UnreaLearning
SUMMER PROJECT 2024
Game Development Club

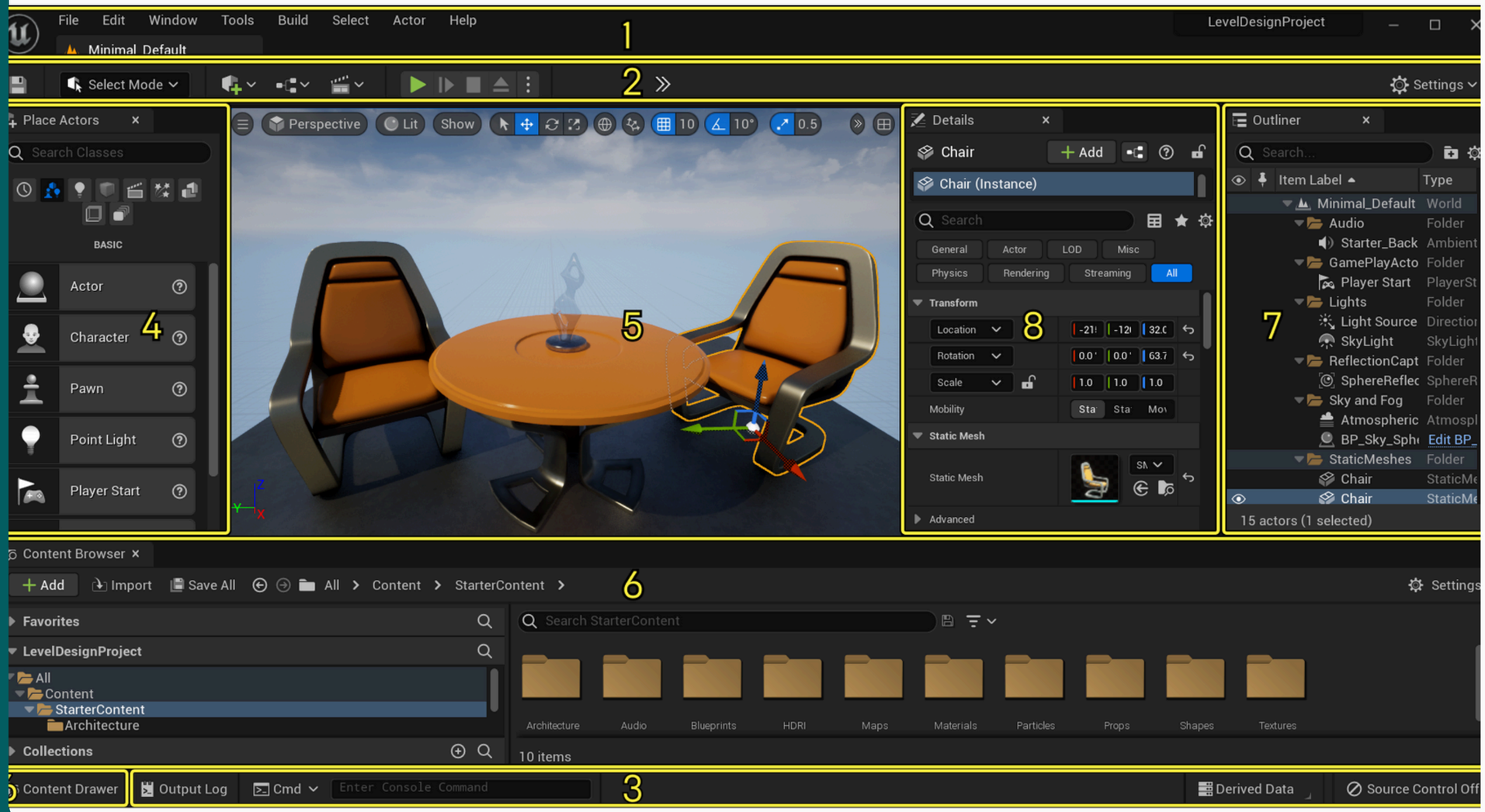
Aim of this Project

Understanding all of the components
that make up a comprehensive
game using Unreal Engine

Module 1

Objects, Event Graph, Blueprint Classes and Importing Assets

UE 5 Interface



1. Tab Bar and Menu Bar
2. Toolbar
3. Buttom Toolbar
4. Place Actor / Modes
5. Viewports
6. Content Browser / Content Drawer
7. Outliner
8. Details

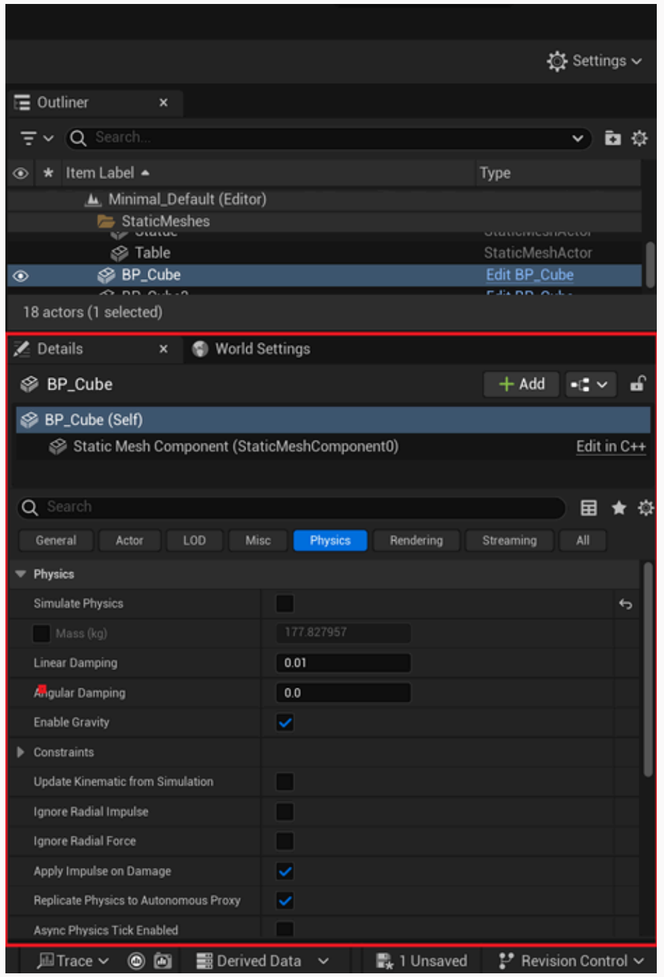
Unreal Objects

In Unreal Engine 5, objects are the basic class that act as building blocks for assets. They contain essential functionality for assets, and almost everything in Unreal Engine inherits functionality from an object.

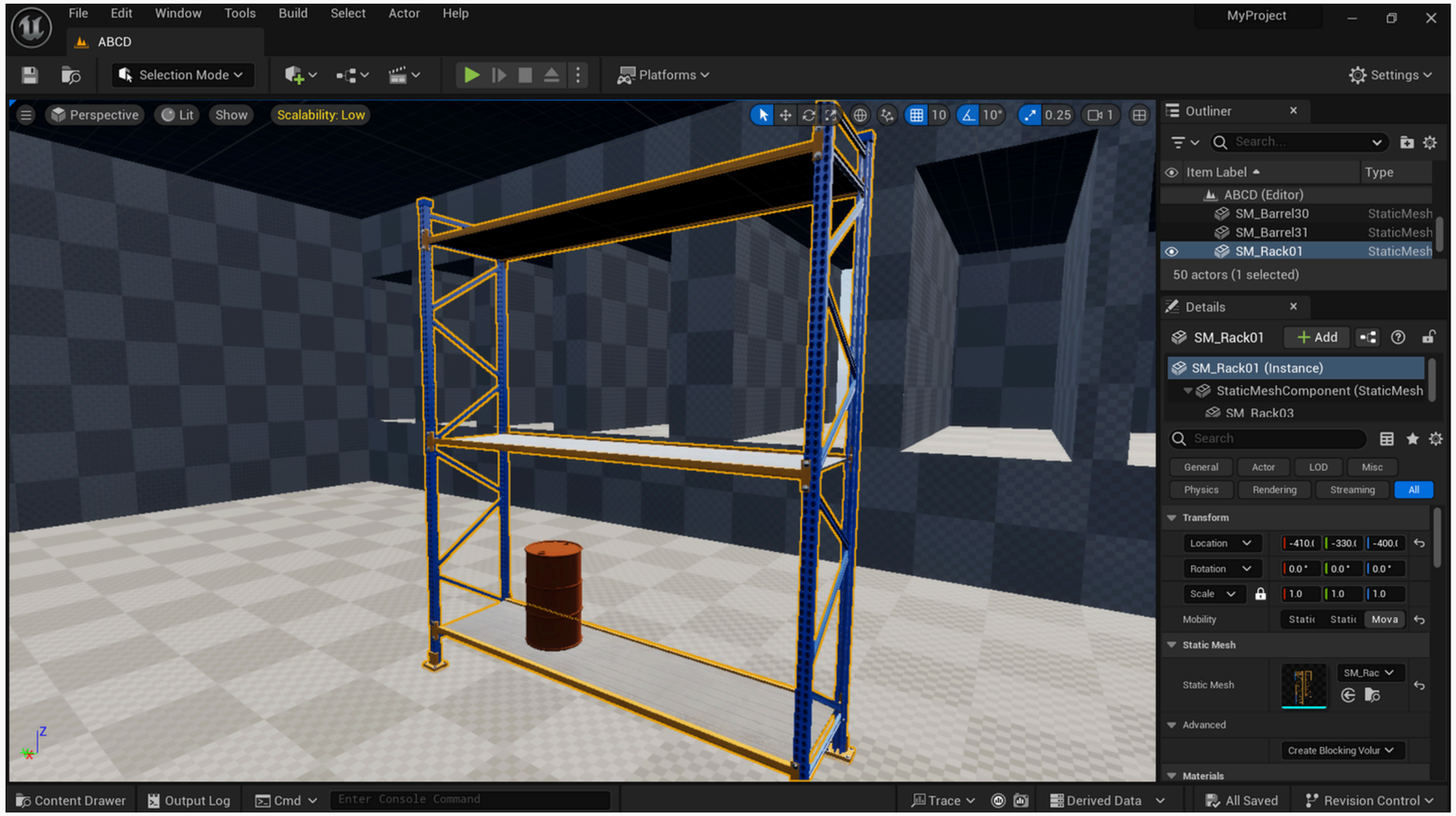
Object Details

When you select an Actor in the Level Viewport, the Details panel will show the settings and properties that affect the Actor you selected.

The details panel gives the option to transform the object, edit the mesh and materials of the object, add physics components to it which is helpful in object motion and also gives other properties to edit the object behaviour.



Objects we used



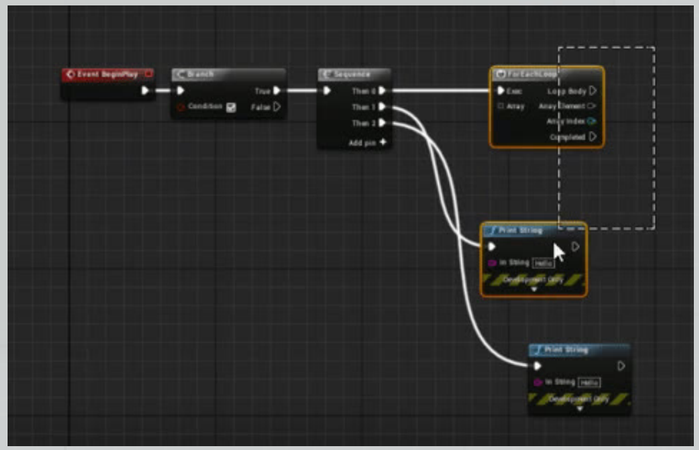
We imported Barrels, Rack and a gun for level design

We used spheres for bullets and cubes for room design.

Apart from this lights were used in the room

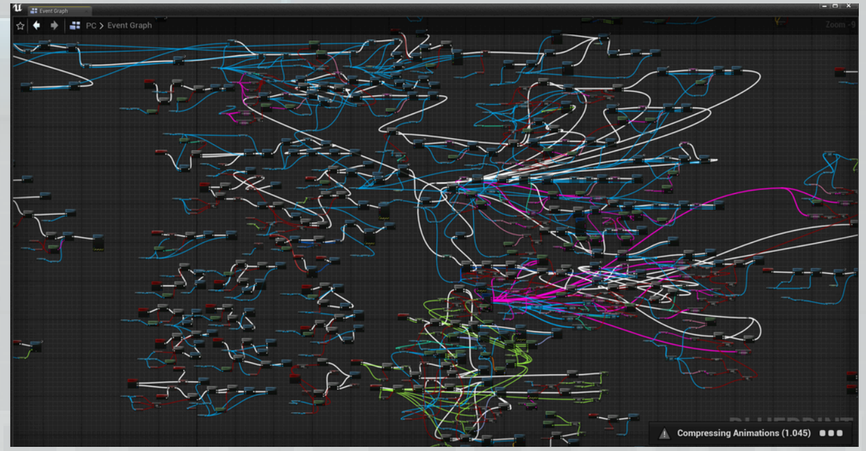
Event Graph

Uses events and function calls to perform actions in response to Blueprint events. The EventGraph of a Level Blueprint contains a node graph that uses events and function calls to perform actions in response to gameplay events.



