IPL

escola superior de tecnologia e gestão

instituto politécnico de leiria

Game Network Programming
Project 1

December 14th, 2025

Gonçalo Sampaio dos Reis Cavaca Gil, 2025212642

Index

1. Planning

The chosen game mechanic to be implemented was an air dash that could be used for mobility. This dash will launch the player forward but also slightly up, only being able to be used the dash once before the player must "recharge" it by stepping inside a red "Recharge Zone". The third-person sidescroller template was chosen as a base for implementing the mechanic as it was the most traditionally suited to the type of mechanic implemented.

2. Implementation

The main c++ classes responsible for the implementation of the mechanic are the SideScrollingCharacter and Prj1_DashRechargeZone classes.

2.1. SideScrollingCharacter

The SideScrollingCharacter class has been modified, implementing functions, variables and input actions to support the dash.

```
//// added code
public:
    /** Handles dash input */
    UPROPERTY(EditAnywhere, BlueprintReadOnly,  Category="Input",meta = (AllowPrivateAccess =
"true"));
    UInputAction* DashAction;

    // function to handle dashing
    UFUNCTION(BlueprintCallable, Category="Input")
    virtual void DoDashStart();

    // function to handle recharging the dash
    UFUNCTION(BlueprintCallable, Category="Dash")
    void RechargeDash();

    /** Boolean to handle when the player can dash*/
    UPROPERTY(EditAnywhere, Category="Dash")
    bool bCanDash = true; // starts as true
```

Figure 1: Code added to the SideScrollerCharacter class's header file

```
void ASideScrollingCharacter::SetupPlayerInputComponent(class UInputComponent *PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);

    // Set up action bindings
    if (UEnhancedInputComponent *EnhancedInputComponent = Cast<UEnhancedInputComponent>(PlayerInputComponent))
    {
        // Jumping
        EnhancedInputComponent->BindAction(JumpAction, ETriggerEvent::Started, this, &ASideScrollingCharacter::DoJumpStart);
        EnhancedInputComponent->BindAction(JumpAction, ETriggerEvent::Completed, this, &ASideScrollingCharacter::DoJumpEnd);

        // Interacting
        EnhancedInputComponent->BindAction(InteractAction, ETriggerEvent::Triggered, this, &ASideScrollingCharacter::DoInteract);

        // Moving
        EnhancedInputComponent->BindAction(MoveAction, ETriggerEvent::Triggered, this, &ASideScrollingCharacter::Move);

        // Dropping from platform
        EnhancedInputComponent->BindAction(DropAction, ETriggerEvent::Triggered, this, &ASideScrollingCharacter::Drop);
        EnhancedInputComponent->BindAction(DropAction, ETriggerEvent::Completed, this, &ASideScrollingCharacter::DropReleased);

        //// Added Code
        // Dashing
        EnhancedInputComponent->BindAction(DashAction, ETriggerEvent::Triggered, this, &ASideScrollingCharacter::DoDashStart);
    }
}
```

Figure 2: Input handler code added to the SideScrollingCharacter class's .cpp file

Along with the code to handle inputs, steps were taken in the unreal engine editor to complete the addition of a new input. This includes creating a new input action, named IA_Dash, adding a new mapping to the IMC_Default data asset, and selecting IA_Dash as the new mapping in the SideScrollingCharacter blueprint.

Regarding the dash's functionality, SideScrollerCharacter implements two functions responsible for performing and recharging the dash, those being DoDashStart() and RechargeDash(), respectively.

```
///// Added Code
void ASideScrollingCharacter::DoDashStart()
{
    // check if player is allowed to dash
    if (bCanDash == true)
    {
        // get movement velocity
        FVector MovementVelocityVector = GetCharacterMovement()->Velocity;

        // if player is moving left or right
        if (MovementVelocityVector.X != 0)
        {
            // get direction of movement, (-1 for left, 1 for right)
            float MovementDirectionX = (MovementVelocityVector.X / abs(MovementVelocityVector.X));
            // get final launch vector by multiplying the direction of movement by the impulse
            FVector DashLaunchVector = FVector(MovementDirectionX * 1000.0f, 0.0f, 400.0f);

            // Launch the character using the launch vector
            LaunchCharacter(DashLaunchVector, true, true);
            // set bCanDash to false
            bCanDash = false;

            // debug prints
            // GEngine->AddOnScreenDebugMessage(-1, 15.0f, FColor::Red, TEXT("Started dash with Vector: {x}=%f,{y}=,{z}="), DashLaunchVector.X);
            //UE_LOG(LogTemp, Warning, TEXT("Started dash with Vector: {x}=%f,{y}=,{z}="), DashLaunchVector.X);
        }
    }
}
```

