行为树

虚幻引擎高级程序开发专业

01

UE行为树概述

UE行为树

一套强大的AI操控系统！用户可以通过简单的编程，完成复杂的AI行为构建！我们在一般的游戏中，经常需要设计一些特殊角色，不限于简单的活动，更多的是和玩家做交互！在UE中我们可以使用行为树，来操作玩家外的其他Actor，让他们灵动起来，和我们一起在游戏世界中奔跑。UE中的行为树具有简单，易读的特性，并且操作繁琐度低，容易编写，区别于传统行为树。

标准行为树

以数据为驱动元，进行逻辑节点检查，寻找合理逻辑叶子节点，然后进行动作执行。对于一般的状态机结构AI，具有扩展方便，容易转接的特点！被广泛使用在各种AI设计中，是一种优良的程序设计方案！可以解决复杂AI需求，使用起来方便，支持横向扩展节点和向下延伸。

02

UE行为树特点

事件驱动型（UE）

我们知道，一般设计中对于驱动元选择，优先选择被动通知的方式。在传统的行为树结构中经常采用遍历检查条件节点，效率低！在UE中，行为树避免了在高速运行的引擎中去检查逻辑节点，采用了事件通知的方式，没有无用的迭代操作，只有执行位置或黑板数值**发生变化**了，才会产生影响。

叶子节点不是条件语句（UE）

在行为树标准模型中，条件语句即为叶子节点，除去成功和失败不执行任何操作。UE中引入了Decorator系统作为条件语句。D语句在UI结构中易读，并且罗列条件，可以直观查看条件状态，并且所有叶子节点（相对于D）均为任务节点，清晰看到执行任务的位置，在传统模型中，**条件语句混于叶节点中**，因此需要花更多时间分辨哪些叶节点是条件语句，哪些叶节点是行动。

并发行为处理（UE）

标准行为树通常使用 Parallel composite 节点来处理并发行为。Parallel 节点**同时执行其所有子项**。在一个或多个子项树完成时，特殊规则将决定如何执行（取决于所需行为）

UE4 行为树**抛弃**了复杂 Parallel 节点，使用 Simple Parallel 节点和称为 Services 的特殊节点，以实现同类行为。

抛弃Parallel节点的原因

1. 容易让人迷惑，在多平行行为中容易迷失，无法追寻逻辑方向
2. Parallel不利于性能优化，无法方便构建事件驱动型集合

Simple Parallel节点

1. 只允许有两个选项，一个必为单独任务节点（含可选 decorators）、另一个为一个完整的分支树
2. 支持Parallel节点的多数用法
3. 利用 Simple Parallel 节点简便地进行一些事件驱动型优化

UE4并发行为树的优点

1. 清晰明了：使用 Services 和 Simple Parallel 节点可创建出易于理解的简单行为树
2. 易于纠错：图表更清晰，便于纠错。除此之外，更少的同时执行路径十分便于观察图表中实际发生的状况
3. 优化简单：如没有较多同时执行的分支树，事件驱动型图表将更易于优化

