Shooting Game PortFolio

Contact
tjsdnr960@naver.com · 010 6304 7262

Updated
2023. 10. 17

Profile

안녕하세요, 김선욱 입니다.

세상과 소통하고 공간 제약 없이 모두가 즐길 수 있는
게임과 메타버스 컨텐츠를 만드는 목표를 가진 개발자 입니다.

Information

Name	김선욱	Birth	1996. 03. 17
Contact	010 6304 7262	Instagram	ws_960
Email	tjsdnr960@naver.com	GitHub	https://github.com/Woogie-Gim

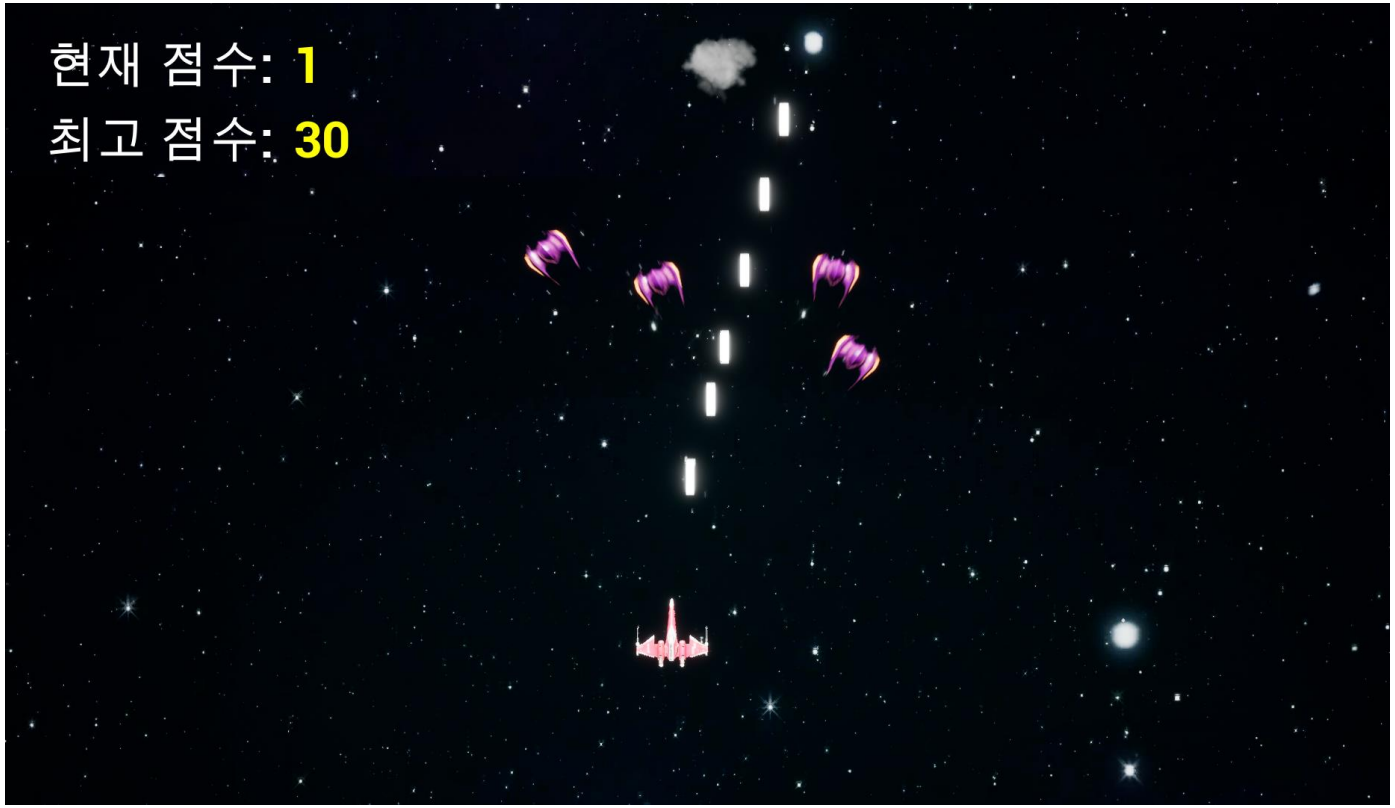
Education

2015.03 – 2020.08	전남대학교 농식품생명화학부 생명화학전공
2020.09 – 2022.08	전남대학교 농화학과 석사 졸업
2023.07 ~	삼성청년 소프트웨어 아카데미 교육 중

Shooting Game

게임 개요

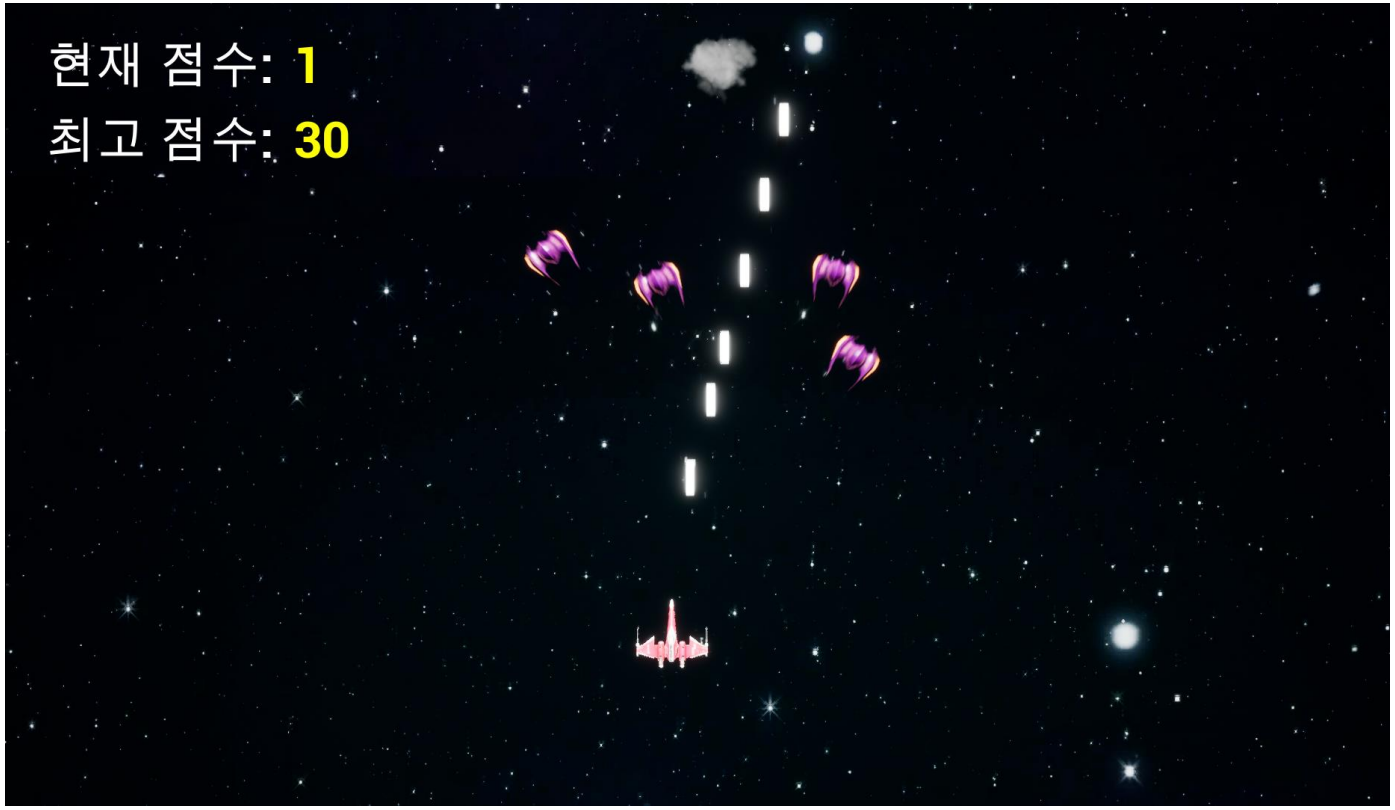
- 장르 : 종 스크롤 슈팅 게임
- 개발 엔진 : Unreal 4.27.2
- 게임 주제 : 몰려드는 적 우주선을 피해 최대한 많은 적을 물리치는 슈팅 게임



