포트폴리오

프로젝트 – UnrealEngine5 : TPS 보스레이드 구현

개발 기간	2022.12.16 ~ 2023.2.14/ 2023.2.28 ~
개발 환경	Unreal Engine5.0 / C++ / Visual Studio 2019
개발 인원	1명
요 약	UnrealEngine을 이용한 TPS 시점의 보스레이드 구현
목표	**1. C++ 디자인패턴 이론 실제 적용점 찾기 및 구현** **2. 벡터, 내적, 외적, 행렬계산 등의 게임수학 학습** **3. 웰논 알고리즘 실제 적용점 찾기 및 구현** **4. 언리얼 엔진 기초 사용법 숙달**
실행 화면	
https://gogogamegaga.tistory.com/60	

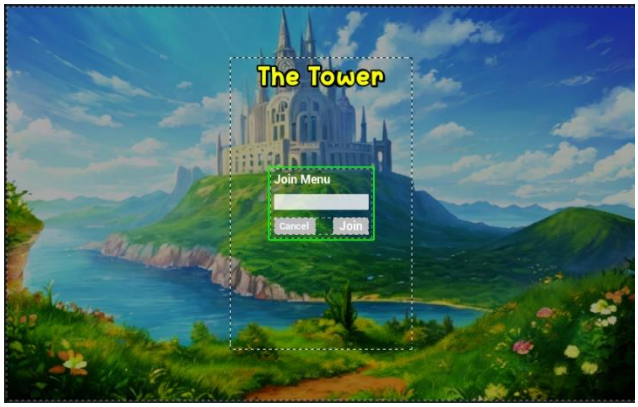
1.1 요구사항

1. TPS 시점 캐릭터의 무브, 공격 등의 애니메이션
2. 적절한 VFX 리소스
3. 라이브러리 사용은 최소화. 만약 한다면 내부 로직 이해하고 넘어갈 것
4. 응집도 최대한 낮추기
5. 호스팅 기능
6. 호스팅 한 방에 Join 기능 (미완)

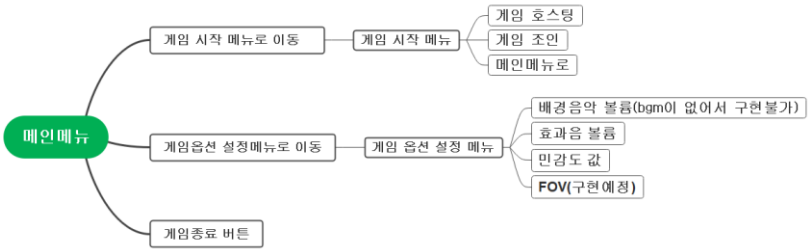
1. 호스팅 기능 – 메뉴

참고	https://gogogamegaga.tistory.com/10 https://gogogamegaga.tistory.com/63

1-1. 유저 인터페이스



메뉴의 구조는 아래와 같다.



Host 버튼을 누를 시 사용자는 자신의 네트워크 주소로 게임을 호스팅

- 상속구조는 MenuWidget.cpp -> MainMenu.cpp -> BP_MainMenu 이루어져있음
- MenuWidget.cpp은 UI가 뷰포트에 추가되는것과 제거되는 기능만 있음
- MenuWidget.cpp에 MenuInterface.cpp 포인터 형식의 변수를 가지고 있음
- MenuInterface.cpp의 인스턴스 생성과 실제 구현부는 GameInstance.cpp에서 이루어짐
    - 선언부와 정의부를 분리하기 위함
    - 호스팅과 같은 서버 기능은 GameMode나 GameInstance에서 실행되는게 맞다고 판단하였음
    - **의존관계 역전원칙 (DIP) 준수**

MenuWidget.h Code

```
*/
class IMenuInterface;

UCLASS()
class SHOOTWITHBOSS_API UMenuWidget : public UUserWidget
{
    GENERATED_BODY()
public:

    void Setup(); // 메뉴를 보이게 하기 위한 함수
    void Teardown(); // 메뉴를 안보이게 하기 위한 함수

    void SetMenuInterface(IMenuInterface* pMenuInterface); //GameInstance에서 MenuInterface를 정의하기 위한 함수

protected:
    IMenuInterface* MenuInterface; // 메뉴 인터페이스 선언, 구현부는 GameInstance에서
};
```

MenuWidget.cpp Code

```
#include "MenuWidget.h"

//GameInstance에서 MenuInterface를 정의하기 위한 함수
void UMenuWidget::SetMenuInterface(IMenuInterface* pMenuInterface)
{
    this->MenuInterface = pMenuInterface;
}

// 메뉴를 보이게 하기 위한 함수
void UMenuWidget::Setup()
{
    this->AddToViewport();

    UWorld* World = GetWorld();

    if (!ensure(World != nullptr)) return;

    APlayerController* PlayerController = World->GetFirstPlayerController();
    if (!ensure(PlayerController != nullptr)) return;

    FInputModeUIOnly InputModeData;
    InputModeData.SetWidgetToFocus(this->TakeWidget());
    InputModeData.SetLockMouseToViewportBehavior(EMouseLockMode::DoNotLock);

    PlayerController->SetInputMode(InputModeData);

    PlayerController->bShowMouseCursor = true;


}
```

```cpp
// 메뉴를 안보이게 하기 위한 함수
void UMenuWidget::Teardown()
{
    this->RemoveFromViewport();

    UWorld* World = GetWorld();

    if (!ensure(World != nullptr)) return;

    APlayerController* PlayerController = World->GetFirstPlayerController();
    if (!ensure(PlayerController != nullptr)) return;

    FInputModeGameOnly InputModeData;
    PlayerController->SetInputMode(InputModeData);


    PlayerController->bShowMouseCursor = false;
}
```

MenuWidget 설명

가장먼저 메인메뉴를 만들어주었으나 앞으로도 메뉴를 많이 만들거라 판단.

따라서 모든 메뉴라면 당연히 있어야할 뷰포트에 설치, 제거하는 기능을 추상화하여 MenuWidget에 작성해 주었음.

MainMenu.h Code (일부)

```cpp
UCLASS()
class SHOOTWITHBOSS_API UMainMenu : public UMenuWidget
{
    GENERATED_BODY()


protected:
    virtual bool Initialize();
private:
```

```cpp
UPROPERTY(meta = (BindWidget))
    class UButton* JoinJoinMenuButton;

UFUNCTION()
    void HostServer(); // Host 버튼 누를시 실행되는 함수

UFUNCTION()
    void OpenJoinMenu(); // Join 버튼 누를시 Join 메뉴로 전환되는 함수

UFUNCTION()
    void OpenMainMenu(); // 메인메뉴로 전환되는 함수

UFUNCTION()
    void JoinMainGame(); // Join the Game시 호스팅된 게임으로 참가하는 함수

UFUNCTION()
    void QuitPressed(); // 취소 누를시 실행되는 함수
};
```

MainMenu.cpp Code (일부)

```cpp
void UMainMenu::HostServer()
{

    if (MenuInterface != nullptr)
    {
        UE_LOG(LogTemp, Warning, TEXT("Onclick Host"));

        MenuInterface->Host(); //GameInstance::Host() 실행
    }
}
```

MainMenu 설명

HostServer() 함수는 후술할 GameInstance.cpp 에서 구현한 Host()함수를 호출해주는 것으로 디자인되었음.

왜냐하면 일단 호스팅은 메뉴 클래스에서 실행되는 것 보단 GameInstance에서 실행되는게 논리적으로 말이 된다고 생각하였음

GameInterface의 Host()함수 구현부와 연결해주는 기능.

나머지 함수들은 단순 기능 구현이기에 설명 생략.

GameInstnace.h의 메뉴와 관련된 부분 Code

```cpp
UCLASS()
class SHOOTWITHBOSS_API UShootWithBossGameInstance : public UGameInstance, public IMenuInterface
{
    GENERATED_BODY()
public:
    UShootWithBossGameInstance(const FObjectInitializer& ObjectInitializer);
    virtual void Init();

    UFUNCTION()
    void Host() override; // MenuInterface의 Host 순수가상함수를 상속

    UFUNCTION()
    void Join(const FString& Address) override; // MenuInterface의 Join 순수가상함수를 상속

    UFUNCTION(BlueprintCallable)
    void LoadMenu(); // 메인메뉴 로드

    UFUNCTION(BlueprintCallable)
    void InGameLoadMenu(); // 인게임 내에서의 메뉴 로드. (메인메뉴 아님)
    UFUNCTION(BlueprintCallable)
    virtual void LoadMainMenu() override; // 메인 메뉴로 돌아가는 기능
```

```cpp
*/
struct Options
{
    float BackgroundVolume;
    float EffectVolume;
    float SenseAbility;
    float FOV;
};
```

GameInstnace.cpp의 메뉴와 관련된 부분 Code

```cpp
// 메인메뉴 로드
void  UShootWithBossGameInstance::LoadMenu()
{
    if (!ensure(MenuClass != nullptr)) return;
    Menu = CreateWidget<UMenuWidget>(this, MenuClass);

    if (!ensure(Menu != nullptr)) return;

    Menu->Setup(); // MenuWidget의 메뉴 뷰포트에 추가
    Menu->SetMenuInterface(this); // Menu 구현부 연결
}

// 인게임 메뉴 로드
void UShootWithBossGameInstance::InGameLoadMenu()
{
    if (!ensure(InGameMenuClass != nullptr)) return;
    InGameMenu = CreateWidget<UMenuWidget>(this, InGameMenuClass);

    UE_LOG(LogTemp, Warning, TEXT("InGameLoadMenu"));

    if (!ensure(InGameMenu != nullptr)) return;

    InGameMenu->Setup(); // MenuWidget의 메뉴 뷰포트에 추가
    InGameMenu->SetMenuInterface(this); // Menu 구현부 연결
}
```

```cpp
// 순수가상함수 MenuInterface::Host()의 구현부
void  UShootWithBossGameInstance::Host()
{
    if (Menu != nullptr) {
        Menu->Teardown(); // 인게임내로 들어가기전, MenuWidget의 메뉴 뷰포트에서 제거를 수행
    }


    UEngine* Engine = GetEngine();
    if (!ensure(Engine != nullptr)) return;

    Engine->AddOnScreenDebugMessage(0, 2, FColor::Green, TEXT("Hosting"));

    UWorld* World = GetWorld();
    if (!ensure(World != nullptr)) return;

    World->ServerTravel("/Game/ThirdPerson/Maps/ThirdPersonMap_2?listen");


}
```

```cpp
// 순수가상함수 MenuInterface::Join()의 구현부
void  UShootWithBossGameInstance::Join(const FString& Address)
{
    if (Menu != nullptr) {
        Menu->Teardown();
    }


    UEngine* Engine = GetEngine();
    if (!ensure(Engine != nullptr)) return;
    Engine->AddOnScreenDebugMessage(0, 2, FColor::Green, FString::Printf(TEXT("Joining %s"), *Address));

    ////APlayerController* PlayerController = GetFirstLocalPlayerController();
    ////if (!ensure(PlayerController != nullptr)) return;

    ////PlayerController->ClientTravel(Address, ETravelType::TRAVEL_Absolute);


}
```

GameInstnace의 메뉴와 관련된 부분 설명
MenuInterface의 순수가상함수의 정의부를 GameInstance에서 구현하였음 Join 기능은 가정용 DHCP를 사용해서 그런지 연결이 잘 안되는 현상. 아직 미완성 또한 옵션 메뉴에서 설정된 값들은 Options 구조체에 저장됨 (초기화는 무조건 50.0f) 이 설정값이 필요해질때면 GameInstance에서 불러오는것으로 설계하였음. 나머지 함수는 단순 구현이므로 생략.

2. 플레이어 Hp바, Mp바

참고	https://gogogamegaga.tistory.com/19

2-1. 유저인터페이스

- 프로그래스바 2개로 Hp바와 Mp바를 최대한 간단하게 구현

Character.h 의HP바, MP바 관련 부분 Code

```
public:
    UFUNCTION(BluePrintCallable)
    int getHp();
    UFUNCTION(BluePrintCallable)
    void setHp(const int& n);
    UFUNCTION(BluePrintCallable)
    int getMp();
    UFUNCTION(BluePrintCallable)
    void setMp(const int& n);
```

Character.cpp 의HP바, MP바 관련 부분 Code

```
//플레이어의 Mp를 조절하고 프로그래스바의 칸을 조절
void AShootWithBossCharacter::setMp(const int& n)
{
    UShootWithBossGameInstance* instance = Cast<UShootWithBossGameInstance>(GetGameInstance());
    UCharacterUI* CharacterUI = instance->GetCharacterUI();
    CharacterUI->MpBar->SetPercent(n/100.0f);
}
//플레이어의 Hp를 조절하고 프로그래스바의 칸을 조절.
void AShootWithBossCharacter::setHp(const int& n)
{
    hp = n;
    UShootWithBossGameInstance* instance = Cast<UShootWithBossGameInstance>(GetGameInstance());
    UCharacterUI* CharacterUI = instance->GetCharacterUI();
    CharacterUI->HpBar->SetPercent(n / 100.0f);

    //만약 Hp가 0이 되면 사망
    if (hp <= 0)
    {
        DoDie();

    }
}
```

```cpp
void AShootWithBossCharacter::doingSprinting()
{
    // 만약 달리고 있을 경우
    if (bIsSprint)
    {
        // 만약 mp가 0 이상이라면
        if (mp > 0)
        {
            setMp(mp - 1);
            mp -= 1;
        }
        // mp가 0이 된다면
        if (mp == 0)
        {
            SprintEnd();
        }
    }
    else // 달리고 있지 않다면
    {
        // mp 회복
        if (mp < 100)
        {
            setMp(mp + 1);
            mp += 1;
        }
    }

}
```

```cpp
void  AShootWithBossCharacter::OnHit(UPrimitiveComponent* HitComponent, AAc
{
    FString tmp = *OtherActor->GetActorLabel();
    UE_LOG(LogTemp, Warning, TEXT("%s"), *OtherActor->GetActorLabel());

    if (tmp.Compare("Bullet") == 0)
    {
        OtherActor->Destroy();
        hp -= 10.0f;
        setHp(hp);
        return;
    }
```

Character의HP바, MP바 관련 부분 설명

캐릭터에서 hp 변경점이 있다면 GameInstance에서 해당 플레이어의 프로그래스바 정보를 불러온 후 프로그래스바를 변경한다.

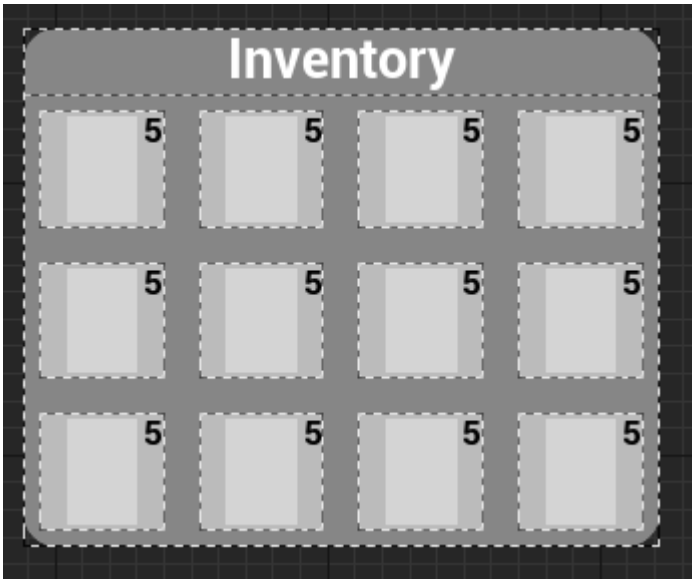
플레이어는 달릴 때 SetMp()함수가 호출되며 mp가 없다면 달릴수 없게 된다.

플레이어는 몬스터의 공격에 피격 될 경우 SetHp()함수가 호출되며 hp가 줄어든다

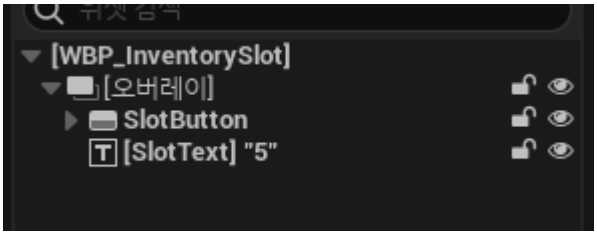
3. 인벤토리 구현

참고	https://gogogamegaga.tistory.com/58

3-1 유저 인터페이스



- WBP_Inventory는 WBP_InventorySLot으로 이루어져있다.



- WBP_InventorySlot은 아이템의 이미지를 담는 이미지와 개수를 담는 Text 그리고 클릭이 가능하게 하는 Button으로 이루어져있다.

- WBP_InventorySlot이 모여서 하나의 인벤토리 UI를 구성해주었다.

Inventory.h Code

```
// 인벤토리 배열의 형식 : 아이템 ID와 개수를 저장함
USTRUCT(Atomic, BlueprintType)
struct SHOOTWITHBOSS_API FItemInfo
{
    GENERATED_BODY()
public:
    UPROPERTY(BlueprintReadWrite)
        int ItemID;
    UPROPERTY(BlueprintReadWrite)
        int ItemCount;


};
```

```cpp
UCLASS()
class SHOOTWITHBOSS_API UInventory : public UUserWidget
{
    GENERATED_BODY()

protected:

public:
    UPROPERTY(BlueprintReadWrite)
    TArray<FItemInfo> Inventory; // 인벤토리 배열
};
```

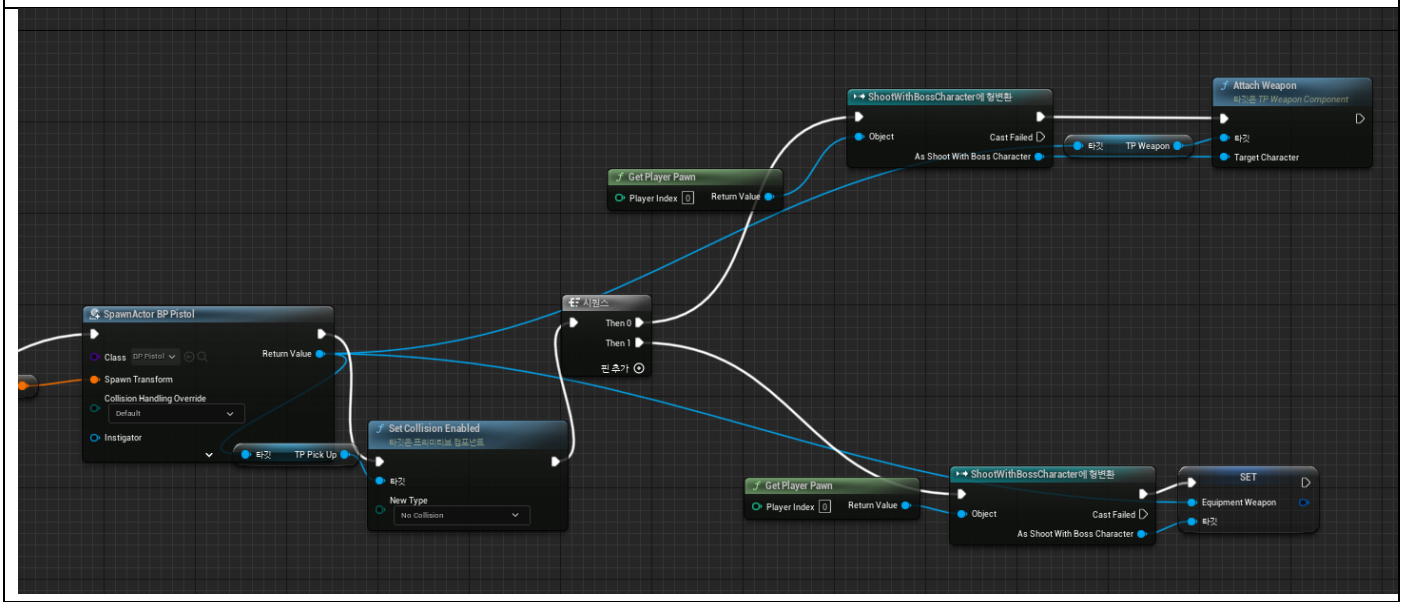
Inventory 설명

가장 먼저 인벤토리 안에 들어가야할 정보들을 구조체로 정리해주었다.

