Unreal Portfolio

김성호

목차

1. 간략한 게임설명
2. 커맨드 시스템
3. Non-Playable character AI
4. UI/UX

1. 게임설명

- 언리얼엔진으로 만든 쿼터뷰 시점의 게임으로 커맨드를 입력하여 스킬을 사용할 수 있도로록 만든 게임입니다.

- 마우스 우클릭으로 지정된 곳을 플레이어가 NavMesh이용하여 이동하고, WASD 키로 커맨드를 입력, 마우스 왼클릭으로 사용하는 시스템을 사용했습니다.

- 드래그로도 이동이 가능하게 구현하였습니다.

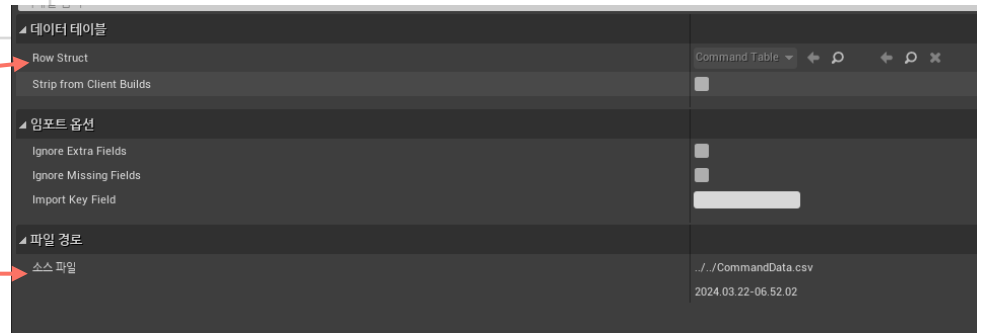
2.커맨드 시스템

-데이터 테이블 Row Struct 형식을 Cpp에서 지정한 후에

이에 맞는 CSV 파일을 연결하여 외부에서 쉽게 커맨드를 추가/제거 할 수 있도록 만들었습니다.

```
USTRUCT(BlueprintType)
struct FCommandTable : public FTableRowBase
{
    GENERATED_BODY()

public:


    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Data")
        FString Command;
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Data")
        FString Text;
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Data")
        int32 SkillDamage;
    /*UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Data")
        int32 Stamina;*/

    //생성자 및 초기화
    FCommandTable() :Command(""), Text(""), SkillDamage()
    {

    };
};
UCLASS()
class PORTFOLIO_CPP_API ACommandDataTable : public AActor
{
    GENERATED_BODY()

};
```

	A	B	C	D
	Command	CommandText		SkillDamage
	41	41	JangPoon	15
	121	121	Hurricanek	4
	4	4	Dodge	0
	NoCommand	NoComma	NormalAtt	10
	421	421	FlyingKick	20



```cpp
#include "PFGameInstance.h"
#include "CommandDataTable.h"

//게임컴파일할때 로드됩니다.
UPFGameInstance::UPFGameInstance()
{
    FString CharacterDataPath = TEXT("DataTable'/Game/Blueprints/CommandData.CommandData'");
    static ConstructorHelpers::FObjectFinder<UDataTable> DtTable(*CharacterDataPath);

    if (DtTable.Succeeded())
    {
        commandTable = DtTable.Object;
        UE_LOG(LogTemp, Warning, TEXT("data Table LoadComplete"));
        //테스트 코드
        //tableSize = DtTable.Object->GetTableData().Num();
    }
}
//테이블 데이터를 찾는 함수
FCommandTable* UPFGameInstance::GetTableData(FString column)
{
    FCommandTable* temp;
    temp = commandTable->FindRow<FCommandTable>(*FString(column), TEXT(""));
    if (temp != nullptr)
    {
        TextOut = temp->Text;
        return temp;
    }
    return nullptr;
}
```

이렇게 만들어진 커맨드 테이블 데이터를 게임인스턴스 생성자에서 로드 될 수 있도록 합니다.