Shooting Game

게임 목적

- 언리얼 엔진 활용 개발 학습
- 언리얼 엔진 클래스와 메서드 등 학습
- 게임을 스스로 만들어 보고 C++ 코딩 학습
- 충돌 함수 구현과 화면 위젯 구현 학습
- 첫 게임 프로젝트를 완성하기 위해
  구현하기 쉽고 간단한 게임 장르로 손택

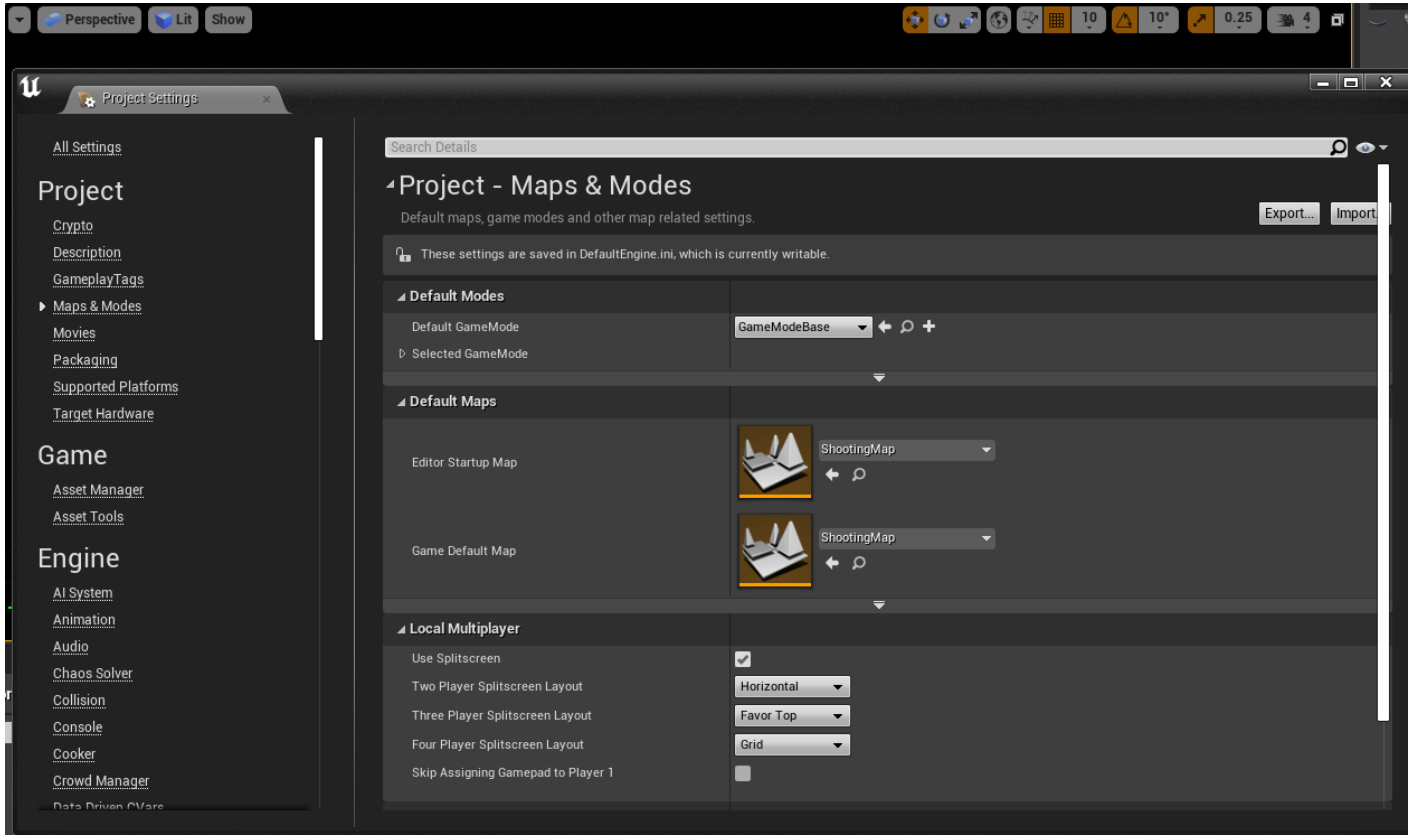


게임 주요 코드 및 포트폴리오:

Shooting Game

환경 구성

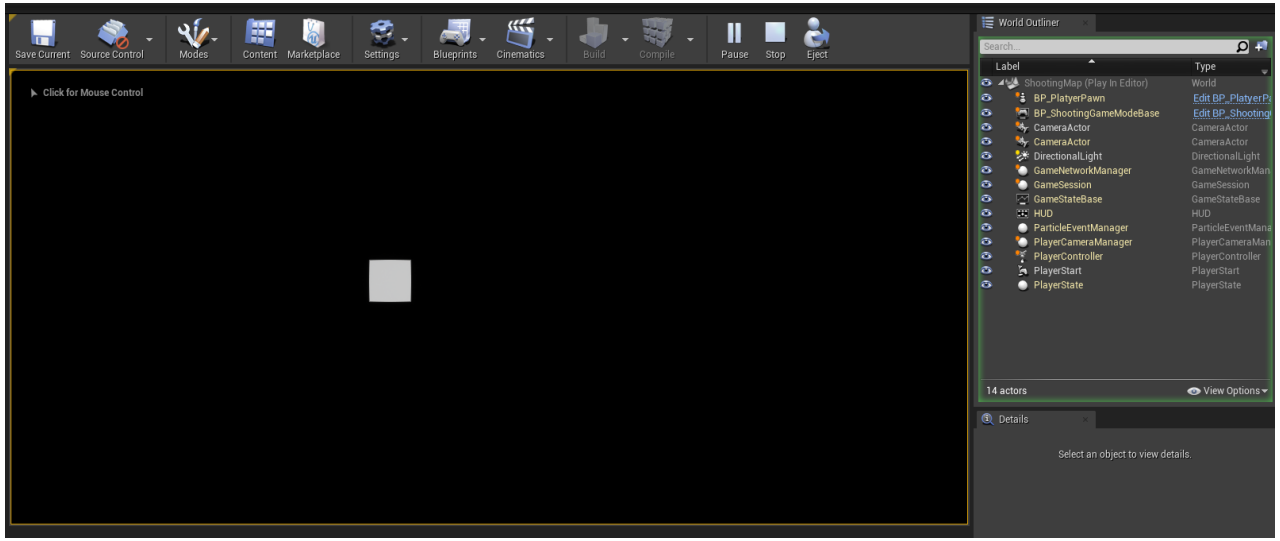
- Shooting map을 생성
- Default map 설정
- Gamemode 생성
- Default gamemode 생성
- 종스크롤 환경을 위한 카메라 및 라이트 설치

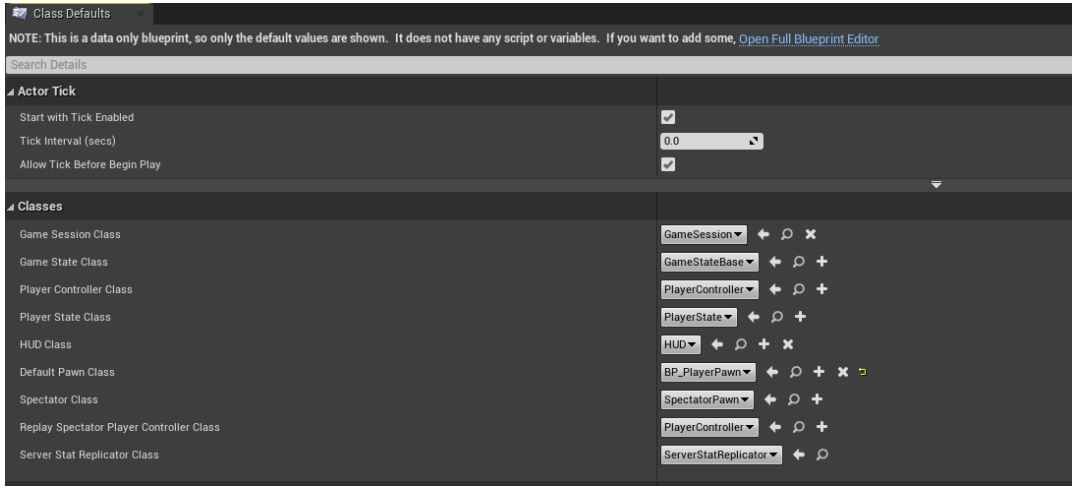


Shooting Game

플레이어 제작

- Player에 대한 C++ class와 블루프린트 생성
- Box 형태의 Player 알파 버전 제작
- 속력, 이동키, 충돌 함수 구현





Shooting Game

플레이어 주요 함수

PlayerPawn.h

```h
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Pawn.h"
#include "PlayerPawn.generated.h"

UCLASS()
class SHOOTINGCPP_API APlayerPawn : public APawn
{
    GENERATED_BODY()

public:
    // 이 폰의 속성에 대한 기본 값을 설정(Sets default values for this pawn's properties)
    APlayerPawn();

protected:
    // 게임이 시작될 때 또는 스폰될 때 호출됨(Called when the game starts or when spawned)
    virtual void BeginPlay() override;

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

    // 기능을 입력에 연결하기 위해 호출됨(Called to bind functionality to input)
    virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override;

    // 박스 충돌체 컴포넌트
    UPROPERTY(EditAnywhere)
    class UBoxComponent* boxComp;

    // 스태틱 메시 컴포넌트
    UPROPERTY(EditAnywhere)
    class UStaticMeshComponent* meshComp;

};
```

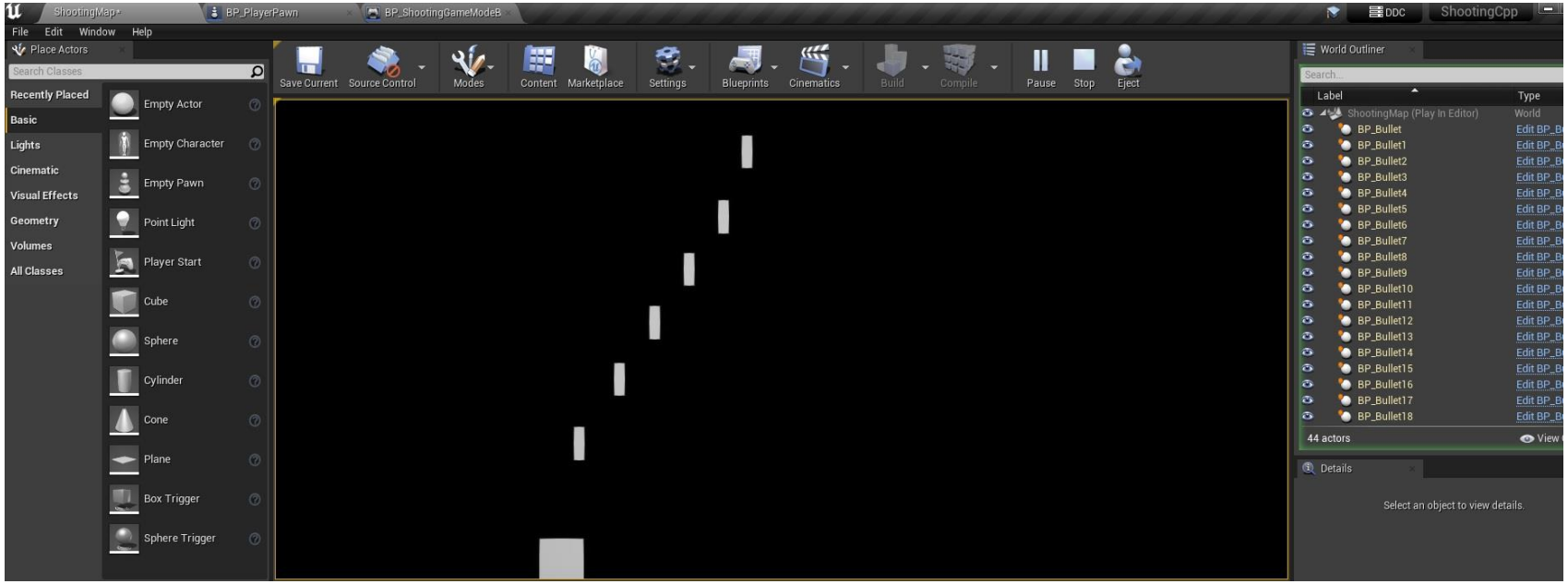
PlayerPawn.cpp

```cpp
#include "PlayerPawn.h"
#include "Components/BoxComponent.h"
#include "Components/StaticMeshComponent.h"

// 기본값 설정(Sets default values)
APlayerPawn::APlayerPawn()
{
    // 매 프레임마다 Tick()를 호출하도록 이 액터를 설정함. 필요하지 않은 경우 이 기능을 해제하여 성능을 향상시킬 수
    // 있음 (Set this pawn to call Tick() every frame.  You can turn this off to improve
    // performance if you don't need it.)
    PrimaryActorTick.bCanEverTick = true;

    // 박스 콜라이더 컴포넌트를 생성한다.
    boxComp = CreateDefaultSubobject<UBoxComponent>(TEXT("My Box Component"));

    // 생선한 박스 콜라이더 컴포넌트를 최상단 컴포넌트로 설정한다b
    SetRootComponent(boxComp);

    //스태틱 메시 컴포넌트를 생성한다
    meshComp = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("My Static Mesh"));

    // 박스 콜라이더 컴포넌트의 자식 컴포넌트로 설정한다.
    meshComp->SetupAttachment(boxComp);
}

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

}
```

