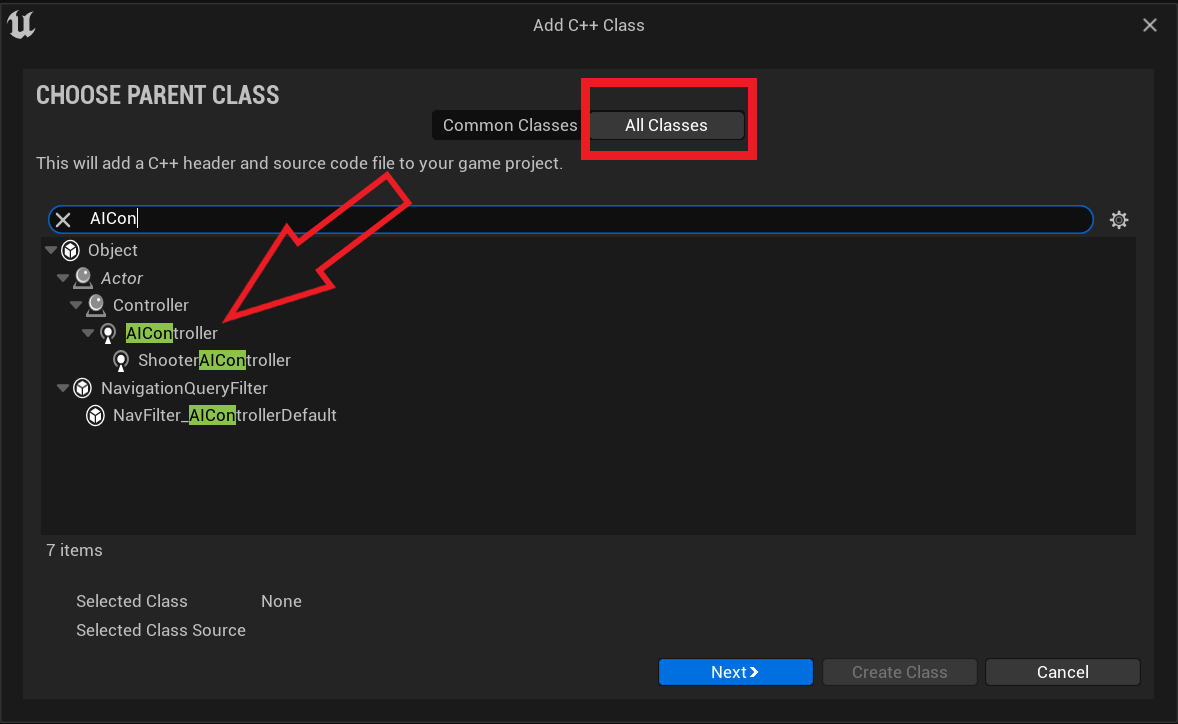
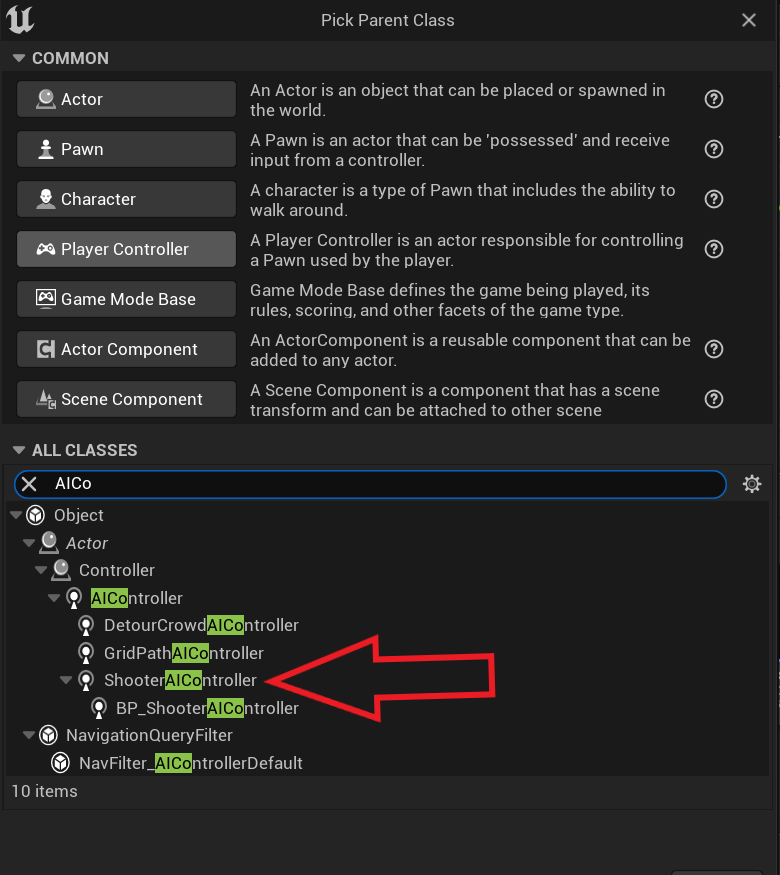
Bu yazı tamamen AI üzerine olacak bu yüzden uzun olabilir. Bu yazıyla beraber düşman yapay zekası ekleme bizleri takip etmesini sağlama ve bizi görünce ateş etmesini sağlama gibi özellikler ekleyeceğiz.