구조체는 아이템의 종류, 그리고 아이템의 개수를 저장할수 있게 해두었다.

그리고 Inventory는 배열 형식으로 단순하게 처리해주었다.

WBP_InventorySlot의 주요 Code



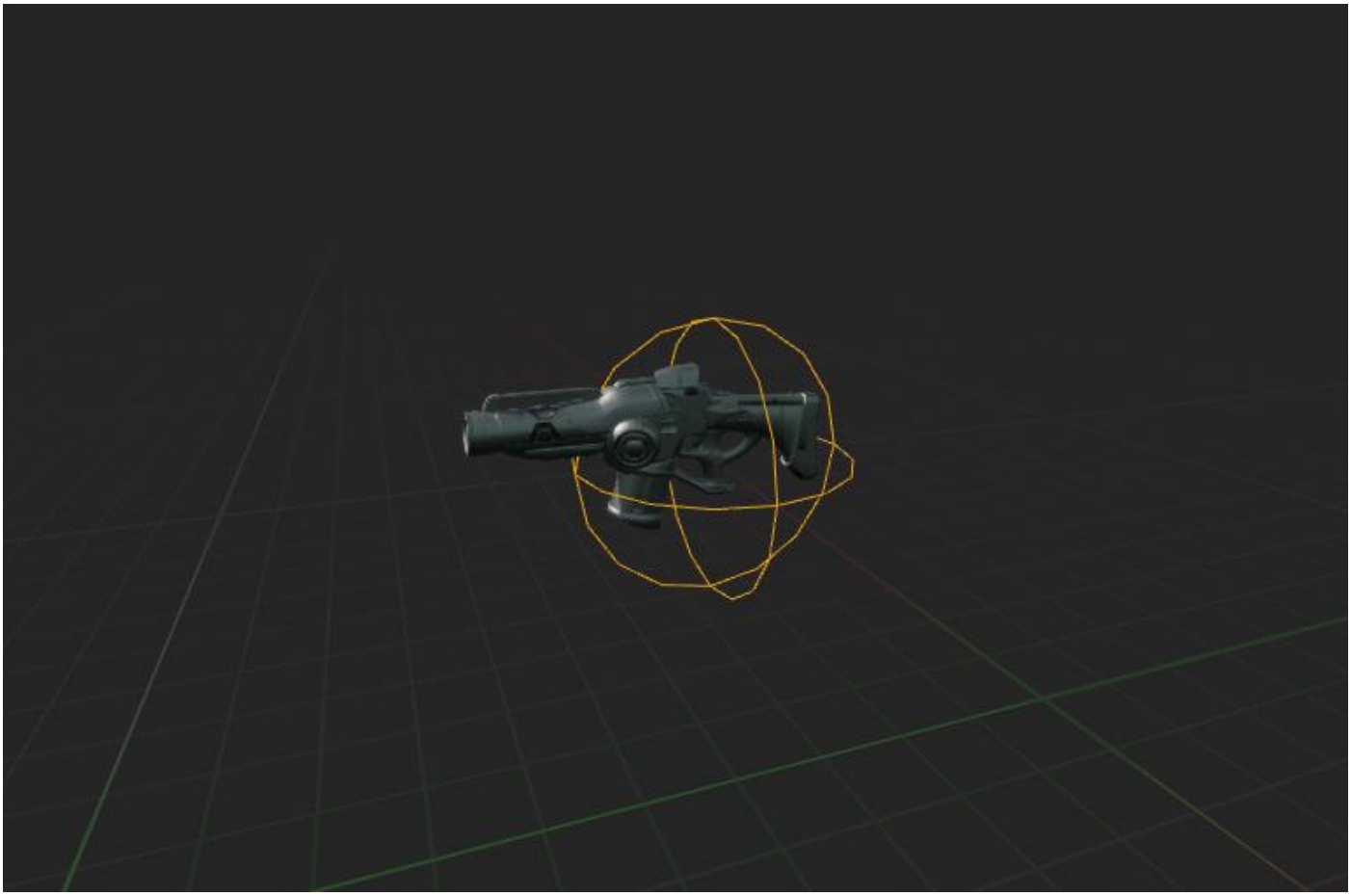
Inventory 설명

클릭할 경우 만약 총이라면 플레이어에게 부착시킴

부착시키는 동시에 장착장비를 알려주는 구조체 Equipment 구조체의 Weapon 멤버변수에 현재 무기를 넣어준다.

4. 무기 구현

참고	https://gogogamegaga.tistory.com/61



- 무기는 플레이어와 상호작용 하는 컴포넌트 (PickUpComponent), 무기의 기능을 하는 컴포넌트 (WeaponComponent)로 이루어져있다.
- PickUpComponent는 플레이어와 부딪히면 플레이어의 인벤토리에 해당 총을 추가시킨다.
- WeaponComponent는 총이 하는 기능을 수행하고 부가 옵션들을 설정한다
    - 총알 종류
    - 총의 연사속도
    - 총알발사 기능
    - 플레이어의 발사 델리게이트 함수에 해당 총의 발사 시퀀스를 달아주기

PickUpComponent.h 의 Code

```h
#pragma once

#include "CoreMinimal.h"
#include "Components/SphereComponent.h"
#include "ShootWithBossCharacter.h"
#include "TP_PickUpComponent.generated.h"

//무기를 주워들었을때 실행될 델리게이트의 선언. 매개변수는 누가 집어들었는지 알려주는 캐릭터 클래스의 레퍼런스
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FOnPickUp, AShootWithBossCharacter*, PickUpCharacter);

UCLASS(Blueprintable, BlueprintType, ClassGroup = (Custom), meta = (BlueprintSpawnableComponent))
class SHOOTWITHBOSS_API UTP_PickUpComponent : public USphereComponent
{
    GENERATED_BODY()

public:

    // 픽업 델리게이트 이벤트. 캐릭터가 무기에 부딪히면 이 이벤트가 실행이 됨
    UPROPERTY(BlueprintAssignable, Category = "Interaction")
        FOnPickUp OnPickUp;

    UTP_PickUpComponent();
protected:

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

    /** Code for when something overlaps this component */
    UFUNCTION()
        void OnSphereBeginOverlap(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor, UPrimitiveComponent* OtherComp, int
};
```

PickUpComponent.cpp 의 Code

```cpp
// Copyright Epic Games, Inc. All Rights Reserved.

#include "TP_PickUpComponent.h"

UTP_PickUpComponent::UTP_PickUpComponent()
{
    // Setup the Sphere Collision
    SphereRadius = 32.f;
}

void UTP_PickUpComponent::BeginPlay()
{
    Super::BeginPlay();

    OnComponentBeginOverlap.AddDynamic(this, &UTP_PickUpComponent::OnSphereBeginOverlap);
}

void UTP_PickUpComponent::OnSphereBeginOverlap(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor, UPri
{
    AShootWithBossCharacter* Character = Cast<AShootWithBossCharacter>(OtherActor);
    if (Character != nullptr)
    {
        // Character에게 이 무기를 득했다고 알려줌 이벤트
        OnPickUp.Broadcast(Character);

        // 오버랩 컴포넌트 제거
        OnComponentBeginOverlap.RemoveAll(this);

    }
}
```

PickUpComponent 설명

무기의 ActorComponent로 부착된다. 플레이어와의 오버랩 이벤트를 책임진다.

WeaponComponent.h 의 Code

```h
UCLASS( ClassGroup=(Custom), meta=(BlueprintSpawnableComponent) )
class SHOOTWITHBOSS_API UTP_WeaponComponent : public UActorComponent
{
    GENERATED_BODY()

public:
    UTP_WeaponComponent();

    // 플레이어에 무기를 부착하는 기능을 수행
    UFUNCTION(BlueprintCallable, Category = "Weapon")
        void AttachWeapon(AShootWithBossCharacter* TargetCharacter);

    // 무기가 발사체를 발사하는 기능
    UFUNCTION(BlueprintCallable, Category = "Weapon")
        void Fire();

    // 무기가 발사할 총알을 정의.
    UPROPERTY(EditDefaultsOnly, Category = Projectile)
        TSubclassOf<class  AFPSProject> ProjectileClass;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Gameplay)
        FVector MuzzleOffset;
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Gameplay)
        float CoolTime;

protected:
    virtual void BeginPlay() override;
    UFUNCTION()
    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) override;

    void achievedCoolTime();
    bool isRifleCooltime;

private:
    AShootWithBossCharacter* Character;
};
```

WeaponComponent.cpp 의 Code

```cpp
void UTP_WeaponComponent::AttachWeapon(AShootWithBossCharacter* TargetCharacter)
{
    Character = TargetCharacter;


    if (Character != nullptr)
    {
        if (!ensure(Character->Equipment.Weapon == nullptr))
        {
            Character->Equipment.Weapon->Destroy();
        }

        Character->OnUseItem.RemoveAll(this);
        UE_LOG(LogTemp, Warning, TEXT("AttachWeapon"));

        // 현재 이 무기를 플레이어의 스켈레톤 메쉬의 hand_r_Soc이름의 소켓에 붙인다.
        FAttachmentTransformRules AttachmentRules(EAttachmentRule::SnapToTarget, true);
        GetOwner()->AttachToComponent(Character->GetMesh1P(), AttachmentRules, FName(TEXT("hand_r_Soc")));

        // 캐릭터가 클릭하면 발생하는 OnUseItem 델리게이트 이벤트와 이 무기 컴포넌트의 발사 함수와 바인딩시킨다
        Character->OnUseItem.AddDynamic(this, &UTP_WeaponComponent::Fire);
    }
}
```

```cpp
void UTP_WeaponComponent::Fire()
{
    // 쿨타임이라면 발사하지 않는다
    if (isRifleCooltime)
    {
        return;
    }
    if (Character == nullptr || Character->GetController() == nullptr)
    {
        return;
    }
    if (ProjectileClass != nullptr)
    {

        UE_LOG(LogTemp, Warning, TEXT("Projectile Fire"));
        UWorld* const World = GetWorld();
        if (World != nullptr)
        {
            APlayerController* PlayerController = Cast<APlayerController>(Character->GetController());
            FRotator SpawnRotation = PlayerController->PlayerCameraManager->GetCameraRotation();
            SpawnRotation.Pitch += 2.0f;
            const FVector SpawnLocation = GetOwner()->GetActorLocation() + SpawnRotation.RotateVector(MuzzleOffset);
            FActorSpawnParameters ActorSpawnParams;
            ActorSpawnParams.SpawnCollisionHandlingOverride = ESpawnActorCollisionHandlingMethod::AdjustIfPossibleButDontSpawnIfColliding;

            World->SpawnActor<AFPSProject>(ProjectileClass, SpawnLocation, SpawnRotation, ActorSpawnParams);
            // 샷건일 경우 양쪽으로 총알 두개를 더 발사한다
            if (CoolTime == 1.1f)
            {
                SpawnRotation.Yaw -= 5.f;
                World->SpawnActor<AFPSProject>(ProjectileClass, SpawnLocation, SpawnRotation, ActorSpawnParams);

                SpawnRotation.Yaw += 10.f;
                World->SpawnActor<AFPSProject>(ProjectileClass, SpawnLocation, SpawnRotation, ActorSpawnParams);

            }

            FTimerHandle timerHandle;
            isRifleCooltime = true; //총알 쿨타임

            GetWorld()->GetTimerManager().SetTimer(timerHandle, this, &UTP_WeaponComponent::achievedCoolTime, CoolTime, false); //CoolTime초 후 쿨타임해제
        }

    }
}
```

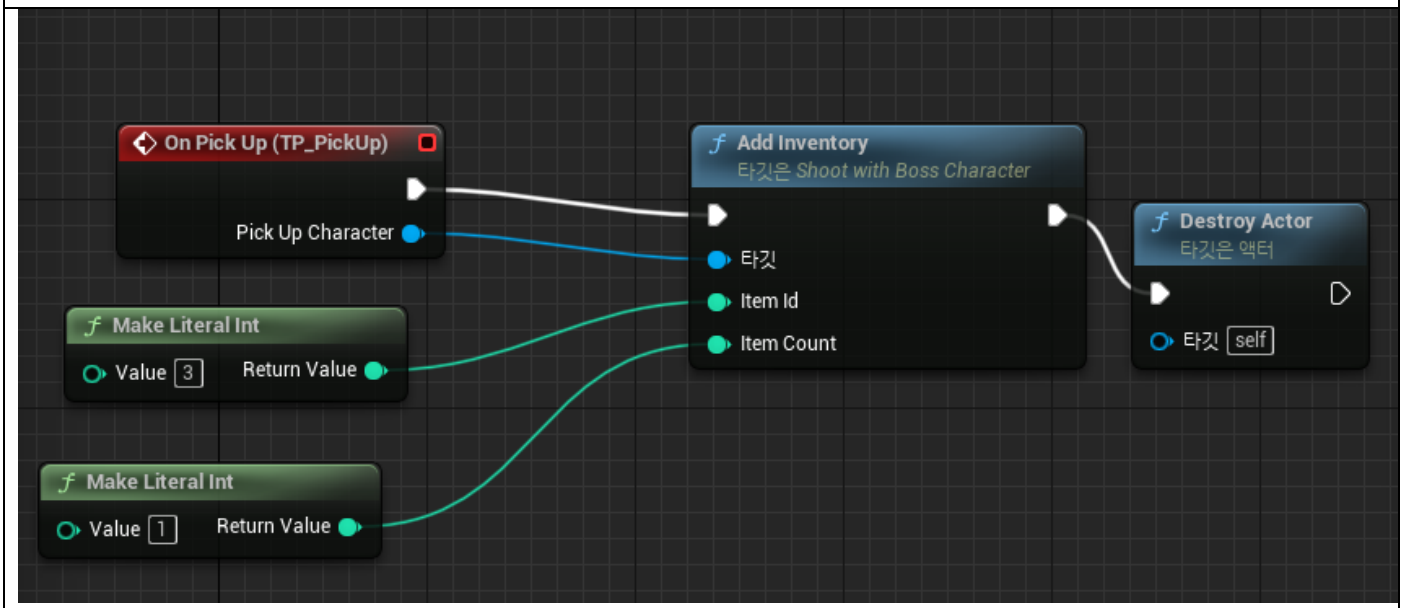
WeaponComponent.의 설명

Fire함수에서 총알이 발사되는 것을 기능한다.

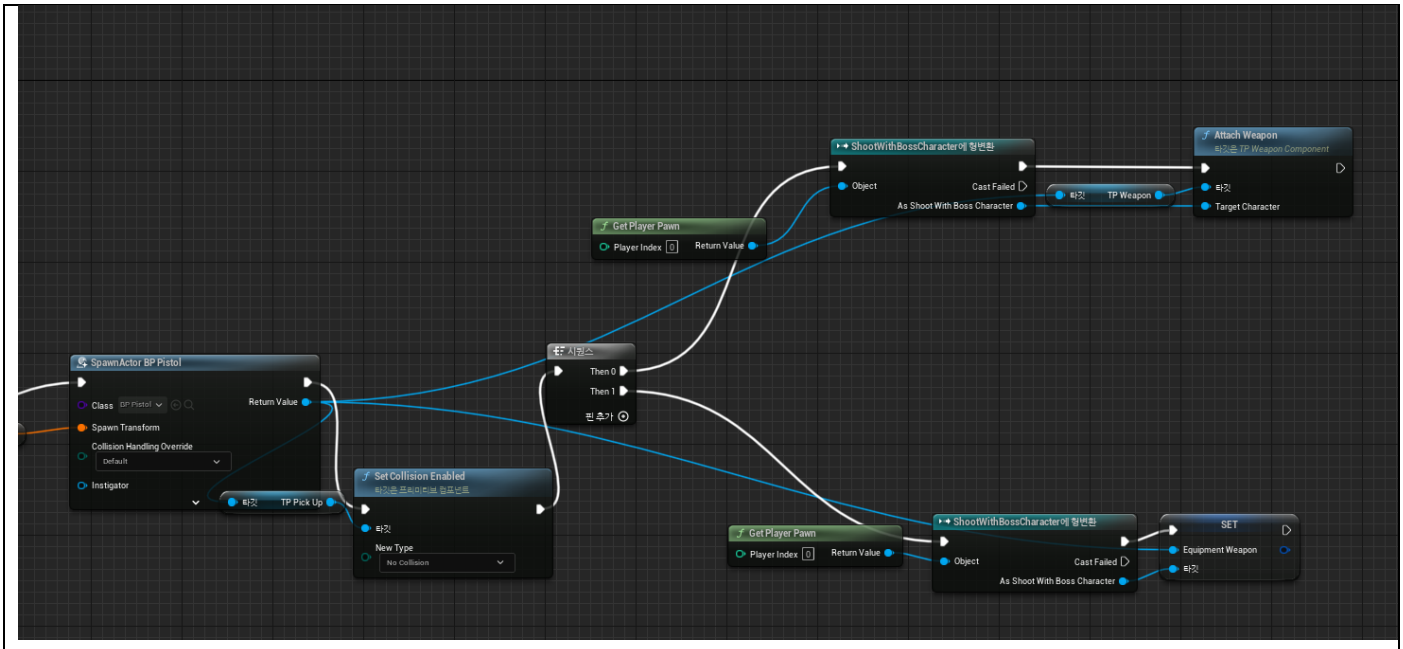
AttachWeapon은 Character의 오른손 위치에 있는 스켈레톤 메쉬의 소켓에 부착한다.

부착한 다음 Character가 마우스 왼쪽 클릭을 하면 실행되는 OnUseItem 델리게이트 이벤트에 현재 Fire함수를 바인딩 시킨다.

무기관련 블루프린트 설명



픽업 이벤트가 실행이 되면 총의 종류와 그리고 총의 개수(무조건 1개)정보를 가지고 인벤토리에 추가시킨다.



또한 InventorySlot에서 총을 클릭하면 해당 총의 정보를 토대로 총을 스폰 한 후플레이어에게 Attach해준다.

Character.h의 무기관련 code

```
// 플레이어의 스켈레톤 메쉬. 이곳의 오른쪽 손의 소켓에 접근하기 위함이다
UPROPERTY(VisibleDefaultsOnly, Category = Mesh)
    USkeletalMeshComponent* Mesh1P;
```

```
USkeletalMeshComponent* GetMesh1P() const { return Mesh1P; }
```

```
#include "GameFramework/Actor.h"
#include "Inventory.h"
#include "ShootWithBossCharacter.generated.h"
DECLARE_DYNAMIC_MULTICAST_DELEGATE(FOnUseItem);
```

```
// 플레이어가 왼쪽 마우스를 클릭하면 실행되는 델리게이트 함수
UPROPERTY(BlueprintAssignable, Category = "Interaction")
    FOnUseItem OnUseItem;
```

Character.cpp의 무기관련 code

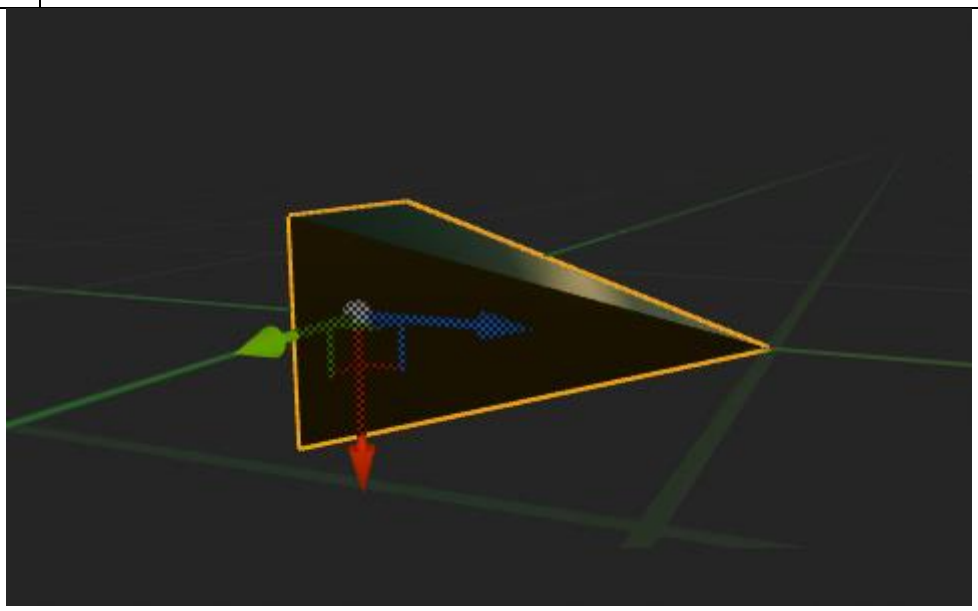
```
void AShootWithBossCharacter::Fire()
{
    OnUseItem.Broadcast();
}
```

Character 무기관련 설명

플레이어는 왼쪽 클릭을 하면 OnUseItem 델리게이트가 실행이 된다.