Shooting Game

총알 제작

- 총알에 대한 C++ class와 블루프린트 생성
- Box 형태의 총알 알파 버전 제작
- 속력, 충돌 함수 구현
- 총알 생성 및 계층 설정
- 총알 발사 키 설정 및 함수 구현
- 플레이어 발사 위치 및 함수 구현
- 총알 발사 효과음 적용



Shooting Game

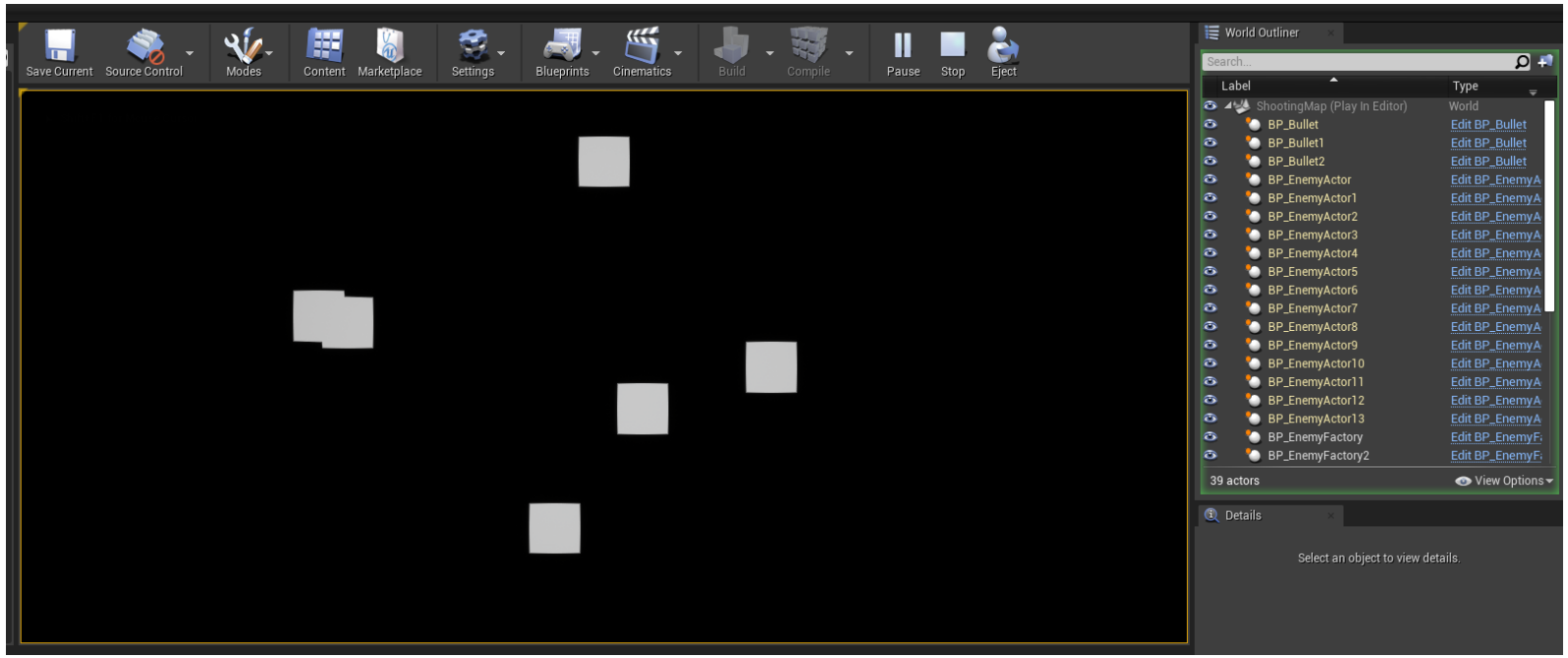
총알 주요 함수

```cpp
#include "Bullet.h"
#include "Components/BoxComponent.h"
#include "Components/StaticMeshComponent.h"

// Sets default values
ABullet::ABullet()
{
    // Set this actor to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
    PrimaryActorTick.bCanEverTick = true;

    boxComp = CreateDefaultSubobject<UBoxComponent>(TEXT("Box Collider"));
    SetRootComponent(boxComp);
    boxComp->SetBoxExtent(FVector(50.0f, 50.0f, 50.0f));

    meshComp = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Static Mesh Componenet"));
    meshComp->SetupAttachment(boxComp);
```

```cpp
#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Bullet.generated.h"

UCLASS()
class SHOOTINGCPP_API ABullet : public AActor
{
    GENERATED_BODY()

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

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)
    class UBoxComponent* boxComp;

    UPROPERTY(EditAnywhere)
    class UStaticMeshComponent* meshComp;
};
```

Enemy 제작

- Enemy에 대한 C++ class와 블루프린트 생성
- Enemy Factory에 대한 C++ class와 블루프린트 생성
- 속력, 충돌 함수 구현
- Enemy 생성에 대한 확률 구현
- Player 기준 생성된 Enemy 방향 벡터 변수 선언



