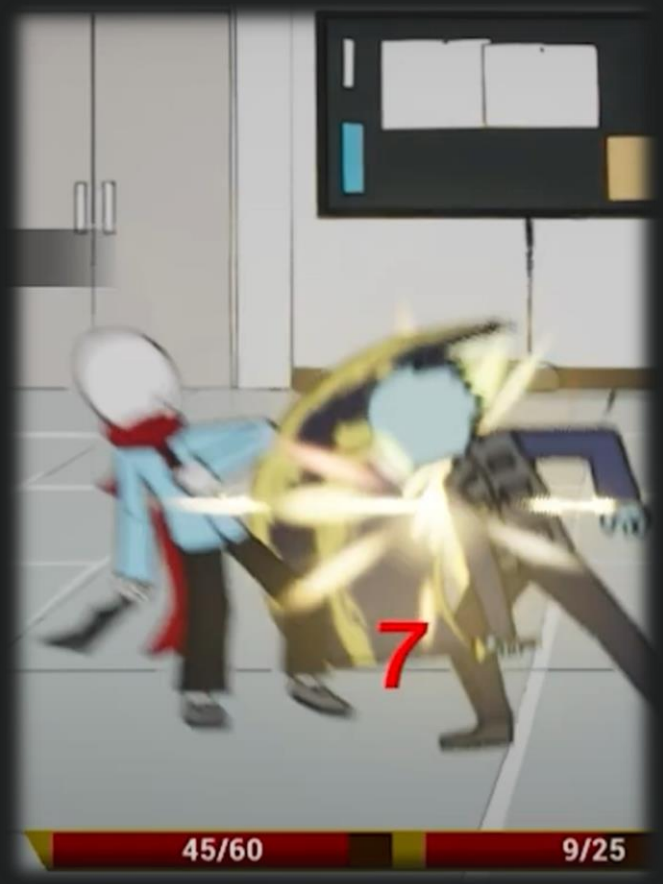
Outland (2024)

- **게임 요약** : 맵을 진행하며 적을 상대하고, NPC와 상호작용해 스토리를 진행하는 게임입니다.
- **제작 기간** : 2023.07.14 ~ 2023.09.06 (2개월 - 전투 시스템 개발),
  2024.02.01 ~ 2024.04.14 (2개월 - 버그 및 UI 수정 및 스킬 및 아이템 데이터 연동)
- **사용 툴 및 기술** : Unreal 5
- **제작 인원** : 6명 (프로그래밍 3명, 그래픽 3명, 기획 1명)
- **제작 동기** : 기존엔 유니티 엔진을 사용해 게임을 제작해왔으나 게임 회사에선 언리얼 엔진을 사용하는 사례가 많기 때문에 해당 프로젝트는 언리얼 엔진으로 진행하게 되었습니다.
- **목표** : 언리얼 엔진을 사용해 복잡한 전투 로직을 구현하고, 기획서에서 요구하는 게임 내 여러 기능들을 적절히 구현해보는 것입니다.



Unreal 5 프로젝트

[맡은 역할]
- 프로그래밍
  - 플레이어 전투, 데이터테이블(스킬, 아이템) 연동, 적 및 보스 AI, 게임 내 UI 등을 담당함.

