**大型角色扮演项目剧情编辑器系统的开发**

吴培健

# 一、绪论

## 1. 引言

游戏发展至今天，为游戏添加故事和背景的做法已经获得了业界的广泛认同。特别是角色扮演类型的游戏，一个好的剧情过渡可以让玩家融入到游戏里面去，并因此获得更好的游戏体验。在游戏体验中，剧情是让玩家沉浸其中的关键环节，它为玩家提供了一个最基本的背景，以及一个清楚明确的目标，可能是寻找宝藏，可能是打败boss。这是这些剧情，让玩家搞清了他们玩游戏的意义。一个完整的剧情CG，是由许许多多的单元指令构成的，每个单元指令实现一种具体的功能，例如，创建NPC指令，移动镜头指令，播放特效指令等等。这些单元指令所具备的功能由剧情编辑员规划，前端程序员搭建好剧情编辑系统的框架，再根据需求实现多种CG指令。剧情编辑员合理的将这些指令组合成一个个过渡剧情CG，配置这些剧情CG的触发点，这样，剧情编辑系统会在游戏进行到某个阶段启动运行剧情CG，使玩家在玩游戏的过程中更有代入感。

## 2. 技术优势

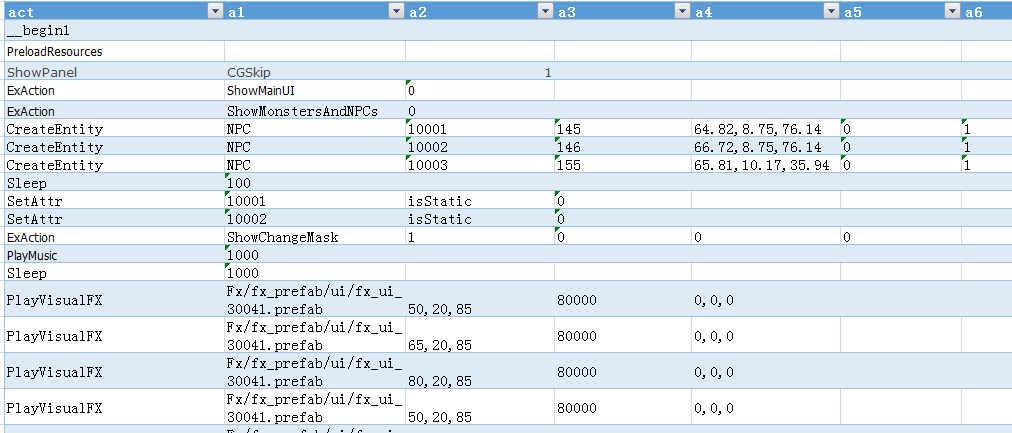
该剧情编辑系统的每一个CG指令都只实现一个明确的功能，满足单一职责原则，指令之间没有数据依赖，逻辑依赖。可复用性高，这些CG指令都是可复用的，剧情编辑员配上不同的参数，触发时机和顺序就可以获得不一样的效果。可扩展性高，当现有的CG指令不满足需求时，可以通过增加指令的方式对系统进行扩展，而这些扩展不会影响已有的指令。另一个优点是易维护，当一个CG播放出错，定位到出错的CG指令并对其进行调试即可。