Figure 3: DoDashStart() code

DoDashStart() first checks if the player is allowed to dash and, if so, uses the SideScrollerCharacter's movement velocity vector to check whether they are moving to the left or right. This check is done by dividing the velocity vector's X value by its absolute value, thus ensuring a result of either 1 or -1 depending on the direction. A launch vector is then constructed using the movement direction, being used as the vector for LaunchCharacter(). This vector will launch the player towards the direction they are moving, using the Z component to also launch the player slightly upwards, making them fly further. Finally, the bCanDash boolean is set to false, requiring the player to recharge the dash before it can be used again.

```
void ASideScrollingCharacter::RechargeDash()
{
    bCanDash = true;
}
```

Figure 4: RechargeDash() code

RechargeDash() is a simple function that sets the bCanDash boolean to true when called. No delay has been implemented because playtesting revealed it was not required.

This function is called when the SideScrollingCharacter overlaps with a Recharge Zone, thus recharging the character's dash.

2.1. Prj1_DashRechargeZone

Prj1_DashRechargeZone defines an actor that can be placed in the world and, when passed through, recharges a SideScrollingCharacter's dash.

```
#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Prj1_DashRechargeZone.generated.h"

class UBoxComponent;

/*
    An intangible zone that recharges the player's dash when they pass through it
*/
UCLASS()
class PROJ1_V2_API APrj1_DashRechargeZone : public AActor
{
    GENERATED_BODY()

    /** Recharge Zone bounding box */
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category="Components", meta = (AllowPrivateAccess = "true"))
    UBoxComponent* Box;

public:
    // Sets default values for this actor's properties
    APrj1_DashRechargeZone();

protected:
    // defines behavior when an actor overlaps with the recharge zone
    UFUNCTION()
    void BeginOverlap(AActor* OverlappedActor, AActor* OtherActor);

};
```

Figure 5: Prj1_DashRechargeZone class's header file code

The Header file defines a bouding box used for overlap checks, a constructor function and a BeginOverlap() function.

```cpp
#include "Variant_SideScrolling/Gameplay/Prj1_DashRechargeZone.h"
#include "Components/BoxComponent.h"
#include "Components/SceneComponent.h"
#include "SideScrollingCharacter.h"

// Sets default values
APrj1_DashRechargeZone::APrj1_DashRechargeZone()
{
    PrimaryActorTick.bCanEverTick = false;

    // create the root component
    RootComponent = CreateDefaultSubobject<USceneComponent>(TEXT("Root"));

    // create the bounding box
    Box = CreateDefaultSubobject<UBoxComponent>(TEXT("Box"));
    Box->SetupAttachment(RootComponent);

    // configure the bounding box
    Box->SetBoxExtent(FVector(200.0f, 90.0f, 200.0f), false);
    Box->SetRelativeLocation(FVector(0.0f, 0.0f, 16.0f));

    Box->SetCollisionObjectType(ECC_WorldDynamic);
    Box->SetCollisionEnabled(ECollisionEnabled::QueryOnly);
    Box->SetCollisionResponseToAllChannels(ECR_Ignore);
    Box->SetCollisionResponseToChannel(ECC_Pawn, ECR_Overlap);

    // add the overlap handler
    OnActorBeginOverlap.AddDynamic(this, &APrj1_DashRechargeZone::BeginOverlap);

}

void APrj1_DashRechargeZone::BeginOverlap(AActor* OverlappedActor, AActor* OtherActor)
{
    // if the overalpped actor is a side-scrolling character
    if (ASideScrollingCharacter* OverlappingCharacter = Cast<ASideScrollingCharacter>(OtherActor))
    {
        // Recharge the character's dash
        OverlappingCharacter->RechargeDash();
    }
}
```

Figure 6: Prj1_DashRechargeZone class's .cpp file code

Prj1_DashRechargeZone's constructor creates a root component and bounding box, configuring the box's parameters and adding an overlap handler for when there is an actor overlap.

The BeginOverlap() function is called when an actor overlap occurs and checks if the overlapping actor is a SideScrollingCharacter, calling the character's RechargeDash() function if so.

Using the Prj1_DashRechargeZone C++ class, a blueprint was created in order to be placed in the level. A cube was added to the blueprint as a child of the bounding box, a translucent material was applied to it to make the recharge zone visible and the cube's collision was turned off so it would not affect the player.