5. 총알 구현

참고	https://gogogamegaga.tistory.com/22 https://gogogamegaga.tistory.com/53



- 히트스캔이 아닌 투사체로 구현
- 언리얼엔진 튜토리얼 참고 많이함

https://docs.unrealengine.com/4.27/ko/ProgrammingAndScripting/ProgrammingWithCPP/CPPTutorials/FirstPersonShooter/3/

- 아직 오브젝트 풀링은 구현하지 않았음

Projectile.h Code

```
    // 프로젝타일의 속도를 발사 방향으로 초기화시키는 함수입니다.
    void FireInDirection(const FVector& ShootDirection);

    UFUNCTION(BluePrintCallable)
        void OnHit(UPrimitiveComponent* HitComponent, AActor* OtherActor, UPrimitiveCompon

    UFUNCTION(BluePrintCallable)
    void testLog();


public:
    // 구체 콜리전 컴포넌트입니다.
    UPROPERTY(VisibleDefaultsOnly, Category = Projectile)
        class USphereComponent* CollisionComponent;

    // 프로젝타일 무브먼트 컴포넌트입니다.
    UPROPERTY(VisibleAnywhere, Category = Movement)
        class UProjectileMovementComponent* ProjectileMovementComponent;

    UPROPERTY(EditDefaultsOnly, Category = Projectile)
        TSubclassOf<class AActor> CollisionVFXClass;

    //타격 사운드
    class USoundBase* Hit_Sound;

    class USoundAttenuation* Hit_Attenuation;

    // 격발 사운드
    class USoundBase* Fire_Sound;

    class USoundAttenuation* Fire_Attenuation;

private:
    FVector direction; //총알 방향

};
```

Projectile.cpp Code

```cpp
// 게임 시작시 또는 스폰시 호출됩니다.
void AFPSProject::BeginPlay()
{
    this->InitialLifeSpan = 3.0f;
    Super::BeginPlay();

    UGameplayStatics::PlaySoundAtLocation(this, Fire_Sound, GetActorLocation(), GetActorRotation(), 1.f, 1.f, 0.f, Fire_Attenuation);

}
```

```cpp
// 프로젝타일의 속도를 발사 방향으로 초기화시키는 함수입니다.
void AFPSProject::FireInDirection(const FVector& ShootDirection)
{
    direction = ShootDirection;
    ProjectileMovementComponent->Velocity = ShootDirection * ProjectileMovementComponent->InitialSpeed;
}
```

```cpp
void AFPSProject::OnHit(UPrimitiveComponent* HitComponent, AActor* OtherActor, UPrimitiveComponent* OtherComponent, int32 OtherBodyIndex, bool bFromSwee
{
    this->Destroy();
    UE_LOG(LogTemp, Warning, TEXT("%s"), *OtherActor->GetActorLabel());

    FString tmp = *OtherActor->GetActorLabel();

    if (tmp.Compare("BP_bossMonster") == 0)
    {
        ABossMonster* bm = Cast<ABossMonster>(OtherActor);
        bm->decreaseHP(50);
    }
    else if (tmp.Compare("BP_MiniMonster") == 0)
    {
        AMiniMonster* bm = Cast<AMiniMonster>(OtherActor);
        bm->decreaseHP(50);
    }


    FActorSpawnParameters SpawnParams;
    SpawnParams.Owner = this;
    SpawnParams.Instigator = GetInstigator();



    FRotator r = GetActorRotation();
    FVector_NetQuantize NormalVector = Hit.Normal;
    FVector ReflectVector = (direction.GetSafeNormal() - 2 * NormalVector * (FVector::DotProduct(direction, NormalVector))).GetSafeNormal(); //반사각

    FVector zv = FVector(0, 0, 1);
    r = ReflectVector.Rotation() - zv.Rotation(); // z 방향 rotate 상쇄


    AActor* CollisionVFX = GetWorld()->SpawnActor<AActor>(CollisionVFXClass, GetActorLocation(), r, SpawnParams);
    CollisionVFX->SetActorScale3D(FVector(0.5f)); // 그냥 이펙트 너무 큰것 같아서 크기 줄임

    UGameplayStatics::PlaySoundAtLocation(this, Hit_Sound, GetActorLocation(), GetActorRotation(), 1.f, 1.f, 0.f, Hit_Attenuation);

}
```

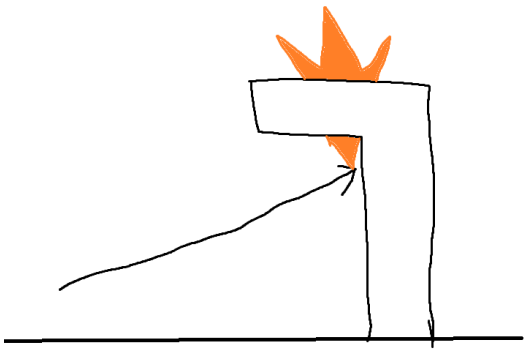
Projectile 설명

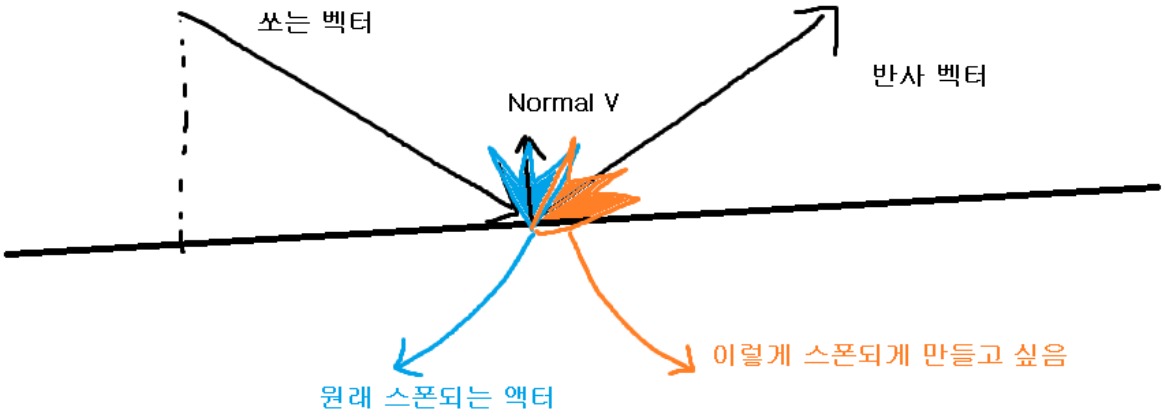
총알 발사 메커니즘은 총알을 스폰하기만 하면 해당방향으로 날라가게 된다.

그리고 총알이 생성될 때 격발음이 생성되게 하였고 총알이 부딪힐 경우 타격음이 생성되게 하였다.

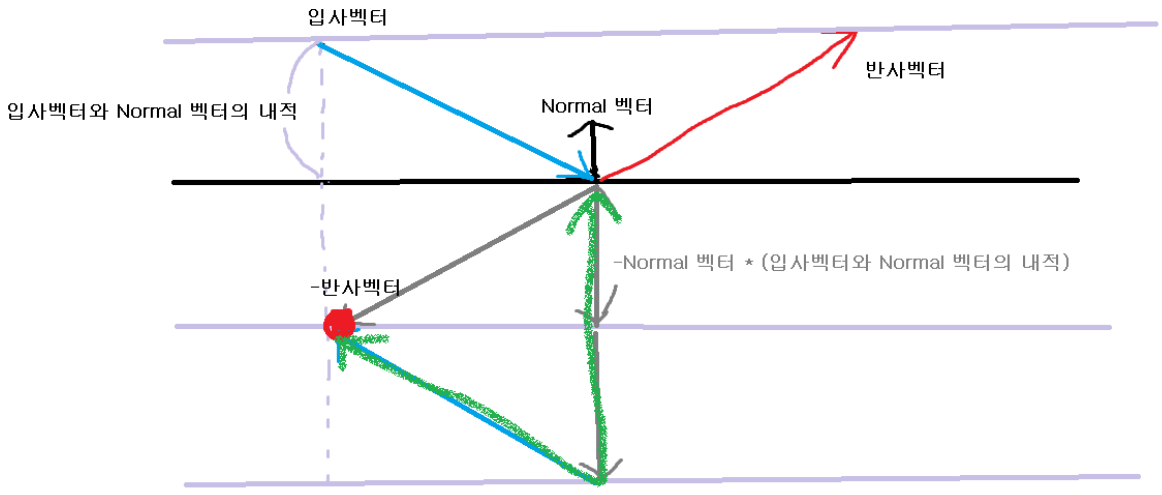
또한 총알이 부딪힐 경우 타격 이펙트가 출력되게 하였다.

이 때 아래와 같은 상황에서 타격 이펙트가 부적절하게 출력되는 현상이 있었다.





그리하여 위 그림처럼 반사벡터를 계산하여 이펙트의 Rotate값을 조정할 필요가 있었다.



반사벡터 계산과정은 위의 그림과 같으며 글로 풀이하면 다음과 같다.

1. 충돌 지점에서의 법선벡터 (검정색 화살표)를 구한 후
2. 법선벡터에다가 dot product(입사벡터, 법선벡터) 결과 스칼라값을 곱한 벡터 (회색 화살표)를 구한 후
3. 회색 화살표의 크기를 2배로 해준 다음
4. 2배가 된 회색 화살표 벡터와 -입사벡터를 빼주면 -반사벡터와 동일하게 됨.

그리고 위의 계산과정을 코드로 풀이한 부분이

```
FRotator r = GetActorRotation();
FVector_NetQuantize NormalVector = Hit.Normal;
FVector ReflectVector = (direction.GetSafeNormal() - 2 * NormalVector * (FVector::DotProduct(direction, NormalVector))).GetSafeNormal(); //반사각

FVector zv = FVector(0, 0, 1);
r = ReflectVector.Rotation() - zv.Rotation(); // z 방향 rotate 상쇄


AActor* CollisionVFX = GetWorld()->SpawnActor<AActor>(CollisionVFXClass, GetActorLocation(), r, SpawnParams);
CollisionVFX->SetActorScale3D(FVector(0.5f)); // 그냥 이펙트 너무 큰것 같아서 크기 줄임
```

이 부분이다.

또한 이펙트 자체가 Z방향으로 치솟고 있기에 올바른 회전을 위해서 Z방향 만큼의 각도를 조정해주었다.

6. 보스몬스터

6-1. 보스몬스터의 총알발사

참조	https://gogogamegaga.tistory.com/29

BossMonster.h 의 총알발사 부분 Code

```
// 투사체 사격
void ShotBullet(const FVector& startLocation, const FVector& targetLocation, const float& pspeed);
```

BossMonster.cpp 의 총알발사 부분 Code

```
void ABossMonster::ShotBullet(const FVector& startLocation, const FVector& pDirection, const float& pspeed)
{
    UWorld* world = GetWorld();
    FActorSpawnParameters spawnParameter;
    FRotator rotater;
    AActor* ABullet = world->SpawnActor<AActor>(BossBulletClass, startLocation, rotater, spawnParameter); //우선 총알 생성
    ABullet->SetActorLabel("Bullet");
    ABossMonsterBullet* bullet = static_cast<ABossMonsterBullet*>(ABullet);

    bullet->SetUpBulletOption(pDirection, pspeed); //생성직후 방향벡터와 스피드를 설정해줌


}
```

BossMonster 의 총알발사 부분 설명

미리 만들어둔 BossMonsterBullet 액터 객체를 스폰 한 직후 방향벡터와 스피드를 설정한다.

BossMonsterBullet.h Code

```
    UFUNCTION()
    void SetUpBulletOption(const FVector& pDirection, const float& speed); //총알 방향, 스피드 설정
private:

    FVector BulletStartLocation; //총알 발사 위치
    FVector Direction; // 총알 방향

    float speed; //총알의 스피드
    float damage; //총알의 데미지
};
```

BossMonsterBullet.cpp Code

```cpp
// Called every frame
void ABossMonsterBullet::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
    if (HasAuthority())
    {
        // 틱마다 이 총알의 위치를 방향만큼 변경
        SetActorLocation(GetActorLocation() + (speed * Direction * DeltaTime));

    }
}

void ABossMonsterBullet::SetUpBulletOption(const FVector& pDirection, const float& pspeed)
{
    this->speed = pspeed;
    BulletStartLocation = GetActorLocation();
    Direction = pDirection;

}
```

BossMonsterBullet 설명

보스 투사체는 매 틱당 플레이어를 향해서 일정한 속도로 날라간다.

그리고 5초 후에 사라지게 된다

그 방향과 속도를 정해주는 함수로 보스 투사체의 방향과 속도를 초기화 시켜준다

6-2. 보스몬스터의 첫번째 패턴

참조	https://gogogamegaga.tistory.com/29

BossMonster.h 의 첫번째 패턴 부분 Code

```cpp
// 패턴 명령큐 정렬법. 가중치 순대로 오름차순으로 정렬한다
struct cmp
{
    bool operator()(std::pair<int, int>& a, std::pair<int, int>& b)
    {
        return a.second < b.second;
    }
};

std::priority_queue<std::pair<int, int>, std::vector<std::pair<int, int>>, cmp> patternQueue; // 패턴명령큐
void Attack(); // 명령큐의 front를 실행

void Pattern1(); //첫번째 패턴 실행
void AddQueuePattern1(); //명령큐에 첫번째 패턴 추가
```

```cpp
void LookAt(const FVector& targetLocation, const float& DeltaTime);
```

BossMonster.cpp 의 첫번째 패턴 부분 Code

```cpp
void ABossMonster::Pattern1()
{

    doingAttacking = true;
    FVector MonsterLocation = GetActorLocation();
    FVector PlayerPosition = GetWorld()->GetFirstPlayerController()->GetPawn()->GetActorLocation();

    float speed = 2000.0f;

    // 플레이어 방향을 향해서 직진으로 한발 발사
    ShotBullet(MonsterLocation, (PlayerPosition - MonsterLocation).GetSafeNormal(), speed);

    FVector directionZ = FVector(0.0f, 0.0f, 1.0f);
    FVector forwardVector = GetActorForwardVector().GetSafeNormal();

    FVector crossPro = FVector::CrossProduct(directionZ, forwardVector).GetSafeNormal();

    // 플레이어 방향으로 부터 90도 방향으로 한발 발사
    ShotBullet(MonsterLocation, crossPro, speed);
    // 플레이어 방향으로 부터 -90도 방향으로 한발 발사
    ShotBullet(MonsterLocation, -crossPro, speed + 500.0f);
    doingAttacking = false;

}
```

```cpp
void ABossMonster::AddQueuePattern1()
{
    if (!doingAttacking)
    patternQueue.push({ 1,2 }); //첫번째 패턴이고 이건 2번째로 중요한 패턴
}
```

```cpp
// 매틱당 호출
void ABossMonster::Attack()
{
    //보스가 공격하지 않고 있을때
    if (!doingAttacking)
    {
        if (!patternQueue.empty())
        {
            std::pair<int, int> pattern = patternQueue.top();
            patternQueue.pop();

            //패턴큐의 front를 실행시킨다
            if (pattern.first == 1)
            {
                Pattern1();
            }
```

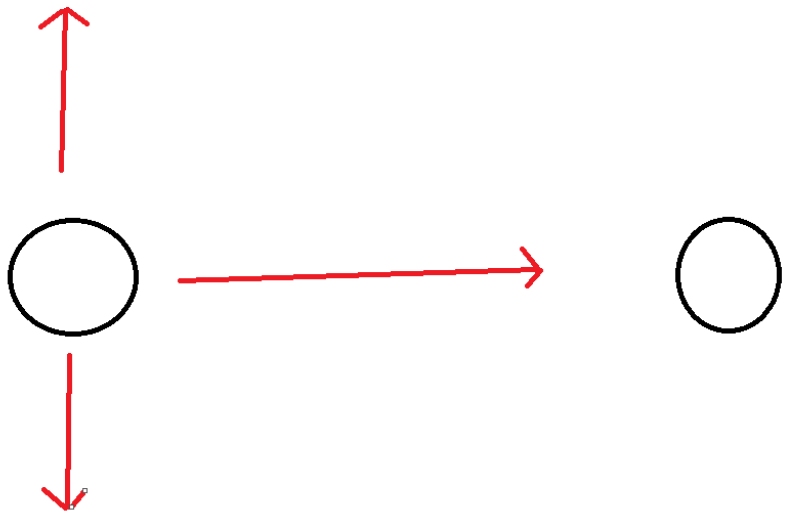
```cpp
void ABossMonster::LookAt(const FVector& targetLocation, const float& DeltaTime)
{
    // 몬스터의 전방 방향 벡터
    FVector forwardVector = GetActorForwardVector().GetSafeNormal();
    // 목표위치의 방향 벡터와 내적을 한다.
    float dot = FVector::DotProduct(forwardVector, targetLocation);

    //arccos값을 구해주고 라디안값에서 도 단위로 바꿔준다
    float angle = FMath::RadiansToDegrees(FMath::Acos(dot));


    FVector crossPro = FVector::CrossProduct(forwardVector, targetLocation);


    if (crossPro.Z < 0)
    {
        FRotator deltaRotation = FRotator(0, angle * -1 * DeltaTime * 5.0f, 0);
        AddActorWorldRotation(deltaRotation);
    }
    else
    {
        FRotator deltaRotation = FRotator(0, angle * DeltaTime * 5.0f, 0);
        AddActorWorldRotation(deltaRotation);
    }
}
```

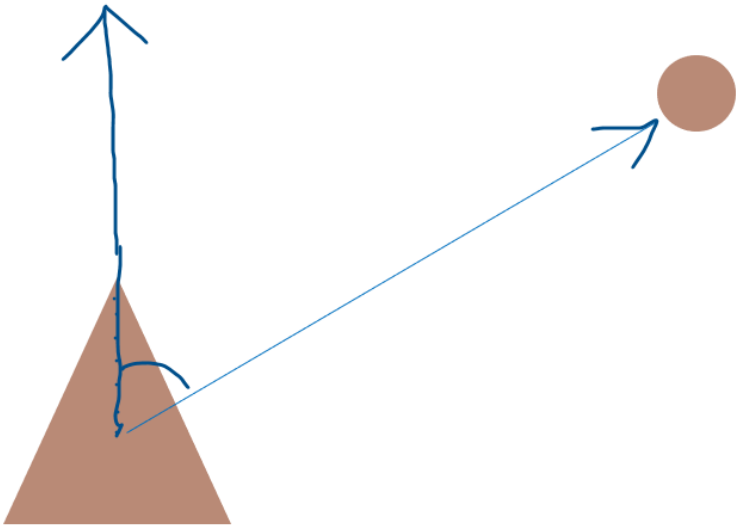
BossMonster 설명



첫번째 보스 패턴은 위의 패턴처럼 플레이어 방향으로 한방, 그리고 직각 기준으로 한방씩 나가게 구현하였다.
이때 직각 방향은 Z방향과 플레이어 방향을 외적하고 하나는 +, 하나는 -해서 구했다.

이때 패턴을 우선순위큐에 담아서 한 이유는 특정 HP에 발동될 강력한 패턴이 발동되어야하는데 명령큐의 명령이 너무 많이 쌓여있으면 큐가 모두 비워지기전까지 강력한 패턴이 실행되지 않을거라고 생각했다.
따라서 중요한 패턴은 큐에서 가장 먼저 실행되게 하기 위하여 우선순위 큐라는 자료구조를 사용하였다.

또한 플레이어쪽으로 총알을 발사할 때 보스가 플레이어를 쳐다보는게 자연스럽다고 생각했다.
이 때 보스몬스터의 전방방향 벡터와 플레이어로의 방향벡터 사잇각을 알 필요가 있었다.