# 二、剧情CG制作流程

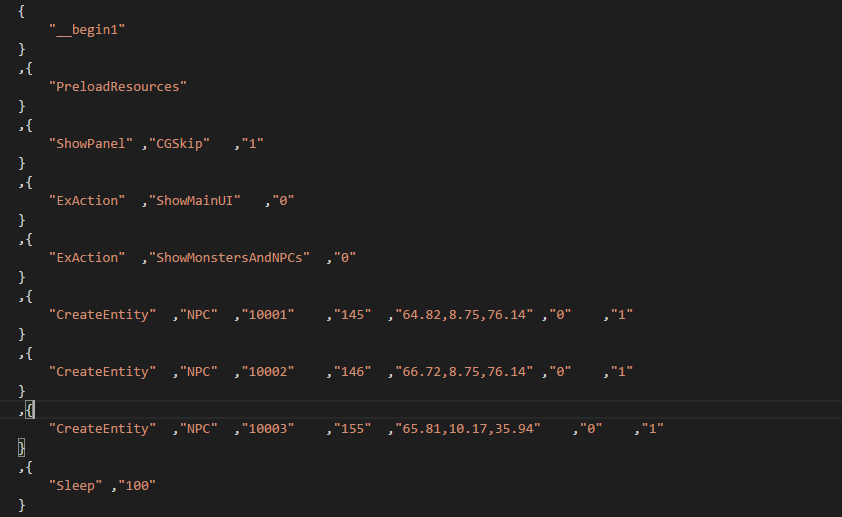
## 1. 制作流程

（1） 剧情编辑员事先设计好CG剧本，然后开始准备播放CG过程中需要用到的资源，首先是角色资源，确定好当前编辑的CG中会出现的所有人物或物品，将这些信息配置在表中，剧情编辑系统会在预加载阶段遍历所有创建角色的CG指令，读到这些配置好的数据将模型资源先加载好。其次，一般剧情都会有旁白或者人物对话，这些剧情都将由剧情编辑员组织并配置到对应表中，由剧情编辑系统在播放过程中显示。接着，需要为这个CG准备多个镜头的切换，优秀的镜头可以让玩家有一种看电影的感觉。同理，有专门的CG指令负责镜头的移动及切换。另外一个可选的是准备特效资源，例如一个涉及到爆炸的剧情需要有爆炸的特效，剧情出现合适的特效将为游戏增色不少。特效资源由美术人员制作，剧情编辑员将特效的路径配置给播放特效的指令作为参数。这样一个剧情CG需要的前期准备工作就大致完成。

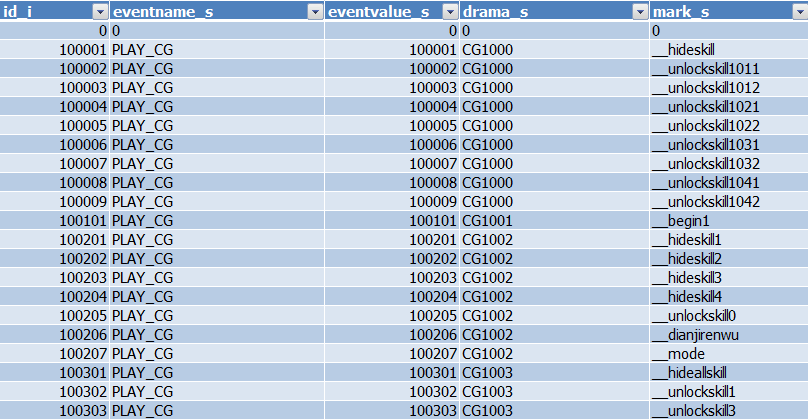
（2） 准备好必要的资源，剧情编辑员就可以开始着手配置如下的CG表，每一行是一个基本的CG指令，每一个指令根据实际参数去做特定的事情，例如创建NPC，打开界面，移动镜头，播放音效等等。



（3） 将（2）步骤中创建的Excel表导出成xml表，再使用XmlToLua.exe转换成lua文件。在游戏中使用该lua文件，需要在drama\_trigger\_cg表中为CG配置触发条件。一个lua文件即是一个剧情CG。最终生成的lua文件如下：

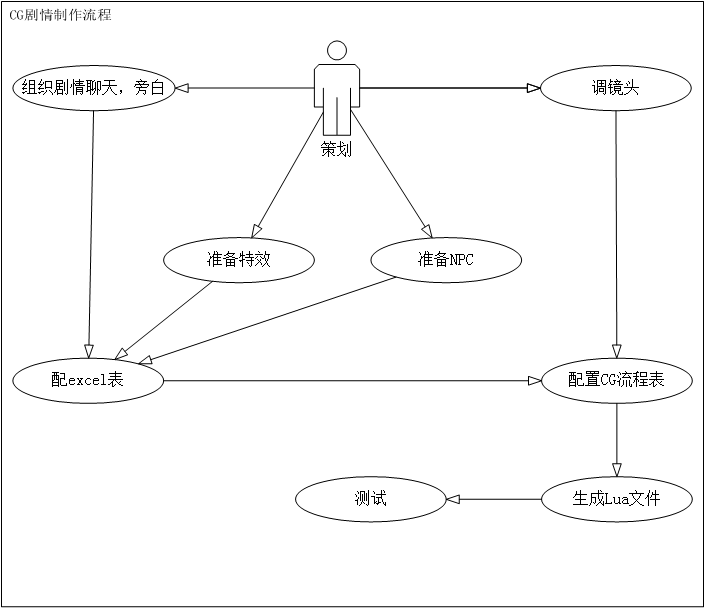


（4） 触发条件配置，剧情编辑系统根据eventname，eventvalue决定触发时机，根据drama决定触发哪个CG：



（5） 生成lua文件后，一个完整的剧情CG就基本实现完成，最后就将CG放进游戏测试结果是否符合预期，根据在游戏中看到的实际效果对CG进行调整，直至符合预期的效果。