또한 테이블 데이터를 얻을 수 있는 함수 GetTableData를 만들어 이후 플레이어가 입력한 커맨드에 대한 데이터를 얻어 올 수 있도록 하였습니다.

```cpp
void AMainPlayer::InputRight()
{
    mCommand = COMMAND::Right;
    UE_LOG(Player, Warning, TEXT("Right : %d"), mCommand);
    commandQueue.push(mCommand);
    CommandTimeOut();
}

void AMainPlayer::InputLeft()
{
    mCommand = COMMAND::Left;
    UE_LOG(Player, Warning, TEXT("Left : %d"), mCommand);
    commandQueue.push(mCommand);
    CommandTimeOut();
}

void AMainPlayer::InputUp()
{
    mCommand = COMMAND::Up;
    UE_LOG(Player, Warning, TEXT("Up : %d"), mCommand);
    commandQueue.push(mCommand);
    CommandTimeOut();
}

void AMainPlayer::InputDown()
{
    mCommand = COMMAND::Down;
    UE_LOG(Player, Warning, TEXT("Down : %d"), mCommand);
    commandQueue.push(mCommand);
    CommandTimeOut();
}
```

플레이어가 커맨드키(WASD) 입력 시 입력중인 커맨드 정보를 가진 Queue에 Push되어집니다.

CommandTimeOut함수를 통해 일정 시간이 지나면 입력되어있는 커맨드가 초기화 되도록 하였습니다.

```cpp
void AMainPlayer::TableRead(FString InputCommand, int32& damage)
{
    UseSkill.Unbind();//이미 바인드 되어있는거 해제
    if (thisGameInstance == nullptr) return; // 게임인스턴스가 설정하지 않았을 경우 등..

    FCommandTable* temp; //커맨드 테이블(행)을 저장할 임시변수
    if (InputCommand.IsEmpty())
        temp = thisGameInstance->GetTableData(FString("NoCommand"));//NormalAttack
    else
        temp = thisGameInstance->GetTableData(InputCommand); // PFGameInstance에 구현한 GetTableData사용 InputCommand와 이름이 같은 행을 찾아 반환

    if (temp != nullptr) // 찾은 경우
    {
        int32 skillDMG = temp->SkillDamage;
        UE_LOG(Player, Warning, TEXT("OutputCommand : %s"), *thisGameInstance->TextOut); //출력로그에 출력
        UseSkill.BindUFunction(this, *thisGameInstance->TextOut); //해당 커맨드와 함수를 바인딩
        damage = skillDMG;

    }
    else //못 찾은 경우
    {
        UseSkill.BindUFunction(this, "NormalAttack");
        UE_LOG(Player, Warning, TEXT("Skill not found, faild TableReading"));
        SkillUI = "NormalAttack";
        return;
    }
}
```

TableRead 함수를 통해 입력된 커맨드의 정보를 가져와 알맞은 함수를 델리게이트에 Bind하여 플레이어가 올바르게 입력했다면 해당하는 스킬을 사용하고,
커맨드가 제대로 입력되지 않았을 경우와 일반공격을 호출할 때는 NoCommand 판정을 주어 일반 공격이 나가도록 하였습니다.

