**Brick Game**

Start a new blank project.

Delete all components in scene and save the level.

Copy assests such as Materials and Mesh folders

Drag and drop SM\_Backdrop and scale to 10.

Add wall and position to the origin

Add a Pawn C++ class of the name “Paddle”

Add 2 variables in header file

UPROPERTY(VisibleAnywhere, BlueprintReadOnly)

UStaticMeshComponent\* SM\_Paddle;

UPROPERTY(VisibleAnywhere, BlueprintReadOnly)

UFloatingPawnMovement\* FloatingMovement;

Forward declaration required

class UFloatingPawnMovement;

Add 2 includes in the C++ file.

#include "GameFramework/FloatingPawnMovement.h"

#include "Components/StaticMeshComponent.h"

In the constructor in C++ file, do the following setup

SM\_Paddle = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("SMPaddle"));

RootComponent = SM\_Paddle;

SM\_Paddle->SetEnableGravity(false);

SM\_Paddle->SetConstraintMode(EDOFMode::XZPlane);

SM\_Paddle->SetCollisionEnabled(ECollisionEnabled::QueryAndPhysics);

SM\_Paddle->SetCollisionProfileName(TEXT("PhysicsActor"));

FloatingMovement = CreateDefaultSubobject<UFloatingPawnMovement>(TEXT("Floating Pawn Movement"));

Add a functions in header file

virtual void MoveHorizontal(float AxisValue);

Create its definition and add the following in its definition.

AddMovementInput(FVector(AxisValue, 0.0f, 0.0f), 1.0f, false);

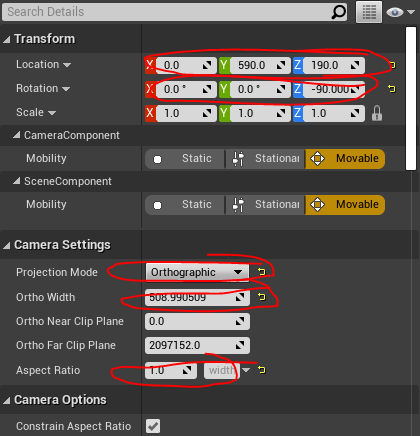
Create a BP version of the pawn in the editor.

Select the SM\_Paddle component in the BP and select the Paddle static mesh from the details panel.

Create a Game mode in the the BP folder and name it BrickGameMode

Select the Paddle pawn as the default pawn in the game mode and set the game mode in the maps and modes in project settings.

Add a camera and adjust the following settings to get a full view of the scene.



Create a PlayerController C++ class and name it Brick\_PC

Add following code in the header file

ABrick\_PC();

UFUNCTION()

virtual void SetupInputComponent() override;

protected:

virtual void BeginPlay() override;

void MoveHorizontal(float AxisValue);

Create definition for all 4 methods

Add following includes in the C++ file

First we set the camera added as target camera to view the game.

For that do this in BeginPlay

TArray<AActor\*> CameraActor;

UGameplayStatics::GetAllActorsOfClass(GetWorld(), ACameraActor::StaticClass(), CameraActor);

FViewTargetTransitionParams Params;

SetViewTarget(CameraActor[0], Params);

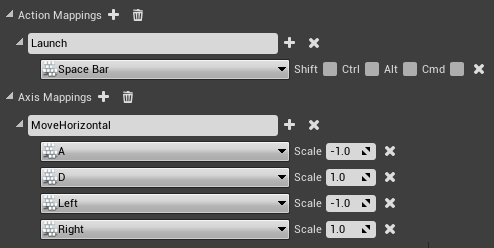
Compile and test

Create a BP version of the player controller

Set it as the PC in the game mode

Play to see the result

Add the following input binding



In the C++ file go to setup input component method and add the following

Super::SetupInputComponent();

EnableInput(this);

InputComponent->BindAxis("MoveHorizontal", this, &ABrick\_PC::MoveHorizontal);

The paddle should be working now.