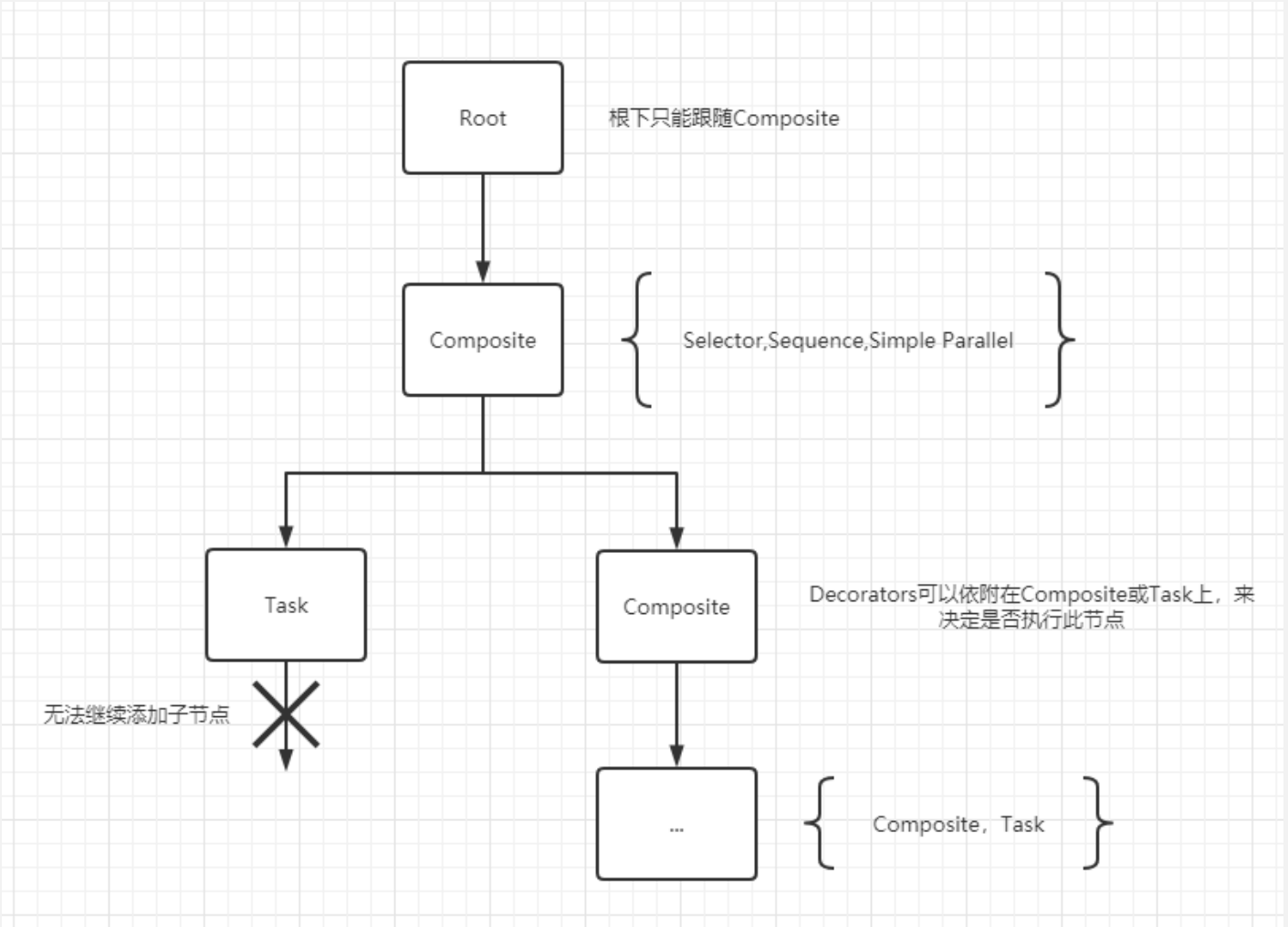
03

UE行为树

行为树五类节点

1. Composite，这种节点定义一个分支的根以及该分支如何被执行的基本规则
2. Task，这种节点是行为树的叶子，实际“执行”操作，不含输出连接
3. Decorator，即为条件语句。这种节点附着于其他节点，决定着树中的一个分支，甚至单个节点是否能被执行
4. Services，这种节点附着在 Composite 或Task节点上，只要其分支节点被执行，它们便将按所定义的频率执行。它们常用于检查和更新黑板。
5. Root，根节点，无法被附着，可以用来设置黑板数据

结构


Root
根下只能跟随Composite
Composite
Selector,Sequence,Simple Parallel
Task
Composite
Decorators可以依附在Composite或Task上，来决定是否执行此节点
无法继续添加子节点
...
Composite，Task

