插件：

GAS Attach Editor：实时查看角色身上的属性，用于debug。

GAS Compaion：GAS辅助功能，可以尽可能少的写代码。

GA

GA的激活

怎么向GA传递数据

GA的冷却和Cost

几个数据

## ASC

UAbilitySystemComponent，谁释放技能，who

作为核心，管理协调其他组件

拥有ASC的Actor才拥有释放技能的能力

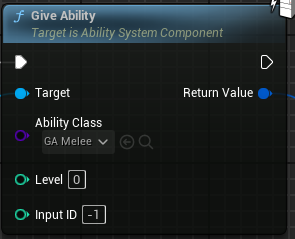
ASC放在Pawn还是PlayerState上，这是一个问题

## GA

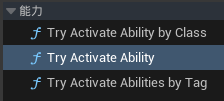
UGameplayAbility,技能逻辑，技能的逻辑，how

某种程度上，GA可以表示技能本身。也就是说，GA就是技能本身。

我们在创建好GA之后，就可以通过ASC的GiveAbility函数来给角色提供技能了。（这里将技能理解为一种特殊的资产即可）



具体怎么使用这个技能呢？通过增强输入的事件节点，调用ASC的TryActivateAbility函数即可。注意这个函数有三种重载



需要关注这三个函数的参数



By none函数的参数Ability to Activate参数，是ASC的GiveAbility函数的返回值。一般建议在GiveAbility之后将返回值直接提升为变量，将这个变量赋值给该函数即可。而且这个返回值本身也可以代表技能本身。

By Class函数的参数，是比较好赋值的，直接选择那个GA的类就可以。但是会要求必须实现了这个GA的GiveAbility。两者相比之下推荐By None的，因为创建变量之后方便技能的管理，就是说可以删除技能替换技能。

By Tag函数的参数，是最不好赋值的，但是也是使用最灵活的。我们后面统一使用by Tag。为了理解这个函数的参数，我们可以将这个函数的这个形参提升为变量。在它的默认值中，可以选择一个标签，这个标签就是我们将要释放的技能。所以我们需要回到GA中。在GA的类默认值细节中，有“标签--能力标签”，这里我们给能力标签一个赋值，就可以通过这个标签来代表这个能力本身。



此时再回到角色蓝图，就可以给刚提升为变量的这个Tag参数进行赋值。



这样就可以通过Tag来ActivateAbility

Tag方法也是非常灵活的一个方式。比如如果有人的被动是受到伤害后触发反击，就可以通过Tag加上对应的标志来达到相应的效果。

ASC还有一种方式来提供能力：GiveAbilityAndActivateOnce函数。这个函数提供的技能只会给一下，一般用于一个临时技能。

另外，GA的PlayMontage函数用的是PlayMontageAndWait节点。这个节点不需要提供Mesh参数，因为会自动匹配，如果匹配不上会自动失败。

在GA中，可以通过GetActorInfo函数来获取到GA的拥有者信息。可以通过这个方式来在GA中得到技能拥有者角色的对象信息。

因为GA中的东西是要去在一帧内实现的，所以是不能去做Timeline这种东西的。如果对于一些技能不是立即完成的，比如我们技能里会有体现角色动作、特效、位移等。这种情况下我们用Task。在Ability中提供了大量的Task，节点的搜索关键字是Wait。比如，PlayMontageAndWait。

技能结束之后要调用EndAbility，这样才能触发EventOnEndAbility。注意，EndAbility应该在众多WaitTask完成之后调用。

## Attribute & AttributeSet：AS

FGameplayAttribute，属性。多个Attribute组合成AttributeSet，挂在Actor上

目前只能在C++中实现。父类为AttributeSet。

要注意的是，Attribute自己的BaseValue和CurrentValue并不等同于MaxHealth和CurrentHealth。MaxHealth也是单独的一个Attribute。

#include "AbilitySystemComponent.h"

#define ATTRIBUTE\_ACCESSORS(ClassName, PropertyName) \

GAMEPLAYATTRIBUTE\_PROPERTY\_GETTER(ClassName, PropertyName) \

GAMEPLAYATTRIBUTE\_VALUE\_GETTER(PropertyName) \

GAMEPLAYATTRIBUTE\_VALUE\_SETTER(PropertyName) \

GAMEPLAYATTRIBUTE\_VALUE\_INITTER(PropertyName)

