10주차 (2023.12.04)

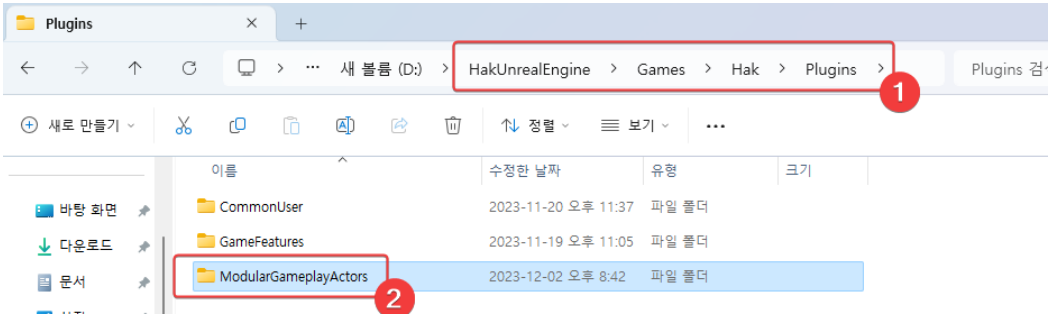
ModularGameplayActors:

▼ **펼치기**

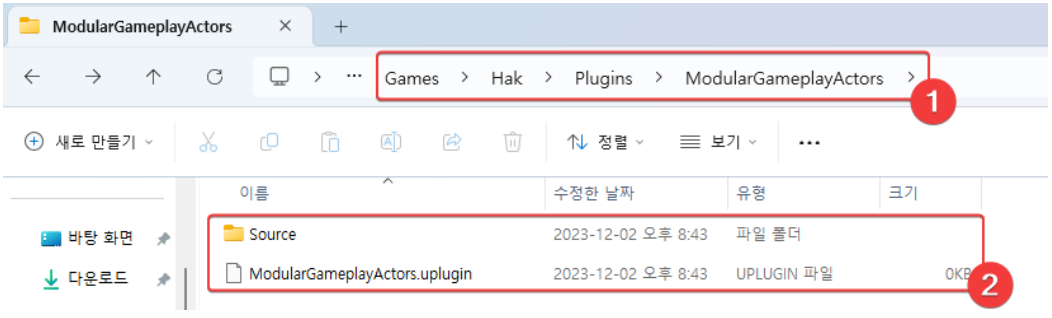
- 우리는 현재 기본 GameFramework에서 제공하는 Actor를 사용하였다:
  - GameFeatureAction을 활용한 동적으로 GameFeature Plugin에 정의된 Component를 Actor에 붙이려면, UGameFeatureFrameworkManager에 Receiver로 등록해야 한다
  - Lyra에서는 이를 용이하게 하기 위해, 따로, ModularGameplayActors의 Plugin을 정의하여 ModularActor와 같은 형태의 클래스로 만들어 Interface를 래핑하였다.

☐ ModularGameplayActors.plugin 생성:

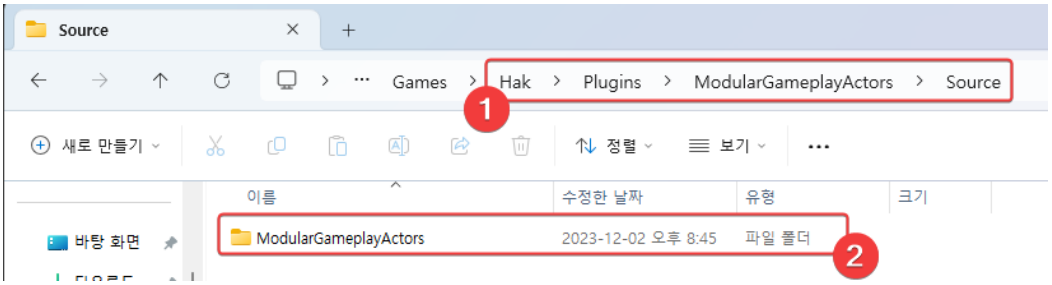
☐ ModularGameplayActors 폴더 생성:



