2단계_09. 무한 맵의 제작

수강일	@2023/07/12
이름	전영재
멘토	Min-Kang Song,현웅 최
작성 상태	Done
단계	2단계
강의 시청 여부	☑

Contents

무한 맵의 제작

💡 **강의 목표**

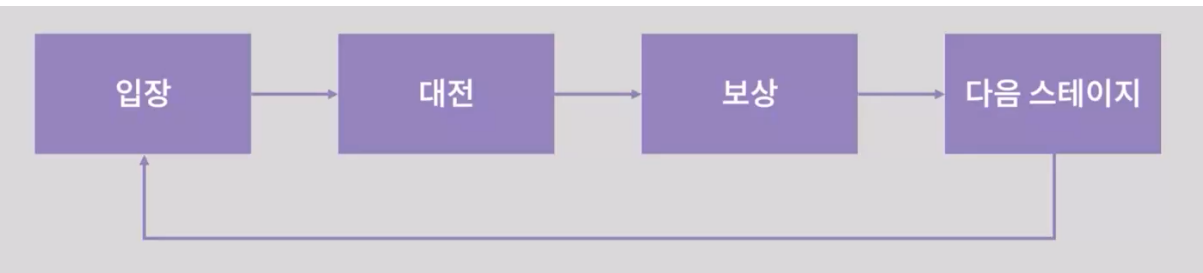
- 무한 맵 생성을 위한 기믹 액터의 설계
- 애셋 매니저를 활용한 애셋 관리 방법의 학습
- 액터의 스폰과 약참조 포인터의 실습

---

[1] 스테이지 기믹 기획

실습의 기믹은 다음과 같다.

1. 입장 : 플레이어가 맵에 입장한다.
2. 대전 : 플레이어와 NPC가 1대1로 대전한다.
3. 보상 : 승리한다면 플레이어는 보상을 획득한다.
4. 다음 스테이지 : 다음 스테이지로 이동한다.

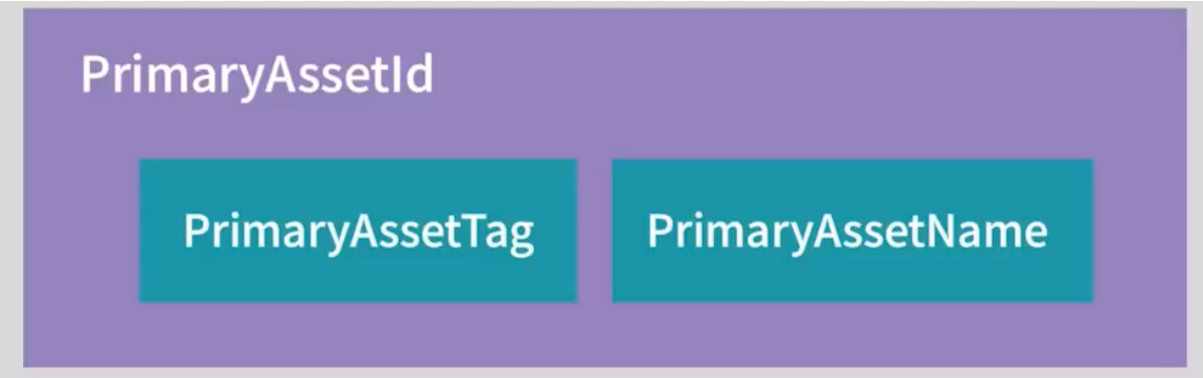


[2] 애셋 매니저

애셋을 관리하는 **싱글톤** 클래스로, 언리얼 엔진이 초기화될 때 단 하나의 인스턴스만을 보장받는다. 이를 통해서 게임이 진행되는 동안 **언제든지 애셋 데이터에 대한 요청**이 가능하다.

지금까지는 애셋의 **레퍼런스 복사**를 통해 수동으로 해당 정보를 얻어왔지만, 애셋매니저를 사용한다면 `PrimaryAssetId`를 통해 프로젝트 내 애셋의 주소를 얻어올 수 있다.

PrimaryAssetId는 다음과 같이 태그와 애셋네임으로 구성되어 있는데, 이를 통해 이름으로 검색해오거나 특정 태그를 가진 모든 애셋 목록을 가져올 수도 있다.

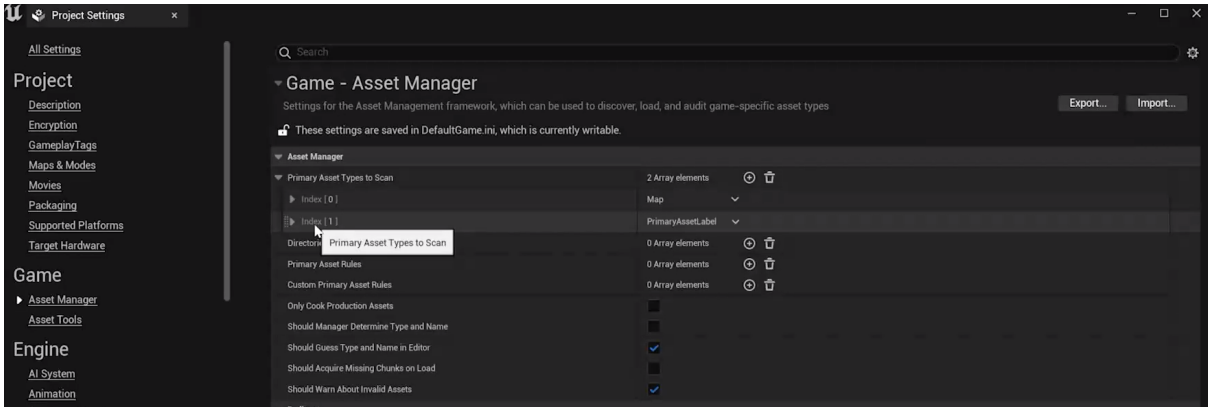


이를 활용해 랜덤 아이템 보상을 구현할 수 있다.

1. 프로젝트 설정에서 애셋들이 담긴 **폴더를 지정**하면 해당 폴더 내의 모든 애셋에 대한 정보를 받아올 수 있고
2. 아이템 데이터에 **애셋 태그를 설정**하여
3. 생성되는 아이템 박스에 아이템태그가 달린 **랜덤의 애셋을 넣어주는** 것이다.

> 💡 나도 이 기능을 사용해 EnemySpawningPool에서 다양한 EnemyClass에 대한 정보를 얻어와 랜덤으로 몬스터를 스폰시킬 수 있을 것이다.

다음과 같이 프로젝트 세팅에서 애셋매니저의 경로와 태그를 설정할 수 있다.



이후 코드에서 다음과 같이 태그를 달아주면 된다.

```
ABItemData.h
...
public:
  FPrimaryAssetId GetPrimaryAssetId() const override
  {
    return FPrimaryAssetId("ABItemData", GetFName()); //왼쪽이 태그, 오른쪽이 데이터네임
  }
```

이렇게 태그가 달린 애셋의 목록을 가져오는 방법은 다음과 같다.

```
void AABItemBox::PostInitializeComponents()
{
  Super::PostInitializeComponents();

  UAssetManager& Manager = UAssetManager::Get();

  TArray<FPrimaryAssetId> Assets;
  Manager.GetPrimaryAssetIdList(TEXT("ABItemData"), Assets); //태그가 달린 애셋목록을 배열로 반환
  ensure(0 < Assets.Num());

  int32 RandomIndex = FMath::RandRange(0, Assets.Num() - 1);
  FSoftObjectPtr AssetPtr(Manager.GetPrimaryAssetPath(Assets[RandomIndex]));
  if (AssetPtr.IsPending())
  {
    AssetPtr.LoadSynchronous();
  }
  Item = Cast<UABItemData>(AssetPtr.Get());
  ensure(Item);
}
```

TMap 사용법

TMap 자료구조를 사용하는 방법이다. 기믹과 관련된 내용은 아니지만 내가 TMap의 사용법에 익숙치 않기에 리마인드 겸 가져와봤다.

```
ABStageGimmick.h
protected:
  UPROPERTY(VisibleAnywhere, Category = Gate, Meta = (AllowPrivateAccess = "true"))
  TMap<FName, TObjectPtr<class UStaticMeshComponent>> Gates;
```

```
ABStageGimmick.cpp
AABStageGimmick::AABStageGimmick()
{
  ...
  // Gate Section
  static FName GateSockets[] = { TEXT("+XGate"), TEXT("-XGate"), TEXT("+YGate"), TEXT("-YGate") };
  static ConstructorHelpers::FObjectFinder<UStaticMesh> GateMeshRef(TEXT("/Script/Engine.StaticMesh'/Game/ArenaBattle/Environment/Prop
  for (FName GateSocket : GateSockets)
  {
    ...
    Gates.Add(GateSocket, Gate);
    ....
  }
  ...
}
```

OnConstruction

이 함수는 액터의 속성이 변한다면 호출이 되는 함수이다.

