## 技能信息

通过一个struct来展示技能的信息。这个结构体存储了技能的所有信息。这样，在MainUI中，就可以获得技能的信息并展示。

创建C++类，父类为GameplayAbility，命名为BaseGameplayAbility。

在.h中，构建结构体

UENUM(BlueprintType)

enum class ECostType : uint8

{

    HP,

    MP

};

USTRUCT(BlueprintType)

struct FGameplayAbilityInfoStruct

{

    GENERATED\_BODY()

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

    float CD;

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

    ECostType CostType;

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

    float CostValue;

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

    UMaterialInstance\* IconMaterial;// 技能图标

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

    TSubclassOf<UBaseGameplayAbility> AbilityClass;

    FGameplayAbilityInfoStruct();

    FGameplayAbilityInfoStruct(float CD, ECostType CostType, float CostValue,UMaterialInstance\* IconMaterial,TSubclassOf<UBaseGameplayAbility> AbilityClass);

};

UCLASS()

class CNGAS\_API UBaseGameplayAbility : public UGameplayAbility

{

    GENERATED\_BODY()

public:

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

    UMaterialInstance\* IconMaterial;

    UFUNCTION(BlueprintCallable,Category="AbilityInfo")

    FGameplayAbilityInfoStruct GetAbilityInfo(int level);

};

实现两个构造函数。

FGameplayAbilityInfoStruct::FGameplayAbilityInfoStruct():

    CD(0),

    CostType(ECostType::MP),

    CostValue(0),

    IconMaterial(nullptr),

    AbilityClass(nullptr)

{}

FGameplayAbilityInfoStruct::FGameplayAbilityInfoStruct(float CD, ECostType CostType, float CostValue,

    UMaterialInstance\* IconMaterial, TSubclassOf<UBaseGameplayAbility> AbilityClass)

{

    this->CD=CD;

    this->CostType=CostType;

    this->CostValue=CostValue;

    this->IconMaterial=IconMaterial;

    this->AbilityClass=AbilityClass;

}

实现GetAbilityInfo函数

对照GE的细节面板，英文版

FGameplayAbilityInfoStruct UBaseGameplayAbility::GetAbilityInfo(int level)

{

    // 这个函数的实现，最好对照着GE的细节面板看

    UGameplayEffect\* CDEffect = GetCooldownGameplayEffect();

    UGameplayEffect\* CostEffect = GetCostGameplayEffect();

    float CD=0;

    ECostType CostType = ECostType::MP;

    float CostValue=0;

    if(CDEffect && CostEffect)

    {

        // 这一句会给CD进行赋值

        CDEffect->DurationMagnitude.GetStaticMagnitudeIfPossible(level,CD);

        if(CostEffect->Modifiers.Num()>0)

        {

            // 获取GE蓝图中，“Gameplay效果--修饰符”中的第一个

            // 修饰符就是修改器，即Modifier

            // 获取花费的是哪一个类型

            FGameplayModifierInfo CostEffectModifierInfo = CostEffect->Modifiers[0];

            CostEffect->DurationMagnitude.GetStaticMagnitudeIfPossible(level,CostValue);

            FString CostTypeName = CostEffectModifierInfo.Attribute.AttributeName;

            if(CostTypeName=="HP")

            {

                CostType=ECostType::HP;

            }

            if(CostTypeName=="MP")

            {

                CostType=ECostType::MP;

            }

            return FGameplayAbilityInfoStruct(CD,CostType,CostValue,IconMaterial,GetClass());

        }

}

return FGameplayAbilityInfoStruct();

}

然后实现：学习技能的时候就获取技能信息。

进入BaseCharacter.h，

UFUNCTION(BlueprintCallable, Category="BaseCharacter")  
FGameplayAbilityInfoStruct GameplayAbilityInfo(TSubclassOf<UBaseGameplayAbility> AbilityClass,int level);

实现该函数

FGameplayAbilityInfoStruct ABaseCharacter::GameplayAbilityInfo(TSubclassOf<UBaseGameplayAbility> AbilityClass,

    int level)

{

    UAbilitySystemComponent\* MyAbilitySystemComponent = this->FindComponentByClass<UAbilitySystemComponent>();

    UBaseGameplayAbility\* AbilityInstance = AbilityClass->GetDefaultObject<UBaseGameplayAbility>();

