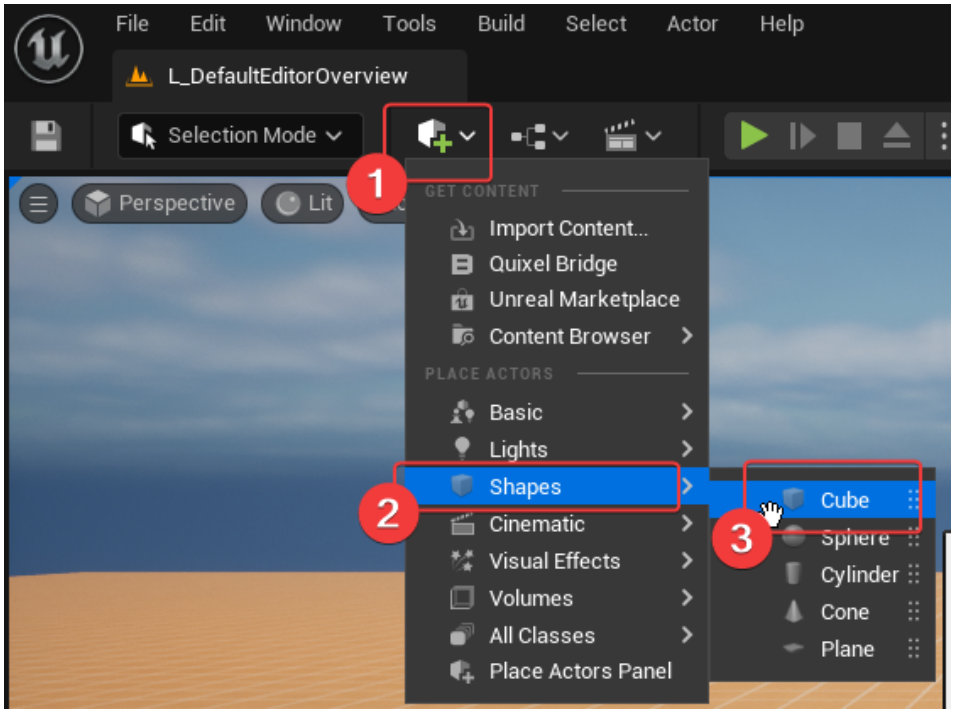
9주차 (2023.11.20)

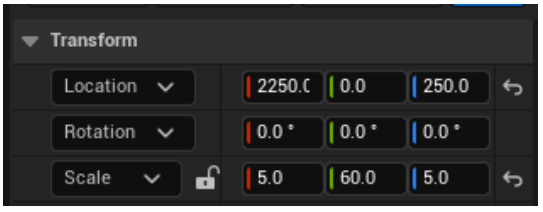
B_TeleportToUserFacingExperience

▼ **펼치기**

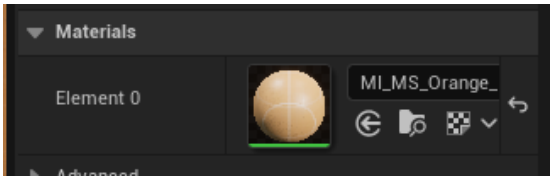
- [ ] 우선 간단한 외벽을 하나를 만들어주자:
  - [ ] Cube Shape를 추가해주자:



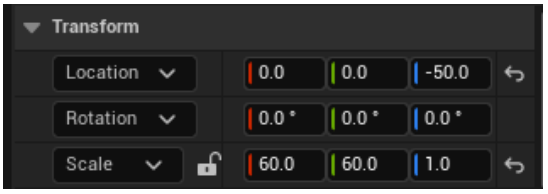
- 해당 Cube 속성값들을 아래와 같이 맞추어주자:



- 그리고 Material도 업데이트해주자:



- [ ] 간단히 이에 맞추어 Platform Cube도 Scale을 재조정해주자:

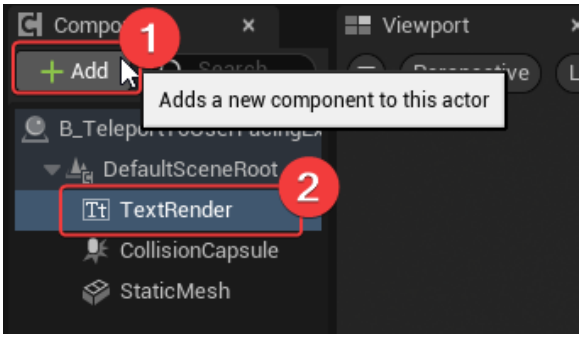


[B_TeleportToUserFacingExperience]

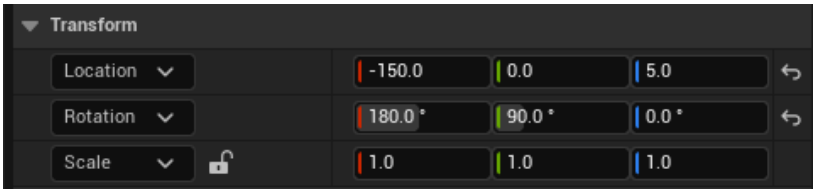
현재 해당 BP의 우리의 구현은 미-구현 상태이다:

- 앞서, 우리의 기억을 회고해보면, 캐릭터가 해당 BP에 설정된 Capsule Collision에 Overlap이 일어나면 알맞는 Experience로서 Teleport 하는 것을 기대한다

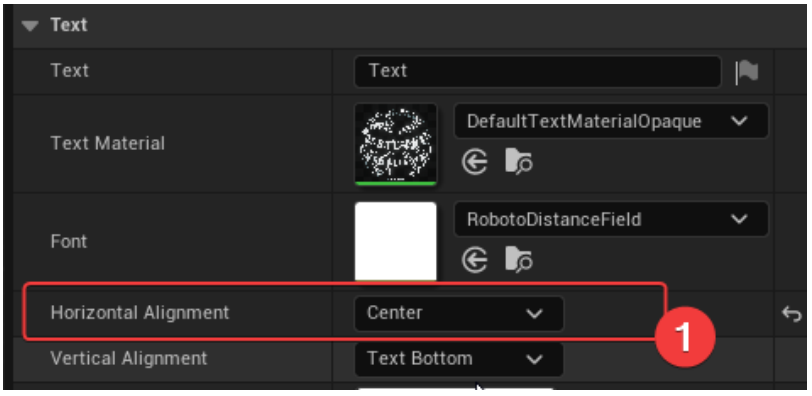
- [ ] 우선 해당 Platform 메시에 대해 어떤 Experience가 엮여 있는지 간단히 TextRender로 만들자



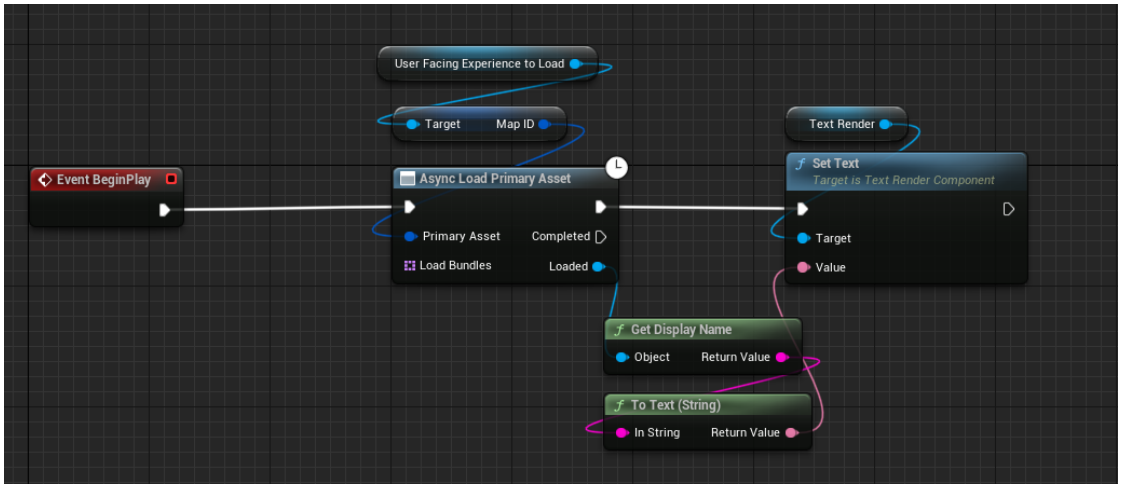
- [ ] 아래와 같이 Transform을 설정해주자:



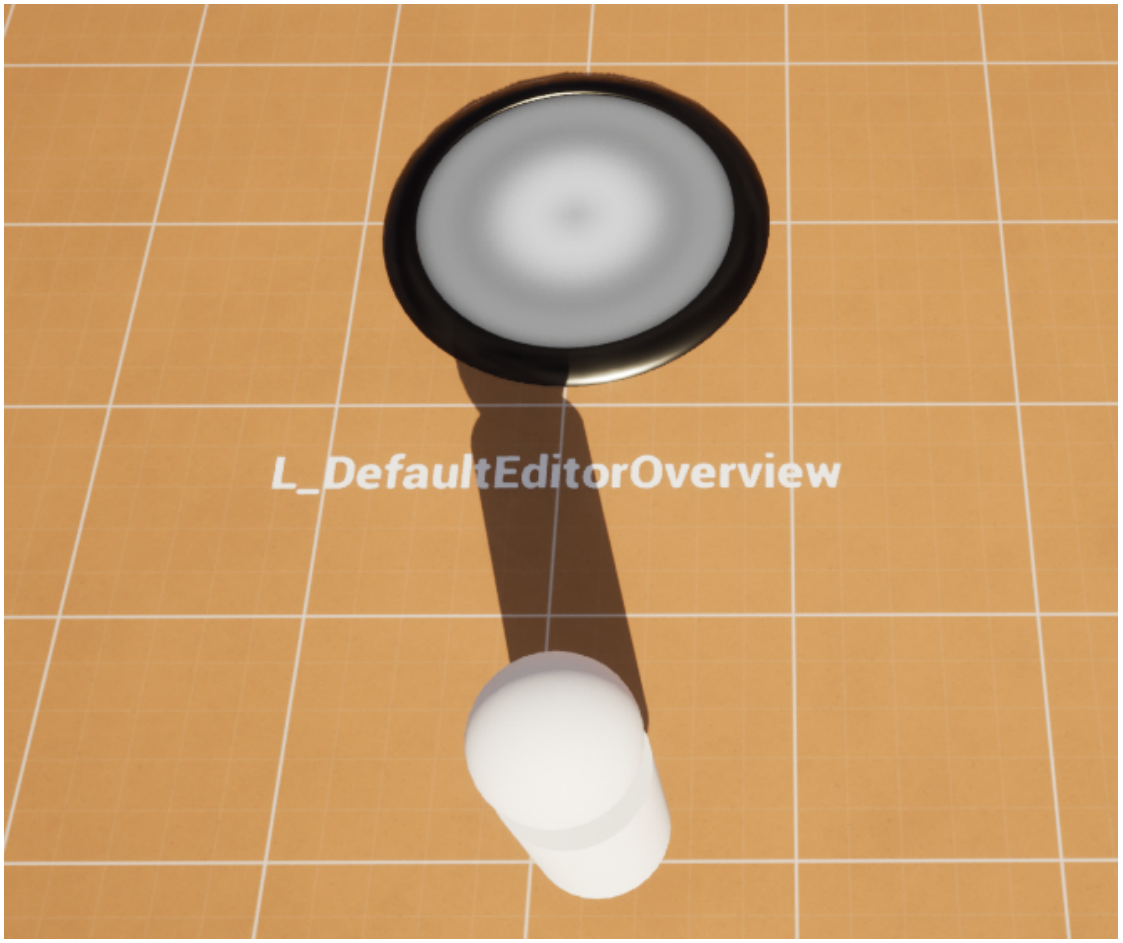
- [ ] Text Alignment도 Center로 변경하자:



- [ ] 그럼 TextRender의 Text를 BeginPlay에서 변경해주자:



☐ 그럼 아래와 같이 손쉽게 나온다:

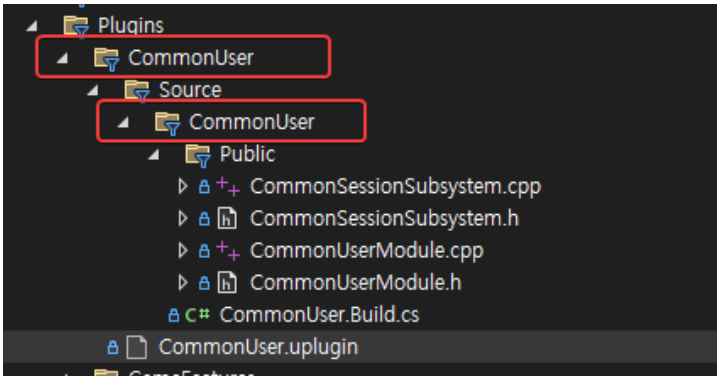


- Event ActorBeginOverlap을 BP에서 구현하기에 앞서, **Map Transition 관련 기능을 제공하는 CommonUser Plugin을 구현**해야 한다

CommonUser

▼ **펼치기**

- [ ] GenerateProjectFiles.bat를 활용하여, CommonUser Plugin의 윤곽을 잡아주자:



  - 위의 CommonUser 폴더를 포함하여 전체적인 계층적 구조를 유의해서 잡아주자

- [ ] CommonUser.uplugin

- [ ] CommonUser.Build.cs

CommonUser.Build.cs 1

D: > HakUnrealEngine > Games > Hak > Plugins > CommonUser > Source > CommonUser > CommonUser.Build.cs

```cs
using UnrealBuildTool;

public class CommonUser : ModuleRules
{
    public CommonUser(ReadOnlyTargetRules Target) : base(Target)
    {
        PCHUsage = ModuleRules.PCHUsageMode.UseExplicitOrSharedPCHs;

        PublicDependencyModuleNames.AddRange(
            new string[]
            {
                "Core",
                "CoreUObject",
                "Engine",
                // ... add other public dependencies that you statically link with here ...
            }
        );
    }
}
```

☐ Public/CommonUserModule[.h/.cpp]

CommonUserModule.h

D: > HakUnrealEngine > Games > Hak > Plugins > CommonUser > Source > CommonUser > Public > CommonUserModule.h

```h
#pragma once

#include "Modules/ModuleInterface.h"

class FCommonUserModule : public IModuleInterface
{
public:
    virtual void StartupModule() override;
    virtual void ShutdownModule() override;
};
```

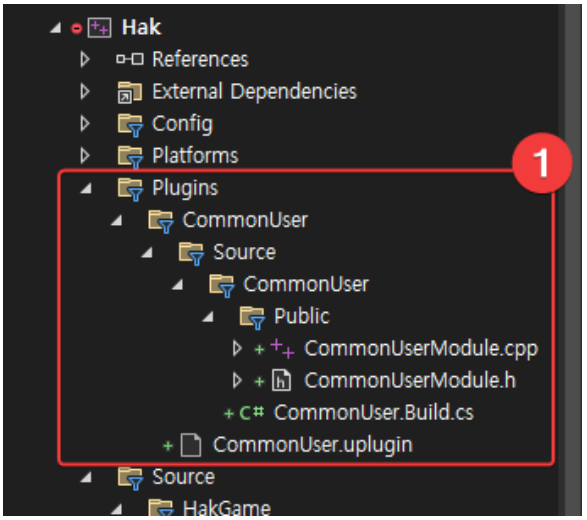
CommonUserModule.cpp

D: > HakUnrealEngine > Games > Hak > Plugins > CommonUser > Source > CommonUser > Public > CommonUserModule.cpp

```cpp
#include "CommonUserModule.h"
#include "Modules/ModuleManager.h"

void FCommonUserModule::StartupModule()
{
}

void FCommonUserModule::ShutdownModule()
{
}

IMPLEMENT_MODULE(FCommonUserModule, CommonUser);
```

💡 Plugin을 생성할 경우, 위의 **세가지 요소를 정의하지 않으면**, **GenerateProjectFiles.bat에서 실패**하니, 폴더 구조와 파일 내용을 꼼꼼히 살피자.

☐ 아래와 같이 VS에 잘 추가되면 성공이다:



- [ ] 이제 CommonSessionSubsystem을 구현하자:

[간단한 설명]

`CommonSessionSubsystem`은 **Experience에 속해있는 맵 로딩과 ExperienceManagerComponent에 Experience 변경에 대한 CmdArgs를 전달하는 역할을 담당**한다.

- [ ] CommonSessionSubsystem.h/.cpp를 추가하자:

```cpp
#pragma once

#include "CoreMinimal.h"
#include "Subsystems/GameInstanceSubsystem.h"
#include "CommonSessionSubsystem.generated.h"

UCLASS()
class COMMONUSER_API UCommonSessionSubsystem : public UGameInstanceSubsystem
{
    GENERATED_BODY()
public:
    UCommonSessionSubsystem() {}

    /**
     * member variables
     */
    /** PendingTravelURL은 흔히 맵의 경로로 생각하면 된다 */
    FString PendingTravelURL;
};
```

```cpp
#include "CommonSessionSubsystem.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(CommonSessionSubsystem)
```

- [ ] HostSession을 구현하자:
  - [ ] 간단히 Backbone을 구현하자:

```h
#pragma once

#include "CoreMinimal.h"
#include "Subsystems/GameInstanceSubsystem.h"
#include "CommonSessionSubsystem.generated.h"

/** forward declaration */
class APlayerController;

/**
 * UCommonSessionSubsystem은 HakGame에서 사용되야 하기 때문에, Module Export를 해줘야하고 그래서 COMMONUSER_API를 추가해줘야 한다!
 * - 여러분들이 {ModuleName}_API의 추가는 ***다른 모듈에서 사용할 경우, 추가해주면 된다***
 */
UCLASS()
class COMMONUSER_API UCommonSessionSubsystem : public UGameInstanceSubsystem
{
    GENERATED_BODY()
public:
    UCommonSessionSubsystem() {}

    UFUNCTION(BlueprintCallable, Category=Session)
    void HostSession(APlayerController* HostingPlayer, UCommonSession_HostSessionRequest* Request);

    /**
     * member variables
     */
    /** PendingTravelURL은 흔히 맵의 경로로 생각하면 된다 */
    FString PendingTravelURL;
};
```

1

2

```cpp
#include "CommonSessionSubsystem.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(CommonSessionSubsystem)

void UCommonSessionSubsystem::HostSession(APlayerController* HostingPlayer, UCommonSession_HostSessionRequest* Request)
{

}
```

1

☐ UCommonSession_HostSessionRequest를 구현하자:

```cpp
void UCommonSessionSubsystem::HostSession(APlayerController* HostingPlayer, UCommonSession_HostSessionRequest* Request)
{
    ULocalPlayer* LocalPlayer = (HostingPlayer != nullptr) ? HostingPlayer->GetLocalPlayer() : nullptr;
    if (!LocalPlayer)
    {
        return;
    }

    // HostSessionRequest에서 MapID와 ExtraArgs를 통해 URL을 생성하여, MapLoad를 시작한다
    GetWorld()->ServerTravel(Request->ConstructTravelURL());
}
```

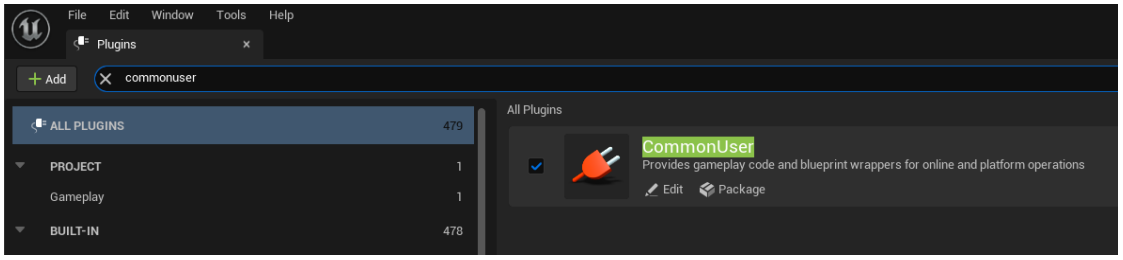
☐ ConstructTravelURL() 구현:

```cpp
FString UCommonSession_HostSessionRequest::ConstructTravelURL() const
{
    FString CombinedExtraArgs;

    for (const auto& ExtraArg : ExtraArgs)
    {
        if (ExtraArg.Key.IsEmpty())
        {
            continue;
        }

        /**
         * ?를 separate로 복수개의 ExtraArg를 추가함:
         * - Key 값 유무에 따라, =(assignment)를 통해 알맞는 CmdArg를 생성
         */
        if (ExtraArg.Value.IsEmpty())
        {
            CombinedExtraArgs += FString::Printf(TEXT("?%s"), *ExtraArg.Key);
        }
        else
        {
            CombinedExtraArgs += FString::Printf(TEXT("?%s=%s"), *ExtraArg.Key, *ExtraArg.Value);
        }
    }

    // Map 경로 앞에 추가하여, 최종 TravelURL 생성
    return FString::Printf(TEXT("%s%s"), *GetMapName(), *CombinedExtraArgs);
}
```

- [ ] GetMapName() 구현:

```cpp
#include "CommonSessionSubsystem.h"
#include "Engine/AssetManager.h"
#include "Engine/LocalPlayer.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(CommonSessionSubsystem)

FString UCommonSession_HostSessionRequest::GetMapName() const
{
    // AssetManager를 통해, MapID를 통해, FAssetData를 반환하고, 이를 통해 PackageName를 통해 Map 경로를 반환한다
    FAssetData MapAssetData;
    if (UAssetManager::Get().GetPrimaryAssetData(MapID, MapAssetData))
    {
        return MapAssetData.PackageName.ToString();
    }
    return FString();
}
```

- [ ] CommonUser의 Plugin을 HakGame에 활성화해주자:



- [ ] HakGame.Build.cs에 CommonUser를 추가하자:

```cpp
using UnrealBuildTool;

1 reference
public class HakGame : ModuleRules
{
    0 references
    public HakGame(ReadOnlyTargetRules Target) : base(Target)
    {
        PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;

        PublicDependencyModuleNames.AddRange(new string[] {
            "Core",
            "CoreUObject",
            "Engine",
            "InputCore",
            // GAS
            "GameplayTags",
            // Game Features
            "ModularGameplay",
            "GameFeatures",
            // Input
            "InputCore",
            "EnhancedInput",
            // CommonUser
            "CommonUser",
        });

        PrivateDependencyModuleNames.AddRange(new string[] {  });

        // Uncomment if you are using Slate UI
        // PrivateDependencyModuleNames.AddRange(new string[] { "Slate", "SlateCore" });

        // Uncomment if you are using online features
        // PrivateDependencyModuleNames.Add("OnlineSubsystem");

        // To include OnlineSubsystemSteam, add it to the plugins section in your uproject file with the Enabled attribute set to true
    }
}
```

- [ ] HakUserFacingExperienceDefinition에 CreateHostingRequest()를 구현하자:

```h
#pragma once

#include "CoreMinimal.h"
#include "Engine/DataAsset.h"
#include "HakUserFacingExperienceDefinition.generated.h"

/** forward declarations */
class UCommonSession_HostSessionRequest;

/**
 * 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:
    /**
     * Map 로딩 및 Experience 전환을 위해, MapID와 ExperienceID를 활용하여, HostSessionRequest 생성
     */
    UCommonSession_HostSessionRequest* CreateHostingRequest() const;

    /**
     * member variables
     */

    /** the specific map to load */
    UPROPERTY(BlueprintReadWrite, EditAnywhere, Category=Experience, meta=(AllowedTypes="Map"))
    FPrimaryAssetId MapID;

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

1

2

```cpp
#include "HakGame/GameModes/HakUserFacingExperienceDefinition.h"
#include "CommonSessionSubsystem.h"

// 항상 generated.h 파일이 헤더에 포함되어 있다면, gen.cpp도 포함시키도록 한다!
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakUserFacingExperienceDefinition)

UCommonSession_HostSessionRequest* UHakUserFacingExperienceDefinition::CreateHostingRequest() const
{
    const FString ExperienceName = ExperienceID.PrimaryAssetName.ToString();