[구현 내용]
- 전투 시스템(다수 전투, 끼어들기), 던전, 대화 씬, 전투 몹 AI, 인벤토리 등

- 스크린샷

유튜브 링크

Unreal 5 프로젝트

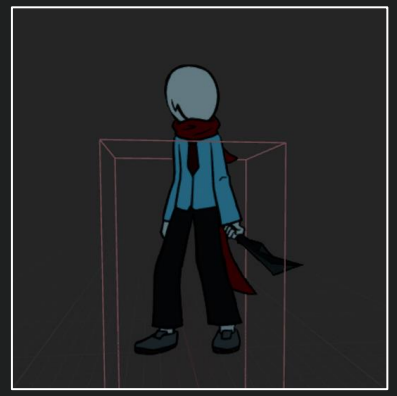
코드 링크

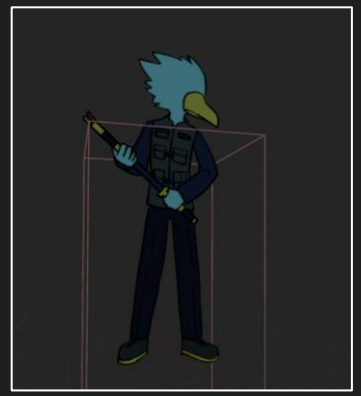
[게임 구조 : 전투 시스템] ABattleController_Base

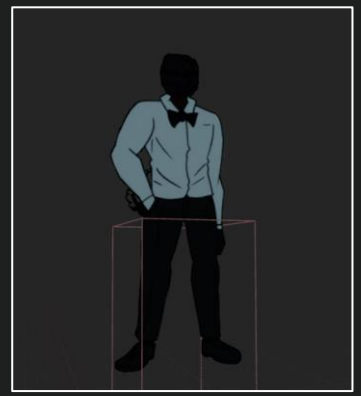
- 개체들을 전투 시작 전 정해진 위치에 생성 후 배치하는 클래스입니다.
- LoadEntity 함수에선 특정 폴더에서 미리 제작해 둔 캐릭터 블루프린트 객체를 생성하여 TArray 변수인 allyPlacementLocation, enemyPlacementLocation에서 지정해 둔 액터의 위치로 배치합니다.

- 해당 클래스는 전투 시작 전 플레이어 데이터와 적 데이터를 통해 불러올 캐릭터가 정해지면 사용됩니다.

```
UFUNCTION(BlueprintCallable)
AActor* LoadEntity(FString entityName, FVector location, FRotator rotation);

UPROPERTY(EditAnywhere, BlueprintReadWrite)
TArray<AActor*> allyPlacementLocation;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
TArray<AActor*> enemyPlacementLocation;
```







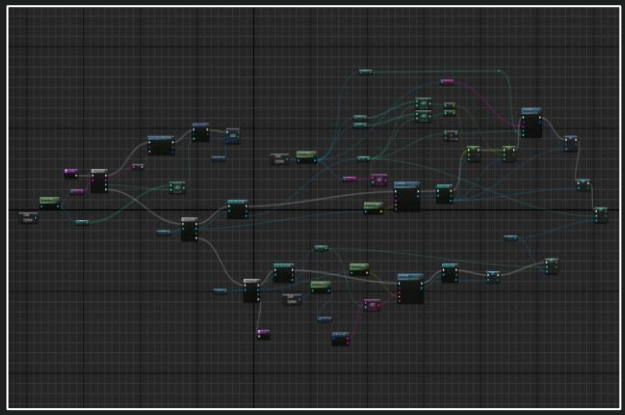
```
int ABattleController_Base::CheckTargetNumber(int targetCode)
{
    switch(targetCode)
    {
        case 0 :
        case 2 :
        case 3 :
        case 5 :
            return 1;
            break;
    }

    return 0;
}

AActor* ABattleController_Base::LoadEntity(FString entityName, FVector location, FRotator rotation)
{
    if (entityName == "")
        return nullptr;
    FActorSpawnParameters SpawnInfo;
    UClass* Entity = StaticLoadClass(AActor::StaticClass(), nullptr, *FString::Printf(TEXT("/Game/BPscripts_KHH/Data/Entity_Load/%s.%s_C"), *entityName, *entityName));
    return GetWorld()->SpawnActor(Entity, &location, &rotation, SpawnInfo);
}
```

Unreal 5 프로젝트

코드 링크

[게임 구조 : 전투 시스템] BattleControl (1)

- ABattleController_Base를 상속하는 블루프린트 클래스입니다.
- 내부에선 전투 시스템을 구현하고 있습니다.
  - LoadData 함수를 구현해 플레이어의 데이터를 불러오는 기능을 하여 플레이어의 캐릭터들의 정보와 아이템, 스킬 등을 가져와 전투를 진행함.
  - 전투는 처음 ReadyBattle 함수에서 모든 개체의 속도 값을 정해진 값에서 랜덤으로 정한 후 전투 UI를 통해 플레이어가 모든 캐릭터의 행동을 결정하면 커스텀 이벤트 Combat을 실행하면서 시작됨.



Unreal 5 프로젝트

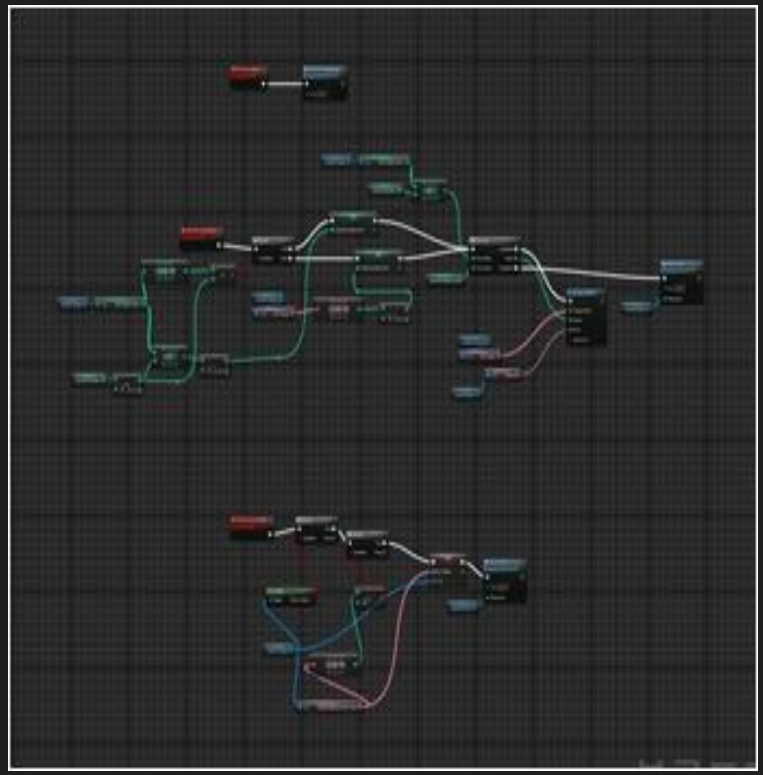
UNREAL ENGINE

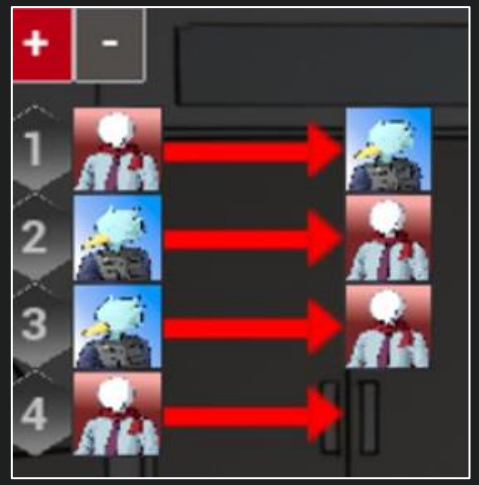
</> 코드 링크

[게임 구조 : 전투 시스템] BattleControl (2)

- 끼어들기(강제 개입) 시스템
  - 해당 게임에서는 캐릭터마다 행동의 순서가 정해져 있는데 순서가 더 앞에 있는 캐릭터의 경우 전투 중간에 다른 캐릭터 대신 끼어드는 것이 가능함.
    - 전략에 따라 적의 공격 대상을 특정 캐릭터로 바꿀 수 있음.

- interfere_Check, interfere preview On, interfere preview Off 등의 커스텀 이벤트는 이러한 끼어들기의 가능 여부와 끼어들기 UI 표시 여부 등을 조작하는 역할을 합니다.





Unreal 5 프로젝트

코드 링크

[게임 구조 : 전투 시스템] UEntityPattern_Auto

- 게임 내 존재하는 적 패턴에 따라 적의 행동을 결정해주는 클래스입니다.
- FindNextSkill 함수는 매개변수로 어떤 패턴인지 값을 받게 됩니다.
  - 매개변수
    - emPattern : 스킬 사용에 필요한 감정(마나와 비슷한 개념)에 따라 기술을 선택하는 패턴.
    - skPattern : 스킬 형태에 따라 기술을 선택하는 패턴.
    - 매개변수를 이용해 적이 다음으로 어떤 스킬을 사용할지 결정하게 됨.
- 해당 클래스는 블루프린트에서 호출이 가능하도록 했습니다.
  - UFUNCTION(BlueprintCallable) 사용
  - BattleController에서 호출됨.

```
#include "EntityPattern_Auto.h"

    /*
        < emPattern >
        (1) 평범한: 모든종류의 기술을 비슷한 빈도로 사용한다.
        (2) 섬세한: 감정소모가 적은 기술을 높은 빈도로 사용한다.
        (3) 신중한: 감정소모가 큰 기술을 높은 빈도로 사용한다.

        < skPattern >
        (1) 저돌적: 공격기술을 더 많이 사용한다.
        (2) 전략적: 보조기술을 더 많이 사용한다.
        (3) 일반적: 모든종류의 기술을 비슷한 빈도로 사용한다.
        (4) 소극적: 방어를 더 많이 사용한다.
        (5) 지정 : 정한 순서대로 반복. 이때는 emPattern 값을 추가할 값으로 정함. ex) 12  => 2번째 스킬 다음 1번째 스킬
    */
int UEntityPattern_Auto::FindNextSkill(int emPattern, int skPattern, bool noDefense, int randomAddValue)
{
    int i = 0, randomValue;
    int best = 0;


    if (tempSkillIndex.Num() != 0)
        tempSkillIndex.Reset();

    if (skPattern != 0)
    {
        if (skPattern != 5)
        {
            for (FSkill currentSkill : skills)
            {
                srand(time(NULL) + randomAddValue);
                randomValue = rand() % 100;
                switch (skPattern)
                {
                case 1:

                    if (currentSkill.value == 1)
                        tempSkillIndex.Add(i);
                    else if (randomValue < 20 || noDefense)
                        tempSkillIndex.Add(i);
                    break;
                case 2:
                    if (currentSkill.value == 2)
                        tempSkillIndex.Add(i);
                    else if(randomValue < 20 || noDefense)
                        tempSkillIndex.Add(i);
                    break;
                case 3:
                    tempSkillIndex.Add(i);
                    break;
                case 4:
                    if (randomValue > 20 && !noDefense) return -1;
                    else if (randomValue <= 20 || noDefense)
                    {
                        tempSkillIndex.Add(i);
                    }
                    break;
                }
                i++;
            }
        }
        else
        {
            if (skillOrderValue > 5) skillOrderValue = 0;

            int temp = 1;
            for (int j = 0; j < skillOrderValue; j++)
            {
                temp *= 10;
            }

            if (temp <= emPattern) i = temp;
            else
            {
                i = 1;
                skillOrderValue = 0;
            }

            emPattern %= (i * 10);
            skillOrderValue++;

            i = (emPattern / i);
            return i-1; // Index는 0부터 시작
        }
    }
```

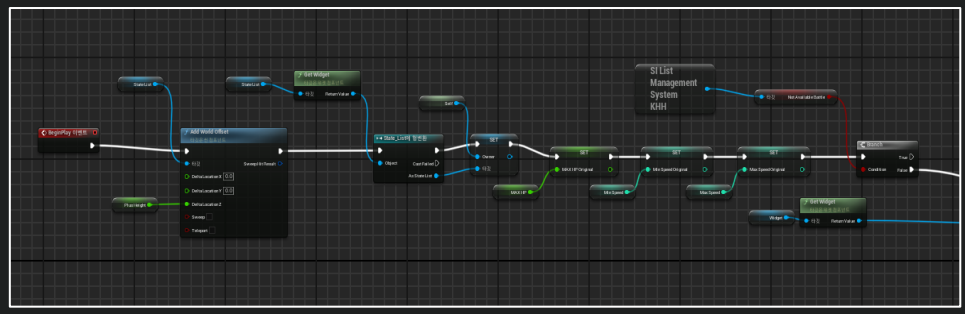
Unreal 5 프로젝트

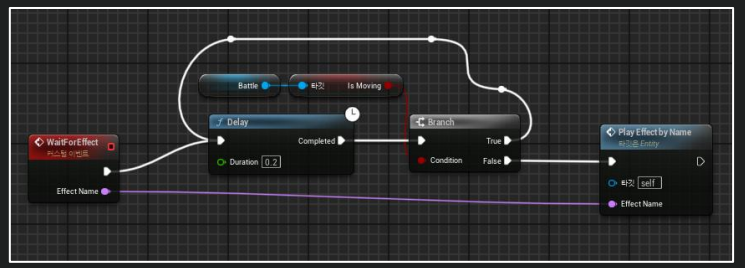
코드 링크

[게임 구조 : 전투 시스템] Entity

- 모든 캐릭터의 부모 클래스가 되는 블루프린트 클래스입니다.
  - 개체에 대한 정보(이름, HP, MP, 동료 여부, 사용 스킬 등)를 멤버 변수로 가지고 있음.
  - 스킬 이펙트 재생 대기 및 플레이어 이동에 대한 함수를 포함한 전투 중 캐릭터에게 발생하는 이벤트를 담당함.

- 스킬의 종류(공격, 방어)를 정하는데 사용하는 SetSkillTagValue 함수, 전투 중 특정 이펙트를 출력하는 PlayEffectByName 함수, 캐릭터의 이동을 담당하는 StartMove 함수 등이 있습니다.





Unreal 5 프로젝트

UNREAL ENGINE

코드 링크

[게임 구조 : 전투 시스템] UBattleSystem (1)

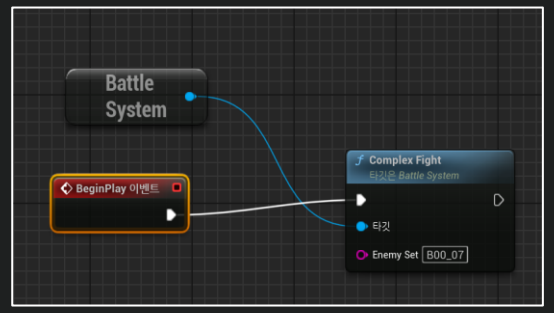
- 전투를 함수로 쉽게 호출 가능하도록 하기 위해 제작한 클래스로 UGameInstanceSubsystem을 상속하여 여러 블루프린트에서 쉽게 호출이 가능합니다.
  - 해당 클래스는 생성자 부분에서 전투 데이터 테이블(미리 설계한 전투 상황(적, 배경 맵 등))을 불러옴.

- 두 함수 SimpleFight와 ComplexFight는 전투 호출 시 필요한 데이터를 미리 데이터테이블에 저장한 경우와 그렇지 않은 경우 각각 구분해서 전투 호출을 할 수 있도록 구현하였습니다.
  - 두 함수는 블루프린트에서 호출할 수 있음.
  - 내부에서 Fight 함수를 호출함.

- 함수 LoadEntityInformation은 특정 캐릭터 객체를 가져오는 클래스입니다.
  - 캐릭터의 기본 정보를 가져오기 위해 사용함.

```cpp
UBattleSystem::UBattleSystem()
{
    static ConstructorHelpers::FObjectFinder<UDataTable> Combat(TEXT("/Game/BPscripts_KHH/Data/CombatData/Combat.Combat"));
    if(Combat.Succeeded())
    {
        BData = Combat.Object;
    }
}
```

```cpp
void UBattleSystem::SimpleFight(FString enemy, int enemyOfNumber, FString stage) // 직접 적과 적의 수를 기입해 전투 시작
{
    Fight(enemy, enemyOfNumber, "", stage);
}

void UBattleSystem::ComplexFight(FString enemySet) // 데이터테이블에 저장해둔 전투 상황을 불러와 전투 시작
{
    Fight("", 0, enemySet, TEXT(""));
}

UClass* UBattleSystem::LoadEntityInformation(FString entityName)
{
    if (entityName == "")
        return nullptr;
    UClass* Entity = StaticLoadClass(AActor::StaticClass(), nullptr, *FString::Printf(TEXT("/Game/BPscripts_KHH/Data/Entity_Load/%s.%s_C"), *entityName, *entityName));
    return Entity;
}
```