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "BaseAttributeSet")

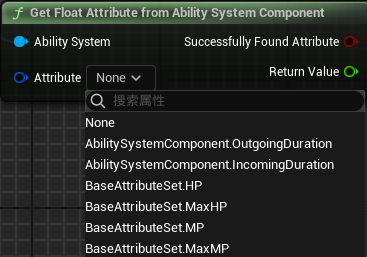
FGameplayAttributeData HP;

ATTRIBUTE\_ACCESSORS(UBaseAttributeSet, HP);

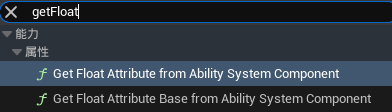
AS的初始化也是个问题。这里我们选用GAS表格进行初始化，也就是行结构为AttributeMetaData的DataTable。注意这个表格的行命名是很有讲究的。命名应当是 BaseAttributeSet.HP 这样的格式，也就是<类名> . <变量名>这样的格式。

然后要给Player初始化，初始化的方式就是在Player的ASC的细节中，“属性测试--默认起始数据”添加一个索引，索引0的默认起始表为刚创建的表格，属性为创建的C++的BaseAttributeSet。

要活的表格中的数据，或者说为了获取属性值，用ASC的Get Float Attribute From Ability System Component函数即可



注意这里有一个获取BaseValue的函数。



额外注意，Attribute的数据类型为FGameplayAttributeData，换言之，都是float类型的。

## GE

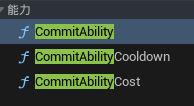
UGameplayEffect，技能效果，change

GE只存储数值不参与逻辑。

以冷却为例，要实现技能冷却，首先要有GE\_CD。在GE的类默认值细节中，“持续时间--持续时间策略”为拥有持续时间，下面的可扩展浮点幅度为CD值。然后在“Gameplay效果--组件”中选择“用标签阻止能力”，并将标签赋值。这个标签需要是技能的本身。也就是说，我们这个GE的实际效果就是通过“用标签阻止能力”来阻止标签表示的能力。

然后需要提交CD。提交CD在GA中提交就可以。只是需要注意什么时候提交CD。有的技能是生效前提交CD，有的技能是生效后提交CD。

提交的函数是CommitAbilityCooldown



然后实现技能的法力消耗。

在GA的类默认值细节中，“开销--成本Gameplay效果类”选择GE\_Cost即可。当然他不是马上就生效了，你需要去调用CommitAbilityCost函数。

如果可以让CD和Cost一起用，就可以直接使用CommitAbility函数，而不是使用两个CommitAbility<>函数。

然后实现伤害，也就是GE\_Damage。

因为伤害为一次性伤害，所以持续时间策略为实时。

伤害要关注修饰符，而不是组件。创建一个修饰符，修饰符中第一个参数就是Attribute。因为Damage，所以修饰符幅度为可扩展浮点幅度-20，注意是负值，因为修改器操作是加。

应用伤害就是对ASC调用ApplyGameplayEffectToTarget函数。

注意，这些应该是写在GA中的。但是怎么把数据从Character传递到GA呢？

先在Player中实现这个功能。因为ASC是Player的成员不是GA的成员。

当应用伤害时，同时也要实现伤害溢出的处理。这个需要在C++中处理。

在UAttributeSet基类中有一个后处理函数

virtual void PostGameplayEffectExecute(const struct FGameplayEffectModCallbackData &Data) { }

这是个额外的计算，所有的伤害计算完后，会执行这个函数。

我们在BaseAttributeSet.cpp中重写该函数，在.h中声明该函数。（不需要标记UFUNCTION，因为不在蓝图中进行处理）

在cpp中实现时，需要加上头文件（如果不加，Data不能 . 出东西）

#include "GameplayEffectExtension.h"

'''

#include "BaseAttributeSet.h"

#include "GameplayEffectExtension.h"

void UBaseAttributeSet::PostGameplayEffectExecute(const FGameplayEffectModCallbackData& Data)

{

Super::PostGameplayEffectExecute(Data);

//对父类的继承处理

if (Data.EvaluatedData.Attribute == GetHPAttribute())