If you organise them wisely they are clear to see

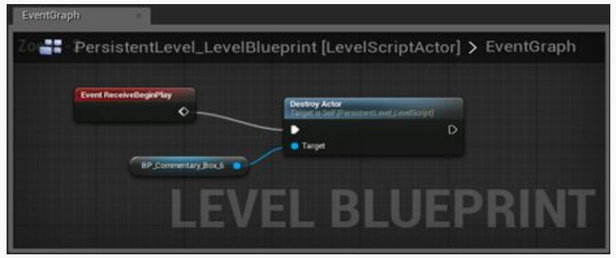
Otherwise Complete chaos



Blueprints Visual Scripting

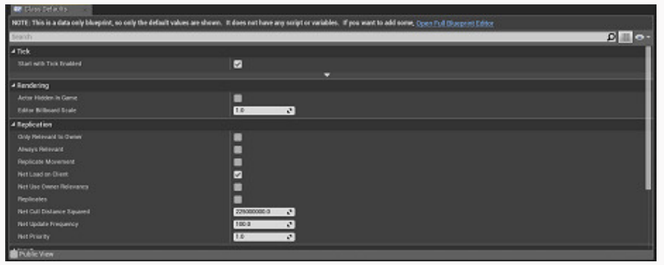
The Blueprint Visual Scripting system in Unreal Engine is a complete gameplay scripting system based on the concept of using a node-based interface to create gameplay elements from within Unreal Editor.

There are different types of Blueprint we have used



Level Blueprint

Blueprint class

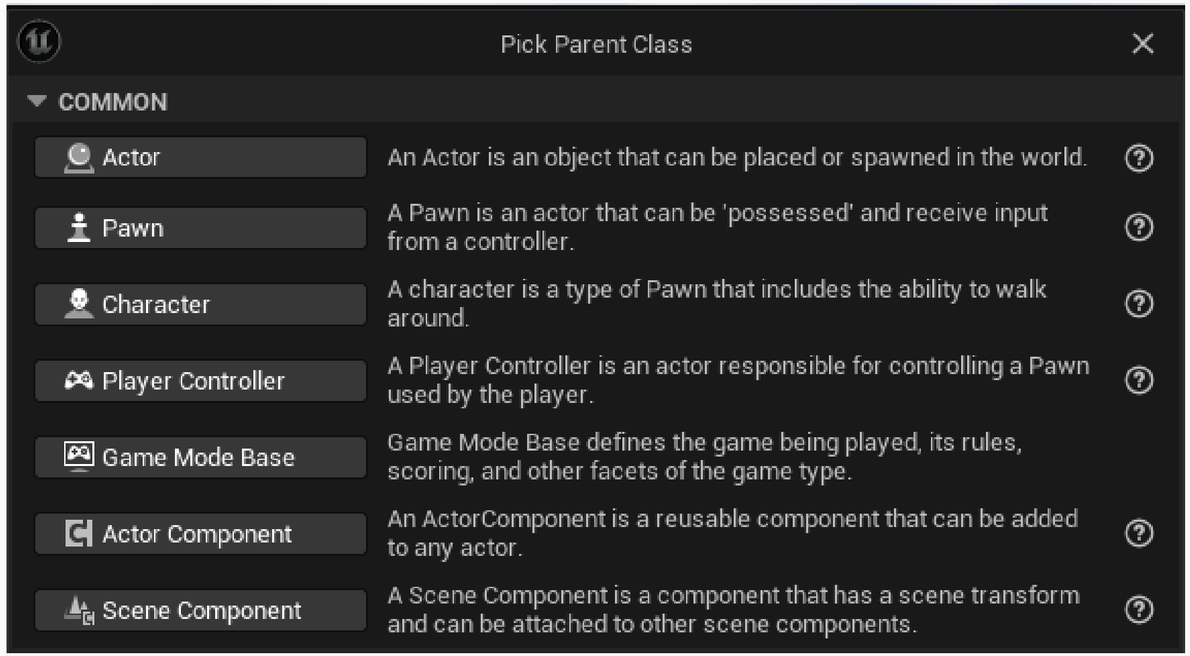


Data-only Blueprint

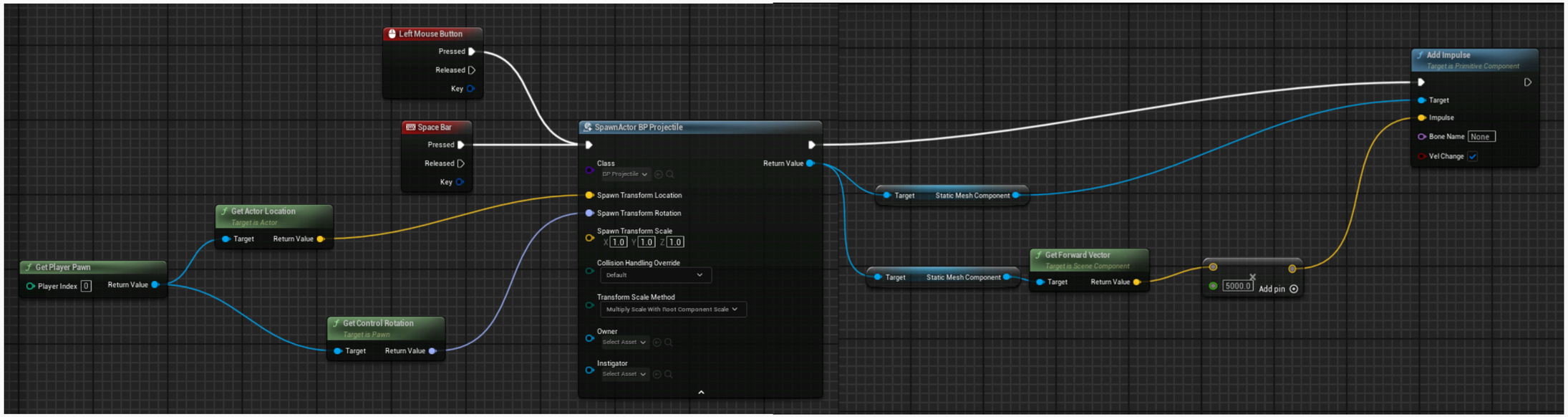
Blueprint Class

A ***Blueprint Class*** , often shortened as ***Blueprint***, is an asset that allows content creators to easily add functionality on top of existing gameplay classes.

Before creating a Blueprint
We need specify its parent class



Our First actual Level Blueprint in Event Graph



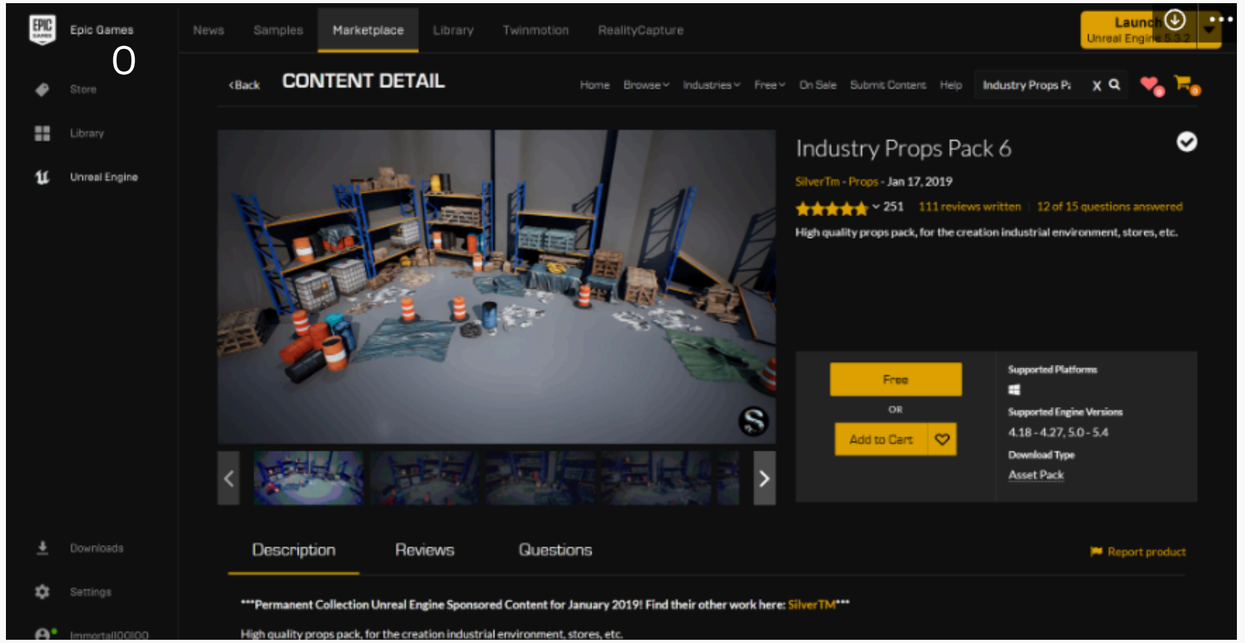
Here we did some spawn and shooting of projectile with the help of some input

Ofcourse after printing some “Hello World”

Assets

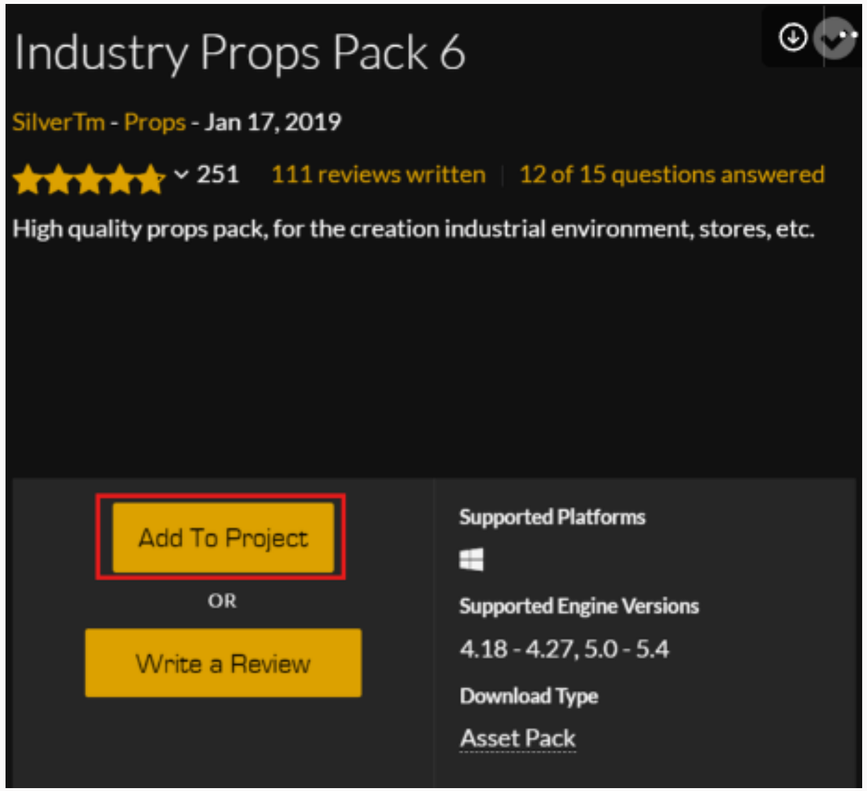
Gaming Assets is everything that can help in the development and visual set-up of the game.

In this module we have used an asset “Industry Props Pack 6” which is available on Epic Games Launcher under Unreal Engine’s Marketplace section..



Your paragraph text

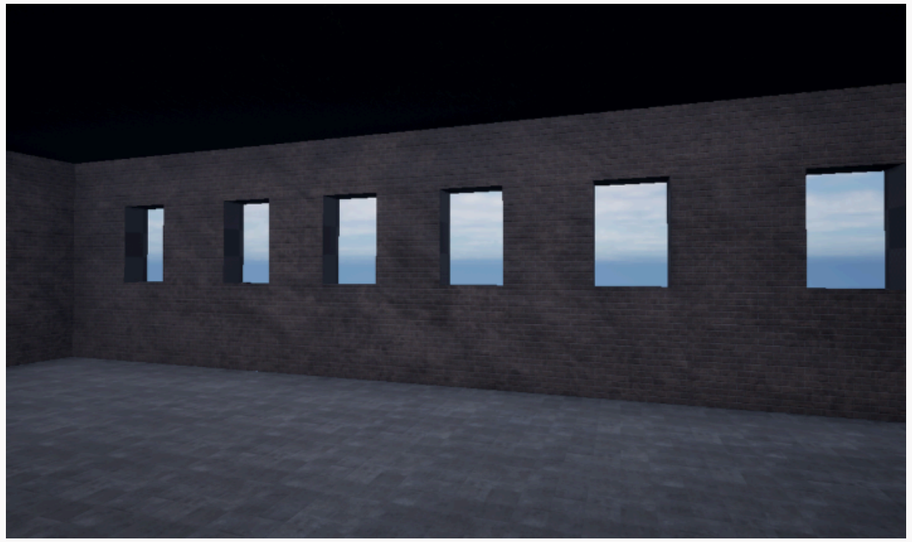
How to import asset in Unreal Game?



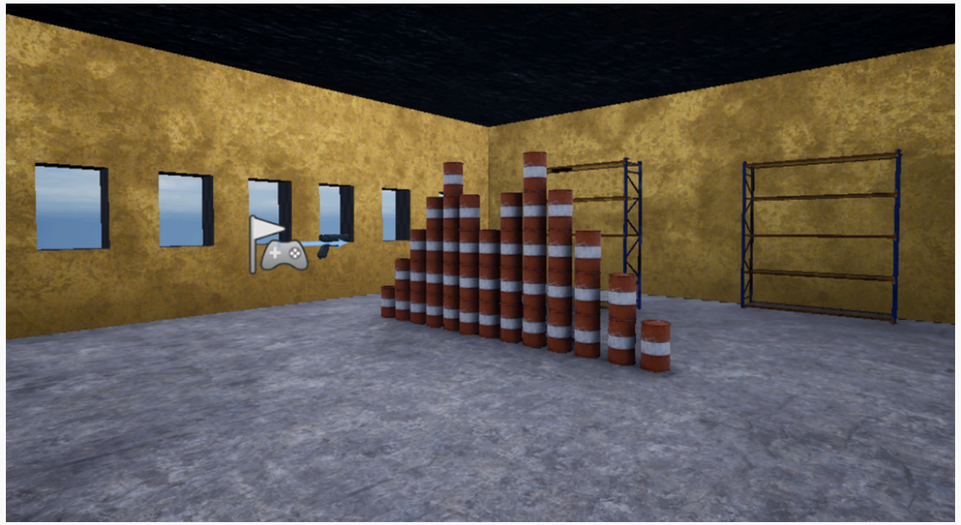
After Clicking on free or buying the asset you can add them to the project you want to add it to via the add to project button.