Enemy 주요 함수

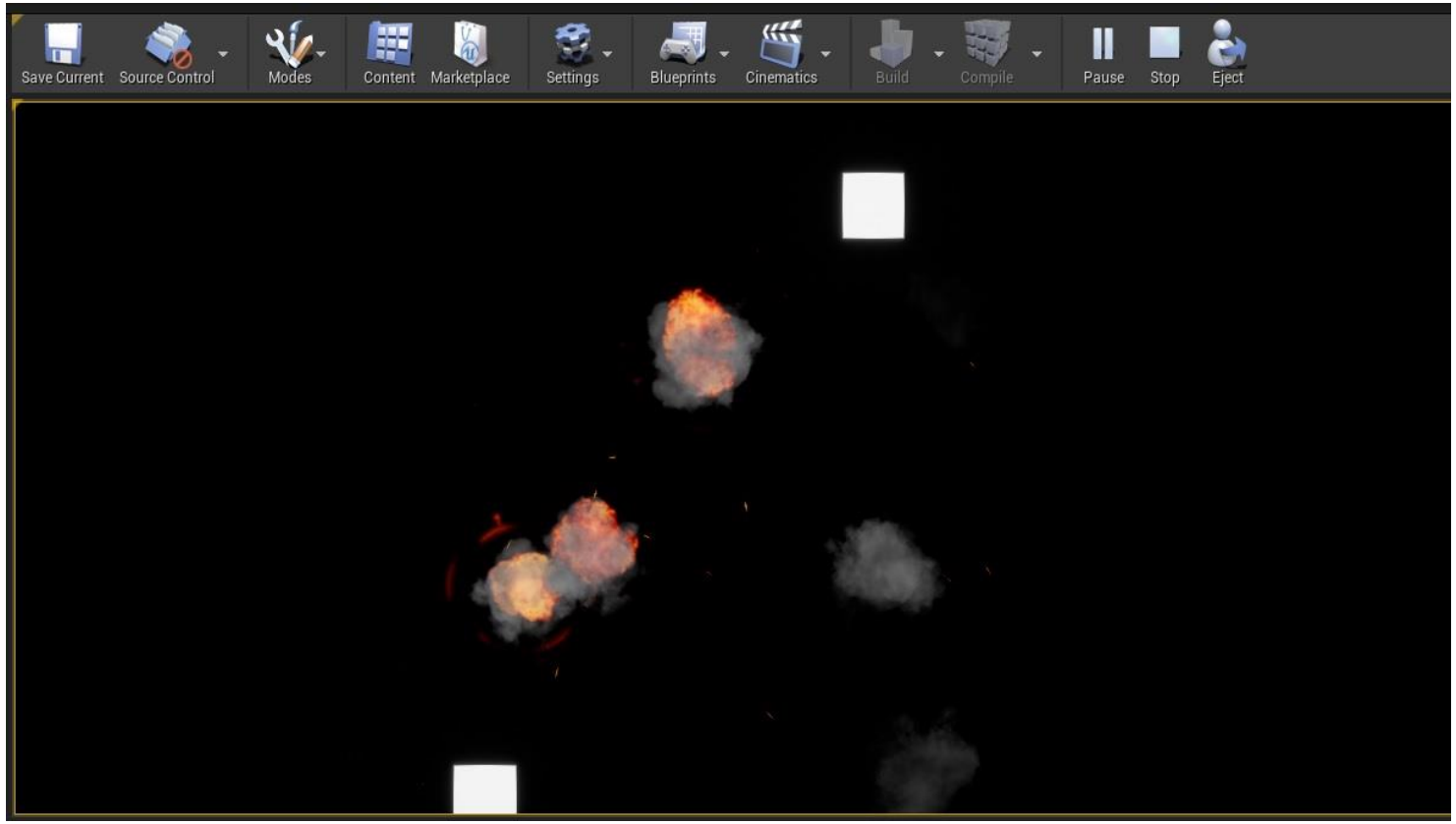
```cpp
// Called when the game starts or when spawned
void AEnemyActor::BeginPlay()
{
    Super::BeginPlay();

    // 1~100 사이의 임의의 정수 값을 추첨한다.
    int32 drawResult = FMath::RandRange(1, 100);

    // 만일, 추첨된 값이 추적 확률 변수보다 작거나 같다면
    if (drawResult <= traceRate)
    {
        // 월드 공간에 APlayerPawn 클래스로 된 액터를 모두 검색한다.
        for (TActorIterator<APlayerPawn> player(GetWorld()); player; ++player)
        {
            // 만일 검색된 액터의 이름에 'BP_PlayerPawn'이란 문구가 포함되어 있다면
            if (player->GetName().Contains(TEXT("BP_PlayerPawn")))
            {
                // 플레이어 액터의 위치 - 자신의 위치
                dir = player->GetActorLocation() - GetActorLocation();
                dir.Normalize();
            }
        }
    }
    // 그렇지 않다면 정면 방향 벡터를 생성한다.
    else
    {
        dir = GetActorForwardVector();
    }
}
```

```cpp
#include "EnemyFactory.h"
#include "EnemyActor.h"

// Sets default values
AEnemyFactory::AEnemyFactory() { ... }

// Called when the game starts or when spawned
void AEnemyFactory::BeginPlay() { ... }

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

    // 만일 경과된 시간이 생성할 시간을 초과했다면
    if (currentTime > delayTime)
    {
        // 경과된 시간을 0초로 초기화 한다.
        currentTime = 0;

        // enemy 변수에 할당된 블루프린트를 자신의 위치에 생성한다.
        AEnemyActor* spawnActor = GetWorld()->SpawnActor<AEnemyActor>(enemy,
            GetActorLocation(), GetActorRotation());
    }
    // 그렇지 않다면
    else
    {
        // 현재 프레임의 경과 시간을 누적시킨다
        // currentTime = currentTime + DeltaTime
        currentTime += DeltaTime;

    }
}
```