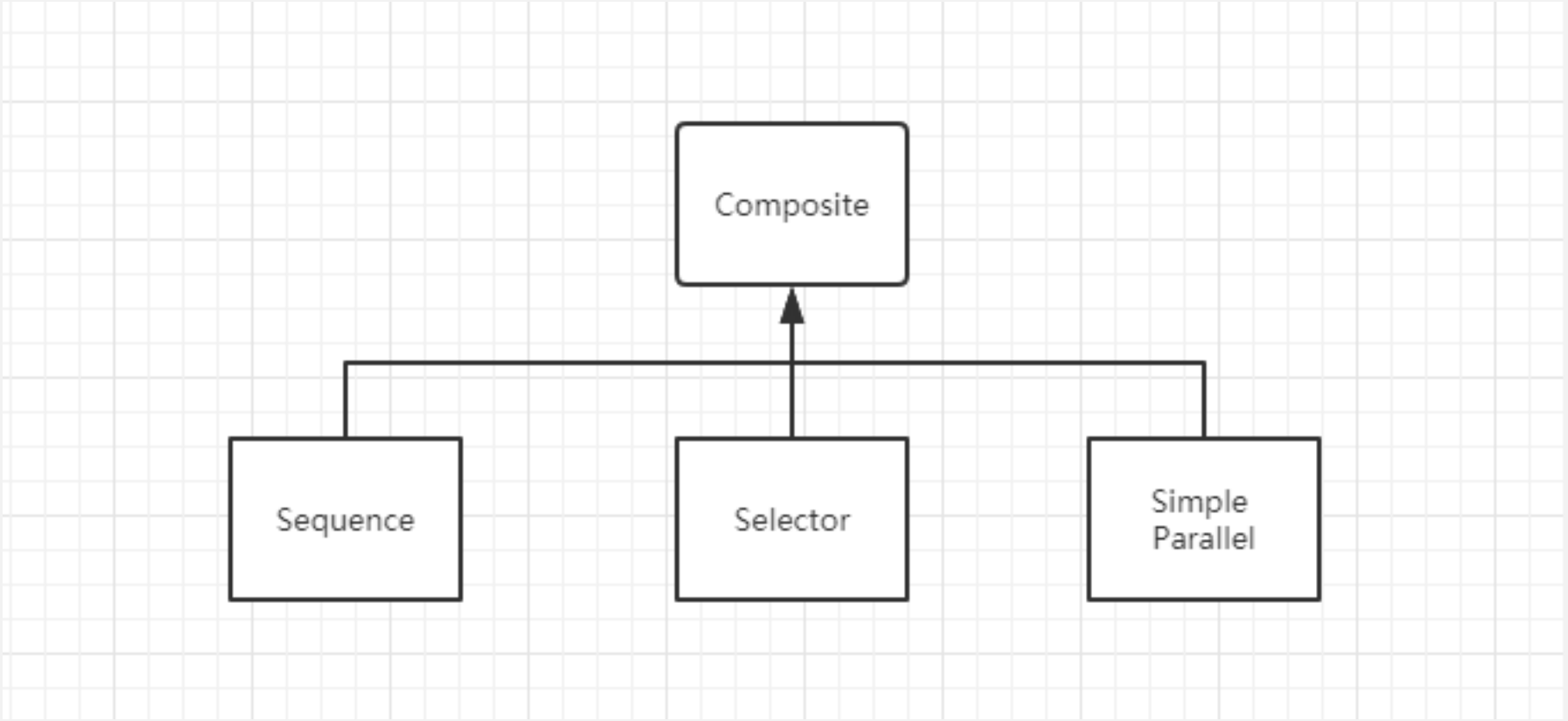
Blackboard黑板

1. 行为树中的核心模块
2. 用于支撑行为树中的数据存储和读取
3. 相当于我们的草稿纸，我们可以把需要参与逻辑的数据罗列在上面
4. 使用方便，数据扩展性强
5. 有较好的维护性

Composite

1. Root下只能跟随composite
2. 定义一个分支的根起点，以及该分支如何被执行的基本规则
3. 包含Selector，Sequence，Simple Parallel
4. 构成行为树的最重要节点，所有逻辑结构设计尽量遵循Composite规则
5. 节点内容简单，但是附加逻辑项多，逻辑点繁琐