After choosing the project in which you want to add the assets, open the project in Unreal engine and you can find your assets in its content drawer now, which can be accessed through Ctrl + space.

What we created?



We build a level Barrel Carnage which is a shooting game in which we have to shoot down the wall of barrel.

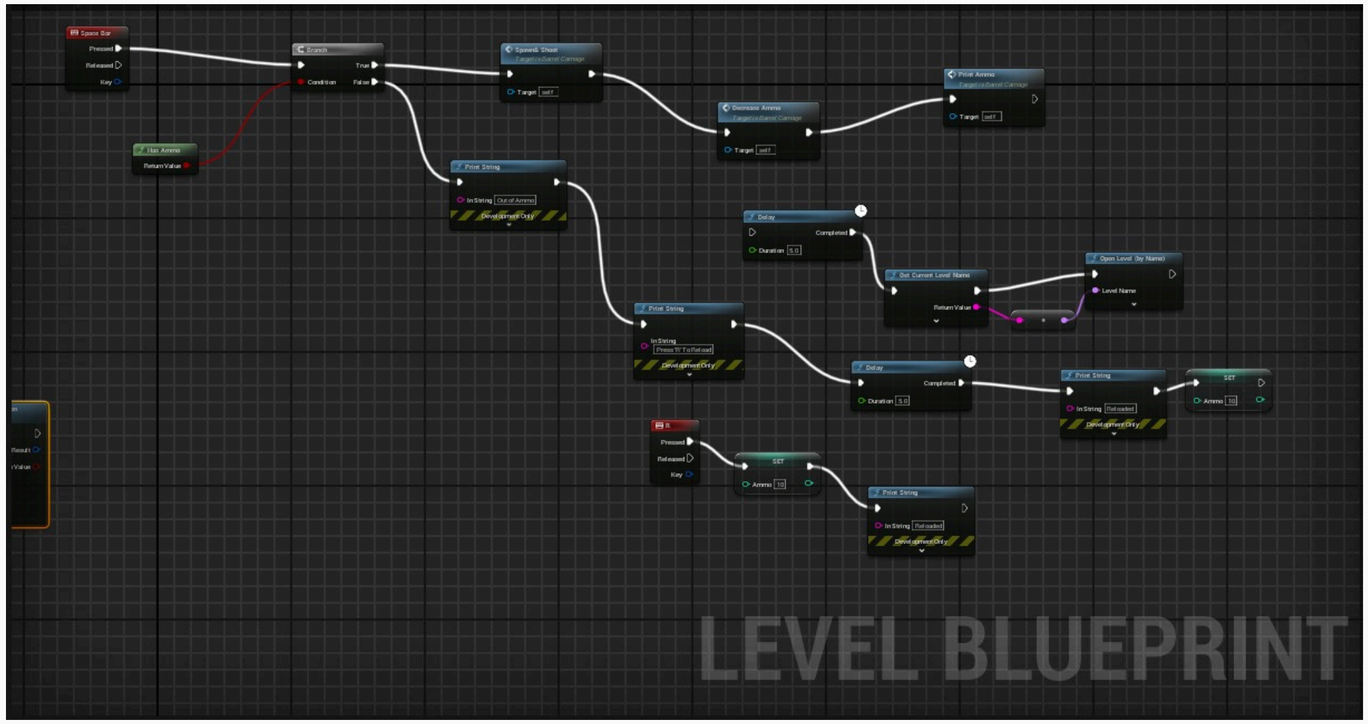


Task 1:

Implement shooting with a gun in First person

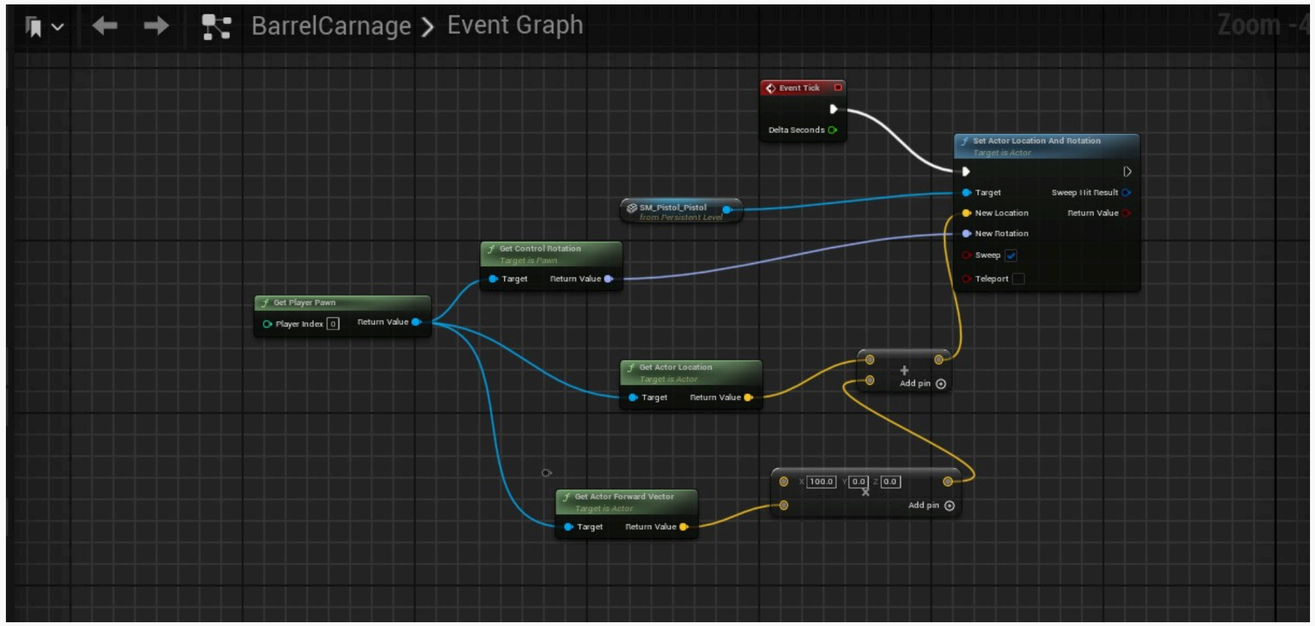
- Add gun object
- Add offset in the shooting mechanic implemented before to shoot from the muzzle of the gun.
- Move the gun with the Player
- Make reload mechanic
- Add timer to restart game after a certain amount of time

Solution



This is a blueprint graph for Ammo reload and timer.

Solution



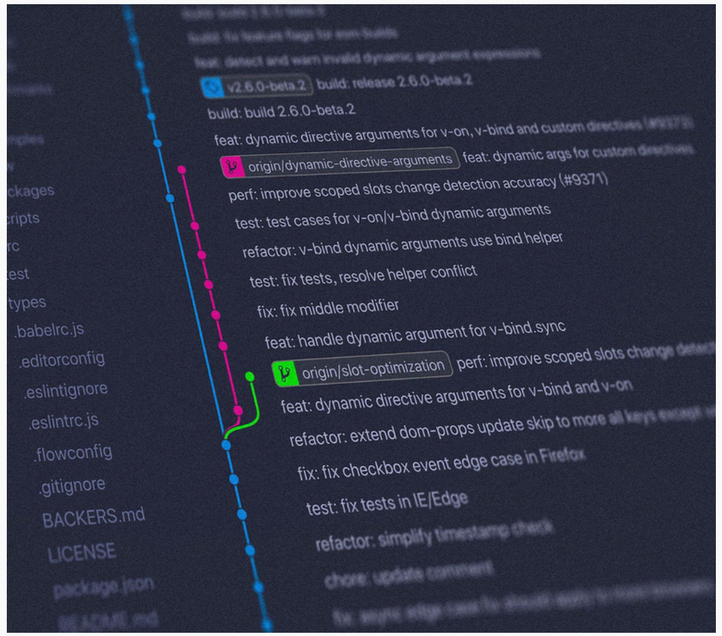
This is for the Gun movement and setting offset for bullets.

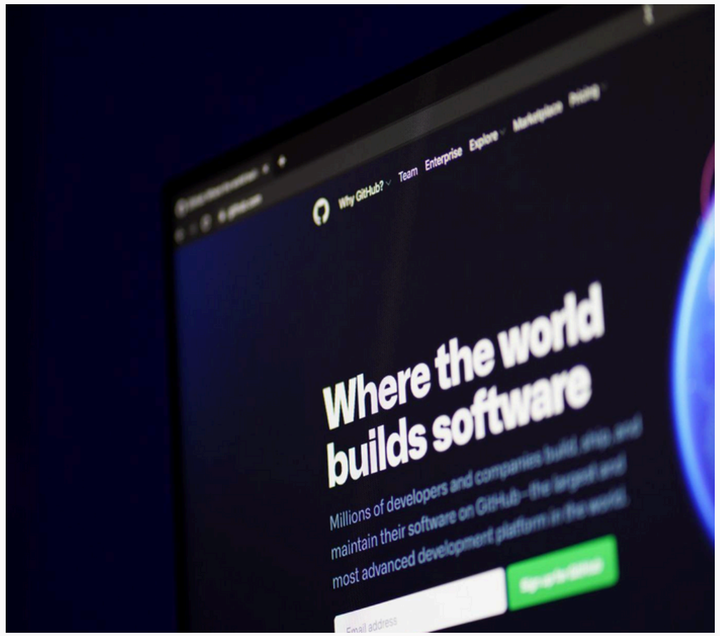
Script structure

Module 2

Using Github, Motion and rotation using C++ scripts, Logging in Unreal and Level design

GitHub provides a platform for developers to collaborate on projects, track changes, and manage source code repositories. It allows multiple developers to work on the same project simultaneously, making it easier to manage and merge code changes.





Setting up Unreal Project on GitHub

The .gitignore file in the project template ignores the “Saved” folder by default. But it also contains autosave files, and is hence necessary. Therefore a small change in the file is needed as shown. Rest of the things are the same

```
53    Build/*
54
55    # Whitelist PakBlacklist-<BuildConfiguration>.txt files
56    !Build/*/
57    Build/*/**
58    !Build/*/PakBlacklist*.txt
59
60    # Don't ignore icon files in Build
61    !Build/**/*.ico
62
63    # Built data for maps
64    *_BuiltData.uasset
65
66    # Configuration files generated by the Editor
67    Saved/*
68
69    # Compiled source files for the engine to use
70    Intermediate/*
71    Plugins/*/Intermediate/*
```

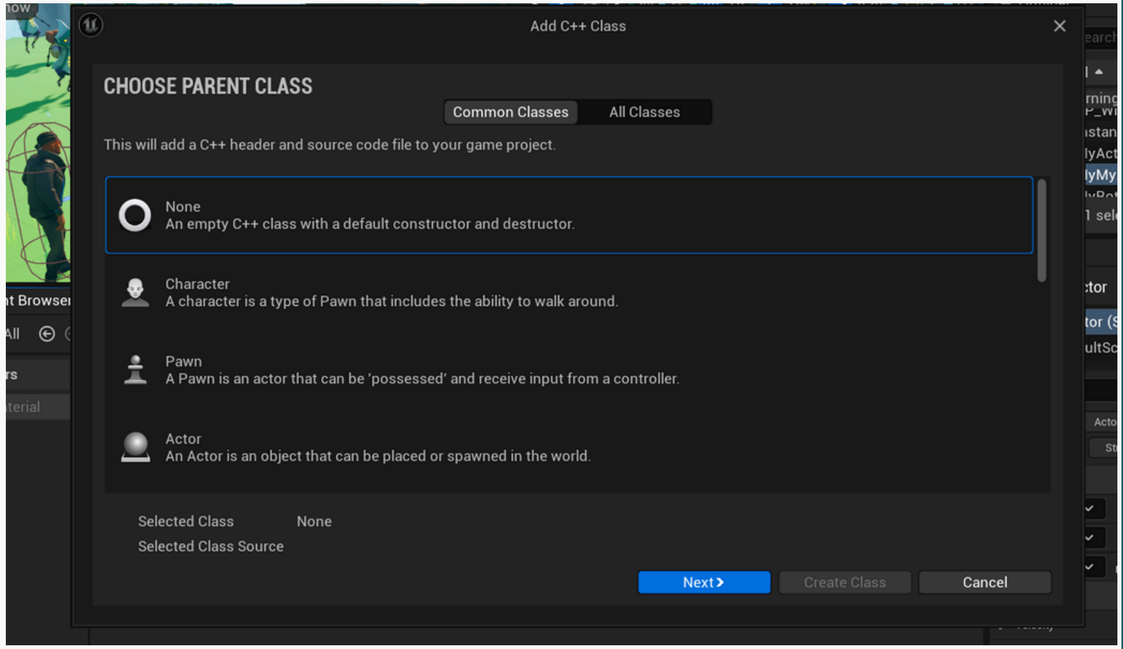
```
53    Build/*
54
55    # Whitelist PakBlacklist-<BuildConfiguration>.txt files
56    !Build/*/
57    Build/*/**
58    !Build/*/PakBlacklist*.txt
59
60    # Don't ignore icon files in Build
61    !Build/**/*.ico
62
63    # Built data for maps
64    *_BuiltData.uasset
65
66    # Configuration files generated by the Editor
67    Saved/Log/*
68
69    # Compiled source files for the engine to use
70    Intermediate/*
71    Plugins/*/Intermediate/*
```

Scripting in Unreal

Scripting in unreal engine is done using C++ classes.