{

SetHP(FMath::Clamp(GetHP(), 0.0, GetMaxHP()));

//set HP，做一个数学上的夹值

//对GetHP的下限设置为0，上限设置为GetMaxHP

//GetMaxHP就是表格里给的那个100

}

if (Data.EvaluatedData.Attribute == GetMPAttribute())

{

SetMP(FMath::Clamp(GetMP(), 0.0, GetMaxMP()));

}

}

'''

**保存**后回到引擎里编译

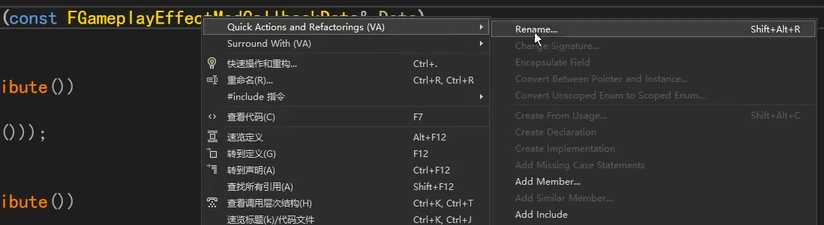
编译成功后，再进入游戏，攻击敌人时，就可以发现，敌人的生命值不会再低于0.

（如果在sln里编译生成，那么一定会报错，报错为MSB307，报错原因是和sln中的live coding实时编译冲突了。所以要在引擎里编译）

（建议使用rider）

要想得到能控制Data的头文件，可以用这样的方法：

右键Data的类型，点击Quick Action And Refactorings（VA），选择Add Include



对于GE，还有要考虑的一点：不同等级不同数值。这一点如何体现？我们首先在外面创建csv，命名为MeleeDamageLevel.csv

保存如下内容

0和1表示等级



创建好表格后，从windows资源管理器中将创建的csv文件拖入到引擎的内容管理器中

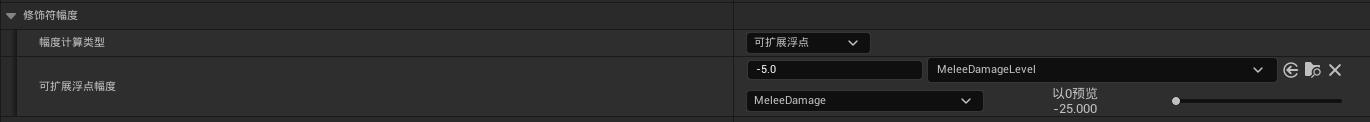


插值类型选择常量即可。

接下来使用这个表格。

进入GE\_Melee\_Damage，

在“Gameplay效果--修饰符--索引0--修饰符幅度--可扩展浮点幅度”中选择刚导入的表。曲线选择MeleeDamage。之后会有预览

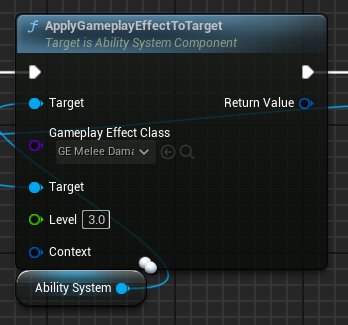


这个预览是用表格中的数据乘上前面给出的-5。我们将-5改成-1即可。

后面如果我们修改了csv中的数据，也并不需要重新导入表格，只需要右键导入的曲线表格资产，选择“重新导入”即可

也就是说，在磁盘上的这个表格文件，不能乱动位置。

具体要怎么用呢，在BP\_BaseCharacter中，有ApplyGameplayEffectToTarget节点，节点有一个形参为Level。我们只需要给Level赋值，就自动使用了曲线表格中的数据



注意，Level是float值，也就是说，他完全可以不代表真正的level

有时我们需要向GA中传递一些参数，比如让GA知道我打的是谁。当这个是从外部传递的时候，我们有下面几种做法：

GA带参数激活、GA中的Task

**带参数激活**：

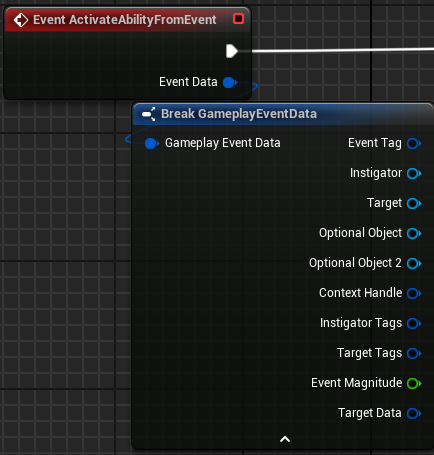
在GA的类默认值细节中“触发器--能力触发”



