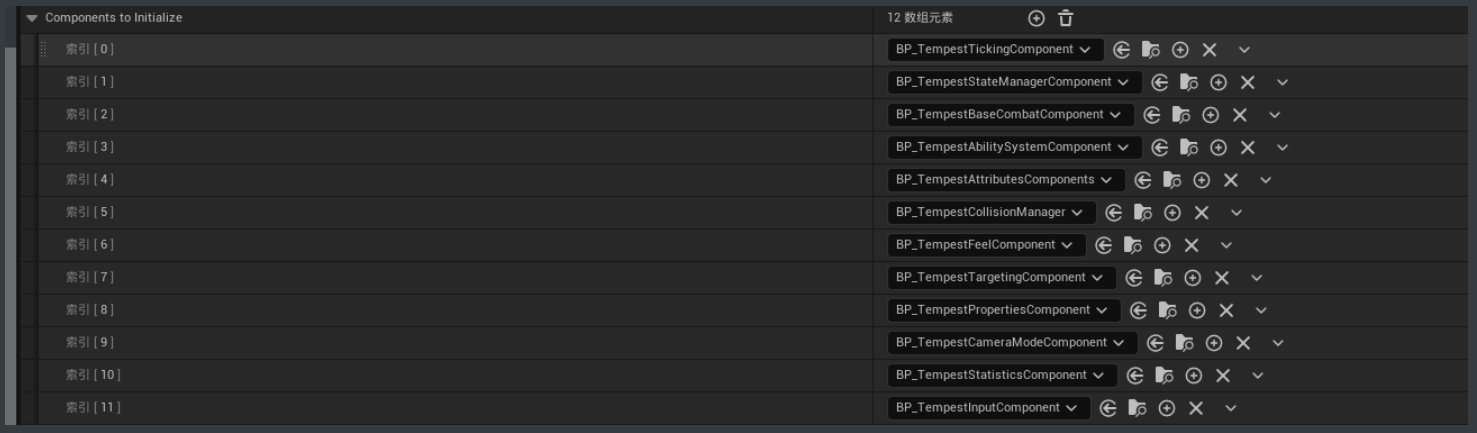
数据资产配置--组件

配置了所需要的各种功能组件，

目前逻辑是在角色捡到武器，会遍历这个资产的所有组件配置把他添加到角色上



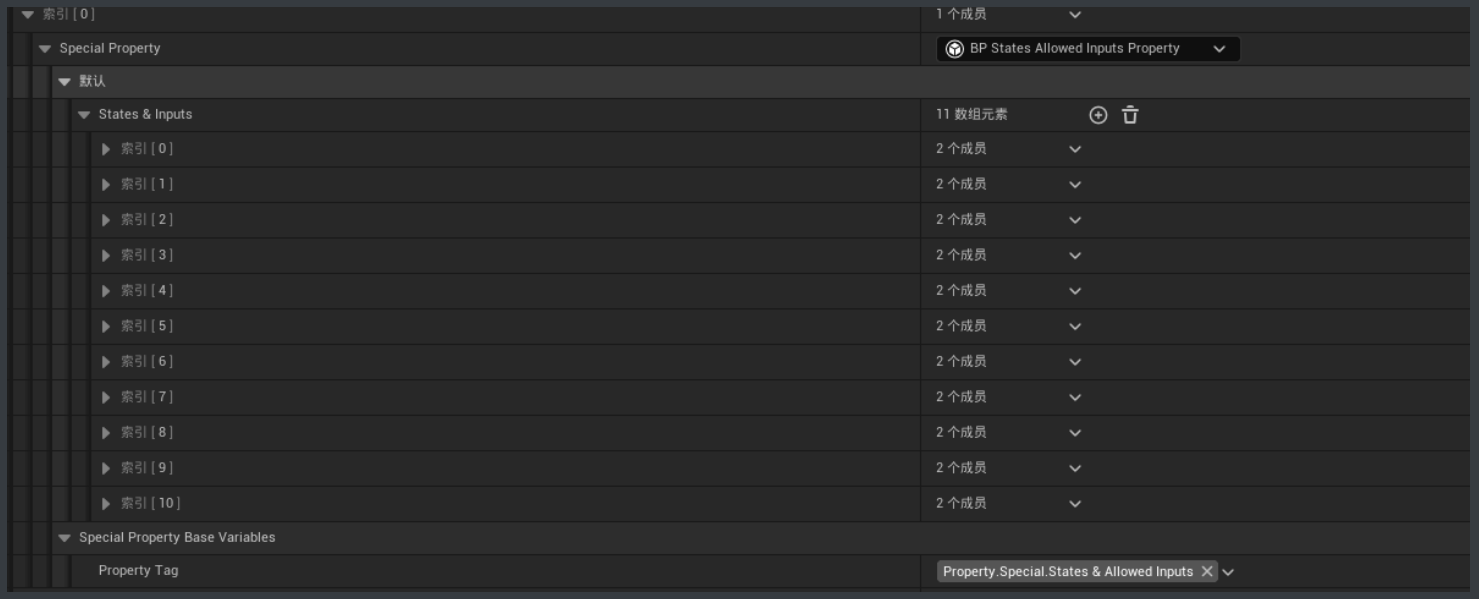
BP_TempestStateManagerComponent

主要的逻辑都放在C++基类中进行实现，主要目的就是为了切换不同的状态，每个状态的作用见状态和能力属性