İlk olarak bir c++ sınıfı oluşturmak ile başlıyoruz. Bu sefer C++ sınıfı oluştururken AI Controller üzerine oluşturacağımız için farklı bir parent seçeceğiz. Normal c++ oluşturma ekranımızı açıyoruz.



Bundan sonraki aşamayı zaten biliyoruz AI adı veriyoruz ben yine kolay anlaşılabilir olması için ShooterAIController ismini seçtim. Artık AI kontrolleri için C++ sayfamız hazır şimdi bunun için blueprint oluşturma zamanı.



Blueprintimize BP\_ShooterAIController ismini verdim. Böylece Bluprintimiz de hazır. BP\_ShooterCharacter Blueprinti açıp içeride değişiklik yapmamız gerekiyor. İçeride Pawn ana başlıklı yerden AI Controller Class’ı yeni oluşturduğumuz BP\_ShooterAIController olarak değiştiriyoruz. Artık AI için kodları yazmaya hazırız.

Şimdi karşı aktörümüzün (düşmanın) bize bakmasını sağlayacağız. Fakat şu an yazacağımız kodlarda hala bizi yürüyerek takip edemeyecek sadece gözleriyle takip edecek arkasına geçersek ise dönerek bize bakmaya devam edecek. O zaman başlayalım.

ShooterAIController.h:

UCLASS()

class SIMPLESHOOTER\_API AShooterAIController : public AAIController

{

GENERATED\_BODY()

protected:

virtual void BeginPlay() override;

ShooterAIController.cpp:

void AShooterAIController::BeginPlay()

{

Super::BeginPlay();

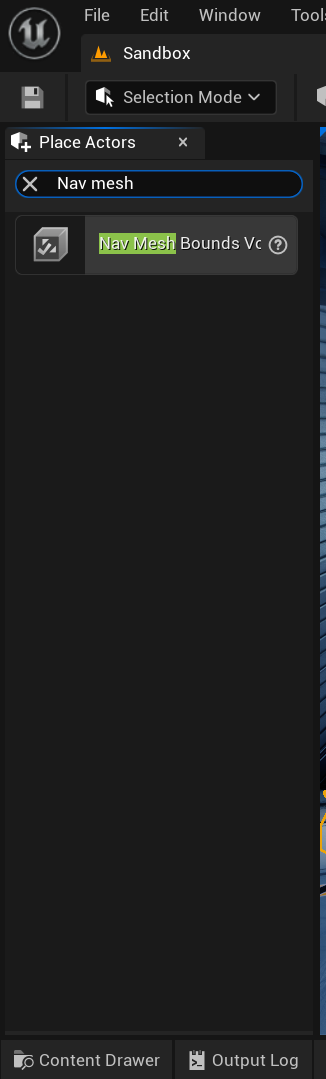
APawn \*PlayerPawn = UGameplayStatics::GetPlayerPawn(GetWorld(), 0);

SetFocus(PlayerPawn);

}

Bu kodları yazıp çalıştırınca göreceğiz ki karşı aktör bizleri gözleriyle takip ediyor.