    // 如果技能存在而且学习到技能了

    if(MyAbilitySystemComponent&&AbilityInstance)

    {

        return AbilityInstance->GetAbilityInfo(level);

    }

    return FGameplayAbilityInfoStruct();

}

然后实现:利用结构体信息创建单个技能UI

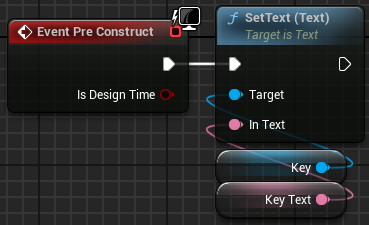
创建WBP\_AbilitySlot

添加画布面板和图像

我们看一下图标上应该有技能的什么信息

计时器文本：一个通用文本，用于计时

按键文本：通过某按键来使用技能。

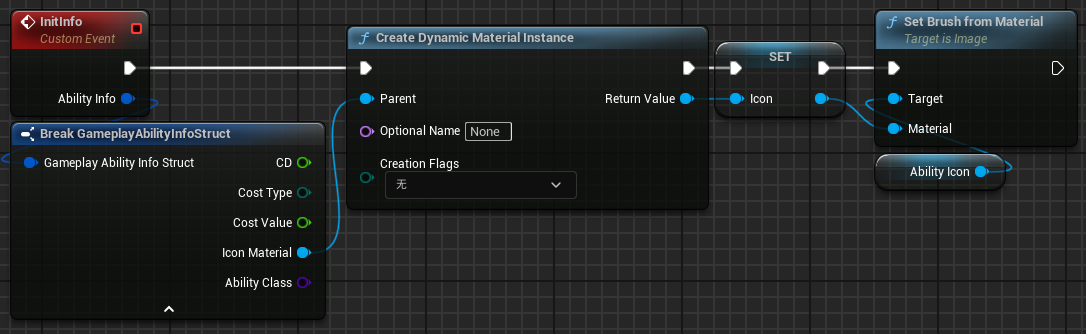


KeyText是一个变量。

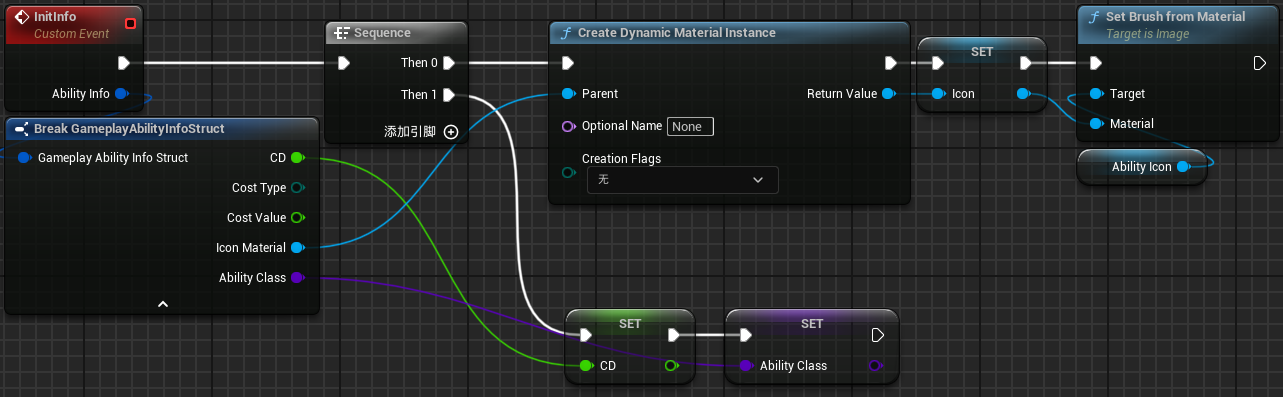
接下来使用刚创建的结构体信息。