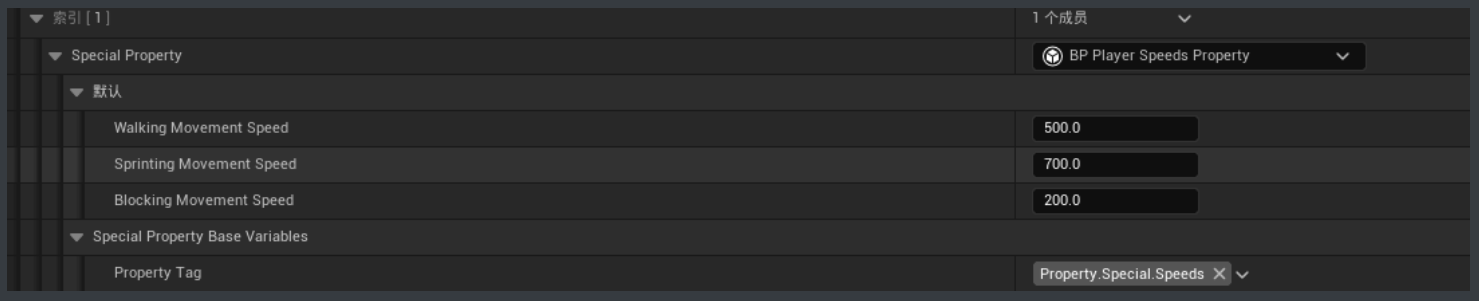
数据资产配置--通用属性（GeneralProperty）

允许的状态输入属性

通过输入来驱动状态和能力

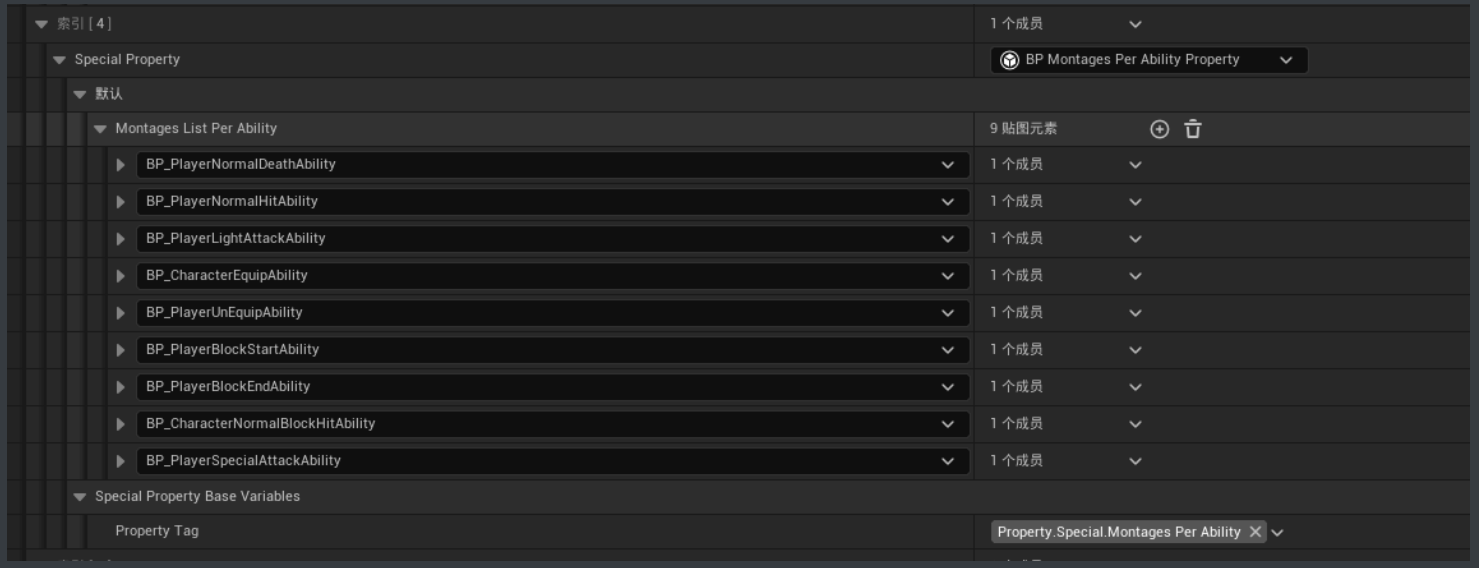


玩家速度属性



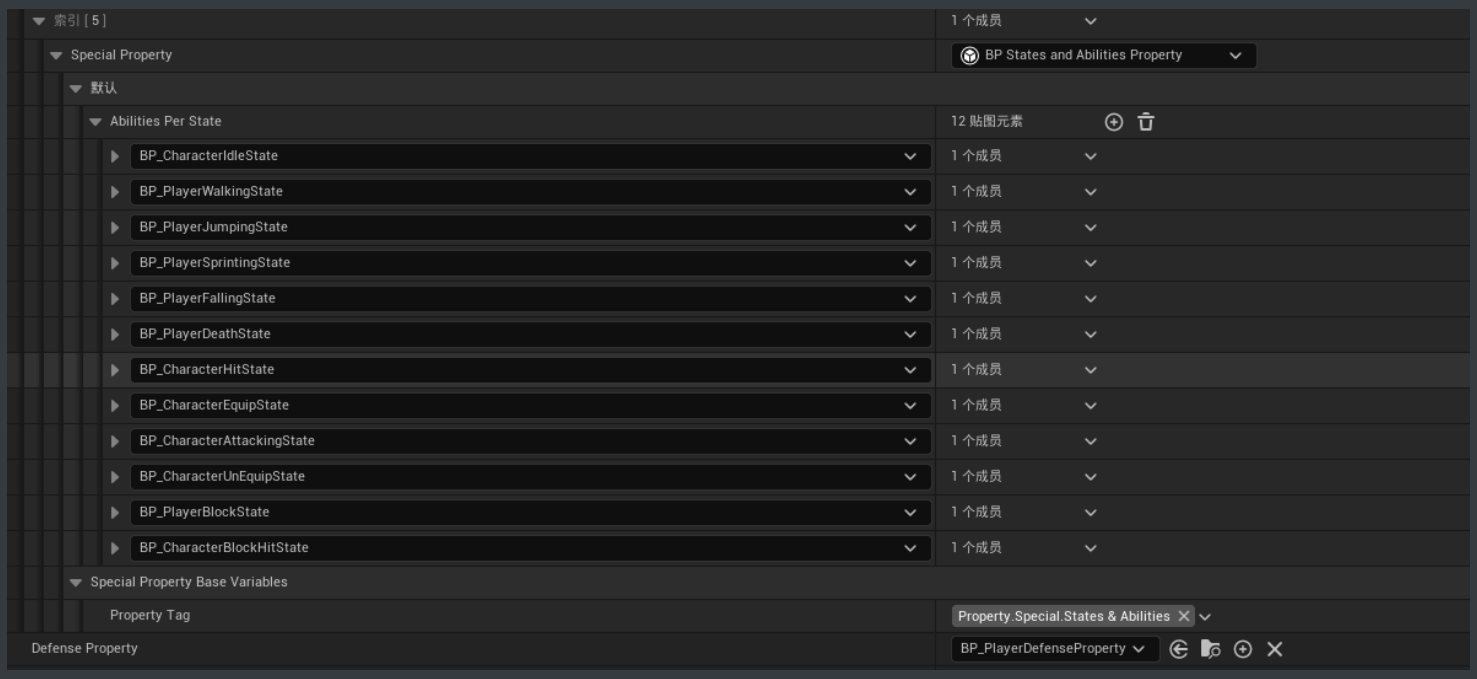
特殊能力的蒙太奇配置