3. Non-PlayableCharacter AI

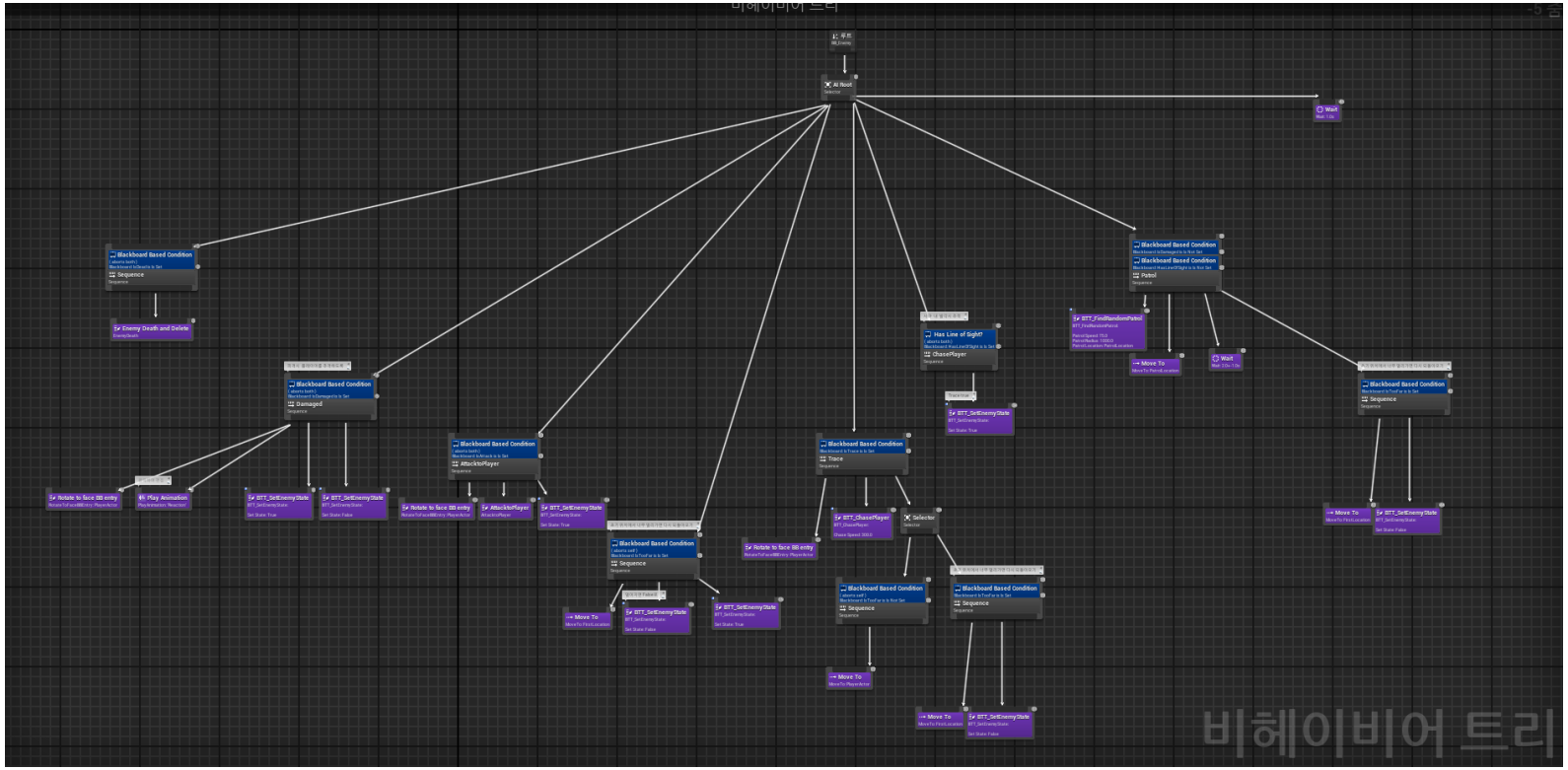
-AI는 일반형 적과 보스로 나누어 제작하였으며 FSM과 BehaviorTree를 이용하였습니다.

-상태에 따라 BehaviorTree의 블랙보드 데코레이터를 통하여 AI의 흐름을 제어하는 구조로 만들었습니다.