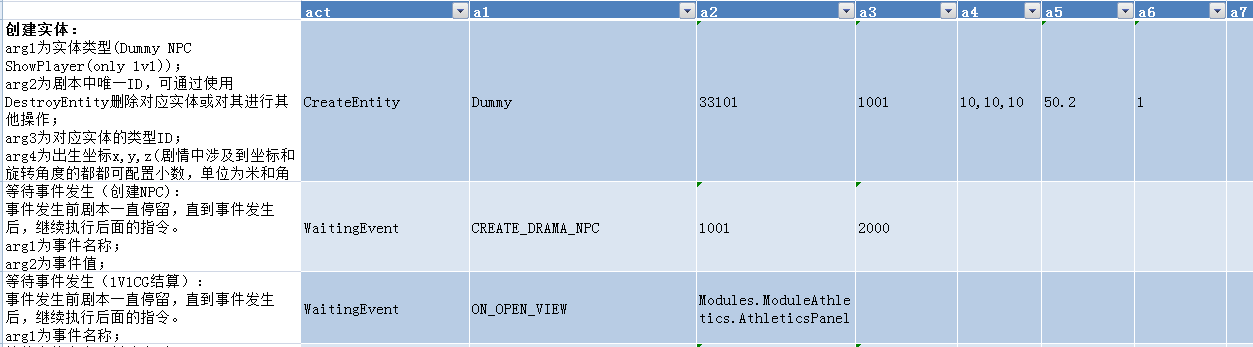
## 2. 流程图



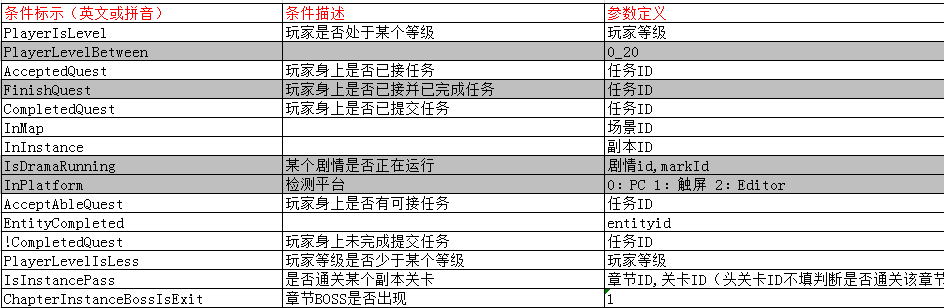
## 3. CG指令的实现及扩展

CG指令是组成剧情的基本单元，每个基本指令都实现一种特定的功能，并且都相互独立。剧情编辑员需要某一个功能，例如让一个角色移动都某一个地点。前端程序员在原有的指令系统上增加一个移动角色的接口，与剧情编辑员协商好必要的参数，例如移动哪个角色，目标位置，移动速度等。在此基础上实现接口功能，测试。最终剧情编辑员就可以在CG中使用这个指令。为防止日益增长的需求使CG指令过于臃肿庞大，该剧情编辑系统将CG指令进行分类：用于判断条件定义的Judge指令；等待事件发生的WaitingEvent指令；还有ExAction扩展指令。下面展示部分指令说明：

