

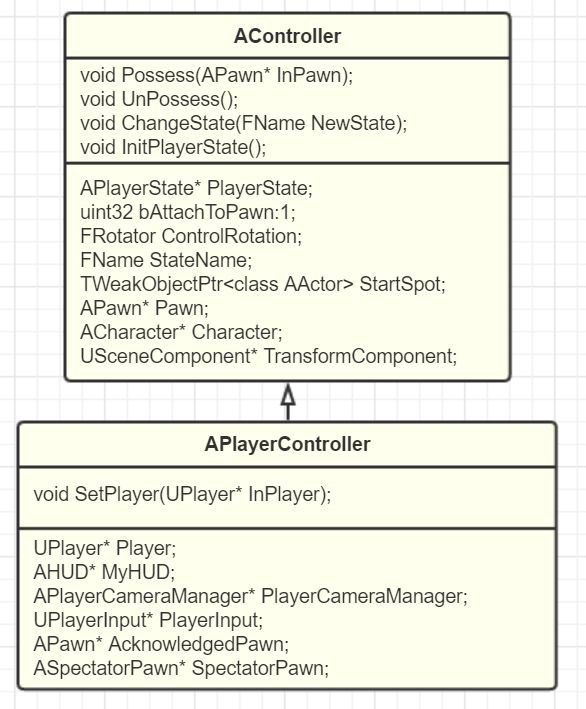
**一、world相关**

1. 多个level组成world；

2. World的信息存储在WorldContext中；

3. WorldContext又可被GameInstance访问，GameInstance始终存在；

**二、Controller**



1. Controller分为PlayerController与AIController，每个Controller中存储有PlayerState的指针，记录玩家信息；

2. **PlayerController实现的功能模块**

(1) Camera的管理，目的都是为了控制玩家的视角，所以有了PlayerCamera Manager这一个关联很紧密的摄像机管理类，用来方便的切换摄像机。PlayerController的ControlRotation、ViewTarget等也都是为了更新Camera的位置。因为跟Camera的关系紧密，而Camera最后输出的是屏幕坐标里的图像，所以为了方便一些拾取的HitResult函数也都是实现在这里面。

(2) Input系统，包括构建InputStack用来路由输入事件，也包括了自己对输入事件的处理。所以包含了UPlayerInput来委托处理。

(3) UPlayer关联，既然顾名思义是PlayerController，那自然要和Player对应起来，这也是PlayerController最核心的部分。一个UPlayer可以是本地的LocalPlayer，也可以是一个网络控制UNetConnection。PlayerController只有在SetPlayer之后，才可以开始正常工作。

(4) HUD显示，用于在当前控制器的摄像机面前一直显示一些UI。

(5) Level的切换，PlayerController作为网络里通道，在一起进行Level Travelling的时候，也都是先通过PlayerController来进行RPC调用，然后由PlayerController来转发到自己World中来实际进行。

(6) Voice，也是为了方便网络中语音聊天的一些控制函数。

3. **AIController实现的功能模块**

(1) Navigation，用于智能根据导航寻路，其中我们常用的MoveTo接口就是做这件事情的。而在移动的过程中，因为少了玩家控制的来转向，所以多了一个SetFocus来控制当前的Pawn视角朝向哪个位置

(2) AI组件，运行启动行为树，使用黑板数据，探索周围环境，以后如果有别的AI算法方法实现成组件，也应该在本组件内组合启动。

(3) Task系统，让AI去完成一些任务，也是实现GameplayAbilities系统的一个接口。目前简单来说GameplayAbilities是为Actor添加额外能力属性集合的一个模块，比如HP，MP等。其中的GamePlayEffect也是用来实现Buffer的工具。另外GamePlayTags也是用来给Actor添加标签标记来表明状态的一种机制。目前来说该两个模块似乎都是由Epic的Game Team在维护，所以完成度不是非常的高，用的时候也往往需要根据自己情况去重构调整。

**三、PlayerState**

(1)PlayerState随Level销毁而销毁；

(2)PlayerState中存储内容：

从应用范围上来说，PlayerState表示的是玩家的游玩数据，所以那些关卡内的其他游戏数据就不应该放进来（GameState是个好选择），另外Controller本身运行需要的临时数据也不应该归PlayerState管理。而玩家在切换关卡的时候，APlayerState也会被释放掉，所有PlayerState实际上表达的是当前关卡的玩家得分等数据。这样，那些跨关卡的统计数据等就也不应该放进PlayerState里了，应该放在外面的GameInstance，然后用SaveGame保存起来。

**四、GameMode**

**哪些逻辑应该写在GameMode里？哪些应该写在Level Blueprint里？**

* 概念上，Level是表示，World是逻辑，一个World如果有很多个Level拼在一起，那么也就是有了很多个LevelScriptActor，无法想象在那么多个地方写一个完整的游戏逻辑。所以GameMode应该专注于逻辑的实现，而LevelScriptActor应该专注于本Level的表示逻辑，比如改变Level内某些Actor的运动轨迹，或者某一个区域的重力，或者触发一段特效或动画。而GameMode应该专注于玩法，比如胜利条件，怪物刷新等。
* 组合上，同Controller应用到Pawn一样道理，因为GameMode是可以应用在不同的Level的，所以通用的玩法应该放在GameMode里。
* GameMode只在Server存在（单机游戏也是Server），对于已经连接上Server的Client来说，因为游戏的状态都是由Sever决定的，Client只是负责展示，所以Client上是没有GameMode的，但是有LevelScriptActor，所以GameMode里不要写Client特定相关的逻辑，比如操作UI等。但是LevelScriptActor还是有的，而且支持RPC，即使如此，LevelScriptActor还是应该只专注于表现，比如网络中触发一个特效火焰。至于UI，可以通过PlayerController的RPC，然后转发到GameInstance来操作。
* 跟下层的PlayerController比较，GameMode关心的是构建一个游戏本身的玩法，PlayerController关心的玩家的行为。这两个行为是独立正交可以自由组合的。所以想想哪些逻辑属于游戏，哪些属于玩家，就应该清楚写在哪里了。
* 跟上层的GameInstance比较，GameInstance关注的是更高层的不同World之间的逻辑，虽然有时候他也把手伸下来做些UI的管理工作，不过严谨来说，在UE里UI是独立于World的一个结构，所以也还算能理解。因此可以把不同GameMode之间协调的工作交给GameInstance，而GameMode只专注自己的玩法世界。

**五、GameState**

APlayerState用来保存玩家的游戏数据，那么同样的，对于一场游戏，也需要一个State来保存当前游戏的状态数据，比如任务数据等。跟APlayerState一样，GameState也选择从AInfo里继承，这样在网络环境里也可以Replicated到多个Client上面去。