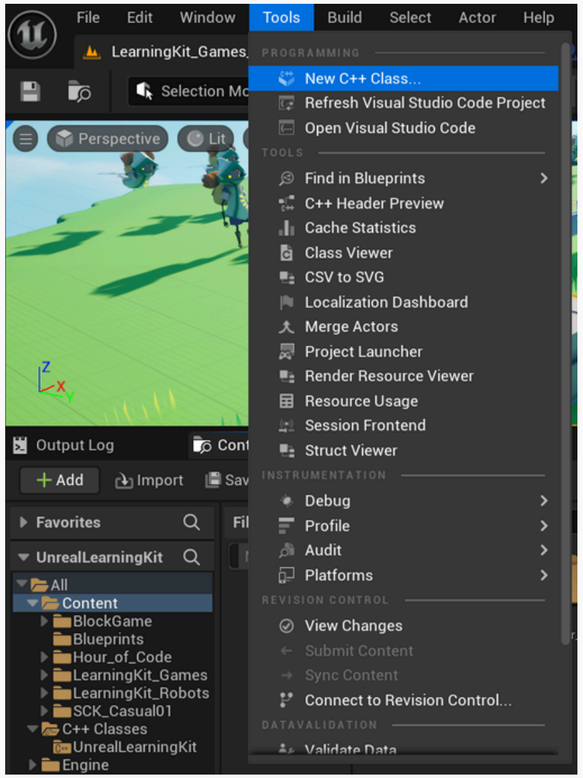
Key Concepts:

- **Classes and Objects:** Define the structure and behavior of game elements.
- **UObjects and AActors:** Core classes in Unreal Engine. UObject is the base class for all UE objects, while AActor is the base class for all actors that can be placed in a level.
- **Inheritance:** Allows creating new classes based on existing ones.

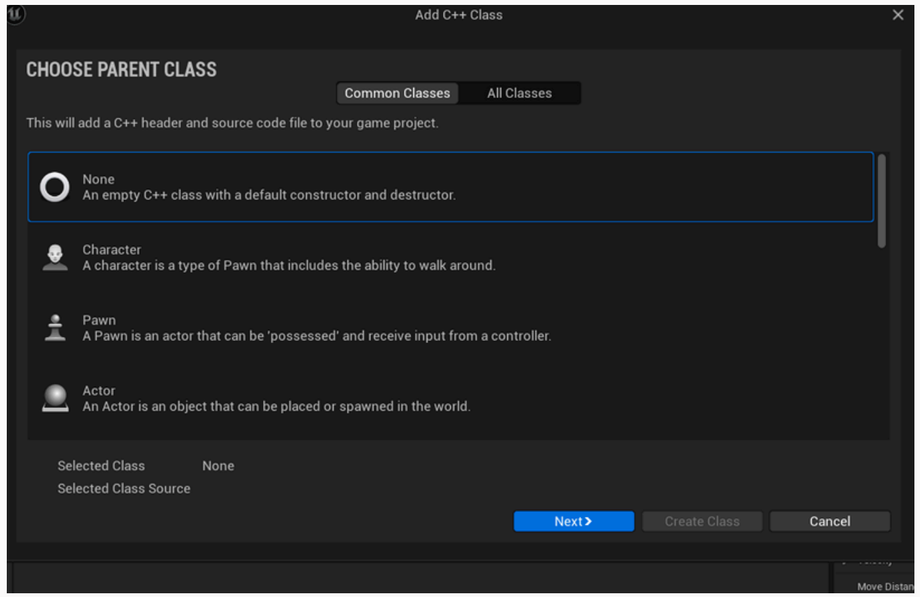


How to create scripts?

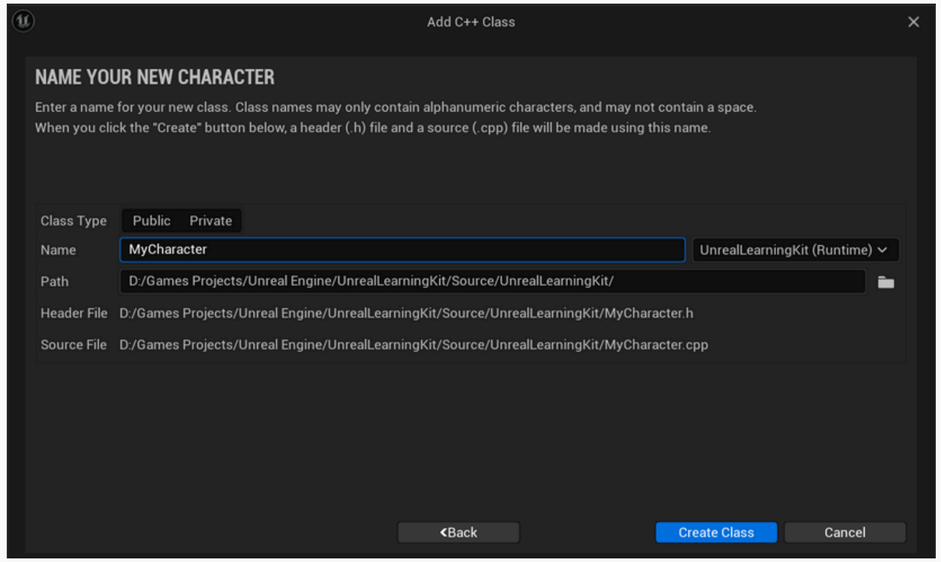
1.



2.



3.



Movement using script

(Oscillating movement of an actor)

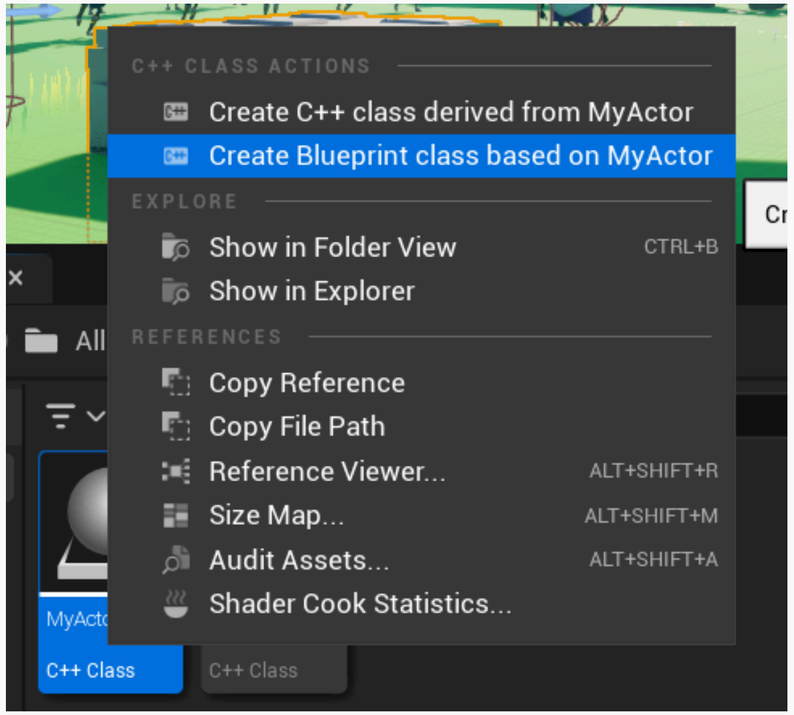
Variables and properties in header file

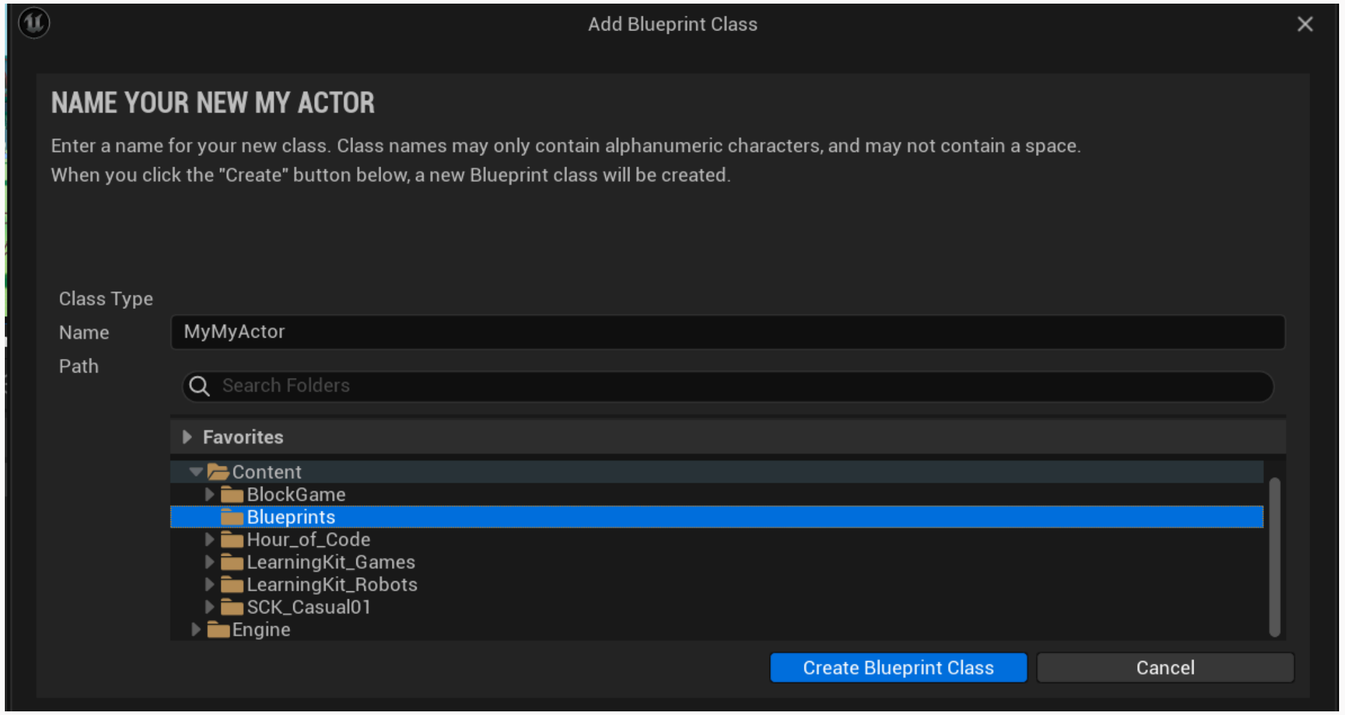
```
class UNREALLEARNINGKIT_API AMyActor : public AActor
{
    GENERATED_BODY()

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

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;
public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

    UPROPERTY(EditAnywhere)
    FVector Velocity = FVector(0,0,0);

    UPROPERTY(EditAnywhere)
    float MoveDistance = 500;
    UPROPERTY(VisibleAnywhere)
    float distanceMoved;
    FVector startLocation;
};
```

Game Logic in the cpp file

```
// Called when the game starts or when spawned
void AMyActor::BeginPlay()
{
    Super::BeginPlay();
    startLocation = GetActorLocation();
}

// Called every frame
void AMyActor::Tick(float DeltaTime)
{
    FVector CurrentLocation = FVector(0,0,0);
    Super::Tick(DeltaTime);
    CurrentLocation = GetActorLocation();
    CurrentLocation += Velocity * DeltaTime;
    SetActorLocation(CurrentLocation);

    distanceMoved = FVector::Dist(startLocation, CurrentLocation);

    if(distanceMoved >= MoveDistance)
    {
        FVector MoveDirection = Velocity.GetSafeNormal();
        startLocation += MoveDistance * MoveDirection;
        SetActorLocation(startLocation);
        Velocity *= -1;
    }
}
```

Create Blueprint actor from c++ class

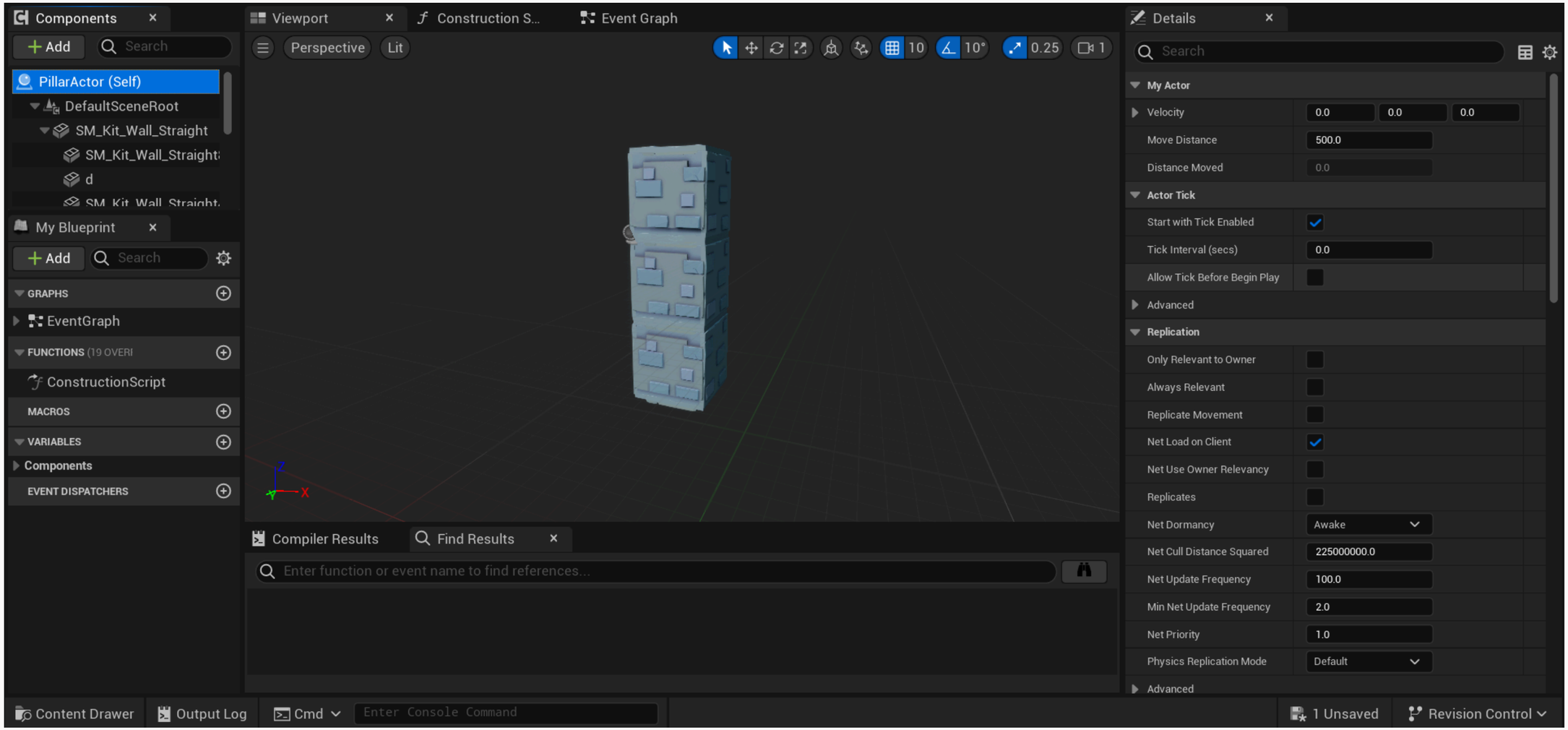




Custom mesh can be added to the blueprint actor to make an object.