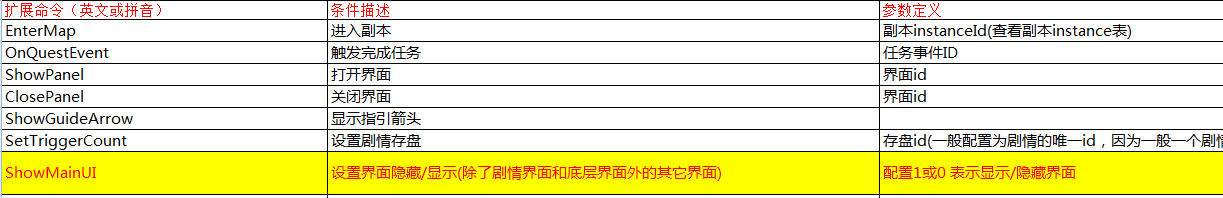
基本CG指令：



Judge指令：



ExAction指令：



# 三、剧情编辑系统

## 1. 组件介绍

（1） DramaManager：DramaManager是剧情编辑系统的管理器，其中Init()方法为播放CG做初始化。首先将DramaManager的Update方法注册到GameWorld.Update中，每帧调用。接着调用DramaTriggerData的InitData()加载drama\_trigger\_cg.lua，drama\_trigger\_guide.lua文件的数据，这两份文件记录了CG，指引的播放条件。然后调用DramaTrigger的Start()对游戏中所有可能引起CG播放的事件进行监听。最后注册自己的监听：监听离开场景时停止CG播放；监听点击跳过按钮时跳过CG。至此，剧情管理器的初始化操作完成。接下来分析一个CG是如何被播放的，当触发一个CG时（怎么触发将在DramaTrigger类中说明），最终是调用DramaManager的Trigger()，Trigger()首先调用TriggerWaitEvent()方法将事件传递到DramaManager管理的正在播放的所有Drama中，因为有可能当前的Trigger类型是某些Drama在等待的事件，这部分可以在Drama更深入了解。如果是Trigger播放CG类型，则根据参数在DramaTriggerData找到具体要播放哪一个CG，如果有找到则调用RunDrama()开始准备播放CG。RunDrama()根据Trigger()传过来的参数创建一个Drama对象，接着调用Drama的Start()方法开始与DramaManager共同完成该CG的播放：Drama的Start()方法将Drama的Update()方法注册到DramaManager管理器的Update方法中，由DramaManager管理器通过不断调用Update方法执行CG指令。

（2） Drama：Drama类保存了一个CG的所有指令以及各种运行状态，例如：一个\_curActionsPos表示当前CG的播放进度；\_state表示当前CG处于运行，未运行或加载状态；\_pastTime表示CG从播放开始到此时此刻经过了多长时间。\_sleepEndTime实现暂停播放指令。等等的这些状态变量共同保证CG的正常播放。DramaManager负责监听CG的播放时机，并且管理多个CG的播放，一旦监听到有CG可以播放便调用Drama的Start()开始播放CG，Start()开启CG播放的逻辑如下：根据drama\_trigger\_cg.lua配置的markName计算CG开始播放的位置，将CG的运行状态设置为运行中，将Update()方法添加到DramaManager:Update()中，这可以保证CG一直运行下去直到结束。调用一次Update()开始播放CG。接下来所以CG指令都再Update()中根据剧情编辑员的配置执行，CG播放的时候都会首先使用一个“PreloadResources”指令来预先加载资源，这个指令将调用DramaResources类的接口将播放CG过程中要用到的资源先准备好。在Update()中还会做一个重要的判断CheckSleepTime()，当播放到CG指令“Sleep”时，这个方法会返回true，此时CG会暂停继续往下播放，直到“Sleep”指令配置时间段结束，这个方法会返回false，我们的CG就可以继续往下播放了。在一个指引（指引也是一种CG）播放过程中，会出现这样的需求，等待某个界面出现再继续执行剩下的CG指令，这种需求通常使用“WaitingEvent”指令等待某个条件的达成， CG不继续往下播放直到它等待的条件达成；剧情CG也同样有“WaitingEvent”指令但参数略有不同，可以为CG指定一个超时时间，过了这个时间也还没等到指定的事件，CG还是可以继续往下播放。在Update()中使用CheckWaitingEvent()来判断当前的CG是否在等待某个条件，是的话则等待。这些CG等待事件都由DramaTrigger:Trigger()方法触发，DramaManager接收到这些事件并通知所有正在运行的Drama进行处理，每个Drama只会处理它当前等待的事件，其余的事件并不关心。

（3）DramaWaitingEvent：该类代表了一个WaitingEvent指令，保存了事件名字（eventName）,事件值（eventValue）,超时等待时间（waitingTime）。