因为走的重定向逻辑，像基础的走跑跳就不用在配置了，但有些特殊动画比如死亡攻击之类的就需要特殊的蒙太奇，而且可能每个武器的这些动作都不一样都需要配置



状态和能力属性

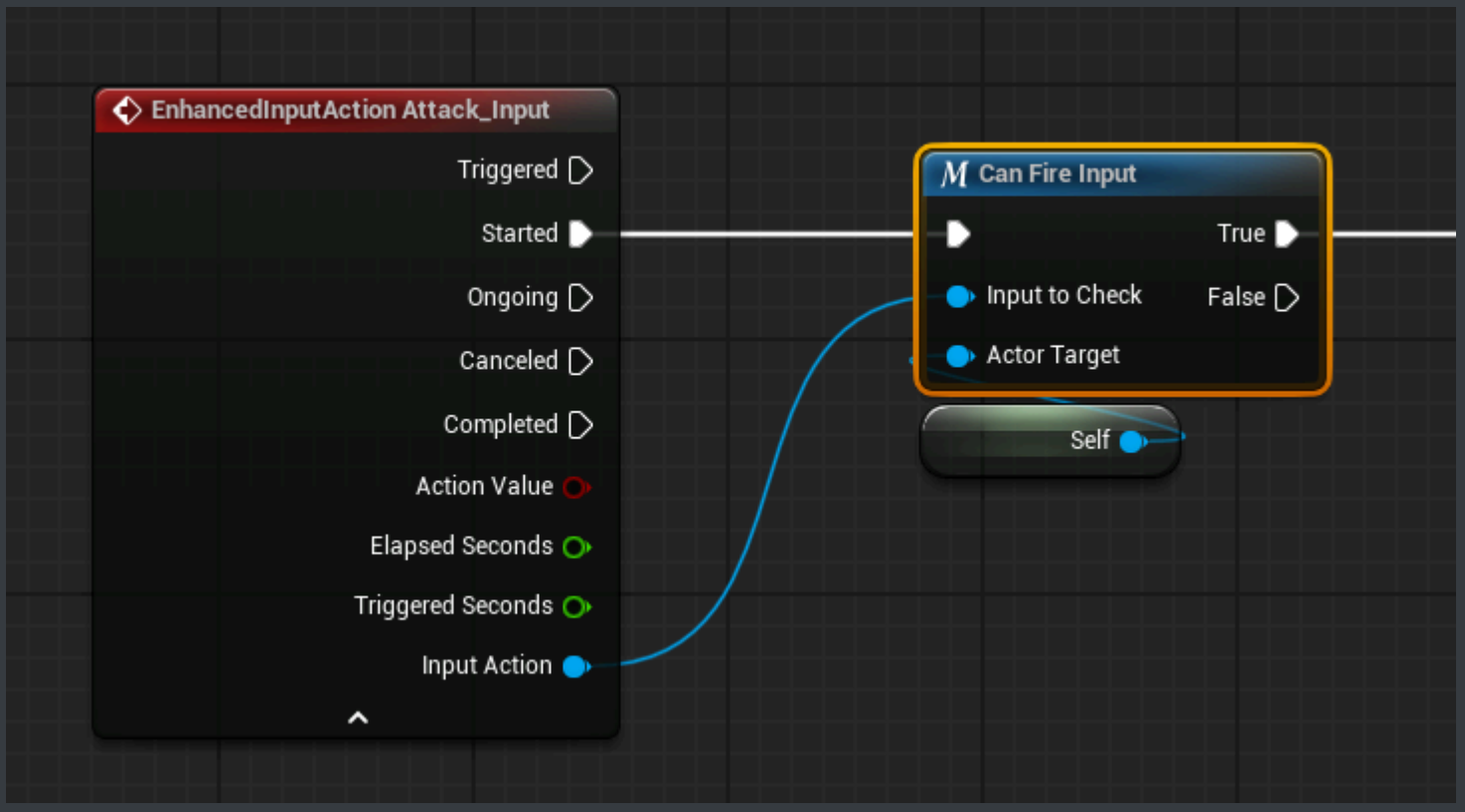
每一个这个武器或者角色可能拥有的状态都配置出来



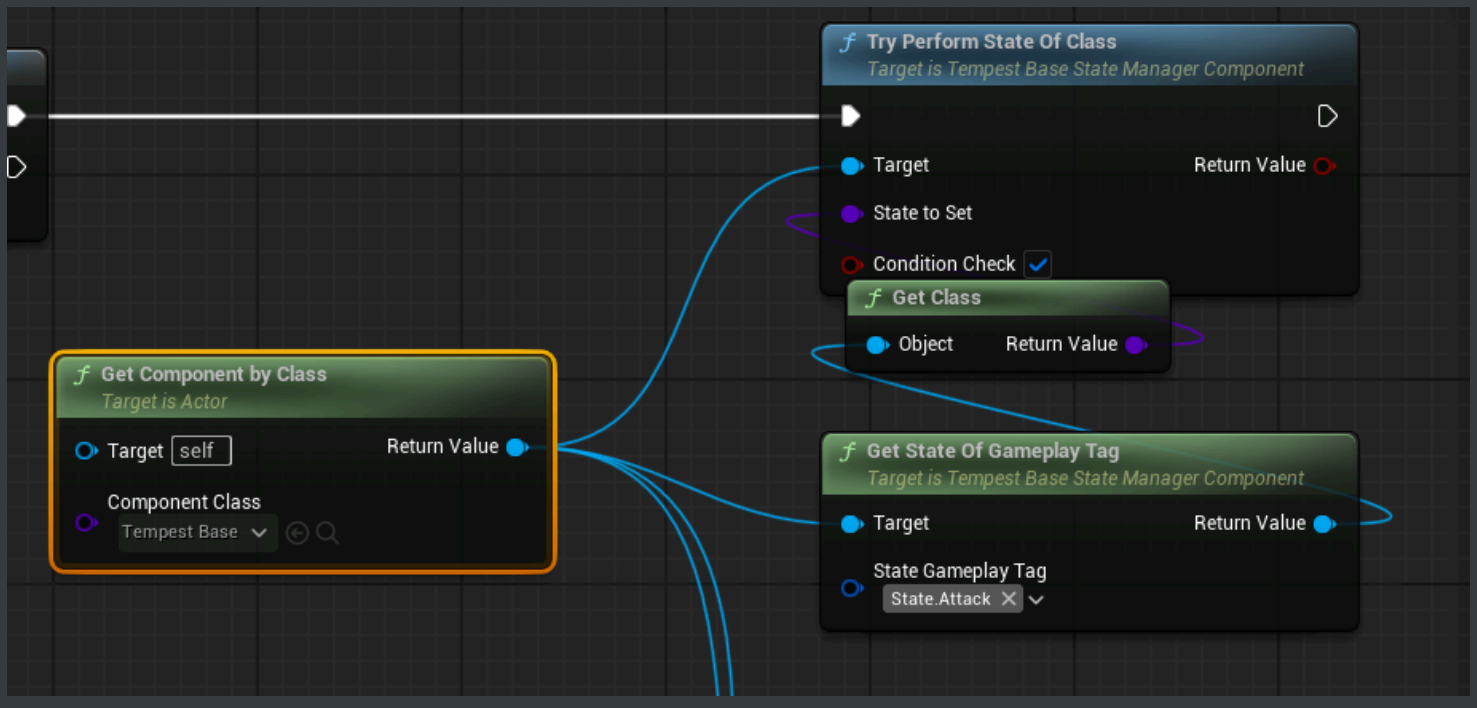
一个攻击的流程示意(将这几个组件串起来):