This object will oscillate around its mean position.

Created Blueprint actor with custom mesh



Can be dragged and drop in the scene to be used as an obstacle

Task 2

- Add a Rotator.
- Design the level

Solutions

Variables and properties in header file

```h
// Fill out your copyright notice in the Description page of

#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Rotor.generated.h"

UCLASS()
class UNREALLEARNINGKIT_API ARotor : public AActor
{
    GENERATED_BODY()

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

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;
    UPROPERTY(VisibleAnywhere)
    FRotator RotationSpeed = FRotator(0,0,0);
    UPROPERTY(EditAnywhere)
    float MoveDistance = 500;
};

```

Game Logic in the cpp file

```cpp
#include "Rotor.h"

// Sets default values
ARotor::ARotor()
{
    // Set this actor to call Tick() every frame.  Yo
    PrimaryActorTick.bCanEverTick = true;

}

// Called when the game starts or when spawned
void ARotor::BeginPlay()
{
    Super::BeginPlay();

}

// Called every frame
void ARotor::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
    FRotator NewRotation= RotationSpeed * DeltaTime;
    AddActorLocalRotation(NewRotation);
}
```

Module 3

Creating an environment using assets available online, detecting collisions, grabbing objects etc.

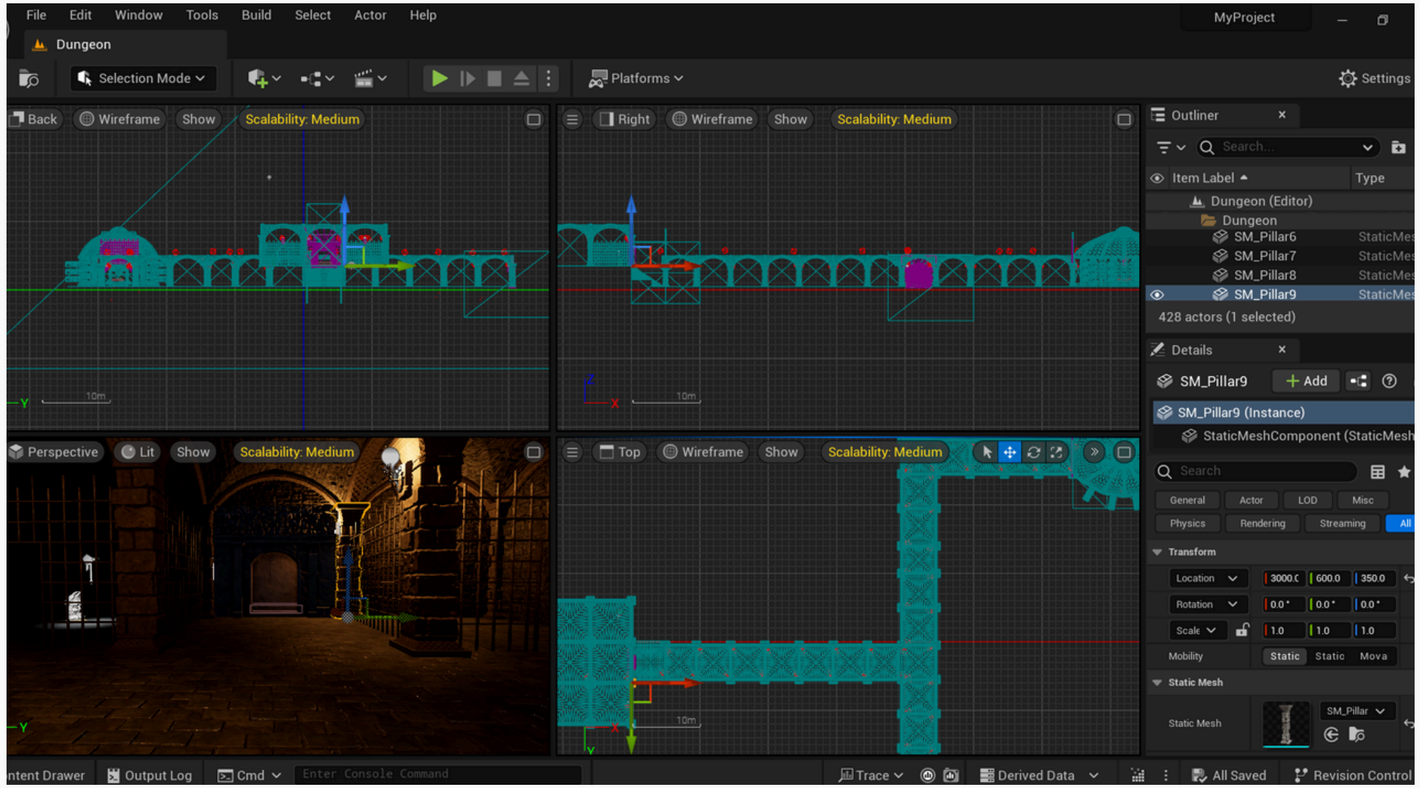
Creating game environment

Unlike our previous modules, we created the game environment using the assets available online which gave us the basic idea of viewport types, various kinds of lights available to us, player pawn and few other components readily available in the template

Viewport design of the level

We used orthographic views of the viewport in order to create the environment of the game.

We have used point lights available in unreal on the torches to create the gloomy environment

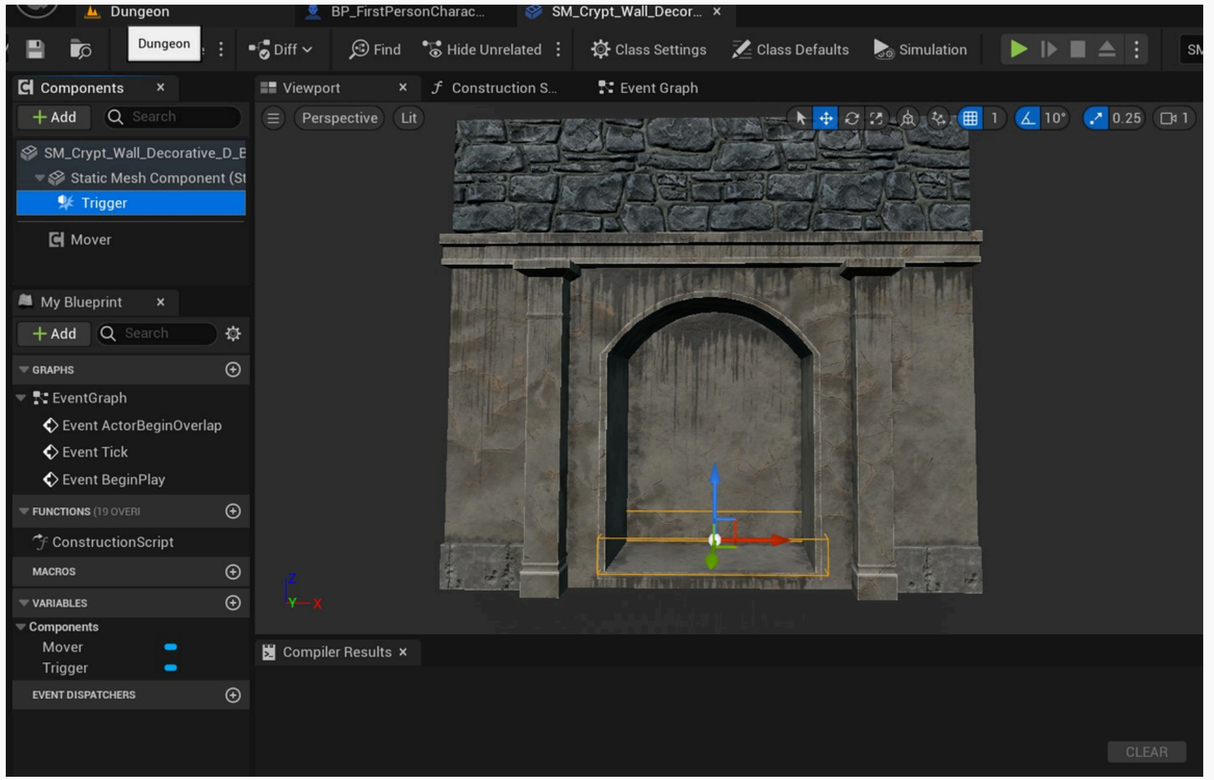


Detecting Collisions

We used to colliders(specifically box collider) to detect object on the door using C++ scripts and accessing the tags on the object

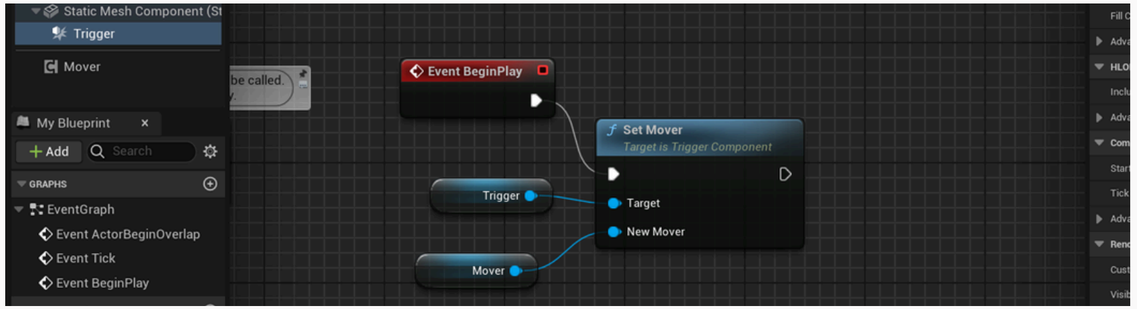
Here, we use the box colliders to detect any relevant object in contact with the door and when the object is detected the game checks the tag of the object that triggered the collider.

BASIC LOGIC-
If the object detected by the collider on the door has the specified tag the door opens



Mechanism of the Door

Once the tigger component detects the object the mover component moves the door upwards.
The mover component and the trigger component are connected through the blueprint



Grabber.h | MoverComponent.cpp 2 | MoverComponent.h 4 | TriggerComponent.h 5 | TriggerComponent.cpp 2 | Release Notes: 1.92.0

MyProject > Source > MyProject > Public > MoverComponent.h > ...

```cpp
#include "CoreMinimal.h"
#include "Components/ActorComponent.h"
#include "MoverComponent.generated.h"

UCLASS( ClassGroup=(Custom), meta=(BlueprintSpawnableComponent) )
class MYPROJECT_API UMoverComponent : public UActorComponent
{
    GENERATED_BODY()

public:
    // Sets default values for this component's properties
    UMoverComponent();

protected:
    // Called when the game starts
    virtual void BeginPlay() override;

public:
    // Called every frame
    virtual void TickComponent(float DeltaTime, ELevelTick TickType, FActorComponentTickFunction* ThisTickFunction) override;

    UPROPERTY(EditAnywhere)
    bool shouldMove = false;

private:
    UPROPERTY(EditAnywhere)
    FVector moveOffset;

    UPROPERTY(EditAnywhere)
    float moveTime = 4.0f;

    FVector OriginalLocation;
    FVector TargetLocation;
};
```

MyProject > Source > MyProject > Public > TriggerComponent.h > UTriggerComponent

```cpp
#pragma once

#include "CoreMinimal.h"
#include "Components/BoxComponent.h"
#include "MoverComponent.h"
#include "TriggerComponent.generated.h"


/**
 *
 */
UCLASS(ClassGroup=(Custom), meta=(BlueprintSpawnableComponent))
class MYPROJECT_API UTriggerComponent : public UBoxComponent
{
    GENERATED_BODY()

    public:
    UTriggerComponent();

    protected:
    // Called when the game starts
    virtual void BeginPlay() override;


    public:
    // Called every frame
    virtual void TickComponent(float DeltaTime, ELevelTick TickType, FActorComponentTickFunction* ThisTickFunction) override;


    UMoverComponent* mover;

    UFUNCTION(BlueprintCallable)
    void SetMover(UMoverComponent* newMover);

};
```

Grabbing an object

For the player to unlock the doors, it is essential the player is able to grab the tagged object and move it around.
We acheive this by using physics handle along with collision channels to grab the objects that are tagged as such and are in the reach of the player.
The Physics Handle Component is an object for "grabbing" and moving physics objects around while allowing the object you are grabbing to continue to use physics.
We also add grabber component to the first person character.

