



# 4주차 (2023.10.09)

## 간단한 소개

### ▼ 펼치기

#### 해당 강의 대한 목표:

- Lyra를 Clone을 통한 전체적인 구조 이해
- Lyra는 굉장히 다양한 요소와 깊은 내용을 담고 있음:
  - 우선 해당 강의에서는 Lyra에 대한 전체적인 구조를 이해하고 이를 활용하여, 깊은 내용을 공부할 수 있는 기초 체력 만드는 것이 목표!

#### 강의 설명:

- 해당 강의는 앞서, 이미 선행 과정을 통해 진도가 나간 상황
- 처음부터 다시 설명하지는 않으나, Asset Manager부터 시작할 예정
- 이전 부분은 아마 Rookiss님이 설명해주실듯? 🦊

#### 강의에 대한 질문답 시간 (10분 정도)

## HakAssetManager

### ▼ 펼치기

#### [HakGameModule]

#### 우선 LyraGameModule을 구현해주시자:

```

1 Context Copyright Epic Games, Inc. All Rights Reserved.
2
3 #include "HakGame.h"
4 #include "Modules/ModuleManager.h"
5
6 /**
7  * FHakGameModule
8  */
9 class FHakGameModule : public FDefaultGameModuleImpl
10 {
11 public:
12     virtual void StartupModule() override;
13     virtual void ShutdownModule() override;
14 };
15
16 void FHakGameModule::StartupModule()
17 {
18 }
19
20 void FHakGameModule::ShutdownModule()
21 {
22 }
23
24
25
26 #IMPLEMENT_PRIMARY_GAME_MODULE(FHakGameModule, HakGame, "HakGame");

```

- 조금 있다가, StartupModule()과 ShutdownModule()을 활용해볼 것이다:

LyraStarterGame의 GameModule 간단히 확인해보기:

## [HakAssetManager]

HakAssetManager를 Constructor로 만들어보자:

```

1 #pragma once
2
3 #include "CoreMinimal.h"
4 #include "Engine/AssetManager.h"
5 #include "HakAssetManager.generated.h"
6
7 /**
8  * hak asset manager
9  *
10 * game implementation of the asset manager that overrides functionality and stores game-specific types
11 * it is expected that most of games will want to override AssetManager as it provides a good place for
12 * this class is used by setting 'AssetManagerClassName' in DefaultEngine.ini
13 */
14
15 UCLASS(Config=Game)
16 class UHakAssetManager : public UAssetManager
17 {
18     GENERATED_BODY()
19 public:
20     /**
21      * member methods
22      */
23     UHakAssetManager();
24
25     /**
26      * static methods
27      */
28 };
29

```

- 꼭 Constructor에서 BaseClass를 상속받는 것을 잊지 말자!

```

1 #include "HakAssetManager.h"
2
3 #include UE_INLINE_GENERATED_CPP_BY_NAME(HakAssetManager)
4
5 /**
6 * member methods
7 */
8 UHakAssetManager::UHakAssetManager()
9 {
10     // 항상 Constructor를 까먹지 않도록 하자!
11     : UAssetManager()
12 }
13

```

## [HakLogChannels.h]

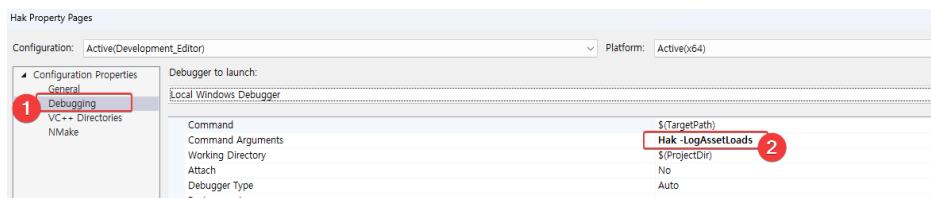
- LyraLogChannels.h를 한번 확인해보기
- HakLogChannels.h 구현
  - Lyra의 경우, 각 Class에 대해 Log 채널을 만들기보다, 하나의 공통된 Log Channel을 선언하여, 통합적 방식을 고려한다.

## [HakAssetManager]

- UHakAssetManager의 static method를 Clone하자:
  - 한번 LyraAssetManager 멤버 변수와 메서드 살펴보기
    - 참고로, 클론 코딩 과정에서 당장 필요하지 않는 변수를 정리하기 위해 메서드를 구현해가며, 멤버 변수를 채워나갈 예정이다:
      - 우리의 클론 코딩의 목표 중 하나는, Lyra의 간략화이다!
    - 클론 코딩 대상들:
      - 당장 AssetManager의 함수를 사용하지 않는다, 허나, AssetManager가 어떤 클래스인지 이해하기 위해 static method에 대해 클론 코딩해보자:
  - UHakAssetManager::TestClone
  - UHakAssetManager::ShouldAssetLoads
    - PRAGMA\_DISABLE\_OPTIMIZATION에 대한 설명
      - bLogAssetLoads를 확인해보면?

Name	Value	Type
&bLogAssetLoads	0x00007ffb8870b50 (UnrealEditor-HakGame.dll!bool bLogAssetLoads) {false}	bool *
UHakAssetManager::ShouldLogAssetLoads	0x00007ffb886f3600 (UnrealEditor-HakGame.dll!UHakAssetManager::ShouldLogAssetLoads...)	bool (void)

- 그리고 -LogAssetLoads를 넣어보자:



- True로 바뀐다는 것을 알 수 있죠?

SynchronousLoadAsset

주석 설명하기

GetAsset, GetSubclass:

AddLoadedAsset

- Thread-safe함을 알 수 있다

- 이를 통해, 물론 GetAsset과 GetSubclass 또한 Thread-safe을 보장함을 확인할 수 있다 (물론 이것두: SynchronousLoadAsset)

UHakAssetManager::SyncObject (FCriticalSection)

### Object Locking에 대한 설명

- Dead-lock free를 위한 설명 → Find-grained locking의 위험성
- Dead-lock 유발 조건: locking 순서!



왜? Asset과 Subclass를 나누어 놓았을까?

정확히는 Class와 Object 구분으로 이해하면 된다:

- Class 는 Template 혹은 Meta Data
- Object 는 Instanced된 객체

예로 들어, 보통 Actor 생성은 BP Class 로딩 → SpawnActor  
StaticMeshComponent 생성은 StaticMesh 로딩(Asset Loading) → CreateObject 후, SetStaticMesh 호출

지금까지 static method를 통해 UAssetManager가 무엇을 하는지 알 수 있다:

- UHakAssetManager는 게임의 로딩을 담당할 시스템으로 이해하면 된다
  - 로딩과 캐싱

#### ✓ UHakAssetManager::StartInitialLoading

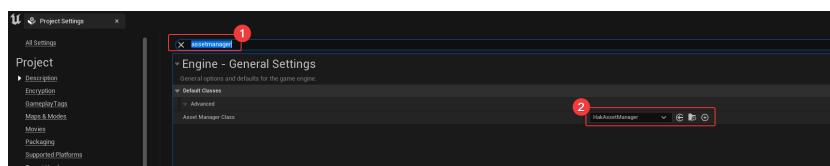
- 단순 로깅만 남기고 넘어가자

#### ✓ 그래도 어떤 callstack으로 호출되는지만 간단히 보자:

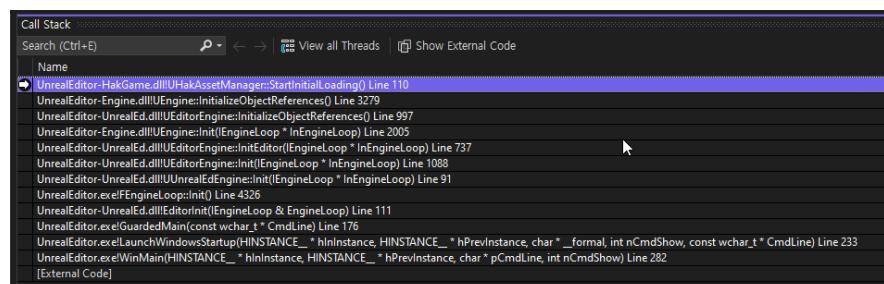
- 어? 안된다..

- 이유는 아직 우리가 AssetManager를 HakAssetManager로 오버라이드 안했기 때문!

#### ✓ HakAssetManager로 오버라이드하자:



- 아래와 같이 정말 초반부에 호출된다:

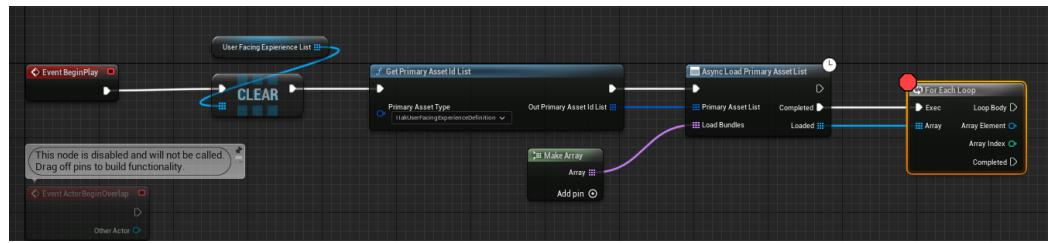


## B\_ExperienceList3D - 0

### ▼ 펼치기

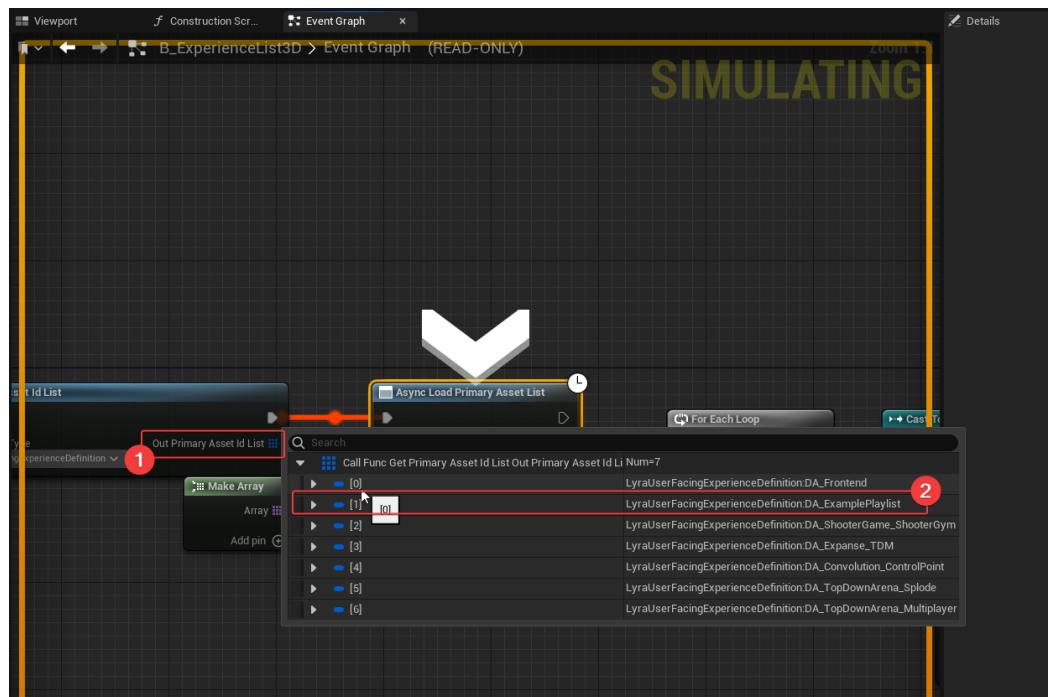
#### ✓ B\_ExperienceList3D 재구성

- 이름과 경로 변경으로 인해 리셋이 필요



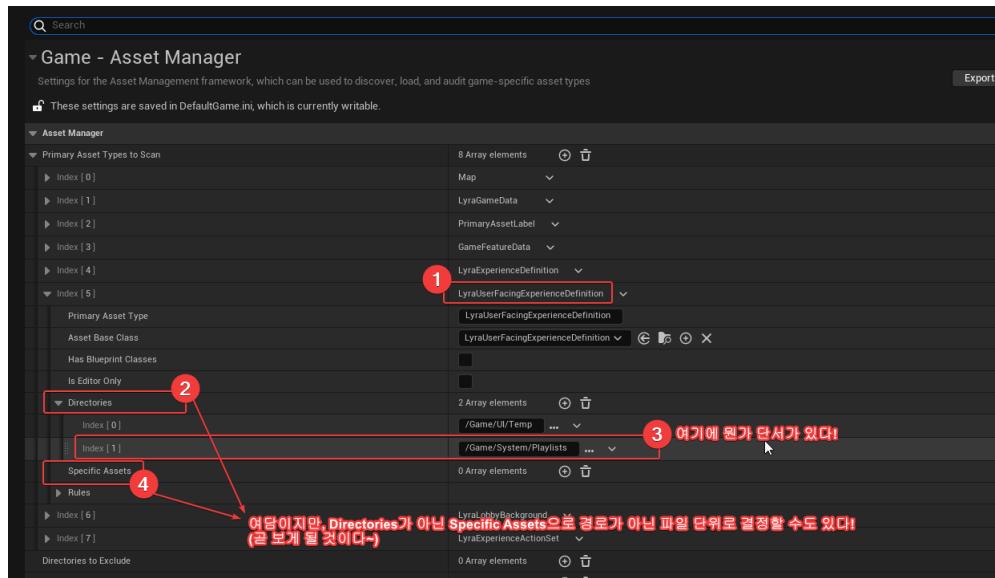
✓ Lyra를 통해 아래의 내용 살펴보기:

- GetPrimaryAssetIdList의 결과물 디버깅:

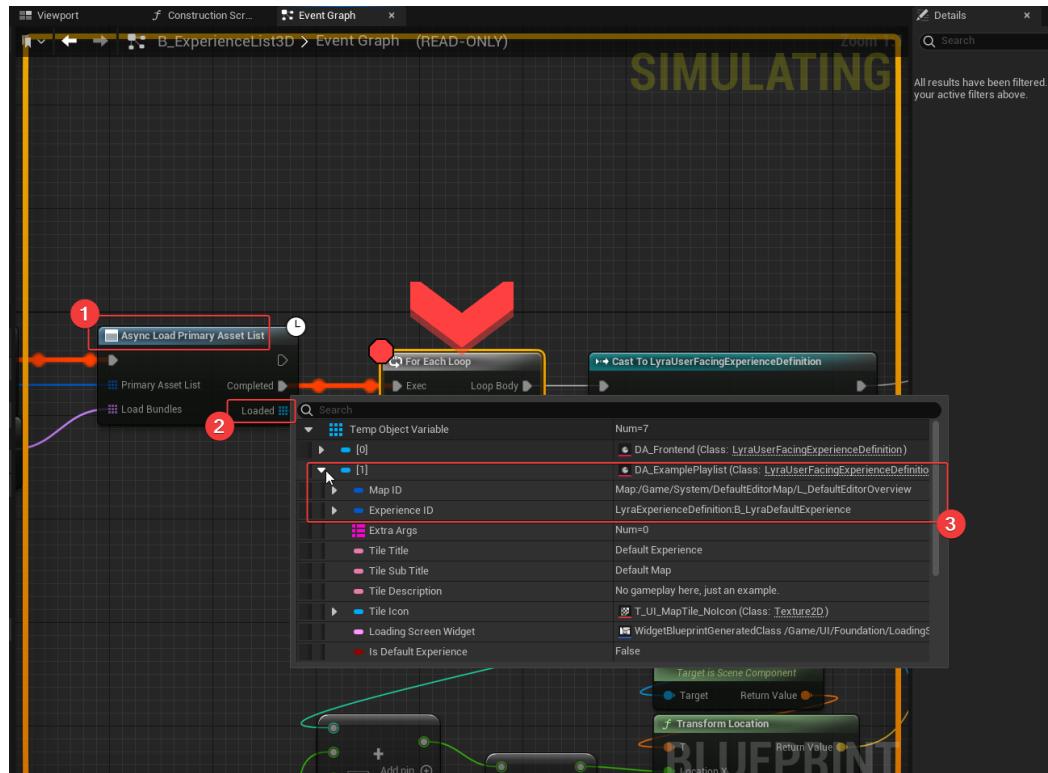


해당 결과물인 UserFacingExperienceDefinition은 어디서 가져오는 걸까?

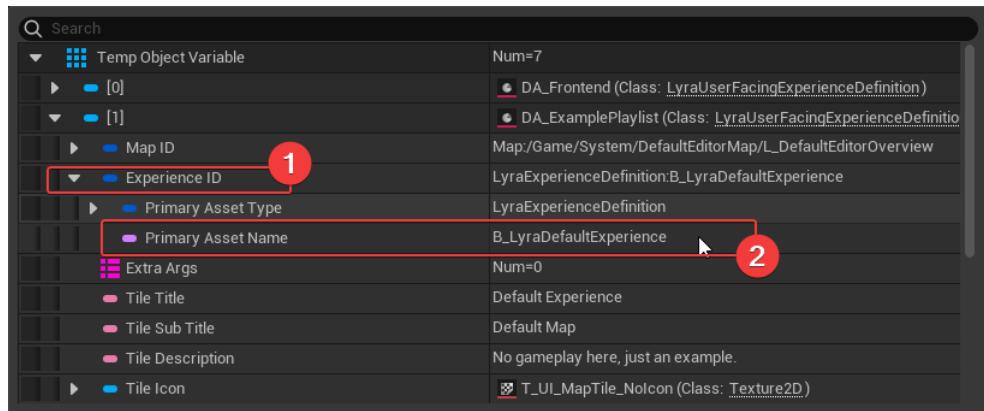
✓ 저는 AssetManager에서 PrimaryDataType을 정의한 곳에서 단서를 찾을 수 있다:



- AsyncLoadPrimaryAssetList의 결과물:



- 우리는 MapID는 이미 알고 있는 L\_DefaultEditorOverview라는 것은 인지하고 있다. 그렇다면, ExperienceID는 무엇인가:
  - 해당 AssetName을 통해 우리는 단서를 유추할 수 있다:



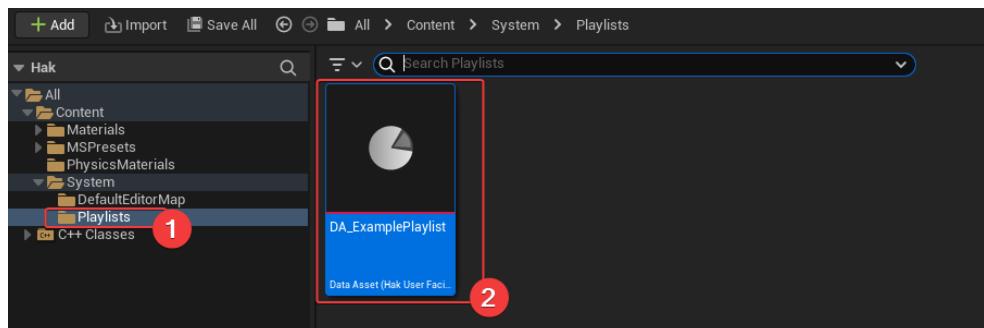
해당 에셋 이름은 `B_LyraDefaultExperience` 이다!

이제 대부분의 단서를 찾았으니, 클론 코딩해보자:

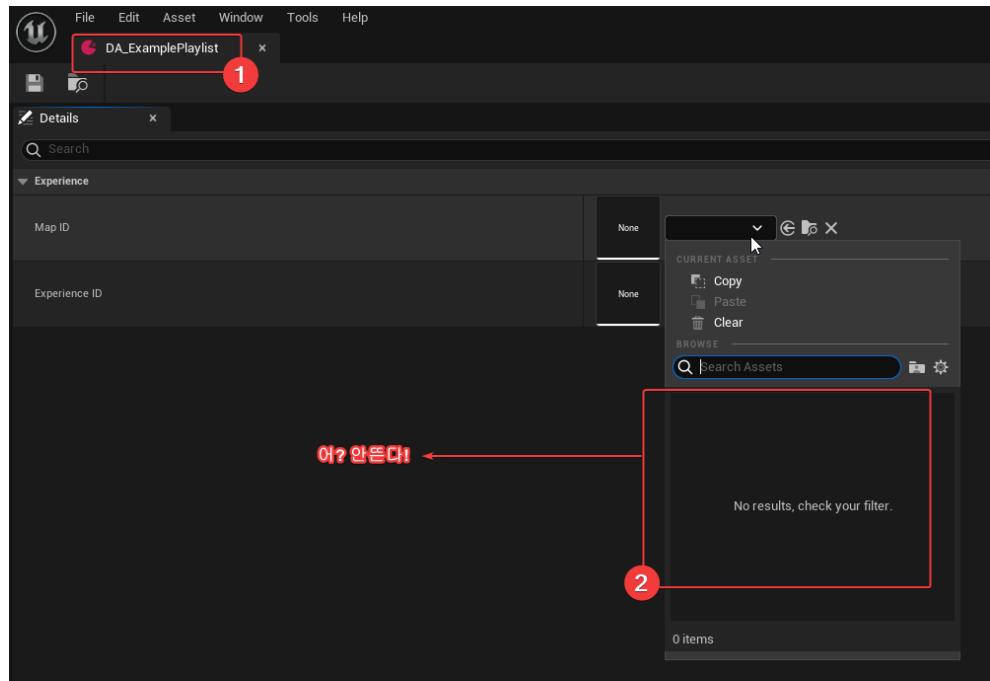
우선 `DA_ExamplePlaylist`를 만들자:

### ▼ 자세히

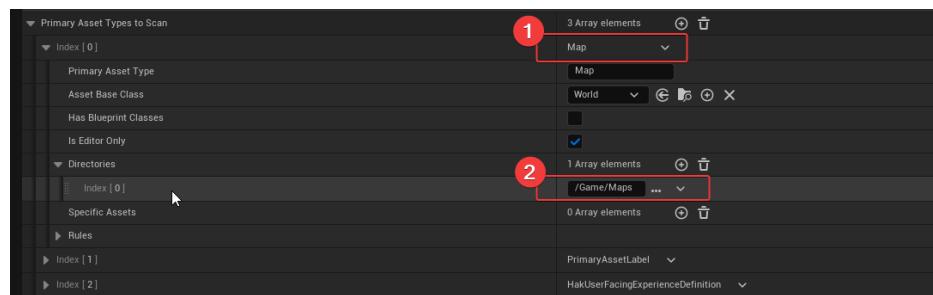
- 아래의 경로에 만들고:



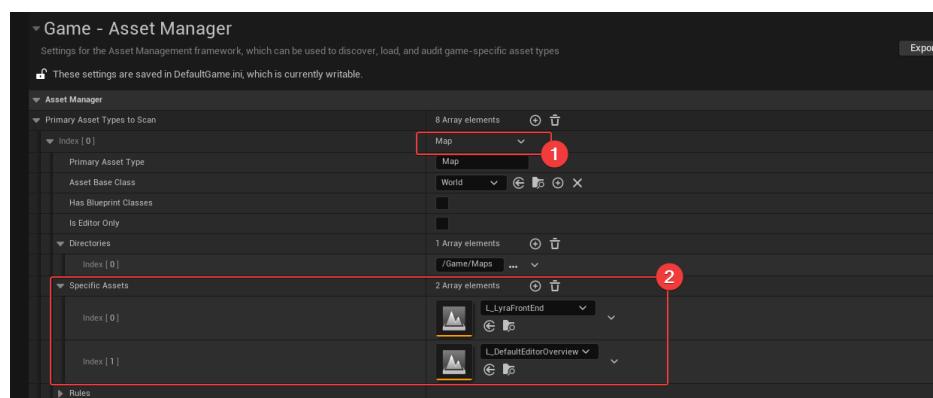
- UserFacingExperienceDefinition을 채우려니...



- 이는 우리가 아마 PrimaryDataType의 경로가 설정되어 있지 않아서 그런가 싶다?

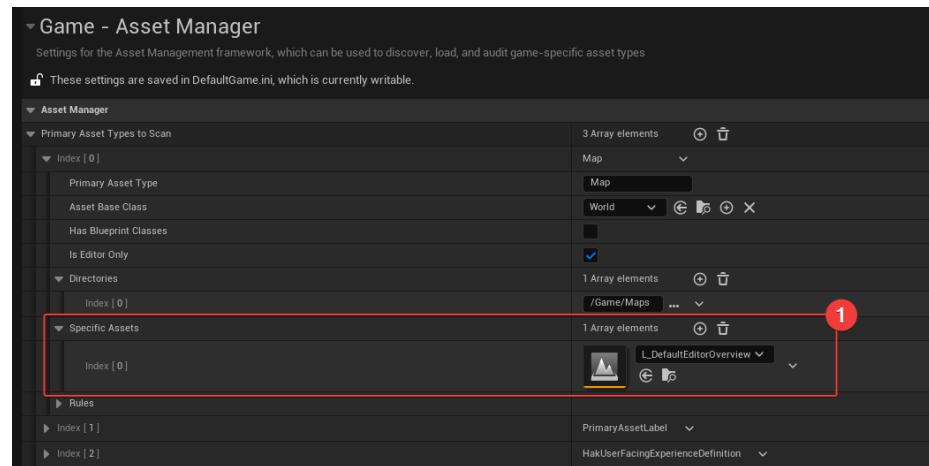


- 역시 Game/Maps로 경로가 설정되어있다
- 혹시 Lyra는 어떻게 설정되어 있는가보자:



## 오! 앞서 언급했던 Specific Assets를 통해서

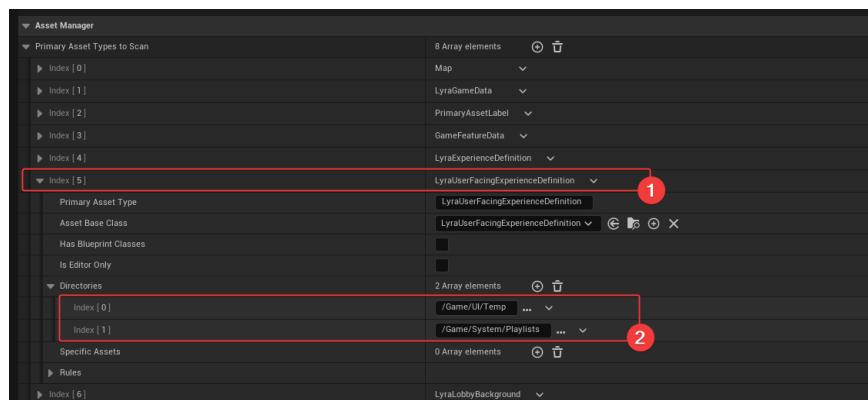
L\_DefaultEditorOverview를 설정했음을 알 수 있다! 그럼 우리도 이렇게 하자!



- 고려할 PrimaryAssetType이 하나 더 있다:

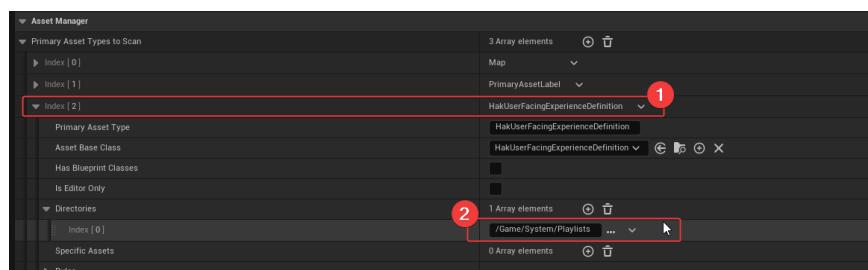
UserFacingExperienceDefinitions

- 이건 Lyra에서 어떻게 되어있는지 한번 보고 가자:



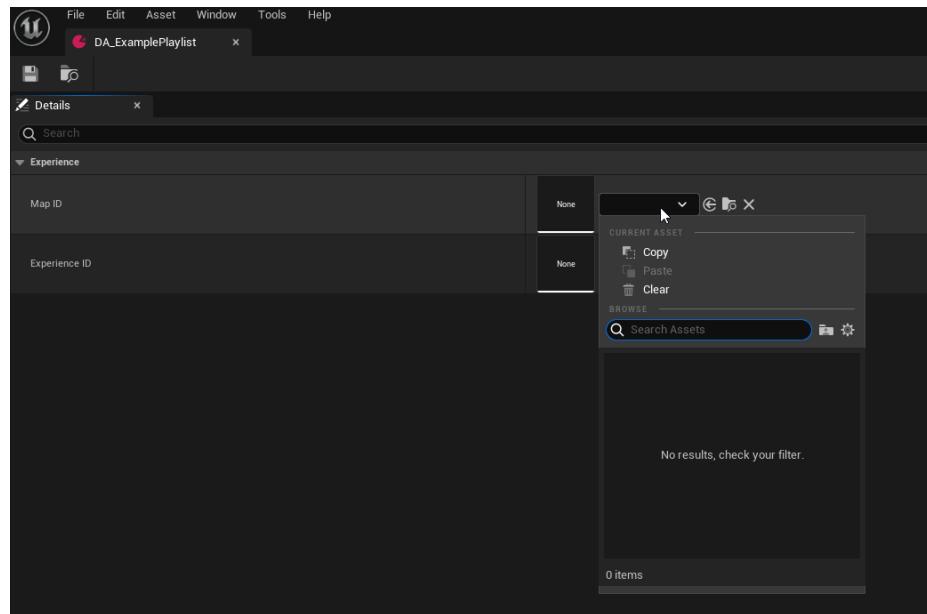
UserFacingExperienceDefinition은 경로가 하나 더 추가되어 있음을 알 수 있다!

- 우리도 이걸 똑같이 해주자:



- 그럼 이제 HakUserFacingExperienceDefinition의 DA\_ExamplePlaylist 가 정의될 것이다:

○ 응? 뭐징?



□ 다시 한번 HakUserFacingExperienceDefinition C++를 점검해보자:

```
/*
 * UHakUserFacingExperienceDefinition
 * - description of settings used to display experiences in the UI and start a new session
 */
UCLASS(BlueprintType)
class HAKGAME_API UHakUserFacingExperienceDefinition : public UPrimaryDataAsset
{
    GENERATED_BODY()
public:
    /**
     * member variables
     */
    /** the specific map to load */
    UPROPERTY(BlueprintReadWrite, EditAnywhere, Category=Experience, meta=(AllowedTypes="Maps"))
    FPrimaryAssetId MapID;

    /** the gameplay experience to load */
    UPROPERTY(BlueprintReadWrite, EditAnywhere, Category=Experience, meta=(AllowedTypes="HakExperienceDefinition"))
    FPrimaryAssetId ExperienceID;
};
```

1  
안! Maps가 아니라 Map이다!

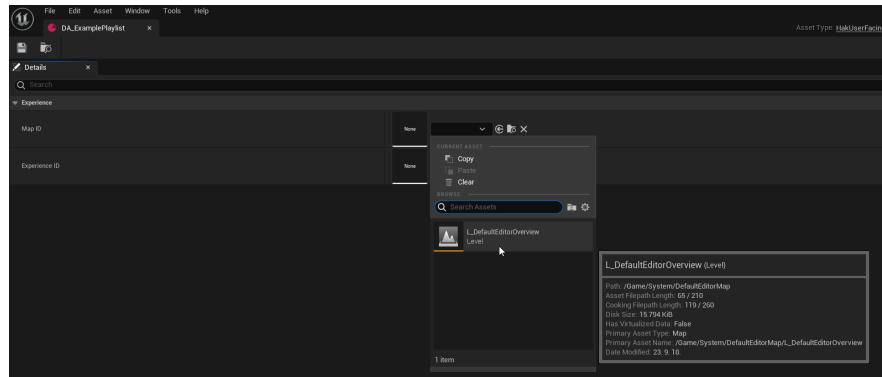
□ 그래도 안된다... 설마?

```
/*
 * UAssetManager's interfaces
 */
void UHakAssetManager::StartInitialLoading()
{
    // 오버라이드할 경우, Super의 호출은 꼭 까먹지 말자
    Super::StartInitialLoading();

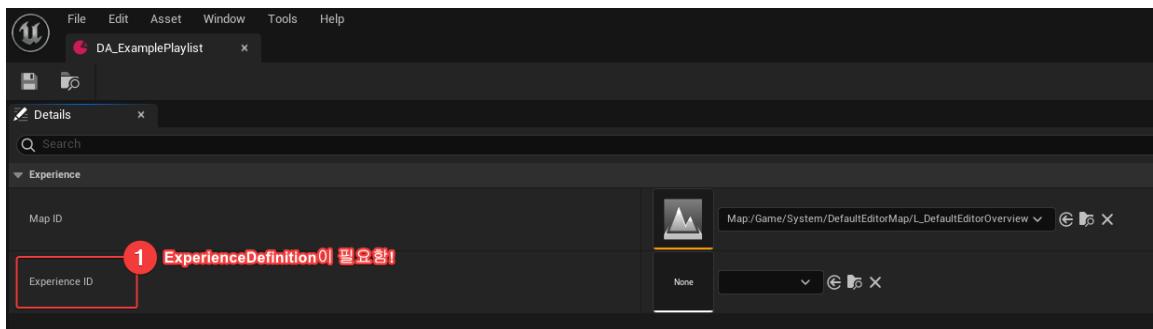
    // 일단 로깅만하고 넘어가자
    UE_LOG(LogHak, Display, TEXT("UHakAssetManager::StartInitialLoading"));
}
```

1

- 이제 잘 뜬다!!~



이제 LyraExperienceDefinition을 Clone할 때가 되었다:

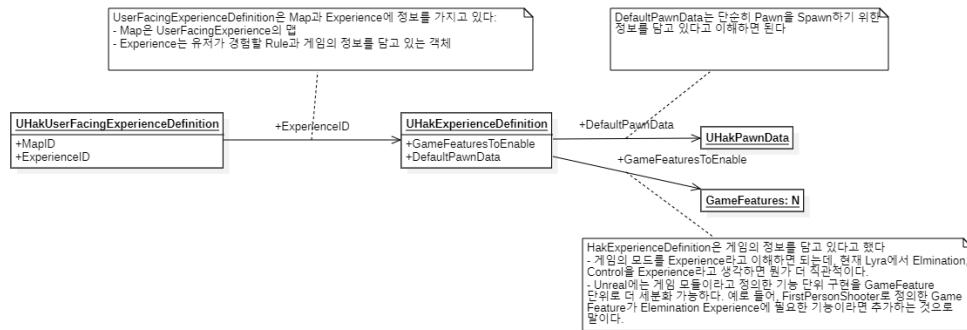


그럼 잠깐 다시 `HakExperienceDefinition` 을 만들고 오자!

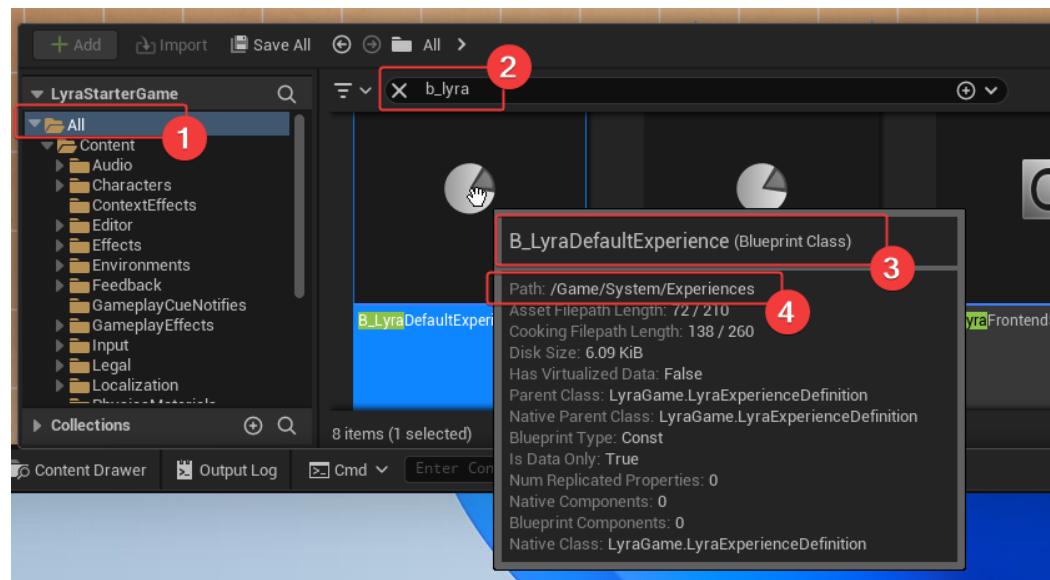
## HakExperienceDefinition

### ▼ 펼치기

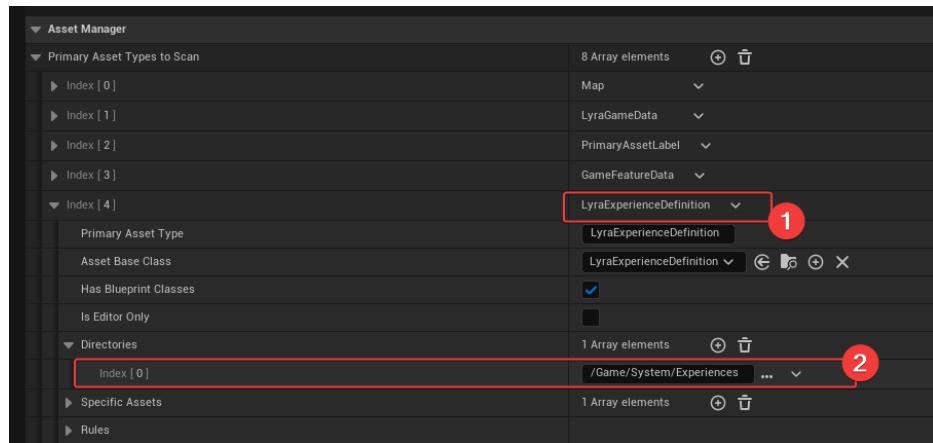
- 간단한 Lyra의 ExperienceDefinition 구조를 살펴보자:
  - `ULyraExperienceDefinition`
  - `ULyraPawnData`
- 잠깐 살펴보는 각 객체들 간의 관계:



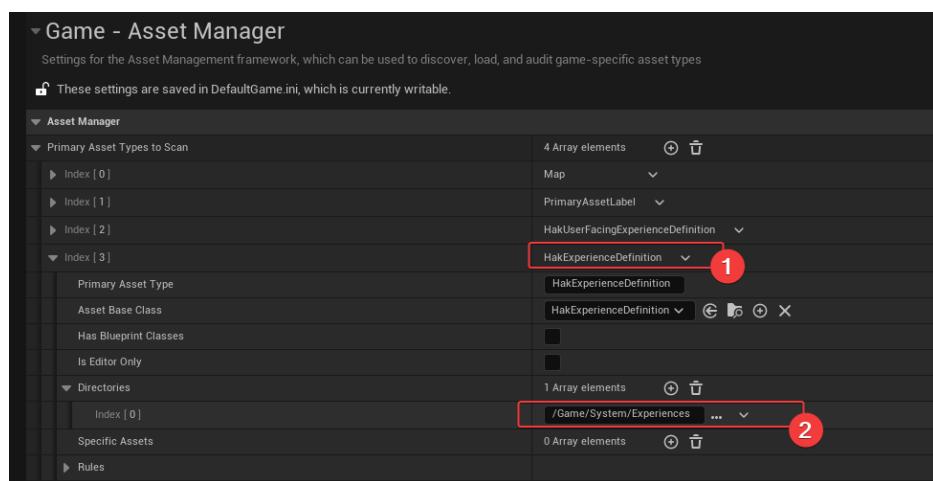
- HakExperienceDefinition, HakPawnData를 구현
- HakUserFacingExperienceDefinition이 DA\_ExamplePlaylist를 완성시키자:
- B\_LyraDefaultExperience 확인



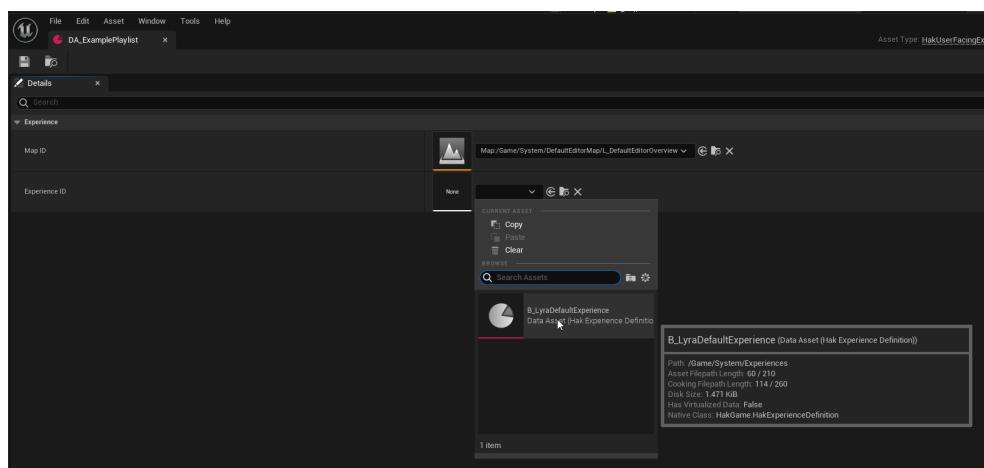
- HakGame에 똑같은 경로 (`/Game/System/Experiences/`)에 생성해주자!
- 당연히 새로운 PrimaryDataType을 추가시켜줘야겠지?
  - 아래는 Lyra의 상태:



- 아래는 Hak의 상태 (똑같이 설정하자):



- 그럼 아래와 같이 잘 확인됨을 볼 수 있다:



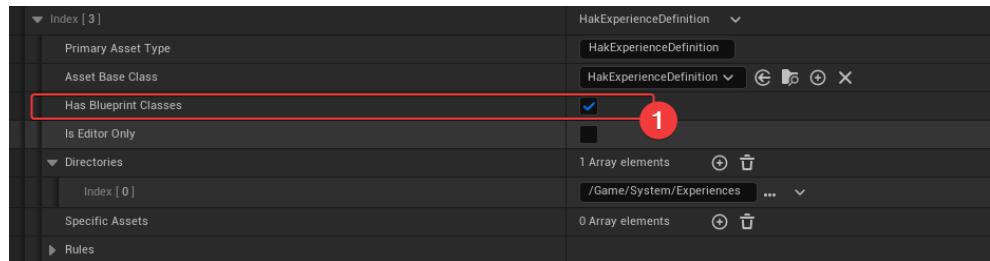
- 참고:

혹시, 여전히 보이지 않을 경우, 아래를 참고하여 한번 확인하자!

```

5  /**
6   * forward declaration */
7   class UHakPawnData;
8
9  UCLASS(BlueprintType)
10 class UHakExperienceDefinition : public UPrimaryDataAsset
11 {
12     GENERATED_BODY()

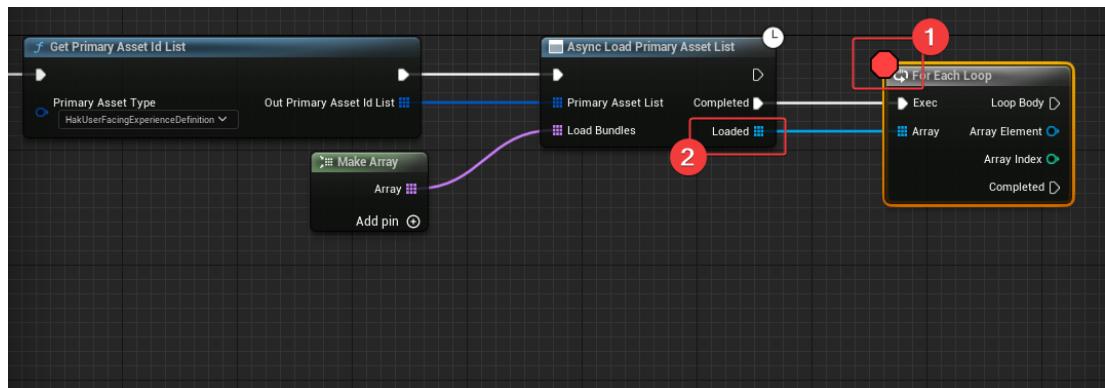
```



## B\_ExperienceList3D - 1

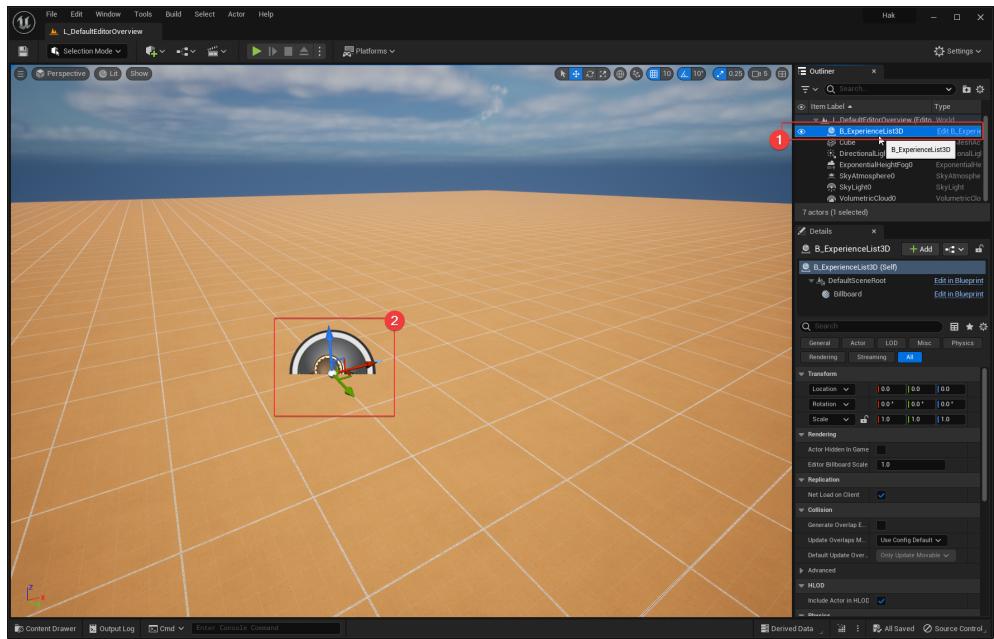
### ▼ 펼치기

- 이제 아래와 같이 코드를 완성하여 AsyncLoadPrimaryAssetList 결과물을 확인해보자:

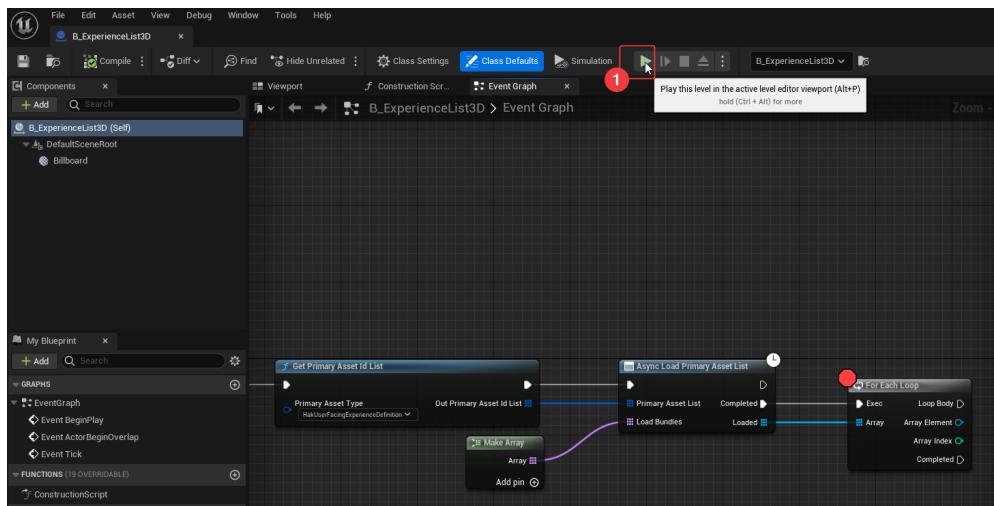


- 아래와 같이 우리가 의도한 결과가 나옴을 확인하자:

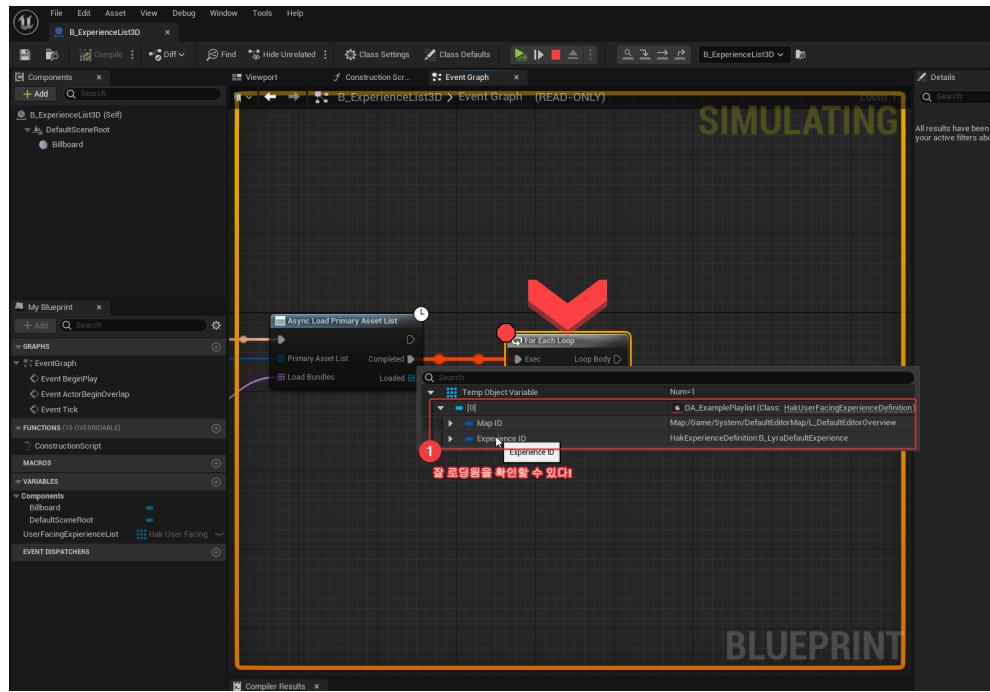
- 우선 우리는 해당 BP를 월드에 배치해야 한다:



□ 아래와 같이 Play 버튼을 통해 디버깅 해보자:

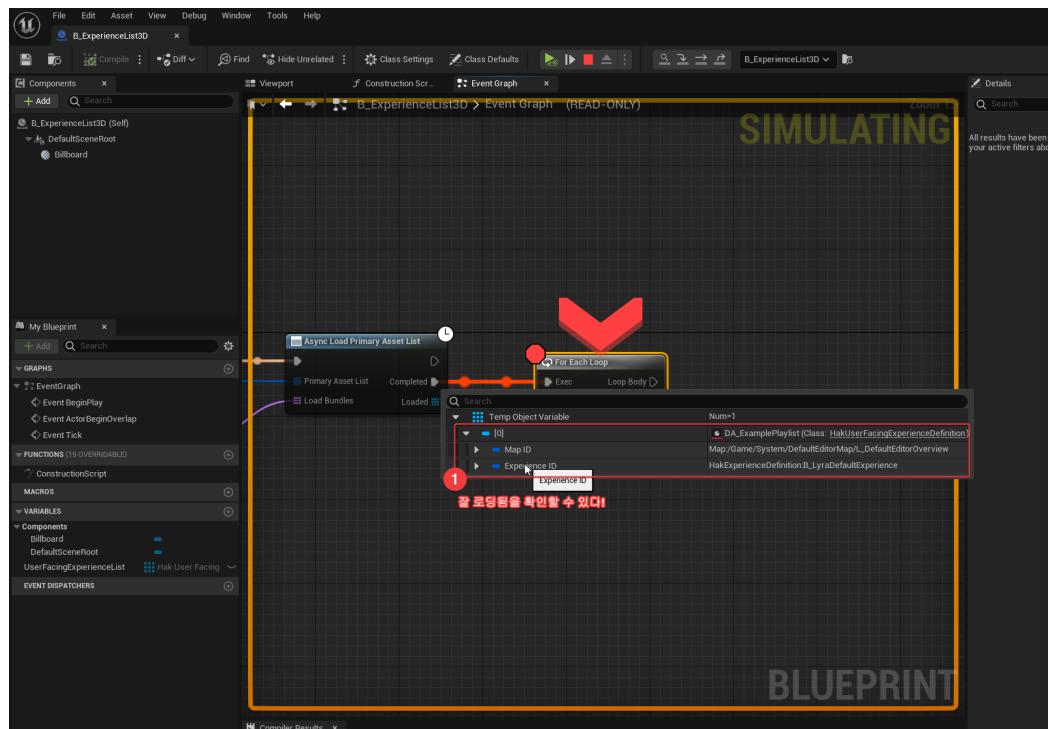


○ 아래와 같이 잘 로딩됨을 확인할 수 있다:



□ 이제 나머지 BP를 Clone해보자:

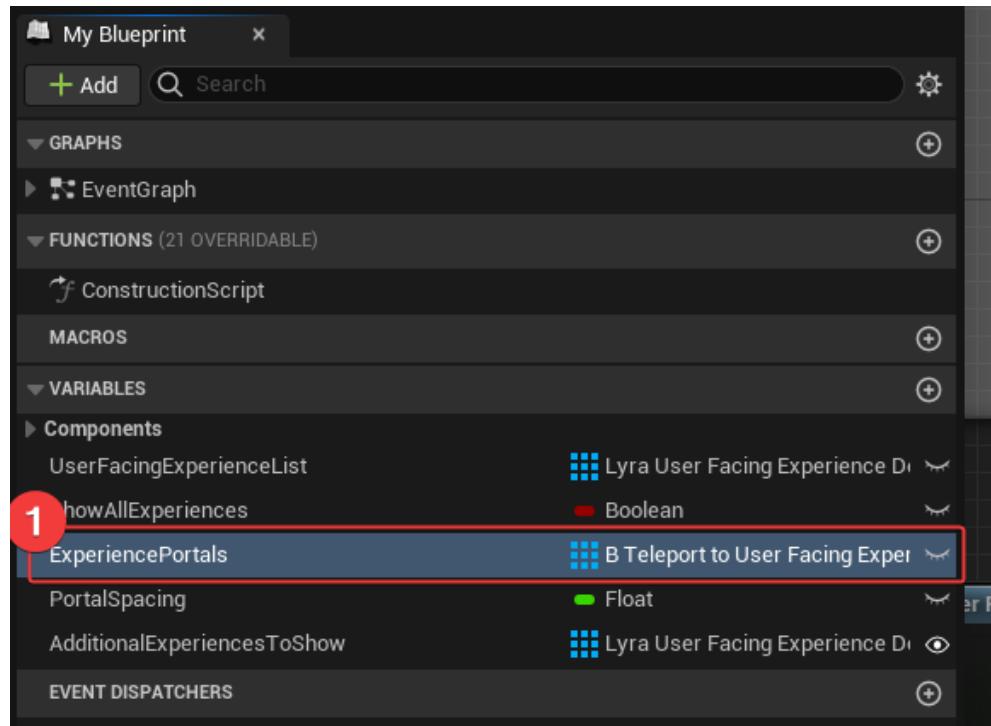
- 로딩된 UserFacingExperienceDefinitions을 순회하면서, UserFacingExperienceList의 멤버변수에 추가해주고, 잘 들어갔는지 디버깅 해보자:



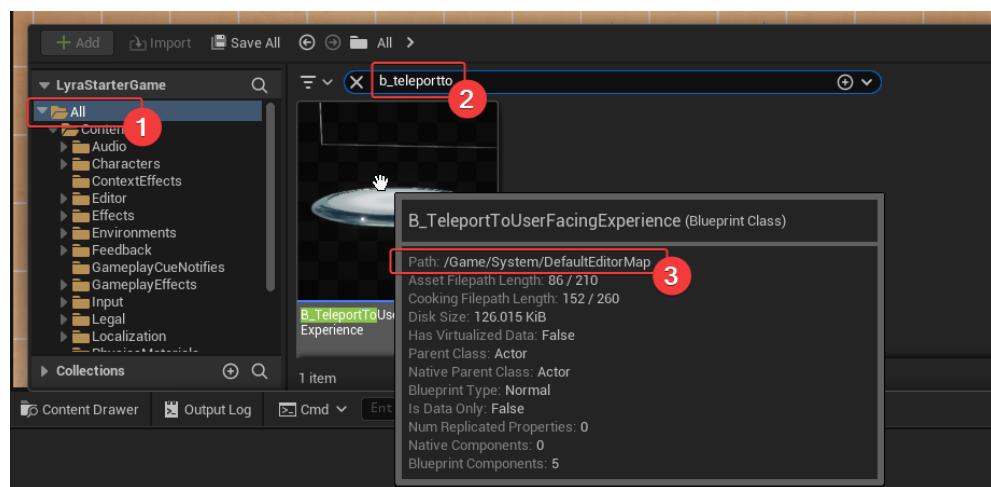
□ 잘 추가되었다는 것을 확인하였다면, Portal을 추가할 차례이다:

□ 우선, Lyra에서 현재 Portal이 무엇인지 확인해보자:

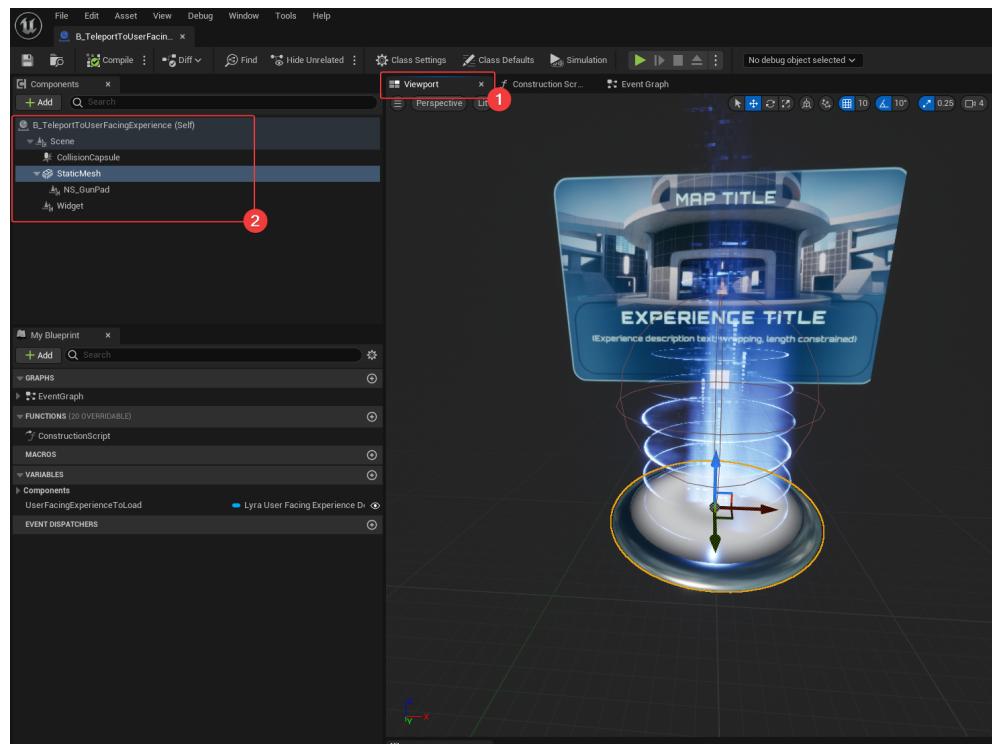
- 우선 Portal은 `B_TeleportToUserFacingExperience` 인거 같다:



- 검색해보면, 현재 `/Game/System/DefaultEditorMap` 경로에 있음을 확인할 수 있다:



- 해당 BP를 열어서 Viewport를 보자:

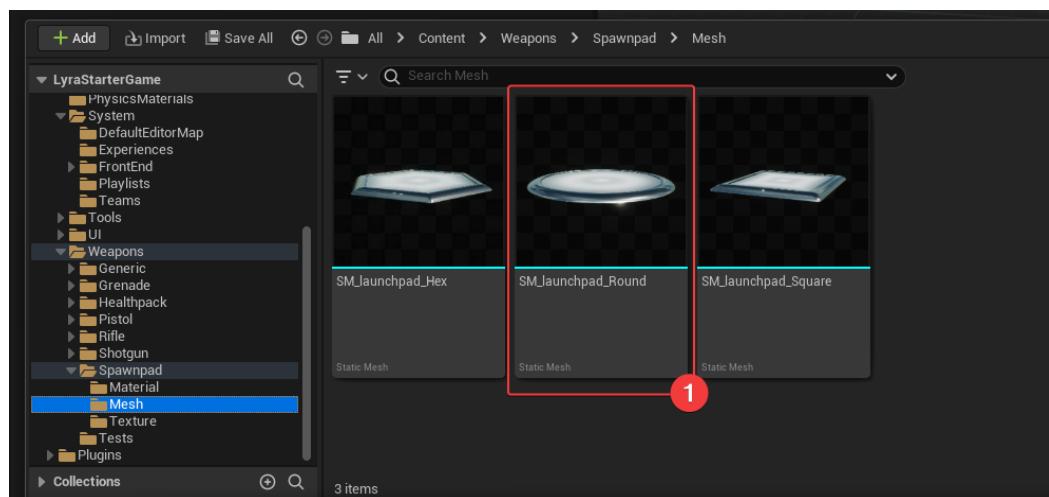


- 우리는 여기서 단순히 Mesh와 CollisionCapsule 정도만 추가하려고 한다



물론, `B_TeleportToUserFacingExperience`에 EventGraph가 많으나, 후일 우리가 이제 Teleport를 구현할 때, 보도록 하고 지금은 넘어가자.

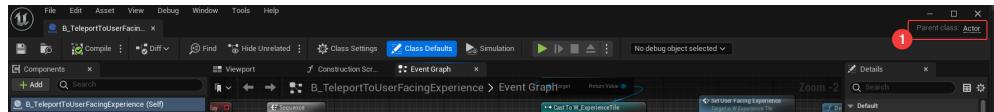
- 앞서 확인한 Lyra에서 필요한 Mesh를 우리쪽 프로젝트에 Migrate시켜주자



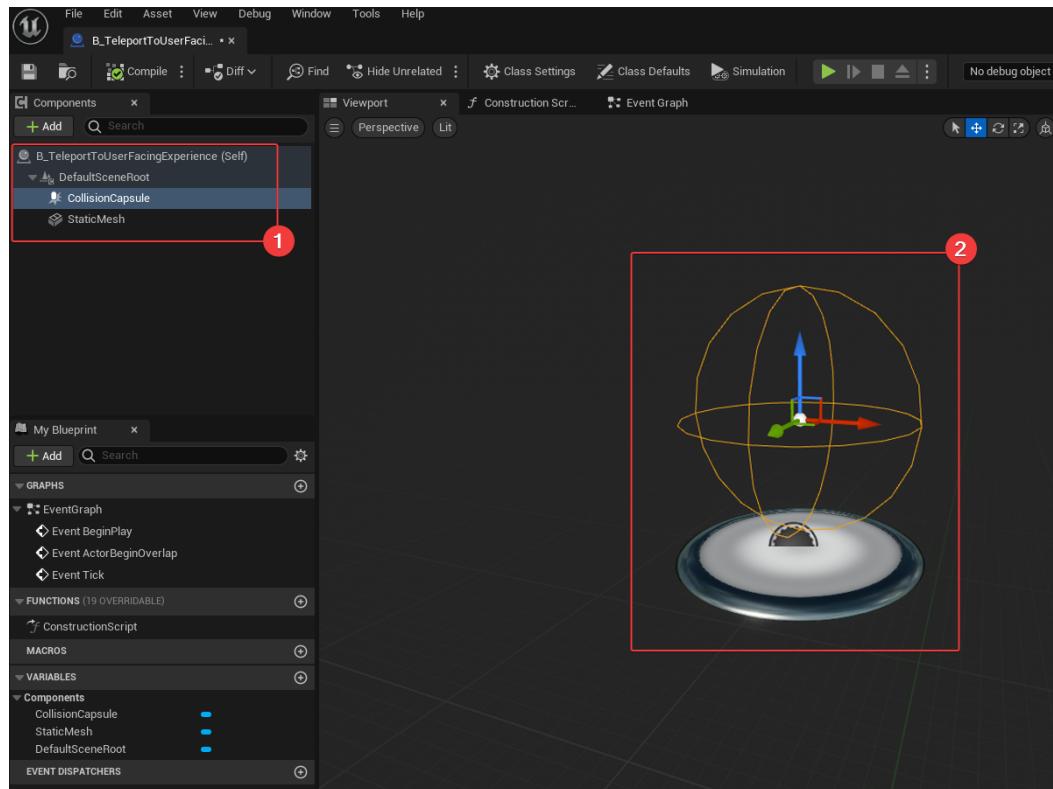
- 이제 `B_TeleportToUserFacingExperience`의 BP를 만들어보자:

- 우선 우리가 체크해야 할 사항은 아래와 같다:

- 무엇을 상속 받고 있는가? **Actor**



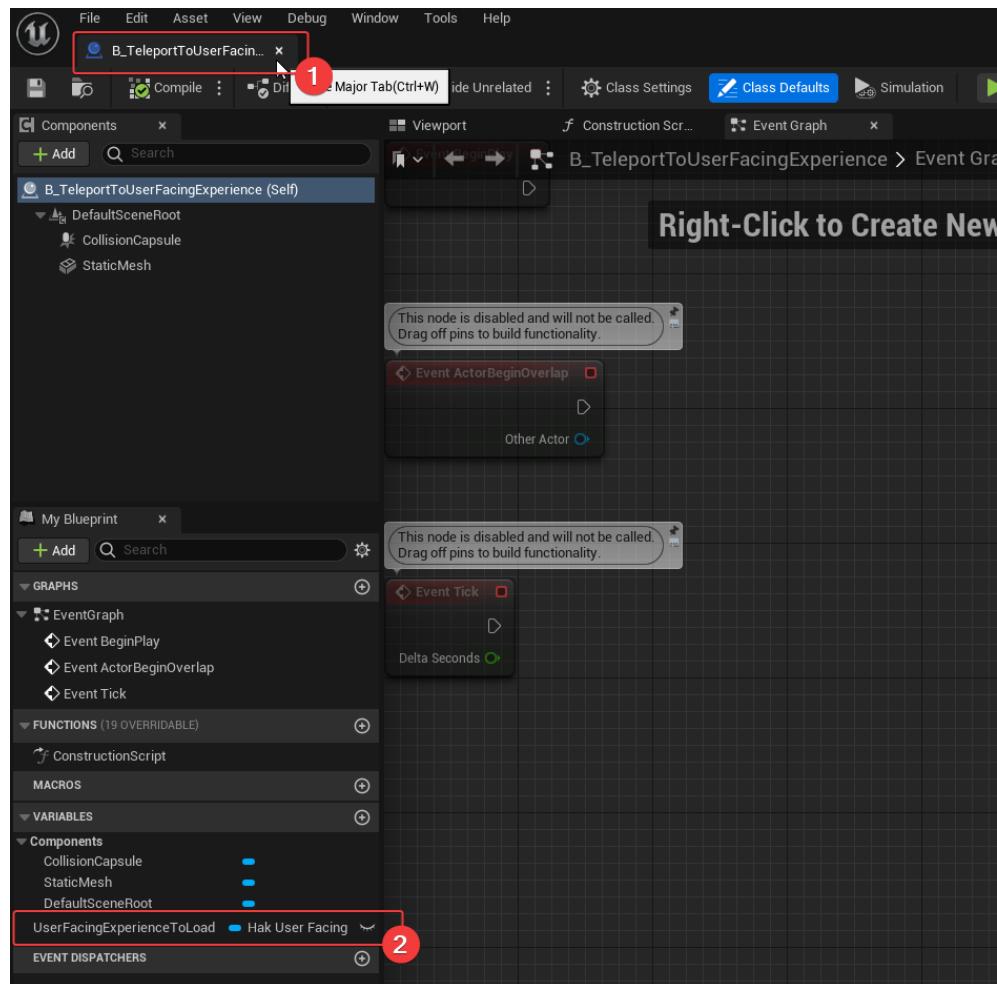
□ BP 생성 후, Viewport를 통해, SCS(Simple Construction Script) 구성 진행:



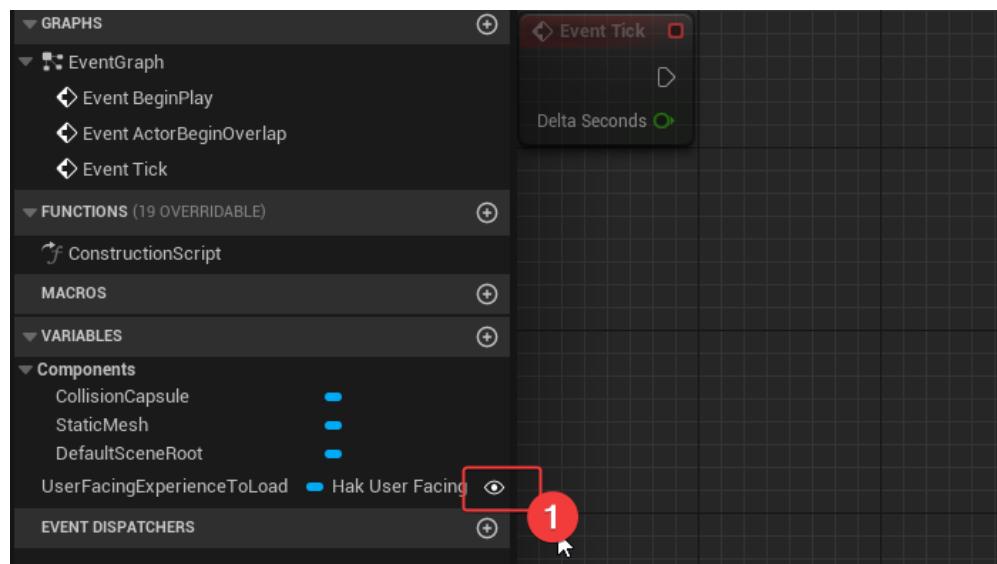
- StaticMesh 추가
- Capcule Collision 추가:
  - Location: **(0, 0, 100)**
  - Shape:
    - Capsule Half Height/Radius: **80.0**

□ 우리는 해당 BP를 만든 이유를 돌이켜보면, Teleport를 하기 위함이다:

- 그러면, Character가 해당 Platform과 충돌이 일어나면, Teleport를 할 대상이 필요한데 이는 UserFacingExperience이다.
- 그럼, 당연히 해당 변수를 추가해줘야겠다:

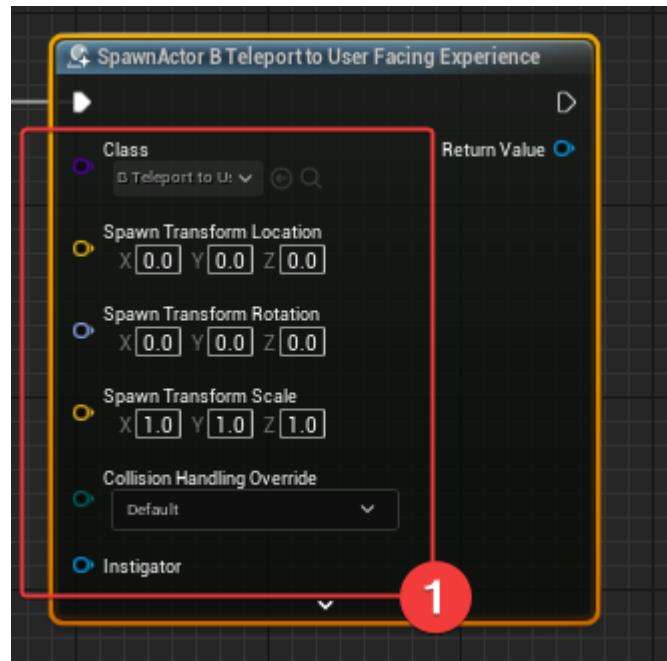


- 또, 당연히 해당 변수를 우리는 다른 BP인 B\_ExperienceList3D에서 참 고해야 하므로, Public화 해주자:

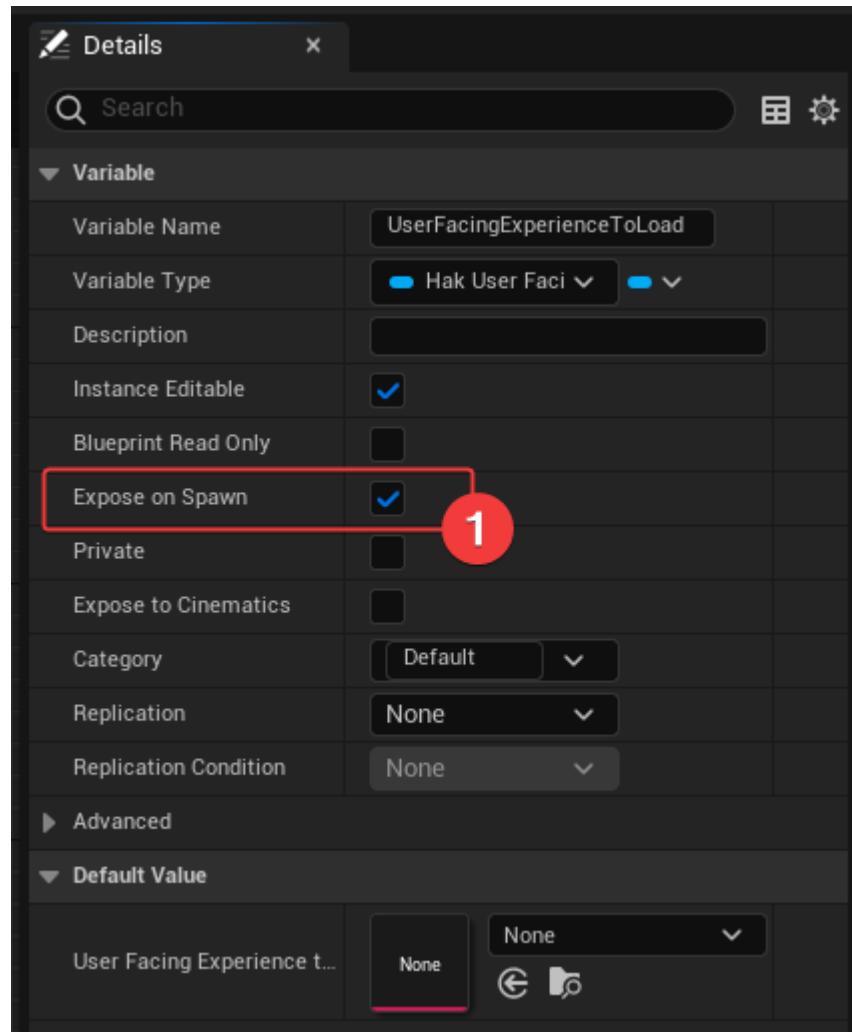


- 이제 이어서, B\_ExperienceList3D의 나머지 BeginPlay 이벤트의 로직을 완성시 켜보자:

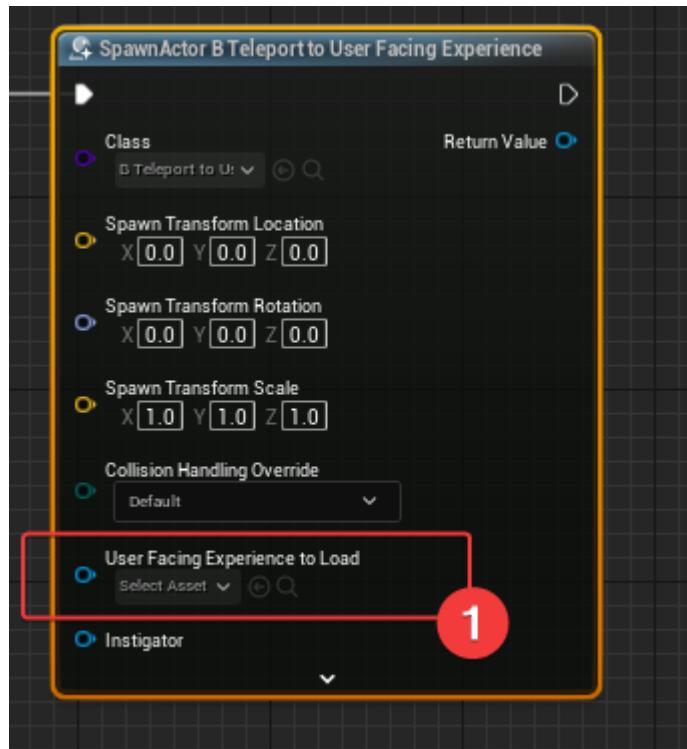
- 어? 근데 SpawnActor에서 앞서 Public으로 선언했던 UserFacingExperienceToLoad가 안보인다:



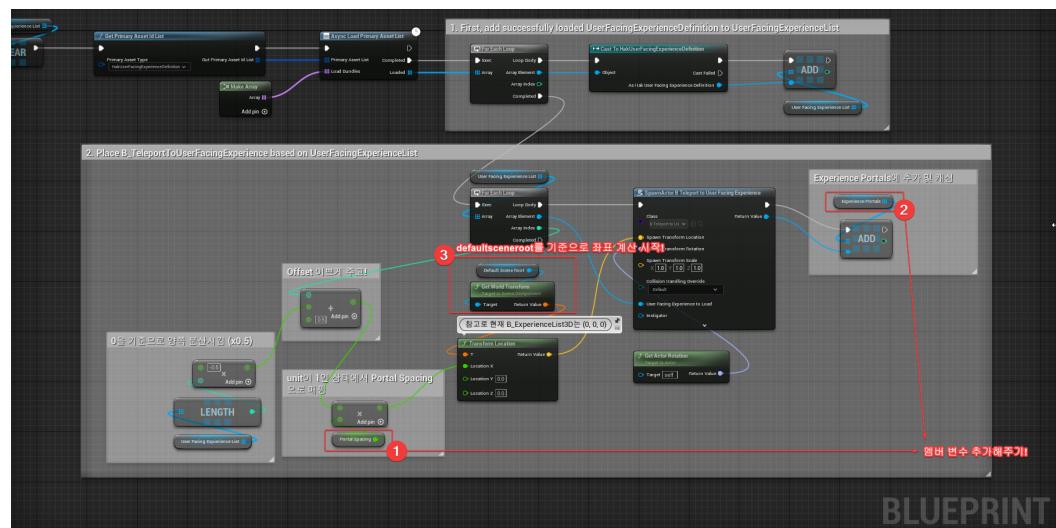
- 이는 아래와 같이 BP Variable의 `Expose on Spawn` 속성이 꺼져 있어 발생:
  - 활성화해주자:



- 이제 보인다:



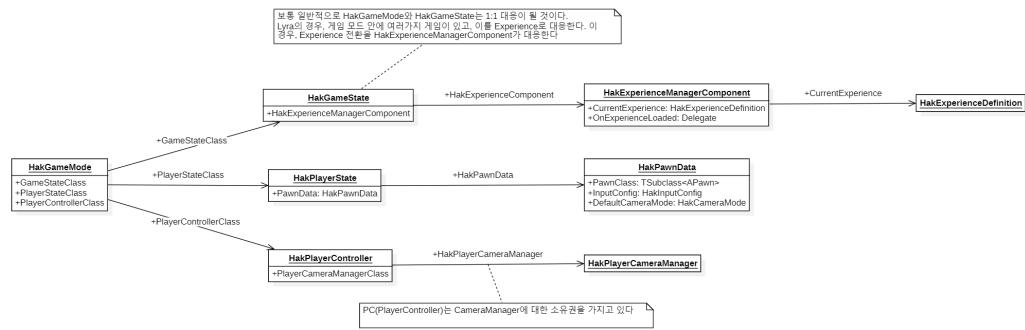
□ B\_TeleportToUserFacingExperience를 배치 및 실행해보자:



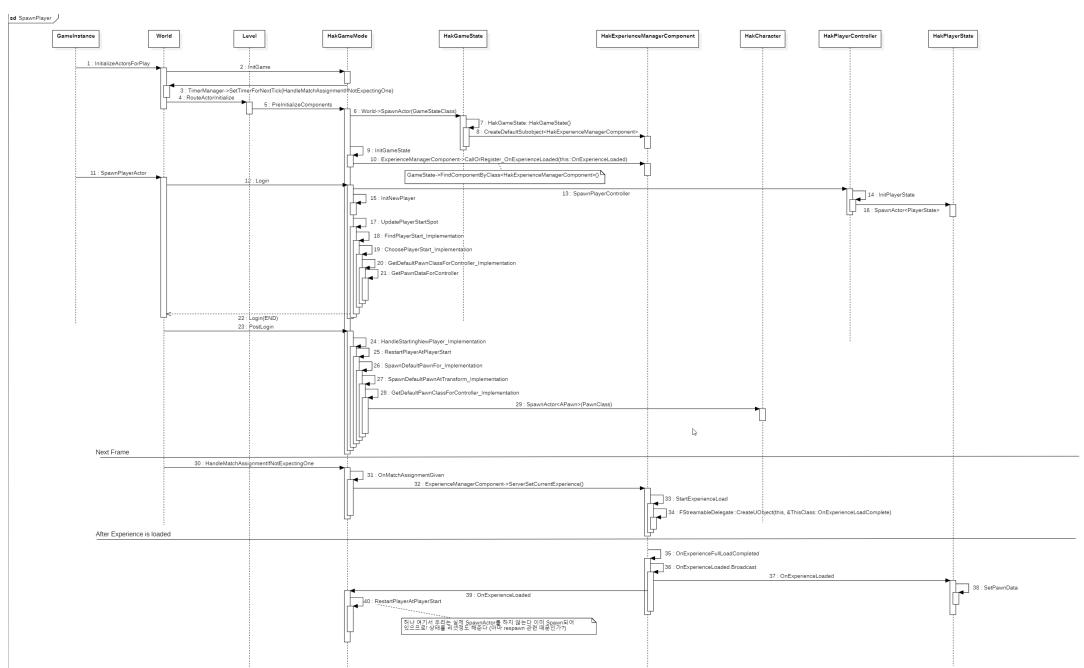
## 캐릭터 Spawn 관련 설명

### ▼ 펼치기

□ 우리가 구현할 클래스들을 살펴보자:



PlayerCharacter가 Spawn되는 과정을 살펴보자:



- 전체적인 순서도이다. 매우 복잡하다!
  - 허나, 하나씩 살펴보면 간단하다:

위의 다이어그램을 좀 더 분석 해보실 분들은 아래를 참고하세요.

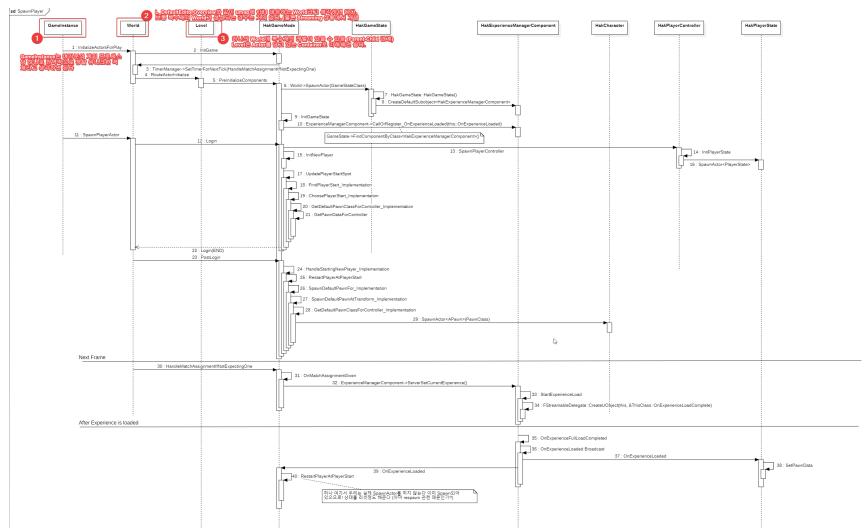
## ▼ 자세히

- #### □ 다양한 관점에서 순서도 분석해보기:

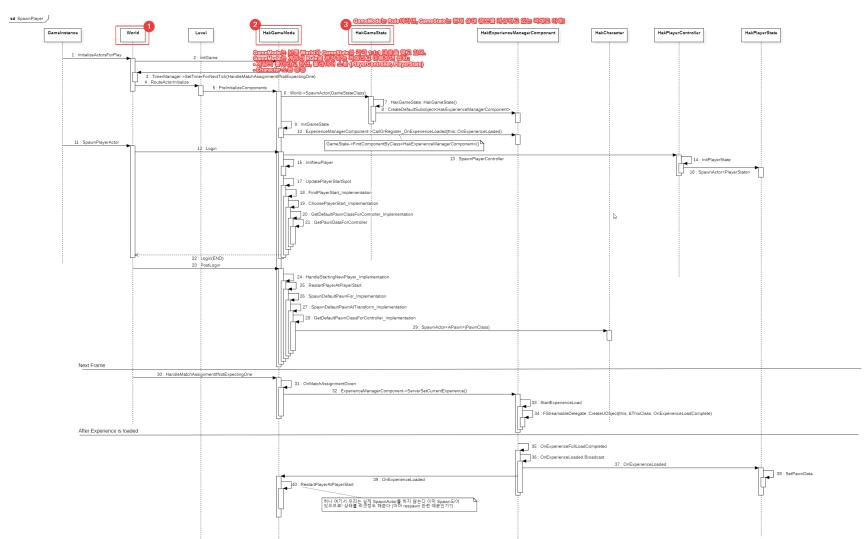
- #### □ 클래스 간의 관계:

- 어떤 클래스가 상위 개념인지 파악해보자:

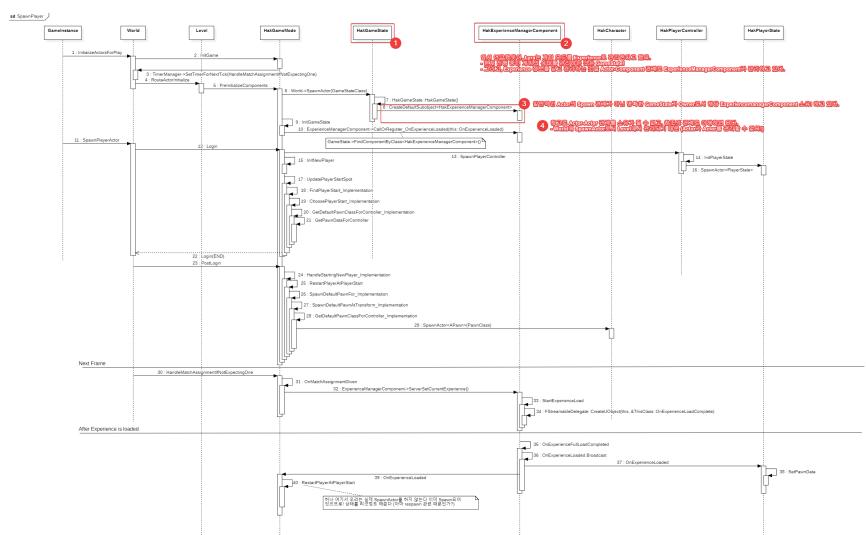
## 1. GameInstance → World → Level 간 관계



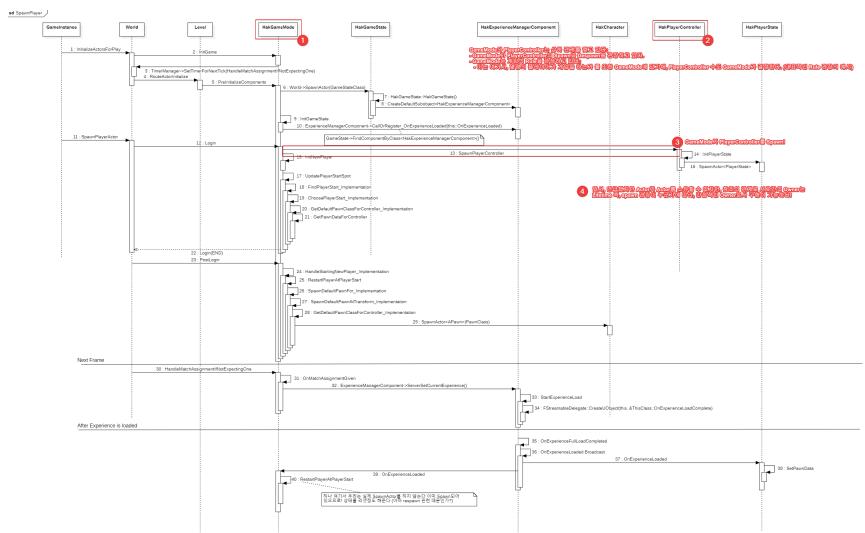
## 2. World → GameMode → GameState 간 관계



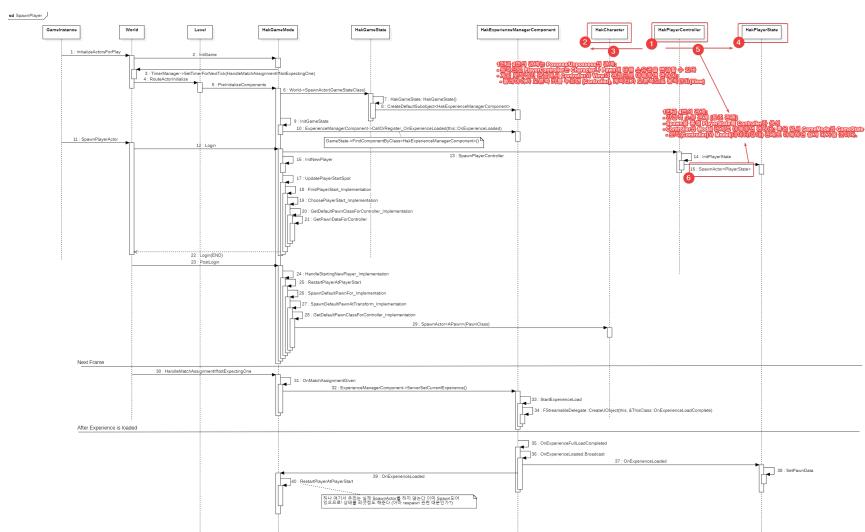
## 3. GameState → ExperienceManagerComponent



#### 4. GameMode → PlayerController

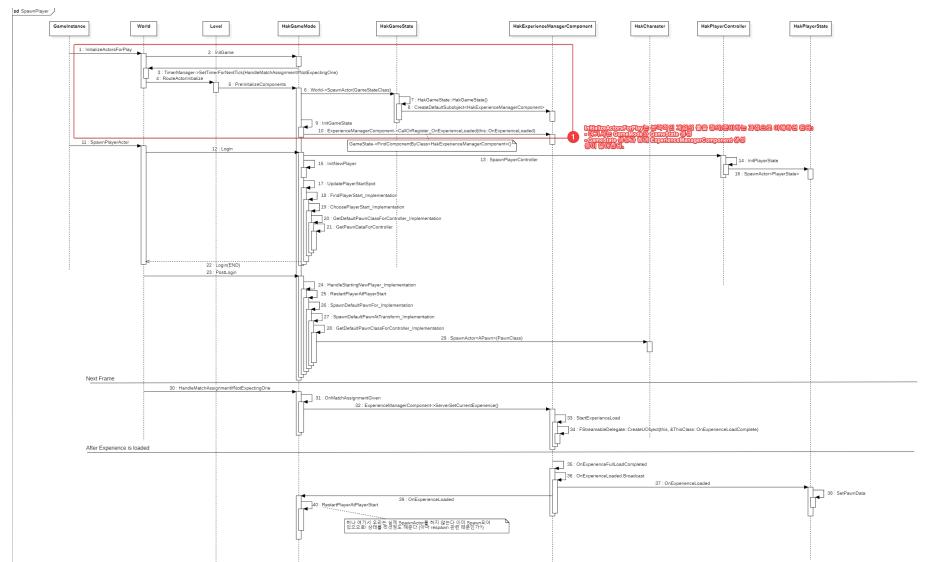


5. (PlayerController → PlayerState),  
(PlayerController → Character[Pawn])



- #### 게임 룰/상태 관점 (Game Mode/State):

- InitializeActorsForPlay

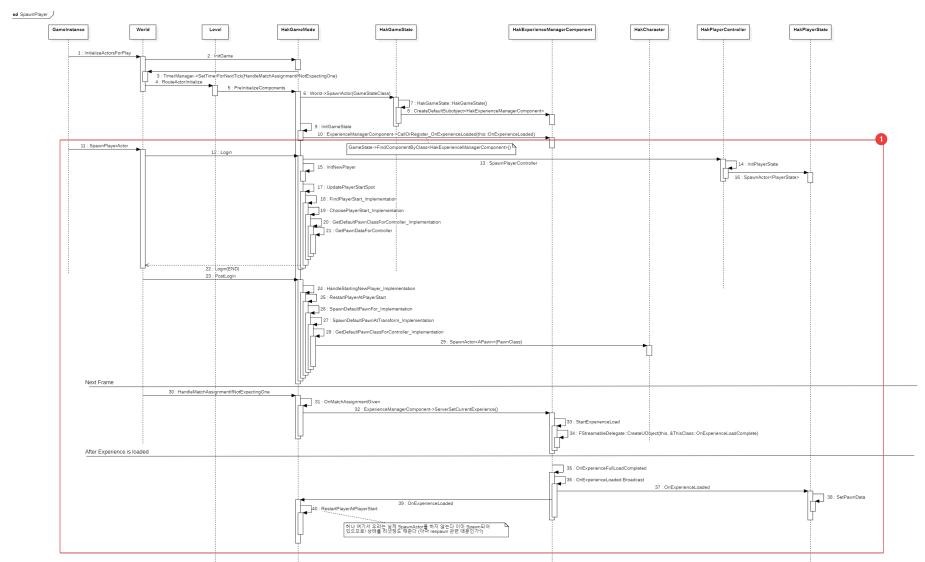


□ 각 과정에 대해 간략하게 브리핑해보자:

- InitGame
- InitGameState

□ 플레이어 관점 (PlayerController, PlayerState, Character[Pawn])

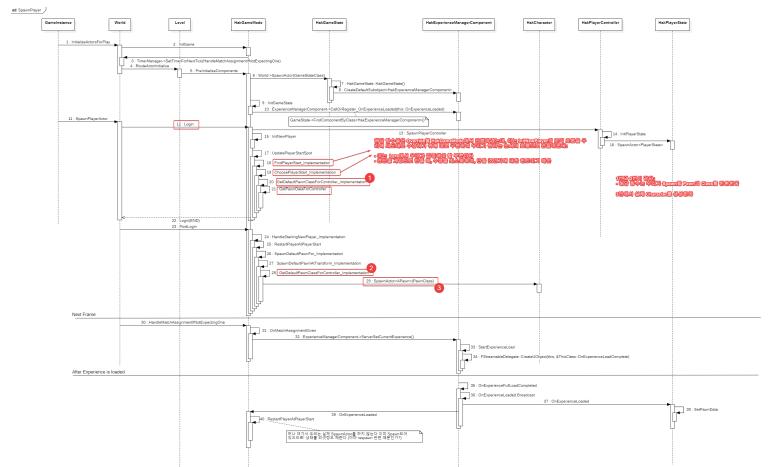
- SpawnPlayActor (오타...)



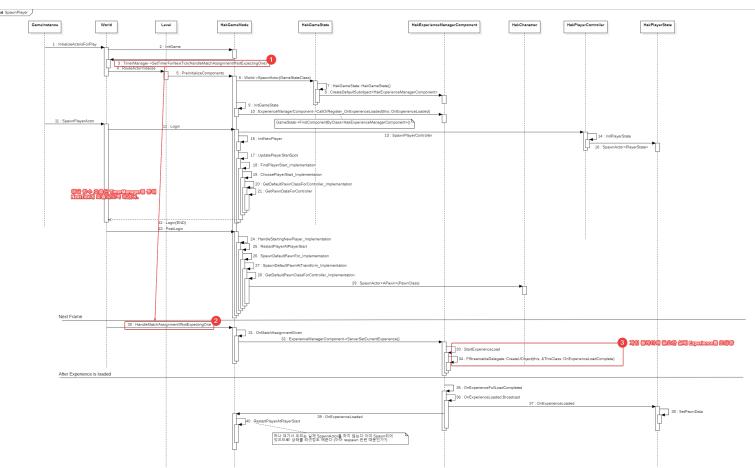
□ 설명하기:

- 해당 과정은 호스트(=현재 직접 조정하고 있는 플레이어)를 생성하는 과정으로 이해하면 된다.
- 해당 과정은 몇 프레임에 걸쳐서 진행된다:

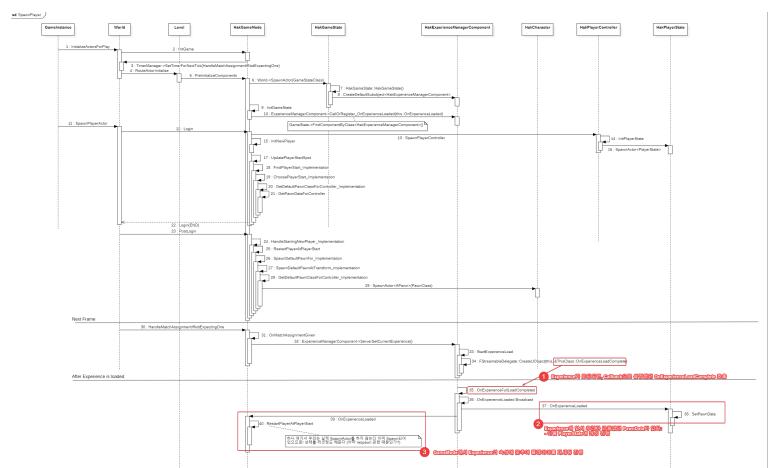
## o Login/PostLogin



## o Next Frame



## o After Experience is loaded



□ QUIZ:

- Lobby에 대한 구현은 어떻게 진행할까?

## 질응답 시간 및 단기적인 목표

### ▼ 펼치기

- 단기적인 목표에 대한 시뮬레이션:
  - 간단한 데모
  - 해당 데모를 만들기 위한 소스코드들 보여주기
- 질문 받기

## 자료:

### ▼ 펼치기

#### StarUML v6

HakUserExperience.mdj