输入驱动状态,状态调用能力,一个状态可以拥有多种能力

在玩家的基类BP_ThirdPersonCharacterBasic中调用攻击,具体能不能广播事件见UTempestBaseInputComponent脚本,



之后将状态设置为攻击状态



这里解释一下切换到攻击状态的原理(后续会移动到UTempestBaseStateManagerComponent这个组件的讲解):

每个状态类的基类UTempestBaseStateManagerComponent

有个构造方法

使用 `NewObject` 动态创建一个新的状态对象实例

将新创建的状态对象添加到可激活状态列表(`ActivatableStates`)中

设置状态对象的执行者为当前组件的拥有者

这样状态对象知道是哪个Actor在执行它

```
void UTempestBaseStateManagerComponent::ConstructStateOfClass(TSubclassOf<UTempestBaseStateObject> Sta
{
    ConstructedState = nullptr;
    if (StateToConstruct)
    {
        UTempestBaseStateObject* LocalNewState;
        LocalNewState = NewObject<UTempestBaseStateObject>( Outer: GetOwner(), StateToConstruct);

        ActivatableStates.AddUnique(LocalNewState);
        LocalNewState->SetPerformingActor(GetOwner());

        LocalNewState->ConstructState();
        ConstructedState = LocalNewState;
    }
}
```

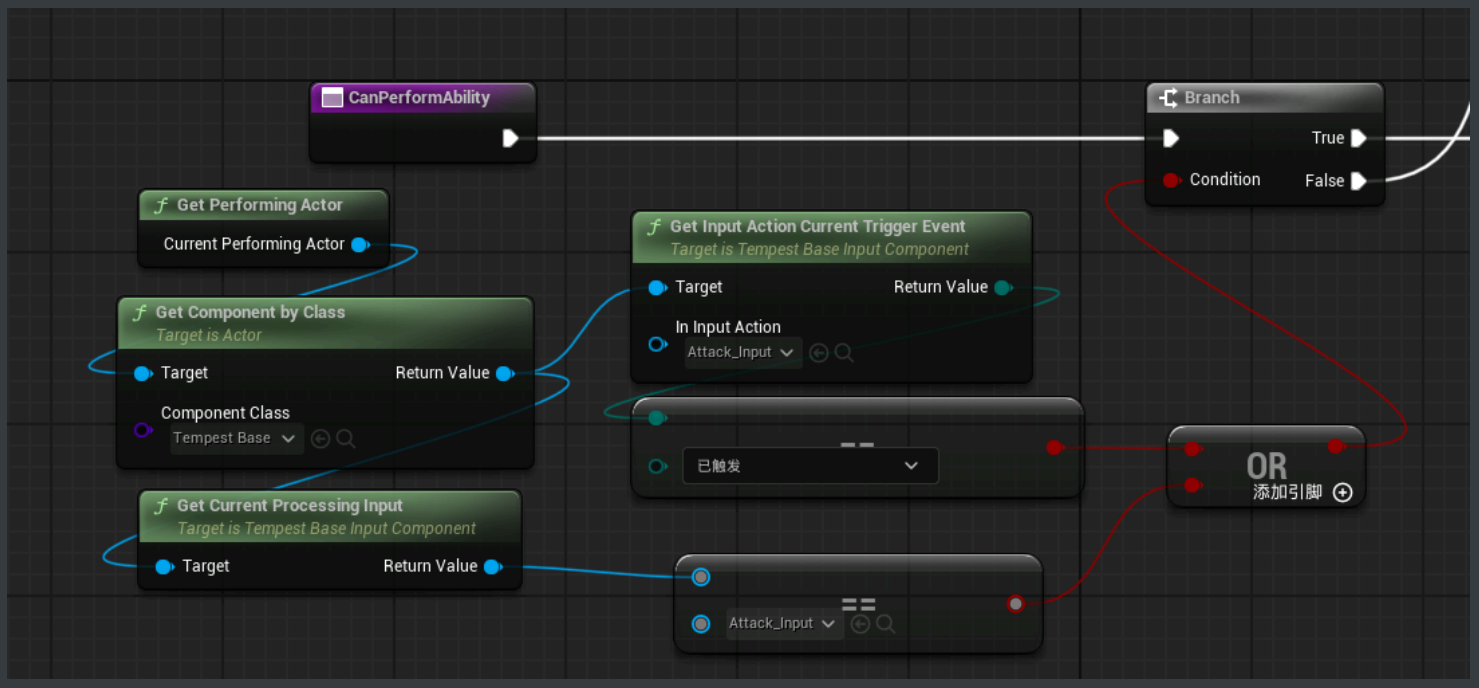
然后再TryPerformStateOfClass尝试执行新的状态的时候需要判断一下条件

```
bool UTempestBaseStateManagerComponent::TryPerformStateOfClass(TSubclassOf<UTempestBaseStateObject> StateToSet, bool
{
    if (StateToSet)
    {
        UTempestBaseStateObject* LocalState = nullptr;
        GetStateOfClass(StateToSet, [&]LocalState);

        if (LocalState)
        {
            if (ConditionCheck)
            {
                if (LocalState->CanPerformState())
                {
                    LocalState->PreStateActivation();
                    LocalState->StartState();
                    LocalState->PostStateActivation();
                    return true;
                }
            }
```