Grabber.cpp

Terminal Help MyProject (Workspace)

Grabber.h | MoverComponent.cpp | MoverComponent.h | TriggerComponent.h | TriggerComponent.cpp | Release Notes: 1.9

Project > Source > MyProject > Public > Grabber.cpp > Release()

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


#include "Grabber.h"
#include "Engine/World.h"
#include "DrawDebugHelpers.h"

// 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 featu
    // off to improve performance if you don't need them.
    PrimaryComponentTick.bCanEverTick = true;

    // ...
}


// Called when the game starts
void UGrabber::BeginPlay()
{
    Super::BeginPlay();
    // ...

}


// Called every frame
void UGrabber::TickComponent(float DeltaTime, ELevelTick TickType, FActorComponentTickFunction* ThisTickFunction)
{
    Super::TickComponent(DeltaTime, TickType, ThisTickFunction);
    physicsHandle = GetOwner()->FindComponentByClass<UPhysicsHandleComponent>();
    if(physicsHandle!= nullptr && physicsHandle->GetGrabbedComponent()!=nullptr){
        physicsHandle->GetGrabbedComponent()->WakeAllRigidBodies();
        FVector targetLocation = GetComponentLocation() + GetForwardVector() * grabbedRadius;
        physicsHandle->SetTargetLocationAndRotation(targetLocation,GetComponentRotation());
    }
}
```

abber.h | MoverComponent.cpp | MoverComponent.h | TriggerComponent.h | TriggerComponent.cpp | Releas

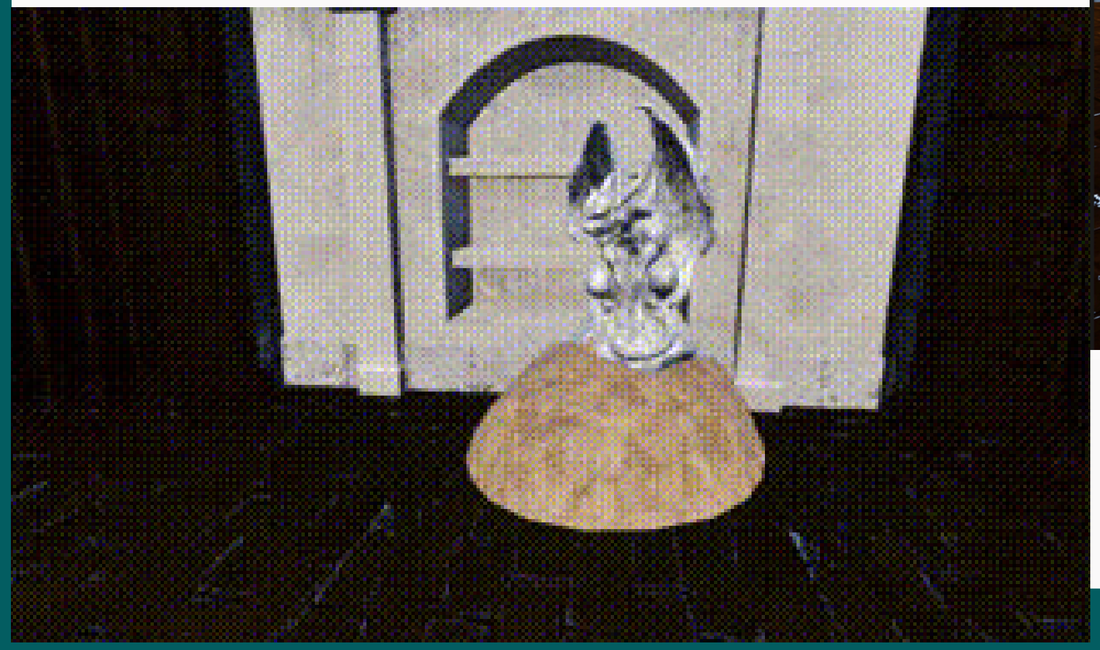
oject > Source > MyProject > Public > Grabber.cpp > Release()

```cpp
}
void UGrabber::Grab(){
    FHitResult hits;
    bool hasHit = GetGrabbableInReach(hits);
    //physicsHandle = GetOwner()->FindComponentByClass<UPhysicsHandleComponent>();
    if(hasHit){
        physicsHandle->GrabComponentAtLocationWithRotation(
            hits.GetComponent(),
            NAME_None,
            hits.ImpactPoint,
            GetComponentRotation()
        );
        UE_LOG(LogTemp,Display,TEXT("Object Grabbed : %s"),*hits.GetActor()->GetActorNameOrLabel());
        hits.GetActor()->Tags.Add(FName("Grabbed"));
    }
    else UE_LOG(LogTemp,Display,TEXT("No object hit"));
}


void UGrabber::Release(){
    if(physicsHandle != nullptr && physicsHandle->GetGrabbedComponent()!=nullptr){
        physicsHandle->GetGrabbedComponent()->GetOwner()->Tags.Remove("Grabbed");
        physicsHandle->ReleaseComponent();
        UE_LOG(LogTemp,Display,TEXT("Object Released"));
    }
}

bool UGrabber::GetGrabbableInReach(FHitResult& OutHits){
    UE_LOG(LogTemp,Display,TEXT("Inside Grab Function"));
    UWorld* world = GetWorld();
    FVector start = GetComponentLocation();
    FVector end = start + GetForwardVector()*grabReach;
    DrawDebugLine(world,start,end,FColor::Green);

    FCollisionShape sphere = FCollisionShape::MakeSphere(grabSphereRadius);
    return world->SweepSingleByChannel(OutHits,start, end,FQuat::Identity,ECC_GameTraceChannel2,sphere);
}
```

The Grabber uses actor tag on the object to grab and release the object

The Grab function in the Grabber.cpp checks for grabbable objects within reach. If an object is found, it grabs the component and logs the event.
The Release function releases the currently grabbed component and logs the event.
The GetGrabbableInReach function uses a sphere sweep to detect objects within reach and logs the event.

Grabbing an object



Task

For the initial door we had to connect trigger component and mover component on the same door. Whereas for the other door we ought to associate the mover component attached on the door with the trigger component attached on the statue stand

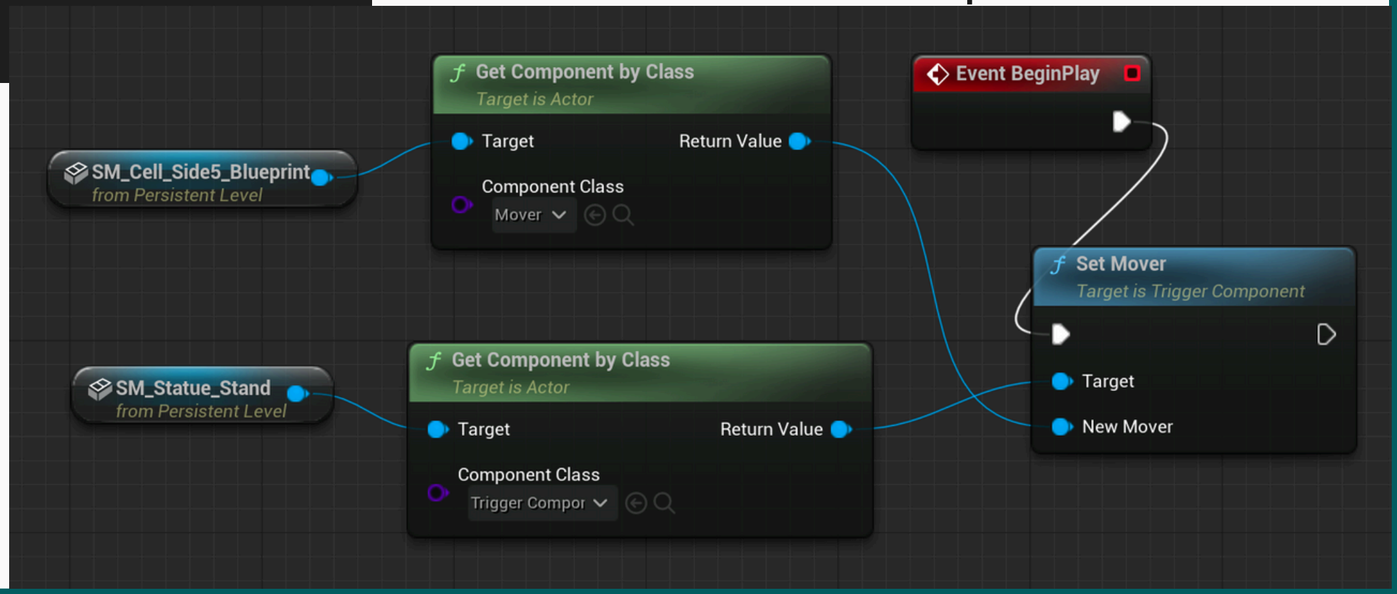
Solutions

```cpp
void UTriggerComponent::SetMover(UMover* newMover)
{
    mover = newMover;
    originalLocation = mover->originalLocation;
    targetLocation = mover->targetLocation;
    UE_LOG(LogTemp, Display, TEXT("Mover Set"));

}
```

TriggerComponent.cpp

Pairing the trigger collider on the statue stand with the movable wall

Level Blueprint



Solutions

TriggerComponent.cpp

```cpp
void UTriggerComponent::TickComponent(float DeltaTime, ELevelTick TickType, FActorComponentTickFunction* ThisTickFunction)
{
    Super::TickComponent(DeltaTime, TickType, ThisTickFunction);
    TArray<AActor*> actors;
    GetOverlappingActors(actors);
    int n = actors.Num();
    if(n > 0)
    {
        for(AActor* actor : actors)
        {
            FString name = actor->GetActorNameOrLabel();
            UE_LOG(LogTemp, Display, TEXT("Overlapping with: %s"), *name);
            if(actor->ActorHasTag(AcceptableTag) && !actor->ActorHasTag("Grabbed"))
            {
                if(GetOwner()->ActorHasTag("Wall2")){
                    mover->originalLocation = targetLocation;
                    mover->targetLocation = originalLocation;
                }
                else
                {
                    mover->shouldMove = true;
                    UPrimitiveComponent* component = Cast<UPrimitiveComponent>(actor->GetRootComponent());
                    if(component != nullptr) component->SetSimulatePhysics(false);
                    actor->AttachToComponent(this, FAttachmentTransformRules::KeepWorldTransform);
                }
            }
            else{
                mover->originalLocation = originalLocation;
                mover->targetLocation = targetLocation;
            }
        }
    }
    else{
                mover->originalLocation = originalLocation;
                mover->targetLocation = targetLocation;
    }
```

Blocks of code added for the second wall case given in the task

Solutions

Slight modification in Mover.cpp for the second movable wall as given in the task

```
// Called every frame
void UMover::TickComponent(float DeltaTime, ELevelTick TickType, FActorComponentTickFunction* ThisTickFunction)
{
    Super::TickComponent(DeltaTime, TickType, ThisTickFunction);
    if((shouldMove || GetOwner()->ActorHasTag("Wall2"))){
    AActor* owner = GetOwner();
    FVector currentLocation = owner->GetActorLocation();

    float speed = moveOffset.Size()/moveTime;

    FVector newLocation = FMath::VInterpConstantTo(currentLocation, targetLocation, DeltaTime, speed);

    owner->SetActorLocation(newLocation);
    }
}
```

Mover.cpp

Solutions

Working Example of
the task

Module 4

Player Movement, Turret Rotation and Aiming of tank, Enemy Guns ,Shooting mechanism

Turret Rotation

Turret rotation where a turret can detect, follow, and aim at moving targets within a game. This is used for defense systems, automated weapons, or even in AI-controlled units that need to track and attack enemies.

Scripts For Turret Rotation

```h
UCLASS()
class TOONTANKS_API ABase : public APawn
{
    GENERATED_BODY()

public:
    // Sets default values for this pawn's properties
    ABase();

protected:
    // Called when the game starts or when spawned
    // virtual void BeginPlay() override;
    UPROPERTY(VisibleAnywhere,BlueprintReadOnly,Meta = (AllowPrivateAccess ="true"))
    UStaticMeshComponent* Turret;

public:
    UPROPERTY(VisibleAnywhere,BlueprintReadOnly,Meta = (AllowPrivateAccess ="true"))
    class UCapsuleComponent* Capsule;
    UPROPERTY(VisibleAnywhere,BlueprintReadOnly,Meta = (AllowPrivateAccess ="true"))
    UStaticMeshComponent* Base;
    UPROPERTY(VisibleAnywhere,BlueprintReadOnly,Meta = (AllowPrivateAccess ="true"))
    USceneComponent* Offset;
public:
    // Called every frame
    // virtual void Tick(float DeltaTime) override;
    void TurretRotate(FVector TargetPoint);
};
```

Base.h

```cpp
#include "Base.h"
#include "Components/CapsuleComponent.h"

// Sets default values
ABase::ABase()
{
    // Set this pawn to call Tick() every frame.
    PrimaryActorTick.bCanEverTick = true;
    Capsule = CreateDefaultSubobject<UCapsuleComponent>(TEXT("Capsule"));
    RootComponent = Capsule;
    Base = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Base"));
    Base->SetupAttachment(Capsule);
    Turret= CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Turret"));
    Turret->SetupAttachment(Base);
    Offset = CreateDefaultSubobject<USceneComponent>(TEXT("Offset"));
    Offset->SetupAttachment(Turret);
}

void ABase::TurretRotate(FVector TargetPoint){
    FVector ToRotation = TargetPoint - GetActorLocation();
    FRotator ToRotate = FRotator(0,ToRotation.Rotation().Yaw,0);
    Turret->SetWorldRotation(FMath::RInterpTo(Turret->GetComponentRotation(),
    ToRotate,GetWorld()->GetDeltaSeconds(),10));
}
```

Base.cpp

Player Movement

Player movement was implemented by configuring inputs in the SetupPlayerInputComponent function, where various actions and axes were bound to specific functions through UInputComponent.

- Move(): This function modifies the actor's position by calculating a delta vector based on movement speed and input value, which varies between -1.0 and 1.0 depending on the pressed key.
- Turn(): This function adjusts the actor's rotation along the z-axis by changing the yaw. The rotation is computed by multiplying the turning speed and input value, and is scaled by delta seconds to maintain consistent rotation speed across different hardware.