（4）DramaJudgment：该类用来处理所有Judge指令，Judge指令是用来判断某个条件是否满足的指令。举个例子，当剧情编辑员配如下数据：{"Judge"，"CompletedQuest:90004" ,"End" ,"null"}，则代表这是一条Judge指令，需要判断的条件是（"CompletedQuest:90004"）是否完成任务90004，判断成功则执行“End”命令，否则执行“null”命令。该类的核心方法是ExecuteCondition(conditionName, conditionValue)方法，该方法根据条件名称参数执行相应的条件判断方法，DramaJudgment中会实现这些配置的条件方法，例如CompletedQuest()，ExecuteCondition()会调用CompletedQuest()方法，返回true或者false。Drama使用该类来处理Judge指令并决定后续的处理。

（5） DramaDataPool：该类提供接口GetDramaData(dramaName)用于根据CG名字等加载CG数据，并且维护了一个table(dramaDataDic)用于缓存加载过的CG数据。同时提供接口可以清除缓存。

（6） DramaResource：DramaResource维护一个table（\_dramaGetMap），存储了预加载方法GetCreateEntityRresource()，GetPlayVisualFXResuorce()，GetPlaySkillResource()，GetAnimatorController()，其作用是在剧情开始播放之前预先加载实体，特效，技能等资源，这样可以保证剧情的流畅地播放。预加载功能是通过该类的PreloadDramaResources(dramaNames, callback)接口提供的：首先从DramaDataPool中获得Drama数据，根据Drama数据获得所有要预加载的GameObject的路径并加载这些资源。

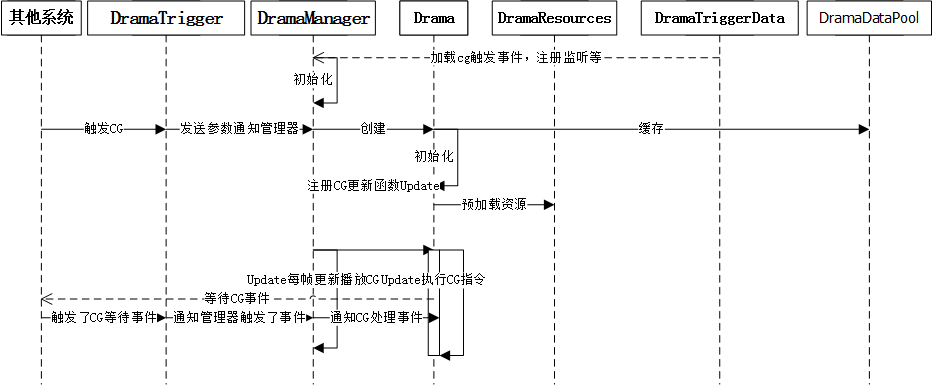
（7 ）DramaTriggerData：该类加载了drama\_trigger\_cg.lua和drama\_trigger\_guide.lua两个文件的内容。这两个文件通常是由剧情编辑员来配置（做法和配置CG类似），存储的是游戏中每一个CG和指引（也是一种CG）的触发条件，以及这些CG从哪个指令开始执行。该类还提供了接口GetDramaTriggers(eventName,eventValue)来获取数据。

（8） DramaExAction：该类用来执行ExAction扩展命令，即所有参数名为ExAction的CG指令。ExAction命令有多种，例如“HideSkillBtn”代表执行隐藏技能按钮，“EnterMap”代表进入副本，“ShowPanel”代表打开界面。实现这些指令的功能的方法也放在这个类中。核心方法ExcuteExAction(drama,actionName,actionValue)根据配置的命令名称actionName以及命令参数actionValue调用特定方法。

（9） DramaTriggerType：定义了触发CG的事件类型，以及触发CG等待的事件类型。

（10） DramaTrigger：该类对游戏中所有可能引起剧情播放的事件进行监听，若监听到一个可以播放的事件，则收集参数，确定DramaTriggerType的类型（这些类型再DramaTriggerType中定义），将参数传入到DramaManager的Trigger方法，在该方法中，从DramaTriggerData中查找是否存在可以触发的CG，若找到则播放，否则不做任何处理。

## 2. 流程图



# 四、小结

该剧情编辑系统使用了观察者设计模式，监听器负责监听其他系统行为，当出现剧情系统感兴趣的行为时将通知剧情管理器进行处理，是一种比较合理的设计方案。各个模块之间耦合度低，复用性高。该剧情编辑器还具有良好的扩展性，可应对复杂多变的需求。