Şimdi sırada bu AI’a yürüme eklememiz gerekiyor. Bunun için kodlara geçmeden önce yapmamız gereken bir şey var o da Nav Mesh eklemek.



Place Actors kısmından Nav Mesh yazarak arama yapıyoruz ve önümüze çıkanı ekliyoruz. Nav Mesh bize bir hacim verecek bu hacmi tüm haritaya yaymak için haritadan iyice uzaklaşalım ve Nav Mesh’i iyice genişletelim. Tüm haritayı kapladıktan sonra ve kodları yazınca artık AI bizi tüm haritada takip edebilecek. Nav Mesh’in şöyle bir güzel özelliği var eğer duvara denk gelirse ona göre kendini kesiyor.

ShooterAıController.h:

public:

virtual void Tick(float DeltaSeconds) override;

ShooterAIController.cpp:

SetFocus(PlayerPawn);

void AShooterAIController::Tick(float DeltaSeconds)

{

APawn \*PlayerPawn = UGameplayStatics::GetPlayerPawn(GetWorld(), 0);

MoveToActor(PlayerPawn, 200);

}

Bu kodların sonunda artık aktör AI bizleri yürüyerek de takip edecek fakat kodu çalıştırıp denediğiniz zaman şunu farkediyorsunuz AI sizi sürekli kadar takip ediyor. İstediğimiz AI bir süre bizi göremeyince eski yerine dönmesi. Yani şuan yapmamız gereken şey bunu düzeltmek.

ShooterAIController.cpp:

void AShooterAIController::Tick(float DeltaSeconds)

{

Super::Tick(DeltaSeconds);

APawn \*PlayerPawn = UGameplayStatics::GetPlayerPawn(GetWorld(), 0);

if(LineOfSightTo(PlayerPawn))

{

SetFocus(PlayerPawn);

MoveToActor(PlayerPawn, AcceptanceRadius);

}

else

{

ClearFocus(EAIFocusPriority::Gameplay);

StopMovement();

}

ShooterAIController.h:

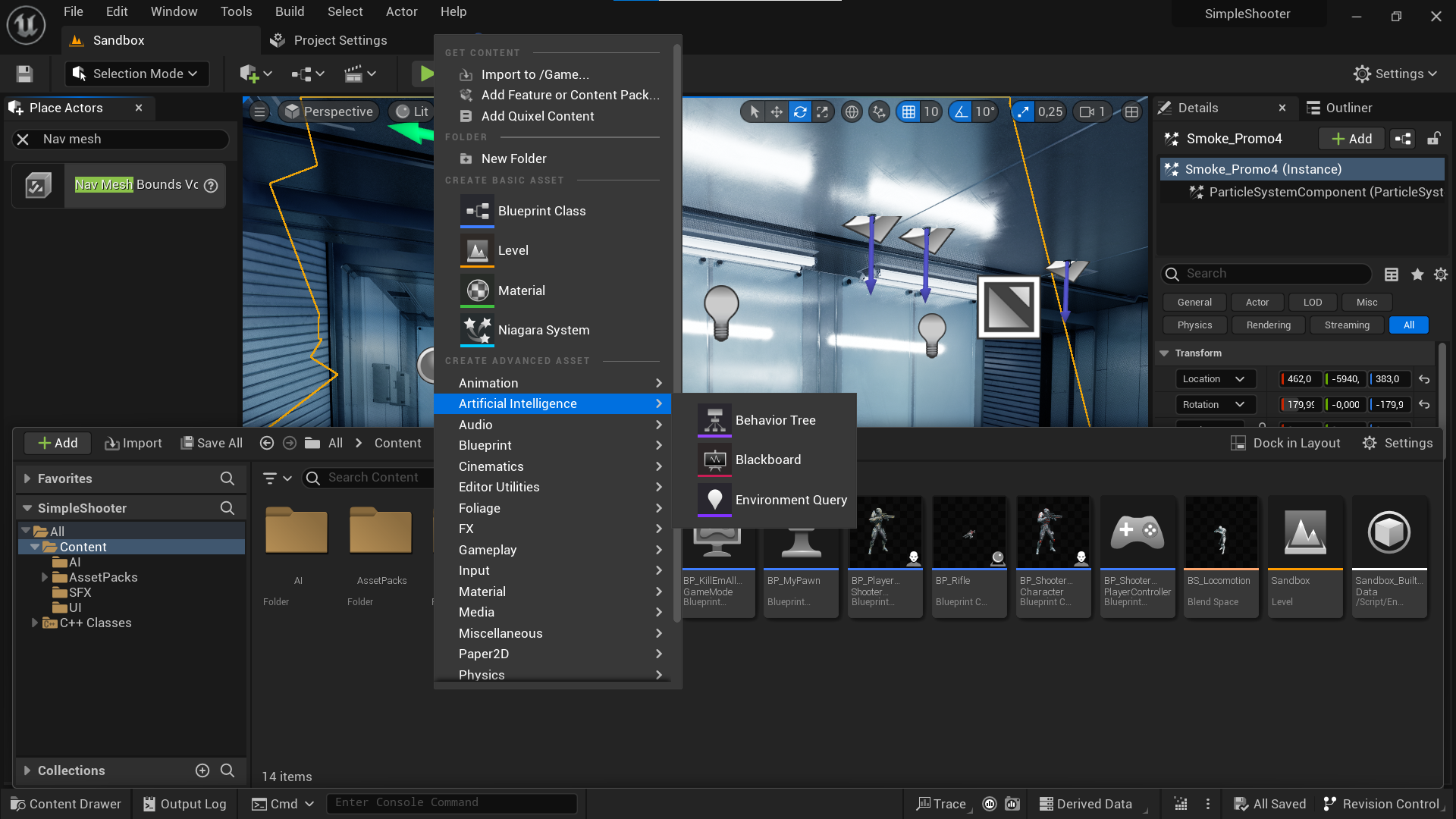
private:

UPROPERTY(EditAnywhere)

float AcceptanceRadius = 200;

Üstteki kodları yazıp çalıştırınca göreceğiz ki AI bizi takip edecek ve başarıp görüş alanından çıkabilirsek bizi takip etmeyi bırakacak. Başarabilirsek diyorum çünkü AI şuan bizle aynı hızda ilerliyor. Fakat bu elbetteki bizim için geçici bir çözüm.

Şimdi blackboard ve behaviortrees konularını göreceğiz. Blackboard ve behaviortrees AI ile farklı olaylar yapabilmemize olanak tanır. Şimdi neler yapabileceğimizi görelim. Öncelikle bir Behaviortree ve blackboard ekleyelim. Aşağıda gördüğümüz şekilde ekliyoruz. Behaviortree için BT\_EnemyAI ve blackboard için BB\_EnemyAI isimlerini veriyoruz. Content kısmımız çok kalabalık olmaya başladı baştan beri temiz ve düzenli çalışmaya çalışıyoruz bu yüzden AI adında yeni bir klasör oluşturup BP\_ShooterAIController, BB\_EnemyAI ve BT\_EnemyAI dosyalarımızı buraya taşıyalım.



BT\_EnemyAI dosyamızı açınca sadece root yazılı bir ekranla karşılaşacağız aslında buradan diğer davranışlara yönelebileceğiz yukarıda ise blackboard’a geçiş yapabildiğimizi göreceksiniz. AI kısmının altında Behaviortre altında oluşturduğumuz blackboard’un bağlı olduğunu göreceksiniz.