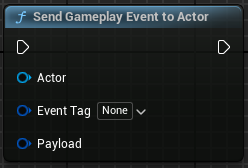
比如我们可以设置成如图所示的参数，表示有这个标签的时候就会触发能力。

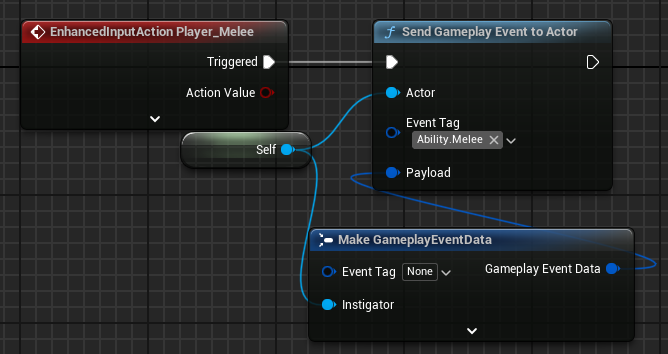


此时就不能用EventActivateAbility了，而是用EventActivateAbilityFromEvent。

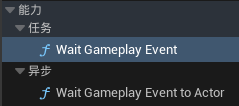


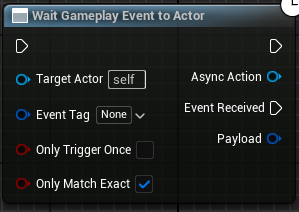
此时我们回到角色蓝图中，就不能用TryActivateAbilities了，要用SendGameplayEventToActor

（该函数可用于在指定Actor上触发能力，并携带有用的有效负载数据）

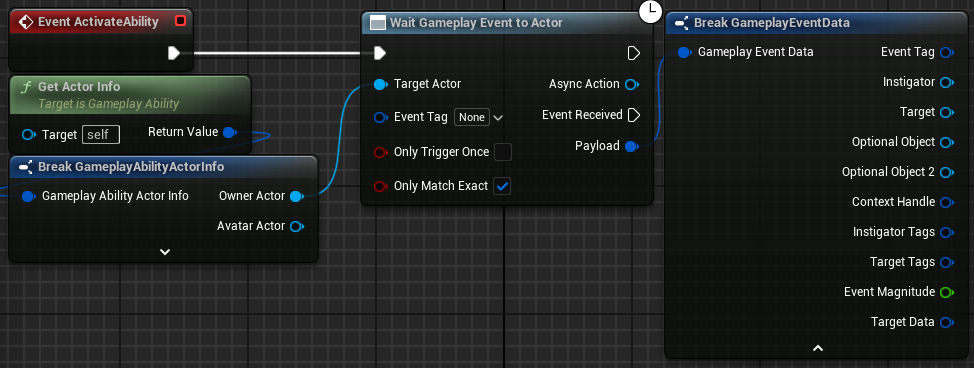


**GA中的Task**：Wait Gameplay Event



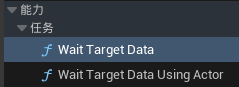


通常，我们可以使用toActor，从GetActorInfo中获取OwnerActor，将OwnerActor传值给TargetActor，通过EventTag进行赋值之后，获得这个GameplayEvent的返回值，来获得对应数据。



如果要用ToNone，则是向GA被激活的这个实例去传递。一般用的不多。

这里也可以用WaitTargetData来实现功能。他功能更强大，甚至可以直接传递碰撞到的结果。但是它的返回值是C++写的，我们不能在蓝图中看到他的返回值有什么，所以这个需要记。



AS与GE

GC，UGameplayCueNotify，特效部分，视觉特效

Tag，FGameplayTag，技能涉及的条件

Task，UGameplayTask，异步操作，技能的长时行动

Event，FGameplayEventData，ASC之间发送一些游戏事件来通知对方。技能消息事件