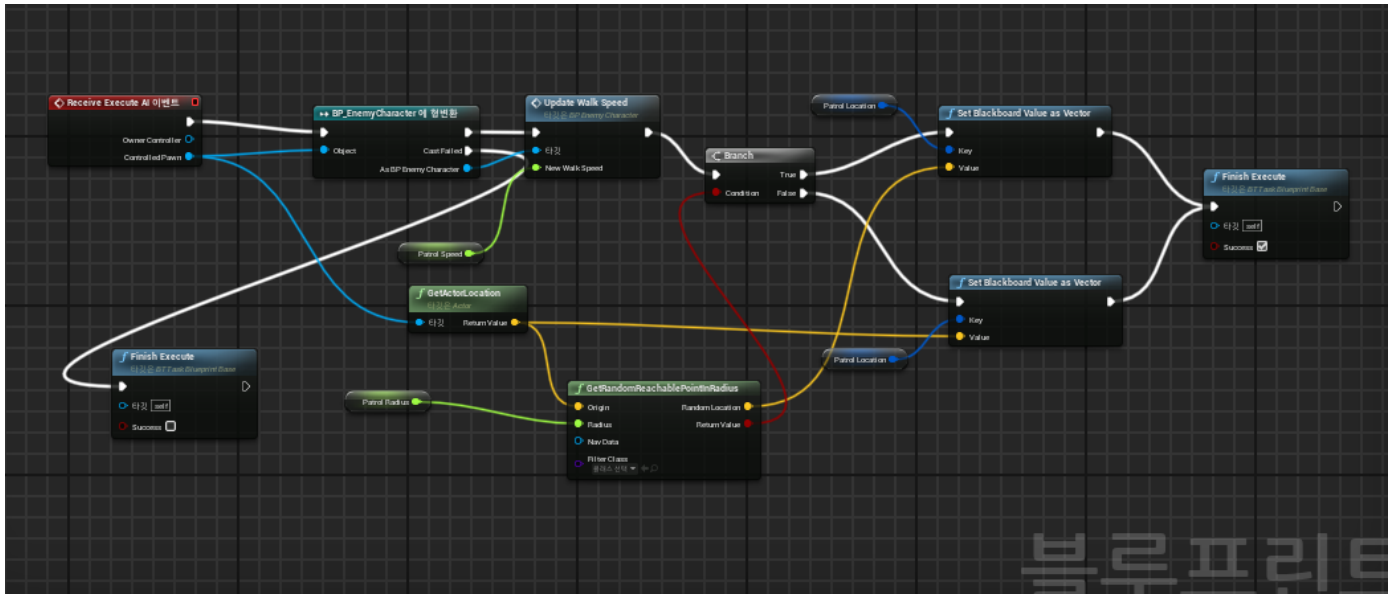
-모든 AI 는 NavMesh를 따라 움직이도록 설계되었습니다.

3-1. 일반형 적 AI

- 시야에 플레이어가 들어올 경우 추격합니다.
- 플레이어로부터 데미지를 입었을 경우 추격합니다.
- 추격상태가 아니라면 최초 스폰위치로부터 랜덤한 지점을 계속 순찰합니다.
- 순찰하거나 추격시 최초스폰위치로부터 일정 거리이상 멀어질 경우 되돌아 갑니다.
-데미지를 입었을 경우 피격애니메이션이 재생됩니다.
- 플레이어가 공격범위내 있는 경우 플레이어를 공격합니다. 이때 Kick과 Punch중 랜덤으로 공격합니다.



각 상태에 대한 부분을 블랙보드 데코레이터를 이용하여 해당 부분의 흐름을 관리합니다.



랜덤한 지점을 순찰하도록 임의의 지점을 선택하는 블루프린트 내용입니다

.

```cpp
//블랙보드의 SelfActor의 값 == 컨트롤 되어지는 EnemyActor 를 ControlledEnemy에 할당 후 Attack 함수 실행
EBTNodeResult::Type UAttacktoPlayer::ExecuteTask(UBehaviorTreeComponent & OwnerComp, uint8 * NodeMemory)
{
    EBTNodeResult::Type Result = Super::ExecuteTask(OwnerComp, NodeMemory);

    if (!ControlledEnemy->isMontagePlaying)
    {
        ControlledEnemy->Attack();
    }


    return EBTNodeResult::InProgress;

}
```

필요한 경우 BehaviorTreeTask Node를 C++를 이용하여 구현하여 사용하였습니다 .
이 코드는 플레이어를 공격할때 이용되는 노드를 구현한 것 이며 TickTask를 이용하여 공격 AnimationMontage 가 끝날때 까지 다른 노드로 넘어가지 않도록 컨트롤합니다.

```cpp
void UAttacktoPlayer::TickTask(UBehaviorTreeComponent& OwnerComp, uint8* NodeMemory, float DeltaSeconds)
{
    Super::TickTask(OwnerComp, NodeMemory, DeltaSeconds);

    //UE_LOG(LogTemp, Warning, TEXT("Attack Task tick"));

    if (!ControlledEnemy -> isMontagePlaying)
    {
        FinishLatentTask(OwnerComp, EBTNodeResult::Succeeded);
    }
    else
    {
        FinishLatentTask(OwnerComp, EBTNodeResult::InProgress);
    }
}
```