Shooting Game

상호 충돌 구현

- 총알과 Enemy가 충돌로 인한 제거 구현
- 충돌에 의한 effect 구현
- 플레이어와 Enemy 충돌로 인한 제거 구현



Shooting Game

상호 충돌 주요 함수

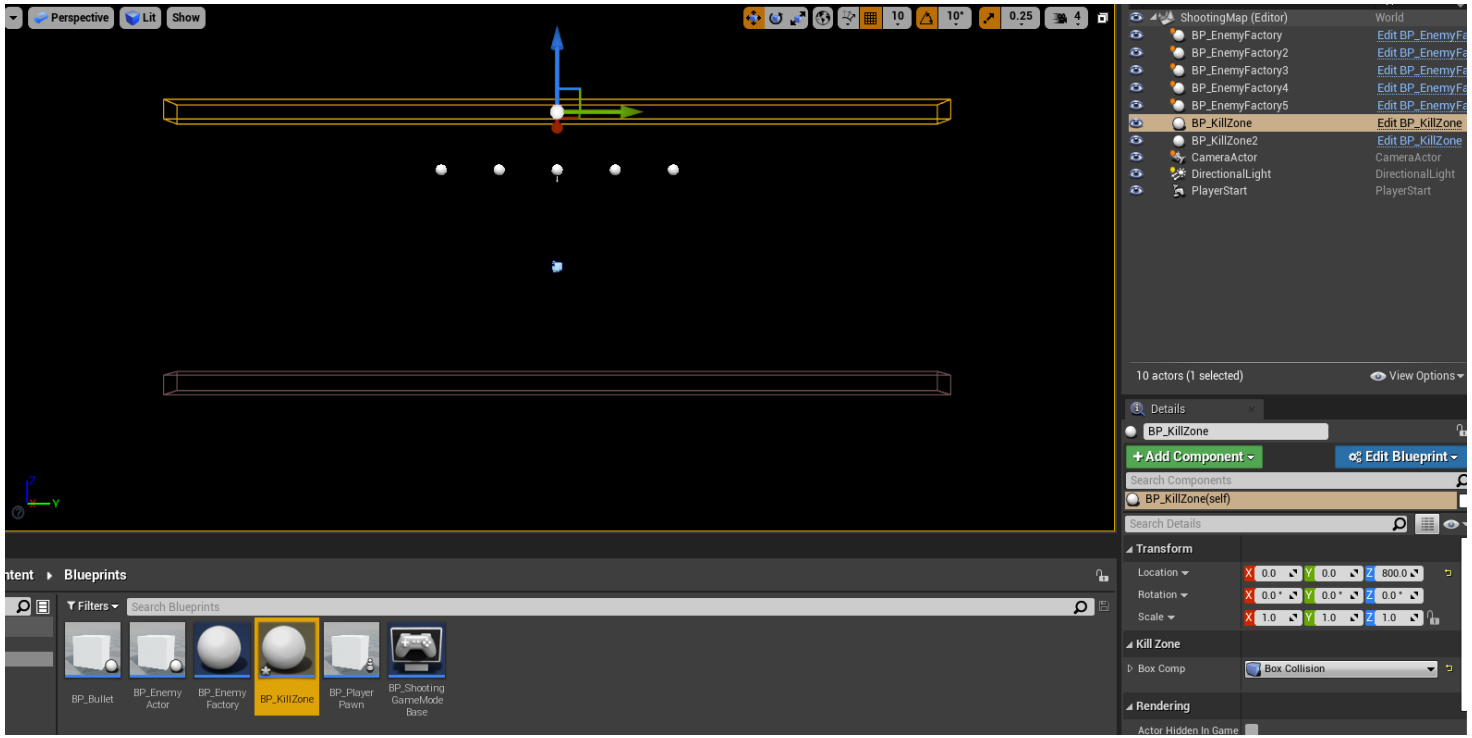
```cpp
#include "Bullet.h"
#include "Components/BoxComponent.h"
#include "Components/StaticMeshComponent.h"
#include "EnemyActor.h"

// Sets default values
ABullet::ABullet() { ... }

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

    // 박스 컴포넌트의 충돌 오버랩 이벤트에 BulletOverlap 함수를 연결한다.
    boxComp->OnComponentBeginOverlap.AddDynamic(this, &ABullet::OnBulletOverlap);
}

// Called every frame
void ABullet::Tick(float DeltaTime) { ... }


void ABullet::OnBulletOverlap(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor,
    UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult& SweepResult)
{
    // 충돌할 액터를 AEnemyActor 클래스로 변환
    AEnemyActor* enemy = Cast<AEnemyActor>(OtherActor);

    // 만일, 캐스팅이 정상적으로 되어서 AEnemy Actor 포인터 변수에 값이 있다면
    if (enemy != nullptr)
    {
        // 충돌할 액터를 제거한다.
        OtherActor->Destroy();
    }

    // 자기 자신을 제거한다.
    Destroy();
}
```

```cpp
#include "EnemyActor.h"
#include "Components/BoxComponent.h"
#include "Components/StaticMeshComponent.h"
#include "EngineUtils.h"
#include "PlayerPawn.h"

// Sets default values
AEnemyActor::AEnemyActor() { ... }

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

    // 1~100 사이의 임의의 정수 값을 추첨한다.
    int32 drawResult = FMath::RandRange(1, 100);

    // 만일, 추첨된 값이 추적 확률 변수보다 작거나 같다면
    if (drawResult <= traceRate) { ... }
    // 그렇지 않다면 정면 방향 벡터를 생성한다.
    else { ... }

    // 박스 컴포넌트의 충돌 오버랩 이벤트에 BulletOverlap 함수를 연결한다.
    boxComp->OnComponentBeginOverlap.AddDynamic(this, &AEnemyActor::OnEnemyOverlap);
}

// Called every frame
void AEnemyActor::Tick(float DeltaTime) { ... }

void AEnemyActor::OnEnemyOverlap(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor,
    UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult& SweepResult)
{
    // 충돌할 액터를 APlayerPawn 클래스로 변환을 시도
    APlayerPawn* player = Cast<APlayerPawn>(OtherActor);

    // 만일, 캐스팅이 정상적으로 되어서 AEnemy Actor 포인터 변수에 값이 있다면
    if (player != nullptr)
    {
        // 충돌할 액터를 제거한다.
        OtherActor->Destroy();
    }

    // 자기 자신을 제거한다.
    Destroy();
}
```