내적을 응용하여 두 방향 벡터를 다음과 같은 연산을 한다면 cos세타 값을 구할수 있다.

$$\cos\theta = \frac{a \cdot b}{\|a\|\|b\|}$$

따라서 위의 식에서 단위벡터를 사용한다면 세타는 arccos(a*b)로 간단하게 나타 낼수 있다.
이때 플레이어가 오른쪽에 있는지, 왼쪽에 있는지 알아낼 필요가 있기에
전방벡터와 플레이어 방향벡터를 외적하여 Z 부호를 본 후 +, - 여부를 결정하였다.

6-3. 보스몬스터의 두번째 패턴

참조	https://gogogamegaga.tistory.com/29

BossMonster.h 의 첫번째 패턴 부분 Code는 생략

BossMonster.cpp 의 두번째 패턴 부분 Code

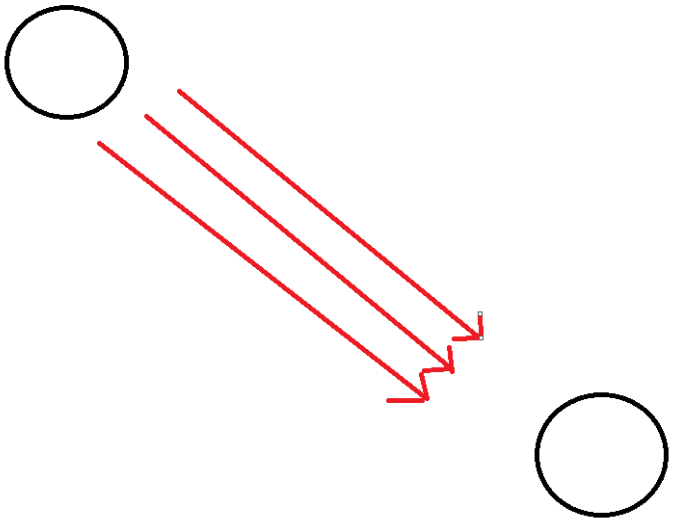
```
void ABossMonster::Pattern2()
{
    doingAttacking = true;
    FVector MonsterLocation = GetActorLocation();
    FVector PlayerPosition = GetWorld()->GetFirstPlayerController()->GetPawn()->GetActorLocation();

    float speed = 2000.0f;

    FVector startLocation = FVector(MonsterLocation.X, MonsterLocation.Y, MonsterLocation.Z + 40.0f);
    ShotBullet(startLocation, (PlayerPosition - MonsterLocation).GetSafeNormal(), speed);
    startLocation = FVector(MonsterLocation.X, MonsterLocation.Y + 40.0f, MonsterLocation.Z);
    ShotBullet(startLocation, (PlayerPosition - MonsterLocation).GetSafeNormal(), speed + 500.0f);
    startLocation = FVector(MonsterLocation.X + 40.0f, MonsterLocation.Y, MonsterLocation.Z);
    ShotBullet(startLocation, (PlayerPosition - MonsterLocation).GetSafeNormal(), speed);


    doingAttacking = false;
}
```

BossMonster의 두번째 패턴 부분 설명



두번째 패턴은 플레이어 방향으로 3개의 투사체를 동시에 쏘는 것을 구현했다.

6-4. 보스몬스터의 세번째 패턴

참조	https://gogogamegaga.tistory.com/31

- 세번째 패턴은 하늘에서 운석이 내리는 패턴이다.
- 가장 먼저 운석이 내릴 위치를 보여준 후
- 그 다음 하늘에서 수직으로 운석이 떨어진다

BossMonster.h 의 세번째 패턴 부분 Code

```
void Pattern3();
void AddQueuePattern3();
```

```
UFUNCTION()
void Pattern3_Shot(const TArray<float>& X_Param, const TArray<float>& Y_Param, const TArray<AActor*>& Light_Param); // 실제 포탄이 떨어지는 함수
```

BossMonster.cpp 의 세번째 패턴 부분 Code

```cpp
void ABossMonster::Pattern3()
{
    doingAttacking = true;


    UWorld* world = GetWorld();
    FActorSpawnParameters spawnParameter;
    FRotator rotater;

    TArray<float> X_Param; // x좌표
    TArray<float> Y_Param; // y좌표
    TArray<AActor*> Light_Param; // 경고존 class 주소
    FVector MonsterLocation = GetActorLocation();

    for (int i = 0; i < 3; i++)
    {
        // 랜덤 x 좌표
        X_Param.Push(MonsterLocation.X + FMath::RandRange(-1000.0f, 1000.0f));
        // 랜덤 y좌표
        Y_Param.Push(MonsterLocation.Y + FMath::RandRange(-1000.0f, 1000.0f));

        // 위험존 표시 좌표
        FVector dangerZoneVec = FVector(X_Param[i], Y_Param[i], GetActorLocation().Z - 50.0f);
        // 위험존 스폰

        AActor* ALight = world->SpawnActor<AActor>(DangerZoneClass, dangerZoneVec, rotater, spawnParameter);



        Light_Param.Push(ALight);
    }


    FTimerHandle timerHandle1;
    FTimerDelegate RespawnDelegate;
    RespawnDelegate.BindUFunction(this, FName("Pattern3_Shot"), X_Param, Y_Param, Light_Param);
    GetWorldTimerManager().SetTimer(timerHandle1, RespawnDelegate, 1.0f, false);


    moveStart(GetActorForwardVector(), 2.0f);
```

```cpp
void ABossMonster::Pattern3_Shot(const TArray<float>& X_Param, const TArray<float>& Y_Param, const TArray<AActor*>& Light_Param)
{

    FVector directionZ = FVector(0.0f, 0.0f, -1.0f);
    float speed = 2000.0f;


    for (int i = 0; i < 3; i++)
    {
        Light_Param[i]->Destroy();
        ShotBullet(FVector(X_Param[i], Y_Param[i], GetActorLocation().Z + 1000.0f), directionZ, speed+500.0f);
    }


}
```

BossMonster의 세번째 패턴 부분 설명

Pattern3() 함수에서 몬스터기준으로 근처에 3개의 포인트를 정한다.

그리고 3개의 포인트에 경고존을 소환 한 후

1초 후에 실제 포탄이 발사가 되는 Pattern3_Shot함수가 실행이 된다.

Pattern3_Shot은 Private으로 만들어야 바람직할거라 생각했지만 SetTimer에 델리게이트함수로 달아주기 위해선 public형식의 UFUNCTION 함수여야하는것을 실험적으로 확인했다.

6-5. 보스몬스터의 네번째 패턴

참조	https://gogogamegaga.tistory.com/32

-네번째 패턴은 메이플스토리의 검은마법사 2페이즈의 패턴에서 아이디어를 얻었다.

- 포인트는 플레이어 머리위로부터 경고존이 뜨고 머리위 랜덤한 구역에서 투사체가 날라오는것이다

BossMonster.h 의 네번째 패턴 부분 Code

```
UFUNCTION()
void Pattern4_Danger(const float& dx, const float& dy); //위험존 표시
UFUNCTION()
void Pattern4_Shot(AActor* ALight, const float& dx, const float& dy); //실제 포탄이 떨어짐
```

BossMonster.cpp 의 네번째 패턴 부분 Code

```
void ABossMonster::Pattern4()
{
    doingAttacking = true;

   // 즉시 경고존 발생
    Pattern4_Danger(FMath::RandRange(-300.0f, 300.0f), FMath::RandRange(-300.0f, 300.0f));

    //0.5초 후 경고존 발생
    FTimerHandle timerHandle1;
    FTimerDelegate RespawnDelegate; //= FTimerDelegate::CreateUObject(this, &ABossMonster::Pattern3_Shot, XY_Param);
    RespawnDelegate.BindUFunction(this, FName("Pattern4_Danger"), FMath::RandRange(-300.0f, 300.0f), FMath::RandRange(-300.0f, 300.0f)
    GetWorldTimerManager().SetTimer(timerHandle1, RespawnDelegate, 0.5f, false);

    //1초 후 경고존 발생
    FTimerHandle timerHandle2;
    FTimerDelegate RespawnDelegate2; //= FTimerDelegate::CreateUObject(this, &ABossMonster::Pattern3_Shot, XY_Param);
    RespawnDelegate2.BindUFunction(this, FName("Pattern4_Danger"), FMath::RandRange(-300.0f, 300.0f), FMath::RandRange(-300.0f, 300.0f
    GetWorldTimerManager().SetTimer(timerHandle2, RespawnDelegate2, 1.0f, false);

    //1.5초 후 경고존 발생
    FTimerHandle timerHandle3;
    FTimerDelegate RespawnDelegate3; //= FTimerDelegate::CreateUObject(this, &ABossMonster::Pattern3_Shot, XY_Param);
    RespawnDelegate3.BindUFunction(this, FName("Pattern4_Danger"), FMath::RandRange(-300.0f, 300.0f), FMath::RandRange(-300.0f, 300.0f
    GetWorldTimerManager().SetTimer(timerHandle3, RespawnDelegate3, 1.5f, false);

    //2초 후 경고존 발생
    FTimerHandle timerHandle4;
    FTimerDelegate RespawnDelegate4; //= FTimerDelegate::CreateUObject(this, &ABossMonster::Pattern3_Shot, XY_Param);
    RespawnDelegate4.BindUFunction(this, FName("Pattern4_Danger"), FMath::RandRange(-300.0f, 300.0f), FMath::RandRange(-300.0f, 300.0f
    GetWorldTimerManager().SetTimer(timerHandle4, RespawnDelegate4, 2.0f, false);

    doingAttacking = false;
}
```

```cpp
void ABossMonster::Pattern4_Danger(const float& dx, const float& dy)
{
    FActorSpawnParameters spawnParameter;
    FRotator rotater;

    FVector playerLocation = GetWorld()->GetFirstPlayerController()->GetPawn()->GetActorLocation();
    playerLocation.X += dx;
    playerLocation.Y += dy;

    playerLocation.Z = playerLocation.Z -50.0f;

    //플레이어 발 위치에 경고존 발생
    AActor* ALight = GetWorld()->SpawnActor<AActor>(DangerZoneClass, playerLocation, rotater, spawnParameter);

    //1초 후 실제 포탄이 떨어짐
    FTimerHandle timerHandle1;
    FTimerDelegate RespawnDelegate;
    RespawnDelegate.BindUFunction(this, FName("Pattern4_Shot"), ALight, dx, dy);
    GetWorldTimerManager().SetTimer(timerHandle1, RespawnDelegate, 1.0f, false);


}
```

```cpp
void ABossMonster::Pattern4_Shot(AActor* ALight, const float& dx, const float& dy)
{
    UE_LOG(LogTemp, Warning, TEXT("4"));
    ALight->Destroy();


    FVector playerLocation = GetWorld()->GetFirstPlayerController()->GetPawn()->GetActorLocation();

    //플레이어 머리위 랜덤 반경에서 포탄이 발사
    FVector startLocation = playerLocation;
    startLocation.Z += 1000.0f;
    startLocation.Y += FMath::RandRange(-300.0f, 300.0f);
    startLocation.X += FMath::RandRange(-300.0f, 300.0f);

    playerLocation.X += dx;
    playerLocation.Y += dy;
    FVector Direction = (playerLocation - startLocation).GetSafeNormal();
    float speed = 1500.0f;
    ShotBullet(startLocation, Direction, speed);
}
```

BossMonster의 네번째 패턴 부분 설명

Pattern4()함수에서 시간차를 두고 경고존을 발생시키는 함수 Pattern4_Danger()를 실행시킨다.

Pattern4_Danger는 플레이어 발 밑에 경고존을 표시하고 1초 후에 포탄을 발사하는 함수 Pattern_Shot()을 실행시킨다.

Pattern_Shot()은 플레이어 머리위 랜덤 반경에서 플레이어를 향해서 포탄을 발사한다.

6-6. 보스몬스터의 다섯번째 패턴

참조	https://gogogamegaga.tistory.com/50 https://gogogamegaga.tistory.com/51

- 특별한 기술이 들어간 것은 아니고 단순 구현이기에 설명은 생략한다. 링크 참조 부탁드립니다

6-7 보스몬스터의 여섯번째 패턴

참조	https://gogogamegaga.tistory.com/54

- 여섯번째 패턴은 보스몬스터의 메인패턴이 된다
- 보스몬스터가 일정 Hp가 되면 플레이어를 랜덤미로로 보낸다.
- 플레이어는 랜덤미로 속에서 포탄3개를 찾고 탈출 해야한다.
- 플레이어는 미로 탈출 후 포탄3개로 미리 준비된 터렛으로 미로의 심장을 파괴 해야한다

6-7-1 랜덤미로 만들기

MakeMaze.h Code

```
public:
    UPROPERTY(EditDefaultsOnly)
        TSubclassOf<class AActor> cube; //미로 벽

    UPROPERTY(BlueprintReadonly)
    FVector2D EndPoint; //바이너리 서치로 발견한 출구, 이곳은 시작점이 된다.

    UPROPERTY(EditDefaultsOnly, Category = Projectile)
        TSubclassOf<class AActor> TurretBullet; // 미로 안에 설치 될 터렛
private:
    void GenerateMakeMaze(); // 미로 생성 알고리즘
    bool IsRange(const int& x, const int& y); // 미로 행렬의 범위 판단 함수

};
```

MakeMaze.cpp Code

```
bool AMakeMase::IsRange(const int& x, const int& y)
{
    if (0 <= x && x < 25 && 0 <= y && y < 25)
    {
        return true;
    }
    return false;
}
```

```cpp
void AMakeMase::GenerateMakeMaze()
{
    struct RoadInfo
    {
        bool isRoad;
        bool isLinked;
    };

    std::vector<std::vector<RoadInfo>> maze(25, std::vector<RoadInfo>(25));
    bool isVisited[25][25];


    //미로 초기화
    for (int i = 0; i < 25; i++)
    {
        for (int j = 0; j < 25; j++)
        {
            maze[i][j].isRoad = false;
            maze[i][j].isLinked = false;
            isVisited[i][j] = false;
        }
    }

    // 미로를 체크무늬로 만들어줌
    for (int i = 1; i < 24; i++)
    {
        for (int j = 1; j < 24; j++)
        {
            if (i % 2 == 0 || j % 2 == 0)
            {
                maze[i][j].isRoad = false;
            }
            else
            {
                maze[i][j].isRoad = true;
```

```cpp
// 이진트리
for (int i = 0; i < 25; i++)
{
    for (int j = 0; j < 25; j++)
    {
        int RandNum = FMath::RandRange(0, 1);

        //체크무늬가 된 시점에서 해당 좌표가 벽이라면 그냥 넘어감
        if (i % 2 == 0 || j % 2 == 0)
        {
            continue;
        }

        // 랜덤숫자에 따라서 두 방향중에 하나 길 뚫음
        if (RandNum == 0)
        {
            maze[i][j+1].isRoad = true;
        }
        else
        {
            maze[i + 1][j].isRoad = true;
        }
    }
}
```

```
const int xx[4] = { 1,-1,0,0 };
const int yy[4] = { 0, 0, 1,-1 };
//BFS. 출구를 찾는다. 그리고 포탄생성 후보지(막다른길)를 탐색한다.
{

    for (int i = 0; i < 25; i++)
    {
        for (int j = 0; j < 25; j++)
        {
            isVisited[i][j] = false;
        }
    }


    std::queue<FVector2D> BFSQueue;
    BFSQueue.push(FVector2D(1, 1));
    isVisited[1][1] = true;
    bool IsSearching = false;

    int InstallBullet = 3;

    TArray<FVector> InstallBulletLocations;
    while (!BFSQueue.empty())
    {
        int x = BFSQueue.front().X;
        int y = BFSQueue.front().Y;
        BFSQueue.pop();

        bool FindRoud = false;
        for (int k = 0; k < 4; k++)
        {
            int CurrX = x + xx[k];
            int CurrY = y + yy[k];

            if ((CurrX == 25 - 1 || CurrY == 25 -1) && maze[CurrX][CurrY].isRoad)
            {
```

```
            if ((CurrX == 25 - 1 || CurrY == 25 -1) && maze[CurrX][CurrY].isRoad)
            {
                IsSearching = true;
                EndPoint = FVector2D(CurrX, CurrY);
                UE_LOG(LogTemp, Warning, TEXT("%d %d"), CurrX, CurrY);
                FindRoud = true;
            }
            // 범위 안이고, 아직 방문하지 않았고, 길 일때
            else if (IsRange(CurrX, CurrY) && !isVisited[CurrX][CurrY] && maze[CurrX][CurrY].isRoad)
            {
                BFSQueue.push(FVector2D(CurrX, CurrY));
                isVisited[CurrX][CurrY] = true;
                FindRoud = true;
            }
        }

        // 만약 막다른길일 경우 포탄 생성 장소 후보에 놓는다
        if (!FindRoud)
        {
            FVector InstallPoint = GetActorLocation();
            InstallPoint.X += 300.0f * x;
            InstallPoint.Y += 300.0f * y;
            InstallBulletLocations.AddUnique(InstallPoint);
        }

    }
```

```cpp
// 미로 내에 포탄 랜덤 생성. 구석 자리에 소환
{
    int BulletInstallLocationOffset = InstallBulletLocations.Num() / 3;
    FActorSpawnParameters SpawnParams;
    SpawnParams.Owner = this;
    SpawnParams.Instigator = GetInstigator();
    AActor* SpawnBullet;
    SpawnBullet = GetWorld()->SpawnActor<AActor>(TurretBullet, InstallBulletLocations[FMath::RandRange(0, BulletInstallLocationOffset)], FRotator(0, 0, 0), SpawnParams);
    SpawnBullet->SetActorLabel("TurretBullet");

    SpawnBullet = GetWorld()->SpawnActor<AActor>(TurretBullet, InstallBulletLocations[FMath::RandRange(BulletInstallLocationOffset + 1, BulletInstallLocationOffset + 2)], FRotator(0, 0, 0
    SpawnBullet->SetActorLabel("TurretBullet");

    SpawnBullet = GetWorld()->SpawnActor<AActor>(TurretBullet, InstallBulletLocations[FMath::RandRange(BulletInstallLocationOffset + 2 + 1, InstallBulletLocations.Num() - 1)], FRotator(0,
    SpawnBullet->SetActorLabel("TurretBullet");
}
```

```cpp
//가장자리 막기
for (int i = 0; i < 25; i++)
{
    if (maze[i][25 - 1].isRoad)
    {
        maze[i][25 - 1].isRoad = false;
    }
    if (maze[25 - 1][i].isRoad)
    {
        maze[25 - 1][i].isRoad = false;
    }
}

maze[0][1].isRoad = true; //출구 뚫기
```

```cpp
// 이진탐색으로 찾은 출구를 출발지로 변경.
for (int i = 0; i < 4; i++)
{
    int x = EndPoint.X + xx[i];
    int y = EndPoint.Y + yy[i];

    if (IsRange(x,y) && maze[x][y].isRoad)
    {
        maze[EndPoint.X][EndPoint.Y].isRoad = false;
        EndPoint = FVector2D(x, y);
        break;
    }
}
```