```
LASS()
ss TOONTANKS_API ATank : public ABase

 GENERATED_BODY()



blic:
 virtual void Tick(float DeltaTime) override;
 virtual void BeginPlay() override;

 ATank();
 virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override
 UPROPERTY(EditAnywhere)
 float speed = 100;
 UPROPERTY(EditAnywhere)
 float turnspeed = 100;

ivate:
 UPROPERTY(VisibleAnywhere,Category = "Component")
 class USpringArmComponent* SpringArm;
 UPROPERTY(VisibleAnywhere,Category = "Component")
 class UCameraComponent* Camera;
 void Move(float Value);
 void Turn(float Value);
 void Turret(float Value);
 void Fire();
 APlayerController* PlayerController;

 UPROPERTY(EditAnywhere)
 TSubclassOf<class ABulletsPlayer> BulletActor;
```

Tank.h

```
void ATank::Move(float Value)
{
    FVector DeltaLocation = FVector(Value*speed*GetWorld()->GetDeltaSeconds(),0,0);
    AddActorLocalOffset(DeltaLocation,true);
}

void ATank::Turn(float Value)
{
    FRotator DeltaRotation = FRotator::ZeroRotator;
    DeltaRotation.Yaw = Value*turnspeed*UGameplayStatics::GetWorldDeltaSeconds(this);
    AddActorLocalRotation(DeltaRotation,true);
}

void ATank::Turret(float Value)
{
    FRotator DeltaRotation = FRotator::ZeroRotator;
    DeltaRotation.Yaw = Value*turnspeed*UGameplayStatics::GetWorldDeltaSeconds(this);
    ABase::Turret -> AddLocalRotation(DeltaRotation,true);
}
```

Projectiles firing

Firing projectiles where an object, such as a turret, weapon, or character, can launch projectiles like bullets, arrows, or rockets.

```h
UCLASS()
class TOONTANKS_API ABulletsPlayer : public AActor
{
    GENERATED_BODY()

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

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;
private:
    UPROPERTY(VisibleAnywhere,BlueprintReadOnly,Meta = (AllowPrivateAccess ="true"))
    class UStaticMeshComponent* Bullet;

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

    UPROPERTY(EditAnywhere)
    class UProjectileMovementComponent* ProjectileMovement;
};
```

BulletsPlayer.h

```cpp
ABulletsPlayer::ABulletsPlayer()
{
    // Set this actor to call Tick() every frame.
    PrimaryActorTick.bCanEverTick = false;

    Bullet = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Bullet"));
    RootComponent = Bullet;

    ProjectileMovement = CreateDefaultSubobject<UProjectileMovementComponent>
    (TEXT("ProjectileMovement"));

    ProjectileMovement->InitialSpeed = 10000;
    ProjectileMovement->MaxSpeed = 10000;
    ProjectileMovement->bRotationFollowsVelocity = true;
    ProjectileMovement->bShouldBounce = true;
    ProjectileMovement->Bounciness = 0.3f;
    ProjectileMovement->ProjectileGravityScale = 0.5f;

}

// Called when the game starts or when spawned
void ABulletsPlayer::BeginPlay()
{
    Super::BeginPlay();

}

// Called every frame
void ABulletsPlayer::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

}
```

BulletsPlayer.cpp

Player Detection

This can be done by storing a reference to the player at the start of the game and simply checking the distance between the player and enemies. This distance-based detection allows enemies to react when the player comes within a certain range, minimizing the system's computational load.

```h
UCLASS()
class TOONTANKS_API AEnemy : public ABase
{
    GENERATED_BODY()
public:
    // Sets default values for this pawn's properties
    class ATank* Tank;
    AEnemy();
    virtual void BeginPlay() override;
    virtual void Tick(float DeltaTime) override;

    UPROPERTY(EditAnywhere, Category = "Combat")
    float DetectionRadius;
```

Enemy.h

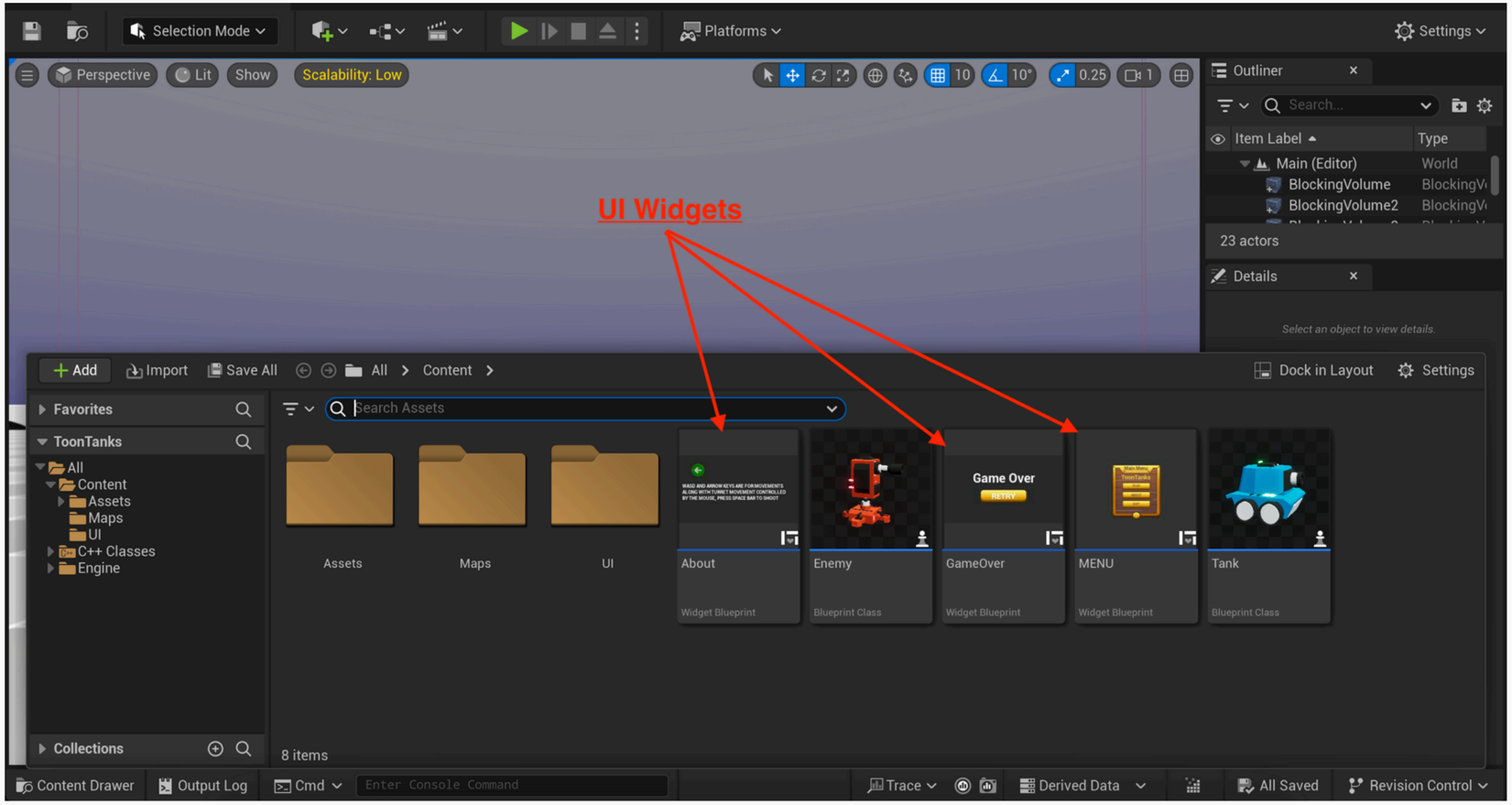
```cpp
#include "Enemy.h"
#include "Tank.h"
#include "Kismet/GameplayStatics.h"