    // 잠깐 한걸음 깊이 생각해보기:
    // - UCommonSession_HostSessionRequest는 UObject로 생성해놓고, 알아서 GC가 된다:
    // - 해당 객체는 현재 프레임에서 사용하기 때문에, GC에 대한 염려가 필요없다: 만약 다음 프레임이든 추가적인 프레임 상에서 해당 객체를 사용할 경우, Lifetime 관리 필요!
    //   - 그렇지 않으면 dangling 난다!
    UCommonSession_HostSessionRequest* Result = NewObject<UCommonSession_HostSessionRequest>();
    Result->MapID = MapID;
    Result->ExtraArgs.Add(TEXT("Experience"), ExperienceName);

    return Result;
}
```

자, 이제 BP를 완성할 준비가 되었다! Experience를 통한 맵간 이동을 진행해보자.

B_TeleportToUserFacingExperience

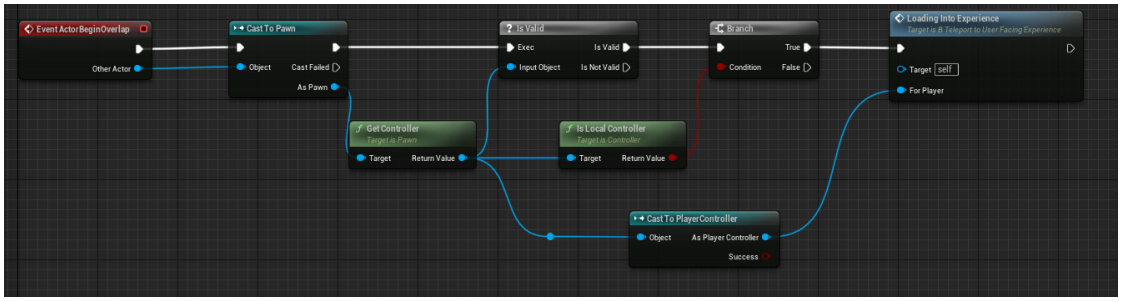
▼ **펼치기**

☐ CreateHostingRequest() 함수를 BlueprintCallable로 정의:

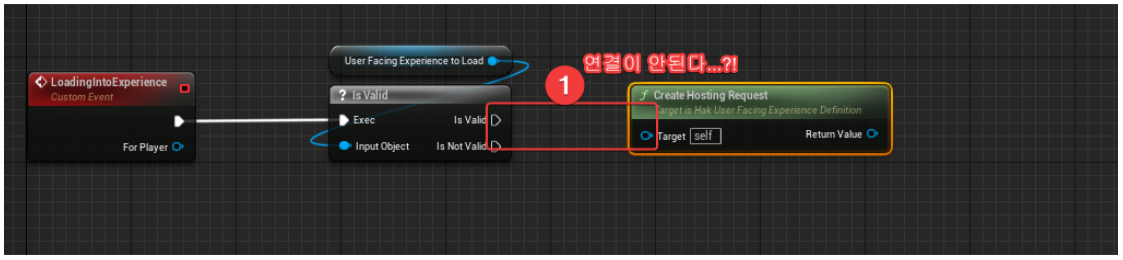
```h
/**
 * UHakUserFacingExperienceDefinition
 * - description of settings used to display experiences in the UI and start a new session
 */
UCLASS(BlueprintType)
class HAKGAME_API UHakUserFacingExperienceDefinition : public UPrimaryDataAsset
{
    GENERATED_BODY()
public:
    /**
     * Map 로딩 및 Experience 전환을 위해, MapID와 ExperienceID를 활용하여, HostSessionRequest 생성
     */
    UFUNCTION(BlueprintCallable)
    UCommonSession_HostSessionRequest* CreateHostingRequest() const;
```

1

☐ Event Actor Begin Overlap 구현:



☐ LoadingIntoExperience 구현:



☐ 앞서, CreateHostingRequest()에 추가적인 BP 옵션을 넣어줘야 한다:

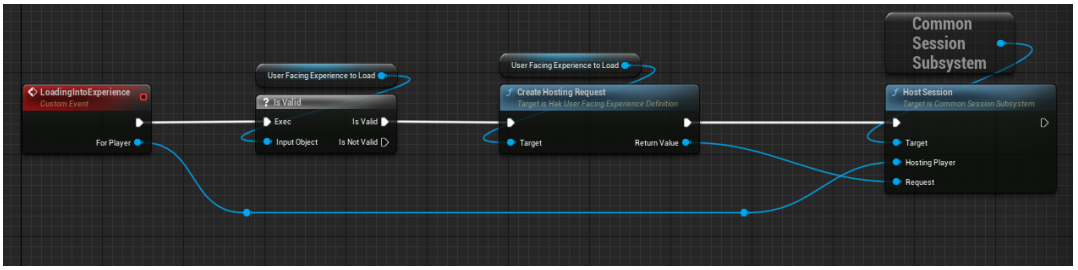
`BlueprintPure`	The function does not affect the owning object in any way and can be executed in a Blueprint or Level Blueprint graph.

- 간단히, 이야기하면, **Owning Object 상관없이 BP에서 함수 호출이 아닌 하나의 Node 단위 실행이 가능한지 여부**라고 생각하면 된다.

☐ 아래와 같이 추가해주자:

```
/**
 * 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:
    /**
     * Map 로딩 및 Experience 전환을 위해, MapID와 ExperienceID를 활용하여, HostSessionRequest 생성
     */
    UFUNCTION(BlueprintCallable, BlueprintPure=false)  1
    UCommonSession_HostSessionRequest* CreateHostingRequest() const;
```

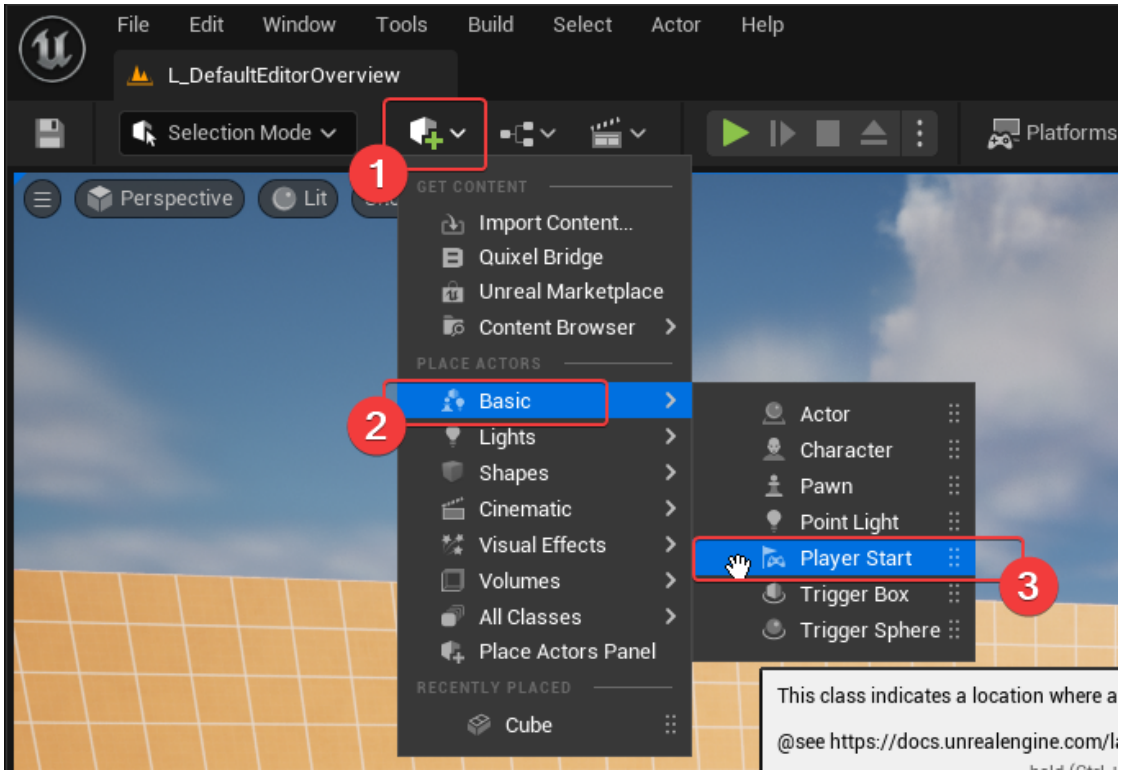
☐ 이어서, 구현하면 아래와 같다:



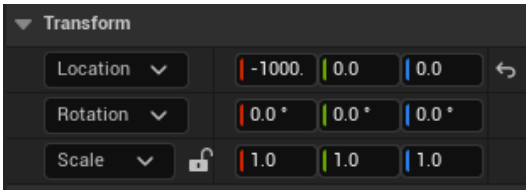
☐ 실행해보면, 뭔가 우리의 의도대로 원래 처음 Spawn Location에서 시작되는거 같지 않다…

- 이는 우리가 **PlayerStart를 생성하지 않았기 때문**이다

☐ PlayerStart를 추가하자:



☐ 아래와 같이 Transform 속성을 입력하자:

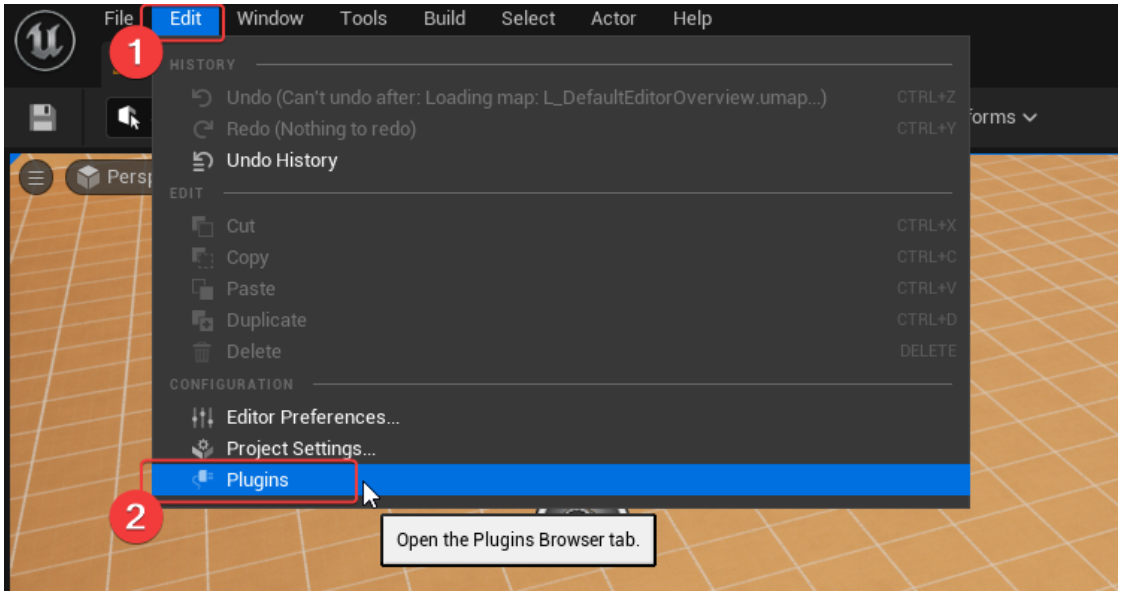


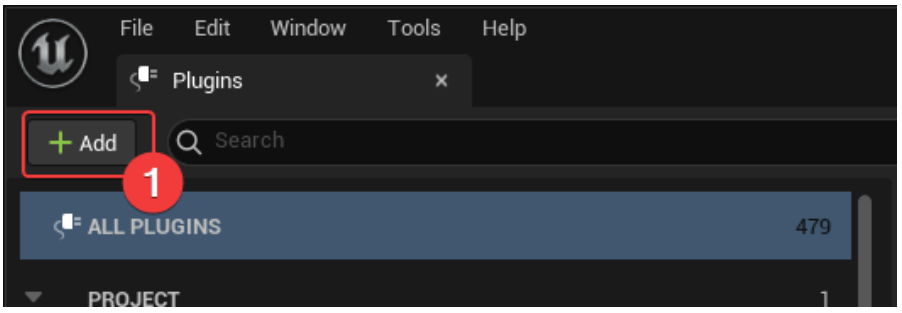
ShooterCore

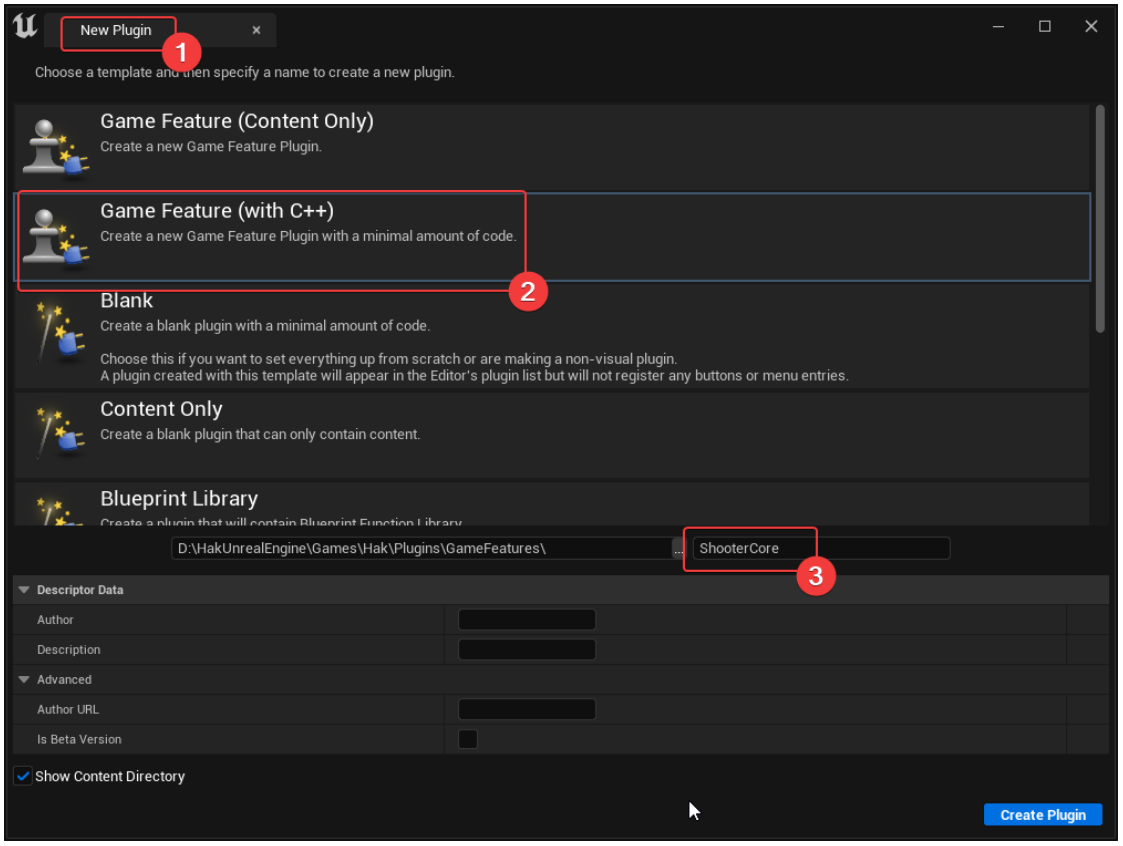
▼ **펼치기**

- 아래의 목표로 진행해보자:
  - 우선, 기존 L_DefatuleEditorOverview와 B_HakDefaultExperience와 형태를 같게 Map을 구성하고, 비슷하게 Experience로 만들어보자

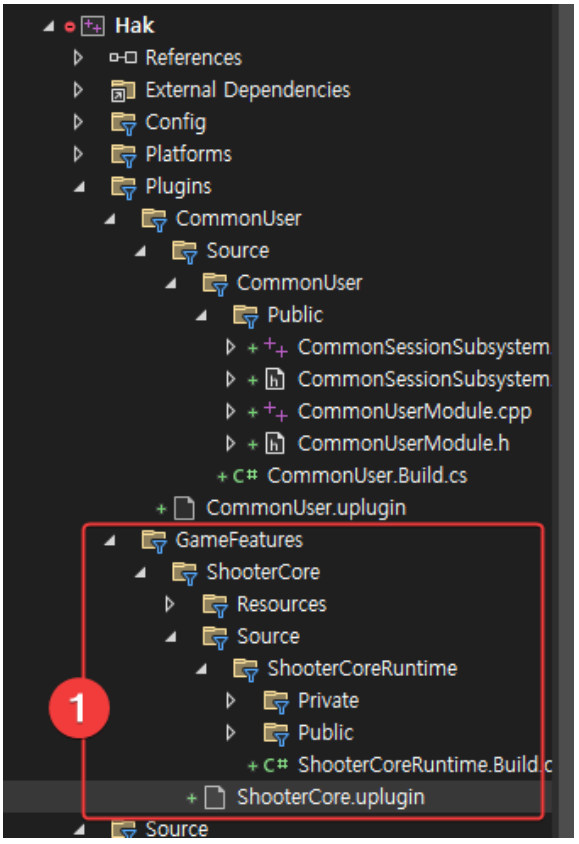
☐ 아래와 같이 GameFeature를 ShooterCore로 추가해주자:



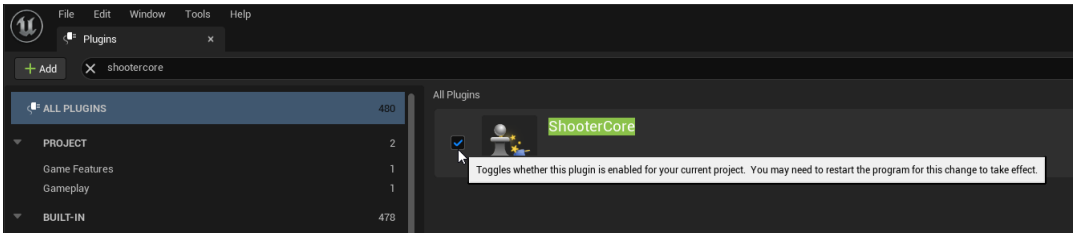




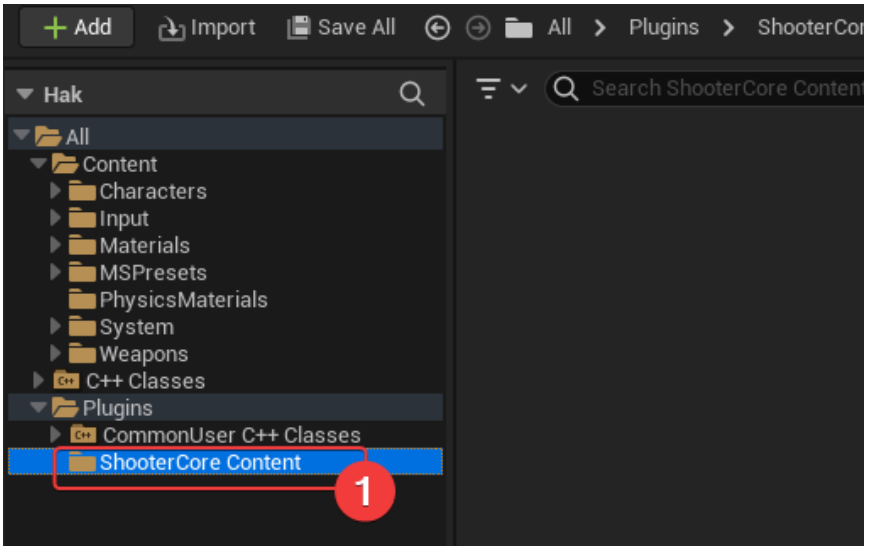
- 아래와 같이 ShoterCore가 추가되었음을 알 수 있다:



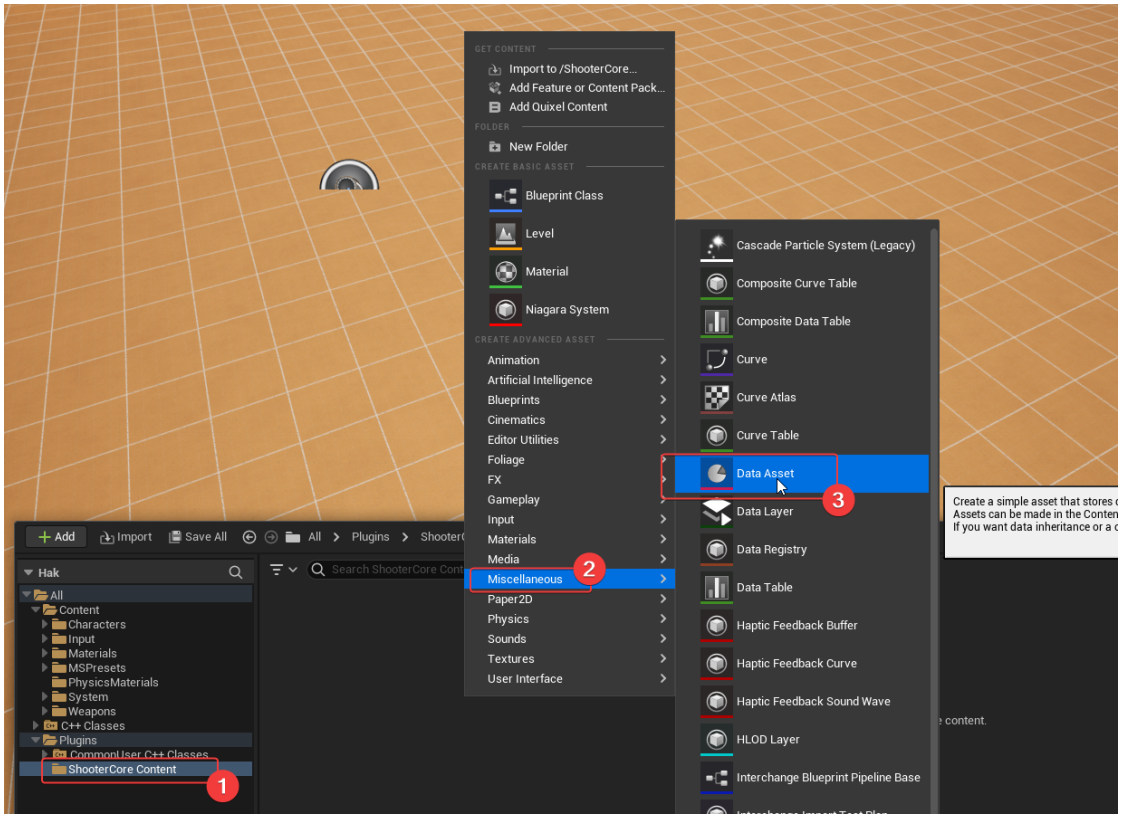
☐ 우선 해당 GameFeature를 활성화해주자:

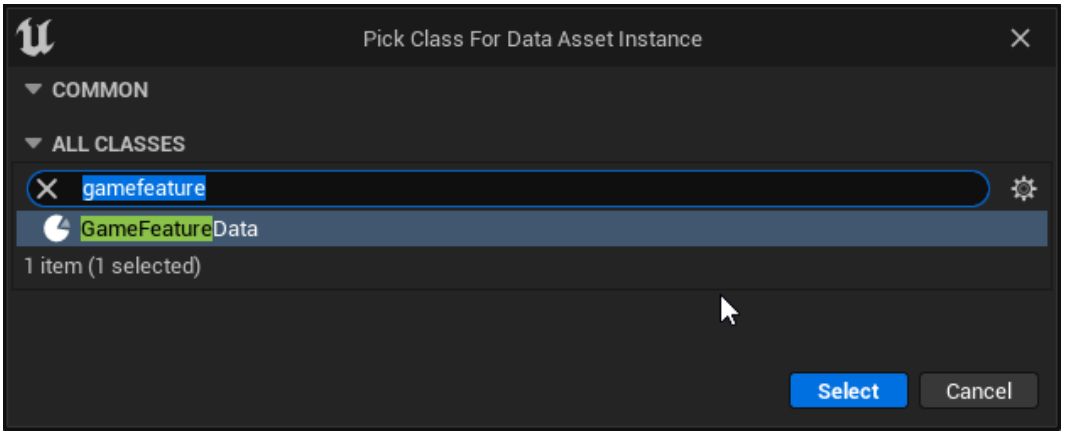


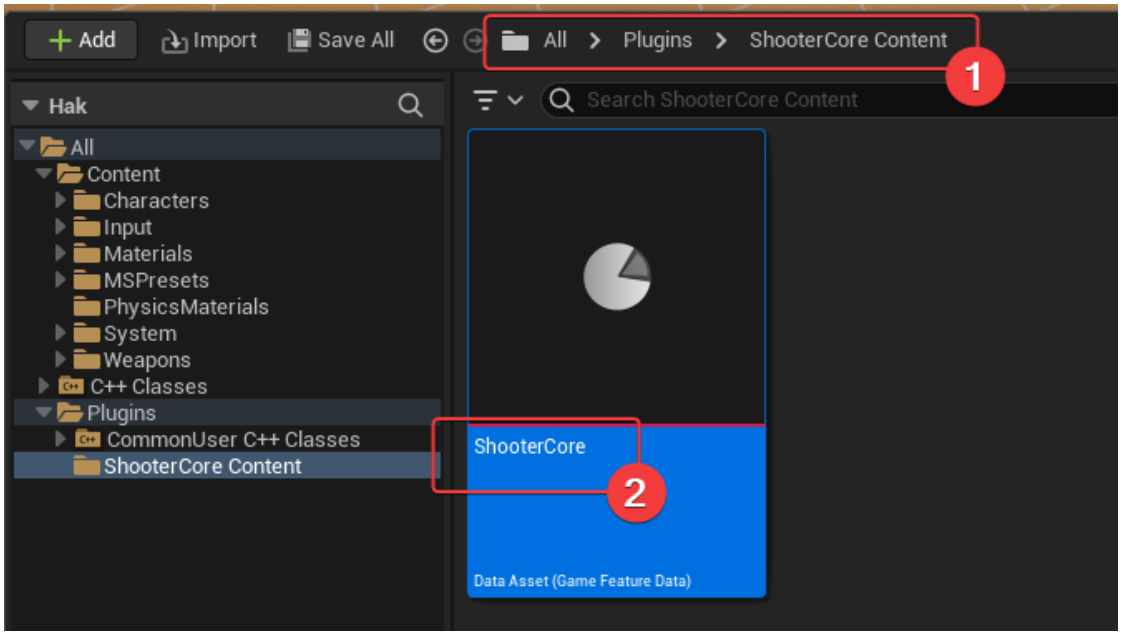
- 아래와 같이 제대로 활성화되었다면, 보인다:



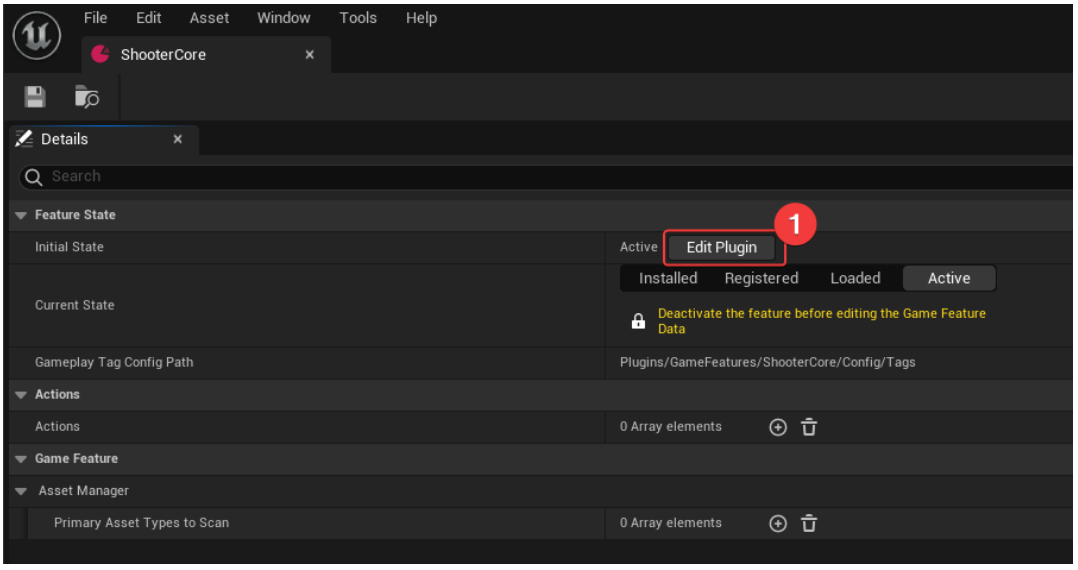
☐ GameFeatureData를 추가해주자:

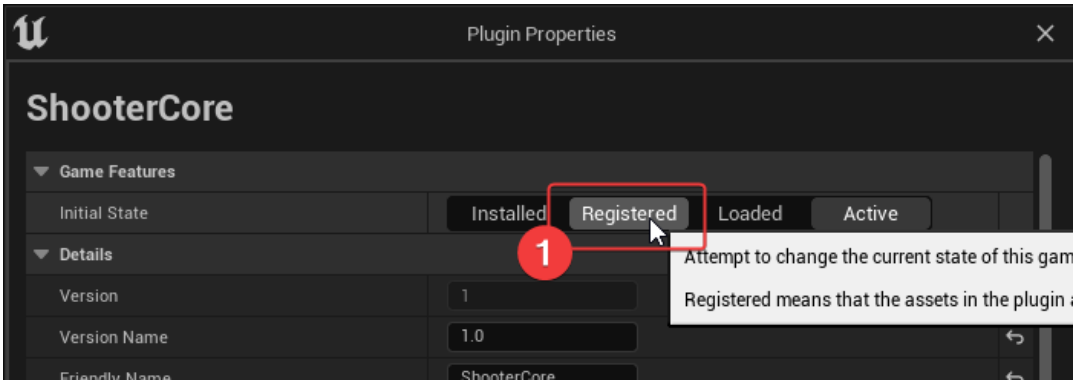

GET CONTENT
Import to /ShooterCore...
Add Feature or Content Pack...
Add Quixel Content
FOLDER
New Folder
CREATE BASIC ASSET
Blueprint Class
Level
Material
Niagara System
CREATE ADVANCED ASSET
Animation
Artificial Intelligence
Blueprints
Cinematics
Editor Utilities
Foliage
FX
Gameplay
Input
Materials
Media
Miscellaneous
Paper2D
Physics
Sounds
Textures
User Interface
Cascade Particle System (Legacy)
Composite Curve Table
Composite Data Table
Curve
Curve Atlas
Curve Table
Data Asset
Data Layer
Data Registry
Data Table
Haptic Feedback Buffer
Haptic Feedback Curve
Haptic Feedback Sound Wave
HLOD Layer
Interchange Blueprint Pipeline Base
Add
Import
Save All
All
Plugins
Hak
All
Content
Characters
Input
Materials
MSPresets
PhysicsMaterials
System
Weapons
C++ Classes
Plugins
CommonUser C++ Classes
ShooterCore Content
1
2
3


Pick Class For Data Asset Instance
COMMON
ALL CLASSES
gamefeature
GameFeatureData
1 item (1 selected)
Select
Cancel



☐ 아래와 같이 ShooterCore의 InitState를 바꾸어주자:

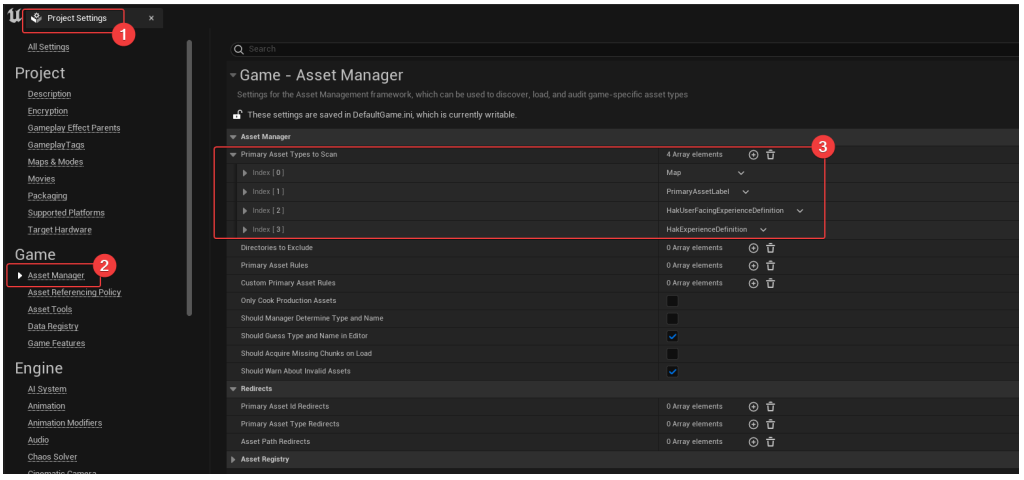




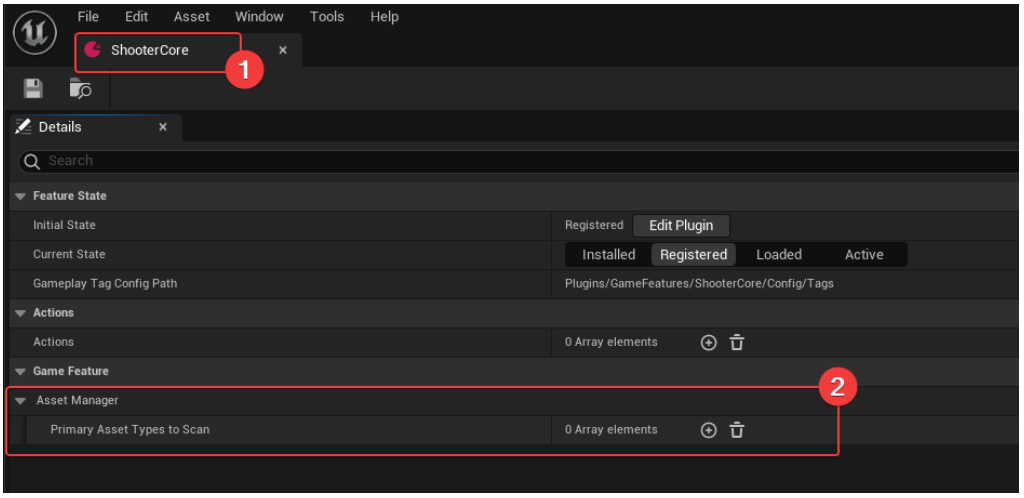
- 우리는 ShooterCore를 항상 활성화가 아닌 **필요할 때, 활성화를 진행할 예정**이다!

☐ GameFeatureData를 간단히 이해해보자:

- **GameFeature Plugin의 설정을 담당하는 DataAsset**으로 생각하면 됨
- 대표적으로, 앞서, 우리가 Project Setting에서 진행했던 PrimaryDataAsset 설정이 Plugin별로 GameFeatureData에서 설정 가능:
  - 기존 Project Settings:



  - GameFeatureData:

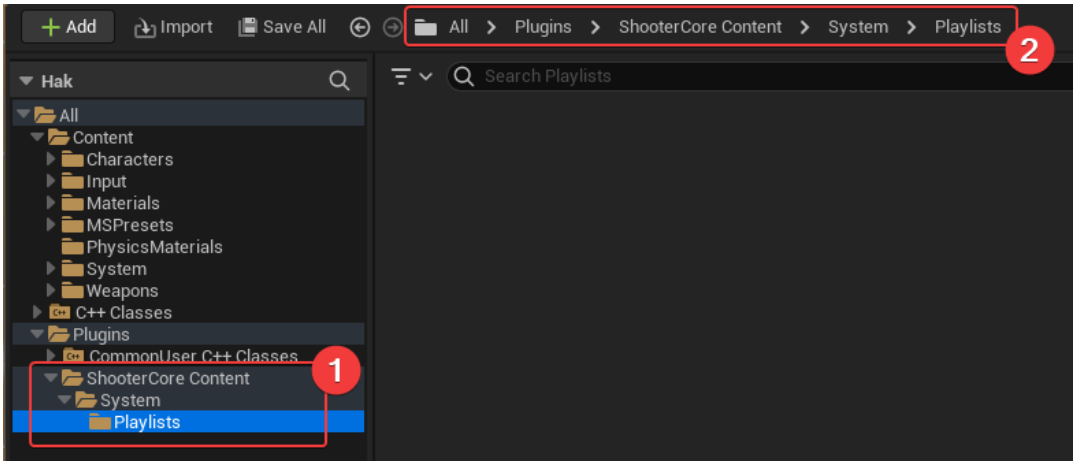


> 💡 GameFeatureData를 통해 **PrimaryAssetType을 추가하면, 자동적으로 Scan할 대상으로 추가**된다.

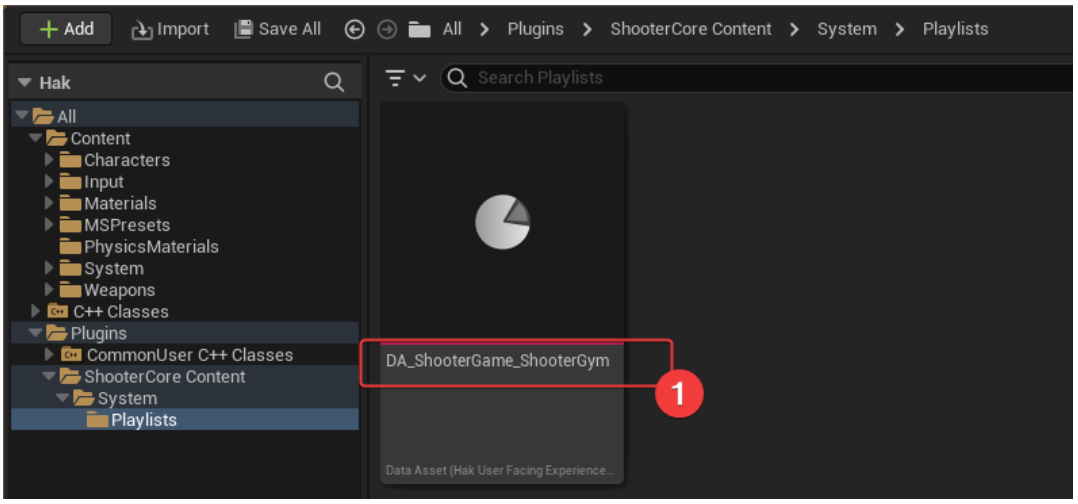
- 이제 과거의 기억을 회고해보며, ShooterCore에 해당하는 새로운 Experience를 추가해보자:

☐ UserFacingExperience에 해당하는 DA_ShooterGame_ShooterGym을 추가하자:

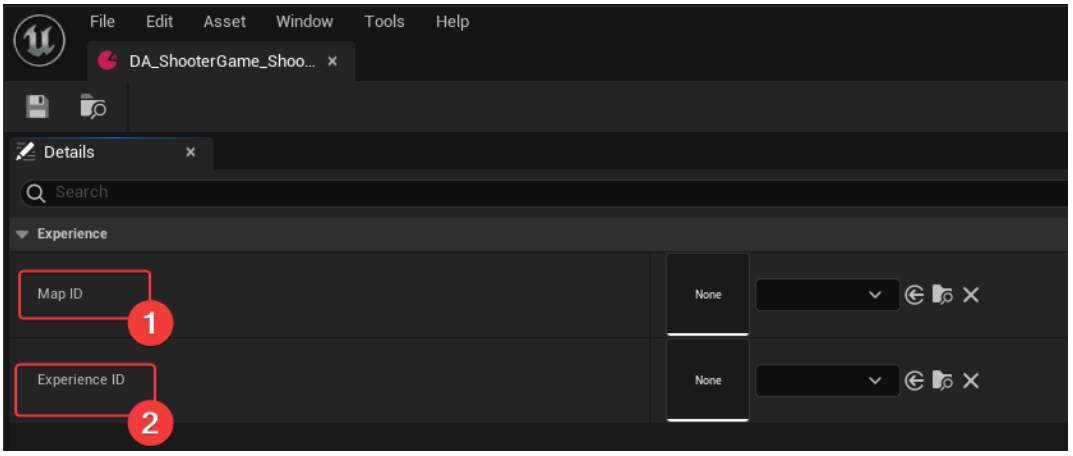
☐ 아래와 같이 폴더를 추가하자:



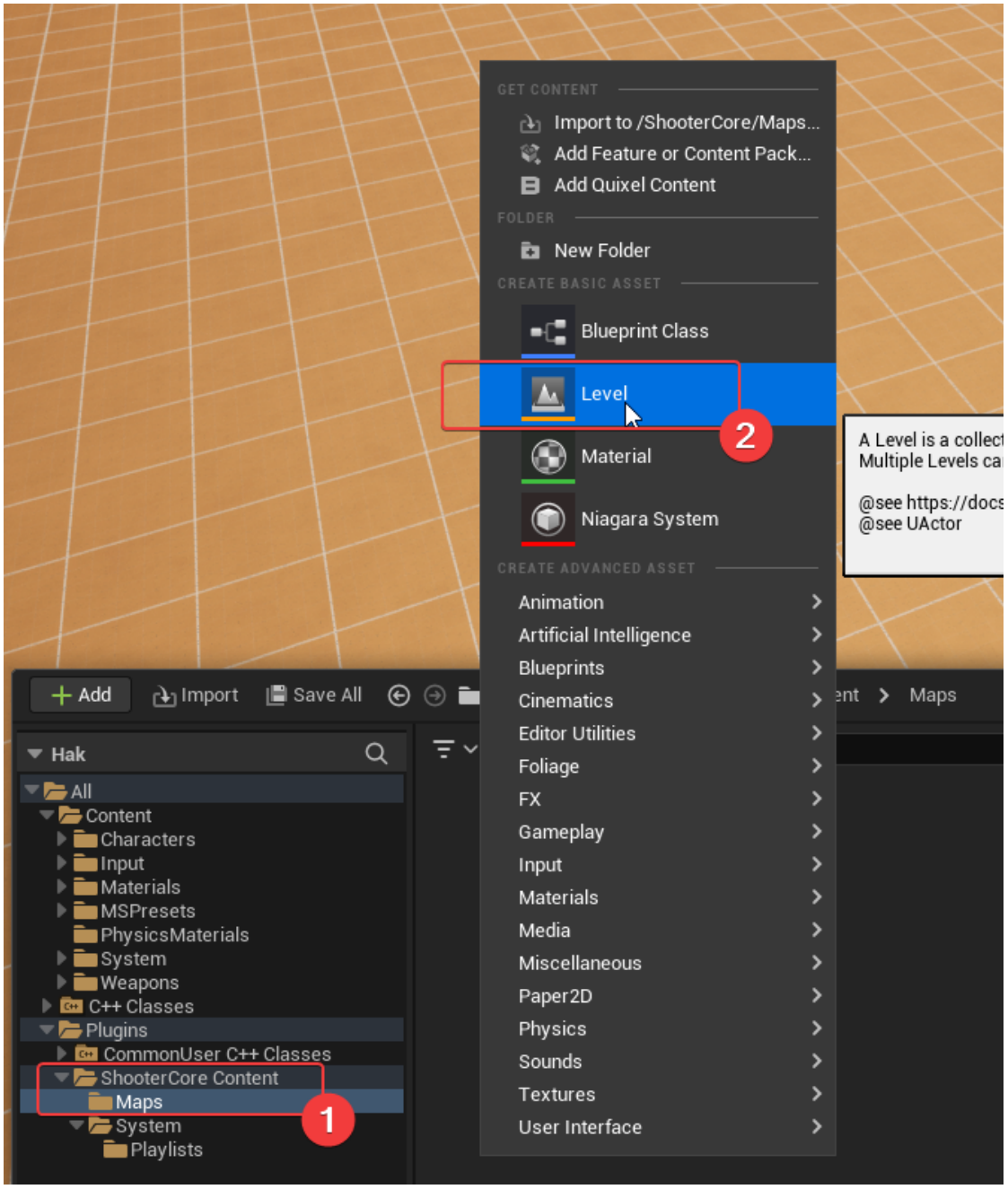
☐ DA_ShooterGame_ShooterGym을 추가하자:

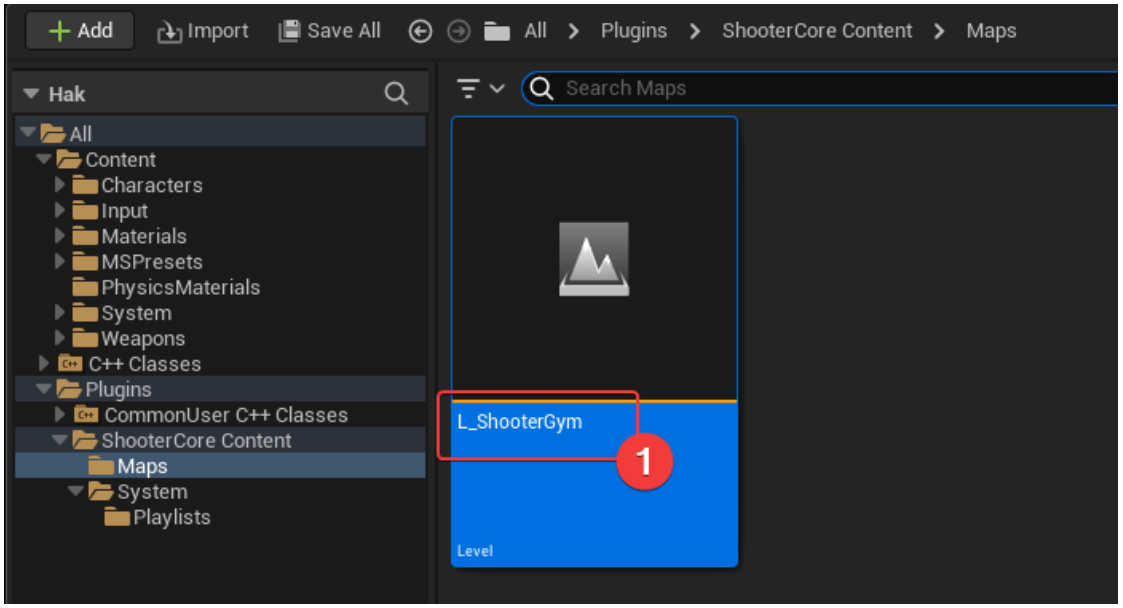


- 아래와 같이 Experience와 Map을 추가해야 한다:

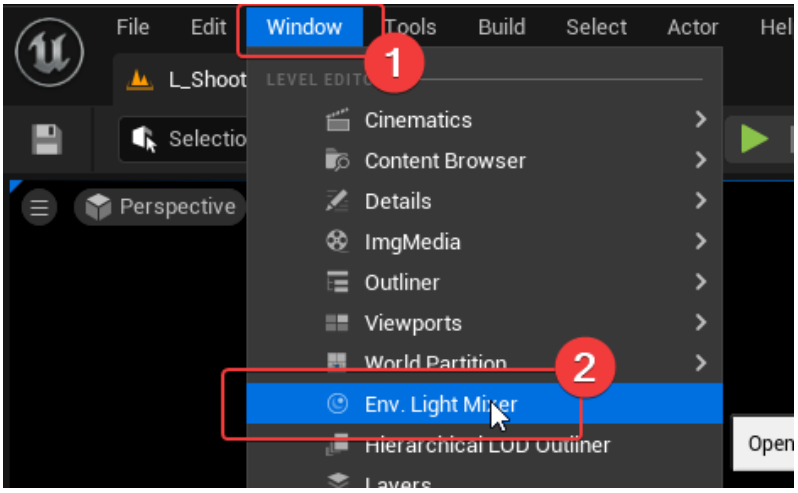


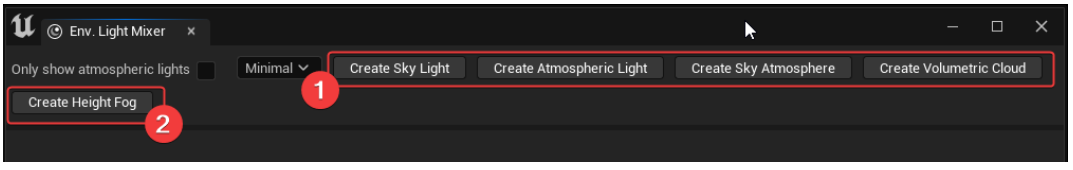
☐ Map을 추가해주자:


GET CONTENT
Import to /ShooterCore/Maps...
Add Feature or Content Pack...
Add Quixel Content
FOLDER
New Folder
CREATE BASIC ASSET
Blueprint Class
Level
Material
Niagara System
CREATE ADVANCED ASSET
Animation
Artificial Intelligence
Blueprints
Cinematics
Editor Utilities
Foliage
FX
Gameplay
Input
Materials
Media
Miscellaneous
Paper2D
Physics
Sounds
Textures
User Interface
2
A Level is a collect
Multiple Levels ca
@see https://docs
@see UActor
Add
Import
Save All
Maps
Hak
All
Content
Characters
Input
Materials
MSPresets
PhysicsMaterials
System
Weapons
C++ Classes
Plugins
CommonUser C++ Classes
ShooterCore Content
Maps
System
Playlists
1

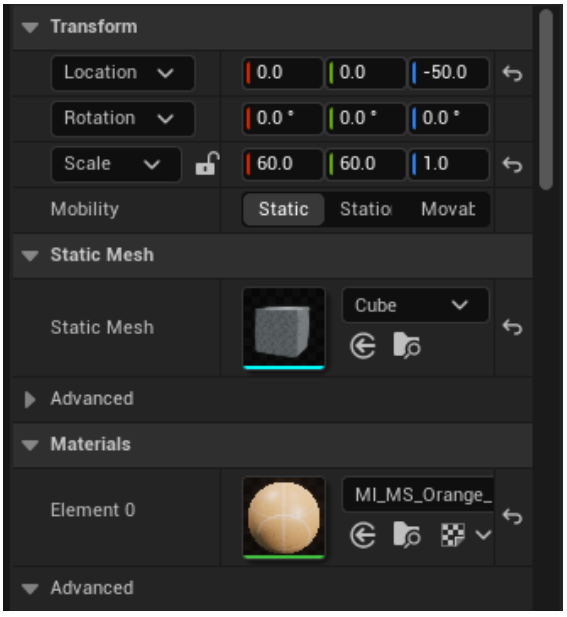


- [ ] L_ShooterGym을 L_DefaultEditorOverview와 똑같이 구성해주자:
  - Env.Light Mixer를 활용한 필수적인 환경세팅 진행하기:

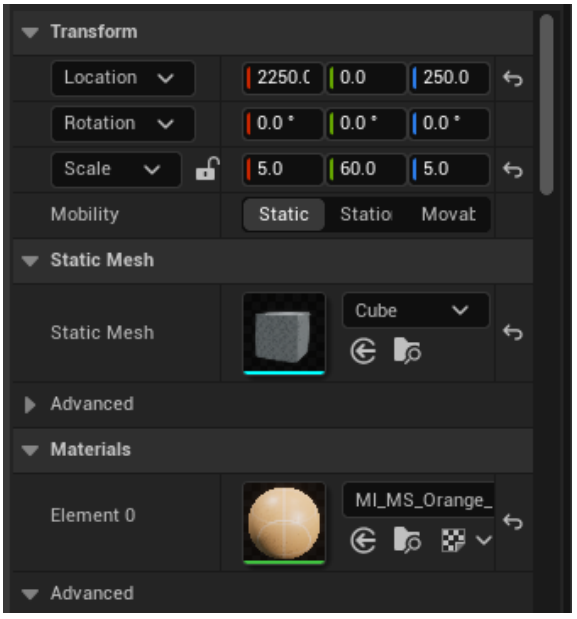




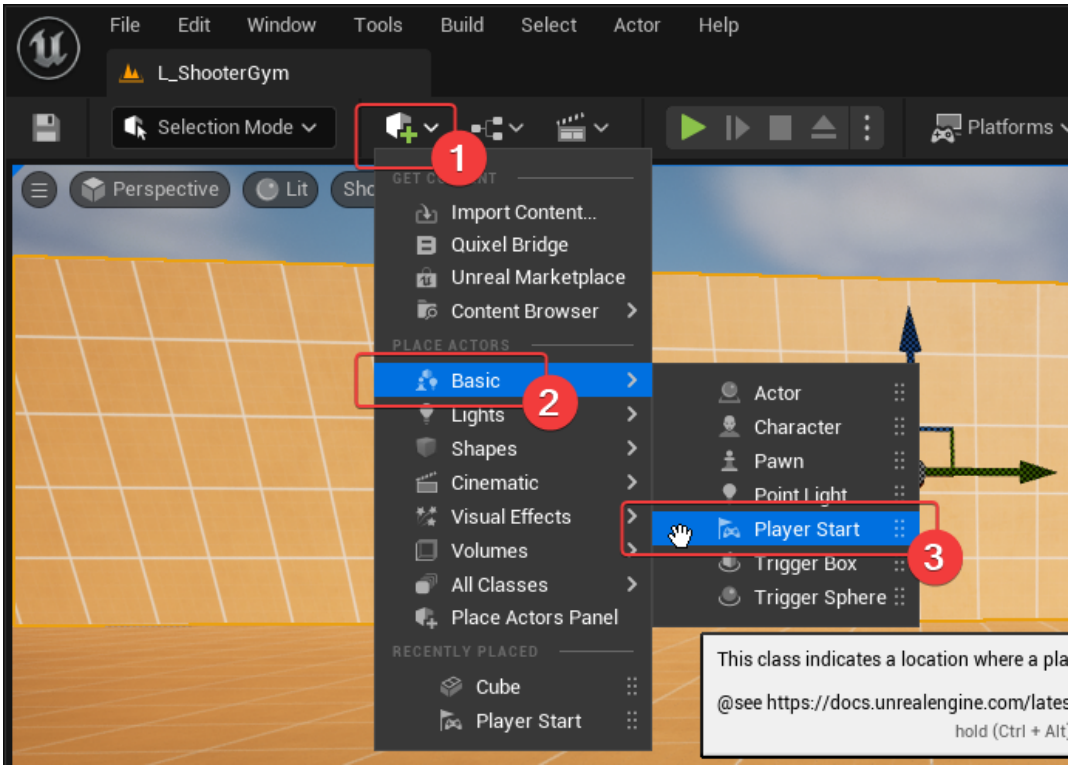
  - Cube, Cube2을 추가하여 Platform Shape 만들어주기:
    - Cube:



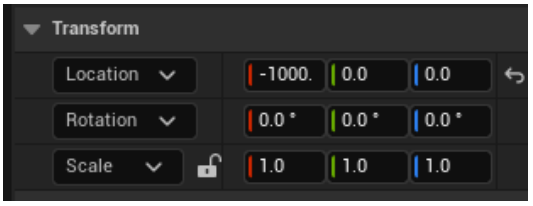
- Cube2:



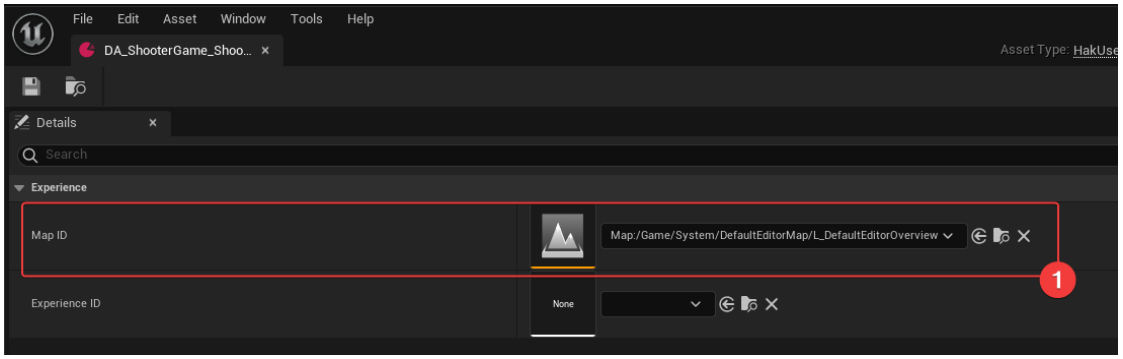
- PlayerStart 추가하기:



- 아래와 같이 속성값 설정:

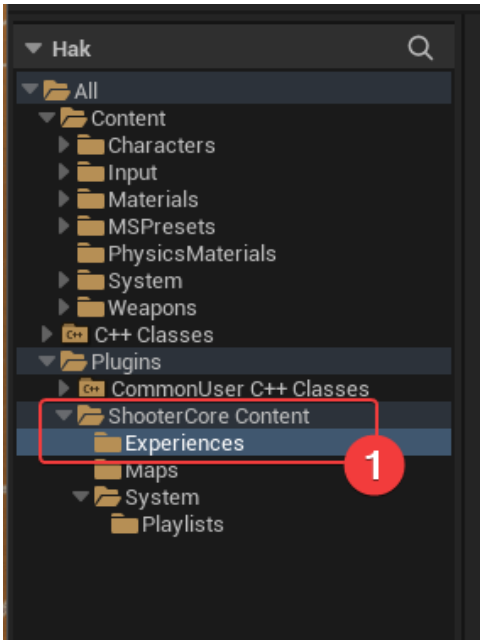


☐ L_ShooterGym을 DA_ShooterGame_ShooterGym에 추가하기:

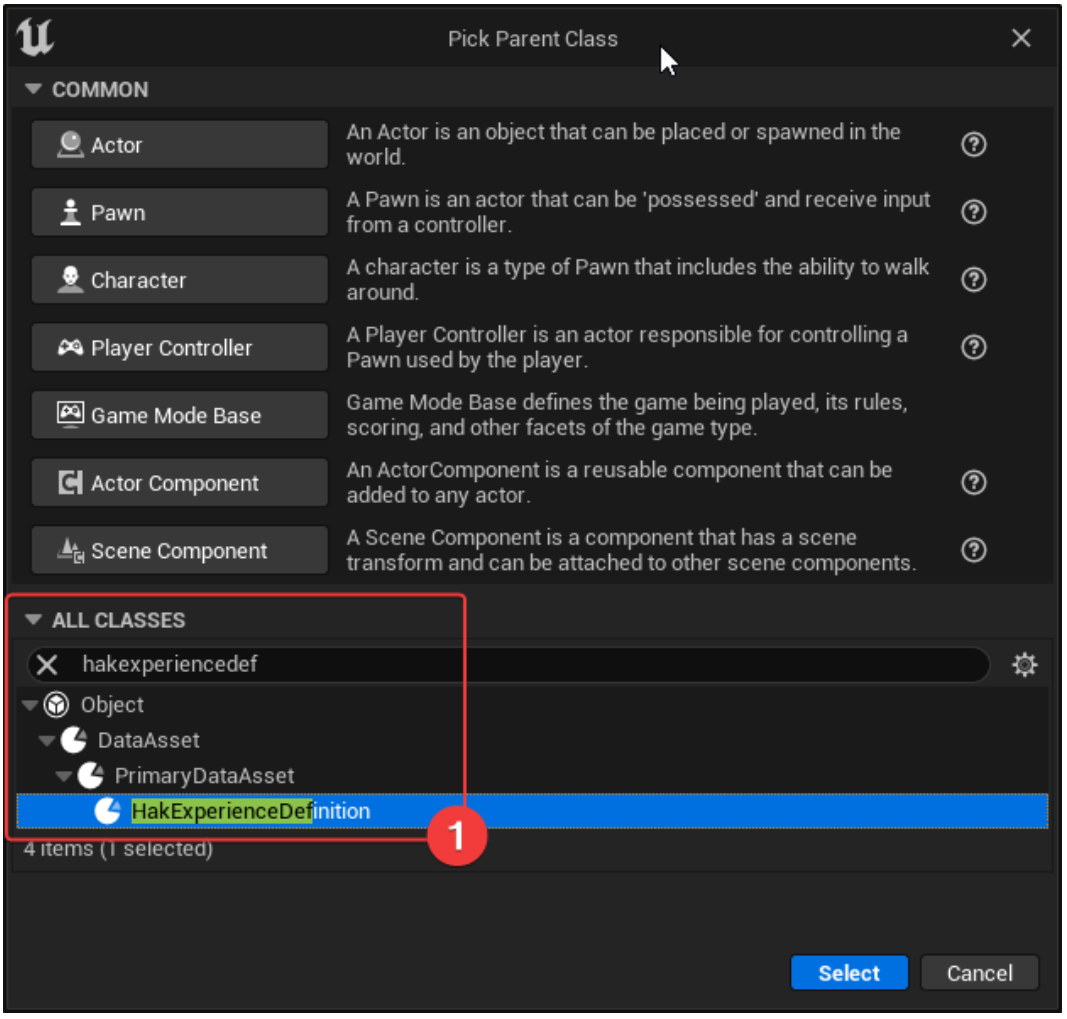


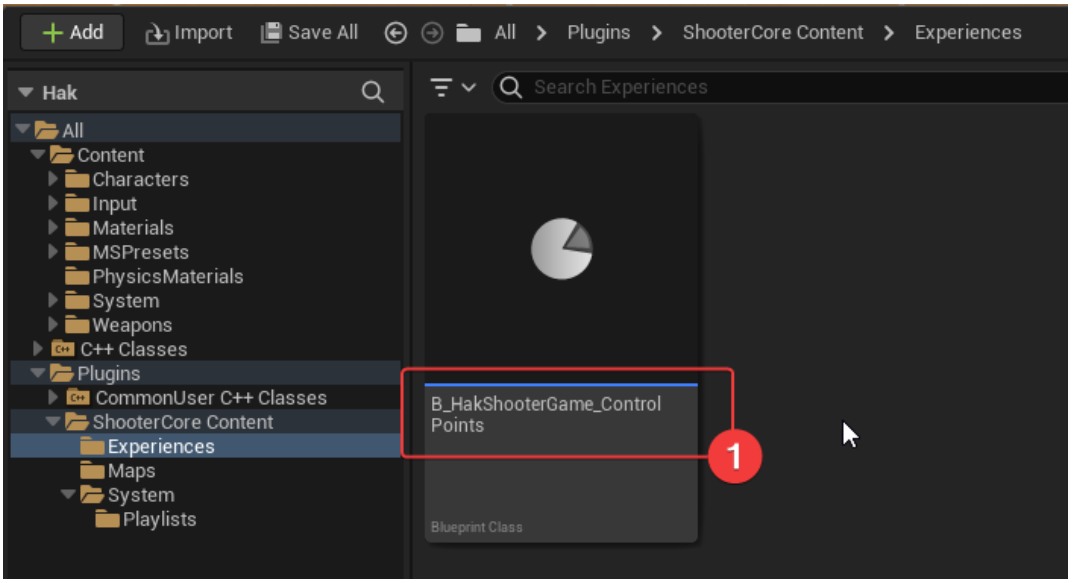
☐ HakExperienceDefinition인 B_HakShooterGame_ControlPoints 생성:

- Experiences 폴더 생성:

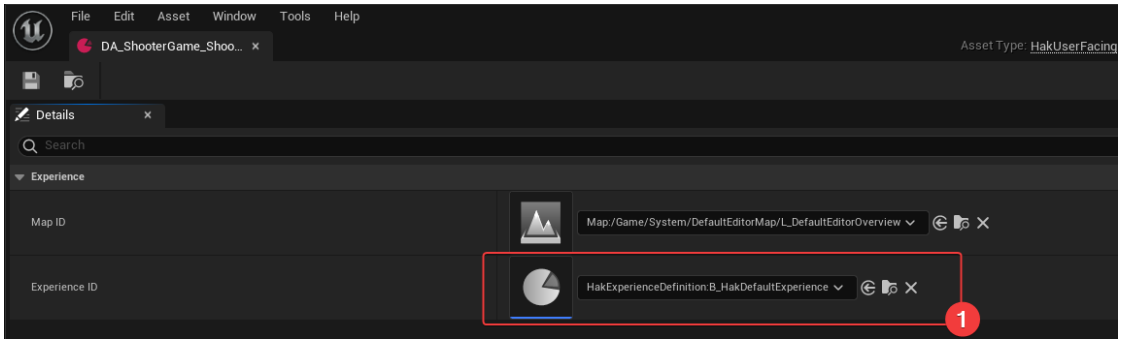


- B_HakShooterGame_ControlPoints 생성:

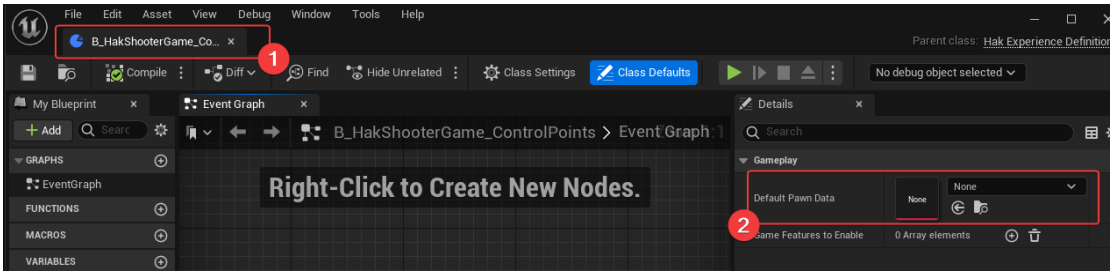




☐ DA_ShooterGame_ShooterGym에 B_HakShooterGame_ControlPoints를 추가해주자:

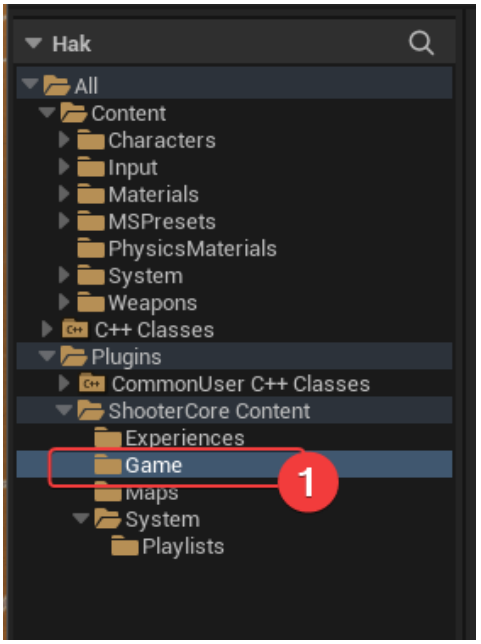


- B_HakShooterGame_ControlPoints에 우리는 DefaultPawnData를 설정해야 한다:

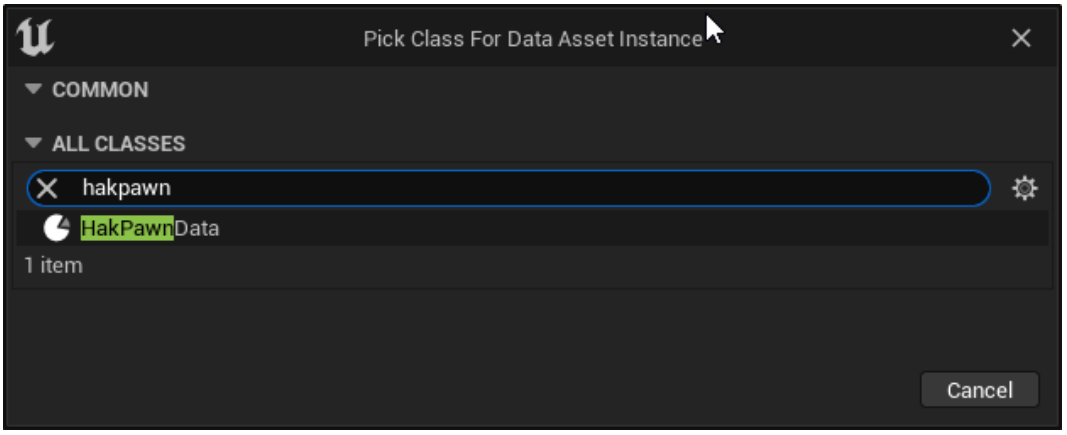


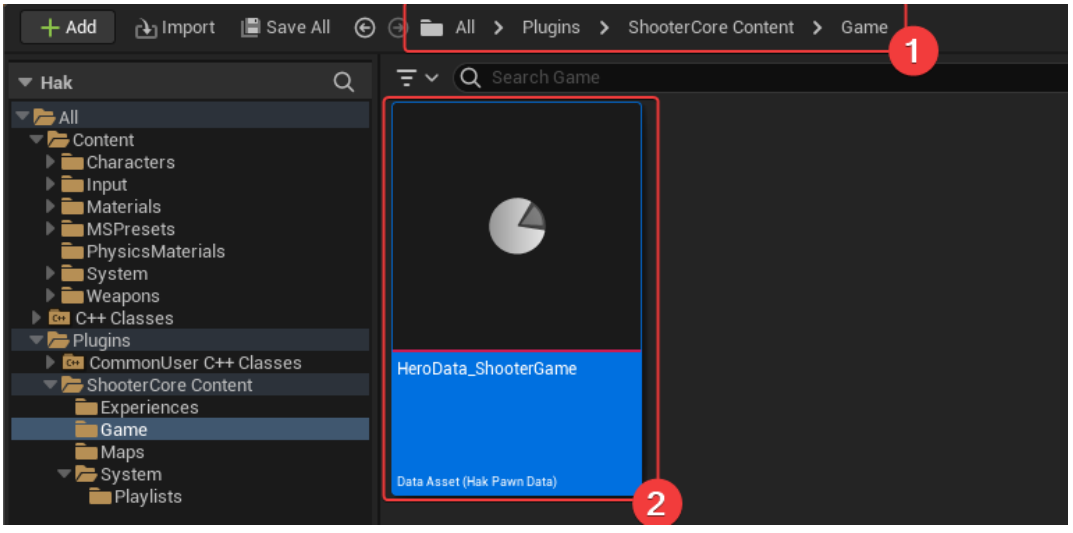
☐ HakPawnData인 HeroData_ShooterGame을 /Game 경로에 생성하자:

- /Game 폴더 생성:

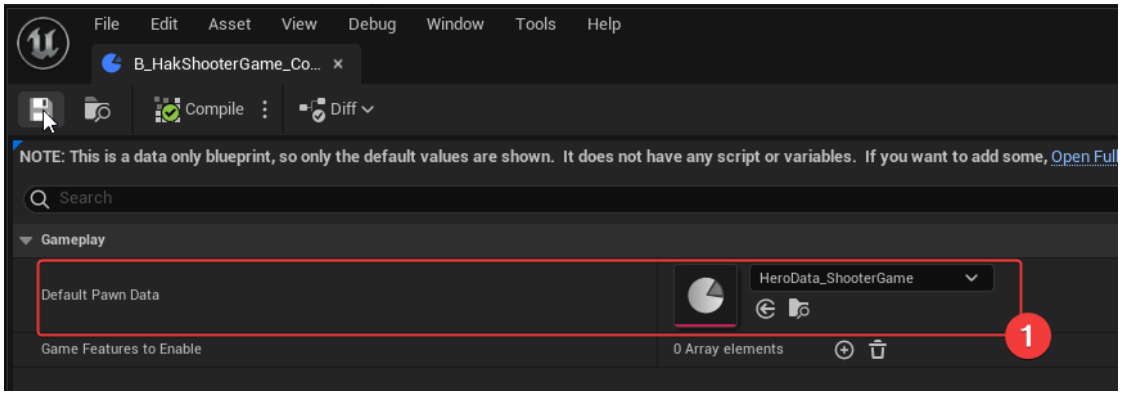


- HeroData_ShooterGame 생성:



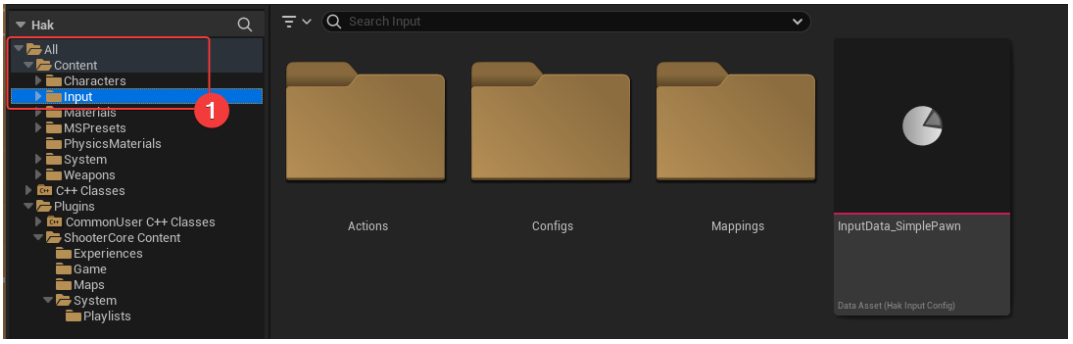


☐ HeroData_ShooterGame을 B_HakShooterGame_ControlPoints에 설정하자:

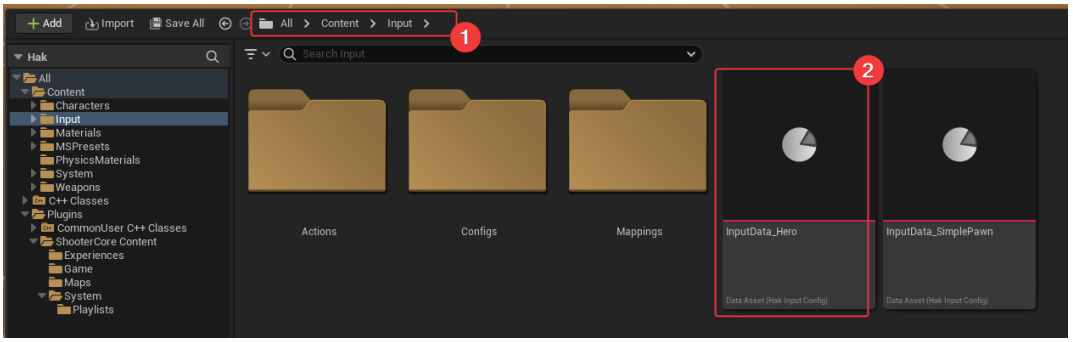


☐ HeroData_ShooterGame에 설정할 InputConfig를 추가하자:

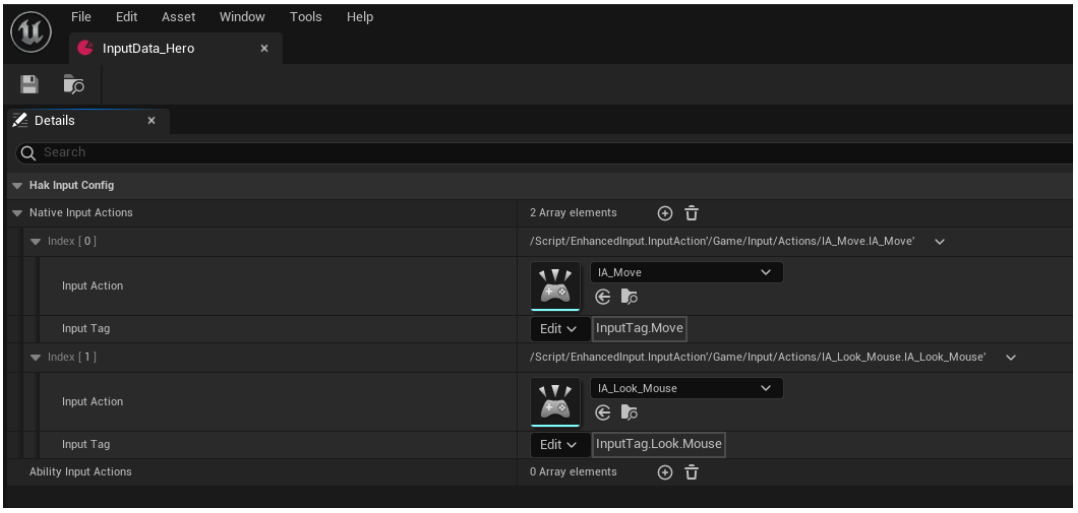
- /Content/Input에 InputData_Hero를 추가해주자 (해당 에셋은 HakGame에 있다):



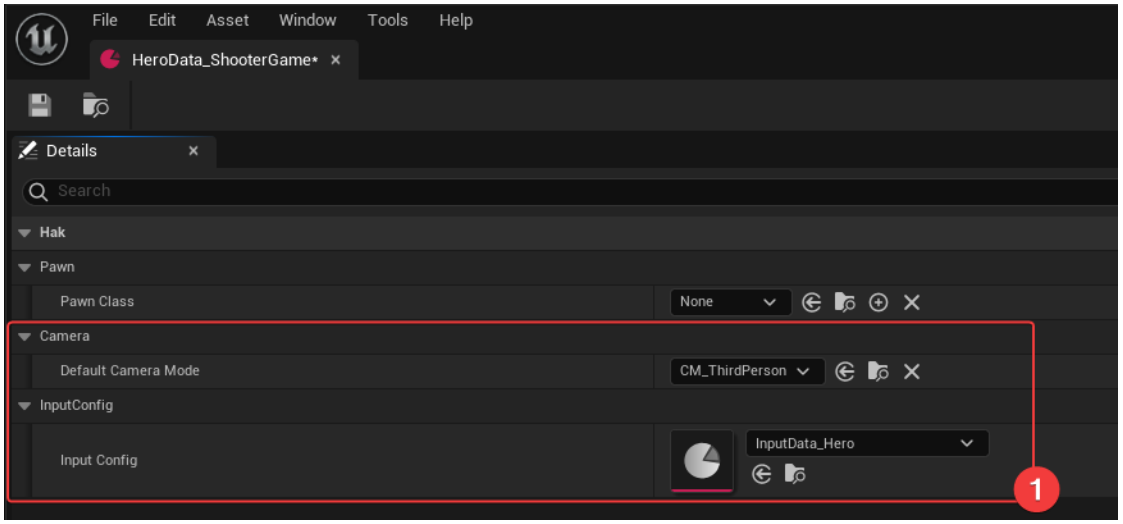
- HakInputConfig를 추가하자:



☐ InputData_Hero에 속성값을 설정하자:



☐ HeroData_ShooterGame의 속성값을 설정하자 (PawnClass만 제외하고):

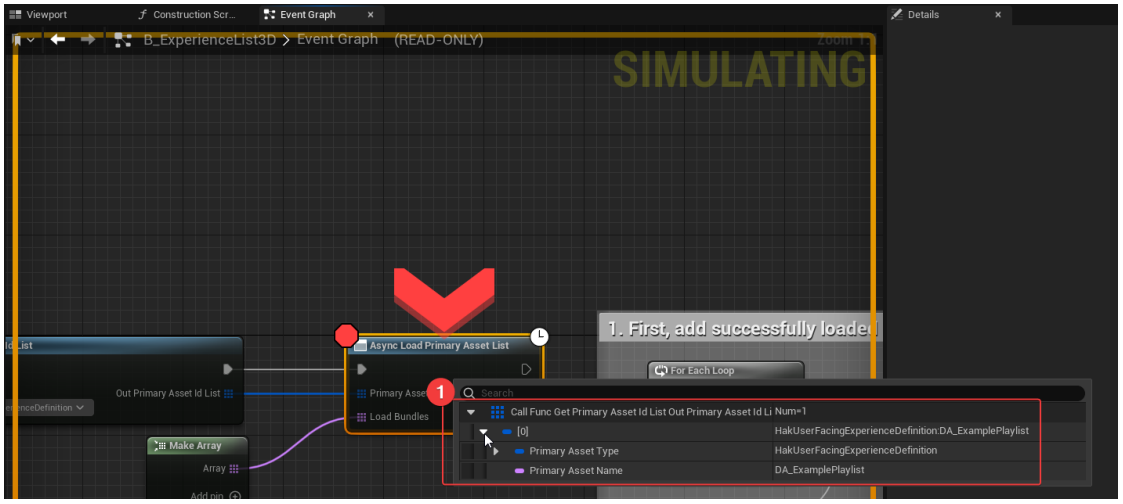


이제 **ShooterCore의 큰 형태를 갖추었고**, CommonSessionSubsystem을 활용하여, Traveral을 진행할 수 있게 되었다.

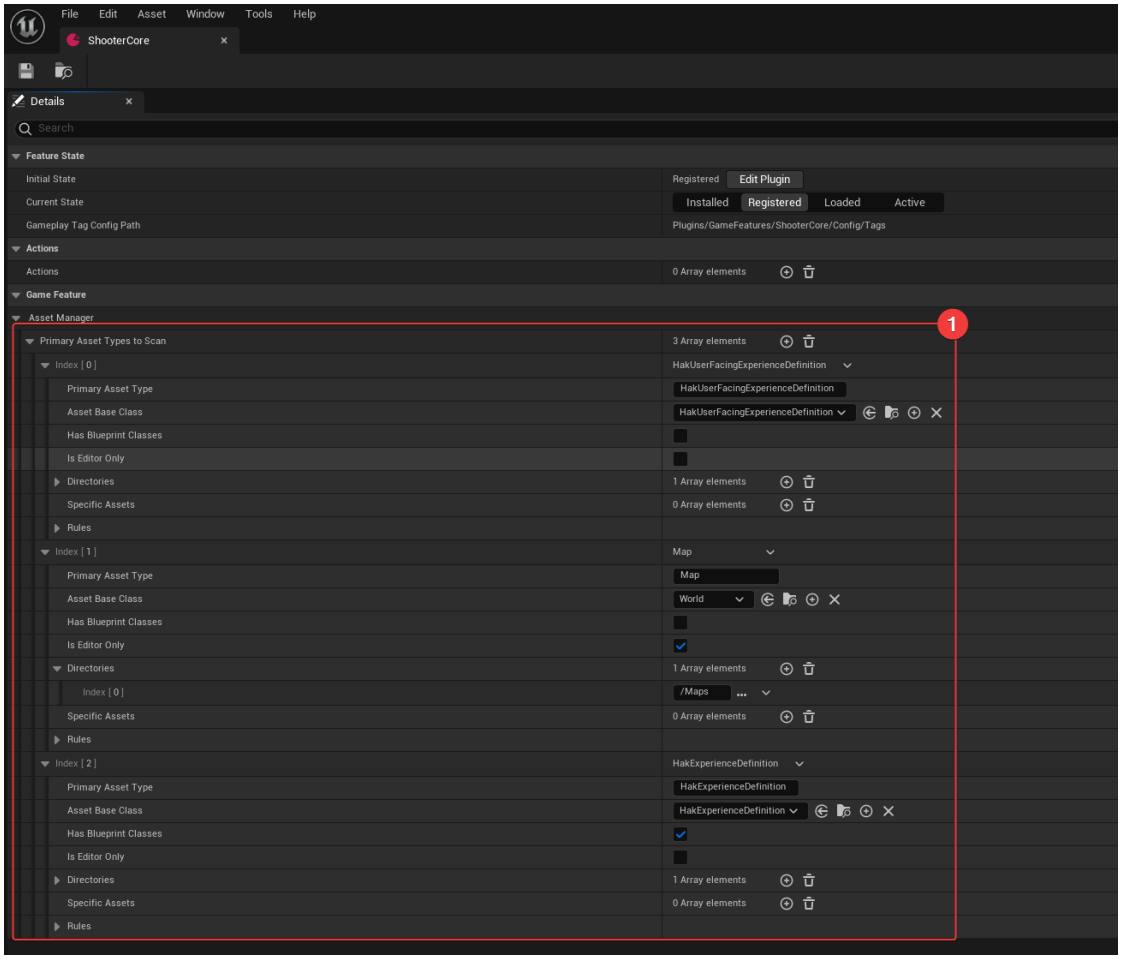
ShooterCore 맵 이동

▼ **펼치기**

☐ 우리가 추가한 DA_ShooterGame_ShooterGym이 보이지 않는다:

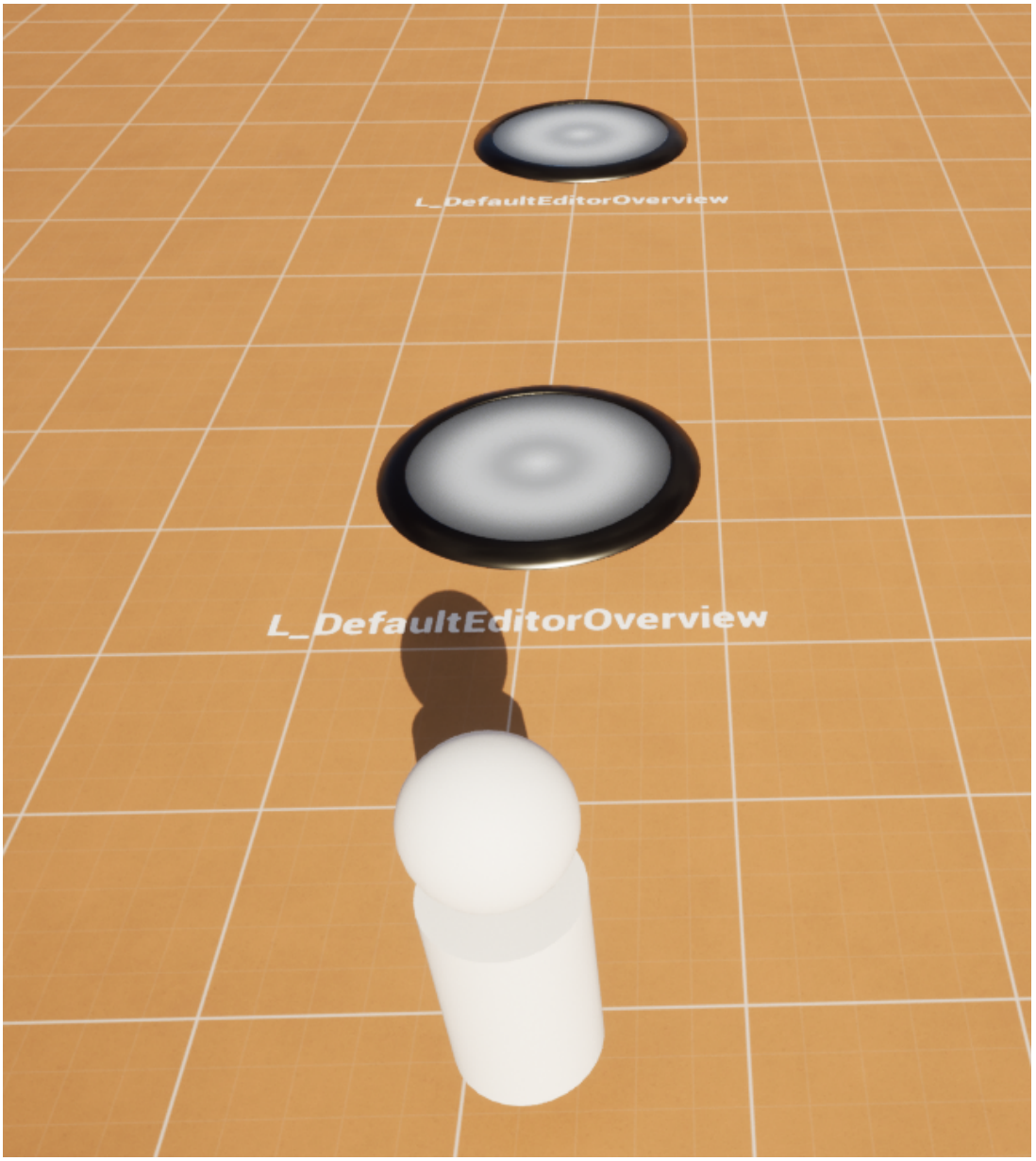


☐ 아래와 같이 PrimaryAssetTypes를 추가해주자:

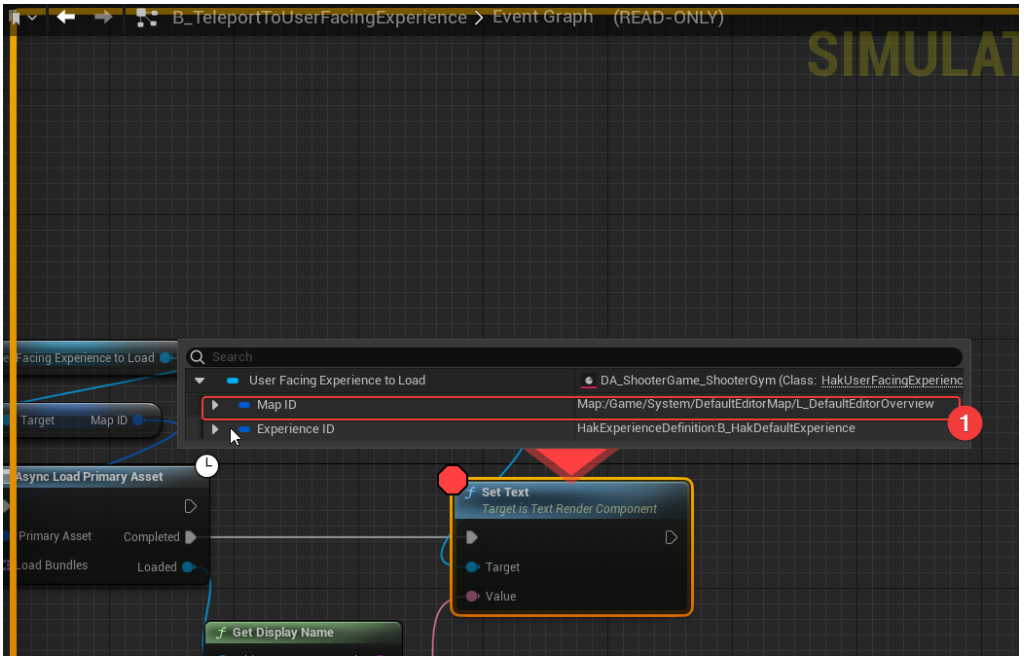


> 위 설정 이후, **에디터를 껐다가 켜야** 다시 Scan을 진행하며, AssetManager 내부에 PrimaryAssetTypes들의 경로가 캐싱된다.

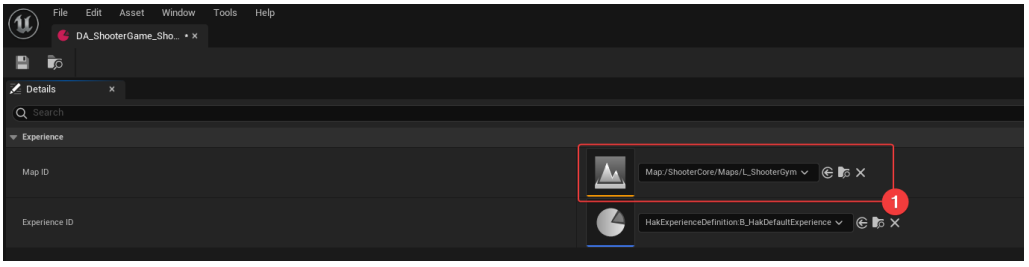
☐ 아래와 같이 2개의 Platform이 생기지만 두가지 문제가 있다:



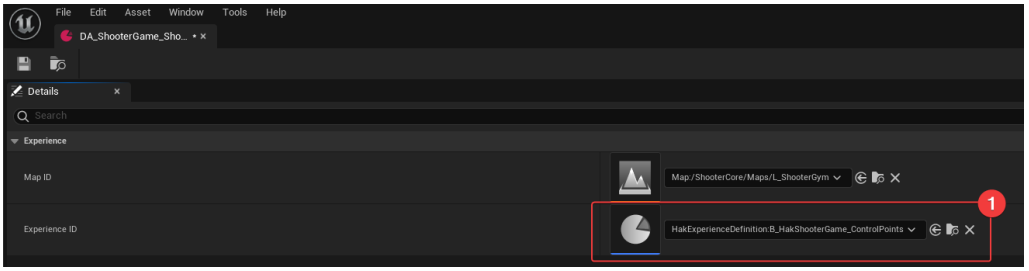
1. **텍스트가 똑같이 나온다...**
    - 디버깅을 해보면, 아래와 같이 우리가 DA_ShooterGame_ShooterGym에 MapID를 잘못 설정하였다:



☐ DA_ShooterGame_ShooterGym을 잘 설정해주자:

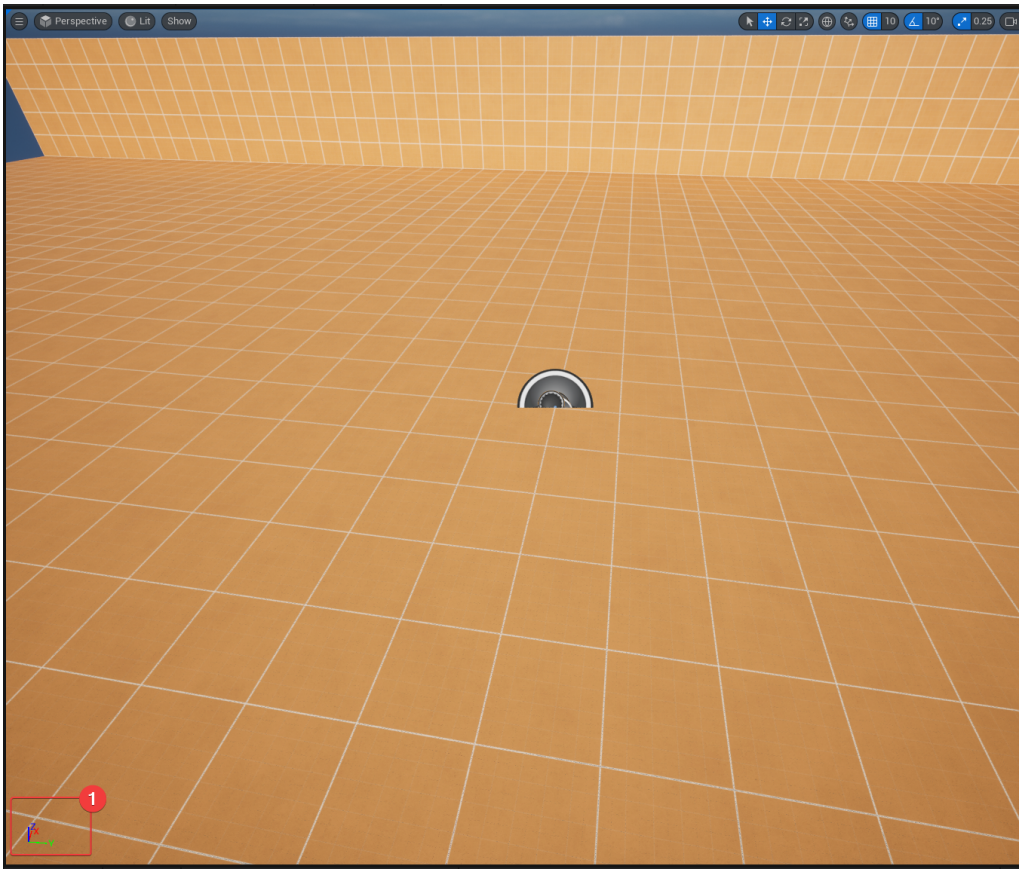


☐ ExperienceID도 제대로 설정해주자:

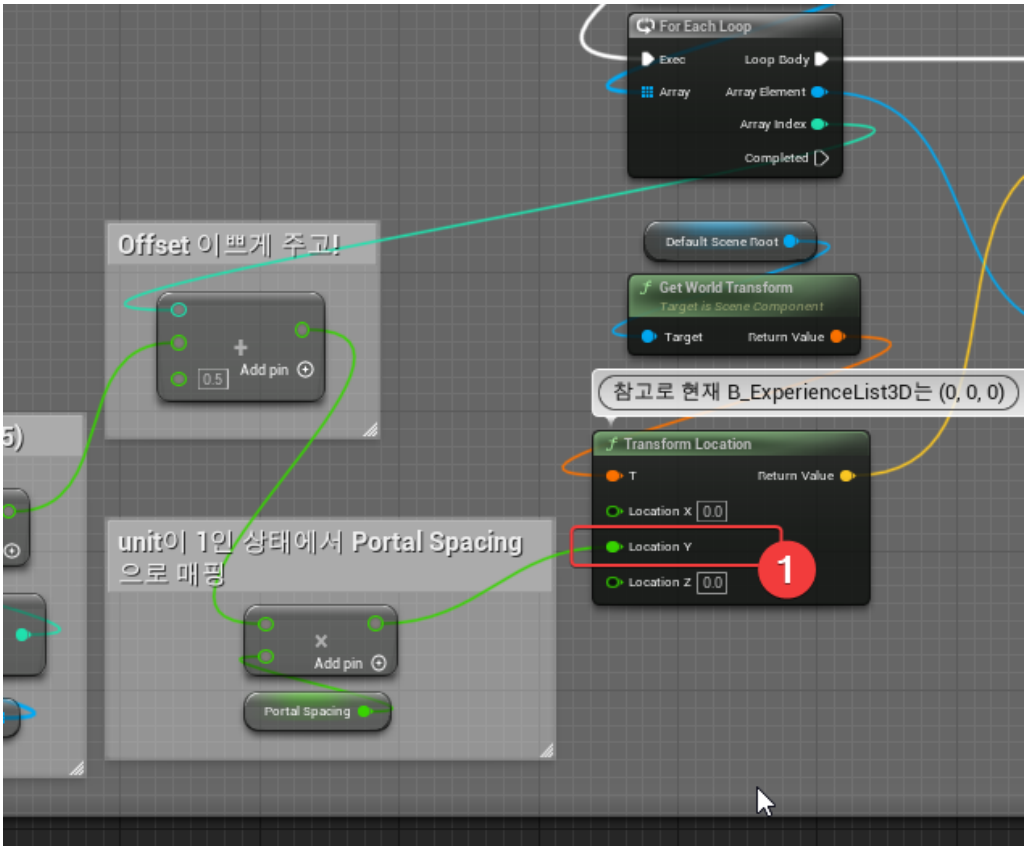


- 그럼 아래의 결과와 같다:

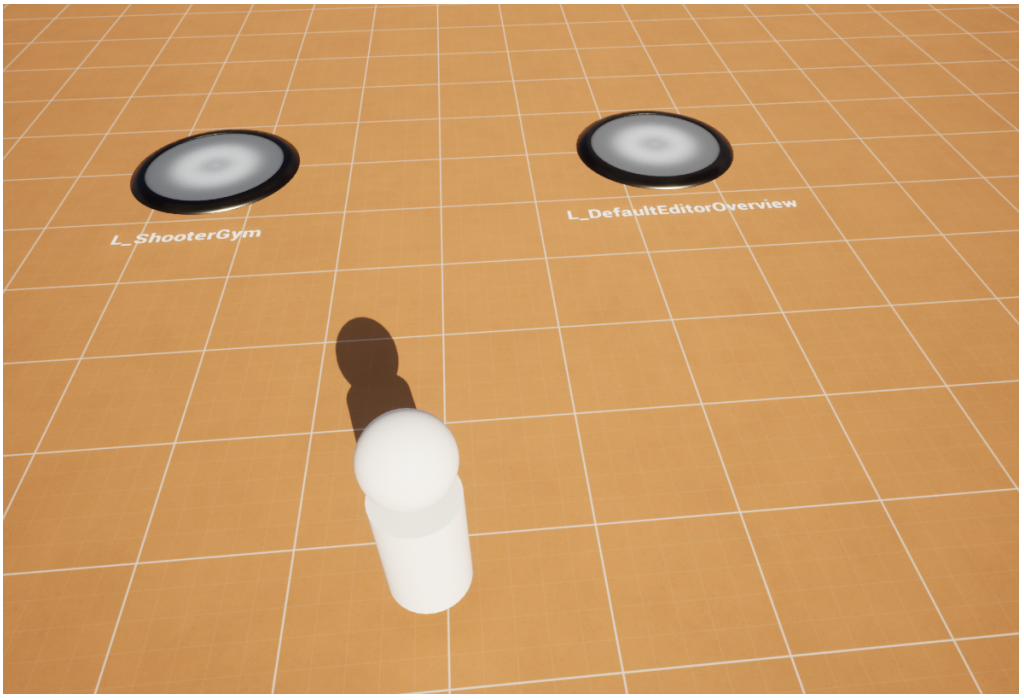
2. **Platform의 위치 나열이 세로**로 되어있다:
    - 아래를 보면 위의 방향이 X로, 우리는 Y축으로 나열되길 기대한다:



☐ 아래의 BP의 로직을 X에서 Y로 변경해주자:

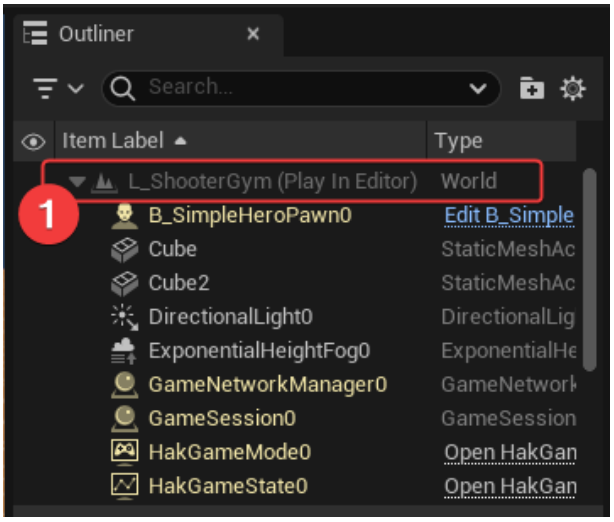


- 아래와 같이 잘 나열되어 Platform이 생성되었음을 확인할 수 있다:

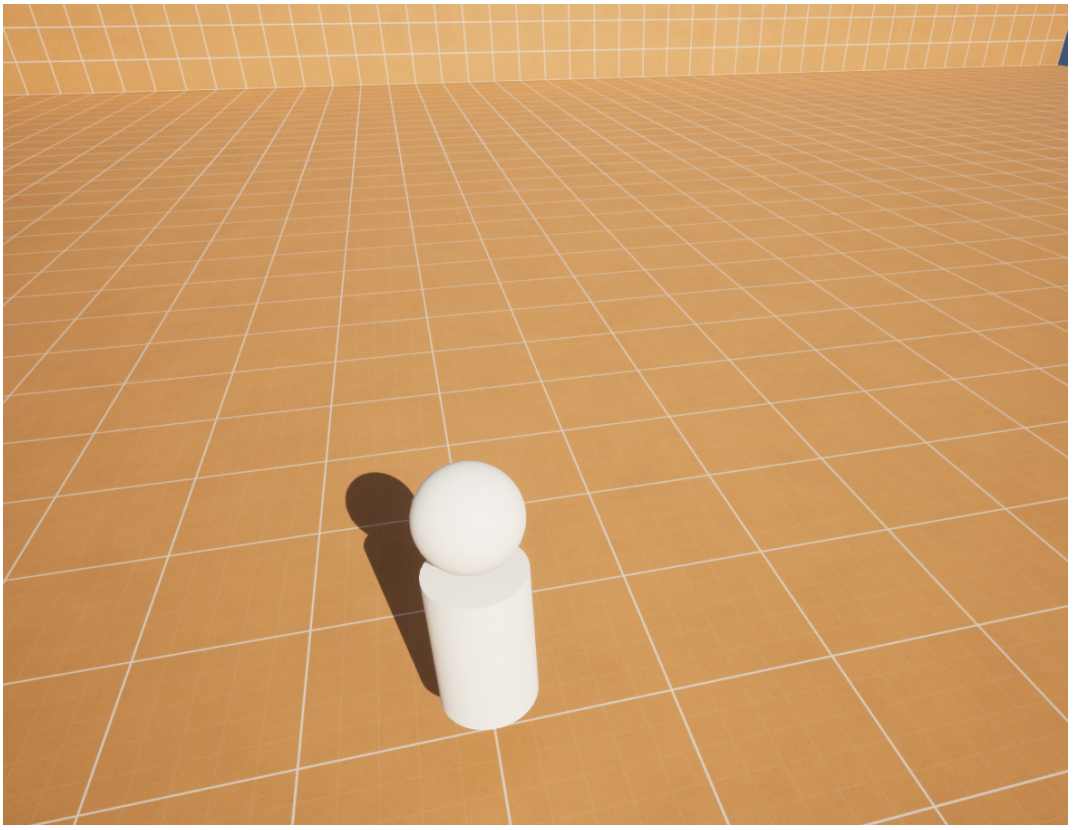


☐ 이동을 해보자:

- 우선 맵은 아래와 같이 L_ShooterGym으로 설정되었음을 확인할 수 있다:



- 그러나, SimplePawn이 조정도 가능하다?! (사실 **우리는 PawnClass를 설정하지 않았다**):



- 그럼 Experience이 **DA_ShooterGame_ShooterGym에 설정된 ExperienceID로 로딩이 안된것이다**

☐ 아래의 코드를 보도록 하자:

```
void AHakGameMode::HandleMatchAssignmentIfNotExpectingOne()
{
    // 해당 함수에서는 우리가 로딩할 Experience에 대해 PrimaryAssetId를 생성하여, OnMatchAssignmentGiven으로 넘겨준다

    FPrimaryAssetId ExperienceId;

    // precedence order (highest wins)
    // - matchmaking assignment (if present)
    // - default experience

    UWorld* World = GetWorld();

    // fall back to the default experience
    // 일단 기본 옵션으로 default하게 B_HakDefaultExperience로 설정놓자
    if (!ExperienceId.IsValid())
    {
        ExperienceId = FPrimaryAssetId(FPrimaryAssetType("HakExperienceDefinition"), FName("B_HakDefaultExperience"));
    }

    // 필자가 이해한 HandleMatchAssignmentIfNotExpectingOne과 OnMatchAssignmentGiven()은 아직 직관적으로 이름이 와닫지 않는다고 생각한다
    // - 후일, 어느정도 Lyra가 구현되면, 해당 함수의 명을 더 이해할 수 있을 것으로 예상한다
    OnMatchAssignmentGiven(ExperienceId);
}
```

1 현재 우리는 무조건 B_HakDefaultExperience를 설정했다...

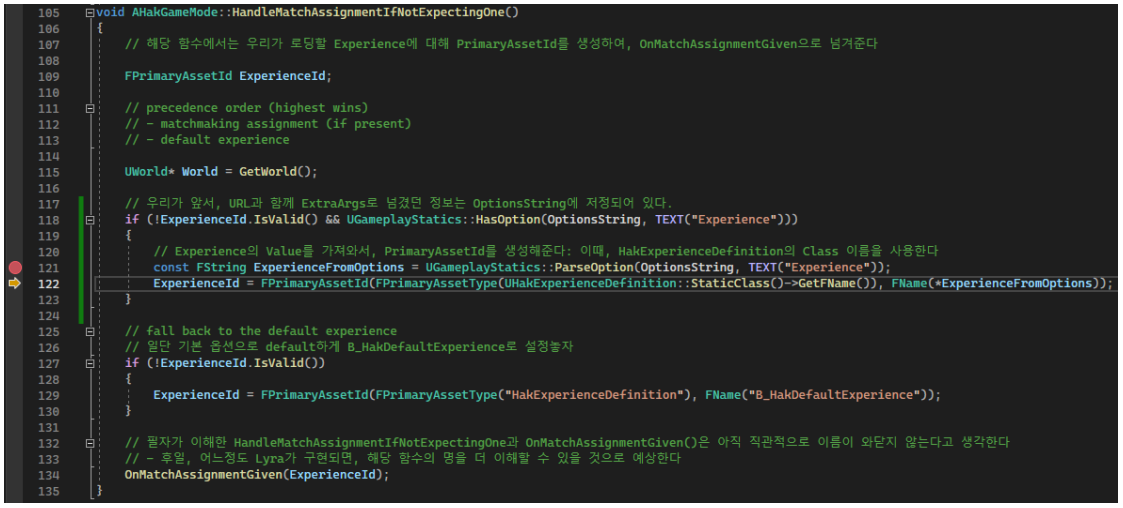
☐ 우리는 앞서 CommonSessionSubsystem에서 Experience에 대한 정보를 ExtraArgs를 통해 CmdArgs로 URL에 포함시켜 넘겨주었다:

- 이를 활용하여, 아래와 같이 구현하면 된다:

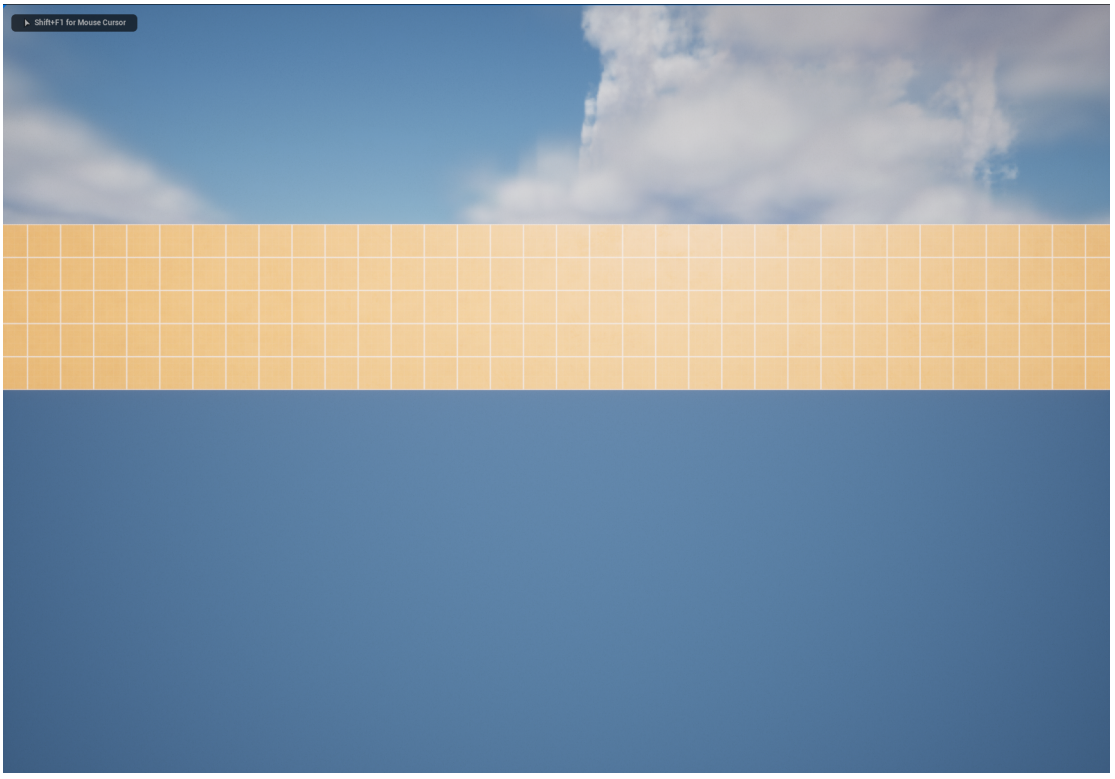
☐ HandleMatchAssignmentIfNotExpectingOne() 수정:

```cpp
void AHakGameMode::HandleMatchAssignmentIfNotExpectingOne()
{
    // 해당 함수에서는 우리가 로딩할 Experience에 대해 PrimaryAssetId를 생성하여, OnMatchAssignmentGiven으로 넘겨준다

    FPrimaryAssetId ExperienceId;

    // precedence order (highest wins)
    // - matchmaking assignment (if present)
    // - default experience

    UWorld* World = GetWorld();

    // 우리가 앞서, URL과 함께 ExtraArgs로 넘겼던 정보는 OptionsString에 저정되어 있다.
    if (!ExperienceId.IsValid() && UGameplayStatics::HasOption(OptionsString, TEXT("Experience")))
    {
        // Experience의 Value를 가져와서, PrimaryAssetId를 생성해준다: 이때, HakExperienceDefinition의 Class 이름을 사용한다
        const FString ExperienceFromOptions = UGameplayStatics::ParseOption(OptionsString, TEXT("Experience"));
        ExperienceId = FPrimaryAssetId(FPrimaryAssetType(UHakExperienceDefinition::StaticClass()->GetFName()), FName(*ExperienceFromOptions));
    }

    // fall back to the default experience
    // 일단 기본 옵션으로 default하게 B_HakDefaultExperience로 설정놓자
    if (!ExperienceId.IsValid())
    {
        ExperienceId = FPrimaryAssetId(FPrimaryAssetType("HakExperienceDefinition"), FName("B_HakDefaultExperience"));
    }

    // 필자가 이해한 HandleMatchAssignmentIfNotExpectingOne과 OnMatchAssignmentGiven()은 아직 직관적으로 이름이 와닫지 않는다고 생각한다
    // - 후일, 어느정도 Lyra가 구현되면, 해당 함수의 명을 더 이해할 수 있을 것으로 예상한다
    OnMatchAssignmentGiven(ExperienceId);
}
```

☐ 해당 코드 디버깅해보기:



- 결과는 아래와 같다:



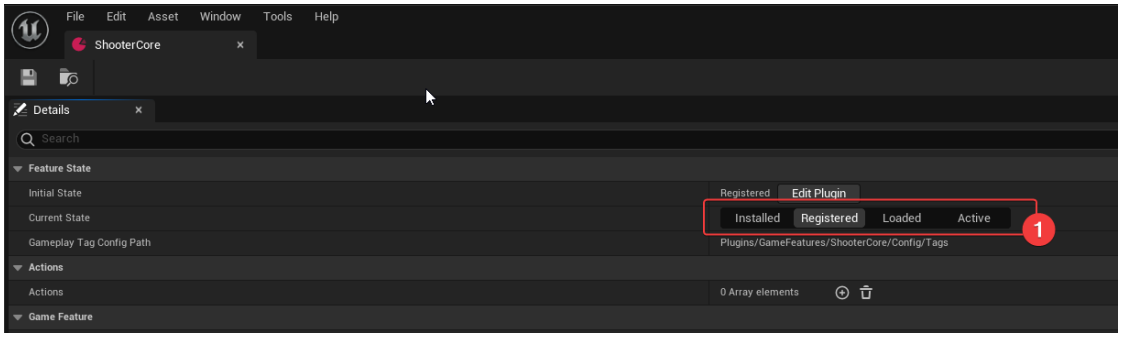
- 카메라 조정도 되지 않고, 우리가 의도한 Experience로 로딩이 잘 되었다.

> 여러분들이 코드에서 직접 어떤 Experience가 로딩되었는지 확인해도 된다!

GameFeature 활성화

▼ **펼치기**

☐ 아래와 같이 ShooterCore의 Map과 Experience도 들어가도, ShooterCore의 Game Feature는 활성화되지 않는다:



☐ ExperienceManagerComponent.h/.cpp에 아래와 같이 추가 구현을 해줘야 한다:

☐ OnExperienceLoadComplete:

☐ ExperienceManagerComponent의 멤버 변수를 추가해주자:

```cpp
/**
 * HakExperienceManagerComponent
 * - 말 그대로, 해당 component는 game state를 owner로 가지면서, experience의 상태 정보를 가지고 있는 component이다
 * - 뿐만 아니라, manager라는 단어가 포함되어 있듯이, experience 로딩 상태 업데이트 및 이벤트를 관리한다
 */
UCLASS()
class UHakExperienceManagerComponent : public UGameStateComponent
{
    GENERATED_BODY()
public:
    UHakExperienceManagerComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * member methods
     */
    bool IsExperienceLoaded() { return (LoadState == EHakExperienceLoadState::Loaded) && (CurrentExperience != nullptr); }

    /**
     * 아래의 OnExperienceLoaded에 바인딩하거나, 이미 Experience 로딩이 완료되었다면 바로 호출함
     */
    void CallOrRegister_OnExperienceLoaded(FOnHakExperienceLoaded::FDelegate&& Delegate);

    void ServerSetCurrentExperience(FPrimaryAssetId ExperienceId);
    void StartExperienceLoad();
    void OnExperienceLoadComplete();
    void OnExperienceFullLoadCompleted();
    const UHakExperienceDefinition* GetCurrentExperienceChecked() const;

    /**
     * member variables
     */

    /**
     * 참고로 해당 멤버 변수는 Lyra에서는 'ReplicatedUsing='으로 선언되어있다:
     * - 현재 우리는 아직 Replication을 신경쓰지 않을 것이기에, 최대한 네트워크 서버 코드를 배제하도록 하겠다
     */
    UPROPERTY()
    TObjectPtr<const UHakExperienceDefinition> CurrentExperience;

    /** Experience의 로딩 상태를 모니터링 */
    EHakExperienceLoadState LoadState = EHakExperienceLoadState::Unloaded;

    /** Experience 로딩이 완료된 이후, Broadcasting Delegate */
    FOnHakExperienceLoaded OnExperienceLoaded;

    /** 활성화된 GameFeature Plugin들 */
    int32 NumGameFeaturePluginsLoading = 0;
    TArray<FString> GameFeaturePluginURLs;
};
```

1

☐ EHakExperienceLoadState에 LoadingGameFeatures 추가하기:

```cpp
enum class EHakExperienceLoadState
{
    Unloaded,
    Loading,
    LoadingGameFeatures,
    Loaded,
    Deactivating,
};
```

1

☐ OnGameFeaturePluginLoadComplete 추가:

```h
#include "CoreMinimal.h"
#include "Components/GameStateComponent.h"
#include "GameFeaturePluginOperationResult.h"
#include "HakExperienceManagerComponent.generated.h"

/** forward declaration */
class UHakExperienceDefinition;

enum class EHakExperienceLoadState
{
    Unloaded,
    Loading,
    LoadingGameFeatures,
    Loaded,
    Deactivating,
};

DECLARE_MULTICAST_DELEGATE_OneParam(FOnHakExperienceLoaded, const UHakExperienceDefinition*);

/**
 * HakExperienceManagerComponent
 * - 말 그대로, 해당 component는 game state를 owner로 가지면서, experience의 상태 정보를 가지고 있는 component이다
 * - 뿐만 아니라, manager라는 단어가 포함되어 있듯이, experience 로딩 상태 업데이트 및 이벤트를 관리한다
 */
UCLASS()
class UHakExperienceManagerComponent : public UGameStateComponent
{
    GENERATED_BODY()
public:
    UHakExperienceManagerComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * member methods
     */
    bool IsExperienceLoaded() { return (LoadState == EHakExperienceLoadState::Loaded) && (CurrentExperience != nullptr); }

    /**
     * 아래의 OnExperienceLoaded에 바인딩하거나, 이미 Experience 로딩이 완료되었다면 바로 호출함
     */
    void CallOrRegister_OnExperienceLoaded(FOnHakExperienceLoaded::FDelegate&& Delegate);

    void ServerSetCurrentExperience(FPrimaryAssetId ExperienceId);
    void StartExperienceLoad();
    void OnExperienceLoadComplete();
    void OnGameFeaturePluginLoadComplete(const UE::GameFeatures::FResult& Result);
    void OnExperienceFullLoadCompleted();
    const UHakExperienceDefinition* GetCurrentExperienceChecked() const;

    /**
     * member variables
     */
```

```h
void UHakExperienceManagerComponent::OnGameFeaturePluginLoadComplete(const UE::GameFeatures::FResult& Result)
{
}
```

☐ OnExperienceLoadComplete 변경:

```h
void UHakExperienceManagerComponent::OnExperienceLoadComplete()
{
    // FrameNumber를 주목해서 보자
    static int32 OnExperienceLoadComplete_FrameNumber = GFrameNumber;

    check(LoadState == EHakExperienceLoadState::Loading);
    check(CurrentExperience);

    // 이전 활성화된 GameFeature Plugin의 URL을 클리어해준다
    GameFeaturePluginURLs.Reset();

    auto CollectGameFeaturePluginURLs = [This = this](const UPrimaryDataAsset* Context, const TArray<FString>& FeaturePluginList)
    {
        // FeaturePluginList를 순회하며, PluginURL을 ExperienceManagerComponent의 GameFeaturePluginURLS에 추가해준다
        for (const FString& PluginName : FeaturePluginList)
        {
            FString PluginURL;
            if (UGameFeaturesSubsystem::Get().GetPluginURLByName(PluginName, PluginURL))
            {
                This->GameFeaturePluginURLs.AddUnique(PluginURL);
            }
        }
    };

    // GameFeaturesToEnable에 있는 Plugin만 일단 활성화할 GameFeature Plugin 후보군으로 등록
    CollectGameFeaturePluginURLs(CurrentExperience, CurrentExperience->GameFeaturesToEnable);

    // GameFeaturePluginURLs에 등록된 Plugin을 로딩 및 활성화:
    NumGameFeaturePluginsLoading = GameFeaturePluginURLs.Num();
    if (NumGameFeaturePluginsLoading)
    {
        LoadState = EHakExperienceLoadState::LoadingGameFeatures;
        for (const FString& PluginURL : GameFeaturePluginURLs)
        {
            // 매 Plugin이 로딩 및 활성화 이후, OnGameFeaturePluginLoadComplete 콜백 함수 등록
            // 해당 함수를 실행보도록 하자
            UGameFeaturesSubsystem::Get().LoadAndActivateGameFeaturePlugin(PluginURL, FGameFeaturePluginLoadComplete::CreateUObject(this, &ThisClass::OnGameFeaturePluginLoadComplete));
        }
    }
    else
    {
        // 해당 함수가 불리는 것은 앞서 보았던 StreamableDelegateDelayHelper에 의해 불림
        OnExperienceFullLoadCompleted();
    }
}
```

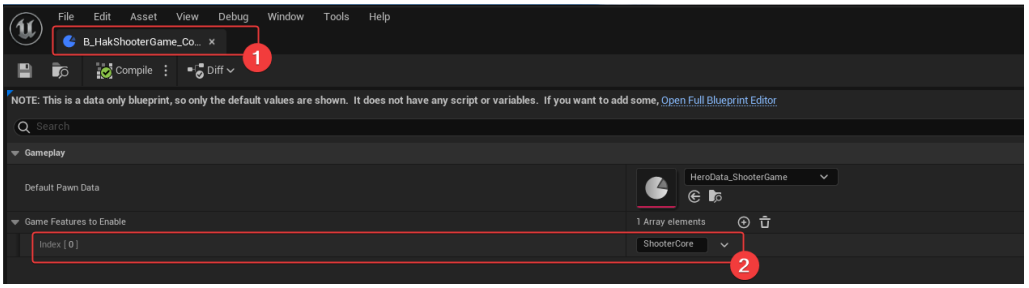
☐ OnGameFeaturePluginLoadComplete() 구현:

```cpp
void UHakExperienceManagerComponent::OnGameFeaturePluginLoadComplete(const UE::GameFeatures::FResult& Result)
{
    // 매 GameFeature Plugin이 로딩될 때, 해당 함수가 콜백으로 불린다
    NumGameFeaturePluginsLoading--;
    if (NumGameFeaturePluginsLoading == 0)
    {
        // GameFeaturePlugin 로딩이 다 끝났을 경우, 기존대로 Loaded로서, OnExperienceFullLoadCompleted 호출한다
        // GameFeaturePlugin 로딩과 활성화가 끝났다면? UGameFeatureAction을 활성화해야겠지 (조금만 있다가 하장)
        OnExperienceFullLoadCompleted();
    }
}
```

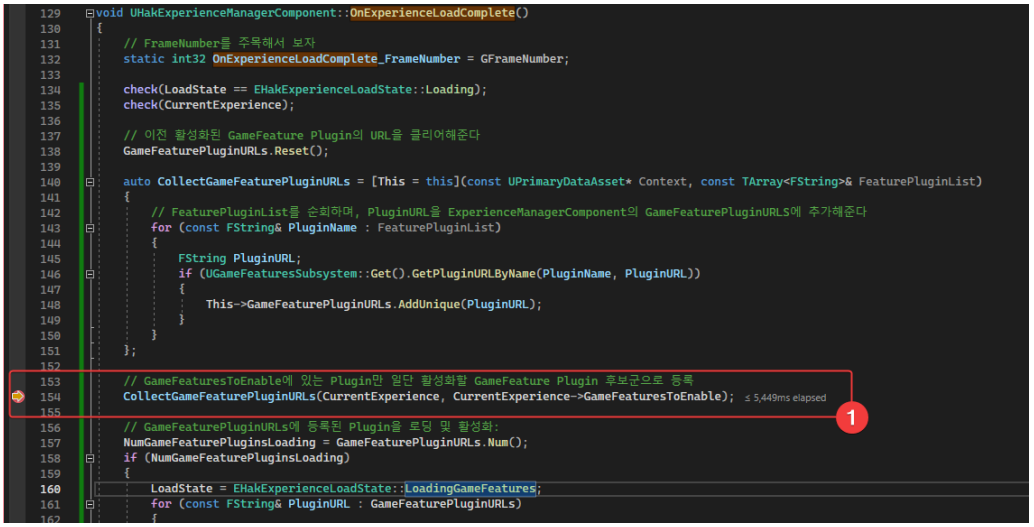
- 그럼 아래와 같이 ShooterCore로 텔레포트 하였을 경우, Plugin이 활성화 됨을 확인:

☐ 앞서, 코드에서 보았듯이, Experience에 대상이 되는 Game Feature를 지정해야 한다:

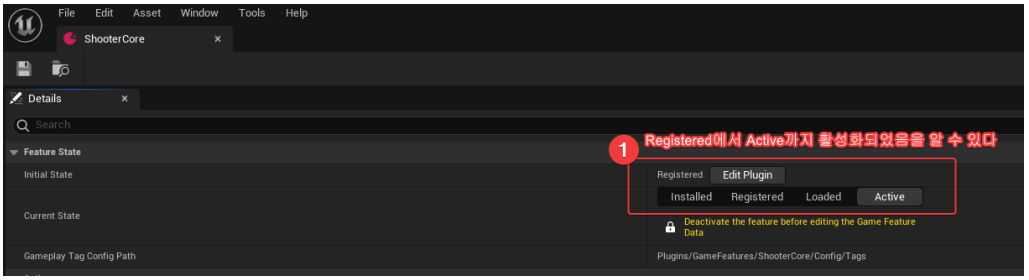
- 아래와 같이 B_HakShooterGame_ControlPoints에 Game Features to Enable에 ShooterCore를 추가해주자:



☐ 그리고 디버깅:



- 다시 확인:



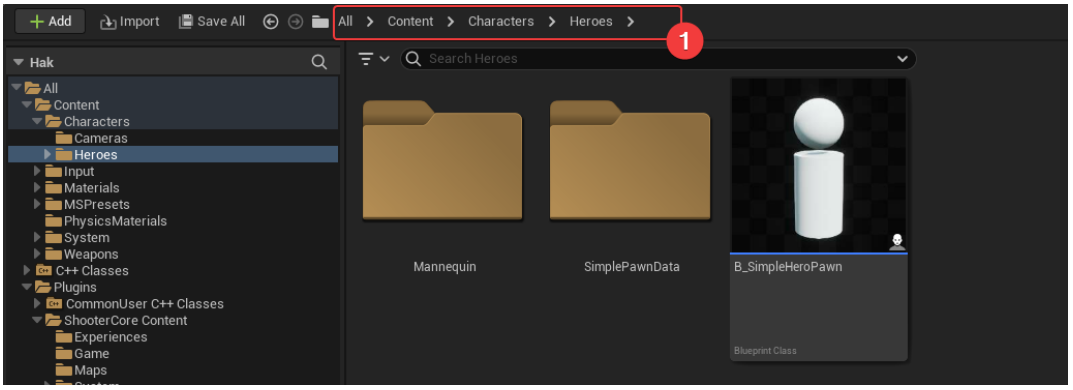
Input 활성화

▼ **펼치기**

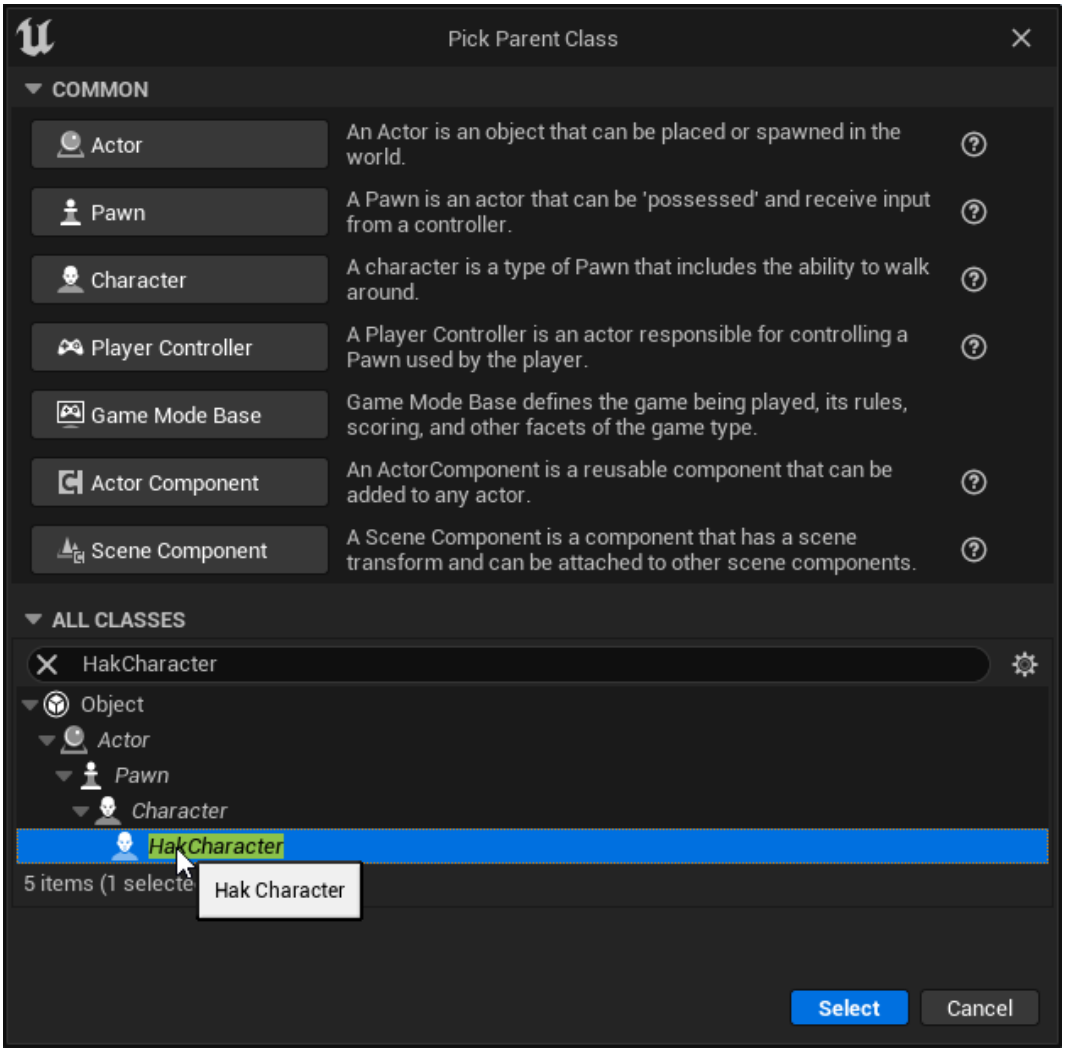
- 앞서, 우리가 SimpleHeroPawn에서 진행했듯이, **Input은 HeroComponent에서 활성화**가 진행된다:
  - ShooterCore의 HakCharacter를 만들어주자: `B_Hero_ShooterMannequin`

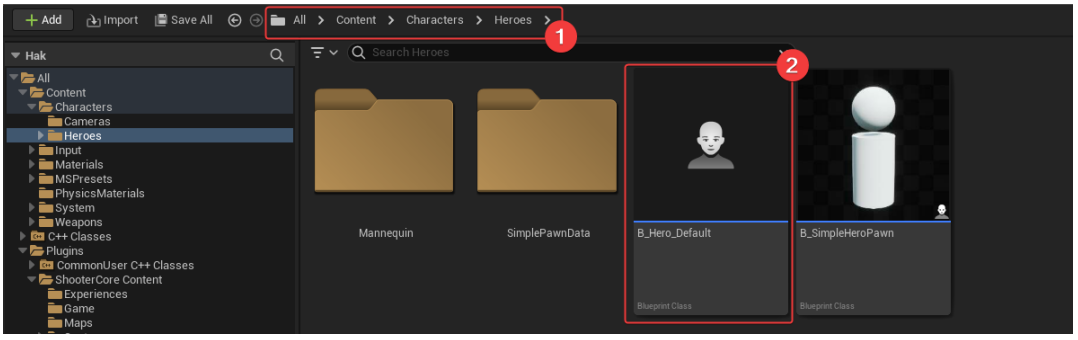
☐ B_Hero_ShooterMannequin가 상속 받을 B_Hero_Default 생성:

- 해당 경로로 위치하자:

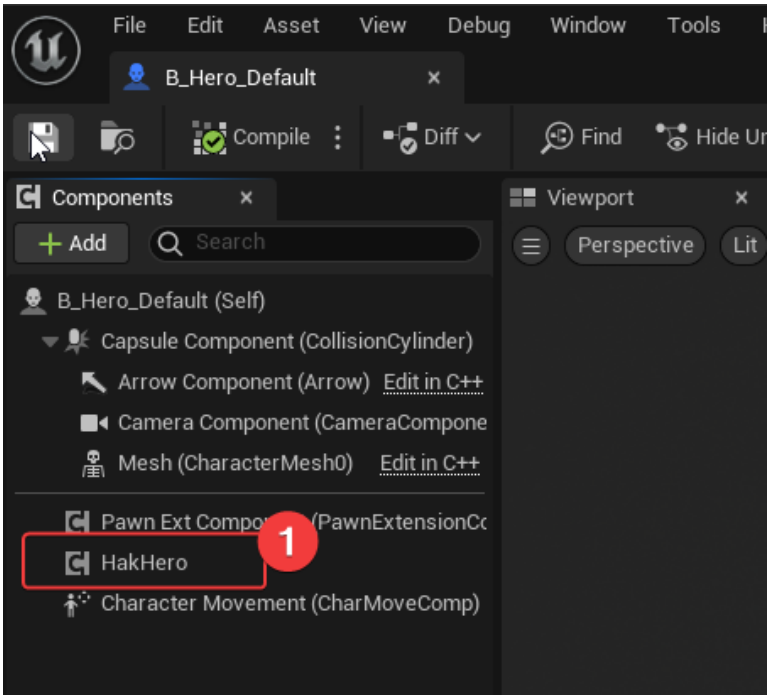


☐ HakCharacter를 상속받는 B_Hero_Default를 만들어주자:

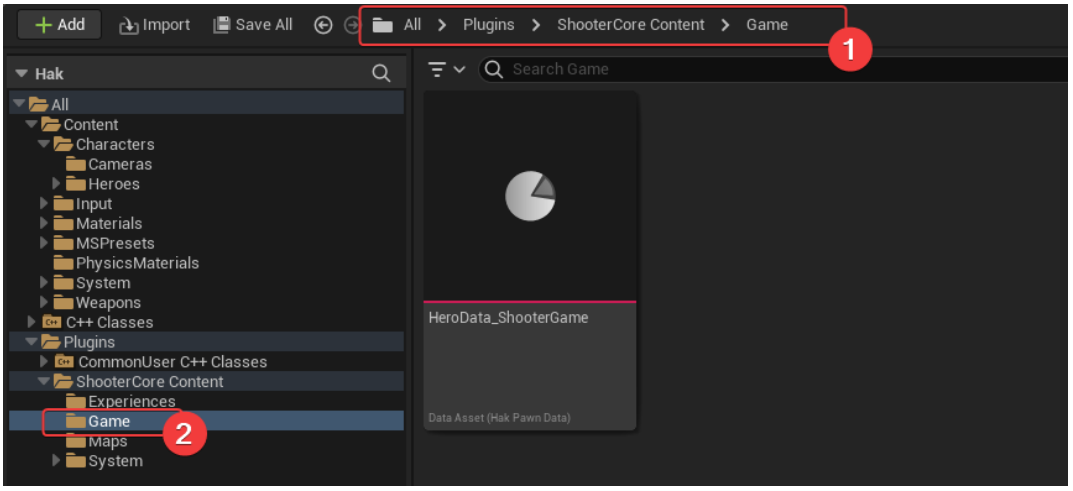




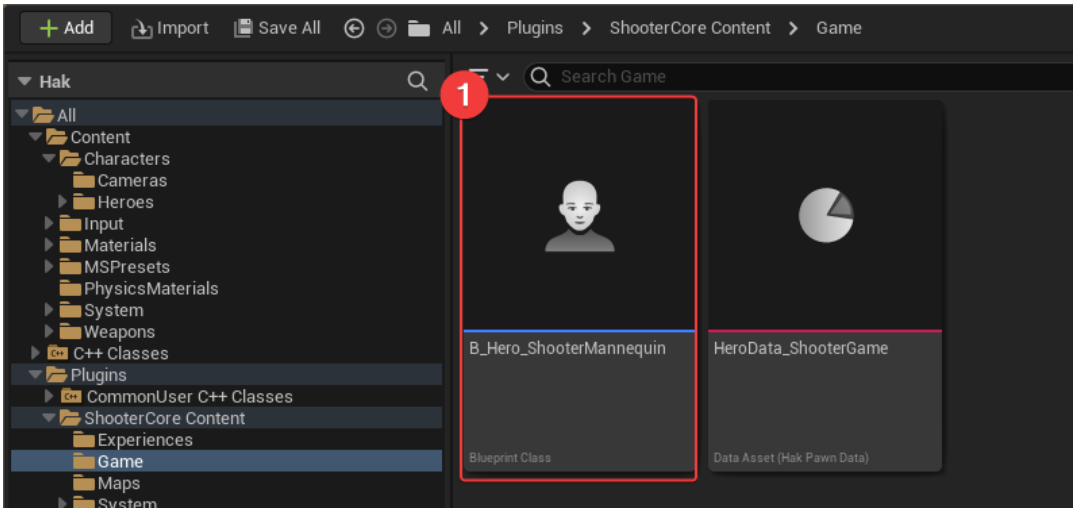
☐ 기존 B_SimpleHerPawn과 같이 HeroComponent를 B_Hero_Default에 추가시켜주자:



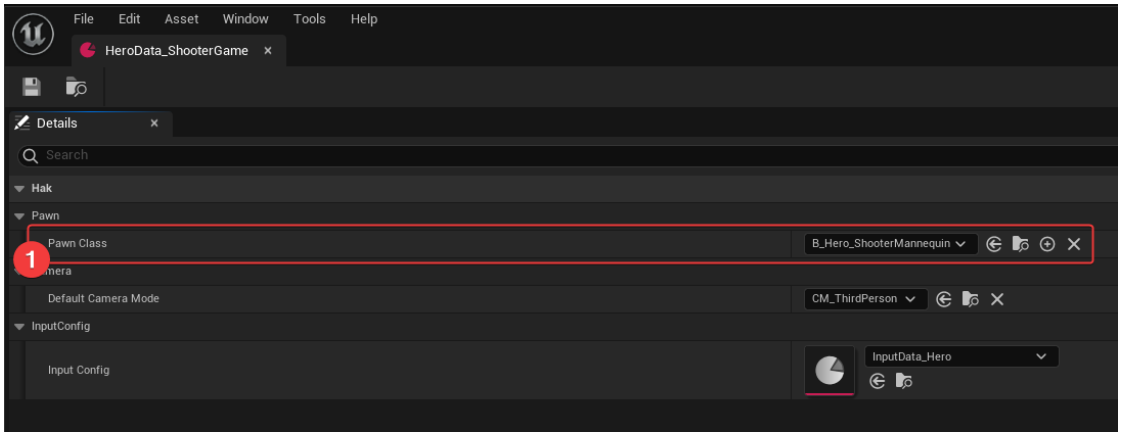
- [ ] B_Hero_ShooterMannequin을 생성하자:
  - 아래의 경로에 위치하자:



- [ ] B_Hero_Default를 상속받아 B_Hero_ShooterMannequin을 생성하자:



☐ B_Hero_ShooterMannequin을 PawnClass로 HeroData_ShooterGame에 추가하자:



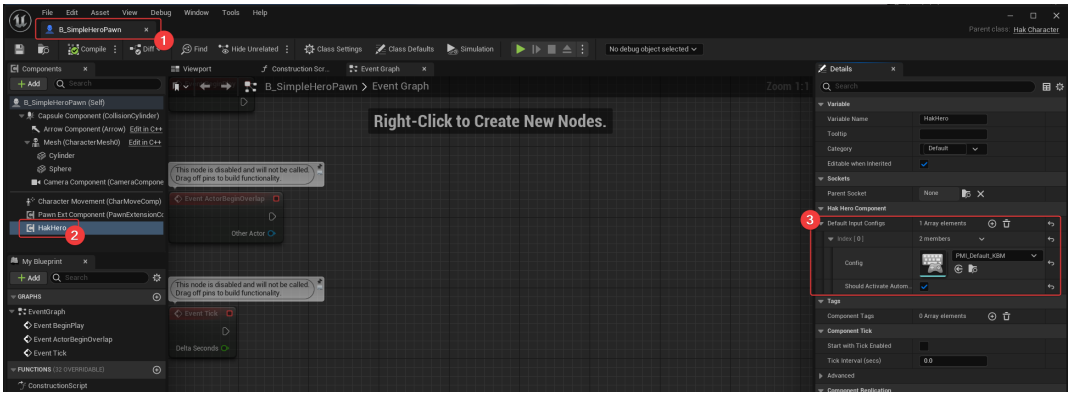
☐ 기존 SimpleHeroPawn을 회고해보면, HeroComponent에서 Input을 설정하는데, 디버깅해보자:

```cpp
void UHakHeroComponent::InitializePlayerInput(UInputComponent* PlayerInputComponent)
{
    check(PlayerInputComponent);

    const APawn* Pawn = GetPawn<APawn>();
    if (!Pawn)
    {
        return;
    }

    // LocalPlayer를 가져오기 위해
    const APlayerController* PC = GetController<APlayerController>();
    check(PC);

    // EnhancedInputLocalPlayerSubsystem 가져오기 위해
    const ULocalPlayer* LP = PC->GetLocalPlayer();
    check(LP);

    UEnhancedInputLocalPlayerSubsystem* Subsystem = LP->GetSubsystem<UEnhancedInputLocalPlayerSubsystem>();
    check(Subsystem);

    // EnhancedInputLocalPlayerSubsystem에 MappingContext를 비워준다:
    Subsystem->ClearAllMappings();

    // PawnExtensionComponent -> PawnData -> InputConfig 존재 유무 판단:
    if (const UHakPawnExtensionComponent* PawnExtComp = UHakPawnExtensionComponent::FindPawnExtensionComponent(Pawn))
    {
        if (const UHakPawnData* PawnData = PawnExtComp->GetPawnData<UHakPawnData>())
        {
            if (const UHakInputConfig* InputConfig = PawnData->InputConfig)
            {
                const FHakGameplayTags& GameplayTags = FHakGameplayTags::Get();

                // HeroComponent 가지고 있는 Input Mapping Context를 순회하며, Enh
                for (const FHakMappableConfigPair& Pair : DefaultInputConfigs)
                {
                    if (Pair.bShouldActivateAutomatically)
                    {
                        FModifyContextOptions Options = {};
                        Options.bIgnoreAllPressedKeysUntilRelease = false;

                        // 내부적으로 Input Mapping Context를 추가한다:
                        // - AddPlayerMappableConfig를 간단히 보는 것을 추천
                        Subsystem->AddPlayerMappableConfig(Pair.Config.LoadSynchronous(), Options);
                    }
                }
```

1 DefaultInputConfigs가 비어있다!

2 PlayerMappableConfig가 추가되지 않는다...

- 이전에는 아래와 같이 우리가 임의적으로 아래와 같이 DefaultInputConfigs에 추가해줬다:



- 우리는 이전과 달리, 이번에는 **<u>GameFeatureAction을 활용하여, InputConfig를 추가한다</u>**

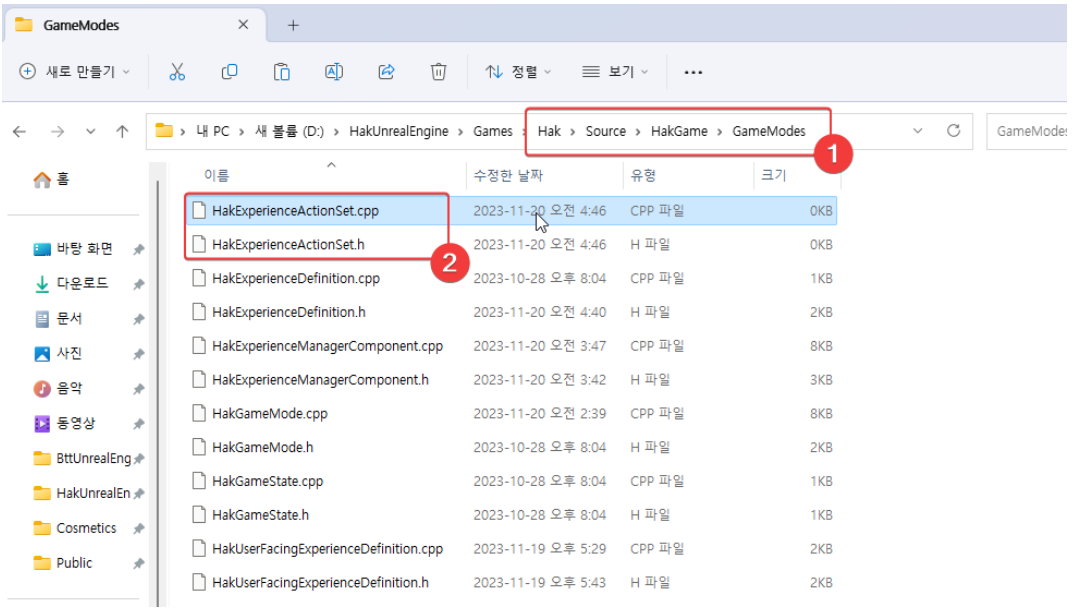
GameFeatureAction

▼ **<u>펼치기</u>**

☐ ExperienceDefinition에 GameFeatureAction을 추가하자:

```cpp
#pragma once

#include "Engine/DataAsset.h"
#include "HakExperienceDefinition.generated.h"

/** forward declaration */
class UHakPawnData;
class UGameFeatureAction;
class UHakExperienceActionSet;

/**
 * UHakExperienceDefinition
 * - definition of an experience
 */
UCLASS()
class UHakExperienceDefinition : public UPrimaryDataAsset
{
    GENERATED_BODY()
public:
    UHakExperienceDefinition(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** the default pawn class to spawn for players */
    UPROPERTY(EditDefaultsOnly, Category=Gameplay)
    TObjectPtr<UHakPawnData> DefaultPawnData;

    /** lost if game feature plugins this experience wants to have active */
    // 해당 property는 단순히 마킹 및 기억용으로 남겨놓도록 하겠다:
    // - GameMode에 따라 필요한 GameFeature plugin들을 로딩하는데 이에 대한 연결고리로 생각하면 된다 (현재는 쓰지 않음)
    UPROPERTY(EditDefaultsOnly, Category=Gameplay)
    TArray<FString> GameFeaturesToEnable;

    /** ExperienceActionSet은 UGameFeatureAction의 Set이며, Gameplay 용도에 맞게 분류의 목적으로 사용한다 */
    UPROPERTY(EditDefaultsOnly, Category = Gameplay)
    TArray<TObjectPtr<UHakExperienceActionSet>> ActionSets;

    /** 일반적인 GameFeatureAction으로서 추가 */
    UPROPERTY(EditDefaultsOnly, Category="Actions")
    TArray<TObjectPtr<UGameFeatureAction>> Actions;
};
```

- [ ] HakExperienceActionSet 추가:
  - 아래의 경로에 파일 추가:



- [ ] HakExperienceActionSet.h/.cpp:

```h
#pragma once

#include "CoreMinimal.h"
#include "Engine/DataAsset.h"
#include "HakExperienceActionSet.generated.h"

class UGameFeatureAction;

/**
 * DataAsset로서 UGameFeatureAction을 카테고리화 시킬 때, 유용한 클래스
 */
UCLASS(BlueprintType)
class UHakExperienceActionSet : public UPrimaryDataAsset
{
    GENERATED_BODY()
public:
    UHakExperienceActionSet();

    /**
     * member variables
     */
    UPROPERTY(EditAnywhere, Category="Actions to Perform")
    TArray<TObjectPtr<UGameFeatureAction>> Actions;
};
```

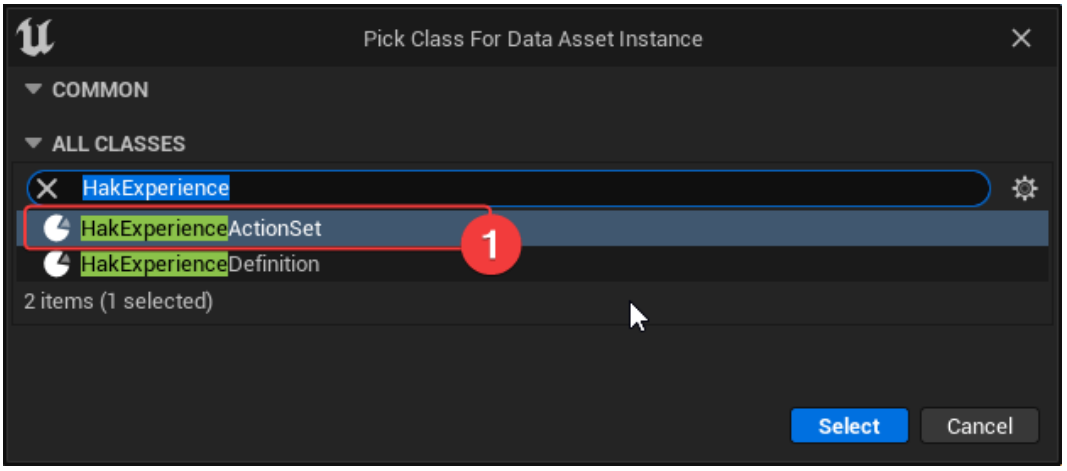
```cpp
#include "HakExperienceActionSet.h"
#include "GameFeatureAction.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakExperienceActionSet)

UHakExperienceActionSet::UHakExperienceActionSet()
    : Super()
{}
```

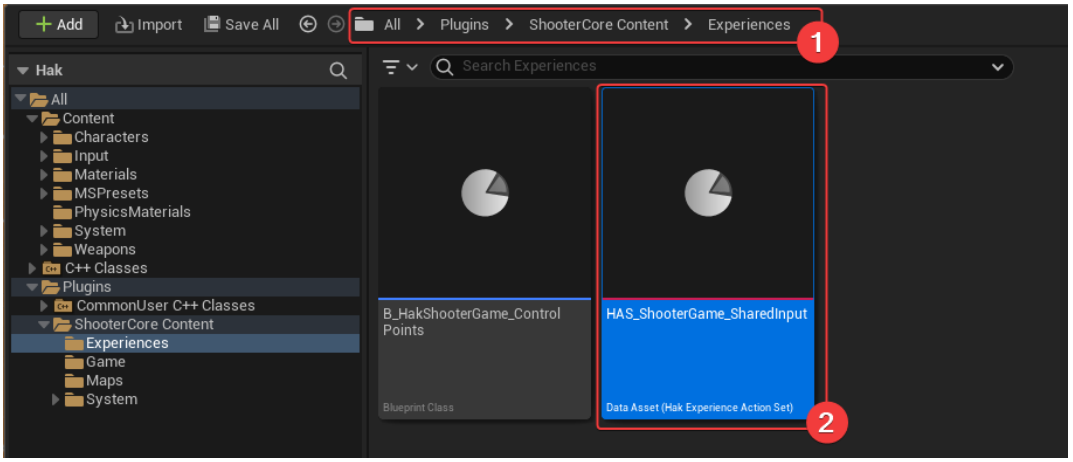
- [ ] HakExperienceDefinition.cpp에 헤더파일 추가:

```cpp
#include "HakExperienceDefinition.h"
#include "HakExperienceActionSet.h"
#include "GameFeatureAction.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakExperienceDefinition)

UHakExperienceDefinition::UHakExperienceDefinition(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{

}
```

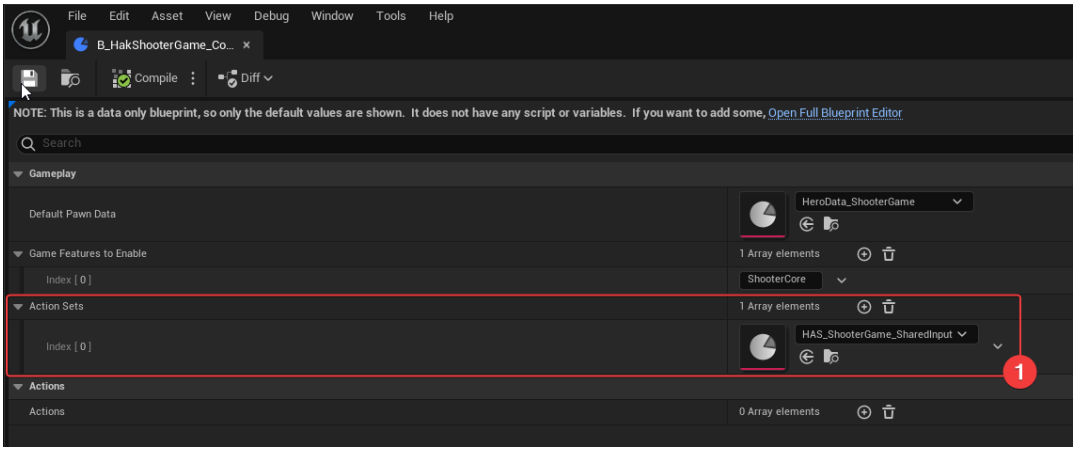
1

- [ ] ExperienceActionSet을 추가하자:
  - 아래의 경로에 HAS_ShooterGame_SharedInput을 추가하자:





- [ ] B_HakShooterGame_ControlPoints에 HAS_ShooterGame_SharedInput을 추가:



- InputConfig의 커스텀 GameFeatureAction 타입을 만들기에 앞서, GameFeatureAction을 활성화하는 로직부터 추가하자:
  - 앞서, GameFeature Plugin 로딩하고 활성화하는 로직을 구현하였다
  - GameFeatureAction 활성화는 이 과정 이후, 명시적으로 진행하면 된다

- [ ] 아래와 같이, OnExperienceFullLoadCompleted()에 추가 구현해주자:
  - [ ] HakExperienceManagerComponent.cpp에 헤더 파일 추가시키기:

```cpp
#include "HakExperienceManagerComponent.h"
#include "HakExperienceDefinition.h"
#include "HakExperienceActionSet.h"
#include "GameFeaturesSubsystem.h"
#include "GameFeaturesSubsystemSettings.h"
#include "HakGame/System/HakAssetManager.h"
#include "Net/UnrealNetwork.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakExperienceManagerComponent)
```

1

- [ ] 마저 구현하기:

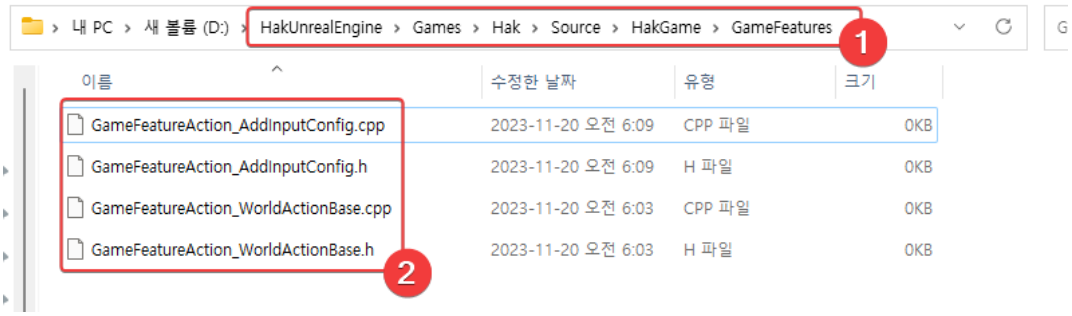
```cpp
void UHakExperienceManagerComponent::OnExperienceFullLoadCompleted()
{
    check(LoadState != EHakExperienceLoadState::Loaded);

    // GameFeature Plugin의 로딩과 활성화 이후, GameFeature Action들을 활성화 시키자:
    {
        LoadState = EHakExperienceLoadState::ExecutingActions;

        // GameFeatureAction 활성화를 위한 Context 준비
        FGameFeatureActivatingContext Context;
        {
            // 월드의 핸들을 세팅해준다
            const FWorldContext* ExistingWorldContext = GEngine->GetWorldContextFromWorld(GetWorld());
            if (ExistingWorldContext)
            {
                Context.SetRequiredWorldContextHandle(ExistingWorldContext->ContextHandle);
            }
        }

        auto ActivateListOfActions = [&Context](const TArray<UGameFeatureAction*>& ActionList)
        {
            for (UGameFeatureAction* Action : ActionList)
            {
                // 명시적으로 GameFeatureAction에 대해 Registering -> Loading -> Activating 순으로 호출한다
                if (Action)
                {
                    Action->OnGameFeatureRegistering();
                    Action->OnGameFeatureLoading();
                    Action->OnGameFeatureActivating(Context);
                }
            }
        };

        // 1. Experience의 Actions
        ActivateListOfActions(CurrentExperience->Actions);

        // 2. Experience의 ActionSets
        for (const TObjectPtr<UHakExperienceActionSet>& ActionSet : CurrentExperience->ActionSets)
        {
            ActivateListOfActions(ActionSet->Actions);
        }
    }

    LoadState = EHakExperienceLoadState::Loaded;
    OnExperienceLoaded.Broadcast(CurrentExperience);
    OnExperienceLoaded.Clear();
}
```

1

GameFeatureAction_AddInputConfig

▼ **펼치기**

- [ ] GameFeatureAction_WorldActionBase와 _AddInputConfig 파일 추가:
  - 아래의 경로에 추가하기:



- [ ] GameFeatureAction_WorldActionBase

```cpp
#pragma once

#include "Containers/Map.h"
#include "GameFeatureAction.h"
#include "GameFeaturesSubsystem.h"
#include "GameFeatureAction_WorldActionBase.generated.h"

class FDelegateHandle;
class UGameInstance;
struct FGameFeatureActivatingContext;
struct FGameFeatureDeactivatingContext;
struct FWorldContext;

UCLASS(Abstract)
class UGameFeatureAction_WorldActionBase : public UGameFeatureAction
{
    GENERATED_BODY()
public:
    /**
     * UGameFeatureAction's interface
     */
    virtual void OnGameFeatureActivating(FGameFeatureActivatingContext& Context) override;

    /**
     * interface
     */
    virtual void AddToWorld(const FWorldContext& WorldContext, const FGameFeatureStateChangeContext& ChangeContext) PURE_VI
};
```

```cpp
#include "GameFeatureAction_WorldActionBase.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(GameFeatureAction_WorldActionBase)

void UGameFeatureAction_WorldActionBase::OnGameFeatureActivating(FGameFeatureActivatingContext& Context)
{
    // 월드를 순회하면서,
    for (const FWorldContext& WorldContext : GEngine->GetWorldContexts())
    {
        // 앞서, ExperienceManagerComponent에서 GameFeatureAction을 활성화하면서, Context에 World를 넣어줌
        // - 이를 통해 적용할 대상인지 판단
        if (Context.ShouldApplyToWorldContext(WorldContext))
        {
            // WorldActionBase의 Interface인 AddToWorld 호출
            AddToWorld(WorldContext, Context);
        }
    }
}
```

- [ ] GameFeatureAction_AddInputConfig:
  - [ ] Reset() 함수 구현:

```cpp
void UGameFeatureAction_AddInputConfig::Reset(FPerContextData& ActiveData)
{
    // ExtensionRequestHandles을 초기화
    ActiveData.ExtensionRequestHandles.Empty();

    // PawnsAddedTo에 대해서, 하나씩 Stack 방식으로 위에서 아래로 직접 InputConfig를 제거 진행
    while (!ActiveData.PawnsAddedTo.IsEmpty())
    {
        TWeakObjectPtr<APawn> PawnPtr = ActiveData.PawnsAddedTo.Top();
        if (PawnPtr.IsValid())
        {
            RemoveInputConfig(PawnPtr.Get(), ActiveData);
        }
        else
        {
            // WeakObjectPtr로 PawnsAddedTo를 관리하고 있기 때문에, GC되었다면, nullptr일 수 있음
            ActiveData.PawnsAddedTo.Pop();
        }
    }
}
```

☐ AddInputConfig/RemoveInputConfig 구현:

```cpp
void UGameFeatureAction_AddInputConfig::AddInputConfig(APawn* Pawn, FPerContextData& ActiveData)
{
    APlayerController* PlayerController = Cast<APlayerController>(Pawn->GetController());
    if (ULocalPlayer* LP = PlayerController ? PlayerController->GetLocalPlayer() : nullptr)
    {
        if (UEnhancedInputLocalPlayerSubsystem* Subsystem = LP->GetSubsystem<UEnhancedInputLocalPlayerSubsystem>())
        {
            FModifyContextOptions Options = {};
            Options.bIgnoreAllPressedKeysUntilRelease = false;

            // 추가된 InputConfigs를 순회하며, EnhancedInputSubsystem에 PlayerMappableConfig를 직접 추가
            for (const FHakMappableConfigPair& Pair : InputConfigs)
            {
                if (Pair.bShouldActivateAutomatically)
                {
                    Subsystem->AddPlayerMappableConfig(Pair.Config.LoadSynchronous(), Options);
                }
            }

            // ActiveData에 Pawn을 관리대상으로 등록
            ActiveData.PawnsAddedTo.AddUnique(Pawn);
        }
    }
}

void UGameFeatureAction_AddInputConfig::RemoveInputConfig(APawn* Pawn, FPerContextData& ActiveData)
{
    APlayerController* PlayerController = Cast<APlayerController>(Pawn->GetController());
    if (ULocalPlayer* LP = PlayerController ? PlayerController->GetLocalPlayer() : nullptr)
    {
        if (UEnhancedInputLocalPlayerSubsystem* Subsystem = LP->GetSubsystem<UEnhancedInputLocalPlayerSubsystem>())
        {
            // InputConfigs를 순회하며, Config를 제거 진행
            for (const FHakMappableConfigPair& Pair : InputConfigs)
            {
                Subsystem->RemovePlayerMappableConfig(Pair.Config.LoadSynchronous());
            }

            ActiveData.PawnsAddedTo.Remove(Pawn);
        }
    }
}
```

☐ OnGameFeatureActivating()

```cpp
void UGameFeatureAction_AddInputConfig::OnGameFeatureActivating(FGameFeatureActivatingContext& Context)
{
    FPerContextData& ActiveData = ContextData.FindOrAdd(Context);
    if (!ensure(ActiveData.ExtensionRequestHandles.IsEmpty()) ||
        !ensure(ActiveData.PawnsAddedTo.IsEmpty()))
    {
        Reset(ActiveData);
    }

    // UGameFeatureAction_WorldActionBase를 호출하면서, AddToWorld() 호출!
    Super::OnGameFeatureActivating(Context);
}
```

- [ ] AddToWorld
  - [ ] HandlePawnExtension 구현:

```cpp
void UGameFeatureAction_AddInputConfig::HandlePawnExtension(AActor* Actor, FName EventName, FGameFeatureStateChangeContext ChangeContext)
{
    APawn* AsPawn = CastChecked<APawn>(Actor);
    FPerContextData& ActiveData = ContextData.FindOrAdd(ChangeContext);

    if (EventName == UGameFrameworkComponentManager::NAME_ExtensionAdded)
    {
        AddInputConfig(AsPawn, ActiveData);
    }
    else if (EventName == UGameFrameworkComponentManager::NAME_ExtensionRemoved)
    {
        RemoveInputConfig(AsPawn, ActiveData);
    }
}
```

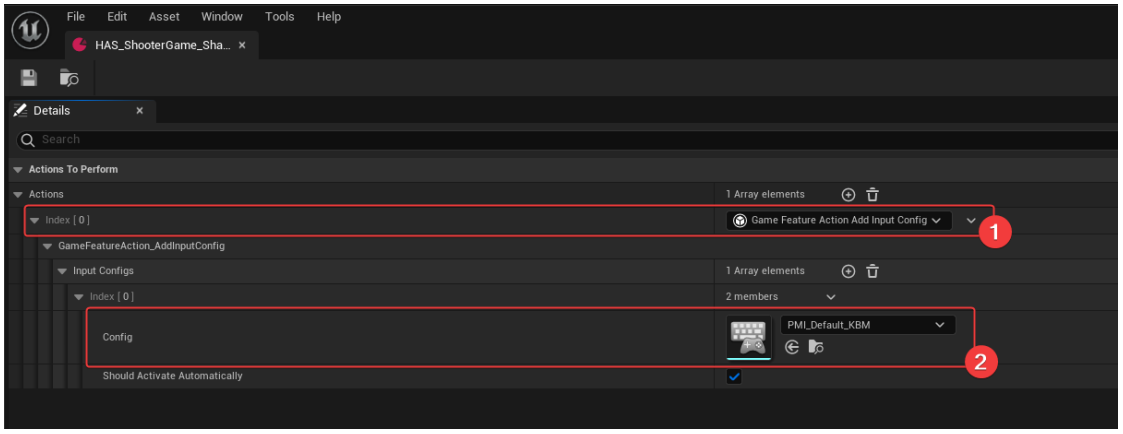
  - [ ] AddToWorld 구현:

```cpp
void UGameFeatureAction_AddInputConfig::AddToWorld(const FWorldContext& WorldContext, const FGameFeatureStateChangeContext& ChangeContext)
{
    UWorld* World = WorldContext.World();
    UGameInstance* GameInstance = WorldContext.OwningGameInstance;
    FPerContextData& ActiveData = ContextData.FindOrAdd(ChangeContext);
    if (GameInstance && World && World->IsGameWorld())
    {
        // GFCM을 이용하여, ExtensionHandler를 추가하여 등록 진행:
        // - HandlePawnExtension 콜백함수로 연결
        if (UGameFrameworkComponentManager* ComponentMan = UGameInstance::GetSubsystem<UGameFrameworkComponentManager>(GameInstance))
        {
            UGameFrameworkComponentManager::FExtensionHandlerDelegate AddConfigDelegate =
                UGameFrameworkComponentManager::FExtensionHandlerDelegate::CreateUObject(this, &ThisClass::HandlePawnExtension, ChangeContext);

            // 등록된 콜백 함수의 핸들을 ActiveData의 ExtensionRequestHandles에 등록
            TSharedPtr<FComponentRequestHandle> ExtensionRequestHandle = ComponentMan->AddExtensionHandler(APawn::StaticClass(), AddConfigDelegate);
            ActiveData.ExtensionRequestHandles.Add(ExtensionRequestHandle);
        }
    }
}
```

- [ ] OnGameFeatureDeactivating()

```cpp
void UGameFeatureAction_AddInputConfig::OnGameFeatureDeactivating(FGameFeatureDeactivatingContext& Context)
{
    Super::OnGameFeatureDeactivating(Context);
    FPerContextData* ActiveData = ContextData.Find(Context);
    if (ensure(ActiveData))
    {
        Reset(*ActiveData);
    }
}
```

- [ ] 앞서 정의했던 HAS_ShooterGame_SharedInput에 AddInputConfig를 추가해주자:



- [ ] 아직 우리에게 보이는 캐릭터가 없으니, 로그를 남기도록 해보자:
  - HeroComponent의 Input_Move에 로그를 임시적으로 추가하자:

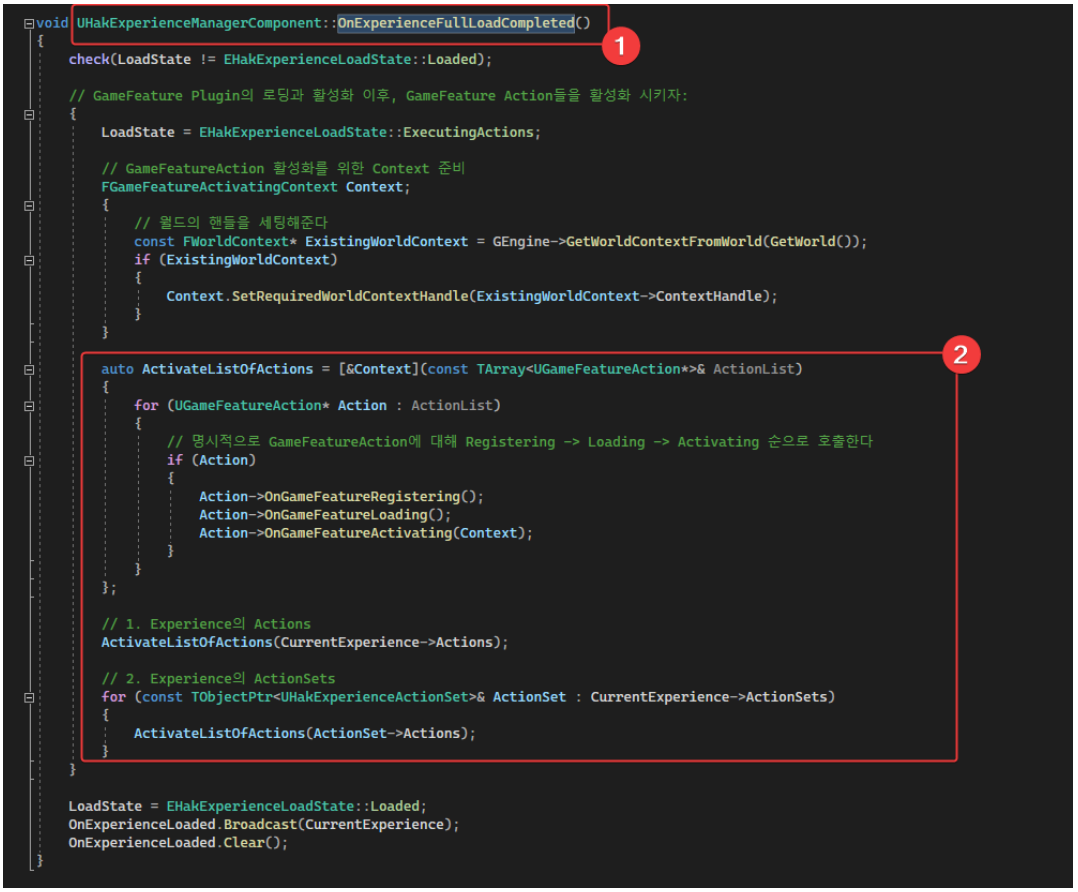
```cpp
void UHakHeroComponent::Input_Move(const FInputActionValue& InputActionValue)
{
    APawn* Pawn = GetPawn<APawn>();
    AController* Controller = Pawn ? Pawn->GetController() : nullptr;

    if (Controller)
    {
        const FVector2D Value = InputActionValue.Get<FVector2D>();

        bool bLogging = true;
        if (bLogging)
        {
            UE_LOG(LogHak, Log, TEXT("Input_Move[X=%.2f][Y=%.2f]"), Value.X, Value.Y);
        }

        const FRotator MovementRotation(0.0f, Controller->GetControlRotation().Yaw, 0.0f);

        if (Value.X != 0.0f)
        {
            // Left/Right -> X 값에 들어있음:
            // MovementDirection은 현재 카메라의 RightVector를 의미함 (World-Space)
            const FVector MovementDirection = MovementRotation.RotateVector(FVector::RightVector);

            // AddMovementInput 함수를 한번 보자:
            // - 내부적으로 MovementDirection * Value.X를 MovementComponent에 적용(더하기)해준다
            Pawn->AddMovementInput(MovementDirection, Value.X);
        }

        if (Value.Y != 0.0f) // 앞서 우리는 Forward 적용을 위해 swizzle input modifier를 사용했다~
        {
            // 앞서 Left/Right와 마찬가지로 Forward/Backward를 적용한다
            const FVector MovementDirection = MovementRotation.RotateVector(FVector::ForwardVector);
            Pawn->AddMovementInput(MovementDirection, Value.Y);
        }
    }
}
```

- [ ] 로그가 생성되는지 확인하자:

```cpp
256  void UHakHeroComponent::Input_Move(const FInputActionValue& InputActionValue)
257  {
258      APawn* Pawn = GetPawn<APawn>();
259      AController* Controller = Pawn ? Pawn->GetController() : nullptr;
260
261      if (Controller)
262      {
263          FVector2D Value = InputActionValue.Get<FVector2D>();
264
265          bool bLogging = true;
266          if (bLogging)
267          {
268              UE_LOG(LogHak, Log, TEXT("Input_Move[X=%.2f][Y=%.2f]"), Value.X, Value.Y);
269          }
270
271          const FRotator MovementRotation(0.0f, Controller->GetControlRotation().Yaw, 0.0f);
272
273          if (Value.X != 0.0f)
274          {
```

1 로그가 찍히지 않는다...

- [ ] 하나씩 순차적으로 디버깅해보자:
  - [ ] 우선 GameFeatureAction을 활성화하는 ExperienceManagerComponent부터 보자:




```cpp
void UHakExperienceManagerComponent::OnExperienceFullLoadCompleted()
{
    check(LoadState != EHakExperienceLoadState::Loaded);

    // GameFeature Plugin의 로딩과 활성화 이후, GameFeature Action들을 활성화 시키자:
    {
        LoadState = EHakExperienceLoadState::ExecutingActions;

        // GameFeatureAction 활성화를 위한 Context 준비
        FGameFeatureActivatingContext Context;
        {
            // 월드의 핸들을 세팅해준다
            const FWorldContext* ExistingWorldContext = GEngine->GetWorldContextFromWorld(GetWorld());
            if (ExistingWorldContext)
            {
                Context.SetRequiredWorldContextHandle(ExistingWorldContext->ContextHandle);
            }
        }

        auto ActivateListOfActions = [&Context](const TArray<UGameFeatureAction*>& ActionList)
        {
            for (UGameFeatureAction* Action : ActionList)
            {
                // 명시적으로 GameFeatureAction에 대해 Registering -> Loading -> Activating 순으로 호출한다
                if (Action)
                {
                    Action->OnGameFeatureRegistering();
                    Action->OnGameFeatureLoading();
                    Action->OnGameFeatureActivating(Context);
                }
            }
        };

        // 1. Experience의 Actions
        ActivateListOfActions(CurrentExperience->Actions);

        // 2. Experience의 ActionSets
        for (const TObjectPtr<UHakExperienceActionSet>& ActionSet : CurrentExperience->ActionSets)
        {
            ActivateListOfActions(ActionSet->Actions);
        }
    }

    LoadState = EHakExperienceLoadState::Loaded;
    OnExperienceLoaded.Broadcast(CurrentExperience);
    OnExperienceLoaded.Clear();
}
```

- [ ] PRAGMA_DISABLE_OPTIMIZATION을 넣어서 디버깅에 용의하게 하자:
- [ ] OnExperienceFullLoadCompleted():

```cpp
PRAGMA_DISABLE_OPTIMIZATION
void UHakExperienceManagerComponent::OnExperienceFullLoadCompleted()
{
    check(LoadState != EHakExperienceLoadState::Loaded);

    // GameFeature Plugin의 로딩과 활성화 이후, GameFeature Action들을 활성화 시키자:
    {
        LoadState = EHakExperienceLoadState::ExecutingActions;

        // GameFeatureAction 활성화를 위한 Context 준비
        FGameFeatureActivatingContext Context;
        {
            // 월드의 핸들을 세팅해준다
            const FWorldContext* ExistingWorldContext = GEngine->GetWorldContextFromWorld(GetWorld());
            if (ExistingWorldContext)
            {
                Context.SetRequiredWorldContextHandle(ExistingWorldContext->ContextHandle);
            }
        }

        auto ActivateListOfActions = [&Context](const TArray<UGameFeatureAction*>& ActionList)
        {
            for (UGameFeatureAction* Action : ActionList)
            {
                // 명시적으로 GameFeatureAction에 대해 Registering -> Loading -> Activating 순으로 호출한다
                if (Action)
                {
                    Action->OnGameFeatureRegistering();
                    Action->OnGameFeatureLoading();
                    Action->OnGameFeatureActivating(Context);
                }
            }
        };

        // 1. Experience의 Actions
        ActivateListOfActions(CurrentExperience->Actions);

        // 2. Experience의 ActionSets
        for (const TObjectPtr<UHakExperienceActionSet>& ActionSet : CurrentExperience->ActionSets)
        {
            ActivateListOfActions(ActionSet->Actions);
        }
    }

    LoadState = EHakExperienceLoadState::Loaded;
    OnExperienceLoaded.Broadcast(CurrentExperience);
    OnExperienceLoaded.Clear();
}
PRAGMA_ENABLE_OPTIMIZATION
```

1

☐ UGameFeatureAction_AddInputConfig::OnGameFeatureActivating():

```cpp
void UGameFeatureAction_AddInputConfig::OnGameFeatureActivating(FGameFeatureActivatingContext& Context)
{
    FPerContextData& ActiveData = ContextData.FindOrAdd(Context);  ≤ 2ms elapsed
    if (!ensure(ActiveData.ExtensionRequestHandles.IsEmpty()) ||
        !ensure(ActiveData.PawnsAddedTo.IsEmpty()))
    {
        Reset(ActiveData);
    }

    // UGameFeatureAction_WorldActionBase를 호출하면서, AddToWorld() 호출!
    Super::OnGameFeatureActivating(Context);
}
```

☐ UGameFeatureAction_WorldActionBase::OnGameFeatureActivating():

```cpp
void UGameFeatureAction_WorldActionBase::OnGameFeatureActivating(FGameFeatureActivatingContext& Context)
{
    // 월드를 순회하면서,
    for (const FWorldContext& WorldContext : GEngine->GetWorldContexts())  ≤ 2ms elapsed
    {
        // 앞서, ExperienceManagerComponent에서 GameFeatureAction을 활성화하면서, Context에 World를 넣어줌
        // - 이를 통해 적용할 대상인지 판단
        if (Context.ShouldApplyToWorldContext(WorldContext))
        {
            // WorldActionBase의 Interface인 AddToWorld 호출
            AddToWorld(WorldContext, Context);
        }
    }
}
```

☐ UGameFeatureAction_AddInputConfig::AddToWorld():

```cpp
void UGameFeatureAction_AddInputConfig::AddToWorld(const FWorldContext& WorldContext, const FGameFeatureStateChangeContext& ChangeContext)
{
    UWorld* World = WorldContext.World(); ≤ 4ms elapsed
    UGameInstance* GameInstance = WorldContext.OwningGameInstance;
    FPerContextData& ActiveData = ContextData.FindOrAdd(ChangeContext);
    if (GameInstance && World && World->IsGameWorld())
    {
        // GFCM을 이용하여, ExtensionHandler를 추가하여 등록 진행:
        // - HandlePawnExtension 콜백함수로 연결
        if (UGameFrameworkComponentManager* ComponentMan = UGameInstance::GetSubsystem<UGameFrameworkComponentManager>(GameInstance))
        {
            UGameFrameworkComponentManager::FExtensionHandlerDelegate AddConfigDelegate =
                UGameFrameworkComponentManager::FExtensionHandlerDelegate::CreateUObject(this, &ThisClass::HandlePawnExtension, ChangeContext);

            // 등록된 콜백 함수의 핸들을 ActiveData의 ExtensionRequestHandles에 등록
            TSharedPtr<FComponentRequestHandle> ExtensionRequestHandle = ComponentMan->AddExtensionHandler(APawn::StaticClass(), AddConfigDelegate);
            ActiveData.ExtensionRequestHandles.Add(ExtensionRequestHandle);
        }
    }
}
```

☐ UGameFeatureAction_AddInputConfig::HandlePawnExtension():

```cpp
void UGameFeatureAction_AddInputConfig::HandlePawnExtension(AActor* Actor, FName EventName, FGameFeatureStateChangeContext ChangeContext)
{
    APawn* AsPawn = CastChecked<APawn>(Actor);
    FPerContextData& ActiveData = ContextData.FindOrAdd(ChangeContext); ≤ 1ms elapsed

    if (EventName == UGameFrameworkComponentManager::NAME_ExtensionAdded)
    {
        AddInputConfig(AsPawn, ActiveData);
    }
    else if (EventName == UGameFrameworkComponentManager::NAME_ExtensionRemoved)
    {
        RemoveInputConfig(AsPawn, ActiveData);
    }
}
```

- **<u>우리가 원하는 Actor인 B_Hero_ShooterMannequin이 아니다…</u>**

- UGameFrameworkComponentManager::SendGameFrameworkComponentExtensionEvent()를 활용하여, **<u>직접적으로 _AddInputConfig의 GameFeatureAction의 AddInputConfig를 활성화 해야 한다!</u>**

☐ NAME_BindInputsNow 정의하자:

```cpp
/**
 * component that sets up input and camera handling for player controlled pawns (or bots that simulate players)
 * - this depends on a PawnExtensionComponent to coordinate initialization
 *
 * 카메라, 입력 등 플레이어가 제어하는 시스템의 초기화를 처리하는 컴포넌트
 */
UCLASS(Blueprintable, Meta=(BlueprintSpawnableComponent))
class UHakHeroComponent : public UPawnComponent, public IGameFrameworkInitStateInterface
{
    GENERATED_BODY()
public:
    UHakHeroComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** FeatureName 정의 */
    static const FName NAME_ActorFeatureName;

    /** Extension Event 이름 정의 */
    static const FName NAME_BindInputsNow;  1

    /**
```

```cpp
#include "HakHeroComponent.h"
#include "HakPawnExtensionComponent.h"
#include "EnhancedInputSubsystems.h"
#include "PlayerMappableInputConfig.h"
#include "HakGame/HakLogChannels.h"
#include "HakGame/HakGameplayTags.h"
#include "HakGame/Player/HakPlayerState.h"
#include "HakGame/Player/HakPlayerController.h"
#include "HakGame/Character/HakPawnData.h"
#include "HakGame/Camera/HakCameraComponent.h"
#include "HakGame/Input/HakMappableConfigPair.h"
#include "HakGame/Input/HakInputComponent.h"
#include "Components/GameFrameworkComponentManager.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakHeroComponent)

/** FeatureName 정의: static member variable 초기화 */
const FName UHakHeroComponent::NAME_ActorFeatureName("Hero");

/** InputConfig의 GameFeatureAction 활성화 ExtensioEvent 이름 */
const FName UHakHeroComponent::NAME_BindInputsNow("BindInputsNow");

UHakHeroComponent::UHakHeroComponent(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
    PrimaryComponentTick.bStartWithTickEnabled = false;
    PrimaryComponentTick.bCanEverTick = false;
}
```

1

☐ SendGameFrameworkComponentExtensionEvent() 추가:

```cpp
void UHakHeroComponent::InitializePlayerInput(UInputComponent* PlayerInputComponent)
{
    check(PlayerInputComponent);

    const APawn* Pawn = GetPawn<APawn>();
    if (!Pawn)
    {
        return;
    }

    // LocalPlayer를 가져오기 위해
    const APlayerController* PC = GetController<APlayerController>();
    check(PC);

    // EnhancedInputLocalPlayerSubsystem 가져오기 위해
    const ULocalPlayer* LP = PC->GetLocalPlayer();
    check(LP);

    UEnhancedInputLocalPlayerSubsystem* Subsystem = LP->GetSubsystem<UEnhancedInputLocalPlayerSubsystem>();
    check(Subsystem);

    // EnhancedInputLocalPlayerSubsystem에 MappingContext를 비워준다:
    Subsystem->ClearAllMappings();

    // PawnExtensionComponent -> PawnData -> InputConfig 존재 유무 판단:
    if (const UHakPawnExtensionComponent* PawnExtComp = UHakPawnExtensionComponent::FindPawnExtensionComponent(Pawn))
    {
        if (const UHakPawnData* PawnData = PawnExtComp->GetPawnData<UHakPawnData>())
        {
            if (const UHakInputConfig* InputConfig = PawnData->InputConfig)
            {
                const FHakGameplayTags& GameplayTags = FHakGameplayTags::Get();

                // HeroComponent 가지고 있는 Input Mapping Context를 순회하며, EnhancedInputLocalPlayerSubsystem에 추가한다
                for (const FHakMappableConfigPair& Pair : DefaultInputConfigs)
                {
                    if (Pair.bShouldActivateAutomatically)
                    {
                        FModifyContextOptions Options = {};
                        Options.bIgnoreAllPressedKeysUntilRelease = false;

                        // 내부적으로 Input Mapping Context를 추가한다:
                        // - AddPlayerMappableConfig를 간단히 보는 것을 추천
                        Subsystem->AddPlayerMappableConfig(Pair.Config.LoadSynchronous(), Options);
                    }
                }

                UHakInputComponent* HakIC = CastChecked<UHakInputComponent>(PlayerInputComponent);
                {
                    // InputTag_Move와 InputTag_Look_Mouse에 대해 각각 Input_Move()와 Input_LookMouse() 멤버 함수에 바인딩시킨다:
                    // - 바인딩한 이후, Input 이벤트에 따라 멤버 함수가 트리거된다
                    HakIC->BindNativeAction(InputConfig, GameplayTags.InputTag_Move, ETriggerEvent::Triggered, this, &ThisClass::Input_Move, false);
                    HakIC->BindNativeAction(InputConfig, GameplayTags.InputTag_Look_Mouse, ETriggerEvent::Triggered, this, &ThisClass::Input_LookMouse, false);
                }
            }
        }
    }

    // GameFeatureAction_AddInputConfig의 HandlePawnExtension 콜백 함수 전달
    UGameFrameworkComponentManager::SendGameFrameworkComponentExtensionEvent(const_cast<APawn>(Pawn), NAME_BindInputsNow);
}
```

1

☐ HandlePawnExtension 수정:

```cpp
void UGameFeatureAction_AddInputConfig::HandlePawnExtension(AActor* Actor, FName EventName, FGameFeatureStateChangeContext ChangeContext)
{
    APawn* AsPawn = CastChecked<APawn>(Actor);
    FPerContextData& ActiveData = ContextData.FindOrAdd(ChangeContext);

    if (EventName == UGameFrameworkComponentManager::NAME_ExtensionAdded || EventName == UHakHeroComponent::NAME_BindInputsNow)
    {
        AddInputConfig(AsPawn, ActiveData);
    }
    else if (EventName == UGameFrameworkComponentManager::NAME_ExtensionRemoved || EventName == UHakHeroComponent::NAME_BindInputsNow)
    {
        RemoveInputConfig(AsPawn, ActiveData);
    }
}
```

- 카메라가 잘 움직이는 것을 확인하자!