比如这个状态被初始化构造了,以及切换这个状态是否需要判断是否可以转化:判断是否可以转化的逻辑在BP_PlayerLightAttackAbility蓝图中编写,

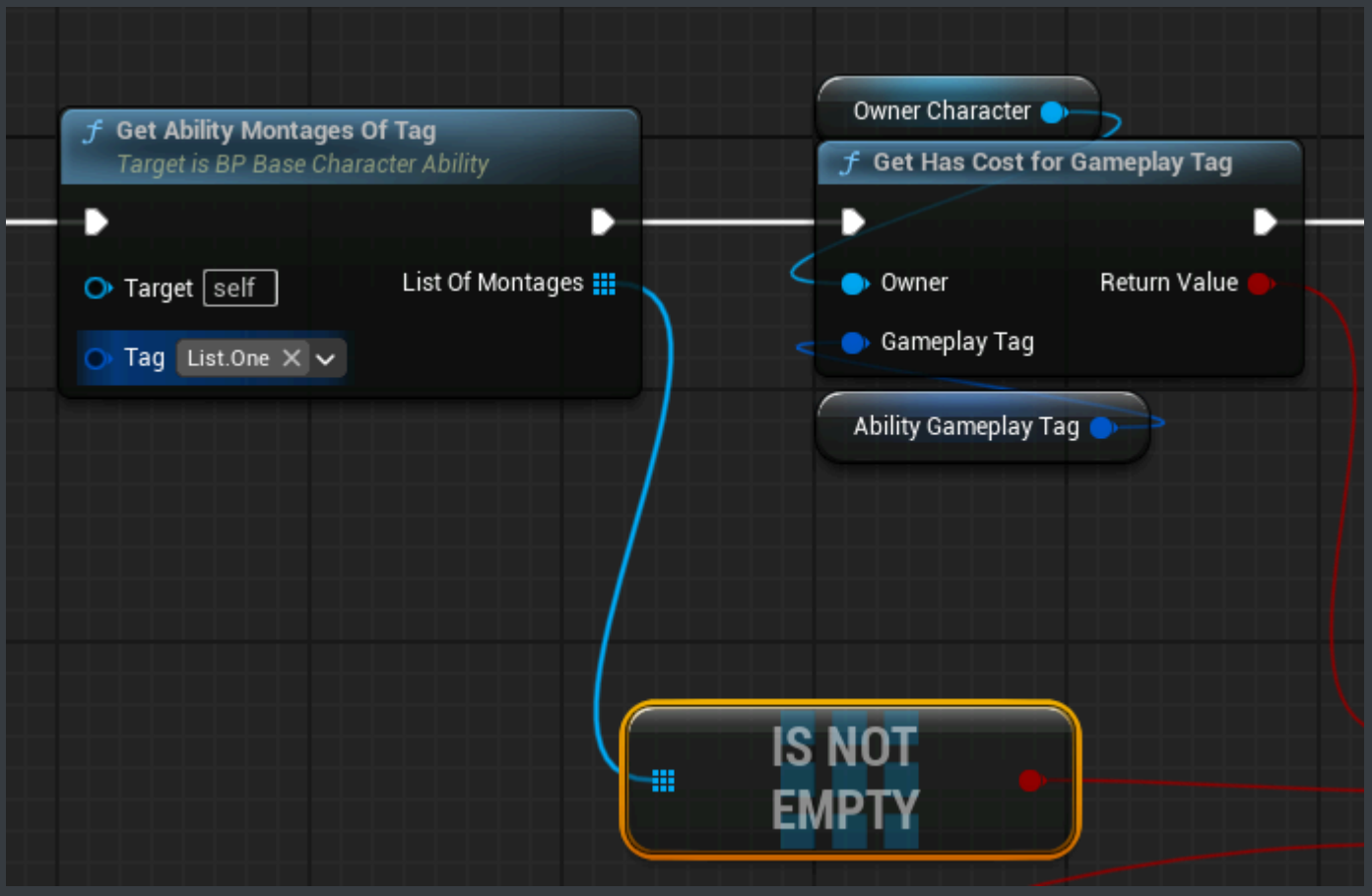
首先判断条件,是否触发了攻击按键



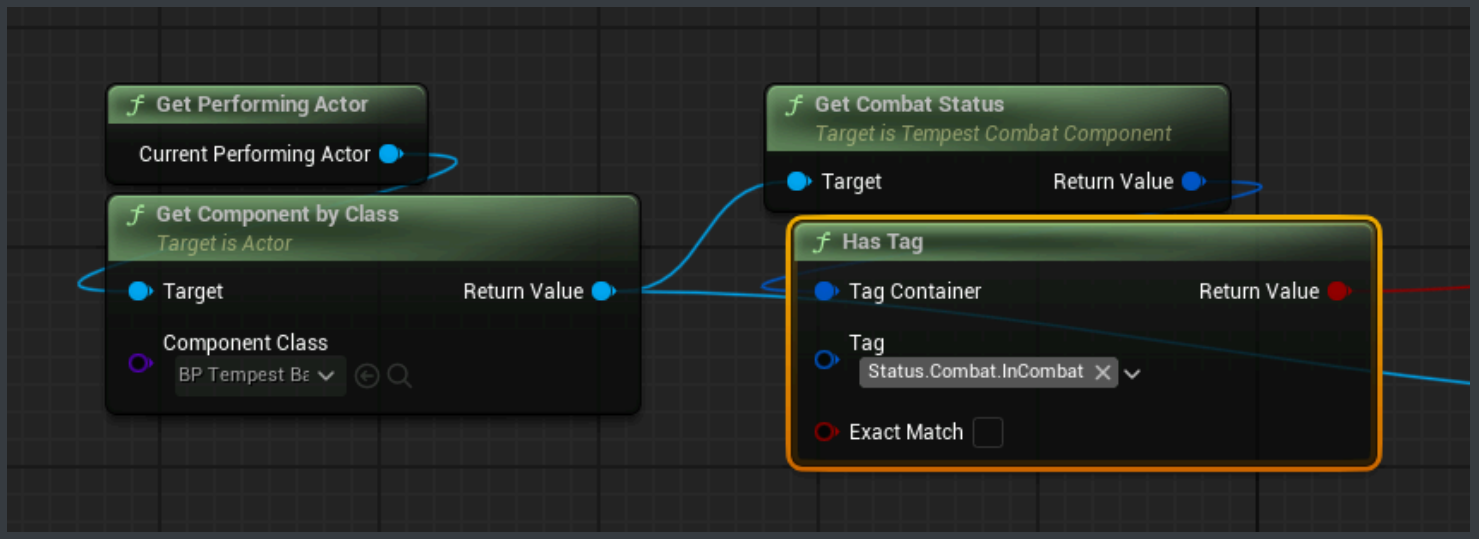
之后需要判断

1.获取攻击动画,然后判断动画是否为空

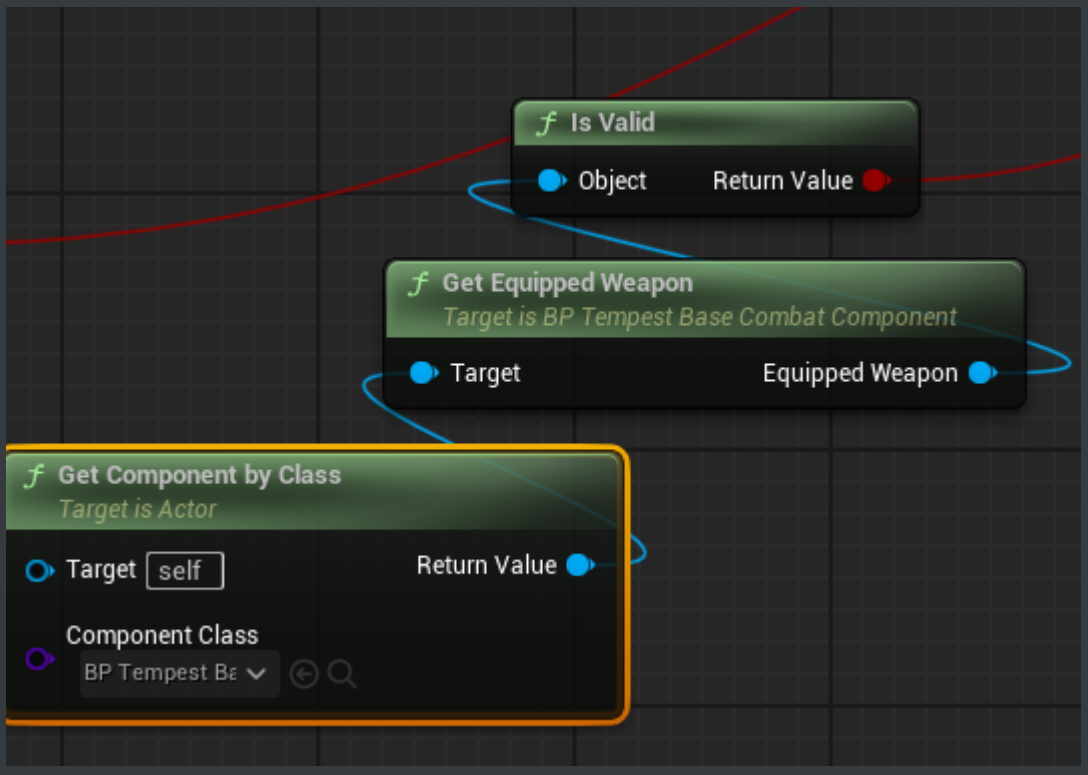
