## 增强输入系统资产

D为默认，W为拌和输入轴值

拌和输入轴值注意排序顺序

## 将输入映射上下文绑定到了玩家控制器

创建C++类

创建AuraPlayerController类

在C++中实现增强输入的绑定

首先在protected中重载BeginPlay函数

然后在public中实现构造函数

因为Aura这个游戏是多人游戏

我们需要确保这个PlayerController是可复制的

（可复制的，是指在多人游戏中，如果这个东西的实体在服务器上发生了变化，这个东西会被复制到客户端中，是上变给到下）

在构造函数中，bReplicates = true;

然后在BeginPlay中，完成输入映射上下文的实现

实现时，需要先创建变量来存储 输入映射上下文

在类中

class UInputMappingContext; // 需要在Build.cs中加上“EnhancedInputs”

UPROPERTY(EditAnywhere,Category="Input")

TObjectPtr<UInputMappingContext> AuraContext;

然后进入.Build.cs中，添加“EnhancedInput”

PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject", "Engine", "InputCore","EnhancedInput" })

然后在BeginPlay中

#include "EnhancedInputSubsystems.h"

void AAuraPlayerController::BeginPlay()

{

    Super::BeginPlay();

    check(AuraContext); // 检查AuraContext是不是空指针

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

    check(Subsystem);

    Subsystem->AddMappingContext(AuraContext,0);

    bShowMouseCursor = true;

    DefaultMouseCursor = EMouseCursor::Default;

    FInputModeGameAndUI InputModeData; //

    InputModeData.SetLockMouseToViewportBehavior(EMouseLockMode::DoNotLock);

    InputModeData.SetHideCursorDuringCapture(false);

    SetInputMode(InputModeData);

}

这时就已经将输入映射上下文绑定到了玩家控制器

## 移动委托

首先protected重载Setup Input Component函数

void AAuraPlayerController::SetupInputComponent()

{

    Super::SetupInputComponent();

    InputComponent;// 父类自带的一个变量

    check(InputComponent);

    // #include "EnhancedInputComponent.h"，用于解决Cast中类不完整的问题

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

    // Cast可以换成CastCheck，既有强制类型转换，也有指针为空检查。下一行可以替换上两行语句

    // UEnhancedInputComponent\* EnhancedInputComponent2 = CastChecked<UEnhancedInputComponent>(InputComponent);

}

为了InputAction，我们也需要在类中定义几个private变量

class UInputAction;

UPROPERTY(EditAnywhere,Category="Input")

TObjectPtr<UInputAction> MoveAction;

在最后我们需要在蓝图中设置它

我们想要绑定一个函数来响应输入并移动角色。

我们需要一个目的是移动角色的函数。这个函数只会被类的内部调用。创建private函数Move

struct FInputActionValue;

void Move(const FInputActionValue& InputActionValue);

这个Move函数将作为我们的回调函数

首先绑定，在BeginPlay中加上一行

EnhancedInputComponent->BindAction(MoveAction,ETriggerEvent::Triggered,this,&AAuraPlayerController::Move);

然后实现Move函数

void AAuraPlayerController::Move(const FInputActionValue& InputActionValue)

{

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

    const FRotator Rotation = GetControlRotation();

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

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

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

    // AddMovementInput

    APawn\* ControlledPawn = GetPawn<APawn>();

    if (ControlledPawn)

    {

        ControlledPawn->AddMovementInput(ForwardDirection,InputAxisVector.Y);

        ControlledPawn->AddMovementInput(RightDirection,InputAxisVector.X);

    }

}

然后创建蓝图类PlayerController，在细节中将相关参数赋值（EditAnywhere的那些参数）