O zaman kodlara geçelim

ShooterAIController.h:

private:

UPROPERTY(EditAnywhere)

float AcceptanceRadius = 200;

Önceden bu kodu yazmıştık şimdi bu kodu siliyoruz yerine :

private:

UPROPERTY(EditAnywhere)

class UBehaviorTree\* AIBehavior;

Bu kodu ekliyoruz.

ShooterAIController.cpp:

Bu kodda da değişiklikler yapacağız öncelikle yazdığımız şu kodları siliyoruz:

void AShooterAIController::Tick(float DeltaSeconds)

{

Super::Tick(DeltaSeconds);

APawn \*PlayerPawn = UGameplayStatics::GetPlayerPawn(GetWorld(), 0);

if(LineOfSightTo(PlayerPawn))

{

SetFocus(PlayerPawn);

MoveToActor(PlayerPawn, AcceptanceRadius);

}

else

{

ClearFocus(EAIFocusPriority::Gameplay);

StopMovement();

}

Ve yerine şu kodları yazıyoruz:

void AShooterAIController::BeginPlay()

{

Super::BeginPlay();

if (AIBehavior != nullptr)

{

RunBehaviorTree(AIBehavior);

}

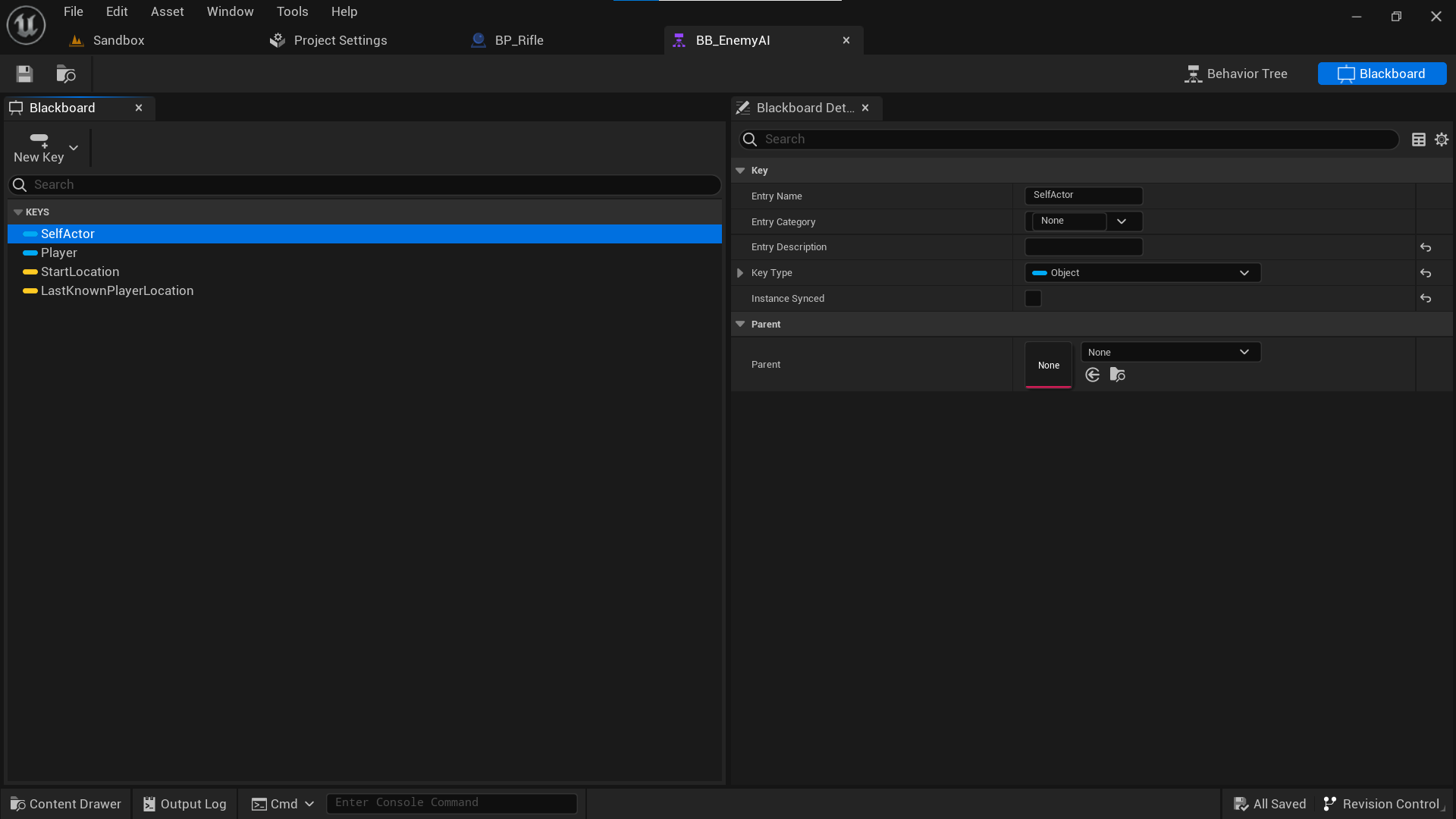
}

Şuanlık kodlarımızda yapacağımız değişiklikler bu kadar şimdi Behaviortree kısmına geri dönüyoruz.

BehaviorTree kısmından root’a sequence ekliyoruz. Ardından ShooterAIController kısmına geliyoruz ve AIBehavior’a BT\_EnemyAI ekliyoruz.