Composite



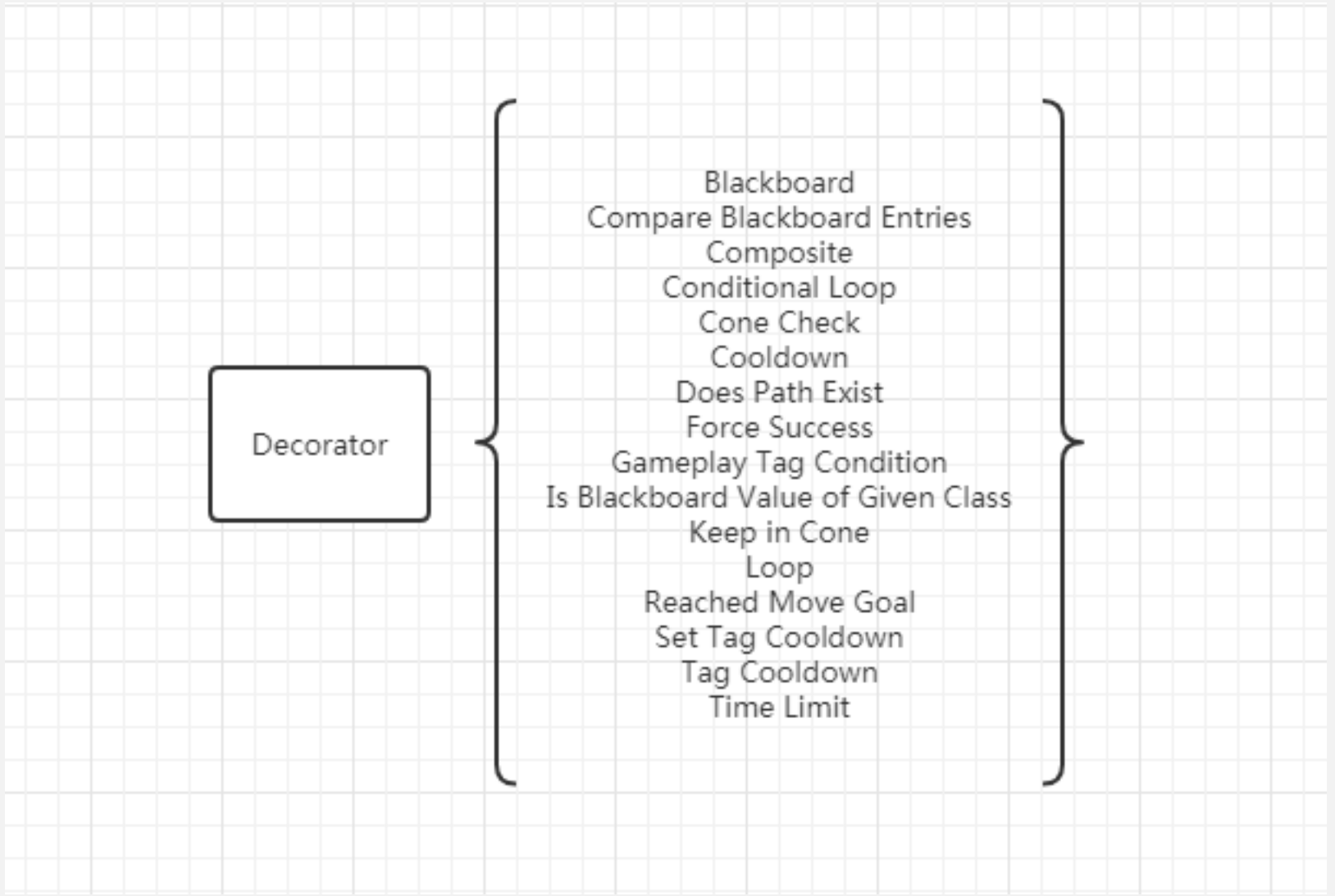
Composite

1. Selector，从左至右执行，只要有一个子项条件成立，则停止向下执行。任意子项成立，则selector成立，所有子项不成立，则selector不成立。
2. Sequence，从左至右执行，只要有一个子项不成立，则停止执行剩余项。所有子项成立，则sequence成立，有一子项不成立，则sequence不成立。
3. Simple Parallel，并行树，允许单个任务在任务树旁执行。Finish Mode 中的设置将确定节点是否立即完成、是否终止次要树，或是否延迟次要树的完成

Decorator

装饰器，用来决定节点是否可以进行执行的关键判定工具。这是一个条件检查工具，可以附着到任务节点和复合节点。用来约束节点或是分支是否可以执行。

装饰器


Decorator
Blackboard
Compare Blackboard Entries
Composite
Conditional Loop
Cone Check
Cooldown
Does Path Exist
Force Success
Gameplay Tag Condition
Is Blackboard Value of Given Class
Keep in Cone
Loop
Reached Move Goal
Set Tag Cooldown
Tag Cooldown
Time Limit

Decorator装饰器

1. Blackboard，借助一个黑板数据，进行逻辑操作判定
2. Compare Blackboard Entries，对比黑板内两个数据
3. Cone Check，锥形检测，可以检测目标是否在锥形范围内
4. Cooldown，锁定节点或是分支，直到时间结束
5. Does Path Exist，检查AB两点见路径是否可行
6. Force Success decorator 将节点结果改为成功
7. Is At Location，检查AI控制的角色是否在指定的位置
8. Keep in Cone，持续检查锥形范围内目标
9. Loop，以一定次数或是无限次数循环节点
10. TimeLimit，节点逻辑时间定时器