그렇기에 override를 통해 특정 로직을 추가해주면, 프로퍼티 변화에 맞춘 액터의 함수 호출 등이 가능하다. 본 강의에서는 프로퍼티의 변화에 맞춰 `SetState`를 해주는 기능을 구현했다.

> 💡 그러나 생각보다 사용에 주의를 필요로 해보인다.
> 모든 프로퍼티의 변화를 추적하기에 원하지 않는 타이밍에 함수가 호출될 수도 있다.

TSubClassOf

TObjectPtr과 같이 프로퍼티를 설정할 때 사용하는 템플릿으로, 설정한 클래스를 **상속받는 클래스만**을 표시하는 기능을 갖고있다.

블루프린트를 통해 확장을 거친다면 많은 액터 목록이 생성될텐데 이 중 해당 클래스에 연관된 데이터만을 표시하기에 편리하다.

```
UPROPERTY(EditAnywhere)
TSubClassOf<class AABCharacterNonPlayer> OpponentClass;
//이경우 AABCharacterNonPlayer를 상속받는 클래스만을 표시한다.
```

TWeakObjectPtr

독립적인 다른 액터에 대한 포인터를 소지할 때, 해당 액터는 개별의 기능으로 인해 `Destroy` 될 수 있다. 하지만 기존의 TObjectPtr을 사용하게 되면 해당 액터를 소지하고 있는 액터는 파괴 된 액터가 **아직 존재한다고 생각**하여 메모리에서 소멸되지 않을 수 있다.

그렇기에 해당 액터와는 **무관하지만 관리는 해야한다는** 의미의 약참조를 사용하는 것이 좋다.

> 💡 내 프로젝트에도 사용할 수 있겠다!
> 터렛들은 적의 존재를 판별하기 위해 overlap으로 그들의 포인터를 소유하게 된다.
> 그러나 해당 액터들은 전투로 인해 파괴 될 수 있고 이것은 터렛이 관여하는 부분은 아니다.
> 그렇기에 약참조를 사용하는 것이 더욱 안전하다.

📌 **Summary**

- TMap, OnConstruction, TSubClassOf, TWeakObjectPtr의 활용
- 애셋매니저를 활용한 특정 태그 애셋 로딩 방법

0n. 무한 맵의 제작 강의 과제

❓ Q1. 현재 프로젝트에서 사용하는 기믹을 설명하고, 이들이 어떤 상태를 거쳐 발전되는지 정리하시오.

게임의 진행상황에 따라 몬스터를 스폰하는 **스포닝풀** 기믹이 존재한다.

- **준비**: 게임시작과 동시에 설정되며 몬스터를 **미리 스폰**하여 게임도중의 생성을 대신한다.
- **생성**: 시작 후 타이머로 인해 상태가 변화하며 미리 스폰해놨던 몬스터들을 **월드 내로 불러온다**.
- **휴식**: 준비해놓은 모든 몬스터를 소환하면 더 이상의 작동을 멈추고 **휴면** 단계로 돌입한다.

게임의 한 스테이지가 종료되면 다시 **준비** 단계로 돌아가며 다음 스테이지를 위한 액터를 스폰하며 위 상태를 반복한다.

❓ Q2. 애셋매니저로부터 다수의 애셋을 필요할 때 제공 받는 방법은 현재 게임에서 어떻게 활용될 수 있는지 생각하시오.

위 스포닝풀 기믹에서는 미리 몬스터를 스폰한다.

이 때 기존의 방식으로는 스포닝풀 액터에서 소지하고 있는 몬스터 클래스를 기반으로 정해진 몬스터를 생성하게 되는데, 이러한 클래스를 **애셋매니저**를 사용하여 런타임 도중 **랜덤한 몬스터**를 클래스로 불러올 수 있다면 게임 플레이의 **다양성**이 증가할 것으로 예상된다.