Şimdi biraz blackboard tarafına geçelim ve blackboard üzerinden çalışalım.

Sol taraftan yeni bir key tanımlayarak başlıyoruz. Adına SelfActor diyorum.

 Type Object olmalı. Ardından kodlaması kısmına geçelim.

ShooterAIController.cpp:

#include "BehaviorTree/BlackboardComponent.h"

void AShooterAIController::BeginPlay()

{

Super::BeginPlay();

if (AIBehavior != nullptr)

{

RunBehaviorTree(AIBehavior);

APawn \*PlayerPawn = UGameplayStatics::GetPlayerPawn(GetWorld(), 0);

GetBlackboardComponent()->SetValueAsVector(TEXT("StartLocation"), GetPawn()->GetActorLocation());

}

}

Az önce bir kısmını yazdığımız kodu şimdi tamamlıyoruz. Kodları tamamlayınca Behaviortree üzerinden Blackboard kısmına geçiyoruz. Yeni bir key ekleyeceğiz kendimize bu sefer bu keyin type’ı vector. Bunu yaptıktan sonra Behaviortree kısmına geri dönelim ve bazı eklemeler yapalım. Bu kısımda kodları tekrar tekrar değiştirmek yerine yapılması gerekenlerin son halini atacağım böylece tekrar tekrar kod yazıp silmeyeceğiz.