观察器终止逻辑

1. NONE，不终止执行
2. Self，中止 self，以及在此节点下运行的所有子树
3. Lower Priority，中止此节点右方的所有节点
4. Both，终止 self、此节点下运行的所有子树、以及此节点右方的所有节点

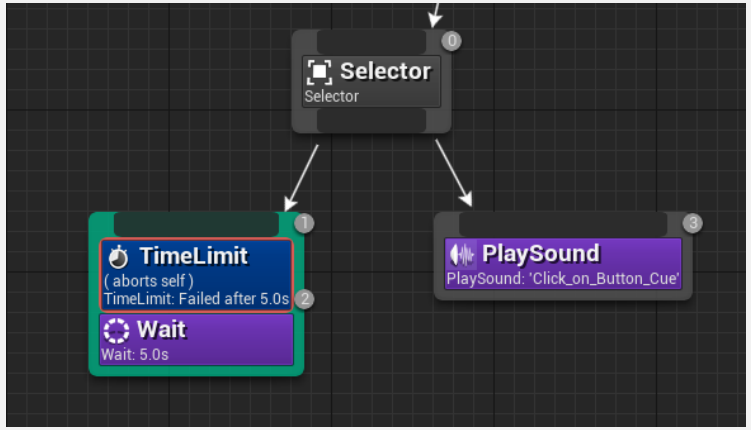
以上逻辑操作，适用于装饰器（**当父节点是Selector时则能看到所有终止选项，当父节点是Sequence时，只存在None和Self选项**）

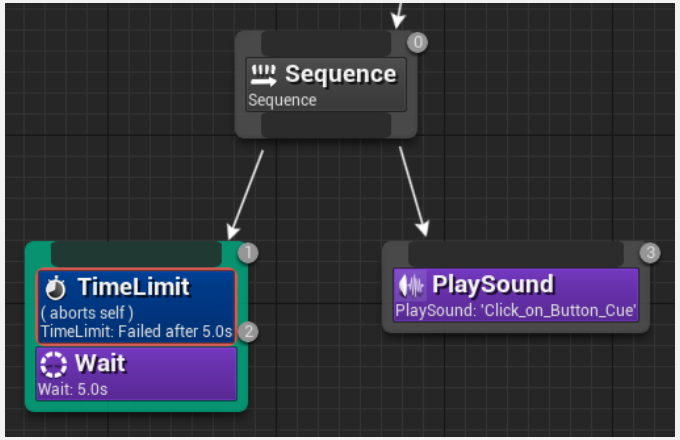
观察器终止逻辑

在使用观察器终止时需要注意，当装饰器所在节点**的父节点**是Sequence并且终止方式为self。当终止成立时，并且节点任务进行中，则返回父节点，并告知执行被意外终止，Sequence则停止向下执行，向上返回（Sequence节点只要有一子节点执行失败，则意味Sequence执行失败）。

如父节点是Selector时，终止条件成立，则停止执行此节点，执行节点右侧节点，参照下图（Selector节点仅当所有子节点失败，Selector节点失败。）。

图一，当TimeLimit终止成立则执行播放声音。图二当TimeLimit终止成立则Sequence节点执行失败，不播放声音，并且向上层节点返回失败。





Service服务

1. 附着在Composite节点上或Task节点
2. 只要节点内执行，则Service会运行
3. 可以按照指定的频率执行
4. 主要意图在于检查和更新黑板内容
5. 它们以行为树系统的形态取代了传统平行节点
6. 我们一般会重写此节点

Task任务节点

1. 真正在干活的角色
2. 可以重写，支持扩展
3. UE提供了很多基本的任务节点

Task任务节点

1. Make Noise，制造噪音，需要Pawn带PawnNoiseEmitter组件，其他Pawn如果带有PawnSensing组件，则可以听到噪音
2. Move To移动到目标点，通过黑板提供
3. Play Sound 播放声音
4. Run Behavior 执行其他行为树
5. Wait，Wait BB Time，等待事件，无法被终止
6. Play Anim，促使控制Pawn播放动画

感谢观看