3-2. 보스형 적 AI

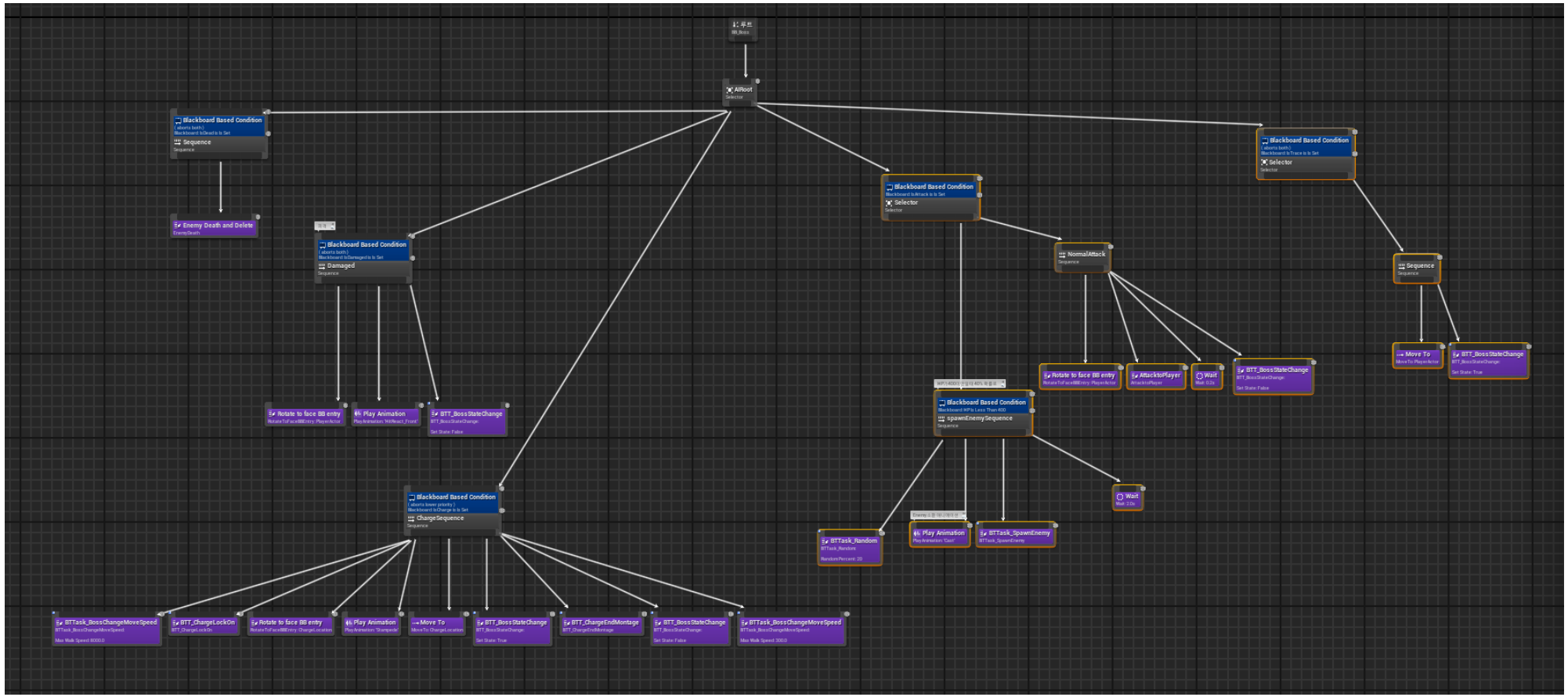
-기본적으로 플레이어를 향해 움직입니다.

-너무 멀어졌을 경우 돌진공격을 하여 거리를 좁힙니다. 이때 데미지주도록 만들었습니다.

-일정 체력 이하가 되었을 경우 공격 모션이 변화합니다.

이때 20% 확률로 공격상태로 들어갈 시 일반형 적을 보스근처 랜덤한 위치에 소환합니다.