Shooting Game

Killzone 제작

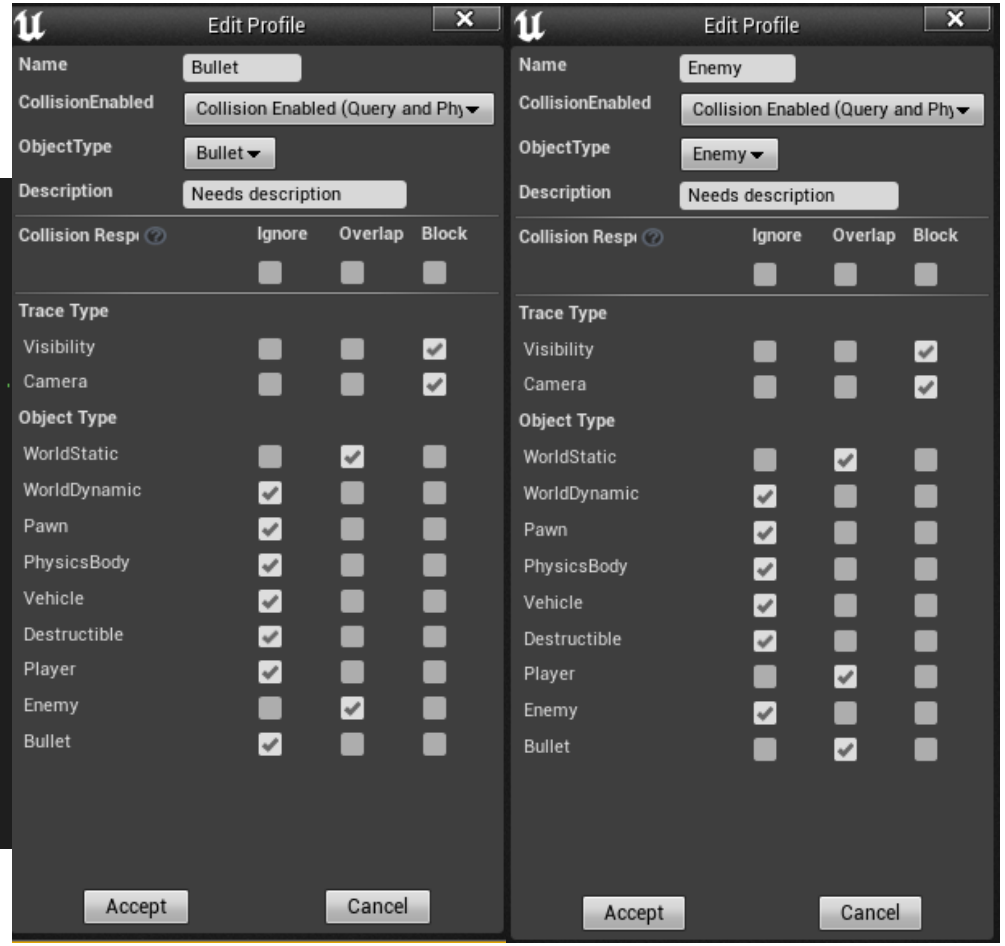
- 시점에서 벗어난 적들을 제거 목적
- 플레이어가 일정 시점을 유지 목적
- 플레이어와 Enemy에 대한 worldstatic 충돌 함수 구현



Killzone 주요 함수

```
UPROPERTY(EditAnywhere)
class UBoxComponent* boxComp;
```

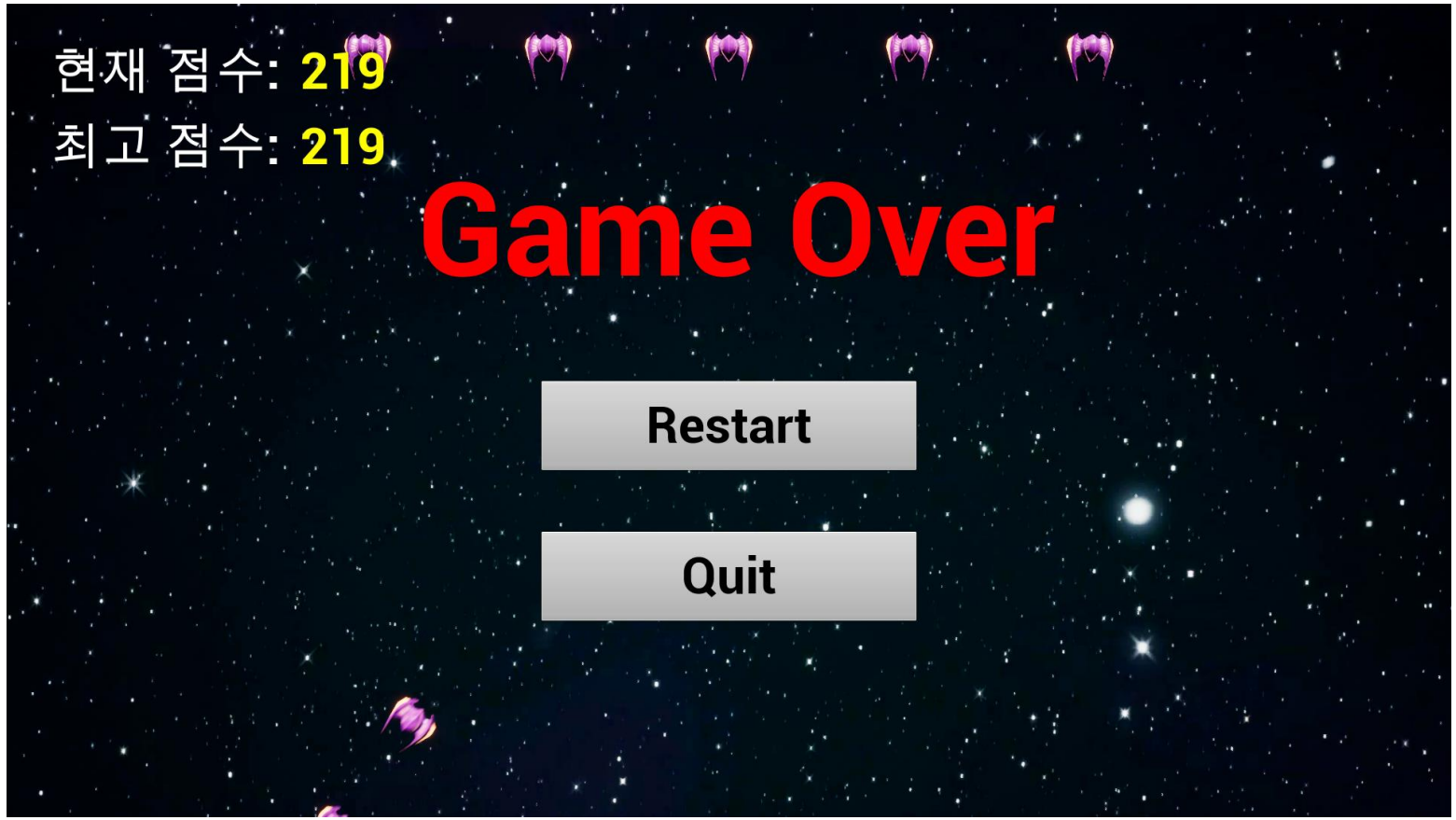
```
#include "KillZone.h"
#include "Components/BoxComponent.h"

// Sets default values
AKillZone::AKillZone()
{
    // Set this actor to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
    PrimaryActorTick.bCanEverTick = true;

    // 박스 컴포넌트를 생성하고 루트 컴포넌트로 설정한다.
    boxComp = CreateDefaultSubobject<UBoxComponent>(TEXT("Box Collider"));
    SetRootComponent(boxComp);

    // 박스 컴포넌트의 모빌리티를 고정 상태로 설정한다.
    boxComp->SetMobility(EComponentMobility::Static);

    // 박스의 크기를 50, 2,000, 50으로 설정한다.
    boxComp->SetBoxExtent(FVector(50, 2000, 50));

    // 박스 컴포넌트의 콜리전 프리셋을 KillZone으로 설정한다
    boxComp->SetCollisionProfileName(TEXT("KillZone"));
}
```