创建自定义事件，给他添加参数，参数类型为刚创建的技能结构体GameplayAbilityInfoStruct。

首先创建一个动态材质实例CreateDynamicMaterialInstance。

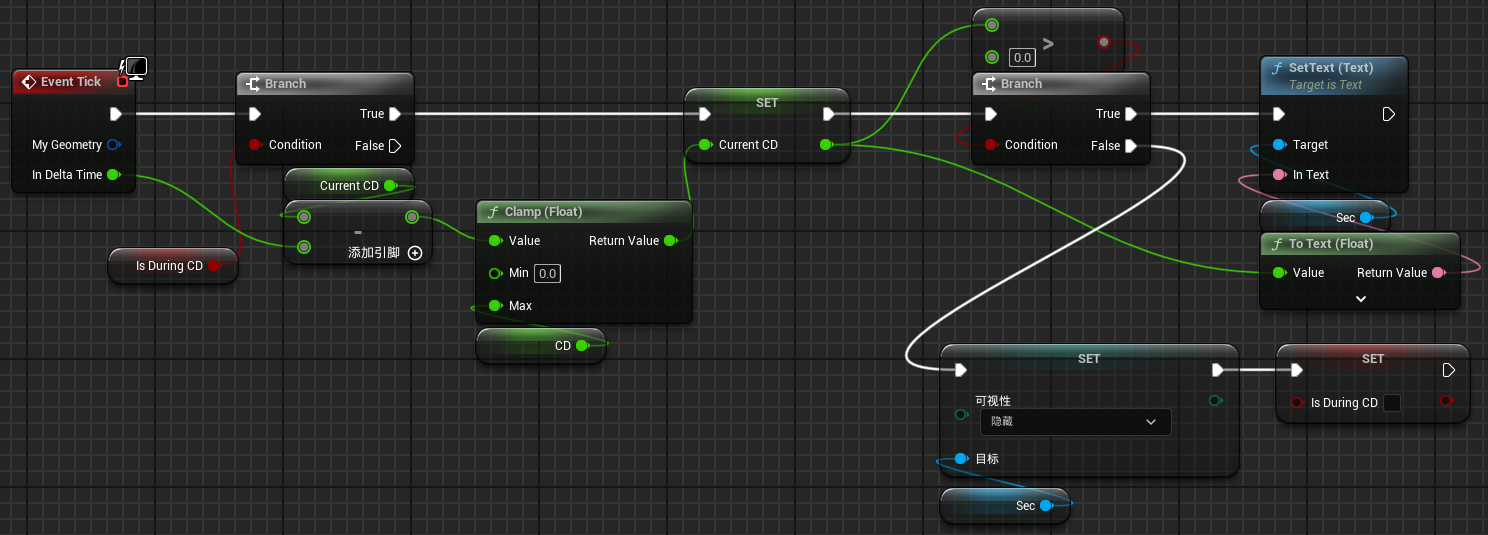


这样就把图标设置好了。



然后实现释放技能后更新技能CD

set visibility节点



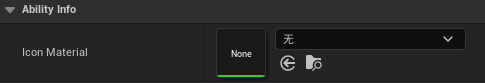
## 技能前置知识

1. 写两个公共的库函数（MVC框架简单介绍）
2. 添加技能图标到MainUI并在Player进行初始化
3. 释放技能时通知UI进入冷却

**打开GA\_BaseAbility，将父类重新选择**

然后再看类默认值，就能看到我们创建的IconMaterial变量

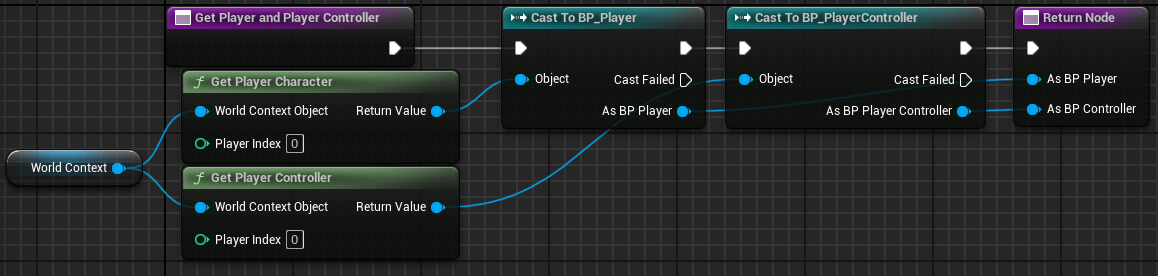
UPROPERTY(BlueprintReadWrite,EditAnywhere,Category="AbilityInfo")



