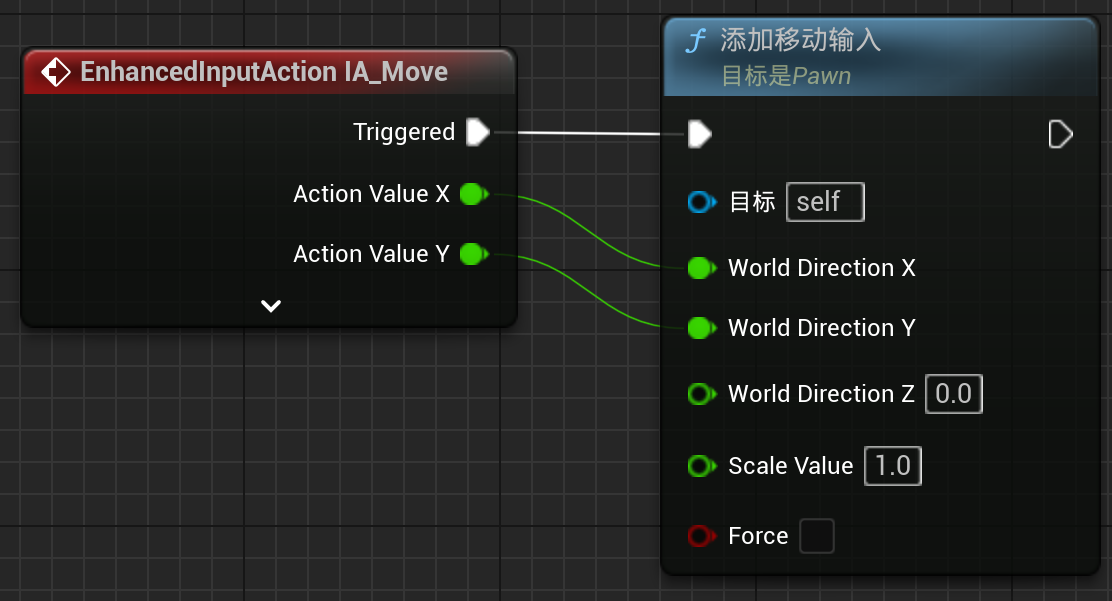
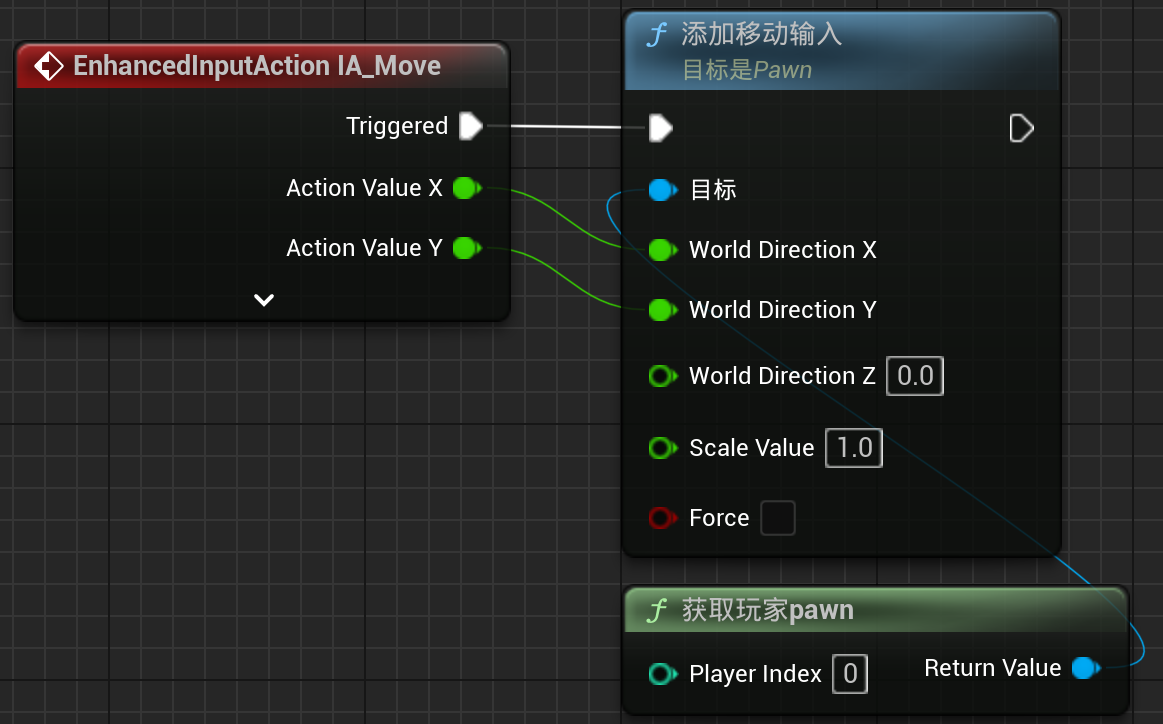
首先转换一个思想