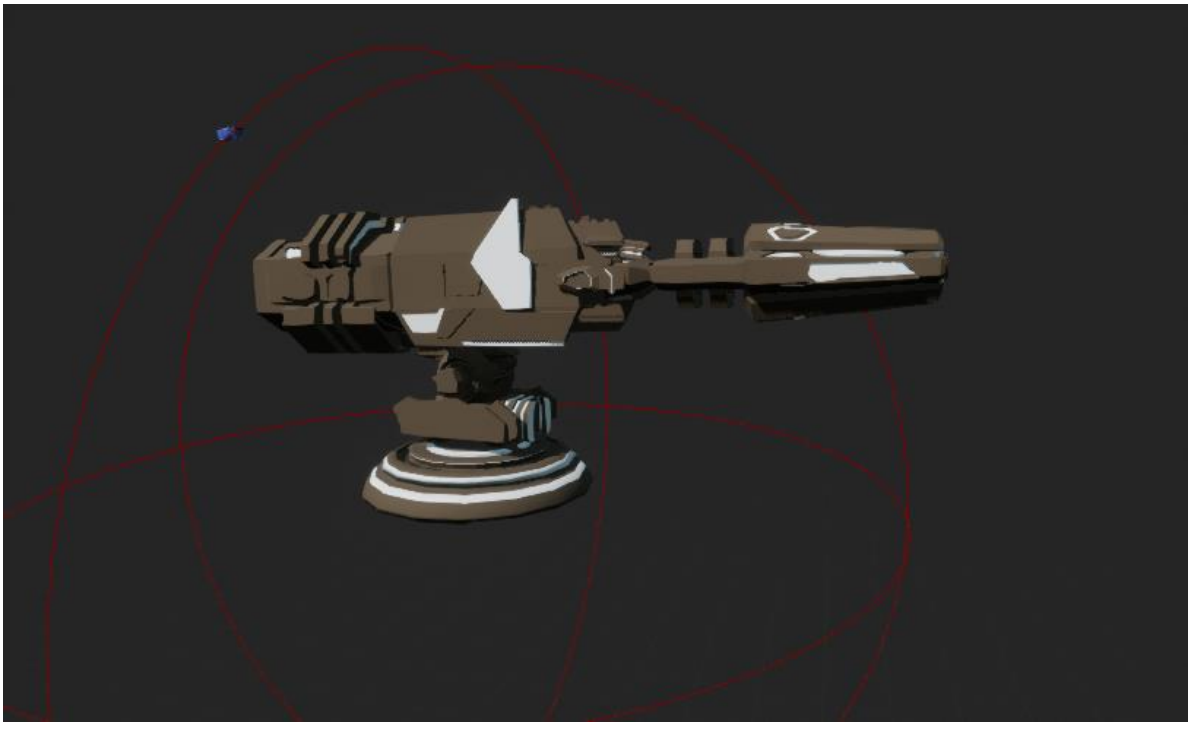
```cpp
//미로 생성
FVector ZeroPoint = GetActorLocation();

for (int i = 0; i < 25; i++)
{
    for (int j = 0; j < 25; j++)
    {
        if (!maze[i][j].isRoad)
        {
            FVector InstallPoint = ZeroPoint;
            InstallPoint.X += i * 300;
            InstallPoint.Y += j * 300;
            FActorSpawnParameters p;
            GetWorld()->SpawnActor<AActor>(cube, InstallPoint, GetActorRotation());
        }
    }
}
```

MakeMaze 설명
랜덤 생성 알고리즘은 바이너리트리를 선택하였다. 바이너리 트리 알고리즘으로 미로생성을 했을 때 생길수 있는 문제점으로 출입구가 여러 개일수도 있는 문제가 있었다. 그리고 또한 실제 게임을 진행할 때 랜덤미로지만 약간의 규칙성을 찾을 수가 있었다. 따라서 이를 해결하기 위하여 완전탐색을 활용하여 가장 먼 거리의 출구를 찾은 다음, 그 출구를 시작지로 바꾸고 원래의 입구를 도착지로 바꾸었고 나머지 도착지들은 모두 막아주었다. 이때 완전탐색은 BFS를 사용하였다. BFS를 사용한 이유는 먼저 탐색된 곳이 최단거리임이 보장이 되기 때문이다. 이게 어디에다 쓰였냐면 1. 가장 마지막에 탐지된 도착지가 최장거리임이 보장된다. 2. 랜덤포탄생성 위치 후보를 정할 때 포탄생성이 한쪽에 몰리는걸 방지 할 필요가 있었다. 따라서 거리 순으로 포탄후보지를 배열로서 저장해둔 후 배열을 3등분하여 각각에서 랜덤으로 위치를 하나씩 정해주었다.

6-7-2 터렛만들기

참조	https://gogogamegaga.tistory.com/56



- 터렛과 인벤토리의 연동 필요
- 플레이어가 터렛 근처로 가면 상호작용 UI가 나타나야함
- 상호작용시 플레이어가 터렛 조종 가능
- 터렛이 사격을 하면 인벤토리의 포탄을 줄어들게 만들어야하는데 인벤토리는 Character.cpp에 있지만 터렛을 사용하기 위해서 제어권을 터렛으로 옮긴다면 Character의 인벤토리로 접근을 못하는 문제가 있었음

Turret.h Code

```
UPROPERTY(BlueprintReadWrite)
class AShootWithBossCharacter* Player; // 여기에 탄 플레이어
```

Turret.cpp Code

```
void ATurretPawn::Fire()
{
    if (!ensure(Player != nullptr))
    {
        UE_LOG(LogTemp, Warning, TEXT("Fire In Casted Failed"));
        return;
    }


    int CannonBallFindIndex = -1;
    // 대포알이 인벤토리 어디에 위치해있는지 알아냄
    for (int i = 0; i < Player->InventoryWidget->Inventory.Num(); i++)
    {
        if (Player->InventoryWidget->Inventory[i].ItemID == 1)
        {
            CannonBallFindIndex = i;
            break;
        }
    }
```

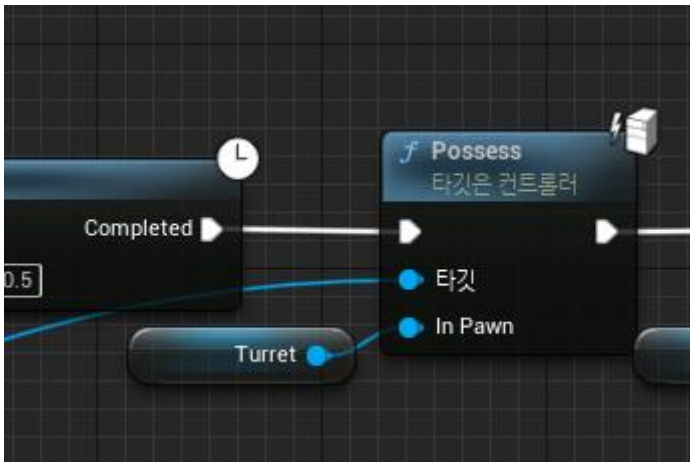
```
// 포탄이 인벤토리 내에 있고 1개 이상 있을때
if (CannonBallClass && CannonBallFindIndex >= 0&& Player->InventoryWidget->Inventory[CannonBallFindIndex].ItemCount > 0)
{
    Player->InventoryWidget->Inventory[CannonBallFindIndex].ItemCount--; // 쏘면 포탄하나 줄어듬
```

Turret 설명

기본적인 것은 Character.cpp 에서 총알을 발사하는 메커니즘과 거의 똑같다.

다른 점은 인벤토리내에서의 포탄을 하나씩 까야한다는 점이다.

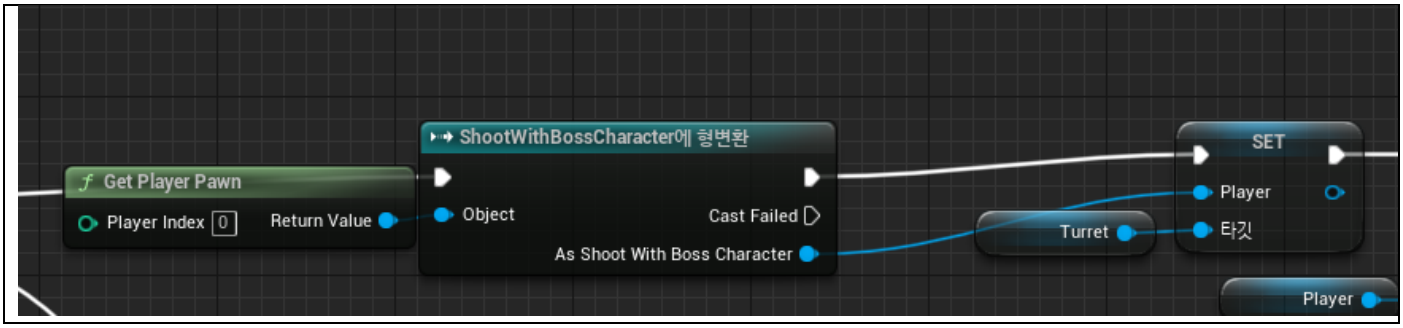
이 때 타워를 조종하기 위해선 Posses()함수를 사용해서 터렛에 대한 제어권을 획득해야했다.



이 때 제어권이 터렛으로 넘어가버렸다면 Character의 Inventory에는 접근할 수가 없게 되므로

터렛으로 제어권이 넘어가기전에 터렛의 변수에 Character 정보를 레퍼런스로 전달해주었고

그리하여 터렛에서 현재 탄 플레이어의 인벤토리에 접근할수 있게 하였다.



프로젝트를 진행하면서 어려웠던 점

1. AI가 플레이어로부터 이동하게 하는 것을 Nav Mesh를 쓰지 않고 자체적으로 A*알고리즘을 활용하여 구현할 구 있을 것 같았다.

노력해서 A*알고리즘 기반으로 최단경로를 구하고 그 경로를 따라서 몬스터가 움직이는 것까진 구현을 하였다.

```
// A* 알고리즘
TArray<FVector2D> AGrid_PathFind::FindPathtoTarget(FVector2D TargetTile, FVector2D StartTile)
{
    // 최단 경로
    TArray<FVector2D> Path;

    if (Tiles.Find(TargetTile) != false) //목표 위치가 맵 상에 있을때
    {
        FStruct_Tile* target = Tiles.Find(TargetTile); // 목표의 타일 정보를 불러옴

        if (target->GroundType != Color_Type::None && target->GroundType != Color_Type::Impossible) // 목표가 도착할수 있는 위치라면
        {
            TMap<FVector2D, FStruct_Tile> TilesTmp;
            // 타일 초기화
            for (auto& tile : Tiles)
            {
                Tiles[tile.Key].FinalCost = 999;
                Tiles[tile.Key].CostFromStart = 999;
                Tiles[tile.Key].EstimitatedCostToTarget = 999;
            }
            FStruct_Tile* finded = Tiles.Find(StartTile);
            Tiles[finded->GridIndex].CostFromStart = Tiles[finded->GridIndex].EstimitatedCostToTarget = GetEstiminatedCostToTarget(StartTile, TargetTile); // 출발 타일의 휴리스틱 추정값 계산 후 저
        }
        FVector2D CurrentTile = StartTile;
        TArray<FVector2D> OpenSet;
        TArray<FVector2D> CloseSet;
        OpenSet.AddUnique(StartTile);
        FVector2D OpenSetCheapest;


        //BFS랑 비슷하게. 그러나 큐는 안씀. 나중에 큐를 쓰는 방향으로 리팩토링 하는것 생각해보기
        while (OpenSet.Num() > 0)
        {

            //UE_LOG(LogTemp, Warning, TEXT("doing--- %f   %f"), OpenSet[0].X, OpenSet[0].Y);
            OpenSetCheapest = OpenSet[0];
```

```
while (OpenSet.Num() > 0)
{

    //UE_LOG(LogTemp, Warning, TEXT("doing--- %f   %f"), OpenSet[0].X, OpenSet[0].Y);
    OpenSetCheapest = OpenSet[0];


    // OpenSet 탐색해서 가장 최단경로일것 같은걸 골라냄
    for (FVector2D OpenSetElement : OpenSet)
    {
        FStruct_Tile* findedOpenSet = Tiles.Find(OpenSetElement);
        FStruct_Tile* findedOpenSetCheapest = Tiles.Find(OpenSetCheapest);

        if (findedOpenSet->FinalCost < findedOpenSetCheapest->FinalCost)
        {
            OpenSetCheapest = OpenSetElement;
        }
        else if (findedOpenSetCheapest->FinalCost == 0 && findedOpenSet->FinalCost < findedOpenSetCheapest->EstimitatedCostToTarget)
        {
            OpenSetCheapest = OpenSetElement;
        }
    }

    CurrentTile = OpenSetCheapest; // 가장 최단 타일이라고 예측되는 타일
    OpenSet.Remove(CurrentTile); // queue pop

    CloseSet.AddUnique(CurrentTile);  // Close Set에 추가


    TArray<FVector2D> CurrentTileNeighbours = GetTileNeighbours(CurrentTile); // int xx[4] = {0,0,1,-1}

    for (FVector2D CurrentNeighbour : CurrentTileNeighbours)
    {

        if (CloseSet.Contains(CurrentNeighbour) == false) // 만약 인접 타일이 접근하지 않은 타일이라면
        {
            int CurrentNeighbourCostfromStart = Tiles.Find(CurrentTile)->CostFromStart + Tiles.Find(CurrentNeighbour)->TileCost; // 현재 타일까지의 값과 인접 타일까지의 값을 더한 값

            if (OpenSet.Contains(CurrentNeighbour) == false) // OpenSet에 없으면 추가한다
            {
```

```
            if (CloseSet.Contains(CurrentNeighbour) == false) // 만약 인접 타일이 접근하지 않은 타일이라면
            {
                int CurrentNeighbourCostfromStart = Tiles.Find(CurrentTile)->CostFromStart + Tiles.Find(CurrentNeighbour)->TileCost; // 현재 타일까지의 값과 인접 타일까지의 값을 더한 값

                if (OpenSet.Contains(CurrentNeighbour) == false) // OpenSet에 없으면 추가한다
                {
                    OpenSet.AddUnique(CurrentNeighbour);
                    FStruct_Tile* findedTile = Tiles.Find(CurrentNeighbour);
                    Tiles[CurrentNeighbour].FinalCost = CurrentNeighbourCostfromStart + GetEstiminatedCostToTarget(CurrentNeighbour, TargetTile);
                    Tiles[CurrentNeighbour].CostFromStart = CurrentNeighbourCostfromStart;
                    Tiles[CurrentNeighbour].EstimitatedCostToTarget = GetEstiminatedCostToTarget(CurrentNeighbour, TargetTile);
                    Tiles[CurrentNeighbour].PriviousTile = CurrentTile;

                }
                else // OpenSet에 있으면?
                {
                    // 그냥 휴리스틱추정값 갱신
                    if (CurrentNeighbourCostfromStart < Tiles.Find(CurrentNeighbour)->CostFromStart)
                    {
                        FStruct_Tile* findedTile = Tiles.Find(CurrentNeighbour);
                        Tiles[CurrentNeighbour].FinalCost = CurrentNeighbourCostfromStart + GetEstiminatedCostToTarget(CurrentNeighbour, TargetTile);
                        Tiles[CurrentNeighbour].CostFromStart = CurrentNeighbourCostfromStart;
                        Tiles[CurrentNeighbour].EstimitatedCostToTarget = GetEstiminatedCostToTarget(CurrentNeighbour, TargetTile);
                        Tiles[CurrentNeighbour].PriviousTile = CurrentTile;

                    }
                }

                // 만약 이웃타일중에 도착지가 있다면?
                if (CurrentNeighbour == TargetTile)
                {
                    // 경로 추적.
                    Path = Retrace(StartTile, TargetTile);
                    return Path;
                }
            }
        }
    }
    return { { -1,-1 } };
}
return { { -2,-2 } };
}
```

이것은 목적지가 고정된 상태에서만 원활하게 사용할수 있었다.

왜냐하면 플레이어의 위치는 실시간으로 계속 변하기에 실시간으로 A*알고리즘으로 탐색 할 필요가 있었다.

그러나 이것을 동기적으로 처리하면 프레임율이 너무나도 떨어지는 상황이었기에 비동기적으로 처리해야할 필요가 있었다. 그러나 시간이 부족한 관계로 끝까지 구현하지 못하고 Nav Mesh로 쉽게 구현하는 것으로 방향을 바꾸었다.

참조 : https://gogogamegaga.tistory.com/47

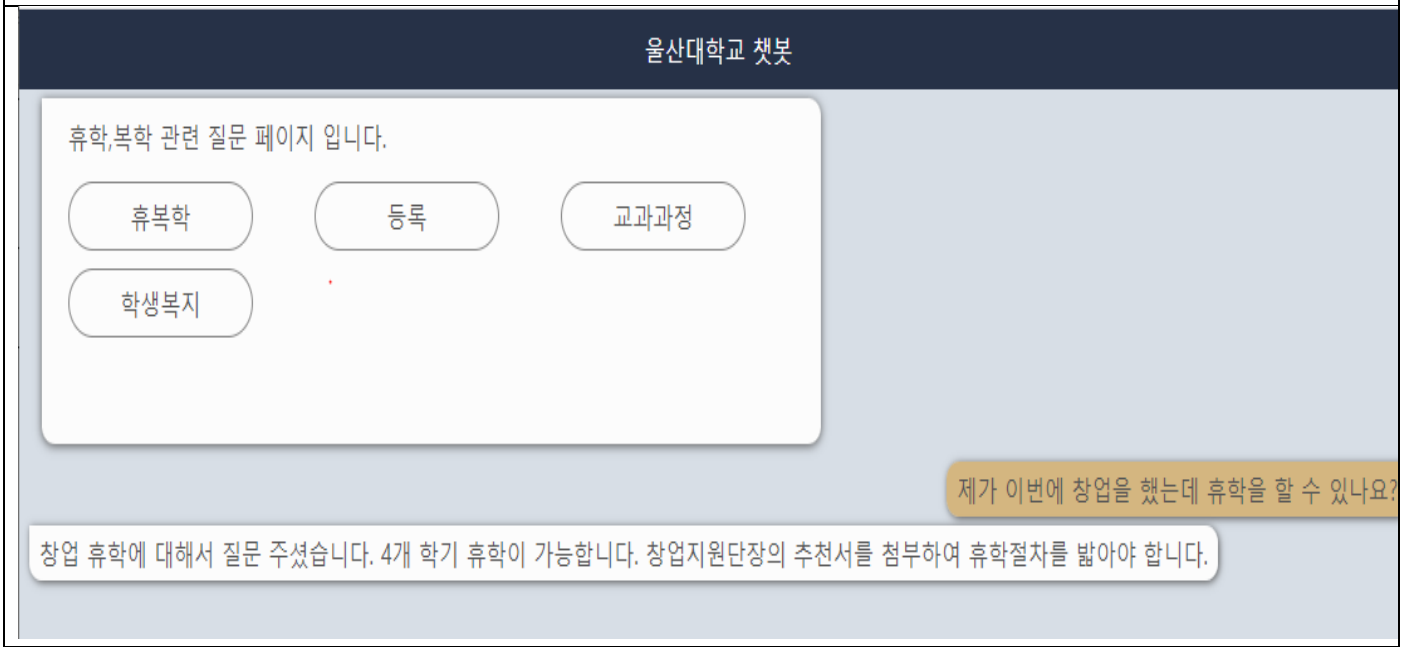
깨달은 점

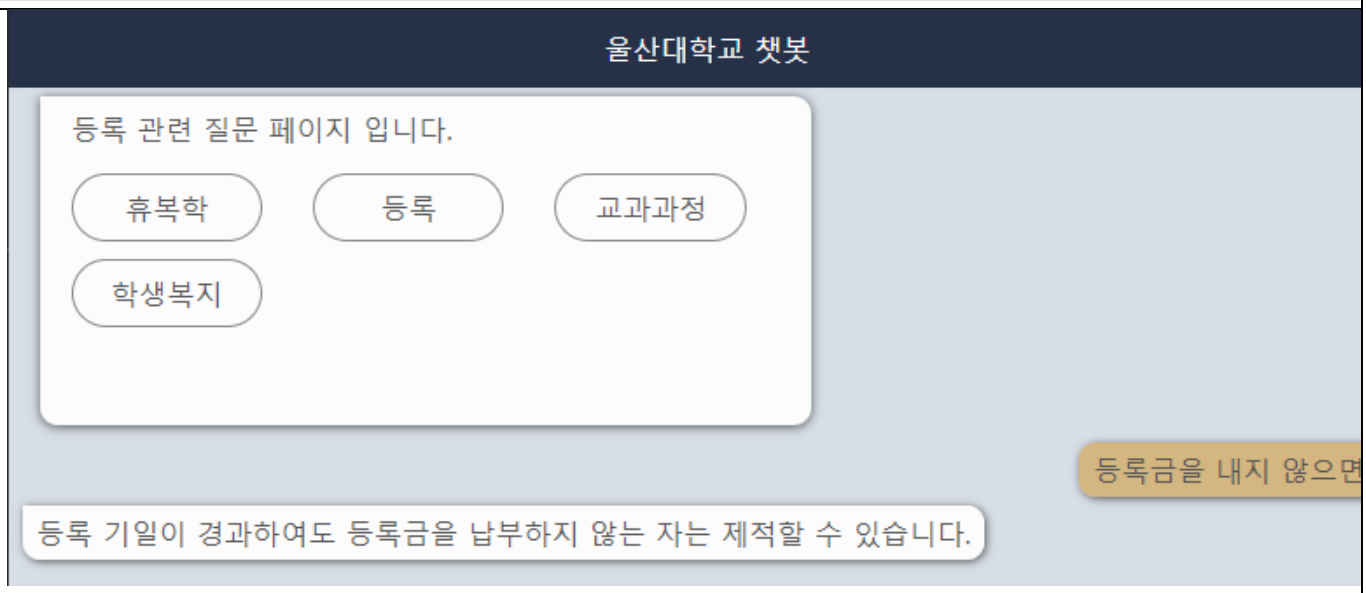
1. 다이렉트X와 C++, 그리고 CS지식이 어느정도 갈무리가 되고 진행한 첫 프로젝트여서 그런지 실제 개발과정에서 모든 이론들이 종합되는 느낌이 들었다. 아는 만큼 보이고 아는 만큼 설계 할수 있음을 깨달았다.
2. 리팩토링과 재사용성이 높은 코드의 중요성을 뼈저리게 느꼈다. 재사용성 = 생산성임을 깨달았다.
3. 섣부른 추상화는 오히려 부적절한 결과를 낳을수 있음을 깨달았다. 익숙해질때까진 다운업 방식으로 추상화를 하는게 좋음을 느꼈다
4. NullSafe하게 프로그래밍 하는게 쉬울줄 알았는데 생각보다 많이 귀찮았고 머리아팠다. 예외상황에 대한 빈틈없는 예외처리가 매우 중요함을 깨달았다.
5. 강력한 권한의 서버가 중요함을 깨달았고 그에따른 리플리케이트 작업을 하지 않을시 난리가 난다는 것을 깨달았다.
6. 어제까진 잘되다가 갑자기 프레임률이 떨어졌을땐 어제 로그를 가장먼저 확인 해보고 뭐가 달라졌는지 비교해야한다. 보통 랜드스케이프 아니면 폴리지 문제였다.
7. 라이브러리가 있는덴 이유가 있다. 웬만하면 내가 구현한것보다 속도가 빠르다 ㅠㅠ
8. 기획자, 아트, 이펙터, 애니메이터, 사운드 등이 없이 혼자서 다 하려고 하니 심각하게 리소스가 부족했다. 게임을 만드는데 다른 직군의 사람과의 협력이 얼마나 중요한지 깨달았다.