**写两个公共的库函数**

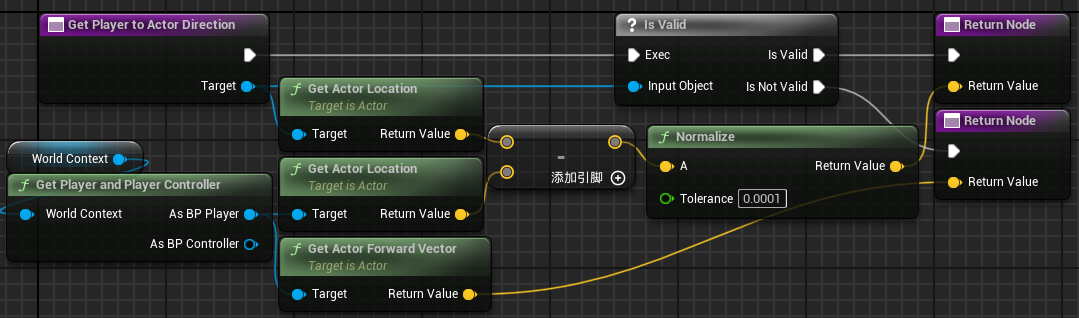
创建函数库，命名为BPF\_Lib

函数：GetPlayerAndPlayerController，纯函数



注意，WorldContext应该连上，不连接的话虽然运行不报错，但是打包的时候报错。

函数：GetPlayerToActerDirection



MVC框架：

Model

View

Controller

Model：业务逻辑与数据结构

View：外观与用户界面

Controller：接收输入，根据输入更新Model和View

**添加技能图标到MainUI并在Player进行初始化**

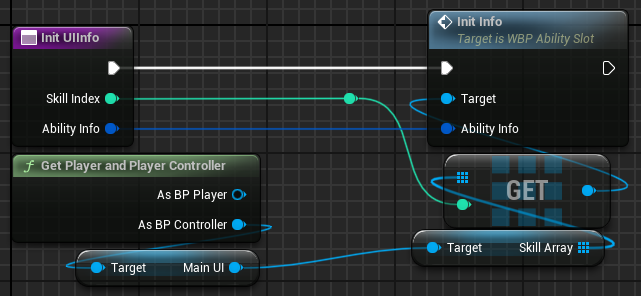
进入WBP\_MainUI，加上WBP\_AbilitySlot，全部设置为变量。

我们设置5个技能。

首先创建UMG\_AbilitySlot数组类型的变量，命名为SkillArray，在EventConstruct时将5个技能加入到数组中



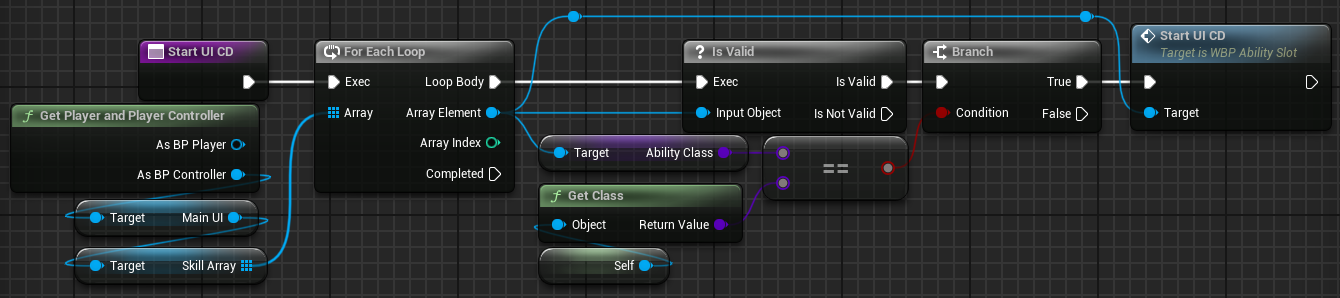
进入BP\_Player，实现函数InitUIInfo。



**释放技能时通知UI进入冷却**

通知UI更新需要写在GA\_BaseAbility中

创建函数，命名为StartUI\_CD



## 内容补充

前面只实现了生命值改变时的通知，并没有实现法力值改变时的通知。

进入BaseCharacter.h，

DECLARE\_DYNAMIC\_MULTICAST\_DELEGATE\_OneParam(FOnManaChangeEvent, float, NewMana);

类中

UPROPERTY(BlueprintAssignable, Category="Ability")  
FOnManaChangeEvent MPChangeEvent;

void OnManaAttributeChanged(const FOnAttributeChangeData& Data);