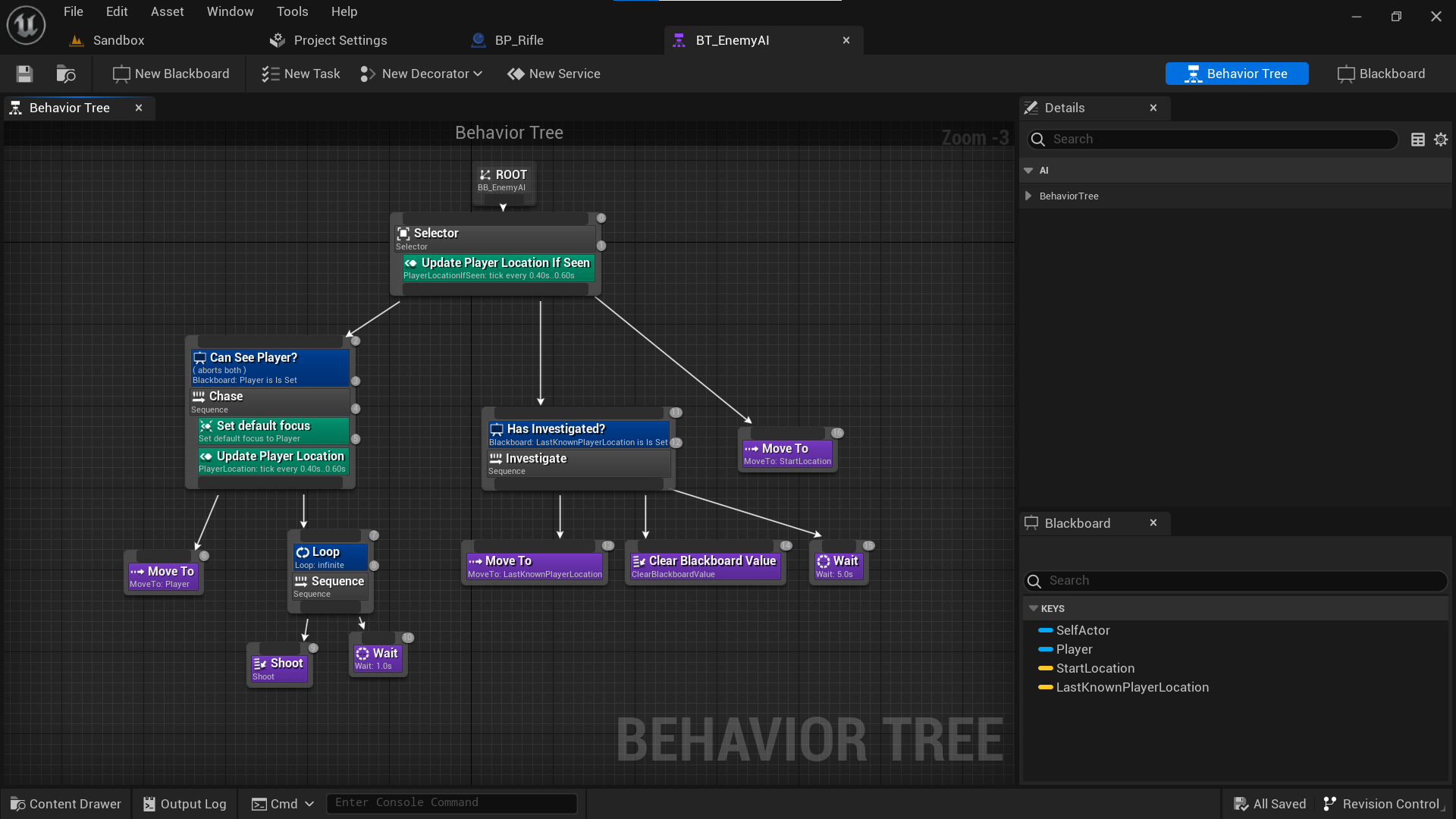
Şimdi iki tane daha C++ sayfası açacağız. İkisi de BTT\_Task BlackBoardBase parents altında olacak. Ben birine BTT\_Task\_BlackBoardBase adını verdim diğerine ise BTT\_Task\_ClearBlackboardValue adını verdim.

Şimdi burada bir hata ile karşılaşabiliriz OutPut Loglarında bir hata olabilir bunu düzeltmek için SimpleShooter.Build.cs kod sayfamızı açalım (Projeyi unreal engine üzerinde oluşturunca otomatik olarak source dosyasına bunu oluşturur) ardından şu kodu bulup değiştirelim:

PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject", "Engine", "InputCore", "GameplayTasks", "UMG" });

Koda "GameplayTasks", "UMG" ekledik.

Sonuç olarak :



Behaviortreemizi bu hale getiriyoruz. Kodlarımızda ise yeni bir c++ kodu açıyoruz ve adı BTService\_PlayerLocation diyoruz. Ardından kodlarımızın son halini verelim: