**Game play logic**

Use the boolean in character to check if character is carrying the objective.

So in the HandleOverlap method add the following.

AFPSObjectiveCharacter\* MyPawn = Cast<AFPSObjectiveCharacter>(OtherActor);

we need to include the character

#include "FPSObjectiveCharacter.h"

Now we need 2 checks

If the variable MyPawn not null pointer and the boolean in it is 'true'

So do the following

if (MyPawn != nullptr && MyPawn->bIsCarryingObjective)

{

}

We need to disable the input if the condition is true.

We will do that in GameMode.

So go to GameMode.h and add this method.

void MissionComplete(APawn\* InstigatorPawn);

and do the definition in cpp file.

void AFPSObjectiveGameMode::MissionComplete(APawn \* InstigatorPawn)

{

if (InstigatorPawn)

{

InstigatorPawn->DisableInput(nullptr);

}

}

Argument passed to the method can be pointer to the player controller or a null pointer.

If nullpointer is passed, then input will be disabled.

We can do some extra logic in BP. For that add another function in game mode header file as follows.

UFUNCTION(BlueprintImplementableEvent, Category = "GameMode")

void OnMissionCompleted(APawn\* InstigatorPawn);

We call this function in MissionComplete method we created earlier after the if condition.

OnMissionCompleted(InstigatorPawn);

This function is exposed to BP. There will be no implementation in cpp.

Now we need to call this function from ExtractionZone.cpp.

To do that add the following code in the if condition of the HandleOverlap function.

AFPSObjectiveGameMode\* GM = Cast<AFPSObjectiveGameMode>(GetWorld()->GetAuthGameMode());

if (GM)

{

GM->MissionComplete(MyPawn);

}

Include needed for GameMode

#include "FPSObjectiveGameMode.h"

Final script in handle overlap function will be like the following.

void AFPSExtraction::HandleOverlap(UPrimitiveComponent \* OverlappedComponent, AActor \* OtherActor, UPrimitiveComponent \* OtherComp,

int32 OtherBodyIndex, bool bFromSweep, const FHitResult & SweepResult)

{

UE\_LOG(LogTemp, Warning, TEXT("Overlapped with extraction zone"));

AFPSObjectiveCharacter\* MyPawn = Cast<AFPSObjectiveCharacter>(OtherActor);

if (MyPawn != nullptr && MyPawn->bIsCarryingObjective)

{

AFPSObjectiveGameMode\* GM = Cast<AFPSObjectiveGameMode>(GetWorld()->GetAuthGameMode());

if (GM)

{

GM->MissionComplete(MyPawn);

}

}

}

Now the game should be working.

We need a Mission Complete message to display.

**Screen Message using Widget**

Now lets show some on screen messages for extraction zone.

We will give that in the BP of GameMode.

So go to BPGameMode event graph.

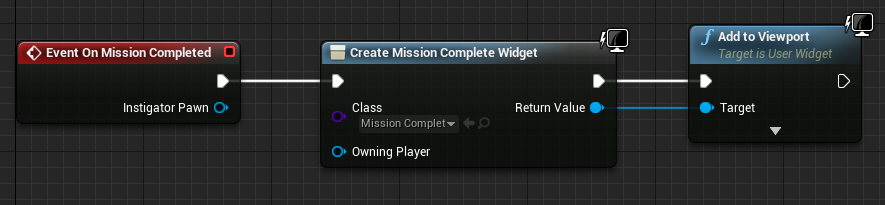
Right click and search OnMissionCompleted. (the function that we created in GameMode.h file)

Create widget named "Game Over"

On design tab just add a text that says "Mission Completed"



Go to GameMode BP and add blueprint to create widget and add to view port.



Now when you extract we should be getting the widget.

**Adding Sound**

Now we need to give a sound when player enters extraction zone without objective.

So go to extractionZone header file and add the USoundBase pointer like the following

UPROPERTY(EditDefaultsOnly, Category = "Sounds")

USoundBase\* ObjectiveMissingSound;

Now go to the Extraction Zone cpp file and add another if statement to return if pointer is nullptr. as follows

if (MyPawn == nullptr)

{

return;

}

Add an else to current if statement to play sound.

add the following to the else part

UGameplayStatics::PlaySound2D(this, ObjectiveMissingSound);

final if statement will look like this

if (MyPawn->bIsCarryingObjective)

{

ACPPLesson02GameMode\* GM = Cast<ACPPLesson02GameMode>(GetWorld()->GetAuthGameMode());

if (GM)

{

GM->CompleteMission(MyPawn);

}

}

else

{

UGameplayStatics::PlaySound2D(this, ObjectiveMissingSound, 0.2f, 0.2f);

}

now go to the FPSExtractionZone BP and add a sound in the defaults. (this was made by the UPROPERTY we added in header file of extraction zone.)

Game should play the sound when entering extraction zone without carrying objective.

(Make sure we have added a blueprint version of the extraction zone in the level.)

Now another game play logic. That when game finishes camera moves from first person view to third person view.

show a prototype in BP

Add a camera that focus on entire level and convert it to a bp.

In GameMode on Mission complete event add a sequence node add the bp as in the image in folder.

Prototyping is done now lets try that in cpp

First we need to get the playercontroller. (looking at bp and decided)

To get player controller do the following

APlayerController\* PC = Cast<APlayerController>(InstigatorPawn->GetController());

Now we need to SetViewTargetWithBlend to player controller. (similar function used in bp)

So do as following

PC->SetViewTargetWithBlend(NewViewTarget, 5.0f, EViewTargetBlendFunction::VTBlend\_Cubic);

' NewViewTarget ' is an actor variable which is not declared at this point.

So lets declare it as a pointer to actor before the if as follows

AActor\* NewViewTarget;

To get a value in to the NewViewTarget in the previous statement we need to get all actors of class "Spectatorviewpoint" (class we created in editor with camera in it.) And get its first value just like we did in BP.

Following is the function.

UGameplayStatics::GetAllActorsOfClass(arg1, arg2, arg3);

Check the documentation (https://docs.unrealengine.com/en-US/API/Runtime/Engine/Kismet/UGameplayStatics/GetAllActorsOfClass/index.html)

For that first we need to create a variable called SpectatingCameraClass in header file.

We also need to expose it to BP as well.

For that do the following.

UPROPERTY(EditDefaultsOnly, Category = "Spectating")

TSubclassOf<AActor> SpectatingCameraClass;

We expose it to variable so that we can select a class in the editor.

Now we need to create an array of actors that will be returned in the function UGameplayStatics::GetAllActorsOfClass

For that do the following in cpp file.

TArray<AActor\*> ReturnedActors;

We created and array and now we can use these variables to call the function "GetAllActorsOfClass"

Following is the final function call

UGameplayStatics::GetAllActorsOfClass(this, SpectatingCameraClass, ReturnedActors);

ReturnedActors is an array and the first element will have the class with camera which is going to be our target ("NewViewTarget" variable).

NewViewTarget = ReturnedActors[0];

The final finished code in MissionComplete function is as follows.

void ACPPLesson02GameMode::CompleteMission(APawn\* InstigatorPawn)

{

if (InstigatorPawn)

{

InstigatorPawn->DisableInput(nullptr);

AActor\* NewViewTarget;

TArray<AActor\*> ReturnedActors;

UGameplayStatics::GetAllActorsOfClass(this, SpectatingCameraClass, ReturnedActors);

NewViewTarget = ReturnedActors[0];

APlayerController\* PC = Cast<APlayerController>(InstigatorPawn->GetController());

PC->SetViewTargetWithBlend(NewViewTarget, 5.0f, EViewTargetBlendFunction::VTBlend\_Cubic);

}

OnMissionCompleted(InstigatorPawn);

}

Can show the best practice of checking each pointers.

`