AEnemy::AEnemy()
{
    PrimaryActorTick.bCanEverTick = true;
}
void AEnemy::BeginPlay()
{
    Super::BeginPlay();
    Tank = Cast<ATank>(UGameplayStatics::GetPlayerPawn(this, 0));
}
void AEnemy::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
    float distance = FVector::Dist(GetActorLocation(), Tank->GetActorLocation()
    if(distance <= DetectionRadius)
    {
        ABase::TurretRotate(Tank->GetActorLocation());
    }
}
```

Enemy.cpp

Task

Adding a basic User Interface in the game of this module and all the previous modules

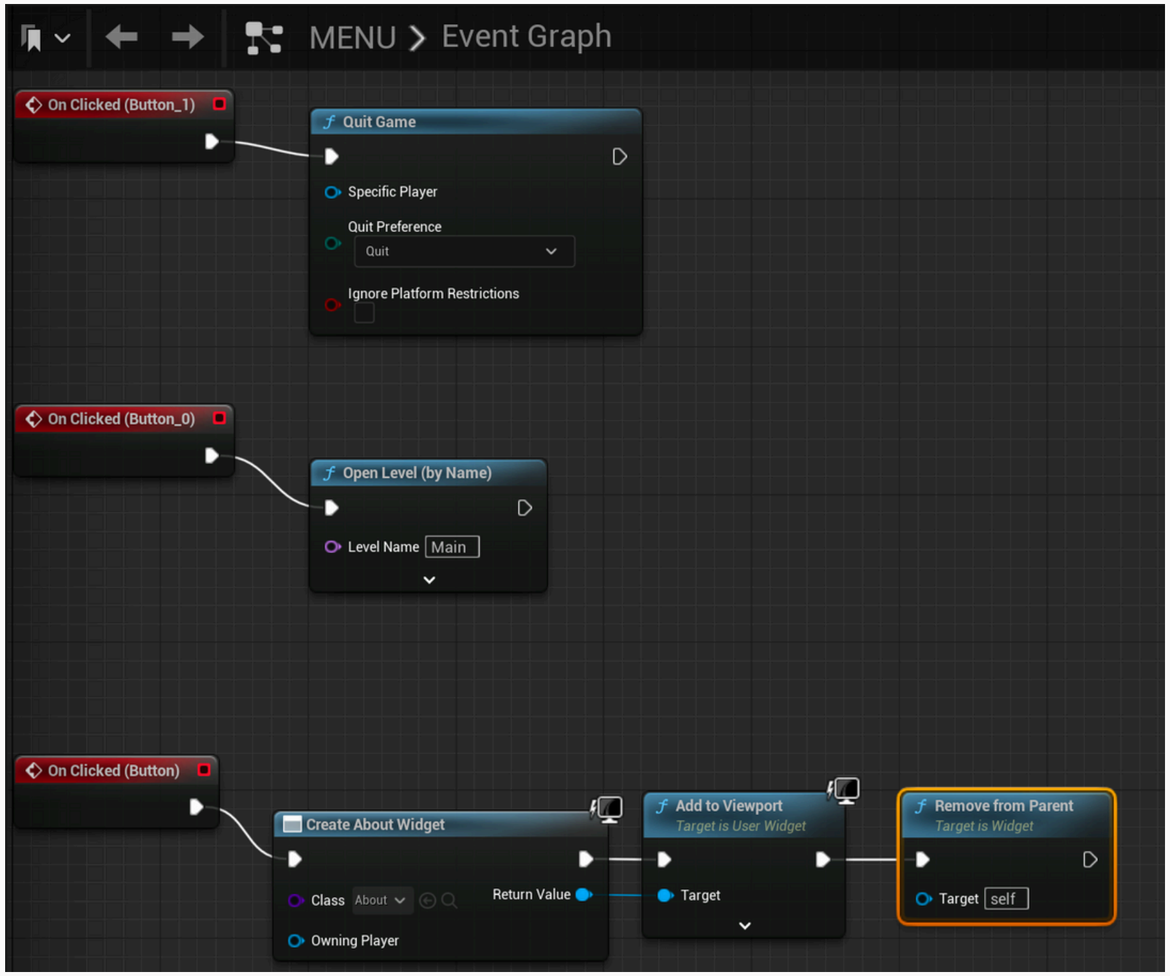
Solutions



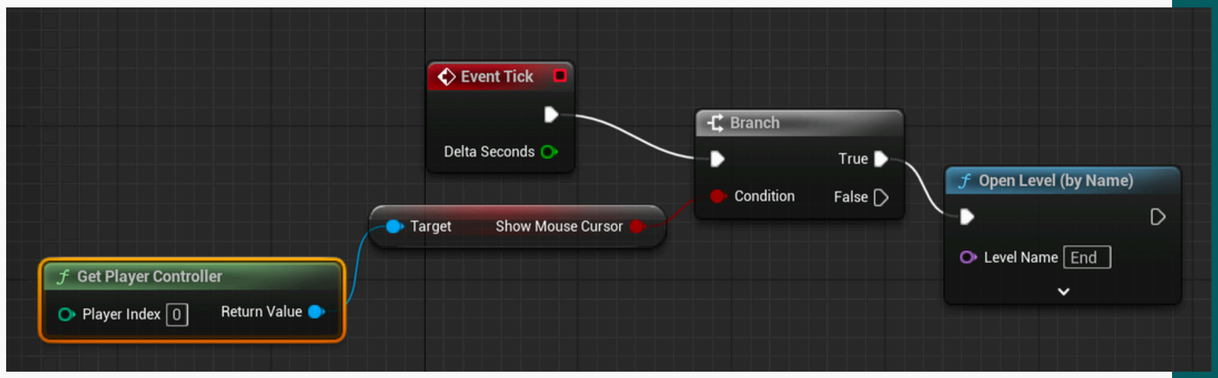
UI Widgets added in the game

Solutions

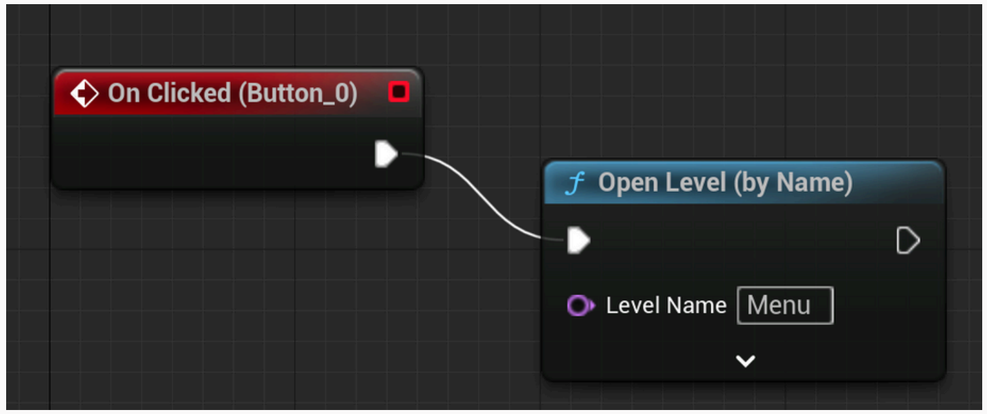
UI Widget Event Graphs



Menu Widget

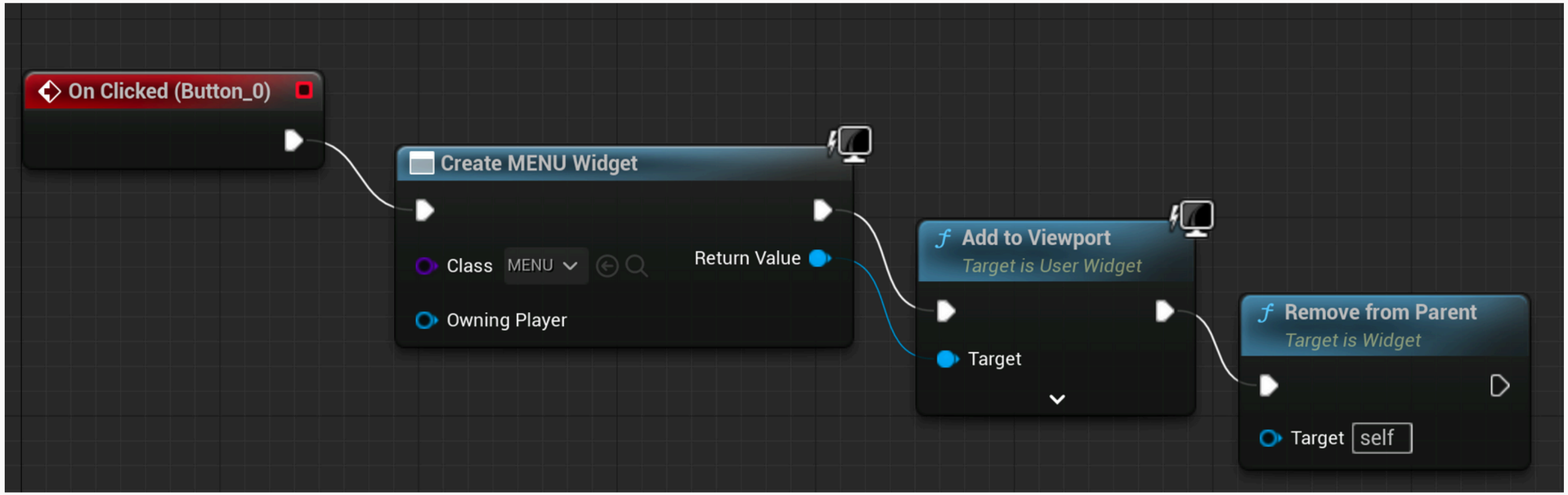


Loading game over screen



About UI Widget

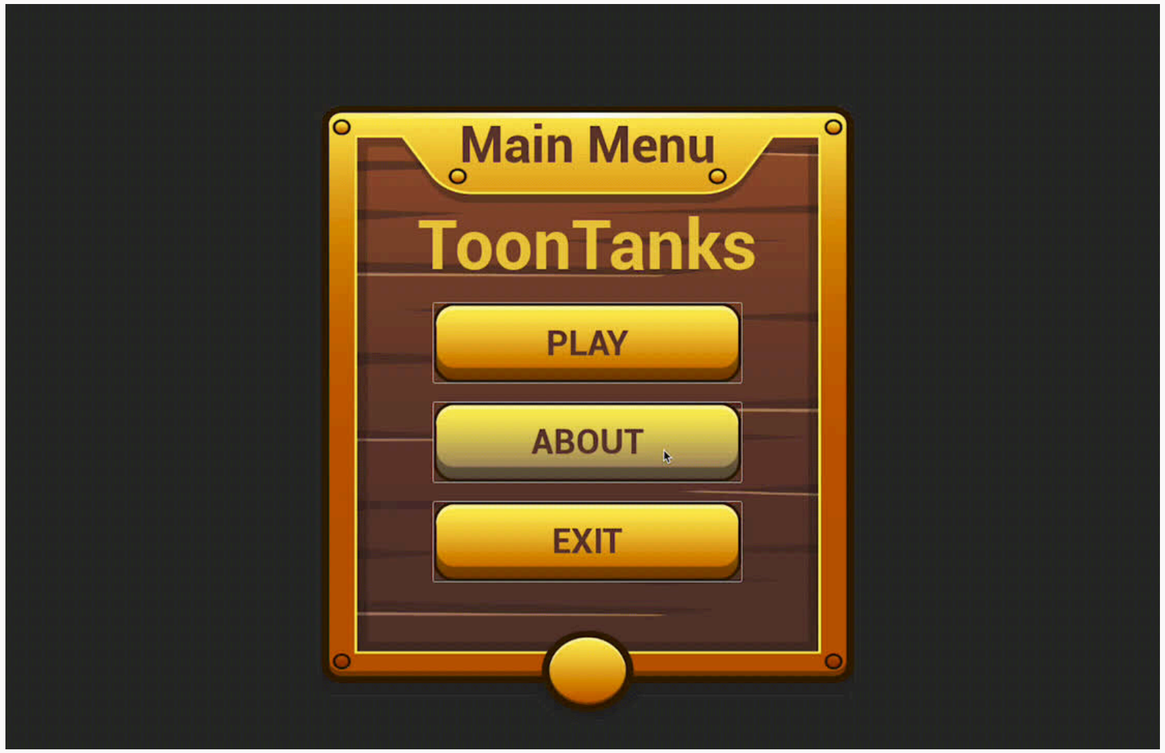
Solutions



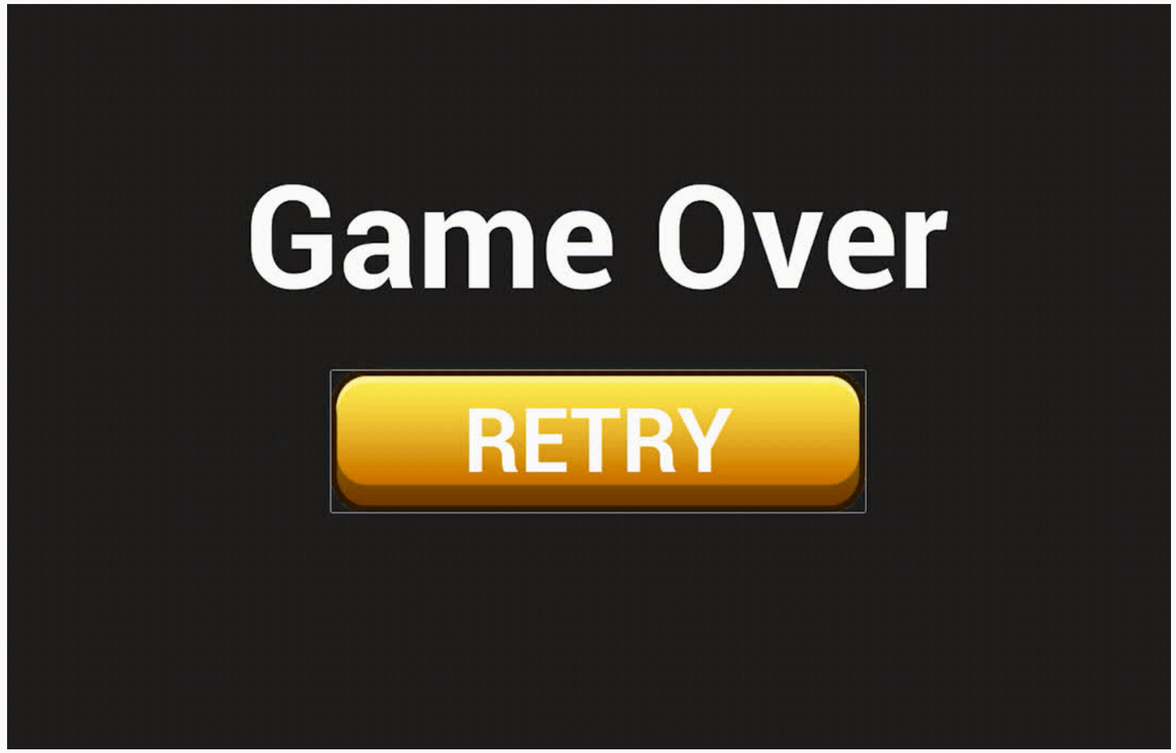
Game Over screen
(Restart Button)

Solutions

USER INTERFACE

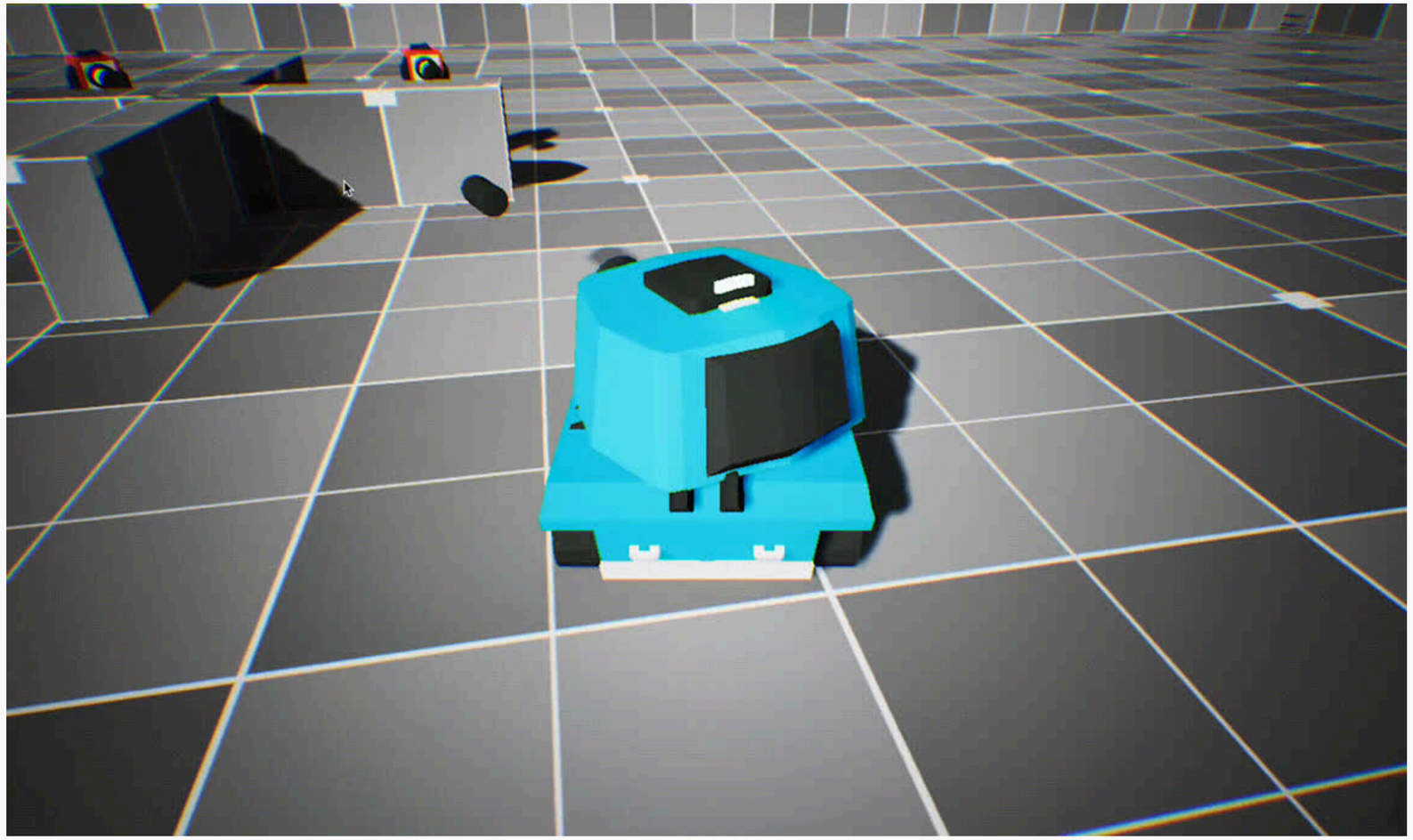


Main Menu



Game Over Screen

Solutions



Game Play

Thank You