프로젝트 – 울산대학교 챗봇 시스템 개발

개발 기간	2021.06.30 ~ 2022.06.09
개발 환경	Python3 / hugging face – KoElectra / pycharm – professional -
개발 인원	3명 -> 2명 (도중에 한명 휴학)
요 약	흩어져있는 울산대학교 Q&A를 챗봇을 통해서 한 곳으로 통합
기능	**1) 휴학복학에 관한 Q&A** **2) 등록에 관한 Q&A** **3) 사용자의 질문이 1000개가 차면 자동으로 학습**

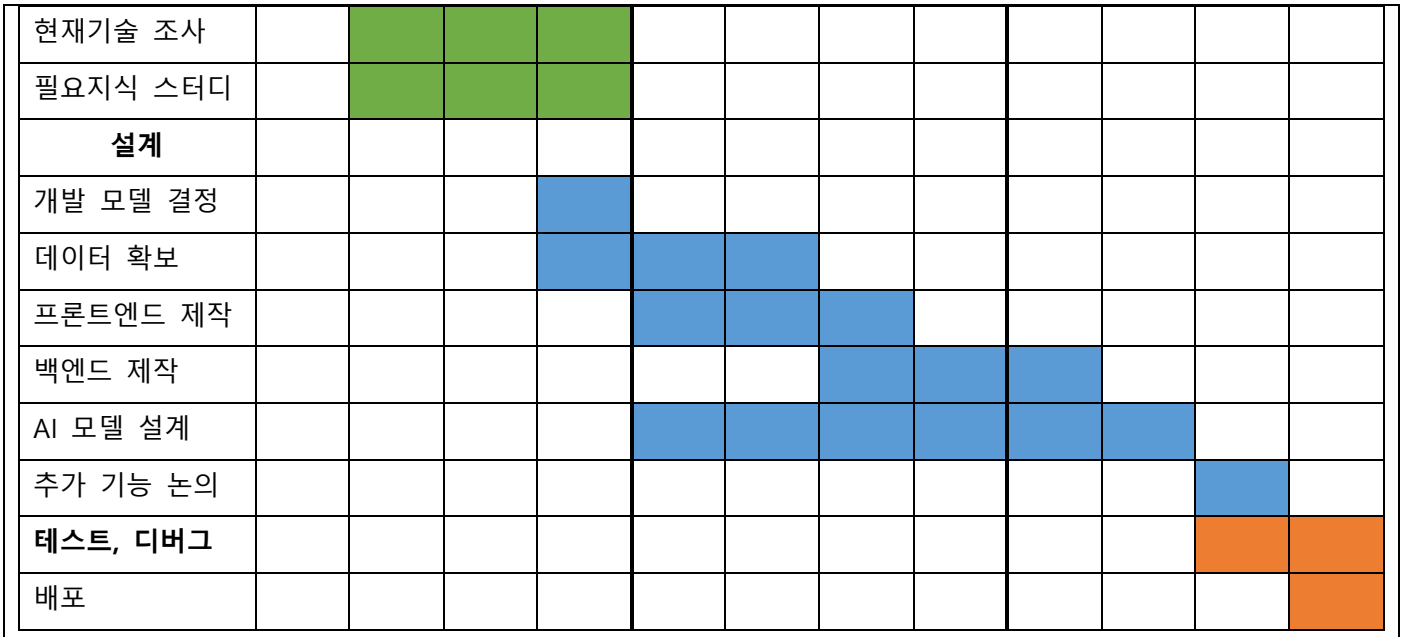
실행 화면





1. 개발 기획

MileStone	21.06	21.07	21.08	21.09	21.10	21.11	21.12	22.01	22.02	22.03	22.04	22.05
분석												
아이디어 회의												
수요 조사												



1.1 요구사항 :

기능적 요구사항

- 시스템은 사용자에게 질문을 입력 할 수 있는 영역을 제공
- 시스템은 사용자가 질문을 입력 하면 적절한 대답을 출력
- 사용자가 원하는 대답을 받지 못했을 때 시스템은 다른 답변 또는 솔루션 제시

프로덕트 요구사항 – 유용성

- 사용자에게 질문 입력 완료를 Enter 키 또는 마우스 클릭으로 표현 할 수 있게 제공
- UI의 채팅 박스는 수직적으로 밑으로 계속 쌓음. 이때 화면의 세로 길이는 증가한다
- 사용자의 입력 또는 시스템의 출력시 자동으로 스크롤바가 제일 밑으로 이동

프로덕트 요구사항 – 효율성

- 사용자가 질문 입력 후 답변 받는데 까지의 시간은 2sec 이내

프로덕트 요구사항 – 신뢰성

- 대답가능한 영역을 축소하더라도 답변에 대한 정확도를 높임
- 변동성이 있는 답변은 울산대학교 공식 홈페이지로 연결을 유도하거나 상담원에게 연결을 유도

조직 요구사항 – 배포

- Django를 이용하여 웹페이지를 만든 후 배포 및 서비스
- 배포는 AWS의 EC2를 사용

조직 요구사항 – 구현

- Python을 메인 언어로 사용
- 딥러닝 라이브러리는 pytorch 사용
- Data크롤링은 울산대학교 정보통신원에서 제공 받되 부족한 것은 BeautifulSoup와 Selenium을 혼합해서 해결

1.2 타사 아이템 분석

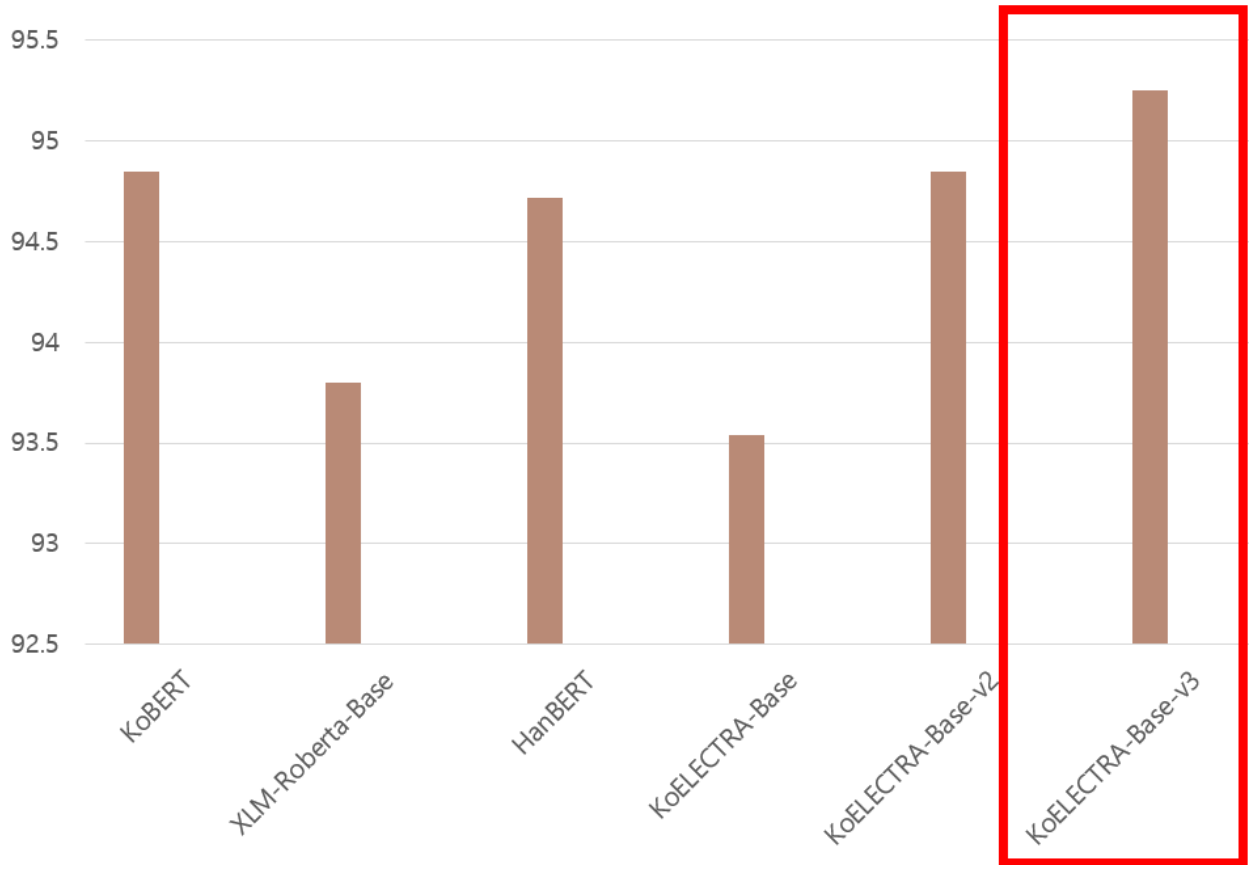
단위 : 1000원

	의뢰 비용	월 지속 비용
채x톡	문의 필요	70
깃x챗	문의 필요	50
단x AI	문의 필요	30
Cloxx xxring	2000	100

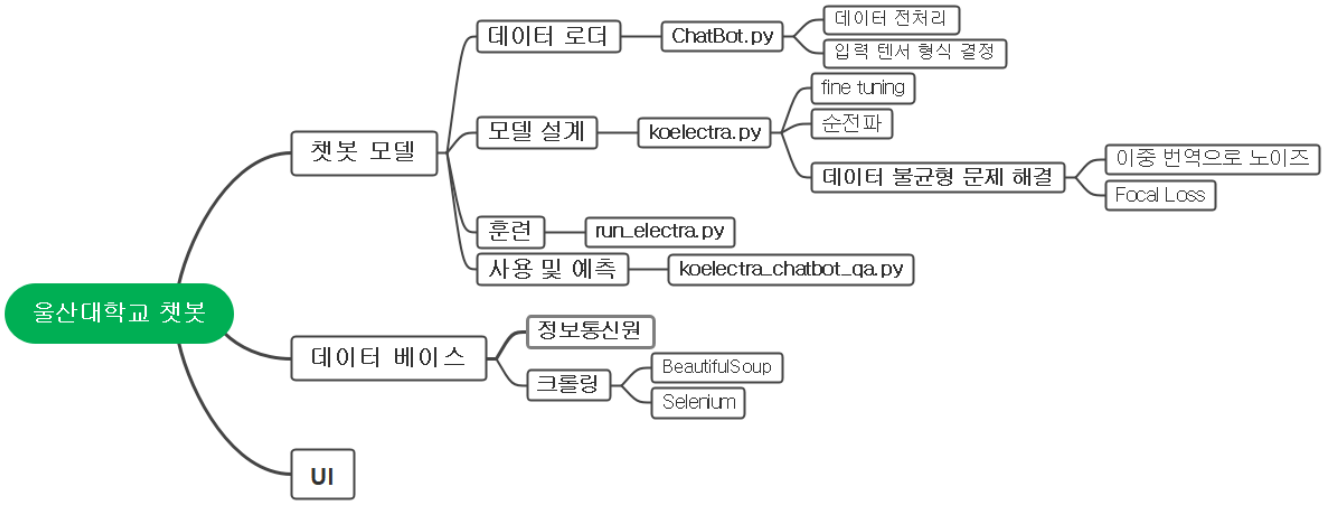
- 초기 설계 비용이 꽤 들어감
- 유지에도 비용이 다소 들어감
- 기존 챗봇 연구는 Bert와 GPT 중심의 연구

차별점 :

- 기존 챗봇은 Bert 모델이 중심이었음 그러나 이 연구에서 사용한 기반기술은 Bert보다 높은 성능을 지니고 있는 최신 기술임
- 학부생이 자체 개발한 기술로서 제작 단가가 저렴
- 데이터가 쌓이면서 유지보수를 하면 무궁무진한 연구 잠재력이 있음



1.3 마인드맵을 이용해 필요한 기능과 로직 분석.



2.1 dataloader.Chatbot.ChatbotTextClassicification 클래스

Code	해설

`index_of_words = self.tokenizer.encode(datas[1])`
`token_type_ids = [0] * len(index_of_words)`
`attention_mask = [1] * len(index_of_words)`
	질문 데이터의 길이를 구하고 그 길이만큼 token_type_ids변수와 attention_mask 변수에 넣음 이 것은 후에 input으로 들어감

`padding_length = max_seq_len - len(index_of_words)`
`index_of_words += [0] * padding_length`
`token_type_ids += [0] * padding_length`
`attention_mask += [0] * padding_length`
	Input 문장 데이터의 길이를 모두 동일하게 맞춰주기 위한 과정

2.2 model.koelectra.ElectraClassificationHead 클래스

Code	해설

`class ElectraClassificationHead(nn.Module):`
`  """Head for sentence-level classification tasks."""`
 	
`  def __init__(self, config, num_labels):`
`  super().__init__()`
`  self.dense = nn.Linear(config.hidden_size, 4*config.hidden_size)`
`  self.dropout = nn.Dropout(config.hidden_dropout_prob)`
`  self.out_proj = nn.Linear(4*config.hidden_size, num_labels)`
 	
`  def forward(self, features, **kwargs):`
`  x = features[:, 0, :]  # take <s> token (equiv. to [CLS])`
`  x = self.dropout(x)`
`  x = self.dense(x)`
`  x = get_activation("gelu")(x)  # although BERT uses tanh here, it s`
`  x = self.dropout(x)`
`  x = self.out_proj(x)`
`  return x`
	순전파가 하나 돌아갈때 fine tune 구조.
	2개의 dropout layer 계층을 줌으로서 overfitting을 방지하고자 함

2.3 model.koelectra.koElectraForSequenceClassification 클래스

Code	해설

`# 분류 레이블이 2개면 다중 분류 mean square err, 그 이상이면 crossentropy`
`  if labels is not None:`
`  if self.num_labels == 1:`
`  #  We are doing regression`
`  loss_fct = MSELoss()`
`  loss = loss_fct(logits.view(-1), labels.view(-1))`
`  else:`
`# crossentropy는 데이터 불균형 문제에서 취약. 논문참조 focal loss`
`  #loss_fct = CrossEntropyLoss()`
`  #loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))`
`  loss = focal_loss(logits.view(-1, self.num_labels), labels.view(-1),`
`  outputs = (loss,) + outputs`
	Loss 값 계산 과정
	레이블이 2개면 MSE를 사용
	그 이상이면 Focal loss 사용

```python
def koelectra_input(tokenizer, str, device = None, max_seq_len = 512):
  index_of_words = tokenizer.encode(str)
  attention_mask = [1] * len(index_of_words)

  # Padding Length
  padding_length = max_seq_len - len(index_of_words)
  # Zero Padding
  index_of_words += [0] * padding_length
  attention_mask += [0] * padding_length

  data = {\
    'input_ids': torch.tensor([index_of_words]).to(device),
    'attention_mask': torch.tensor([attention_mask]).to(device),
  }
  return data
```

Input 문장을 모델에 집어 넣기 위해 형식을 변경 해주는 함수

맡은 역할	1. 조장 2. pretrain 모델 코드 분석 3. fine tuning 설계 및 AI모델 제작 4. 챗봇 아이디어 제공 5. 요구사항 설명서 작성 6. 보고서 작성 7. 웹 프로그래밍 8. 데이터 전처리
프로젝트를 진행하면서 어려웠던 점	**1. 팀원 한명의 따라오는 속도가 다소 느렸음.** -> 능력의 문제라 어쩔수 없었기에 빠르게 능력 파악 후 충분히 할 있는 업무를 할당 **2. 팀원 한명이 졸업작품에 투자하는 시간이 매우 적었음.** -> 소통의 부재 또는 의지의 문제라고 판단하였음. 꾸준히 식사 자리와 삶을 물어보면서 소통 길을 열어두고 멘탈을 케어 하고자 하였음. 딱히 다른 이유가 없음에도 불구하고 5개월에 걸친 권유와 기다림과 믿음에도 변화가 없자 강력하게 경고 . **3. 팀원들의 기여도가 줄어들수록 나의 일이 너무 많아짐** -> 버티고 버텼음. 그리고 1년이 지난 지금 인성과 실력면에서 많이 성장하였음. **4. 데이터 불균형 문제** -> 레이블간의 데이터의 개수가 차이가 나서 pytorch라이브러리에서 제공하는 cross entropy 함수로만 계산하면 개수가 적은 레이블의 데이터는 지나치게 적게 훈련이 되어 정확도가 떨어지는 현상이 나타났음. cross entropy의 함수를 변형시켜서 만든 Focal Loss를 자체적으로 작성하여 개수가 적은 레이블에 대해서 가중치를 많이 줌으로서 해결함

깨달은 점	1. 혼자서는 절대 양질의 프로젝트를 완성시킬 수 없음 2. 프로젝트에서 가장 중요한 것은 협업 3. 사업 아이디어 내는 것이 정말 힘듦 4. 듣기 싫은 소리도 리더로서 해야 할 때가 있음 5. 발표자료는 측량 가능한 수치로 할 것 6. 보고서 쓸 때 단위 표기가 매우 중요함

프로젝트 – 유니티 메타버스 시스템 개발

개발 기간	2022.03.23 ~ 2022.05.08
개발 환경	Unity 2020.03.33 / VisualStudio 2019 / C#
개발 인원	3명
요 약	웹기반 메타버스 플랫폼 운영
기능	**1. 임의 위치 동영상 재생** **2. 임의 위치 유튜브 링크 재생** **3. 유니티 내 웹페이지 연동** **4. SNS 연동** **5. 방명록 구현** **6. 게시판 구현** **7. 채팅 구현 (텍스트, 영상)** **8. 아바타 기본 동작 (걷기, 손흔들기, V포즈, 손흔들기)** **9. 특정 아바타랑 나의 아바타랑 사진 찍기**
실행 화면	

1. 개발 기획

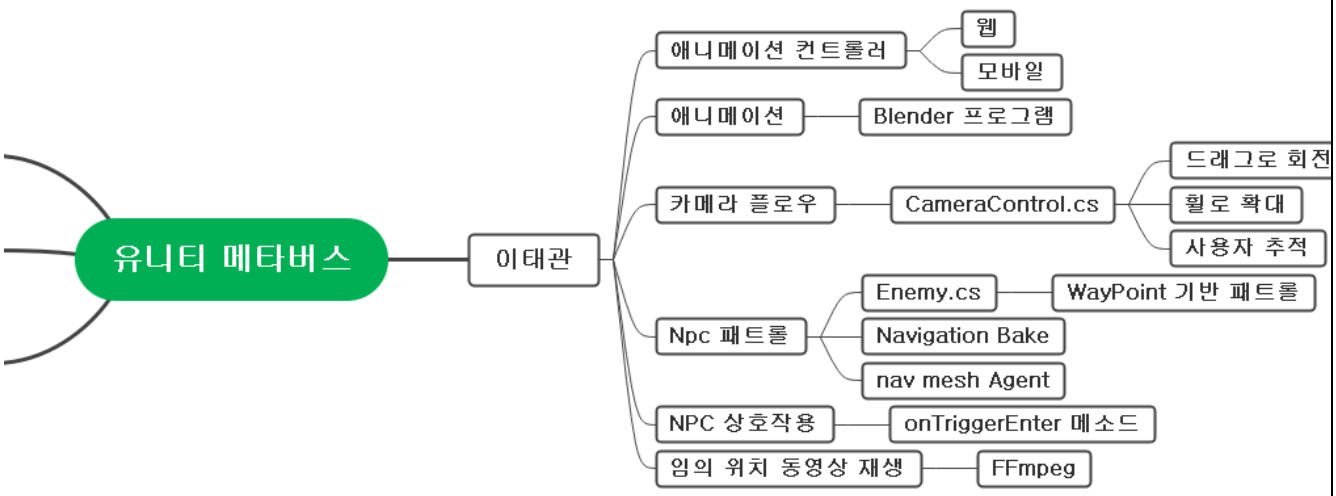
MileStone	**03.23**	**03.30**	**04.06**	**04.13**	**04.20**	**04.27**	**05.04**	**05.09**	**05.16**	**05.23**	**05.30**	**06.06**
기능구현												
동영상 재생												
유튜브 재생												
웹페이지 연동												
SNS 연동												
방명록												
게시판												
채팅												
아바타												
애니메이팅												
상호작용												
카메라 플로우												

Npc 패트롤												
운영테스트												
부하 테스트												
추가 수정 작업												

1.1 요구사항 : (전 말단이라 정확하게는 잘 모릅니다)

1. 디자인을 제외한 흔한 쿼터뷰 게임에서 볼 수 있는 기능들
2. 웹 기반 메타버스 플랫폼 운영
3. Unity 기반 메타버스 플랫폼 구축
4. 메타버스 내 아바타 및 물체 컨트롤 기능 구현
5. 전광판 내 영상 재생(스트리밍도 되게)
6. 채팅 기능 구현
7. 응원 게시판 구현

1.3 마인드맵을 이용해 필요한 기능과 로직 분석.