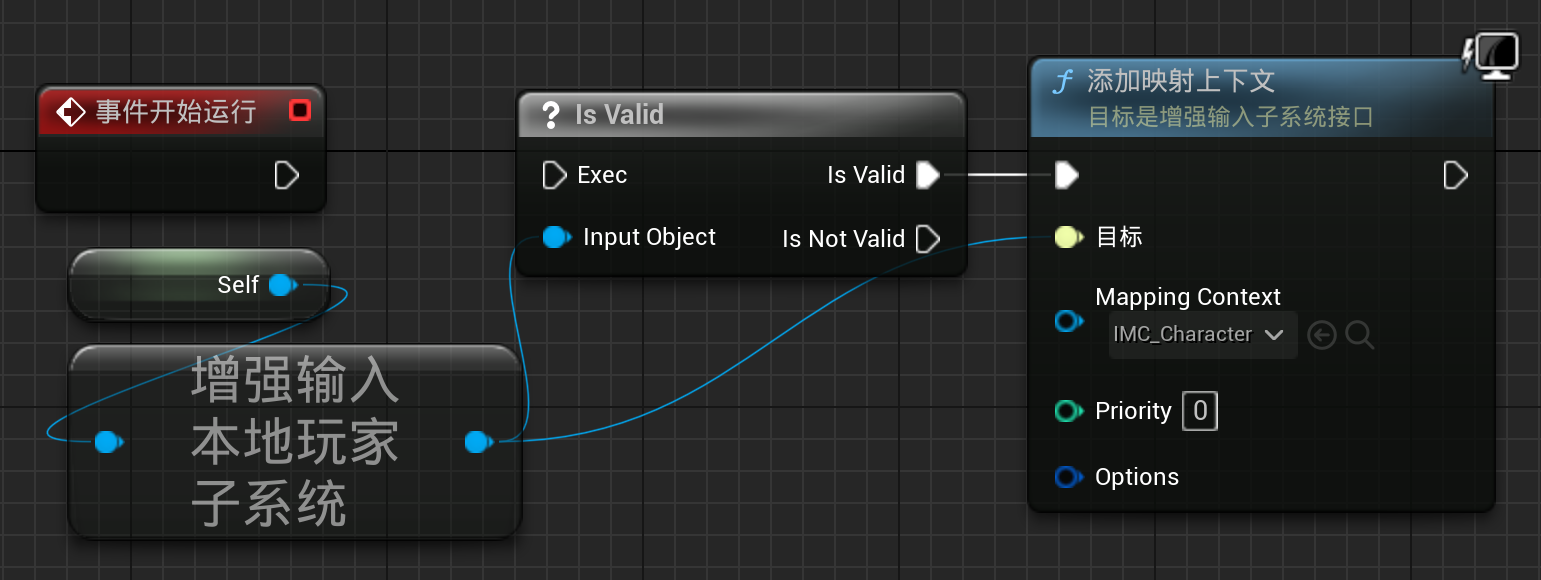
之前我们经常在BP\_Character中在IA\_Move后调用AddMovementInput，也就是如图



现在我们不要这样了，我们要在PlayerController中实现这个节点族



另外的，在PlayerController中，我们也需要实现增强输入系统



下面我们就在C++中实现这些功能

前置引用类

class UInputMappingContext;

class UInputAction;

struct FInputActionValue;

在头文件中，我们需要声明一些变量和函数

protected:

    virtual void BeginPlay() override;

    // 实现移动委托

    virtual void SetupInputComponent() override;

public:

    ABasePlayerController();

    UPROPERTY(EditAnywhere,Category="Input")

    TObjectPtr<UInputMappingContext> IMC\_Character;

private:

    // 实现Input Action的赋值

    UPROPERTY(EditAnywhere,Category="Input")

    TObjectPtr<UInputAction> IA\_Move;

    // 响应输入移动角色

    // 在SetupInputComponent中进行绑定

    void Move(const FInputActionValue& InputActionValue);

然后就是实现这些函数

void ABasePlayerController::BeginPlay()

{

    Super::BeginPlay();

    // 输入映射上下文与Add Mapping Context

    check(IMC\_Character);

    UEnhancedInputLocalPlayerSubsystem\* Subsystem=ULocalPlayer::GetSubsystem<UEnhancedInputLocalPlayerSubsystem>(GetLocalPlayer());

    check(Subsystem);

    Subsystem->AddMappingContext(IMC\_Character,0);

    // 下面这一部分是什么我还不太清楚，这个是Aura课程上的

    FInputModeGameAndUI InputModeData; //

    InputModeData.SetLockMouseToViewportBehavior(EMouseLockMode::DoNotLock);

    InputModeData.SetHideCursorDuringCapture(false);

    SetInputMode(InputModeData);

}

void ABasePlayerController::SetupInputComponent()

{

    Super::SetupInputComponent();

    check(InputComponent);// InputComponent是父类自带的一个变量,此处的检查用于下一行的强转判断

    UEnhancedInputComponent\* EnhancedInputComponent=Cast<UEnhancedInputComponent>(InputComponent);

    check(EnhancedInputComponent);

    // 绑定Move函数

    EnhancedInputComponent->BindAction(IA\_Move,ETriggerEvent::Triggered,this,&ABasePlayerController::Move);

}

void ABasePlayerController::Move(const FInputActionValue& InputActionValue)

{

    const FVector2D InputAxisVector = InputActionValue.Get<FVector2D>();

    const FRotator Rotation = GetControlRotation();

    const FRotator YawRotation(0,Rotation.Yaw,0);

    const FVector ForwardDirection = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::X);

    const FVector RightDirection = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::Y);

    // AddMovementInput

    if(GetPawn<APawn>())

    {

        GetPawn<APawn>()->AddMovementInput(ForwardDirection,InputAxisVector.X);

        GetPawn<APawn>()->AddMovementInput(RightDirection,InputAxisVector.Y);

    }

}

使用时，创建BasePlayerController的蓝图类，在类默认值中给EditAnywhere的那些变量进行赋值

最后，在构造函数中初始化一些细节面板中的相关参数

ABasePlayerController::ABasePlayerController()

{

    bReplicates=true;

    bShowMouseCursor = true;

    bEnableClickEvents = true;

    bEnableTouchEvents = true;

    bEnableMouseOverEvents = true;

    bEnableTouchOverEvents = true;

}