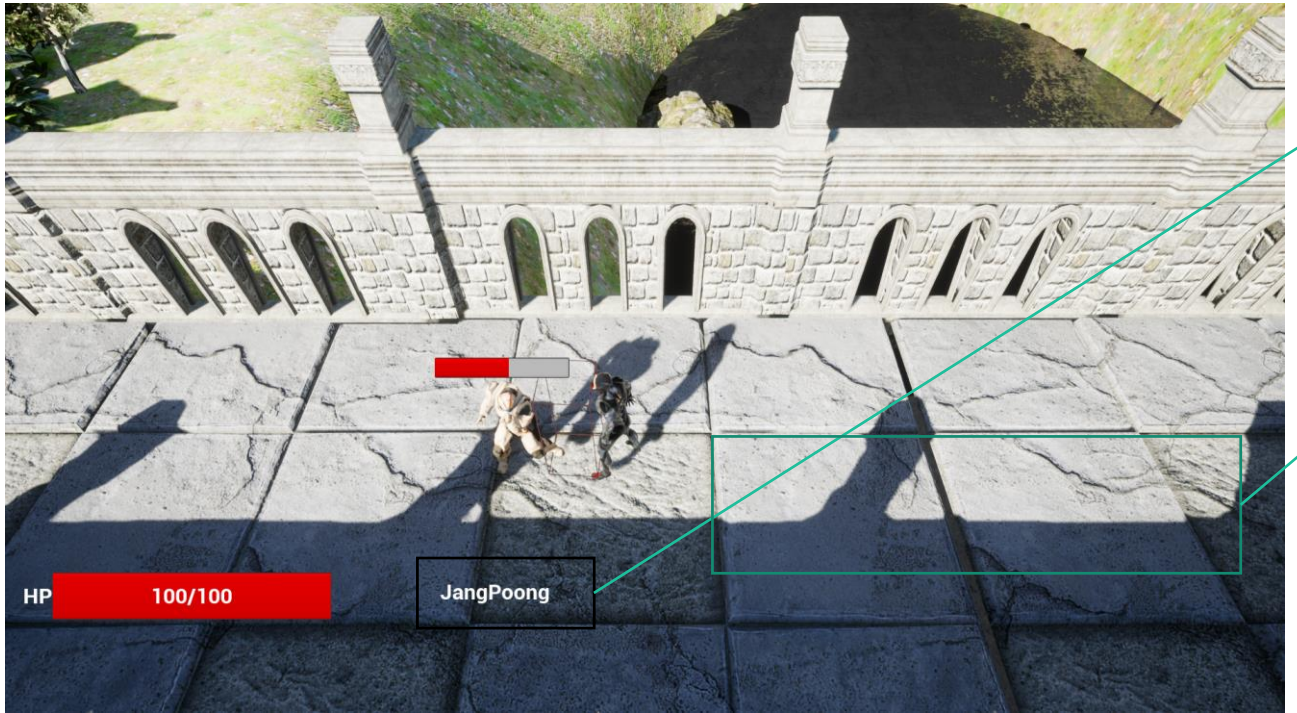
-피격시 즉각적으로 피격 애니메이션이 재생됩니다.



일반형 적 보다는 단순한 AI 이지만 블랙보드 데코레이터에 의해 흐름이 제어된다는 부분은 같습니다. 특정 HP이하로 내려가게 되면 공격력과 애니메이션이 달라지도록 만들었습니다.

4.UI/UX

게임플레이 화면 중에서 입력 키와 발동된 스킬을 알 수 있도록 화면상에 UI로 나타나게 하였고, 일반형 적에게는 체력 UI를 머리에 띄움으로 현제 체력이 얼마나 남았는지 알 수 있게 하였습니다.