**Creating Ball**

Create and actor class called “Ball”

In the header file of Ball add the following in Public

virtual void LaunchBall();

bool BallLaunched;

In protect under the begin play method add the following

UPROPERTY(VisibleAnywhere, BlueprintReadOnly)

UStaticMeshComponent\* SM\_Ball;

UPROPERTY(VisibleAnywhere, BlueprintReadOnly)

UProjectileMovementComponent\* ProjectileMovement;

Forward declaration required

class UProjectileMovementComponent;

In public add the following method deceleration

UFUNCTION()

UStaticMeshComponent\* GetBall();

Create definition of both functions LaunchBall and GetBall.

In C++ file add following include files

#include "Components/StaticMeshComponent.h"

#include "GameFramework/ProjectileMovementComponent.h"

Add the following in the constructor of the C++ file

SM\_Ball = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Ball Mesh"));

RootComponent = SM\_Ball;

SM\_Ball->SetSimulatePhysics(true);

SM\_Ball->SetEnableGravity(false);

SM\_Ball->SetConstraintMode(EDOFMode::XZPlane);

SM\_Ball->SetCollisionEnabled(ECollisionEnabled::QueryAndPhysics);

SM\_Ball->SetCollisionProfileName(TEXT("PhysicsActor"));

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

ProjectileMovement->bShouldBounce = true;

ProjectileMovement->Bounciness = 1.1f;

ProjectileMovement->Friction = 0.0f;

ProjectileMovement->Velocity.X = 0.0f;

Add the folloing code to LaunchBall

if (!BallLaunched)

{

SM\_Ball->AddImpulse(FVector(140.0f, 0.0f, 130.0f), FName(), true);

BallLaunched = true;

}

Add the following to GetBall

return SM\_Ball;

Create a BP version of the Ball class called “Ball\_BP”

Select the ball mesh in the static mesh component in the BP version

Place the ball in the world and resize accordingly.

**Ball Mechanics**

In Brick PC header file add forward declaration of Ball

class ABall;

Add the following to the end

void Launch();

UPROPERTY(EditAnywhere)

TSubclassOf<ABall> BallObj;

ABall\* MyBall;

FVector SpawnLocation = FVector(10.0f, 0.0f, 40.0f);

FRotator SpawnRotation = FRotator(0.0f, 0.0f, 0.0f);

FActorSpawnParameters SpawnInfo;

public:

void SpawnNewBall();

Create definition for both methods Launch and SpawnNewBall

In the C++ file first include the ball.h

#include "Ball.h"

Add the following to the method Launch

MyBall->LaunchBall();

Add input binding to SetInputComponent method

InputComponent->BindAction("Launch", IE\_Pressed, this, &ABrick\_PC::Launch);

Add the following in SpawnNewBall method to launch ball

if (!MyBall)

{

MyBall = nullptr;

}

if (BallObj)

{

MyBall = GetWorld()->SpawnActor<ABall>(BallObj, SpawnLocation, SpawnRotation, SpawnInfo);

}

Need the following include for getworld to work.

#include "Engine.h"

Now we need to call SpawnNewBall from begin play.

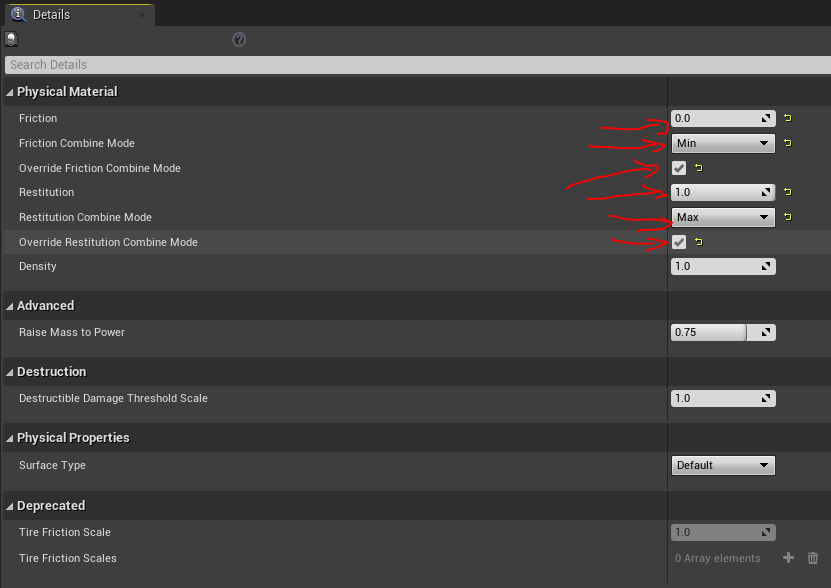
SpawnNewBall();

In the editor in the BrickPC BP select the BallBP in the class defaults.

