增加卡槽非常的困难。 这个我不太理解， 结构不是完全相同吗？为何会很困难。自己的套牌也不涉及和其他玩家的交互问题， 这个我很困惑。单纯的增加记录也不应该会占用非常大的存储空间吧， 最多就是 （30 + 1 + 1）* sizeof(DWORD) 吧。

2楼 zephyr1125 2014-10-06 14:51发表

AdventrueSubScenes的枚举应该不是对应那些按钮的。而是对应冒险模式下的几种子界面：
Chooser - 冒险模式选择,
Practice - 练习模式难度选择,
MissionDeckPicker - 冒险用的牌组选择,
NormalHeroic - 普通/英雄模式关卡选择,
ClassChallenge - 职业挑战模式关卡选择。

Re: 房燕良 2014-10-07 11:31发表

回复zephyr1125：有一起研究的了，太好了。有空到我的QQ群里来交流一下吧。

1楼 Golden_Shadow 2014-09-29 13:44发表

哈哈.... 分析得好.另外,还真是会玩啊...

Re: 房燕良 2014-09-29 17:12发表

回复Golden_Shadow：做游戏的嘛，多少也要会玩一点。：）

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker
OpenStack VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC
WAP jQuery BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML
LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra
CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App
SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP
HBase Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap