



## 포트폴리오 게임 (3D 액션 RPG)

제작자	홍을석
기간	@2023년 4월 1일 → 2023년 7월 31일
대표 이미지	
제작 진행도	Done
태그	3D Action Rpg   C++   Unreal4

## 목차

-  1. 게임 소개
  -  1.1. 개발 일정표
-  2. 개발 내용
  -  2.1 개발 내용 간략하게 살펴보기
  -  2.2 핵심 시스템 및 기능 영상
    - 2.2.1. 인벤토리 시스템
    - 2.2.2. 아이템
    - 2.2.3. 스킬 시스템
    - 2.2.4. 스킬
    - 2.2.5. 빠른 사용 시스템
    - 2.2.6. AI 시스템
    - 2.2.7. 퀘스트 시스템
    - 2.2.8. UI 시스템
    - 2.2.9. 능력치 시스템
-  3. 게임 개발을 하면서 느낀 점

## 1. 게임 소개

<https://youtu.be/fYX28SCsdQY>

[게임 실행 파일 다운로드](#)



[GitHub에서 코드 보기](#)

본 게임은 Unreal Engine4를 이용해 C++ 기반으로 제작되었으며, 처음 제작하는 게임이라 게임성보단 다양한 기능을 구현하고, 다양한 상황을 경험하는 것에 초점을 두고 제작했습니다.

도움말을 통해 플레이 방법을 숙지할 수 있으며, NPC가 부여하는 퀘스트를 진행해 보스 몬스터를 잡는 것이 게임 클리어 조건입니다.

- 게임명 : 신전 RPG
- 개발 언어 및 엔진 : C++, UnrealEngine4
- 개발 인원 (참여도) : 총 1명 (100%)
- 개발 기간 : 2023.04 ~ 2023.07 (4개월)
- 지원 플랫폼 : Windows (64-bit)



## 1.1. 개발 일정표

추진계획		1월	2월	3월	4월	5월	6월	7월	8월
제작 준비	게임 기획								
	게임 리소스 구하기								
	시스템 설계								
컨텐츠 제작	레벨 제작								
	아이템 제작								
	캐릭터 스킬 제작								
	몬스터 스킬 제작								
	애니메이션 작업								
	AI 작업								
시스템 제작	위젯 작업								
	인벤토리 시스템 제작								
	스킬 시스템 제작								
	빠른 사용 시스템 제작								
버그 수정 및 기타 작업	퀘스트 시스템 제작								
	버그 수정 및 기타 작업								

단위 : month

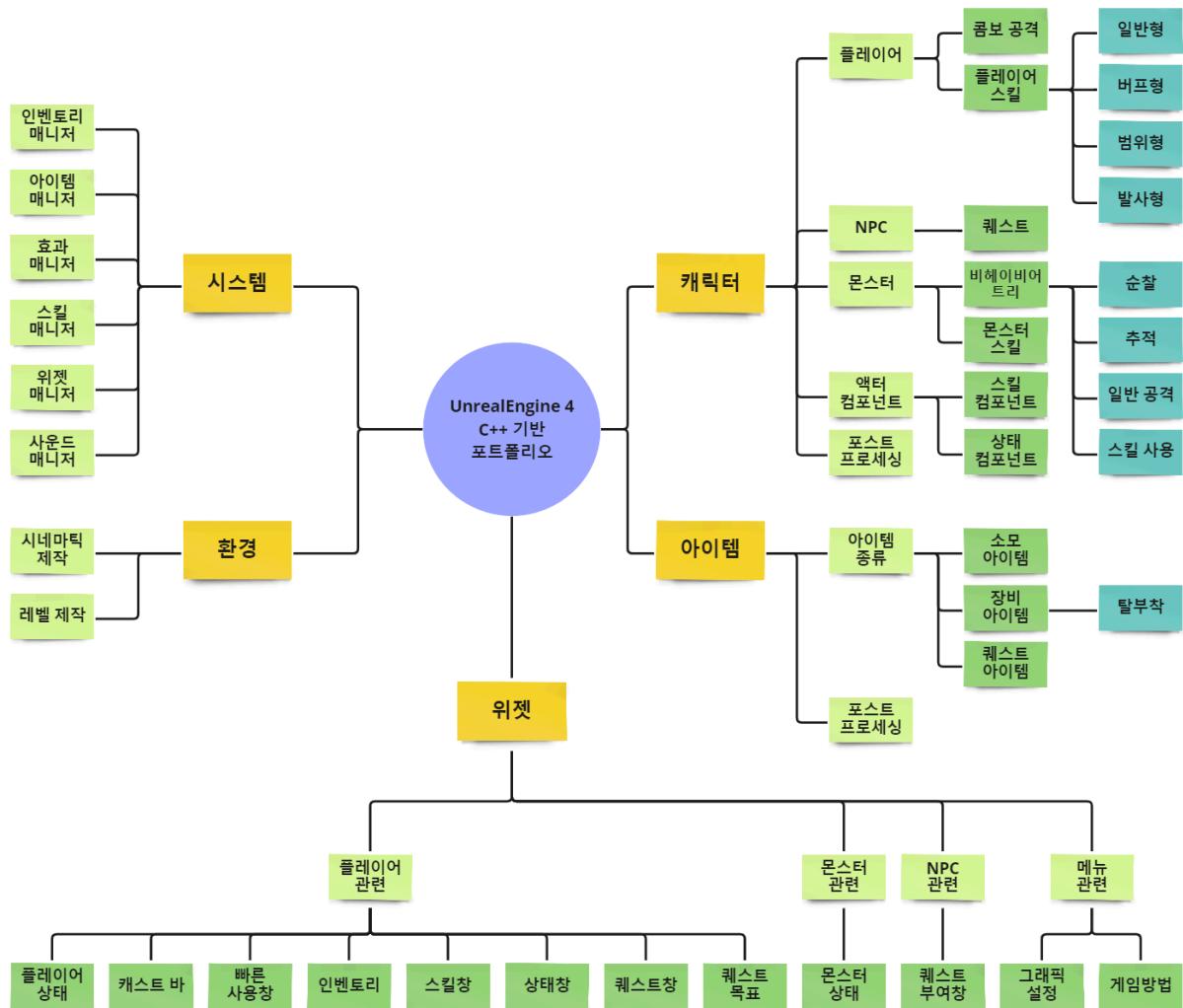
총 소요시간 : 4 개월



## 2. 개발 내용



## 2.1 개발 내용 간략하게 살펴보기



위의 그림은 제가 개발한 내용들을 간략하게 마인드 맵으로 나타낸 것으로, 크게 5가지 영역 (시스템, 환경, 캐릭터, 아이템, 위젯)으로 분류했습니다.

### 시스템

```

AGameMgr : AGameMgr()
{
    m_pInventoryMgr{}, m_pItemMgr{}, m_pWidgetMgr{}, m_pAffectMgr{}, m_pSkillMgr{}, m_pSoundMgr{}
}

m_pInventoryMgr = CreateDefaultSubobject<UInventoryManager>(TEXT("m_pInventoryMgr"));
m_pItemMgr = CreateDefaultSubobject<UItemManager>(TEXT("m_pItemMgr"));
m_pWidgetMgr = CreateDefaultSubobject<UWidgetManager>(TEXT("m_pWidgetMgr"));
m_pAffectMgr = CreateDefaultSubobject<UAffectManager>(TEXT("m_pAffectMgr"));
m_pSkillMgr = CreateDefaultSubobject<USkillManager>(TEXT("m_pSkillMgr"));
m_pSoundMgr = CreateDefaultSubobject<USoundManager>(TEXT("m_pSoundMgr"));
m_QuestLogComponent = CreateDefaultSubobject<UQuestLog>(TEXT("m_QuestLogComponent"));
}

```

BP\_GameMgr(설프)

- M P Inventory Mgr (m\_pInventoryMgr) (상속됨)
- M P Item Mgr (m\_pItemMgr) (상속됨)
- M P Widget Mgr (m\_pWidgetMgr) (상속됨)
- M P Affect Mgr (m\_pAffectMgr) (상속됨)
- M P Skill Mgr (m\_pSkillMgr) (상속됨)
- M P Sound Mgr (m\_pSoundMgr) (상속됨)
- M Quest Log Component (m\_QuestLogComponent) (상속됨)

게임의 기본적인 코드 구조는 코드 어디에서든 쉽게 호출할 수 있는 게임 스테이트(GameState)를 게임 매니저(AGameMgr)로 두고, 게임 스테이트에 인벤토리나 스킬 시스템 등을 관리하는 매니저들을 액터 컴포넌트(ActorComponent)로 부착하여 게임 스테이트를 통해 각종 매니저들에 접근할 수 있도록 설계했습니다.

### ◆ 인벤토리 매니저 (InventoryManager)

인벤토리 매니저에서는 인벤토리로의 아이템 추가와 인벤토리 내 아이템 스왑(Swap) 등의 인벤토리와 관련된 전반적인 기능들을 담당합니다.

### ◆ 아이템 매니저 (ItemManager)

아이템 매니저에서는 아이템과 관련된 기능들(사용, 버리기, 파괴하기)을 담당하며, 인벤토리 매니저의 기능을 분담하기 위해 제작했습니다.

### ◆ 효과 매니저 (AffectManager)

효과 매니저에서는 아이템이나 스킬로 인해 캐릭터의 상태를 변화시켜야 하는 경우, 그 와 관련된 효과들을 관리하는 역할을 합니다.

예를 들어, 캐릭터의 공격력이나 방어력을 변화시키거나 캐릭터의 HP나 MP를 변화시켜야 하는 경우에 사용되며, 효과(Affect)들은 상속을 통해 여러 개의 Object로 만들었고, 아이템에 기능을 조립하여 사용합니다.

### ◆ 스킬 매니저 (SkillManager)

스킬 매니저에서는 플레이어가 가진 스킬들을 스킬창에 등록하거나, 스킬의 사용 가능 여부를 확인하는 등의 플레이어 스킬과 관련된 전반적인 기능들을 담당합니다.

### ◆ 위젯 매니저 (WidgetManager)

위젯 매니저에서는 C++로 기능을 제작한 위젯을 상속한 블루프린트 위젯을 C++에서 사용할 수 있도록 가져오고, 게임 플레이에 필요한 위젯들의 동작을 제어하는 역할을 합니다.

## ◆ 사운드 매니저 (SoundManager)

사운드 매니저에서는 게임에서 사용되는 사운드들을 관리하며, 코드 작업 시 원하는 사운드를 쉽게 불러오기 위해 제작했습니다.

### 환경

## ◆ 레벨 제작 (Level Design)

레벨 제작은 하이트맵(Height Map)을 통해 지형의 기본적인 틀을 정하였고, 랜드 스케이프 머티리얼(Material)을 제작해 지형의 높낮이에 따라 다른 머티리얼 함수가 적용되도록 만들었습니다.

또한, 레벨에 배치된 물체들은 기존 에셋들을 새로 조립하거나 재배치하였고, 폴리지를 사용해 맵의 분위기를 조성했습니다.

## ◆ 시네마틱 제작 (Cinematic)

시네마틱은 게임의 최종 보스(Boss)가 등장할 시에 사용되었으며, 플레이어가 보스 소환 아이템을 지난 채 특정 지역에 도달하게 되면 시네마틱이 재생된 후에 보스가 소환(Spawn)되도록 제작했습니다.

### 캐릭터

캐릭터들에는 여러 가지의 콜리전(Collision)들을 배치해 각각 다른 충돌을 담당하도록 만들었습니다.



현재, 플레이어와 몬스터를 각각 `ACharacter` 를 상속받아 만들었는데, 게임을 제작하다 보니 플레이어와 몬스터가 비슷한 기능을 가진 부분들이 많았습니다.

`ACharacter` 를 상속받은 `ACharacter_Base` 를 만들어서 `ACharacter_Base` 를 상속받아 플레이어와 몬스터를 만들어 작업하는 것이 나을 것 같습니다.

## ◆ 플레이어 (Player)

플레이어 캐릭터에서는 피격(Hit)이나 공격(Attack)과 관련된 작업과 무기 장착이나 몽타주(Montage) 실행 중 등의 캐릭터 동작 상태 확인 및 행동과 관련된 작업을 했습니다. 또한, UI와 관련된 내용들은 전부 플레이어 컨트롤러에서 관리하도록 분리했습니다.

- **콤보 공격**

마우스 클릭을 통한 기본 공격 시, 일정 시간 안에 추가 입력을 하면 다음 동작이 이어지도록 제작했습니다.

- **플레이어 스킬**

플레이어 스킬은 일반형(Melee), 버프형(버프, 디버프), 범위형, 발사형으로 구성되어 있으며, 스킬 정보는 일괄적으로 관리하기 위해 데이터 테이블(Data Table)을 이용했습니다.

## ◆ NPC

NPC에는 쿼스트(Quest) 액터를 월드상에서 부착하여 플레이어가 쿼스트를 부여받을 수 있도록 제작했으며, 텍스트 렌더(Text Render)를 이용해 NPC의 이름을 표시했습니다.

## ◆ 몬스터 (Monster)

몬스터의 기본 정보나 피격 시스템을 작업했으며, 몬스터가 사망할 때 사망 애니메이션이 없을 경우 래그돌(Ragdoll)이 적용되도록 제작했습니다.

- **비헤이비어 트리(Behavior Tree)**

비헤이비어 트리는 크게 4가지(순찰, 추적, 일반 공격, 스킬 사용) 행동으로 나누어져 있으며, 각 행동을 직접 제작한 비헤이비어 태스크(Behavior Task)를 통해 전환 시켰습니다.

- **몬스터 스킬**

몬스터 스킬은 보스 몬스터만 지닐 수 있도록 기획했고, 3가지 형태(발사, 범위, 버프)의 스킬을 제작했습니다.

### ◆ 액터 컴포넌트 (Actor Component)

액터 컴포넌트는 스킬 컴포넌트와 상태 컴포넌트로 구성되어 있으며, 플레이어와 몬스터에 부착해 관련 변수들을 모아서 관리하기 위해 제작했습니다.

- **스킬 컴포넌트(Skills Component)**

스킬 컴포넌트를 통해 플레이어나 몬스터의 스킬을 블루프린트 상에서 등록할 수 있고, 보유한 스킬의 레벨을 관리합니다.

- **상태 컴포넌트(Status Component)**

상태 컴포넌트는 플레이어와 몬스터의 상태나 능력치와 관련된 전반적인 내용들을 관리합니다.

### ◆ 포스트 프로세싱 (Post Processing)

포스트 프로세스 머티리얼을 사용해 플레이어나 몬스터의 피격 시 메시(Mesh)의 외곽선(Outline) 색을 변경했습니다.

### 아이템

아이템에는 아이템 획득을 위한 충돌 체크 기능이 있고, 아이템이 지닐 효과는 블루프린트에서 직접 선택하여 집어넣을 수 있게 했습니다.

## ◆ 아이템 종류

- **소모 아이템**

본 게임의 소모 아이템은 HP와 MP를 회복할 수 있는 포션입니다. 아이템을 효과를 담는 그릇으로 생각했고, 아이템에 원하는 효과를 선택하여 집어넣을 수 있게 제작했습니다.

- **장비 아이템**

본 게임의 장비 아이템은 무기(Weapon) 종류만 제작했으며, 공격(Attack)과 관련된 충돌 체크와 데미지 계산 작업이 이루어집니다. 충돌 체크는 라인 트레이스(Line Trace) 대신 Overlap을 이용했습니다.

- **퀘스트 아이템**

본 게임에 존재하는 퀘스트 아이템은 보스를 소환하는 아이템뿐이며, 보스 소환 아이템을 지니고 특정 지역에 도달하면 보스가 소환되도록 제작했습니다.

## ◆ 포스트 프로세싱 (Post Processing)

포스트 프로세스 머티리얼을 사용해 플레이어 캐릭터가 현재 획득 가능한 아이템일 경우, 아이템의 스태틱 메시(Static Mesh) 외곽선 색이 변경되도록 작업했습니다.

### 위젯

모든 위젯의 기능은 C++를 통해 구현했으며, 위젯의 디자인은 C++ 위젯을 상속받아 언리얼 엔진에서 작업했습니다. 또한, '인벤토리', '스킬창', '상태창', '퀘스트창'에는 제목 막대 (Title Bar)를 부착해 위젯을 화면상에서 이동시키거나 종료시킬 수 있습니다.

## ◆ 플레이어 관련

- **플레이어 상태**

플레이어의 HP, MP, 경험치(Exp)가 표시되며, 플레이어가 버프 스킬을 사용할 시에 버프 스킬의 아이콘과 버프의 지속 시간을 나타내는 진행 바(Progress Bar)가

표시되도록 제작했습니다.

- **캐스트 바 (Cast Bar)**

캐스트 바는 캐스팅 시간을 필요로 하는 스킬을 사용할 시에 표시되며, 캐스팅 시간을 가지는 스킬은 캐스트 바가 전부 진행되어야 사용이 되며, 중간에 모션을 방해받으면 캐스팅이 취소되어 플레이어의 마나가 소모되지 않습니다.

- **빠른 사용창 (QuickSlot Window)**

빠른 사용창에 아이템이나 플레이어 스킬을 등록하면 숫자 버튼을 눌러 아이템이나 플레이어 스킬을 사용할 수 있습니다.

빠른 사용창의 슬롯에는 아이템이나 플레이어 스킬을 구분 없이 등록할 수 있으며, 플레이어 스킬의 경우 스킬 사용 시 진행 바를 통해 스킬의 쿨타임이 표시됩니다.

- **인벤토리 (Inventory)**

인벤토리에는 현재 플레이어가 지니고 있는 아이템들이 표시됩니다. 아이템 1개가 인벤토리 1칸을 차지하며, 아이템 드래그를 통해 인벤토리 내 위치를 자유롭게 변경할 수 있습니다.

또한, 드래그한 아이템을 인벤토리 밖에서 놔둘 경우 아이템이 버려지며, 마우스 우클릭을 통해 아이템 팝업창을 띄워 아이템의 '사용', '버리기', '파괴하기' 기능을 선택할 수 있습니다.

그리고, 인벤토리 내 아이템에 마우스를 갖다 대면 아이템의 정보를 확인할 수 있는 툴팁(ToolTip)이 나타납니다.

- **스킬창 (Skill Window)**

스킬창에는 현재 플레이어가 지니고 있는 스킬이 표시됩니다.

스킬 레벨을 올릴 수 없는 스킬의 경우 회색으로 표시되며, 스킬 레벨을 올릴 수 있는 스킬의 경우 노란색 틴트(Tint)가 적용되어 표시됩니다.

또한, 현재 보유한 스킬 포인트와 스킬 정보를 확인할 수 있는 툴팁이 표시됩니다.

- **상태창 (Status Window)**

상태창에는 플레이어의 HP, MP가 표시되고, 플레이어의 현재 공격력과 방어력이 표시됩니다. 또한, 상태창에서는 플레이어의 능력치를 확인할 수 있으며, 레벨업을 통해 능력치를 올릴 수 있습니다.

- **퀘스트창 (Quest Window)**

퀘스트창은 플레이어가 수락한 퀘스트 정보를 확인할 수 있는 창입니다. 퀘스트 창에는 퀘스트 이름, 퀘스트 설명, 퀘스트 목표가 표시되도록 만들었습니다.

- **퀘스트 목표**

퀘스트 목표는 퀘스트 창에 존재하는 퀘스트 목록 중에서 퀘스트를 한 개 선택하면 그에 해당하는 퀘스트 목표가 플레이어의 화면에 노출되도록 제작했습니다.

## ◆ 몬스터 관련

- **몬스터 상태**

몬스터의 이름과 HP가 표시되고, 플레이어가 몬스터에게 디버프 종류의 스킬을 사용하면 디버프 스킬의 아이콘과 디버프의 지속 시간을 나타내는 진행 바가 표시됩니다.

## ◆ NPC 관련

- **퀘스트 부여창**

NPC가 플레이어에게 퀘스트를 부여할 때 나타나는 창입니다. 퀘스트 부여창에는 퀘스트 이름, 퀘스트 설명, 퀘스트 목표, 퀘스트 보상 정보가 담겨있습니다.

## ◆ 메뉴 관련

- **그래픽 설정**

플레이어의 게임 플레이 환경에 맞게 게임의 그래픽을 설정할 수 있으며, '윈도우 모드', '해상도', '안티에일리어싱', '그림자 퀄리티', '텍스쳐 퀄리티'를 조절하는 기능을 제공합니다.

- **게임 방법 (Game Method)**

게임 플레이를 위한 조작법을 배울 수 있으며, 게임 조작 방법을 설명하는 이미지들로 구성되어 있습니다.

## ✓ 2.2 핵심 시스템 및 기능 영상

### 2.2.1. 인벤토리 시스템

<https://youtu.be/uHLpKJGK0WU>

```
bool UInventoryManager::F_ItemAdd(FName* ItemRowName)
{
    bool bSuccess;
    FItemData* newItemData = m_pItemDataTable->FindRow<FItemData>(*ItemRowName, "");
    for (uint8 Index = 0; Index < (*F_GetInventorySlotArray()).Num(); Index++)
    {
        if ((*F_GetInventorySlotArray())[Index] -> F_GetItemData() == m_pItemDataDefault)
        {
            (*F_GetInventorySlotArray())[Index] -> F_SetItemData(newItemData);
            (*F_GetInventorySlotArray())[Index] -> F_UpdateInventorySlotIcon();
            m_pGameMgr->F_GetSoundMgr()->F_PlaySound(EPlaySound::E_SoundItemPickUp);

            bSuccess = true;
            return bSuccess;
        }
    }
    if (!bSuccess)
    {
        FText AlertText = FText::FromString(FString::Printf(TEXT("인벤토리가 가득 차 있습니다")));
        m_pGameMgr->F_GetWidgetMgr()->F_GetHUD()->F_DisplayTextAlert(&AlertText);
    }
    return bSuccess;
}
```

인벤토리 시스템은 **UInventoryManager**에서 인벤토리 위젯(Widget)의 인벤토리 슬롯 배열에 인덱스로 접근하여 인벤토리로의 아이템 추가, 인벤토리 내 아이템 스왑(Swap)을 할 수 있도록 설계했고, **UInventoryManager**가 아이템 데이터 테이블을 들고 있어서 아이템에게서 아이템 이름만 넘겨받아 아이템 데이터를 찾도록 설계했습니다.

```
void UAffectManager::F_Init()
{
    m_arAffect.Reserve((uint8)EAffectType::E_Affect_Max);
    m_arAffect.EmplaceAt((uint8)EAffectType::E_Default, NewObject<UAffect_Default>(this, TEXT("Default")));
    m_arAffect.EmplaceAt((uint8)EAffectType::E_HealthPointChange, NewObject<UAffect_HealthPointChange>(this, TEXT("HealthPointChange")));
    m_arAffect.EmplaceAt((uint8)EAffectType::E_ManaPointChange, NewObject<UAffect_ManaPointChange>(this, TEXT("ManaPointChange")));
    m_arAffect.EmplaceAt((uint8)EAffectType::E_AttackStrikingPowerChange, NewObject<UAffect_AttackStrikingPowerChange>(this, TEXT("AttackStrikingPowerChange")));
    m_arAffect.EmplaceAt((uint8)EAffectType::E_BuffAppliedStrikingPowerChange, NewObject<UAffect_BuffAppliedStrikingPowerChange>(this, TEXT("BuffAppliedStrikingPowerChange")));
    m_arAffect.EmplaceAt((uint8)EAffectType::E_BuffAppliedDefensivePowerChange, NewObject<UAffect_BuffAppliedDefensivePowerChange>(this, TEXT("BuffAppliedDefensivePowerChange")));
}
```

```

bool UItemManager::ApplyAffect(uint8 Index)
{
    EAffectType ItemAffect = (*F_GetInventorySlotArray())[Index]->F_GetItemData()->m_eAffectType;
    float ItemValueApplied = (*F_GetInventorySlotArray())[Index]->F_GetItemData()->m_fAffectValue;
    m_Interface_Affect = Cast<IInterface_Affect>(m_Player->F_GetGameMgr()->F_GetAffectMgr()->F_GetAffect(ItemAffect));
    bool bSuccess = m_Interface_Affect->F_Apply(m_Player, ItemValueApplied, EOperatorType::E_Plus);
    return bSuccess;
}

```

`UItemManager` 예선 인벤토리의 아이템을 사용하고, 버리고, 파괴하고, 스폰(Spawn)할 수 있는 기능을 담당하도록 설계했으며, 아이템의 효과는 아이템 데이터에 `enum class` 변수로 해당 효과를 들고 있게 했습니다.

`UAffectManager`에서 `UAffect_Base`를 상속한 오브젝트(Object)들은 `TArray` 컨테이너를 사용하여 관리하였고, `enum class` 변수로 컨테이너에 접근하여 해당하는 Affect를 불러올 수 있도록 설계했습니다. 또한, 불러온 Affect는 `IInterface_Affect`를 통해 사용했습니다.



처음 인벤토리 시스템을 제작할 때 `TArray` 컨테이너를 사용하는 변수에 `UPROPERTY` 매크로를 선언해주지 않아 가비지 컬렉션(Garbage Collection, GC)을 방지하지 못해 많은 시행착오를 겪었습니다. 하지만 시행착오를 겪은 덕분에 포인터와 엔진의 기능에 대해서 좀 더 깊게 생각해 볼 수 있었습니다.



해당 인벤토리는 아이템 1개당 슬롯 1개를 차지하도록 설계했는데, 아이템을 중첩시키고 싶으면 `TMap` 자료 구조를 추가하여 습득한 아이템을 검색해 중복 아이템 일 경우를 체크해서 중첩시킬 것 같음.

## 2.2.2. 아이템

### 【보스 소환 아이템】

[https://youtu.be/cKR\\_3ZLrV2U](https://youtu.be/cKR_3ZLrV2U)

아이템은 3가지 타입(소모, 장비, 퀘스트)으로 구성되어 있으며, 소모 아이템은 `Altem`을 상속받아 블루프린트 액터를 만들고, 그 블루프린트 액터를 상속받아 아이템을 확장했습니다. 장비 아이템의 경우 `Altem`을 상속받아 `AWeapon_Base`를 만들어 블루프린트를 통해 확장했습니다.

The screenshot shows a software interface titled 'DT\_ItemData'. The top menu bar includes '파일' (File), '편집' (Edit), '에셋' (Assets), '창' (Window), and '도움말' (Help). Below the menu is a toolbar with icons for '저장' (Save), '닫기' (Close), '리임포트' (Reimport), '추가' (Add), '복사' (Copy), '붙여넣기' (Paste), '복제' (Duplicate), and '제거' (Delete). A search bar at the top right contains the text '데이터 테이블 디테일'. The main area is a table with the following columns: 행 이름 (Row Name), M Item Type, M Item Class, M Item Name, M E Affect Type, M F Affect Value, and M Item Description. The table contains five rows of data:

행 이름	M Item Type	M Item Class	M Item Name	M E Affect Type	M F Affect Value	M Item Description
1	Default	Default	BlueprintGeneratedClass'/Game/Item/BP_Item_Default.BP_Item_Default_C'	Default	0.000000	기본값
2	HealthPotion	Consumption	BlueprintGeneratedClass'/Game/Item/BP_HealthPotion.BP_HealthPotion_C'	HP 회복 포션	50.000000	HP를 소량 회복한다.
3	ManaPotion	Consumption	BlueprintGeneratedClass'/Game/Item/BP_ManaPotion.BP_ManaPotion_C'	MP 회복 포션	50.000000	마나를 소량 회복한다.
4	Katana	Equipment	BlueprintGeneratedClass'/Game/Weapon/BP_Weapon_Katana.BP_W_Katana_C'	기본 카타나	Default	0.000000
5	BossKey	Quest	BlueprintGeneratedClass'/Game/Item/BP_BossKey.BP_BossKey_C'	템페이지의 열쇠	Default	0.000000

또한, 모든 아이템의 정보를 일괄적으로 관리하기 위해 아이템 데이터 테이블을 이용했습니다.



현재 장비 아이템은 무기 종류만 있기 때문에, `AWeapon_Base`로 만들어 사용했는데, 다음번에는 다양한 장비가 있을 것을 고려하여 `AEquipmentItem_Base`로 만들어 장비 타입에 따라 기능을 확장할 수 있도록 만들어야겠다고 생각했습니다.

### 2.2.3. 스킬 시스템

<https://youtu.be/BzLS4q-xtbg>

```

void USkillManager::F_PlayerSkillsRegistration(uint8 Index)
{
    TArray<TSubclassOf<ASkill_Base>>* arPlayerSkill = m_Player->F_GetPlayerSkillsComponent()->F_GetPlayerSkillsList();
    if ((*arPlayerSkill).IsValidIndex(Index))
    {
        TSubclassOf<ASkill_Base> Skill = (*arPlayerSkill)[Index];
        FSkillData* PlayerSkillData = m_oSkillDataTable->FindRow<FSkillData>(*Skill.GetDefaultObject()->F_GetSkillRowName(), "");
        if (PlayerSkillData->m_SkillWeaponType == ESkillWeaponType::E_Katana)
        {
            (*F_GetSkillSlotArray())[m_nKatanaSkill[SlotIndex]]->F_SetSkillData(PlayerSkillData);
            (*F_GetSkillSlotArray())[m_nKatanaSkill[SlotIndex]]->F_SetSkillSlotIcon();
            m_nKatanaSkill[SlotIndex]++;
        }
        else if (PlayerSkillData->m_SkillWeaponType == ESkillWeaponType::E_Common)
        {
            (*F_GetSkillSlotArray())[m_nCommonSkill[SlotIndex]]->F_SetSkillData(PlayerSkillData);
            (*F_GetSkillSlotArray())[m_nCommonSkill[SlotIndex]]->F_SetSkillSlotIcon();
            m_nCommonSkill[SlotIndex]++;
        }
    }
}

```

```

UCLASS( ClassGroup=(Custom), meta=(BlueprintSpawnableComponent) )
class TESTCPP1_API UPlayerSkillsComponent : public UActorComponent
{
    GENERATED_BODY()

public:
    UPlayerSkillsComponent();

protected:
    virtual void BeginPlay() override;

protected:
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TArray<TSubclassOf<class ASkill_Base>> m_arPlayerSkillList;
    UPROPERTY()
    TMap< FName, uint8> m_MapSkillUnlocks;

public:
    bool F_AddSkill(FName SkillName);
    bool F_QuerySkill(FName SkillName, uint8& nSkillLevel);
    TArray<TSubclassOf<ASkill_Base>>* F_GetPlayerSkillsList();
};

```

**USkillManager**에선 플레이어 스킬을 스킬창에 등록하고, 플레이어 스킬 사용과 관련된 기능을 담당합니다. **UPlayerSkillsComponent**를 플레이어 캐릭터에 부착해 플레이어가 보유할 스킬을 추가할 수 있도록 만들었고, **TMap** 컨테이너를 이용해 플레이어가 사용하고자 하는 스킬의 사용 가능 유무와 스킬 레벨을 체크할 수 있도록 설계했습니다. 또한, 스킬 트리 기능을 집어넣어 추후 더 다양한 종류의 스킬이 추가될 상황을 대비했습니다.



스킬창에는 사용 무기 형태에 따라 스킬이 분리되어 있습니다. 그래서 각 사용 무기 형태에 따른 스킬들을 관리하기 위한 컨테이너를 사용하는 것 외에도 모든 스킬들을 효과적으로 관리하기 위한 컨테이너를 별도로 사용했습니다.



현재는 카타나(Katana) 스킬과 공용(Common) 스킬뿐인데, 추후 더 많은 무기 타입에 따른 스킬을 추가해야 할 필요성을 느낌.

## 2.2.4. 스킬

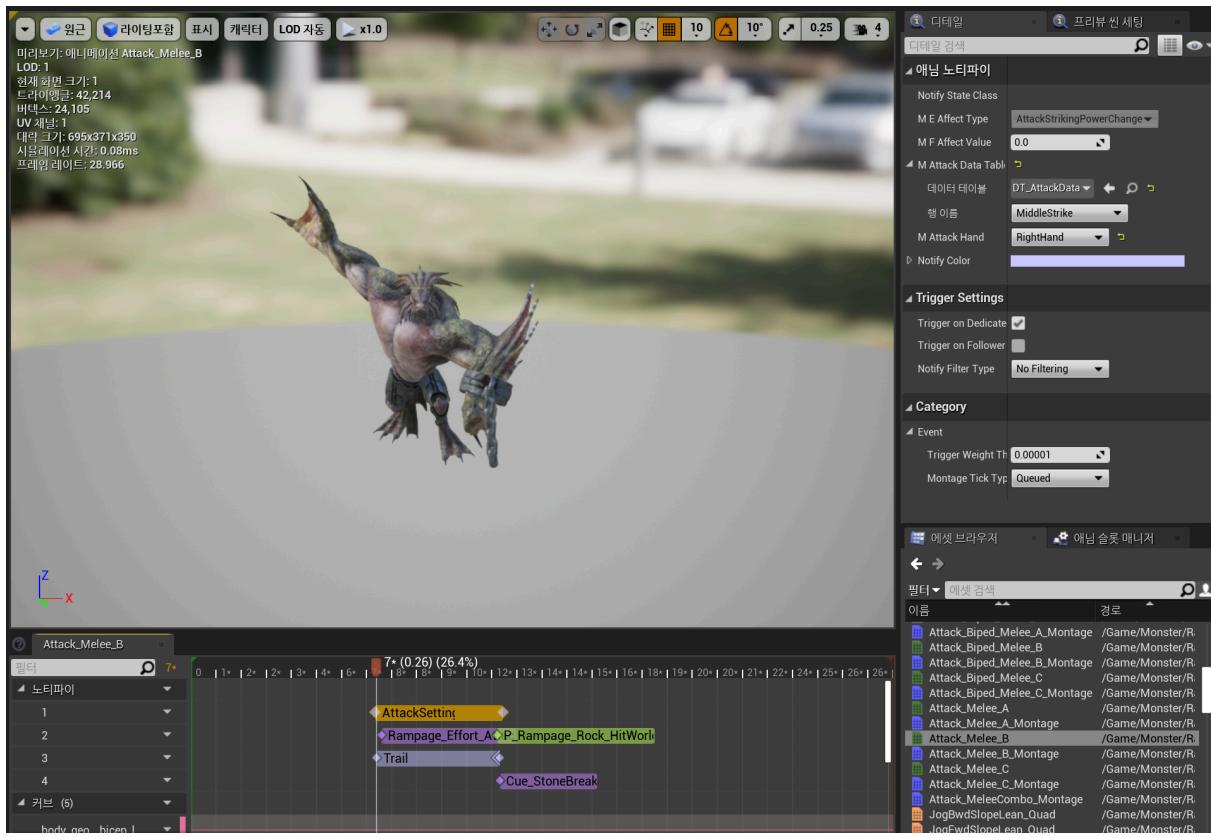
### [플레이어 스킬]

<https://youtu.be/lEP8zWpjvbE>

### [몬스터 스킬]

<https://youtu.be/0HfOgcvFIIA>

플레이어 스킬은 일반형(Melee), 버프형(버프, 디버프), 범위형, 발사형으로 구성되어 있으며, `ASkill_Base` 를 상속받아 일반형을 만들었고, 일반형을 상속받아 버프형, 범위형, 발사형으로 확장했습니다. 그런 다음, 각 타입의 스킬을 상속받아 블루프린트 액터를 만들었고, 다시 그 블루프린트 액터를 상속받아 스킬을 확장했습니다.



그리고, 일반 공격이나 일반형 스킬은 애니메이션에서 데이터 테이블을 이용해 각각의 동작마다 공격력과 공격 타입을 변경할 수 있게 만들었으며, 스킬 사용 시스템에는 캐스팅과 타겟팅의 개념을 추가하여 다양한 스킬 사용 형태를 만들었습니다.

행 이름	M Skill Weapon Type	M Skill Class	M Skill Name	M Skill Description	M Skill Image
1	Default	Class'/Script/TestCpp1.Skill_Base'	Default	한 줄 기미의 검기를 버스듬히 날려 경로상의 적들에게 피해를 입힌다.	Texture2D'/Game/Widgets/icon/Icon_SlotBack.Image_SlotBack'
2	Katana	BlueprintGeneratedClass'/Game/Skill/Projectile/Skill_Katana1.Skill_Katana'	검격	적을 연달아 4번 공격한다.	Texture2D'/Game/Skill/icon/icon_Katana1.Icon_Katana1'
3	Katana	BlueprintGeneratedClass'/Game/Skill/Melee/Skill_Katana2.Skill_Katana2'	연속베기	카타나를 휙껏 휘둘러 전방의 적들에게 피해를 입힌다.	Texture2D'/Game/Skill/icon/icon_Katana2.Icon_Katana2'
4	Katana	BlueprintGeneratedClass'/Game/Skill/AOE/Skill_Katana3.Skill_Katana3_C'	힘껏 치른기	검기를 가로로 날려 다수의 적들에게 피해를 입힌다.	Texture2D'/Game/Skill/icon/icon_Katana3.Icon_Katana3'
5	Katana	BlueprintGeneratedClass'/Game/Skill/Projectile/Skill_Katana4.Skill_Katana4'	검기 날리기	검기를 가로로 날려 다수의 적들에게 피해를 입힌다.	Texture2D'/Game/Skill/icon/icon_Katana4.Icon_Katana4'
6	Katana	BlueprintGeneratedClass'/Game/Skill/Melee/Skill_Katana5.Skill_Katana5'	도검난무	짧은 시간 안에 여러번 공격하여 다수의 피해를 입힌다.	Texture2D'/Game/Skill/icon/icon_Katana5.Icon_Katana5'
7	Common	BlueprintGeneratedClass'/Game/Skill/Buff/Skill_Common1.Skill_Common'	강화	본인의 공격력을 증가시킨다.	Texture2D'/Game/Skill/icon/icon_BuffSkill.Icon_BuffSkill'
8	Common	BlueprintGeneratedClass'/Game/Skill/Buff/Skill_Common2.Skill_Common'	약화	지정한 대상의 방어력을 감소시킨다.	Texture2D'/Game/Skill/icon/icon_DebuffSkill.Icon_DebuffSkill'

또한, 스킬 정보는 일괄적으로 관리하기 위해 데이터 테이블(Data Table)을 이용했습니다.



플레이어 스킬을 먼저 제작하면서 캐스트 바와 엣어 작업했는데, 몬스터 스킬을 만들려고 하니 캐스트 바가 엣여있어 몬스터 스킬은 따로 제작하게 되었습니다. 다음부턴 기능을 더 잘게 나누어 컨텐츠 확장을 용이하게 만들어야겠다고 생각했습니다.

## 2.2.5. 빠른 사용 시스템

[https://youtu.be/Kr\\_rqfmZluE](https://youtu.be/Kr_rqfmZluE)

```
bool UQuickSlot::F_QuickSlotRegistration(ESlotType PairSlotType, uint8 PairSlotIndex)
{
    F_ResetQuickSlot();
    m_sQuickSlotData.m_ePairSlotType = PairSlotType;
    m_sQuickSlotData.m_nPairSlotIndex = PairSlotIndex;
    if (m_sQuickSlotData.m_ePairSlotType == ESlotType::E_InventorySlot)
    {
        ItemData* pItemData = (*m_pGameMgr->F_GetInventoryMgr()->F_GetInventorySlotArray())[m_sQuickSlotData.m_nPairSlotIndex]->F_GetItemData();
        if (pItemData->m_eItemType == EItemType::E_Quest)
        {
            F_ResetQuickSlot();
            F_UpdateQuickSlotIcon();
            FText AlertText = FText::FromString(FString::Printf(TEXT("퀘스트 관련 아이템은 등록할 수 없습니다")));
            m_pGameMgr->F_GetWidgetMgr()->F_GetHUD()->F_DisplayTextAlert(&AlertText);
            m_pGameMgr->F_GetSoundMgr()->F_PlaySound(EPaySound::E_SoundDoNotUse);
            return false;
        }
        m_sQuickSlotData.m_pQuickSlotItemData = pItemData;
        m_pGameMgr->F_GetInventoryMgr()->F_SetQuickSlotMap(InventoryIndex(m_sQuickSlotData.m_nPairSlotIndex, m_eQuickSlotNumber));
    }
    else if (m_sQuickSlotData.m_ePairSlotType == ESlotType::E_SkillSlot)
    {
        FSkillData* pSkillData = (*m_pGameMgr->F_GetSkillMgr()->F_GetSkillSlotArray())[m_sQuickSlotData.m_nPairSlotIndex]->F_GetSkillData();
        m_sQuickSlotData.m_pQuickSlotSkillData = pSkillData;
        m_pGameMgr->F_GetSkillMgr()->F_SetQuickSlotMap(SkillIndex(m_sQuickSlotData.m_nPairSlotIndex, m_eQuickSlotNumber));
        F_UpdateQuickSlotCooldownTime();
    }
    m_sQuickSlotData.m_bRegistration = true;
    F_UpdateQuickSlotIcon();
    return true;
}
```

```

void UQuickSlot::F_QuickUse()
{
    if (_sQuickSlotData.m_bRegistration)
    {
        if (_sQuickSlotData.m_ePairSlotType == ESlotType::E_InventorySlot)
        {
            m_pGameMgr->F_ItemUse(_sQuickSlotData.m_nPairSlotIndex);
        }
        else if (_sQuickSlotData.m_ePairSlotType == ESlotType::E_SkillSlot)
        {
            if (!_sQuickSlotData.m_bCooldown && !m_Player->F_GetPlayingMontage() && m_Player->F_CheckMana(m_cost, _sQuickSlotData.m_pQuickSlotSkillData->m_Cost))
            {
                m_pGameMgr->F_SkillMgr()->F_UseSkill(_sQuickSlotData.m_nPairSlotIndex);
            }
            else
            {
                FText AlertText = FText::FromString(FString::Printf(TEXT("스킬을 사용할 수 없습니다")));
                m_pGameMgr->F_WidgetMgr()->F_DisplayTextAlert(&AlertText);
                m_pGameMgr->F_SoundMgr()->F_PlaySound(EPaySound::E_SoundDoNotUse);
            }
        }
    }
}

```

빠른 사용 시스템은 `UQuickSlot` 위젯 자체에서 처리할 수 있도록 설계했습니다.

`USlot_Base` 위젯을 상속받아 `UInventorySlot`, `USkillSlot`, `UQuickSlot`을 만들었고, 빠른 사용 등록 시에 `UQuickSlot` 내의 구조체에 `ESlotType`과 `Index` 정보를 저장할 수 있도록 만들었습니다. 또한, `UInventoryManager` 와 `USkillManager`에 `TMap` 컨테이너를 만들어 각각의 해당 `Index`에 `QuickSlot` 번호를 등록해 아이템이나 스킬 사용이 유기적으로 이루어지도록 설계했습니다.



처음에는 `UQuickSlot` 에다가 스킬 쿨타임 타이머를 갖도록 제작했는데, 빠른 사용 등록을 취소하거나 등록 위치를 바꾸면 문제가 발생했습니다.

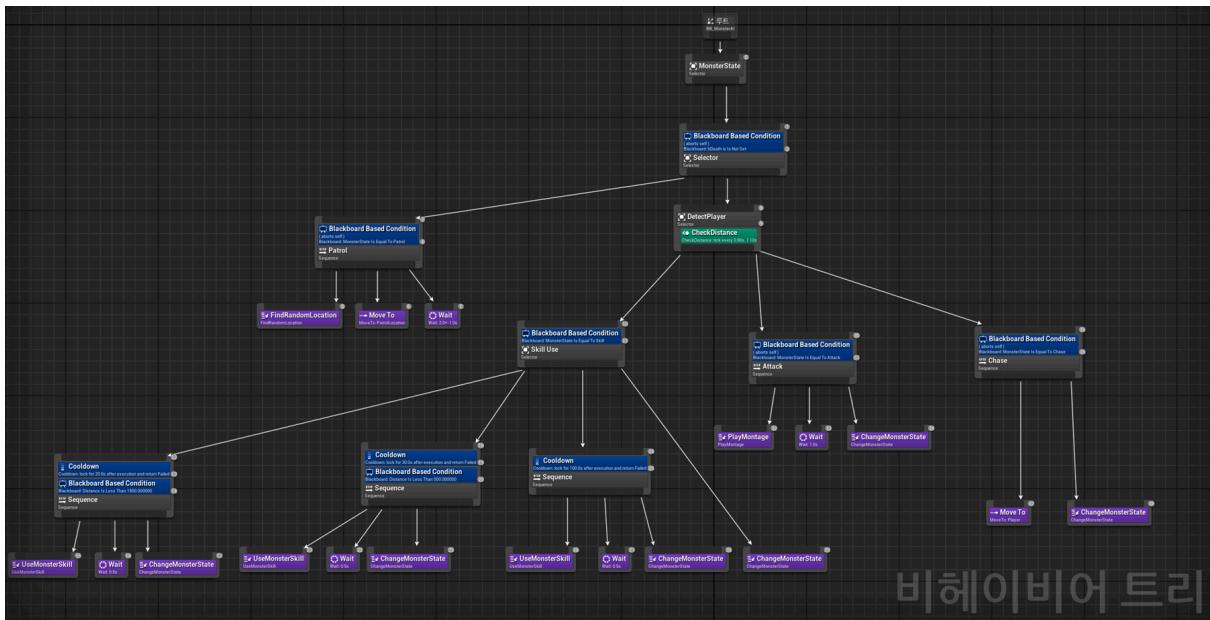
그래서 스킬 쿨타임 타이머는 스킬 창의 `USkillSlot` 이 갖도록 만들었고, `UQuickSlot`은 `USkillSlot`의 타이머 핸들의 포인터를 갖도록 수정했습니다.



지금은 같은 `UQuickSlot`에 스킬과 아이템을 모두 등록할 수 있도록 되어 있는데, 아이템 빠른 사용과 스킬 빠른 사용으로 나누어서 관리한다면 인벤토리나 스킬 창의 `Index`를 `QuickSlot`까지 관리할 수 있도록 확장해서 제작해도 될 것 같음.

## 2.2.6. AI 시스템

[https://youtu.be/lyFS\\_55N2-U](https://youtu.be/lyFS_55N2-U)



비헤이비어 트리

비헤이비어 트리(BehaviorTree)에서 `UBTService` 를 상속해 만든 `UCheckDistance` 를 통해 몬스터와 플레이어 사이의 거리를 체크할 수 있도록 하였고, `UBTTaskNode` 를 상속받은 `UChangeMonsterState` 를 통해 몬스터가 처한 상황에 따라 몬스터의 상태를 '순찰(Patrol)', '추적(Chase)', '일반 공격(Attack)', '스킬 사용(Skill Use)'의 4가지 상태로 전환할 수 있도록 만들었습니다.



몬스터의 스킬을 제작하면서 몬스터가 스킬 액터를 스폰하고 몽타주를 플레이하고 있을 때, 태스크 노드가 진행되는 문제가 있었는데, 노드의 진행을 원하는 타이밍까지 멈췄다 진행할 수 있는 관련 함수를 찾아 해결할 수 있었습니다.



스킬을 사용하는 보스 몬스터가 여러 종류일 경우를 대비해 하나의 비헤이비어 트리로 모든 경우를 아우를 수 있는 구조를 설계해보려고 했지만, 비헤이비어 트리의 가독성이나 보스 몬스터 움직임의 퀄리티를 고려했을 때 보스 몬스터마다 비헤이비어 트리를 만들어 배치해 주는 것이 더 합리적일 거라 판단함.

## 2.2.7. 퀘스트 시스템

<https://youtu.be/uNDp6mLzz1Q>

```
UCLASS( ClassGroup=(Custom), meta=(BlueprintSpawnableComponent) )
class TESTCPP1_API UQuestLog : public UActorComponent
{
    GENERATED_BODY()

public:
    UQuestLog();

protected:
    UPROPERTY()
    TArray<class AQuest*> m_arQuest;
    UPROPERTY()
    AQuest* m_ActiveQuest;

public:
    void F_AddQuestToLog(AQuest* Quest);

public:
    void F_SetActiveQuest(AQuest* Quest);

public:
    TArray<AQuest*>>* F_GetarQuest();
    AQuest* F_GetActiveQuest();
};
```

**UQuestLog** 를 통해 플레이어가 NPC에게 부여받은 퀘스트를 추가하고, 퀘스트창에서 클릭한 퀘스트를 활성화(Active) 시켜 퀘스트 목표를 플레이어 화면에 띄울 수 있도록 만들었습니다.

```
void AGameMgr::F_NPCUpdateTextRenderHasQuest()
{
    TArray<AActor*> arWorldNPC;
    UGameplayStatics::GetAllActorsOfClass(GetWorld(), ANPC::StaticClass(), arWorldNPC);
    for (int i = 0; i < arWorldNPC.Num(); ++i)
    {
        Cast<ANPC>(arWorldNPC[i])->DELE_UpdateTextRenderHasQuest.Broadcast();
    }
}
```

퀘스트를 클리어하면 게임 매니저에서 NPC들이 보유한 퀘스트의 사전 퀘스트 클리어 유무를 체크하도록 했고, 조건이 충족되었을 경우엔 새로운 퀘스트가 등장하도록 만들었습니다.



NPC의 수가 얼마 되지 않아 새로운 쿼스트를 등장시키기 위해 모든 NPC를 순회하도록 작업했는데, NPC의 수가 많은 경우에는 비효율적일 것으로 보이며, 쿼스트를 지닌 NPC만을 모아 관리하는 것이 나을 듯함.

## 2.2.8. UI 시스템

<https://youtu.be/w0eoJCKfw>

```
void UWidgetManager::F_ToggleUI(EUIType UIType)
{
    if (UIType == EUIType::E_SkillWindow)
    {
        ToggleSkillWindow();
    }
    else if (UIType == EUIType::E_StatusWindow)
    {
        ToggleStatusWindow();
    }
    else if (UIType == EUIType::E_Inventory)
    {
        ToggleInventory();
    }
    else if (UIType == EUIType::E_QuestWindow)
    {
        ToggleQuestWindow();
    }
    else if (UIType == EUIType::E_QuestContent)
    {
        ToggleQuestContent();
    }

    if (_m_bIsOpenedSkillWindow || _m_bIsOpenedStatusWindow || _m_bIsOpenedInventory || _m_bIsOpenedQuestWindow || _m_bIsOpenedQuestContent)
    {
        UWidgetBlueprintLibrary::SetInputMode_GameAndUIEx(_m_PlayerController);
        _m_PlayerController->SetShowMouseCursor(true);
        _m_bIsOpenedAnyUI = true;
    }
    else
    {
        UWidgetBlueprintLibrary::SetInputMode_GameOnly(_m_PlayerController);
        _m_PlayerController->SetShowMouseCursor(false);
        _m_bIsOpenedAnyUI = false;
    }
}
```

게임 플레이에 사용되는 위젯들은 `UWidgetManager`에서 제어하도록 설계했으며, 블루프린트 위젯을 불러오는 창구로도 활용했습니다.

```

void AMyPlayerController::ToggleInventory()
{
    m_pGameMgr->F_GetWidgetMgr()->F_ToggleUI(EUIType::E_Inventory);
}

void AMyPlayerController::ToggleSkillWindow()
{
    m_pGameMgr->F_GetWidgetMgr()->F_ToggleUI(EUIType::E_SkillWindow);
}

void AMyPlayerController::ToggleStatusWindow()
{
    m_pGameMgr->F_GetWidgetMgr()->F_ToggleUI(EUIType::E_StatusWindow);
}

void AMyPlayerController::ToggleQuestWindow()
{
    m_pGameMgr->F_GetWidgetMgr()->F_ToggleUI(EUIType::E_QuestWindow);
}

void AMyPlayerController::TogglePauseMenu()
{
    UWidgetManager* WidgetManager = m_pGameMgr->F_GetWidgetMgr();
    if (!WidgetManager->F_GetbOpenedAnyUI())
    {
        WidgetManager->F_TogglePauseMenu();
    }
}

void AMyPlayerController::CloseUI()
{
    UWidgetManager* WidgetManager = m_pGameMgr->F_GetWidgetMgr();
    if (WidgetManager->F_GetbOpenedAnyUI())
    {
        WidgetManager->F_CloseAllUI();
    }
}

void AMyPlayerController::UseQuickSlot1()
{
    m_pGameMgr->F_GetWidgetMgr()->F_GetQuickSlotWindow()->F_QuickSlotUse(0);
}

void AMyPlayerController::UseQuickSlot2()
{
    m_pGameMgr->F_GetWidgetMgr()->F_GetQuickSlotWindow()->F_QuickSlotUse(1);
}

```

**AMyPlayerController** 에서는 플레이어 입력에 대응하는 동작을 명확하게 하기 위해서 코드를 최대한 간결하게 작성했고, 원하는 위젯을 토클(Toggle)시킬 때는 **enum class** 를 사용하여 코드를 명시적으로 작성했습니다.

## 2.2.9. 능력치 시스템

<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/eb1b82a8-5e7a-4c87-85e1-1782dd14ff43/%EC%83%81%ED%83%9C%EC%B0%BD.mp4>

플레이어 캐릭터에 부착된 `UPlayerStatusComponent` 와 상태창 위젯이 유기적으로 동작하도록 설계했습니다.

캐릭터는 '힘', '민첩', '체력', '지력' 능력치를 가지고 있으며, 능력치를 올리면 오른쪽 표처럼 증가합니다. 증가 내용은 바로 상태창에 반영이 되며, 공격 스킬을 사용하거나 버프 스킬을 사용해도 변경 내용이 즉각적으로 반영되도록 만들었습니다.

스탯	증가 내용
힘	공격력 + 3
민첩	방어력 +3
체력	최대 HP +50
지력	최대 MP +50

그리고, 캐릭터들의 능력치를 이용해서 데미지 시스템을 제작하고자, 캐릭터의 스킬이나 공격에 따라 그 순간에만 능력치가 변화하도록 설계했습니다.



공격 스킬이나 버프형 스킬로 인한 능력치 변화가 캐릭터의 장비나 캐릭터의 상태에 따라 유동적으로 변화하도록 만드는 작업이 어려웠는데, 각각의 상황을 변수로 백업하여 작업하는 것으로 해결했습니다.



다음부턴 캐릭터의 능력치를 변화시키는 것으로 데미지 시스템을 구축하지 않고, 캐릭터 능력치의 변화 없이 오직 변수만을 이용해 데미지 시스템을 구축할 것 같음.



### 3. 게임 개발을 하면서 느낀 점

포트폴리오를 끝까지 봐주셔서 감사드립니다.

마지막으로, 게임 개발하면서 느꼈던 점과 앞으로의 목표에 대해 간략하게 말씀드리고자 합니다.

처음 하게 된 게임 개발은 어려움으로 가득했습니다. 게임 학원을 통해 C++ 언어와 언리얼 엔진의 블루프린트로 게임을 제작하는 방법에 대해서는 배웠지만, 실제로 블루프린트가 아닌 C++를 이용해 게임을 제작하는 것은 별개의 문제처럼 느껴졌고, 막막함을 느꼈습니다. 하지만, 차근차근 기능들을 하나, 둘씩 C++를 통해 구현해 나가면서 점점 C++로 게임을 개발하는 것에 익숙해지는 느낌을 받을 수 있었습니다. 분명, 아직 클래스를 설계하고, 설계한

기능들이 버그 없이 한 번에 돌아가도록 만들기에는 미숙한 부분이 있습니다. 그럼에도 불구하고, 버그를 발견하거나, 막히는 부분이 발생했을 때, 처음에는 헤매거나 갈피를 못 잡을지라도 고민 끝에 이를 해결했을 때 한 층 성장한 저 자신을 발견할 수 있었습니다.

게임을 개발하면서 느꼈던 가장 큰 어려움은 동료의 부재였습니다. 처음 만드는 게임인 만큼 모든 게임 개발 과정을 겪어봐야 다른 직무 사람들의 고충을 이해할 수 있겠다고 생각했습니다. 역시, 기획 단계를 시작으로 자료 조사, 레벨 제작, 콘텐츠 제작, 애니메이션 제작, 게임 테스트에 이르기까지 쉬운 일은 없었고, 그 과정을 통해 동료의 필요성을 느낄 수밖에 없었습니다.

저는 게임 개발을 진행하면서 협업의 중요성에 대해 다시 한번 깨달을 수 있었습니다. 회사에 입사하게 되면 다양한 직무의 사람들과 협업하게 될 것이고, 그들과 원활한 소통을 나누어야 좋은 게임이 탄생할 것입니다. 그래서, 저는 개발자로서의 성장뿐만 아니라, 다른 직무 사람들의 고충을 이해하고, 그들과 소통해 나가면서 팀에서 중심축을 담당할 수 있는 회사에서 꼭 필요로 하는 인재가 되는 것이 목표입니다.