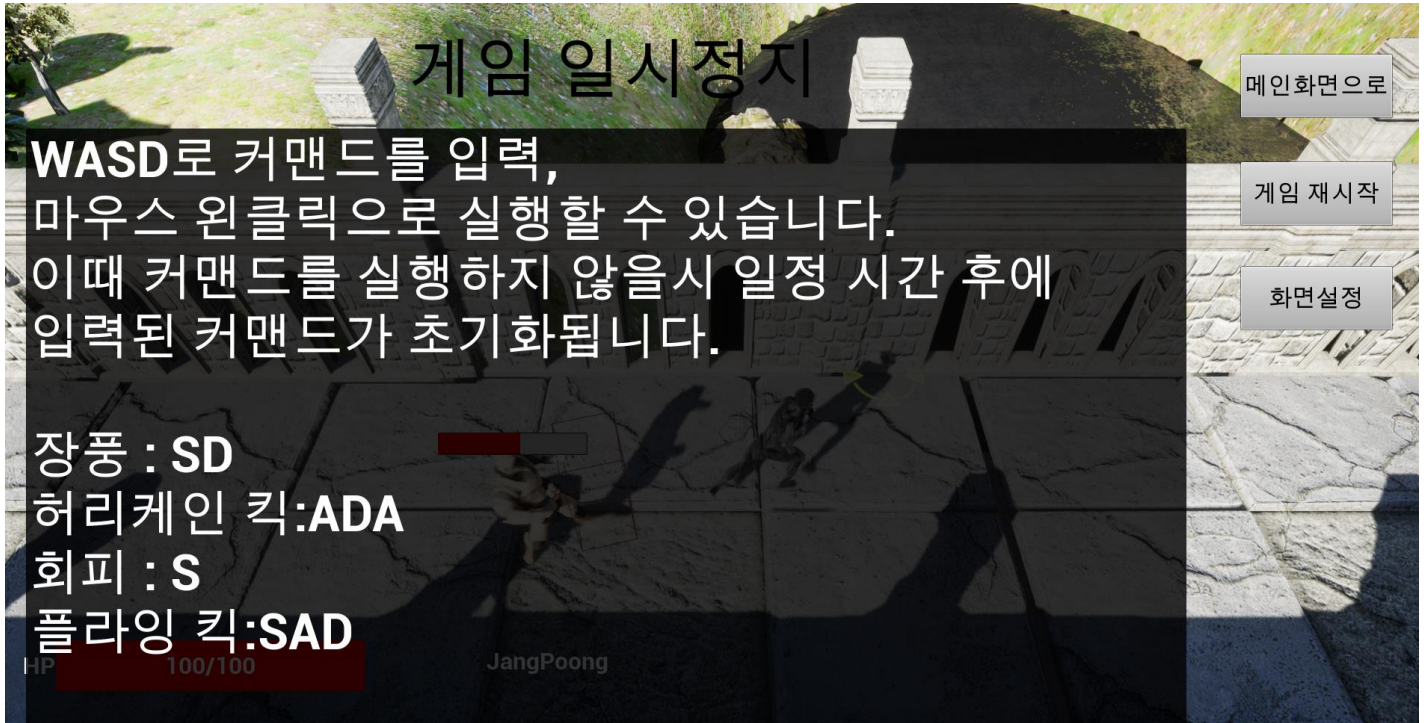
마지막으로 발동된 스킬이 표시됩니다.

입력된 키를 표시합니다.

메인 화면에서의 UI/UX 는 이렇게 구성 되어 있습니다.

간단한 화면 설정과 게임시작, 게임방법, 나가기 버튼으로 구성되어있습니다.

튜토리얼 레벨상에서의 설명 UI입니다.
튜토리얼 레벨의 특정 구간을 지날때 나타납니다.



게임 플레이중 일시정지 상황시 UI/UX 입니다.

ESC를 눌러 게임을 일시정지 할 수 있으며, 메인화면 이동 및 화면설정을 할 수 있도록 구성되어 있습니다.

다시ESC를 누르거나 게임재시작을 누르면 일시정지가 해제됩니다.

포스트프로세스 볼륨과 커스텀뎁스를 이용하여 플레이어 캐릭터가 장애물에 가려졌을때도 표시되게 하였습니다.

플레이어의 체력이 0이 되는 즉시 Game Over 표시가 뜨며 게임이 일시정지 됩니다.
이때 레벨을 다시 플레이 하거나 메인메뉴로 돌아갈 수 있도록 하였습니다.

깃허브 페이지 및 시연영상

깃허브 : https://github.com/KSH0074/Unreal_Cpp_Project_PF

시연영상 유튜브 :

https://www.youtube.com/watch?si=L2Z72pb0O2KsqyjK&v=wbS75SG8i1U&feature=youtu.be