Unreal 5 프로젝트

코드 링크

[게임 구조 : 전투 시스템] UBattleSystem (2)

- Fight 함수에서는 보스와의 전투와 일반 몬스터와의 전투를 구분하여 화면 효과 출력 여부와 효과음을 재생 여부를 결정합니다.
- TimerDelegate를 이용해 일정 시간(DelaySeconds) 후에 함수 LoadFight를 호출하게 했습니다.
- 함수 PlaySound를 이용해 효과음이 레벨이 전환되어도 끊기지 않게 하였습니다.

```cpp
void UBattleSystem::Fight(FString enemy, int enemyOfNumber, FString enemySet, FString stage)
{
    FTimerDelegate TimerDelegate;
    TimerDelegate.BindUFunction(this, FName("LoadFight"), enemy, enemyOfNumber, enemySet, stage);
    FTimerHandle TimerHandle;
    FTimerManager& TimerManager = GetWorld()->GetTimerManager();
    float DelaySeconds = 0.5f;
    if (enemySet == "B00_07")
    {
        PlaySound(LoadObject<USoundWave>(nullptr, TEXT("/Game/Resource_2_KHH/Sound/SE/horn-a.horn-a")));
        DelaySeconds = 1.5f;
        UClass* bossIntro = LoadClass<UUserWidget>(nullptr, *FString::Printf(TEXT("/Game/BPscripts_KHH/Systems/Battle/UI/Boss_Appearace_%s.Boss_Appearace_%s_C"), *enemySet, *enemySet));
        CreateWidget<UUserWidget>(UGameplayStatics::GetPlayerController(GetWorld(), 0), bossIntro)->AddToViewport();
    }
    TimerManager.SetTimer(TimerHandle, TimerDelegate, DelaySeconds, false);
}

void UBattleSystem::PlaySound(USoundWave* Sound)
{
    UGameplayStatics::SpawnSound2D(this, Sound, 1.0f,1.0f, 0.0f, nullptr, true);
}
```

Unreal 5 프로젝트

코드 링크

[게임 구조 : 전투 시스템] UBattleSystem (3)

- LoadFight 함수는 전투 레벨로 이동함과 동시에 전투 상황에 맞게 캐릭터들을 배치하는 역할을 하는 함수입니다.
  - 처음엔 전투의 돌입 효과를 재생하고, 이후 TimerManager를 이용해 일정 시간 대기 후 전투 맵으로 이동하게 되어있음.
  - 전투 맵 이동 전에는 전투 데이터를 불러오고, 이동할 전투 맵을 설정하는데 이때 battleCase 라는 구조체를 이용함.
    - 해당 구조체는 전투 맵에서 전투 데이터(적)와 스테이지 정보를 저장하는 일종의 데이터 덩어리임.
  - 불러온 전투 데이터는 현재의 전투 상황으로 저장하여 전투 맵에서 이용하게 함.
    - 전투 레벨 이동 전에는 현재 레벨 또한 기록하는데 이는 전투 이후 원래의 레벨로 돌아가야 하기 때문임.
- 전투 레벨에 이동하면 BattleControl 클래스가 존재하며 해당 클래스가 전투 진행을 처리합니다.

```
void UBattleSystem::LoadFight(FString enemy, int enemyOfNumber, FString enemySet, FString stage)
{
    UClass* battleStart = LoadClass<UUserWidget>(nullptr, TEXT("/Game/BPscripts_KHH/Systems/Battle/UI/Enter_Battle.Enter_Battle_C"));
    CreateWidget<UUserWidget>(UGameplayStatics::GetPlayerController(GetWorld(), 0), battleStart)->AddToViewport();

    FTimerHandle TimerHandle;
    FTimerManager& TimerManager = GetWorld()->GetTimerManager();

    USaveDataManager* newData = NewObject<USaveDataManager>();
    USaveDataManager* save = Cast<USaveDataManager>(UGameplayStatics::LoadGameFromSlot(TEXT("PlayerSave"), 0));

    if (save != nullptr)
        newData = save;

    TSharedPtr<struct FBattleCase> battleCase = MakeShared<struct FBattleCase>();

    if (enemySet == "") // 상황 1) 같은 몬스터를 여러 개 등장시킬 때
    {
        for (int i = 0; i < enemyOfNumber; i++)
        {
            battleCase->Enemys.Add(enemy);
        }
        for (int j = 0; j < 5 - enemyOfNumber; j++)
        {
            battleCase->Enemys.Add("");
        }
    }
    else // 상황 2) 전투 상황을 미리 설계한 경우
    {
        FBattleCase* Item = BData->FindRow<FBattleCase>(FName(enemySet), FString(""));
        if (Item != nullptr)
        {
            for (FString obj : Item->Enemys)
            {
                battleCase->Enemys.Add(obj);
            }
            int randomNum = FMath::RandRange(0, Item->StageName.Num()-1);
            stage = Item->StageName[randomNum];
        }
    }

    // 불러올 전투 데이터 등록
    newData->CurrentBattleState = (*battleCase);

    // 현재 레벨 등록 <= 전투 후 복귀하기 위함.
    newData->CurrentLevel = GetWorld()->GetMapName();
    newData->CurrentLevel.RemoveFromStart(GetWorld()->StreamingLevelsPrefix);

    // 현재 상태 저장
    UGameplayStatics::SaveGameToSlot(newData, TEXT("PlayerSave"), 0);

    map = FName(stage);
    // 전투 레벨 호출
    TimerManager.SetTimer(TimerHandle, FTimerDelegate::CreateLambda([this]()
        {
            UGameplayStatics::OpenLevel(this, map);
        }), 2.0f, false);
}
```