2/然后判断是否有足够的消耗值



3.判断是否处于战斗状态



4.是否持有武器



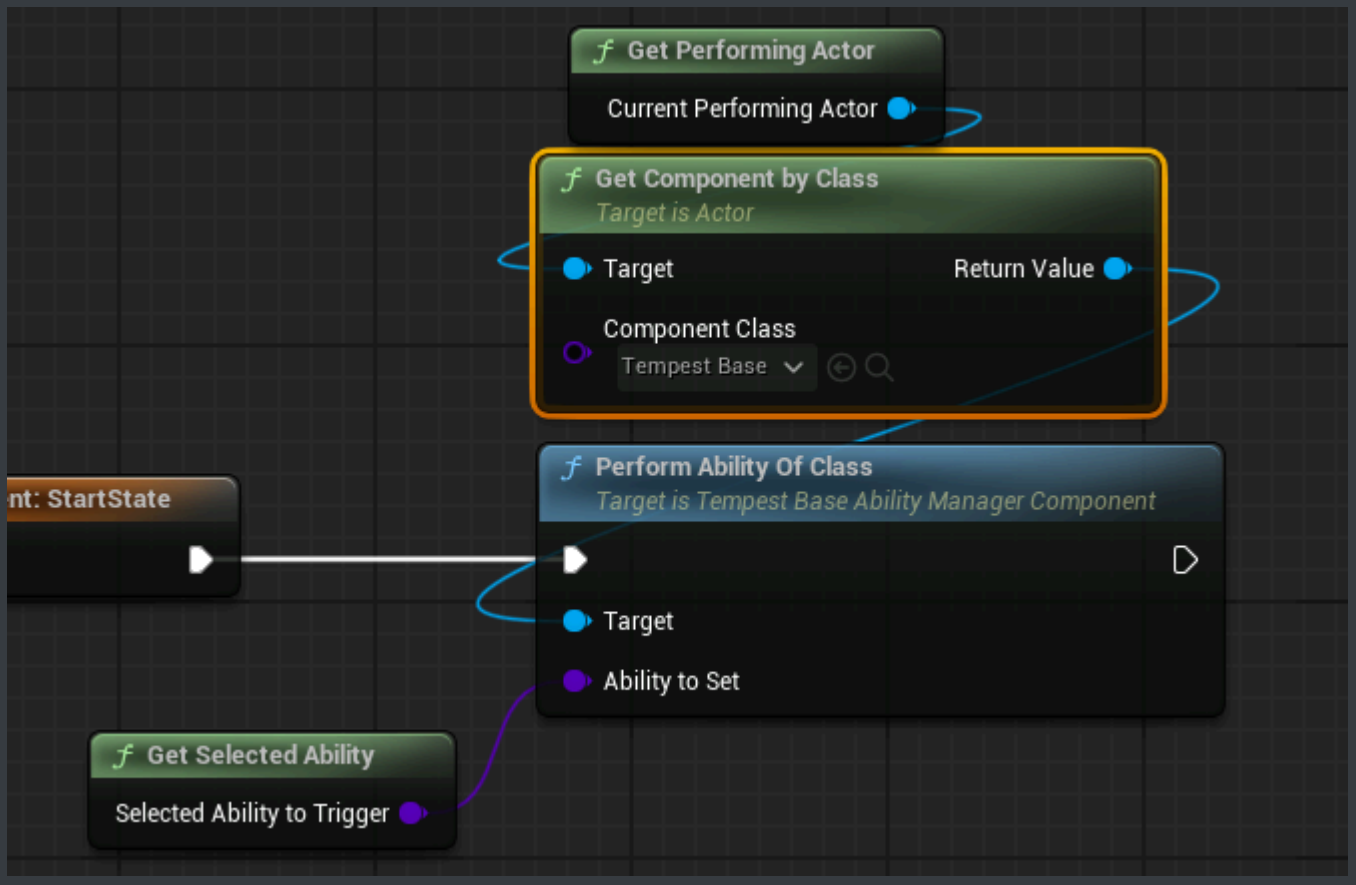
然后这四个条件都通过则可以成功通过判断条件来执行转换状态的逻辑:

```
LocalState->PreStateActivation();
LocalState->StartState();
LocalState->PostStateActivation();
return true;
```