2.1 CameraControl 클래스 : 카메라의 줌, 휠업시의 확대 휠 다운시 축소 및 카메라 회전 역할

Code	해설

```
// by태관 카메라 줌. 휠업시 확대 휠 다운시 축소
void Zoom()
{
    if (Input.GetAxis("Mouse ScrollWheel") != 0)
    {
        Distance += Input.GetAxis("Mouse ScrollWheel") * ZoomSpeed * -1;

        AxisVec = transform.forward * -1;

        if (Input.GetAxis("Mouse ScrollWheel") > 0)
        {
            if (Distance <= 9.8f)
            {
                Distance = 9.8f;
            }
            else
            {
                AxisVec *= (Distance * -1);
                transform.position = MainCamera.position + AxisVec;
            }
        }
        else
        {
            if (Distance >= 10.1f)
            {
                Distance = 10.1f;
            }
            else
            {
                AxisVec *= Distance;
                transform.position = MainCamera.position + AxisVec;

            }
        }
    }
}
```

마우스 휠을 돌려서 위로 돌렸을때는 확대(Distance를 줄이기)하고 아래로 돌리면 축소(Distance를 늘리기) 하는 기능.
최대확대(최소 Distance)와 최소 축소(최대 Distance)는 각각 9.8, 10.1로 설정 해놓은 모습

```
// by태관 카메라 회전.
void CameraRotation()
{
    if (transform.rotation != TargetRotation)
        transform.rotation = Quaternion.Slerp(transform.rotation, TargetRotation, RotationSpeed * Time.deltaTime);

    if (Input.GetMouseButton(1))
    {
        // 값을 축적.
        Gap.x += Input.GetAxis("Mouse Y") * RotationSpeed * -1;
        Gap.y += Input.GetAxis("Mouse X") * RotationSpeed;

        // 카메라 회전범위 제한.
        Gap.x = Mathf.Clamp(Gap.x, -5f, 85f);
        // 회전 값을 변수에 저장.
        TargetRotation = Quaternion.Euler(Gap);

        // 카메라벡터 객체에 Axis객체의 x,z회전 값을 제외한 y값만을 넘긴다.
        Quaternion q = TargetRotation;
        q.x = q.z = 0;
        CameraVector.transform.rotation = q;
    }
}
```

쿼터뷰 게임의 회전을 구현
마우스 오른쪽 벝튼을 누르면 회전 쿼터니언 값을 받아와서 회전 후 카메라는 target(Player)를 정 중앙에 오게끔 다시 움직인다

2.2 Player 클래스 : Player의 움직임과 애니메이션컨트롤러의 연동

Code	해설

```
void Move()
{
    moveVec = new Vector3(hAxis, 0, vAxis).normalized;

    if (isDodge)
    {
        moveVec = dodgeVec;
    }

    if (wDown)
    {
        transform.position += moveVec * speed * 0.3f * Time.deltaTime;
    }
    else
    {
        transform.position += moveVec * speed * Time.deltaTime;
    }
    anim.SetBool("isRun", moveVec != Vector3.zero);
    anim.SetBool("isWalk", wDown);
}
```

캐릭터가 움직이는 것을 구현.

wDown은 걷기임

```
void Jump()
{
    if (jDown && !isJump && moveVec == Vector3.zero)
    {
        rigid.AddForce(Vector3.up * 15, ForceMode.Impulse);
        isJump = true;
        anim.SetBool("isJump", true);
        anim.SetTrigger("doJump");
    }
}
```

```
void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.tag == "Floor")
    {
        anim.SetBool("isJump", false);
        isJump = false;
    }
}
```

캐릭터가 점프 하는 것을 구현.

두번째 메소드는 캐릭터가 바닥과 닿았을때 애니메이션 컨트롤러에게 '점프 끝났다' 라고 알려줌

```
void Dodge()
{
    if (jDown && !isJump && moveVec != Vector3.zero && !isDodge)
    {
        dodgeVec = moveVec;
        speed *= 2;
        anim.SetTrigger("doDodge");
        isDodge = true;

        Invoke("DodgeOut", 0.4f);
    }
}

void DodgeOut()
{
    speed *= 0.5f;
    isDodge = false;
}
```

캐릭터가 대쉬를 했했을 때 구현.

2.3 GameManager 클래스 : Agora asset 사용하여 유니티 내 영상통화 기능

Code	해설

`void onClickAdd() // by태관 참여자가 추가됐을때 화면 오른쪽 리스트에 추가 시키는 함수`
`{`
 	
`GameObject temp = Instantiate(test, new Vector3(0, 0, 0), Quaternion.identity);`
`temp.name = "user" + userCount.ToString();`
`userCount++;`
`temp?.GetComponent<Button>()?.onClick.AddListener(changeObject);`
 	
`GameObject faceContent = GameObject.Find("faceContent");`
`temp.transform.SetParent(faceContent.transform);`
`}`
	1:1의 상황일때는 상대방의 화면을 큰 화면에 위치.
	후에 추가로 사람이 추가되면 오른쪽 리스트뷰에 작은 화면에 상대방들의 화면을 위치
	temp객체에 user(숫자) 이름을 붙이고 faceContent 객체 밑으로 옮김

`void changeObject() // by 태관 / 오른쪽 화면 리스트를 클릭시 실행되는 큰 화면과 교체하는 이벤트 리스너 함수`
`{`
 	
`GameObject clickObject = EventSystem.current.currentSelectedGameObject;`
`if (clickObject.name != "RemoteView"){ // 큰화면 클릭시 반응 x`
`GameObject panel = GameObject.Find("Panel");`
`RectTransform clickObject_T = clickObject.GetComponent<RectTransform>();`
 	
`// 클릭한 화면을 화면밖으로 빼줄림`
`clickObject.transform.SetParent(panel.transform);`
`GameObject changeRemoteObject = GameObject.Find("RemoteView");`
`RectTransform remoteView_T = changeRemoteObject.GetComponent<RectTransform>();`
 	
`clickObject_T.SetSizeWithCurrentAnchors(RectTransform.Axis.Horizontal, remoteView_T.rect.width);`
`clickObject_T.SetSizeWithCurrentAnchors(RectTransform.Axis.Vertical, remoteView_T.rect.height);`
`clickObject.transform.position = changeRemoteObject.transform.position;`
 	
`changeRemoteObject.transform.SetParent(GameObject.Find("faceContent").transform);`
`//remoteView_T.transform.position = new Vector3(0, 0, 0);`
`remoteView_T.SetSizeWithCurrentAnchors(RectTransform.Axis.Horizontal, 100);`
`remoteView_T.SetSizeWithCurrentAnchors(RectTransform.Axis.Vertical, 100);`
 	
`string temp = changeRemoteObject.name;`
`changeRemoteObject.name = clickObject.name;`
`clickObject.name = temp;`
`}`
`}`
	오른쪽 리스트뷰 작은 화면을 클릭하면 그 작은 화면이 큰 화면으로 옮겨지고 큰 화면은 작은 화면으로 옮겨짐

2.4 Enemy 클래스 : 자동차, 사람, 세그웨이 패트롤 기능

Code	해설

`void Update()`
`{`
`//Debug.Log(target);`
 	
`if (Vector2.Distance(new Vector2(transform.position.x,transform.position.z), new Vector2(target.x, target.z)) < 1)`
`{`
`IterateWaypointIndex();`
`UpdateDestination();`
 	
`}`
	WayPoint와 가까워졌을 때 ItererateWayPointIndex()를 호출 하여 다음 waypoint를 찾고 UpdateDestination()을 호출 하여 다음 waypoint로 이동한다

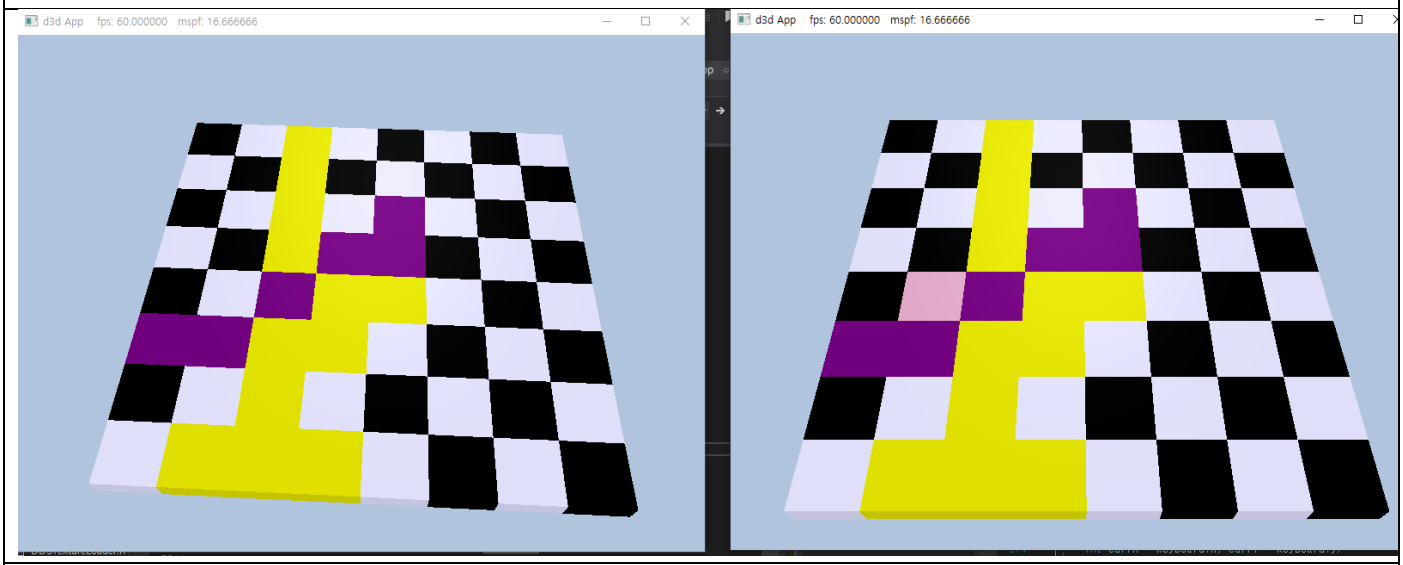
}
public void UpdateDestination()
{
target =waypoints[waypointIndex].position;
agent.SetDestination(target);
}
public void IterateWaypointIndex()
{
waypointIndex++;
if (waypointIndex == waypoints.Length)
{
waypointIndex = 0;
}
}
	Waypoint의 해당 인덱스로 이동한다
	Waypoint 배열의 다음 index를 가져온다

맡은 역할	1. 대량 트래픽을 한번에 감당할수 있는 서버 – 클라이언트 구조 개발 2. 애니메이션 컨트롤러 로직 설계 3. 간단한 애니메이션 제작 4. 쿼터뷰 게임에서 볼 수 있는 카메라 플로우 로직 설계 5. 자동차와 세그웨이, 지하철의 패트롤 설계 6. npc와 부딪혔을때의 상호작용 개발 7. 전광판에서 동영상 재생 8. Agora asset 을 사용한 영상채팅.
프로젝트를 진행하면서 어려웠던 점	**1. photon asset을 사용하면 한 채널에 최대 16명까지 수용 가능하지만 쉽게 구현이 가능하고 연구실 서버 컴퓨터를 활용해서 자체적으로 서버를 돌릴 수 만 있다면 100명이고 200명이고 수용이 가능하다고 교수님이 그렇게 해보라고 요구 하셔서 이틀 정도 집에도 못들어가고 연구실에서 하루종일 해커톤을 했음. 이틀째 밤새고 다음날 오전 9시 교수님 출근하시고 나서 그냥 하던거 폐기하고 photon으로 쉽게 쉽게 가자고 하셨음. 이틀의 노력이 물거품이 되어서 멘탈적으로 힘들었지만 그래도 어디가서 1억짜리 자체 서버 컴퓨터로 서버개발 처음부터 할 기회 없을텐데 귀한 경험했다고 자기암시를 하며 멘탈을 지켜내며 극복.** 2. 우리는 메타버스 플랫폼을 개발만 하면 되었음. 메타버스 장소와 사람 디자인과 애니메이션은 모두 다른 회사가 만들어서 전달해주기로 했는데 너무 늦게 4월 28일에 전달받았음. 때문에 개발일정이 배로 빠듯해졌었음 3. 미리 요구를 하지못한 간단한 애니메이션 (손흔들기, v사진포즈)들이 필요했었는데 그나마 할일이 적었던 내가 구현했음. 아예 생전 처음보는 툴을 갑작스럽게 공부해서 성과를 내야하는게 부담되었지만 진득하게 의자에 하루종일 앉아서 공부함으로서 극복 .

깨달은 점	**1. 개발 계획대로 착착 진행되는 경우는 거의 없다** **2. 연구개발실 분위기 축 처지게 안되게 만들고 집중할 수 있는 환경을 만드는 것도 리더의 능력이다. 그래서 팀장님이 존경스러웠다.** **3. 실력 향상엔 모르더라도 일단 일이 맡겨지면 코피터져가면서 데드라인 지키기위해 발버둥 치는 것이 최고다** **4. 작업을 의뢰한 사람은 본인이 무엇을 원하는지 잘 모른다** **5. 뭘 물어볼땐 바로 앞기수 대학원생에게 물어봐야한다** **6. 개발과 야근은 관련성이 많다. 미리 마음의 준비를 해야한다** **7. 라이브 서비스 할 때 가장 중요한 것은 안정성**

미니프로젝트 – 오셀로 게임 구현

개발 기간	2022.09.27 ~ 2022.10.03
개발 환경	C++ / Visual Studio 2019
개발 인원	1명
요 약	DirectX12를 활용한 오셀로 게임 제작
기능	**5. 키보드 조작을 통한 오셀로 게임 기능** **6. tcp 프로토콜 기반 1:1 기능**
실행 화면	



1.1 요구사항

1. 메이플스토리의 미니게임 배틀리버스를 최대한 구현하는 것을 목표
2. TCP 소켓 프로그래밍으로 1:1 대결 구현
3. 최종은 사용자가 마우스로 클릭함으로서 조작할 수 있게 하는 것을 목표.
4. DirectX12 책의 예제를 적절히 활용

2.1 : ochello_host.sln 주요 코드

Code	해설

```
//스레드 함수, 나중에 class 안에 static으로 해볼것
void fn()
{
    int retval;

    // 윈속 초기화
    WSADATA wsa;
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
        return;

    // socket()
    SOCKET listen_sock = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_sock == INVALID_SOCKET) err_quit("socket()");

    // bind()
    SOCKADDR_IN serveraddr;
    ZeroMemory(&serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
    serveraddr.sin_port = htons(SERVERPORT);
    retval = bind(listen_sock, (SOCKADDR*)&serveraddr, sizeof(serveraddr));
    if (retval == SOCKET_ERROR) err_quit("bind()");

    // listen()
    retval = listen(listen_sock, SOMAXCONN);
    if (retval == SOCKET_ERROR) err_quit("listen()");

    // 데이터 통신에 사용할 변수
    SOCKET client_sock;
    SOCKADDR_IN clientaddr;
    int addrlen;
    char buf[BUFSIZE + 1]; // 2로 줄여볼것
```

TCP 통신 기능을 수행하는 스레드 함수

```
while (1) {
    // accept()
    addrlen = sizeof(clientaddr);
    client_sock = accept(listen_sock, (SOCKADDR*)&clientaddr, &addrlen);
    if (client_sock == INVALID_SOCKET) {
        err_display("accept()");
        break;
    }

    // 접속한 클라이언트 정보 출력
    // 클라이언트와 데이터 통신
    while (1) {
        // 데이터 받기
        retval = recvn(client_sock, buf, 2, 0);
        if (retval == SOCKET_ERROR) {
            err_display("recv()");
            break;
        }
        else if (retval == 0)
            break;

        // 받은 데이터 출력
        theApp->receivedPoint(buf[0], buf[1]);
        theApp->checkChangeBlock(buf[0], buf[1]);
        // 대기했다가
        while (!theApp->received) // 비효율적. 다른 방법 생각해볼것
        {

        }
        theApp->received = false;
        // 데이터 보내기
        buf[0] = theApp->getX() + '0';
        buf[1] = theApp->getY() + '0';
        theApp->checkChangeBlock(buf[0], buf[1]);

        retval = send(client_sock, buf, retval, 0);
        if (retval == SOCKET_ERROR) {
            err_display("send()");
            break;
            break;
        }
    }

    // closesocket()
    closesocket(client_sock);
}

// closesocket()
closesocket(listen_sock);

// 윈속 종료
WSACleanup();
return;
```

프레임 n에서

```cpp
void LitColumnsApp::Update(const GameTimer& gt)
{
    OnKeyboardInput(gt);
    UpdateCamera(gt);

    // 순환적으로 자원 프레임의 다음 원소에 접근한다
    mCurrFrameResourceIndex = (mCurrFrameResourceIndex + 1) % gNumFrameResources;
    mCurrFrameResource = mFrameResources[mCurrFrameResourceIndex].get();

    /*
    GPU가 현재 프레임 자원의 명령들을 다 처리했는지 확인.
    아직 다 처리하지 않았으면 GPU가 이 울타리 지점까지의 명령들을 처리할때까지 기다린다
    */
    if (mCurrFrameResource->Fence != 0 && mFence->GetCompletedValue() < mCurrFrameResource
    {
        HANDLE eventHandle = CreateEventEx(nullptr, false, false, EVENT_ALL_ACCESS);
        ThrowIfFailed(mFence->SetEventOnCompletion(mCurrFrameResource->Fence, eventHandle)
        WaitForSingleObject(eventHandle, INFINITE);
        CloseHandle(eventHandle);
    }

    AnimateMaterials(gt);
    UpdateObjectCBs(gt);
    UpdateMaterialCBs(gt);
    UpdateMainPassCB(gt);
}
```