Shooting Game

UI 및 Widget 제작

- GameOver 시 Restart / Quit 구현
- 점수 구현
- 실감나는 게임을 위한 배경 스크롤
- 플레이어와 Enemy 모델링



Shooting Game

UI 및 Widget 주요 함수

```h
#include "CoreMinimal.h"
#include "GameFramework/GameModeBase.h"
#include "ShootingGameModeBase.generated.h"

/**
 *
 */
UCLASS()
class SHOOTINGCPP_API AShootingGameModeBase : public AGameModeBase
{
    GENERATED_BODY()

public:
    void AddScore(int32 point);

    UPROPERTY(EditAnywhere)
    TSubclassOf<class UMainWidget>mainWidget;

    UPROPERTY(EditAnywhere)
    TSubclassOf<class UMenuWidget>menuWidget;

    void ShowMenu();

protected:
    virtual void BeginPlay() override;

private:
    // 현재 점수 저장용 변수
    int32 currentScore = 0;

    // 현재 뷰 포트에 로드된 위젯 저장용 변수
    class UMainWidget* mainUI;

    class UMenuWidget* menuUI;

    void PrintScore();
};
```

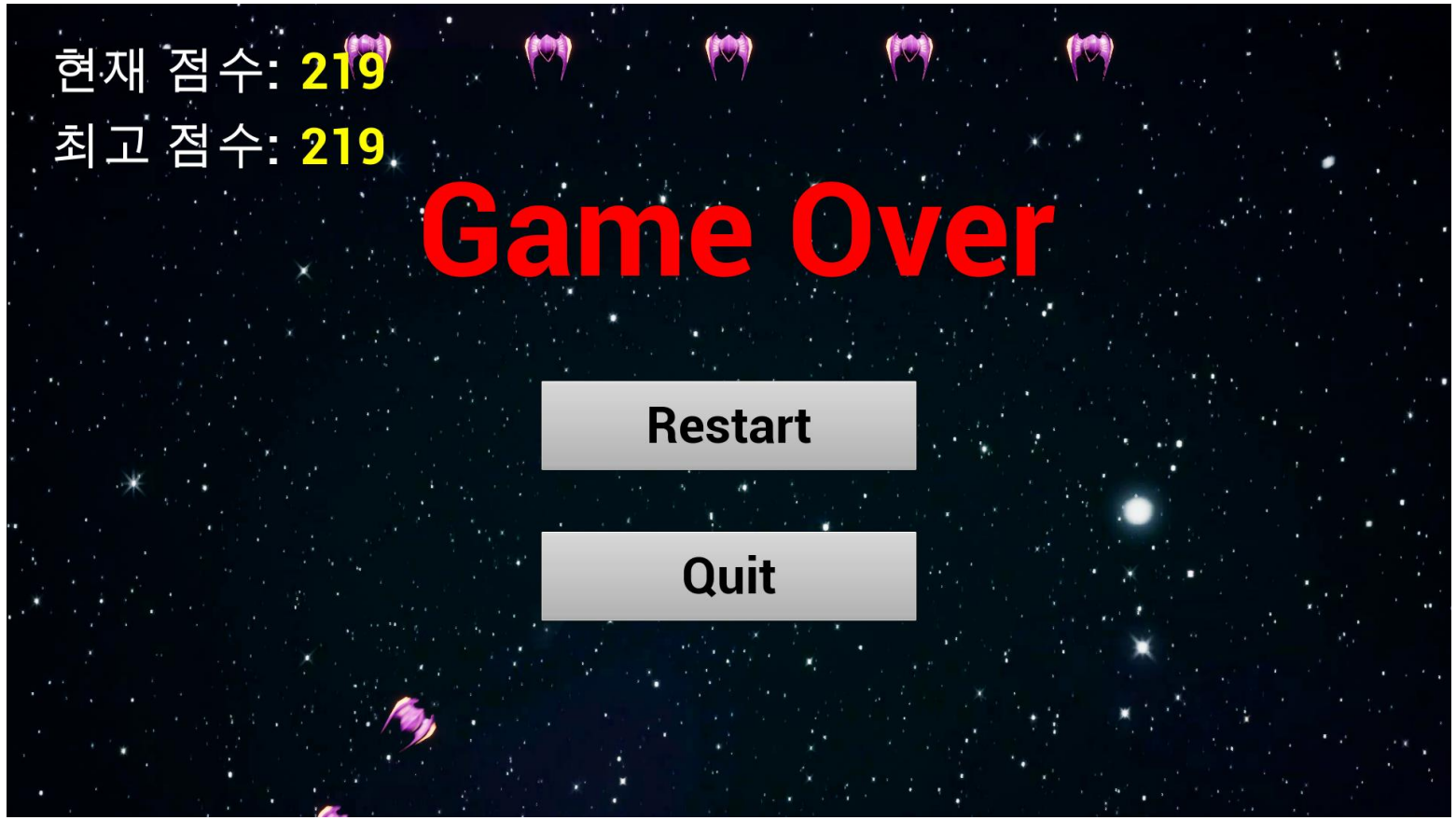
```cpp
#include "ShootingGameModeBase.h"
#include "Blueprint/UserWidget.h"
#include "MainWidget.h"
#include "Components/TextBlock.h"


void AShootingGameModeBase::PrintScore()
{
    if (mainUI != nullptr)
    {
        // scoreData 텍스트 블록에 현재 점수 값을 입력한다.
        mainUI->scoreData->SetText(FText::AsNumber(currentScore));
    }
}
void AShootingGameModeBase::BeginPlay() { ... }

// 현재 점수를 계산하는 함수
void AShootingGameModeBase::AddScore(int32 point)
{
    // 매개변수 point를 통해 넘겨받은 점수를 현재 점수에 누적한다.
    currentScore += point;

    // 현재 점수를 위젯에 반영한다
    PrintScore();
}
```

```cpp
#include "ShootingGameModeBase.h"
#include "Blueprint/UserWidget.h"
#include "MainWidget.h"
#include "Components/TextBlock.h"
#include "MenuWidget.h"


void AShootingGameModeBase::PrintScore() { ... }
void AShootingGameModeBase::BeginPlay() { ... }

// 현재 점수를 계산하는 함수
void AShootingGameModeBase::AddScore(int32 point) { ... }

void AShootingGmaeModeBase::ShowMenu()
{
    if (menuWidget != nullptr)
    {
        // 메뉴 위젯을 생성한다.
        menuUI = CreateWidget<UMenuWidget>(GetWorld(), menuWidget);
    }

    if (menuUI != nullptr)
    {
        // 생성한 메뉴 위젯을 뷰 포트에 출력한다.
        menuUI->AddToViewport();
    }
}
```

Shooting Game

개선점

- 단순한 슈팅 게임에도 불구하고 큰 용량
- 단순 기능만 구현이 됨
- 기획 단계의 부족



읽어 주셔서 감사합니다 :)