Unreal 5 프로젝트

코드 링크

[게임 구조 : 전투 시스템] UBattleSystem (4)

- 전투 데이터 테이블은 csv파일로 관리하는데 다수의 적과 싸울 경우 이를 직접 기술하기엔 어려움이 있다 판단하여 자동적으로 배열 표기 및 정렬을 진행해주는 엑셀 수식을 만들어 이용했습니다.

  - 이를 통해 전투 데이터를 기획자가 쉽게 설계할 수 있도록 도왔음.

전투 이름	전투 설명	전투 배경 맵	적 배치(간략화)	적 배치(실제)
B00_01	스테이터스 스킬사용 튜토리얼		001_sceardEmployee	("001_sceardEmployee")
B00_02	아이템 사용 튜토리얼		002_resistantEmployee	("002_resistantEmployee")
B00_03			003_riotAgent	("003_riotAgent")
B00_04	타임라인 튜토리얼		003_riotAgent[2]	("003_riotAgent","003_riotAgent")
B00_05			003_riotAgent[2]	("003_riotAgent","003_riotAgent")
B00_06			003_riotAgent[3]	("003_riotAgent","003_riotAgent","003_riotAgent")
B00_07			003_riotAgent[2], 004_A-15	("003_riotAgent","003_riotAgent", "004_A-15")
B01_01				

```
=CONCAT(CONCAT("(",LEFT(D2,LEN(D2)-1)),")")
```

행 이름	Enemys	Stage Name
1 B00_01	("001_sceardEmployee")	("D00_01_1","D00_01_2","D00_01_3")
2 B00_02	("002_resistantEmployee")	("D00_01_1","D00_01_2","D00_01_3")
3 B00_03	("003_riotAgent")	("D00_02_1","D00_02_2","D00_02_3","D00_02_4")
4 B00_04	("003_riotAgent","003_riotAgent")	("D00_01_1","D00_01_2","D00_01_3")
5 B00_05	("003_riotAgent","003_riotAgent")	("D00_02_1","D00_02_2","D00_02_3","D00_02_4")
6 B00_06	("003_riotAgent","003_riotAgent","003_riotAgent")	("D00_02_1","D00_02_2","D00_02_3","D00_02_4")
7 B00_07	("003_riotAgent","003_riotAgent","004_A-15")	("D00_02_1","D00_02_2","D00_02_3","D00_02_4")