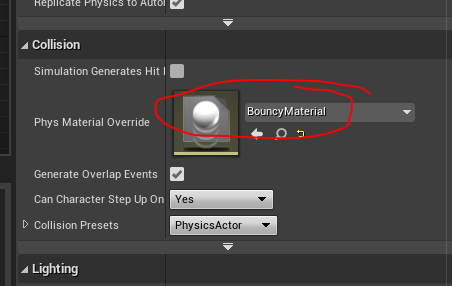
Ball should be launchable.

To fix the sliding issue RIghtclick and add a physics material in the content browser

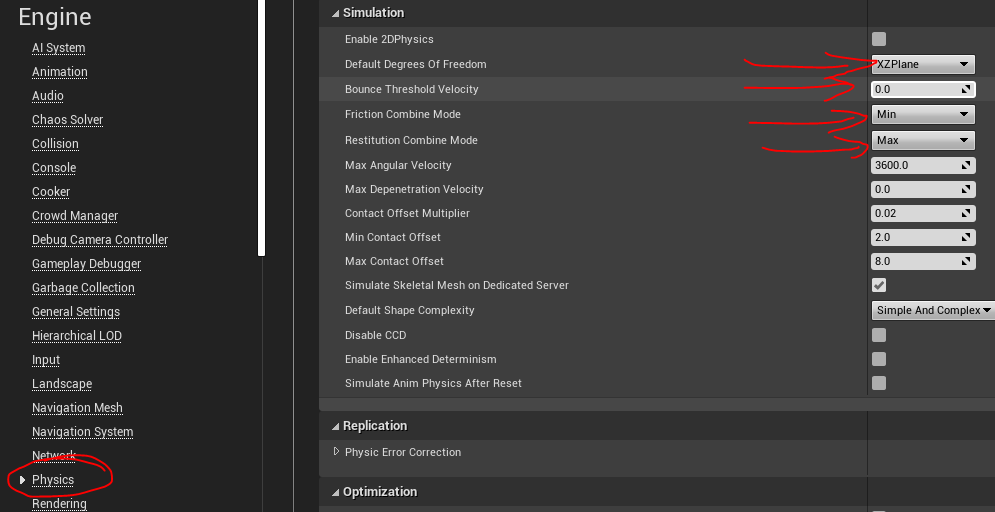
Change the following attributes in physics material



Set this Physicsmaterial as the override material for the sm\_ball in the ball BP



Now in the Project Settings/Engine/Physics/



**Bricks**

Create a C++ actor class called Brick

Inside the header file add the following.

UPROPERTY(VisibleAnywhere, BlueprintReadOnly)

UStaticMeshComponent\* SM\_Brick;

UPROPERTY(VisibleAnywhere, BlueprintReadOnly)

UBoxComponent\* BoxComp;

float SpeedModifierOnBounce = 1.0f;

void DestroyBrick();

Forward decleration required

class UBoxComponent;

Create definition of the function

In the C++ file add these includes

#include "Components/StaticMeshComponent.h"

#include "Components/BoxComponent.h"

#include "Ball.h"

In the constructor add the following

SM\_Brick = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Brick"));

SM\_Brick->SetCollisionEnabled(ECollisionEnabled::QueryAndPhysics);

BoxComp = CreateDefaultSubobject<UBoxComponent>(TEXT("BoxCollision"));

BoxComp->SetCollisionEnabled(ECollisionEnabled::QueryAndPhysics);

BoxComp->SetBoxExtent(FVector(25.0f, 10.0f, 10.0f));

RootComponent = BoxComp;

Create a BP version of the brick.