CPU가 하는 일들.
순환배열을이용하여CPU와 GPU의 균형을 맞춰준다.

```cpp
void LitColumnsApp::Draw(const GameTimer& gt)
{
    auto cmdListAlloc = mCurrFrameResource->CmdListAlloc;

    // 명령 기록에 관련된 메모리의 재활용을 위해 명령 할당자를 재설정.
        // 재설정은 gpu가 관련 명령 목록들을 모두 처리한 후에 일어남
    ThrowIfFailed(cmdListAlloc->Reset());

    //명령목록을 executeCommandList를 통해서 명령 대기열에 추가했다면 명령목록재설정 가능
    // 명령 목록을 재설정하면 메모리가 재활용
    ThrowIfFailed(mCommandList->Reset(cmdListAlloc.Get(), mOpaquePSO.Get()));

    mCommandList->RSSetViewports(1, &mScreenViewport);
    mCommandList->RSSetScissorRects(1, &mScissorRect);

    //자원 용도에 관련된 상태 전이를 direct3d에 통지
    mCommandList->ResourceBarrier(1, &CD3DX12_RESOURCE_BARRIER::Transition(CurrentBackBuffer(),
        D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

    //후면 버퍼와 깊이 버퍼를 지운다
    mCommandList->ClearRenderTargetView(CurrentBackBufferView(), Colors::LightSteelBlue, 0, nul
    mCommandList->ClearDepthStencilView(DepthStencilView(), D3D12_CLEAR_FLAG_DEPTH | D3D12_CLEA

    // 렌더링 결과가 기록될 렌더대상 버퍼들을 지정
    mCommandList->OMSetRenderTargets(1, &CurrentBackBufferView(), true, &DepthStencilView());
```

update() 함수와 연계해서 프레임 자원들을 CPU와 GPU를 모두 효율적이게 사용한다.
GPU에게 계속해서 일거리를 줌으로서 효율적이게 사용할 수 있다

```
    mCommandList->SetGraphicsRootSignature(mRootSignature.Get());

    auto passCB = mCurrFrameResource->PassCB->Resource();
    mCommandList->SetGraphicsRootConstantBufferView(2, passCB->GetGPUVirtualAddress());

    DrawRenderItems(mCommandList.Get(), mOpaqueRitems);

    // Indicate a state transition on the resource usage.
    mCommandList->ResourceBarrier(1, &CD3DX12_RESOURCE_BARRIER::Transition(CurrentBackBuffer(),
        D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

    ThrowIfFailed(mCommandList->Close());

    // 명령큐 실행
    ID3D12CommandList* cmdsLists[] = { mCommandList.Get() };
    mCommandQueue->ExecuteCommandLists(_countof(cmdsLists), cmdsLists);

    // 스왑체인 스왑
    ThrowIfFailed(mSwapChain->Present(0, 0));
    mCurrBackBuffer = (mCurrBackBuffer + 1) % SwapChainBufferCount;

    // 현재 fence 지점까지의 명령들을 표시하도록 fence 멤버를 전진
    mCurrFrameResource->Fence = ++mCurrentFence;
    /*
    새 fence 지점을 설정하는 signal(명령)을 명령 대기열에 추가.
    지금 우리는 GPU 시간선 (timeline) 위에 있으므로 새 fence 지점은
    GPU가 이 signal() 명령 이전까지의 모든 명령을 처리하기 전까지는 설정되지 않음
    */
    mCommandQueue->Signal(mFence.Get(), mCurrentFence);
}
```

```
struct RenderItem
{
    RenderItem() = default;

    // 뭐 그 늘 보던 세계공간 기준으로 물체의 국소공간 서술하는 세계행렬
    // 이 행렬은 세계 공간 안에서 물체의 위치와 방향 크기를 결정
    XMFLOAT4X4 World = MathHelper::Identity4x4();

    /*
    물체의 자료가 변해서 상수 버퍼를 갱신해야하는 지의 여부를 뜻하는 '더러움' 플래그
    FrameResource마다 물체의 cbuffer가 있으므로 FrameResource마다 갱신을 적용해야함
    따라서 물체의 자료를 수정할땐 반드시 NumFramesDirty = gNumFrameResources로 설정해야함
    그래야 각각의 프레임 자원이 갱신된다
    */
    int NumFramesDirty = gNumFrameResources;

    // 이 렌더항목의 물체 상수에 해당하는 GPU 상수 버퍼의 인덱스
    UINT ObjCBIndex = -1;

    MeshGeometry* Geo = nullptr;

    // Primitive topology.
    D3D12_PRIMITIVE_TOPOLOGY PrimitiveType = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST;

    // DrawIndexedInstances 매개변수들
    UINT IndexCount = 0;
    UINT StartIndexLocation = 0;
    int BaseVertexLocation = 0;
};
```

렌더링 하는 물체의 정보들을 담은 구조체.
물체의 세계 행렬과 정점버퍼 정보, 인덱스버퍼 정보 위상 기본정보 등을 담는다

```cpp
//상수 버퍼 업데이트
void LitColumnsApp::UpdateObjectCBs(const GameTimer& gt)
{
    auto currObjectCB = mCurrFrameResource->ObjectCB.get();
    for (auto& e : mAllRitems)
    {
        /*
        상수들이 바뀌었을때만 cubuffer 자료 갱신
        이러한 갱신은 저기 매개변수 오듯 프레임 자원마다 갱신 해야함 당연한거
        */
        if (e->NumFramesDirty > 0)
        {
            XMMATRIX world = XMLoadFloat4x4(&e->World);
            XMMATRIX texTransform = XMLoadFloat4x4(&e->TexTransform);

            ObjectConstants objConstants;
            XMStoreFloat4x4(&objConstants.World, XMMatrixTranspose(world));
            XMStoreFloat4x4(&objConstants.TexTransform, XMMatrixTranspose(texTransform));

            currObjectCB->CopyData(e->ObjCBIndex, objConstants);

            // 다음 프레임 자원으로 넘어감
            e->NumFramesDirty--;
        }
    }
}
```

update 함수가 프레임당 한번씩 호출. 상수 버퍼들이 만약 수정해야한다면 수정 기능을 수행. 아니면 그냥 넘어감

```cpp
/*
    passCB == FrameResource.h에서 변하지 않는 상수 자료를 저장.
    ex) 시점 위치, 시야 행렬, 투영행렬 등등..
*/
void LitColumnsApp::UpdateMainPassCB(const GameTimer& gt)
{
    XMMATRIX view = XMLoadFloat4x4(&mView);
    XMMATRIX proj = XMLoadFloat4x4(&mProj);

    XMMATRIX viewProj = XMMatrixMultiply(view, proj);
    XMMATRIX invView = XMMatrixInverse(&XMMatrixDeterminant(view), view);
    XMMATRIX invProj = XMMatrixInverse(&XMMatrixDeterminant(proj), proj);
    XMMATRIX invViewProj = XMMatrixInverse(&XMMatrixDeterminant(viewProj), viewProj);

    XMStoreFloat4x4(&mMainPassCB.View, XMMatrixTranspose(view));
    XMStoreFloat4x4(&mMainPassCB.InvView, XMMatrixTranspose(invView));
    XMStoreFloat4x4(&mMainPassCB.Proj, XMMatrixTranspose(proj));
    XMStoreFloat4x4(&mMainPassCB.InvProj, XMMatrixTranspose(invProj));
    XMStoreFloat4x4(&mMainPassCB.ViewProj, XMMatrixTranspose(viewProj));
    XMStoreFloat4x4(&mMainPassCB.InvViewProj, XMMatrixTranspose(invViewProj));
    mMainPassCB.EyePosW = mEyePos;
    mMainPassCB.RenderTargetSize = XMFLOAT2((float)mClientWidth, (float)mClientHeight);
    mMainPassCB.InvRenderTargetSize = XMFLOAT2(1.0f / mClientWidth, 1.0f / mClientHeight);
    mMainPassCB.NearZ = 1.0f;
    mMainPassCB.FarZ = 1000.0f;
    mMainPassCB.TotalTime = gt.TotalTime();
    mMainPassCB.DeltaTime = gt.DeltaTime();
    mMainPassCB.AmbientLight = { 0.25f, 0.25f, 0.35f, 1.0f };
    mMainPassCB.Lights[0].Direction = { 0.57735f, -0.57735f, 0.57735f };
    mMainPassCB.Lights[0].Strength = { 0.6f, 0.6f, 0.6f };
    mMainPassCB.Lights[1].Direction = { -0.57735f, -0.57735f, 0.57735f };
    mMainPassCB.Lights[1].Strength = { 0.3f, 0.3f, 0.3f };
    mMainPassCB.Lights[2].Direction = { 0.0f, -0.707f, -0.707f };
    mMainPassCB.Lights[2].Strength = { 0.15f, 0.15f, 0.15f };

    auto currPassCB = mCurrFrameResource->PassCB.get();
    currPassCB->CopyData(0, mMainPassCB);
}
```

update() 함수에서 프레임당 한번 호출 passCB 별 상수 버퍼 갱신. 상수 버퍼들이 만약 수정해야한다면 수정 기능을 수행. 아니면 그냥 넘어감

```cpp
void LitColumnsApp::BuildRootSignature()
{
    /*
    일반적으로 셰이더 프로그램은 특정 자원 (상수 버퍼, 텍스처, 표본 추출기 등)이 입력된다고 기대함
    루트 서명은 셰이더 프로그램이 기대하는 자원들을 정의한다
    셰이더 프로그램은 본질적으로 하나의 함수이고 셰이더에 입력되는 자원들은 함수의 매개변수에 해당한다
    따라서 루트서명은 곧 함수 수명을 정의하는 수단이라고 볼 수 있음
    */

    // 루트 매개변수는 서술자 테이블이거나 루트 서술자 또는 루트 상수.
    CD3DX12_ROOT_PARAMETER slotRootParameter[3];

    // cbv 하나를 담는 서술자 테이블을 생성한다
    slotRootParameter[0].InitAsConstantBufferView(0);
    slotRootParameter[1].InitAsConstantBufferView(1);
    slotRootParameter[2].InitAsConstantBufferView(2);

    //루트 서명은 루트 매개변수들의 배열.
    CD3DX12_ROOT_SIGNATURE_DESC rootSigDesc(3, slotRootParameter, 0, nullptr, D3D12_ROOT_SIGNATURE_FLAG_AL

    //상수 버퍼 하나로 구성된 서술자 구간을 가리키는 슬롯하나로 이루어진 루트 서명 생성
    ComPtr<ID3DBlob> serializedRootSig = nullptr;
    ComPtr<ID3DBlob> errorBlob = nullptr;
    HRESULT hr = D3D12SerializeRootSignature(&rootSigDesc, D3D_ROOT_SIGNATURE_VERSION_1,
        serializedRootSig.GetAddressOf(), errorBlob.GetAddressOf());

    if (errorBlob != nullptr)
    {
        ::OutputDebugStringA((char*)errorBlob->GetBufferPointer());
    }
    ThrowIfFailed(hr);

    ThrowIfFailed(md3dDevice->CreateRootSignature(
        0,
        serializedRootSig->GetBufferPointer(),
        serializedRootSig->GetBufferSize(),
        IID_PPV_ARGS(mRootSignature.GetAddressOf())));
}
```

루트 서명 설정.

셰이더 프로그램이 기대하는 자원들을 정의

```cpp
// 박스를 조합해서 체스판을 만듦
void LitColumnsApp::BuildRenderItems()
{
    UINT ObjCBind = 0;
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            auto boxRitem = std::make_unique<RenderItem>();
            // 세계 행렬 정의.
            XMStoreFloat4x4(&boxRitem->World, XMMatrixScaling(2.0f, 2.0f, 2.0f) * XMMatrixTranslation(-10.0f +(
            // 크기 행렬 정의
            XMStoreFloat4x4(&boxRitem->TexTransform, XMMatrixScaling(1.0f, 1.0f, 1.0f));
            boxRitem->ObjCBIndex = ObjCBind;
            ObjCBind++;

            //흰색 검정, 가운데 4타일 대로 체스판 색을 초기화
            if (initColor[i][j]==1)
            {
                boxRitem->Mat = mMaterials["stone1"].get();
            }
            else if (initColor[i][j] == 0)
            {
                boxRitem->Mat = mMaterials["stone0"].get();
            }
            else if (initColor[i][j] == -1)
            {
                boxRitem->Mat = mMaterials["user1"].get();
            }
            else if (initColor[i][j] == -2)
            {
                boxRitem->Mat = mMaterials["user2"].get();
            }
```

박스 Render와 material을 조합하여 8x8의 체스판을 만들고 색깔을 초기화한다

```
        else if (initColor[i][j] == -2)
        {
            boxRitem->Mat = mMaterials["user2"].get();
        }
        boxRitem->Geo = mGeometries["shapeGeo"].get();
        boxRitem->PrimitiveType = D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST;
        boxRitem->IndexCount = boxRitem->Geo->DrawArgs["box"].IndexCount;
        boxRitem->StartIndexLocation = boxRitem->Geo->DrawArgs["box"].StartIndexLocation;
        boxRitem->BaseVertexLocation = boxRitem->Geo->DrawArgs["box"].BaseVertexLocation;
        mAllRitems.push_back(std::move(boxRitem));

    }
}
for (auto& e : mAllRitems)
{
    mOpaqueRitems.push_back(e.get());
}
```

```
// 사용자 입력. 돌을 놓거나 놓을 위치 이동
void LitColumnsApp::OnKeyboardDown(WPARAM btnState)
{
    TCHAR pressedChar = static_cast<TCHAR>(btnState);
    // 사용자가 space를 눌렀을때 그게 가능한지 판단 후 돌을 놓는다
    if (btnState == VK_SPACE && haveControl && initColor[keyboard.x][keyboard.y] >= 0)
    {
        mOpaqueRitems[keyboard.x + keyboard.y + 8]->Mat = mMaterials["user1"].get();
        initColor[keyboard.x][keyboard.y] = -1;
        received = true;
        haveControl = false;
        return;
    }

    int currMat = initColor[keyboard.x][keyboard.y];
    int currX = keyboard.x, currY = keyboard.y;

    // 키보드 커서 이동
    if (pressedChar == 'w')
    {
        keyboard.x++;
    }
    else if (pressedChar == 'd')
    {
        keyboard.y++;
    }
    else if (pressedChar == 'a')
    {
        keyboard.y--;
    }
    else if (pressedChar == 's')
    {
        keyboard.x--;
    }
```

```
    if (isRange(keyboard.x, keyboard.y))
    {
        mOpaqueRitems[keyboard.x + keyboard.y + 8]->Mat = mMaterials["stone2"].get();
        if (currMat == 1)
        {
            mOpaqueRitems[currX + currY + 8]->Mat = mMaterials["stone1"].get();
        }
        else if (currMat == 0) {
            mOpaqueRitems[currX + currY + 8]->Mat = mMaterials["stone0"].get();
        }
        else if (currMat == -1) {
            mOpaqueRitems[currX + currY + 8]->Mat = mMaterials["user1"].get();
        }
        else if (currMat == -2) {
            mOpaqueRitems[currX + currY + 8]->Mat = mMaterials["user2"].get();
        }
    }
    else
    {
        mOpaqueRitems[currX + currY + 8]->Mat = mMaterials["stone2"].get();
        keyboard.x = currX;
        keyboard.y = currY;
    }

}
```

사용자 입력 처리. space는 자신의 돌을 체스판에 놓음

wasd는 위, 아래 이동

```
//시간복잡도 주의. 돌이 놓여졌을때 작동. 변경되어야할 색깔들을 변경함
void LitColumnsApp::checkChangeBlock(char x, char y)
{
    int cx = x - '0';
    int cy = y - '0';

    int xp = cx, yp = cy, xm = cx, ym = cy;

    Material* colorMat;
    int color = initColor[cx][cy];

    if (color == -1) { ... }
    else { ... }

    for (int i = 1; i < 8; i++)
    {

        if (++xp < 8)
        {
            // 한번이라도 검정, 흰타일 나오면 끝
            if (initColor[xp][cy] < 0)
            {

                // 검색하는 동안에 흰타일 검정타일 안나왔고, 범위 내 일인데 원래 내 색이 나왔을 때
                if (initColor[xp][cy] == color)
                {

                    // cx+1부터 xp-1 까지 쭉 color의 색으로로 변경
                    for (int j = cx + 1; j < xp; j++)
                    {
                        mOpaqueRitems[j + cy * 8]->Mat = colorMat;
                        initColor[j][cy] = color;
                    }
                    xp += 100;
                }
```

상대방에게 상대방이 어디뒀는지 통신을 받거나 내가 돌을 뒀을때 호출. 색깔이 변경되어야할 위치의 블록의 색을 변경함 일직선, 대각선 전부 탐색

2.2 Ochello_Client2.sln 주요 코드 : host.sln과 중복되는 것은 제외

Code	해설

```
void fn()
{
    int retval;

    // 윈속 초기화
    WSADATA wsa;
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
        return ;

    // socket()
    SOCKET sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock == INVALID_SOCKET) err_quit("socket()");

    // connect()
    SOCKADDR_IN serveraddr;
    ZeroMemory(&serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_addr.s_addr = inet_addr(SERVERIP);
    serveraddr.sin_port = htons(SERVERPORT);
    retval = connect(sock, (SOCKADDR*)&serveraddr, sizeof(serveraddr));
    if (retval == SOCKET_ERROR) err_quit("connect()");

    // 데이터 통신에 사용할 변수
    char buf[BUFSIZE];
    int len;

    // 서버와 데이터 통신
    while (1) {
        // 데이터 입력
        while (!theApp->sended) //비효율적이므로 다른거 생각해보기
        {

        }
        buf[0] = theApp->getX() + '0';
```

host.sln과 tcp 통신을 하는 스레드 함수

```
        buf[0] = theApp->getX() + '0';
        buf[1] = theApp->getY() + '0';

        theApp->checkChangeBlock(buf[0], buf[1]);
        // 데이터 보내기
        retval = send(sock, buf, 2, 0);
        if (retval == SOCKET_ERROR) {
            err_display("send()");
            break;
        }

        // 데이터 받기
        retval = recvn(sock, buf, 2, 0);

        theApp->receivedPoint(buf[0], buf[1]);
        theApp->checkChangeBlock(buf[0], buf[1]);
        if (retval == SOCKET_ERROR) {
            err_display("recv()");
            break;
        }
        else if (retval == 0)
            break;

        // 받은 데이터 처리
    }

    // closesocket()
    closesocket(sock);

    // 윈속 종료
    WSACleanup();
    return ;
}
```

프로젝트를 진행하면서 어려웠던 점	1. 언리얼과 유니티로 편하게 작업하다가 아예 맨땅에 렌더링 하려고 하니까 어색해서 적응기간이 생각보다 오려 걸렸음 2. 책에서도 안나오고 검색해도 잘 안나오는 경험적인 실력이 많이 부족해서 비효율적인 구현을 많이 하게 되었음 3. 비동기 스레드 실전 프로그래밍 실력이 부족함을 깨달았음. 4. CommandList를 멀티스레드로 여러 개를 생성하여 멀티스레드 렌더링이 필수인데 몇 개가 가장 최적화 된 개수인지 알기가 쉽지 않았음. (현재 진행, 아직 해결못함) 5. Vertex Buffer View와 Index Buffer View의 Alignment가 64kb라서 무조건 64kb의 용량으로 생성이된다.(상수버퍼도 256kb단위) 이때 index buffer의 크기가 4kb라면 60kb의 용량을 낭비하게 된다. 이 현상으로 인하여 GPU 메모리 낭비가 심한 상태이다. 이것을 해결하기 위해 여러곳에 서칭을 해봤지만 서칭이 쉽지 않았고, 힌트를 얻기 위해 현재 DirectX12 게임을 서비스 중인 문명6 개발팀에게 문의를 해보았으나 긍정적인 답변을 받지 못하고 해결을 못하고 있음. 6. 정점 셰이딩 단계에서 절두체 생성 부분 이해하는데 시간이 오래 걸렸음
깨달은 점	**1. 난 모르는게 많다. 항상 정진하자** **2. 상용엔진 사용법도 물론 중요하지만 그래픽스 기본과 기초cs 지식이 상당히 중요하다.**