첫번째 적			
적 이름	반복 횟수	실제 적 배열	적 배열 간략화
003_riotAgent	1	"003_riotAgent",	003_riotAgent[1],
두번째 적			
적 이름	반복 횟수	실제 적 배열	적 배열 간략화
			[],
세번째 적			
적 이름	반복 횟수	실제 적 배열	적 배열 간략화
			[],
네번째 적			
적 이름	반복 횟수	실제 적 배열	적 배열 간략화
			[],
다섯번째 적			
적 이름	반복 횟수	실제 적 배열	적 배열 간략화
			[],

결과	해당 값을 전투 테이블에 복사해주시오.
적 배치(간략화)	적 배치(실제)
003_riotAgent[1],[],[],[],[]	("003_riotAgent")

Unreal 5 프로젝트

코드 링크

[게임 구조 : 저장 시스템] USaveSystem

- 플레이 데이터를 저장하는 클래스로 UGameInstanceSubsystem을 상속하여 여러 블루프린트에서 쉽게 호출이 가능합니다.
  - GetSaveData 함수로 플레이 데이터를 불러오고 SetSaveData 함수로 데이터를 저장.
  - USaveDataManager 클래스 객체를 이용해 하나의 매개변수로 데이터를 저장할 경우엔 EditSaveData 함수를 사용할 수 있음.
- 저장 방식
  - 언리얼에서 기본 제공하는 UGameplayStatics 클래스의 SaveGameToSlot 함수를 통해 게임 데이터를 저장하고 있음.

```
USTRUCT(BlueprintType)
파생된 블루프린트 클래스 0개
struct FBattleCase : public FTableRowBase
{
    GENERATED_USTRUCT_BODY()
public:
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    TArray<FString> Enemys;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    TArray<FString> StageName;
};
```

```
void USaveSystem::SetSaveData(TArray<FString>newAllies, TArray<int> newHpStates, TArray<int> newEmStates, FBattleCase ne
{
    USaveDataManager* playerSave = Cast<USaveDataManager>(UGameplayStatics::LoadGameFromSlot("PlayerSave", 0));
    if (playerSave == nullptr)
    {
        playerSave = NewObject<USaveDataManager>();
    }
    playerSave->Allies = newAllies;
    playerSave->HpStates = newHpStates;
    playerSave->EmStates = newEmStates;
    playerSave->CurrentBattleState = newCurrentBattleState;
    playerSave->CurrentLevel = newLevel;
    playerSave->Items = newItems;
    playerSave->NumberOfItem = itemNum;
    playerSave->CurrentLocation = newLocation;
    for (int j = 0; j < 5; j++)
    {
        playerSave->MaxHPALL.Push(-1);
    }
    UGameplayStatics::SaveGameToSlot(playerSave, "PlayerSave", 0);
}

void USaveSystem::EditSaveData(USaveDataManager* save)
{
    UGameplayStatics::SaveGameToSlot(save, "PlayerSave", 0);
}

USaveDataManager* USaveSystem::GetSaveData()
{
    USaveDataManager* save = Cast<USaveDataManager>(UGameplayStatics::LoadGameFromSlot(TEXT("PlayerSave"), 0));
    if (save != nullptr)
        return save;
    else
        return GetMutableDefault<USaveDataManager>();
}
```

```
UCLASS()
파생된 블루프린트 클래스 0개
class OUTLAND_API USaveDataManager : public USaveGame
{
    GENERATED_BODY()

public:
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    TArray<FString> Allies;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    TArray<int> HpStates;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    TArray<int> EmStates;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    FBattleCase CurrentBattleState;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    FString CurrentLevel;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    TArray<FString> Items;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    TArray<int> NumberOfItem;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    FVector CurrentLocation;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    TArray<FString> States;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    TArray<int> StateCounts;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    bool IsBattleStart;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    TArray<int> MaxHPALL;
};
```

Unity 프로젝트

- 성과
  - 데이터 테이블 관리 및 연동에 이전보다 익숙해졌습니다.
  - 복잡한 전투 로직을 블루프린트로 구현하는데 성공했으며, 다양한 기능들을 CPP 코드를 이용해 블루프린트에서 불러와 활용했습니다.
  - 블루프린트와 CPP 코드의 연동 과정에 대해 상세히 알 수 있었습니다.