// Fill out your copyright notice in the Description page of Project Settings.

#include "Grabber.h"

#include "DrawDebugHelpers.h"

#include "Engine/World.h"

#include "GameFramework/PlayerController.h"

#define OUT // Allow us to add some text as comments within the code (informative only!). Here it means that the variables are out in the log

// Sets default values for this component's properties

UGrabber::UGrabber()

{

// Set this component to be initialized when the game starts, and to be ticked every frame. You can turn these features

// off to improve performance if you don't need them.

PrimaryComponentTick.bCanEverTick = true;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Called when the game starts

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void UGrabber::BeginPlay()

{

Super::BeginPlay();

// TESTING PURPOSE ONLY: Text to log out to check for the spawn\_default\_BP works well:

// UE\_LOG(LogTemp, Warning, TEXT("%s works fine."), \*GetOwner()->GetActorLabel());

FindPhysicsHandle();

SetupInputComponent();

}

void UGrabber::FindPhysicsHandle()

{

// Checking physics handle component

PhysicsHandle = GetOwner()->FindComponentByClass<UPhysicsHandleComponent>();

if (PhysicsHandle)

{

// Physics handle is bound

}

else

{

UE\_LOG(LogTemp, Error, TEXT("No physics handle component found in %s"), \*GetOwner()->GetName());

}

}

// Set up the input component

void UGrabber::SetupInputComponent()

{

InputComponent = GetOwner()->FindComponentByClass<UInputComponent>();

//log out if component is associated to the player (i.e. 'pawn\_default\_BP')

if (InputComponent)

{

UE\_LOG(LogTemp, Warning, TEXT("Input component found on %s"), \*GetOwner()->GetName());

InputComponent->BindAction("Grab", IE\_Pressed, this, &UGrabber::Grab);

InputComponent->BindAction("Grab", IE\_Released, this, &UGrabber::Release);

// EXPLANATIONS:

// "Grab": name of the object binding (see [Project settings/Input/ActionMappings] in UE)

// IE\_Pressed: type of action required for the function to trigger

// this (\*Object): refers to this class

// &UGrabber is the ADDRESS where to find the function 'Grab' defined below. It tells the BindAction function where to go in the permanent memory in order to get the 'Grab' function.

// It is used to store the function Grab into the permanent storage rather than in the dynamic storage (i.e. stack, heap)

// We access anything that is the permanent storage using the :: (scope) operator

}

else

{

UE\_LOG(LogTemp, Warning, TEXT("Input component missing on %s"), \*GetOwner()->GetName());

}

}

void UGrabber::Grab()

{

UE\_LOG(LogTemp, Warning, TEXT("Grabber pressed"));

FVector PlayerViewPointLocation;

FRotator PlayerViewPointRotation;

GetWorld()->GetFirstPlayerController()->GetPlayerViewPoint( //'GetPlayerViewPoint' function is void and has 2 parameters with 'out' function, i.e. it will change the vaue of a variable

OUT PlayerViewPointLocation, //The 'OUT' macro (see #define above) does nothing to the code.

OUT PlayerViewPointRotation

);

FVector LineTraceEnd = PlayerViewPointLocation + PlayerViewPointRotation.Vector() \* Reach;

// 1. Ray-cast when key is pressed and see if we reached any actor with any physic body collision channel set

FHitResult HitResult = GetFirstPhysicsBodyInReach();

UPrimitiveComponent\* ComponentToGrab = HitResult.GetComponent();

// If we hit something, then attach the physics handle

if (HitResult.GetActor())

{

// Attach physics handle

PhysicsHandle->GrabComponentAtLocation

(

ComponentToGrab,

NAME\_None,

LineTraceEnd

);

}

}

void UGrabber::Release()

{

UE\_LOG (LogTemp, Warning, TEXT("Grabber released"))

//Remove/release physics handle

PhysicsHandle->ReleaseComponent();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Called every frame

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void UGrabber::TickComponent(float DeltaTime, ELevelTick TickType, FActorComponentTickFunction\* ThisTickFunction)

{

Super::TickComponent(DeltaTime, TickType, ThisTickFunction);

// Get the player's view point

FVector PlayerViewPointLocation;

FRotator PlayerViewPointRotation;

GetWorld()->GetFirstPlayerController()->GetPlayerViewPoint( //'GetPlayerViewPoint' function is void and has 2 parameters with 'out' function, i.e. it will change the vaue of a variable

OUT PlayerViewPointLocation, //The 'OUT' macro (see #define above) does nothing to the code.

OUT PlayerViewPointRotation //It is just for readibility because the 2 param will be modified

);

FVector LineTraceEnd = PlayerViewPointLocation + PlayerViewPointRotation.Vector() \* Reach;

// If the physics handle is attached

if (PhysicsHandle->GrabbedComponent)

{

PhysicsHandle->SetTargetLocation(LineTraceEnd);

}

// move the object we are holding

}

FHitResult UGrabber::GetFirstPhysicsBodyInReach() const

{

// Get the player's view point

FVector PlayerViewPointLocation;

FRotator PlayerViewPointRotation;

GetWorld()->GetFirstPlayerController()->GetPlayerViewPoint( //'GetPlayerViewPoint' function is void and has 2 parameters with 'out' function, i.e. it will change the vaue of a variable

OUT PlayerViewPointLocation, //The 'OUT' macro (see #define above) does nothing to the code.

OUT PlayerViewPointRotation //It is just for readibility because the 2 param will be modified

);

FVector LineTraceEnd = PlayerViewPointLocation + PlayerViewPointRotation.Vector() \* Reach;

/\* #### CODE TESTING PURPOSE ONLY - START ####

// Logout the player view

UE\_LOG(LogTemp, Warning, TEXT("ViewPoint location: %s, Viewpoint rotation: %s"),

\*PlayerViewPointLocation.ToString(),

\*PlayerViewPointRotation.ToString()

);

// Ray-cast out to a certain distance Draw line from player to show the reach

DrawDebugLine(GetWorld(), PlayerViewPointLocation, LineTraceEnd, FColor(0, 255, 0), false, 0.f, 0, 5);

#### CODE TESTING PURPOSE ONLY - END #### \*/

// See what is hitting

FHitResult Hit;

FCollisionQueryParams TraceParams( // The 'FCollisionQueryParams' requires (FName InTraceTag, bool bInTraceComplex, const AActor \* InIgnoreActor)

FName(TEXT("")), // FName value required if the object we want to hit have a name. Here, they do not -> leave the text empty ("")

false, // bool bInTraceComplex: is our system of collision complex (e.g. see throught objects) or not? Here, false because we do not use the complex collision system (i.e. player collision in UE)

GetOwner() // we ignore ourself (i.e. 'GetOwner') because the raycast starts from withinour pawn object. If true, collision would happen all the time.

);

GetWorld()->LineTraceSingleByObjectType(

OUT Hit, // it outs a results, i.e. the object that is being hit

PlayerViewPointLocation, // from where the ray-cast starts (i.e. middle of the spawn sphere, see above))

LineTraceEnd, // where the ray-cast ends (see above)

FCollisionObjectQueryParams(ECollisionChannel::ECC\_PhysicsBody), // type of collision object type used in UE (see Collision/Collision Presets/Object type, in UE)

TraceParams);

// See what the ray hits

AActor\* ActorHit = Hit.GetActor();

if (ActorHit)

{

UE\_LOG(LogTemp, Warning, TEXT("The object hit is: %s"), \*(ActorHit->GetName()));

}

return Hit;

}