Select the brick mesh in details panel of the SM\_Brick in BP

Add the Brick in the level.

In the begin play create dynamic binding for OncomponentBeginOverlap as follows.

BoxComp->OnComponentBeginOverlap.AddDynamic(this, &ABrick::OnOverlapBegin);

We need to create a method called OnOverlapBegin with the same parameters as OnComponentBeginOverlap.

To get the parameters right click on the “OnComponentBeginOverlap” and go to definition.

It will take you to PrimitiveComponents.h file.

Right click on the FComponentBeginOverlapSignature and go to definitions and we get the parameters.

Create a function in header file named OnOverlapBegin and paste the parameters.

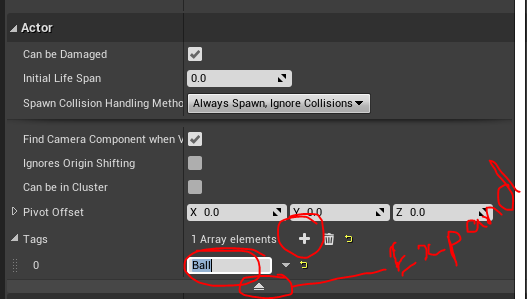
UFUNCTION()

void OnOverlapBegin(class UPrimitiveComponent\* OverlappedComponent, class AActor\* OtherActor, class UPrimitiveComponent\* OtherComp, int32 OtherBodyIndexType, bool bFromSweep, const FHitResult& SweepResult);

Create its definition.

Before programming we need to add tag to ball

Select the BallBP, go to class defaulta and add tag as follows



Add following script to OnOverlapBegin method

if (OtherActor->ActorHasTag("Ball"))

{

ABall\* MyBall = Cast<ABall>(OtherActor);

FVector BallVelocity = MyBall->GetVelocity();

BallVelocity \*= (SpeedModifierOnBounce - 1.0f);

MyBall->GetBall()->SetPhysicsLinearVelocity(BallVelocity, true);

this->Destroy();

}

Compile and run. This will not be perfect.

So move the this->Destroy(); to DestroyBrick method.

Add the following to to OnOverlapBegin method inside if

FTimerHandle UnusedHandle;

GetWorldTimerManager().SetTimer(UnusedHandle, this, &ABrick::DestroyBrick, 0.1f, false);

**Ball Bunce and Kill**

Create C++ class BallBound

Add following in the headerfile

UPROPERTY(VisibleAnywhere, BlueprintReadOnly)

UBoxComponent\* Box\_Collision;

ABrick\_PC\* PlayerControllerRef;

Forward declaration required

class UBoxComponent;

class ABrick\_PC;

In the C++ file add the following include

#include "Components/BoxComponent.h"

#include "Kismet/GameplayStatics.h"

#include "Brick\_PC.h"

In the constructor add the following

Box\_Collision = CreateDefaultSubobject<UBoxComponent>(TEXT("Box Collision"));

RootComponent = Box\_Collision;

In the beginPlay create dynamic binding for oncomponentbeginoverlap as follows

Box\_Collision->OnComponentBeginOverlap.AddDynamic(this, &ABallBound::OnOverlapBegin);

PlayerControllerRef = Cast<ABrick\_PC>(UGameplayStatics::GetPlayerController(GetWorld(), 0));

We need to create the function OnOverlapBegin with parameters of OnComponentBeginOverlap.

So rightclick on OnComponentBeginOverlap and go to definitions and rightclick in Dynamic and get the parameters

Create function declaration in header file as follows

UFUNCTION()

void OnOverlapBegin(UPrimitiveComponent\* OverlappedComponent, AActor\* OtherActor, UPrimitiveComponent\* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult& SweepResult);

Create function definition and add the following

if (OtherActor->ActorHasTag("Ball"))

{

OtherActor->Destroy();

PlayerControllerRef->SpawnNewBall();

}

Compile and save and create a BP version of the class

Place in the bottom of the level.

It should respawn when ball hits the block BallBound.