实现函数

void ABaseCharacter::OnManaAttributeChanged(const FOnAttributeChangeData& Data)  
{  
 MPChangeEvent.Broadcast(Data.NewValue);  
}

进入BeginPlay函数

这个函数相比之前，改的只有if里面加的一句。

void ABaseCharacter::BeginPlay()

{

    Super::BeginPlay();

    UAbilitySystemComponent\* MyAbilitySystemComponent = this->FindComponentByClass<UAbilitySystemComponent>();

    if(MyAbilitySystemComponent)

    {

        MyAbilitySystemComponent->GetGameplayAttributeValueChangeDelegate(UBaseAttributeSet::GetHPAttribute()).AddUObject(this,&ABaseCharacter::OnHealthAttributeChanged);

        MyAbilitySystemComponent->GetGameplayAttributeValueChangeDelegate(UBaseAttributeSet::GetMPAttribute()).AddUObject(this,&ABaseCharacter::OnManaAttributeChanged);

    }

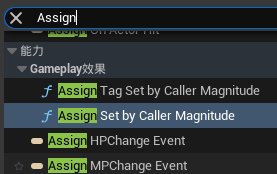
}

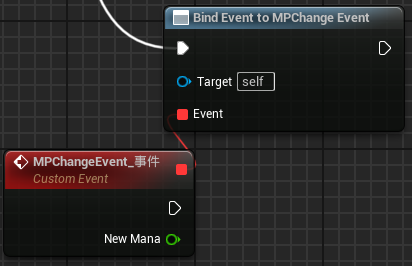
If里面加的这一句的AddUObject(this,&ABaseCharacter::OnManaAttributeChanged)这一部分，最后的OnManaAttributeChanged就是前面的创建新属性时的函数名。

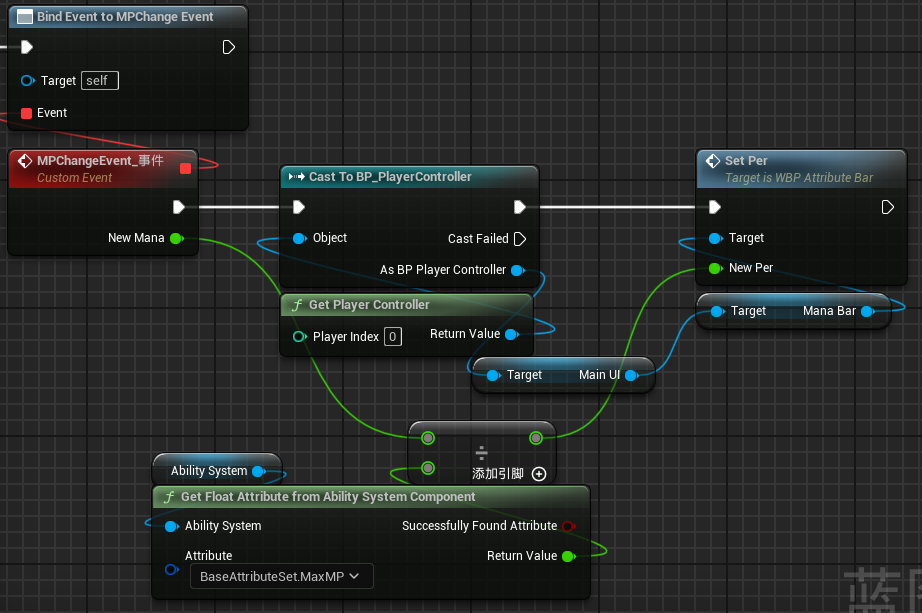
然后对于魔法值改变的通知，我们要用它

进入BP\_Player

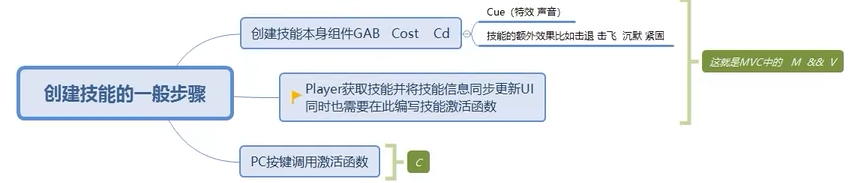
怎么用呢，首先，绑定它，搜索Assign就可以直接绑定。这就是事件分发器的“分配”







## 创建技能的一般步骤



首先获取技能，然后更新UI和激活技能。通过Tag激活技能。