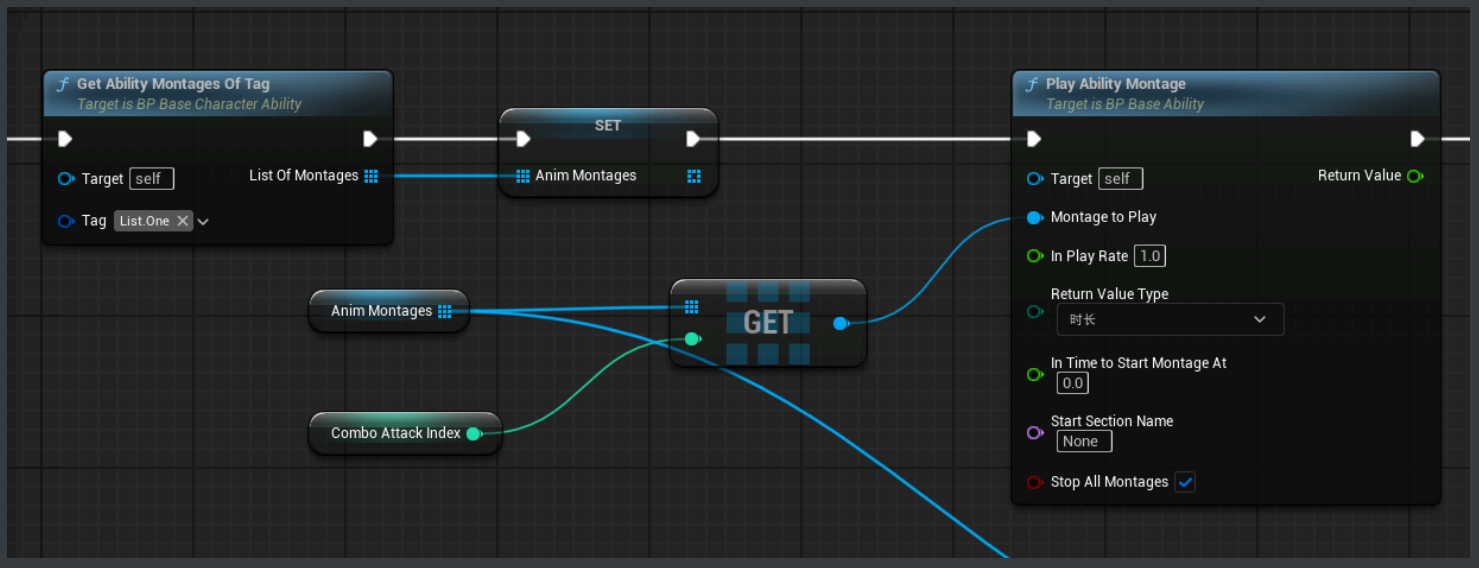
执行该状态激活前状态 -- 开始状态 --状态激活后状态

攻击只在自己的类写了开始状态:

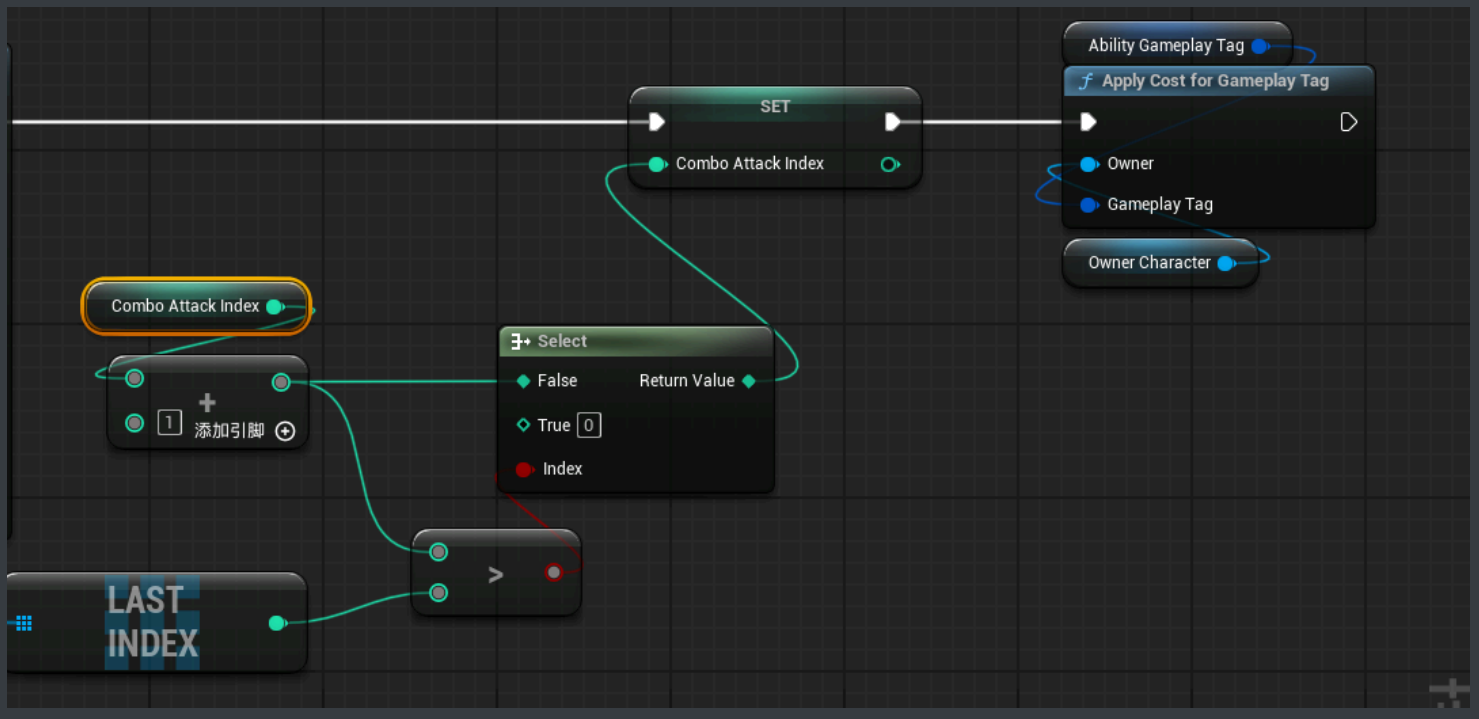
调用自己在数据资产里面的攻击能力BP_PlayerLightAttackAbility,能力组件和状态组件类似



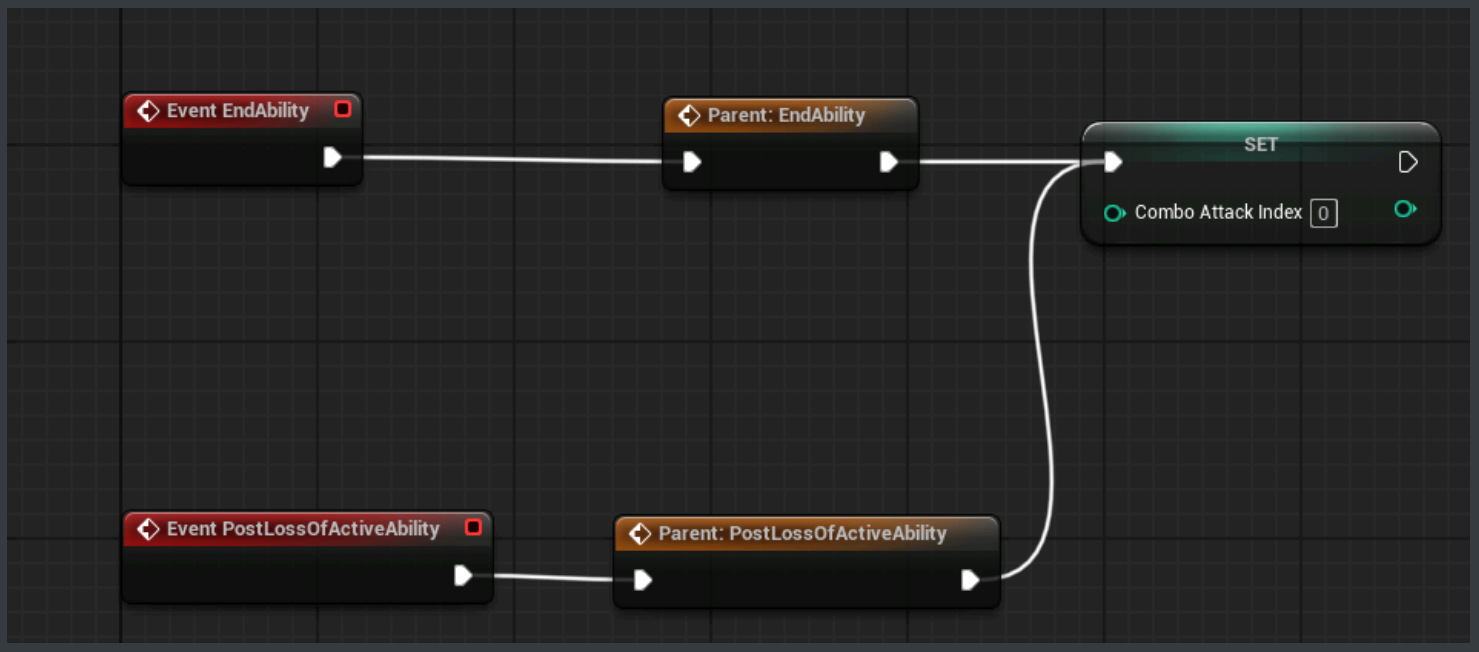
这个能力先获取要攻击的蒙太奇动画并播放



之后增加攻击索引以及计算攻击消耗



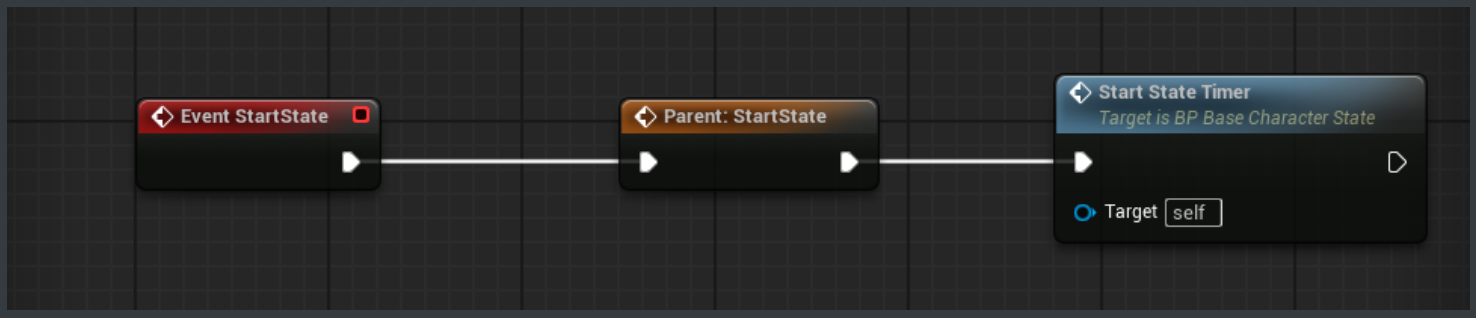
在攻击能力结束时重置索引

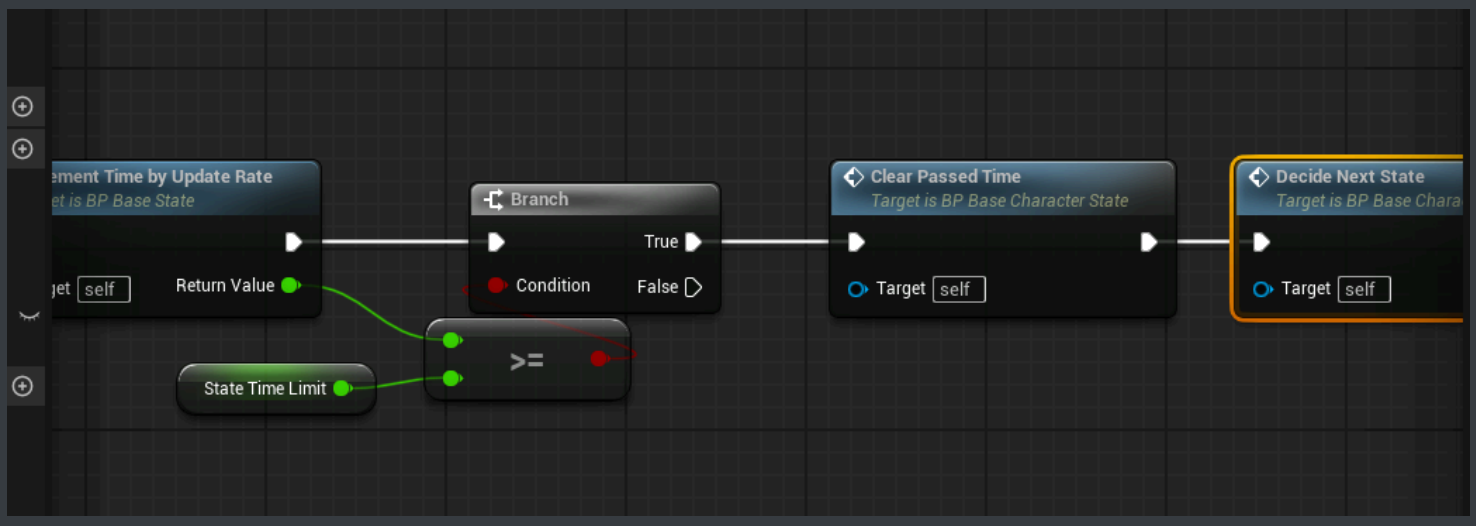


接下来只有最后一个问题,如何结束这个状态,在父类中

在开始状态是会启用一个计时,在时间超过StateTimeLimit之后就会调用状态的EndState方法

但目前StateTimeLimit是0秒也就是每次执行完就结束状态


Event StartState
Parent: StartState
Start State Timer
Target is BP Base Character State
Target
self


ement Time by Update Rate
et is BP Base State
get
self
Return Value
Branch
True
Condition
False
Clear Passed Time
Target is BP Base Character State
Target
self
Decide Next State
Target is BP Base Chara
Target
self
>=
State Time Limit