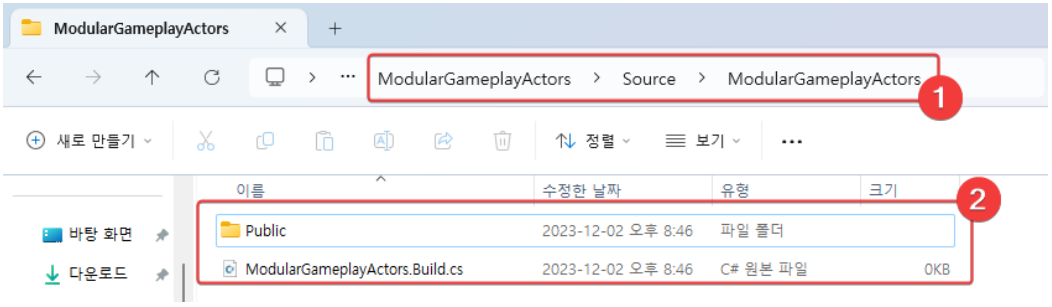
☐ Source 폴더 생성과 ModularGameplayActors.uplugin 생성:



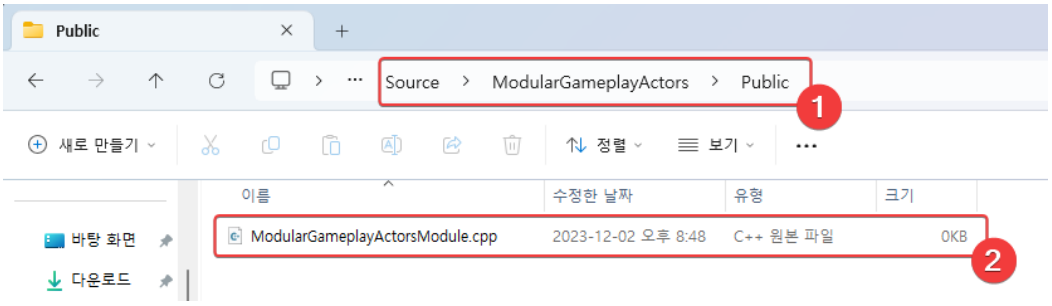
☐ ModularGameplayActors 생성 → Modular 경로 생성



☐ ModularGameplayActors의 기본 Module에 필요한 파일 및 폴더 추가:

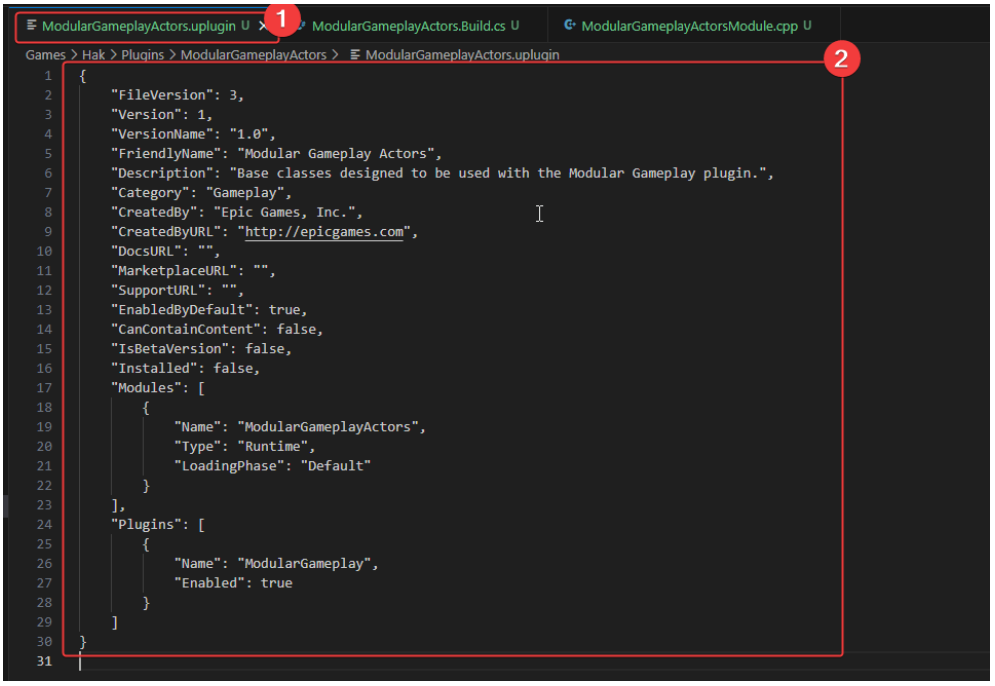


☐ 기본 모듈 파일 생성: `ModularGameplayActorsModule.cpp`

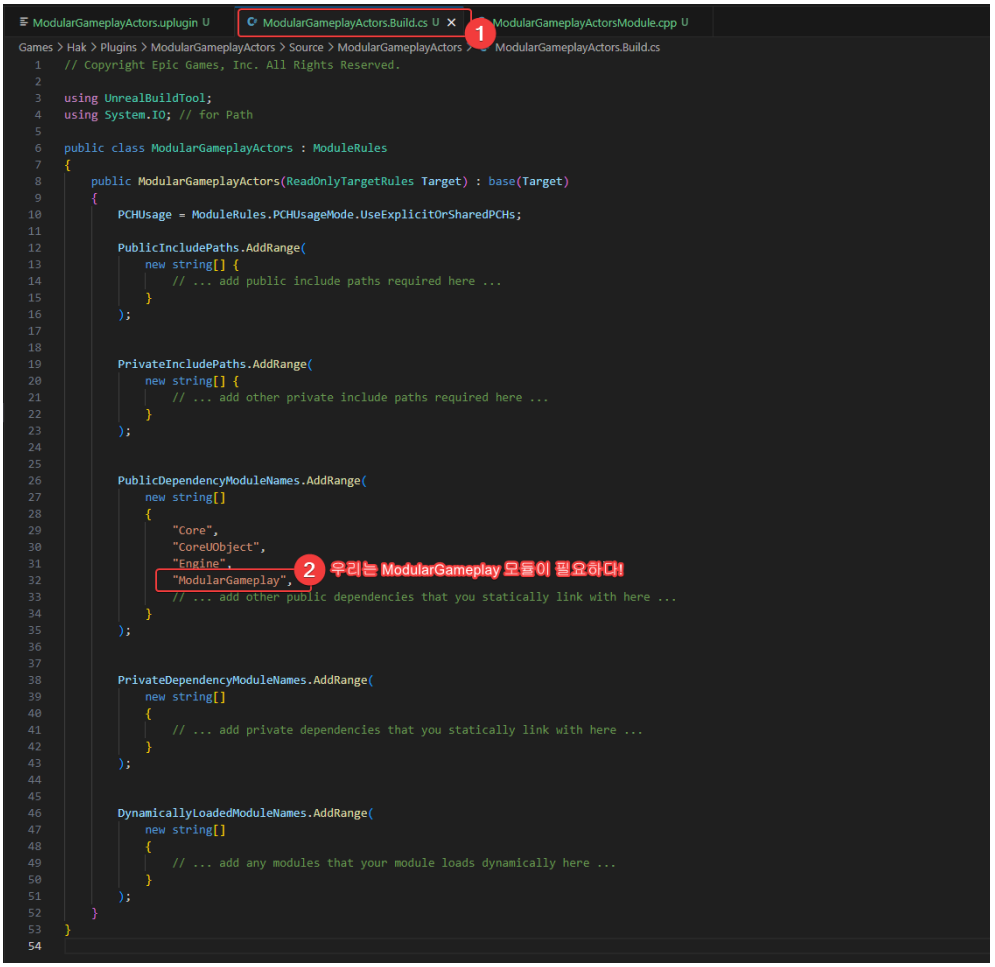


☐ 아래와 같이 각각 파일에 대한 내용 추가:

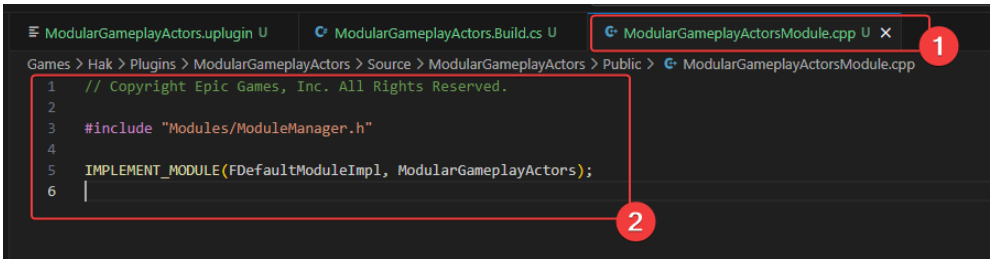
- ModuleGameplayActors.uplugin:



- ModularGameplayActors.Build.cs:

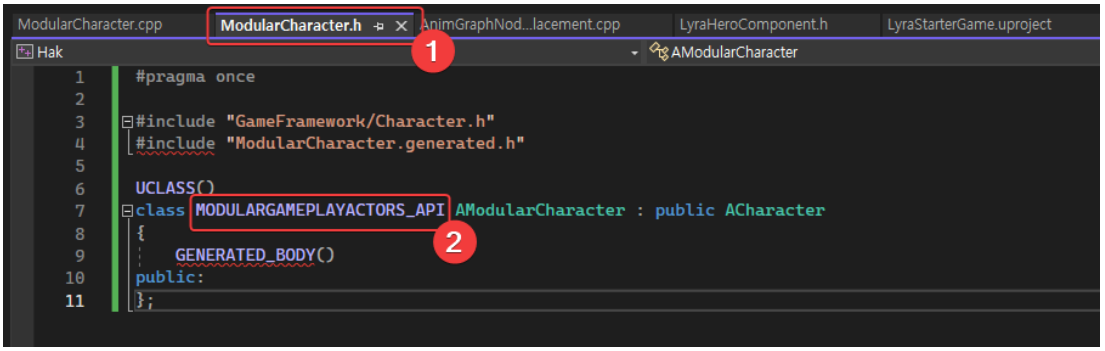


- ModuleGameplayActorsModule.cpp:

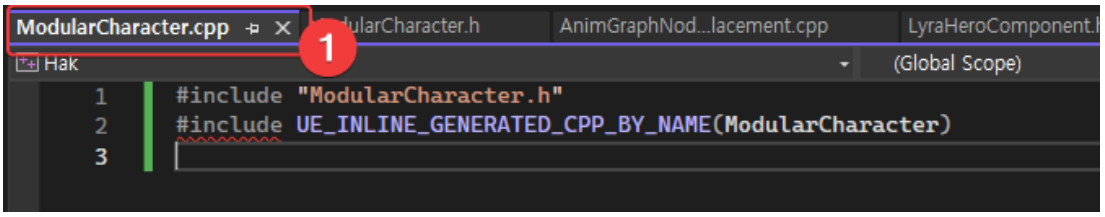


- [ ] GenerateProjectFiles.bat를 실행시켜, 해당 Plugin을 생성시키자.

- [ ] 이제 ModuleGameplayActors에서 Actor들 중에, ACharacter와 APlayerCharacter를 Modular Actor 타입으로 생성해주자:
  - [ ] 우선, ModularCharacter.h/.cpp와 ModularPlayerController.h/.cpp를 생성해주자:
- [ ] ModularCharacter.h/.cpp 구현:



- 꼭 잊지말자! 해당 플러그인과 모듈은 외부이므로 HakGame에서 써야하기 때문에 _API 매크로를 사용해야 한다



- [ ] GameFrameworkComponentManager를 활용하여, Character를 모듈화해주자:
  - ModularCharacter.h:

```cpp
#pragma once

#include "Engine/EngineTypes.h"
#include "GameFramework/Character.h"
#include "ModularCharacter.generated.h"

UCLASS()
class MODULARGAMEPLAYACTORS_API AModularCharacter : public ACharacter
{
    GENERATED_BODY()
public:
    /**
    * character's interface
    */
    virtual void PreInitializeComponents() override;
    virtual void BeginPlay() override;
    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) override;
};
```

- ModularCharacter.cpp:

```cpp
#include "ModularCharacter.h"
#include "Components/GameFrameworkComponentManager.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(ModularCharacter)

void AModularCharacter::PreInitializeComponents()
{
    Super::PreInitializeComponents();
    UGameFrameworkComponentManager::AddGameFrameworkComponentReceiver(this);
}

void AModularCharacter::BeginPlay()
{
    UGameFrameworkComponentManager::SendGameFrameworkComponentExtensionEvent(this, UGameFrameworkComponentManager::NAME_GameActorReady);
    Super::BeginPlay();
}

void AModularCharacter::EndPlay(const EEndPlayReason::Type EndPlayReason)
{
    UGameFrameworkComponentManager::RemoveGameFrameworkComponentReceiver(this);
    Super::EndPlay(EndPlayReason);
}
```

☐ ModularPlayerController.h/.cpp 구현:

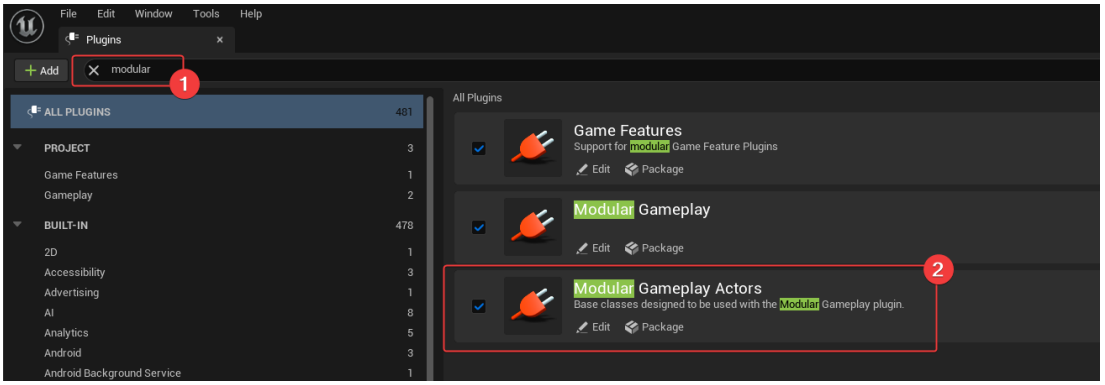
앞서, ModularCharacter랑 똑같이 바로 Interface까지 같이 구현해주자:

- ModularGameplayController.h:

```cpp
#pragma once

#include "Engine/EngineTypes.h"
#include "GameFramework/PlayerController.h"
#include "ModularPlayerController.generated.h"

UCLASS()
class MODULARGAMEPLAYACTORS_API AModularPlayerController : public APlayerController
{
    GENERATED_BODY()
public:
    virtual void PreInitializeComponents() override;
    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) override;
};
```

- ModularGameplayController.cpp:

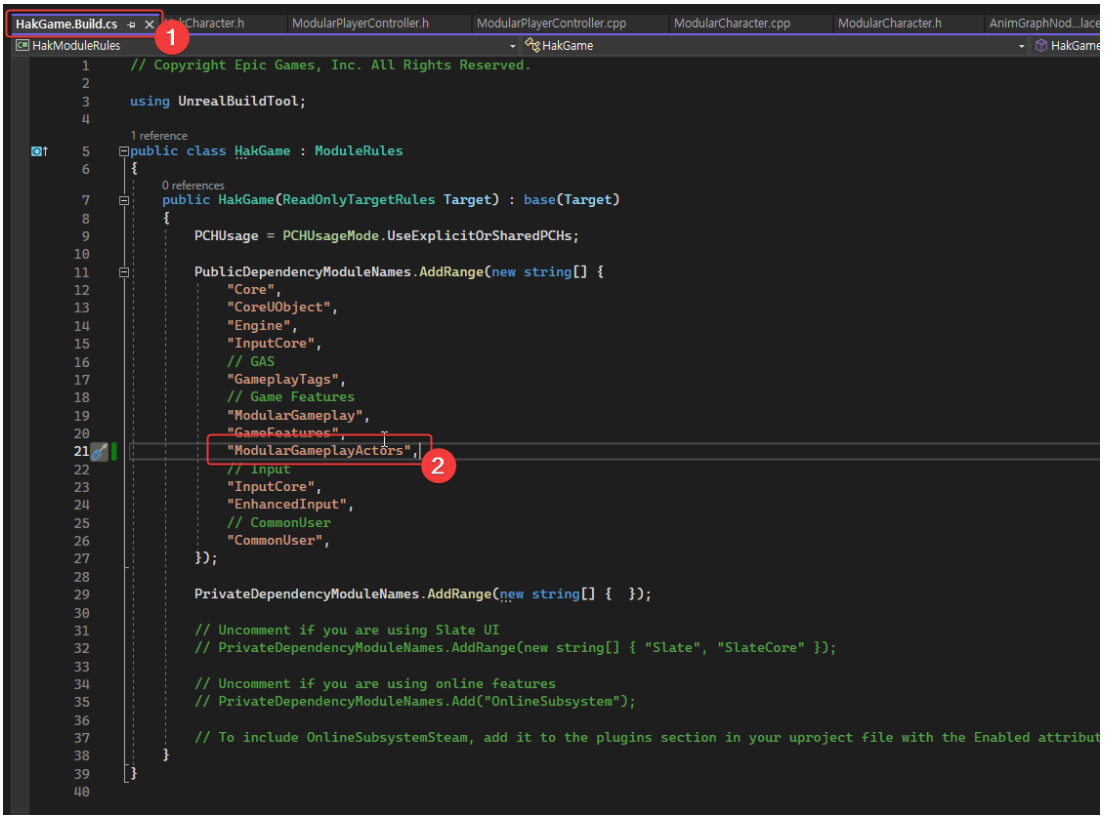
☐ ModularGameplayActors을 활성화하자:



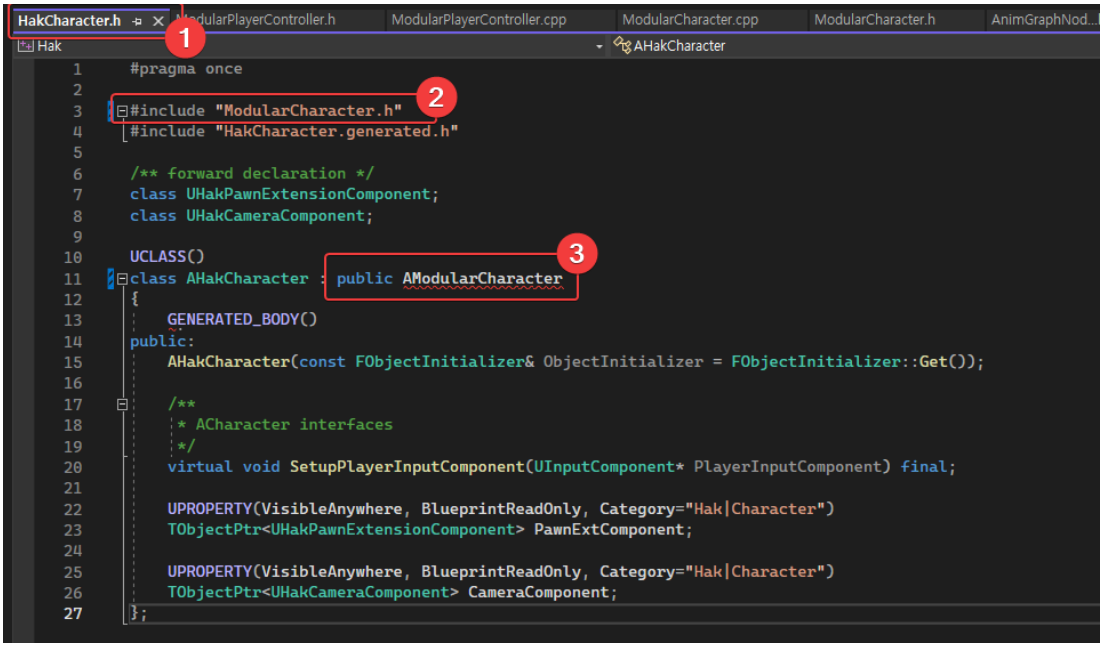
ModularGameplayActor에 대해 상속:

▼ **펼치기**

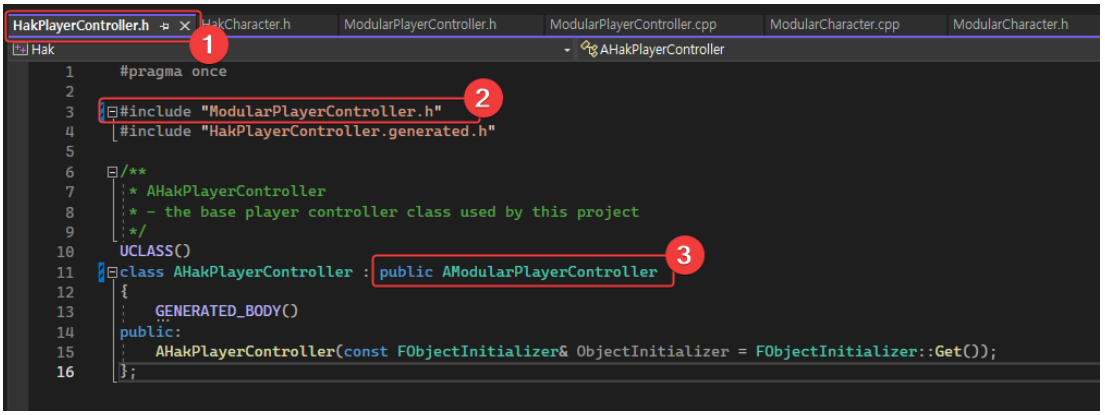
☐ HakGame.Build.cs에서 ModularGameplayActors를 추가하자:



☐ HakCharacter를 ModularCharacter 상속으로 교체하자:



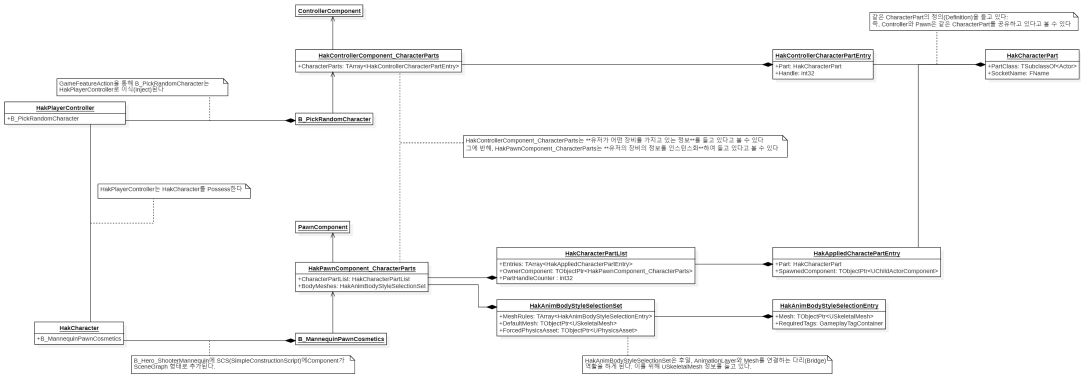
☐ HakPlayerController를 ModularPlayerController 상속으로 교체하자:



Cosmetics Classes

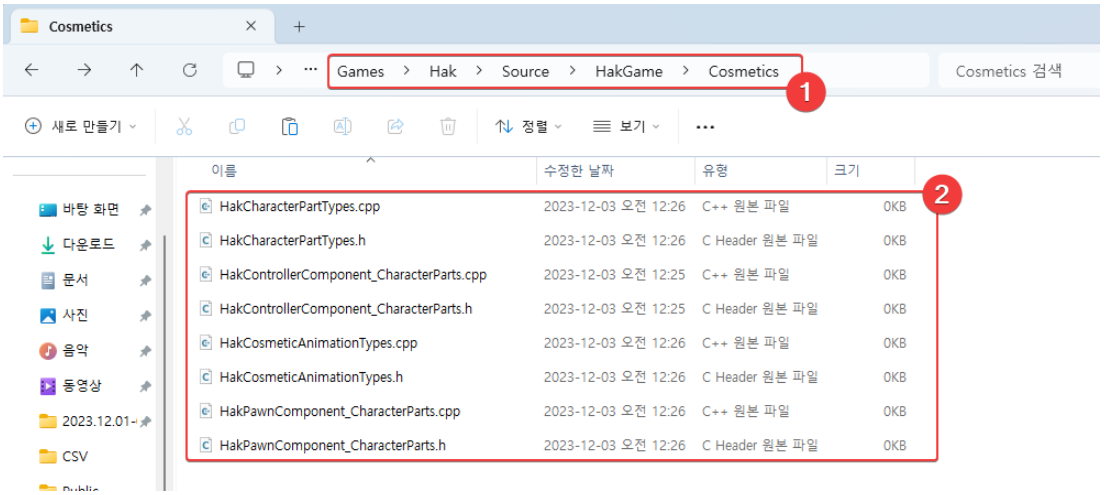
▼ **펼치기**

☐ 전체적인 클래스 구조도 보기:

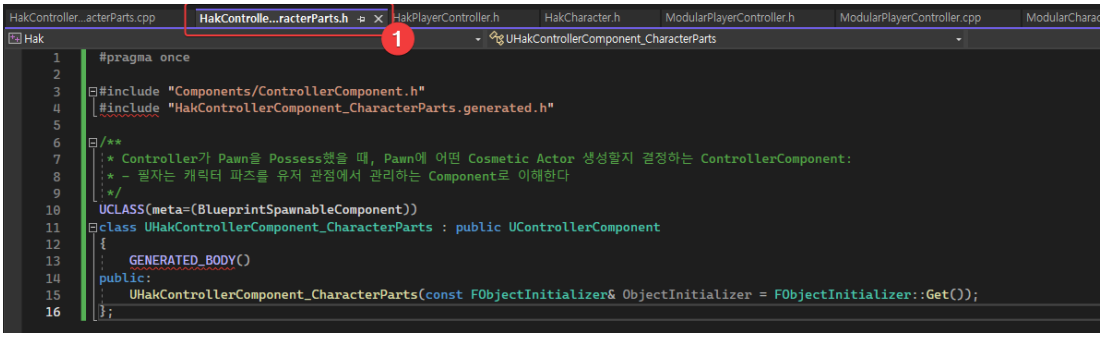


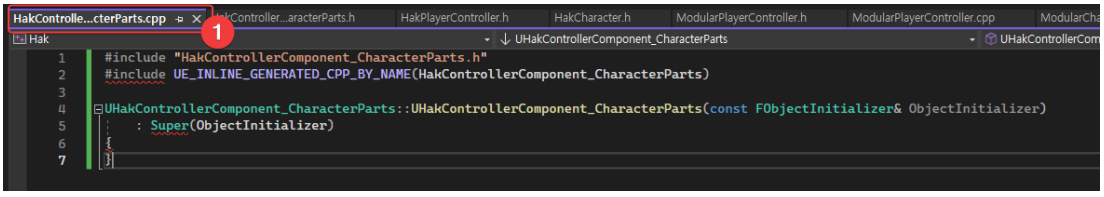
☐ 아래의 파일들 생성:

- ☐ HakControllerComponent_CharacterParts.h/.cpp
- ☐ HakPawnComponent_CharacterParts.h/.cpp
- ☐ HakCharacterPartTypes.h/.cpp
- ☐ HakCosmeticAnimationTypes.h/.cpp

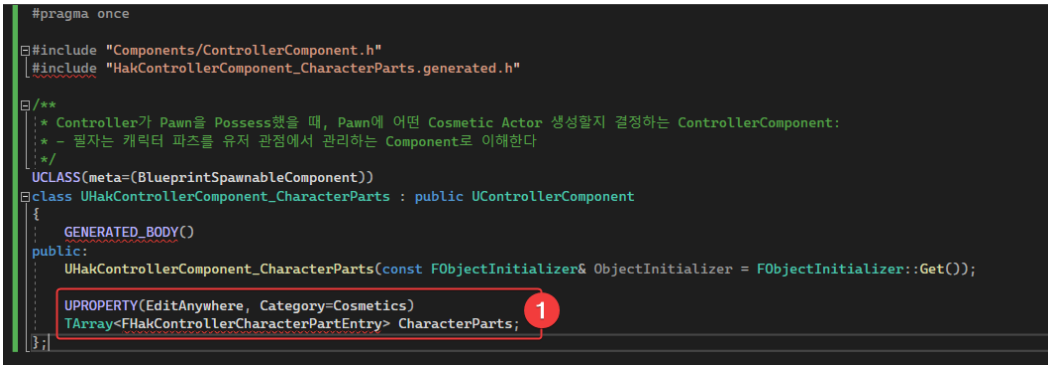


☐ HakControllerComponent_CharacterParts 구현:





☐ HakControllerComponent_CharacterParts의 멤버 변수를 구현하자:



☐ HakControllerCharacterPartEntry

```cpp
/** ControllerComponent가 소유하는 Character Parts */
USTRUCT()
struct FHakControllerCharacterPartEntry
{
    GENERATED_BODY()

    /** Character Part에 대한 정의(메타 데이터 == MetaData) */
    UPROPERTY(EditAnywhere)
    FHakCharacterPart Part;

    /** Character Part 핸들 (고유값) - Controller가 Possess하고 있는 Pawn에서 생성한(인스턴스) Character Part 핸들값 */
    FHakCharacterPartHandle Handle;
};
```

☐ HakCharacterPartTypes.h/.cpp 구현:

HakCharacterPartTypes.cpp | LyraCharacterPartTypes.h | HakCharacterPartTypes.h | HakController...acterParts.cpp | HakController...aracterParts.h

Hak | FHakCharacterPart

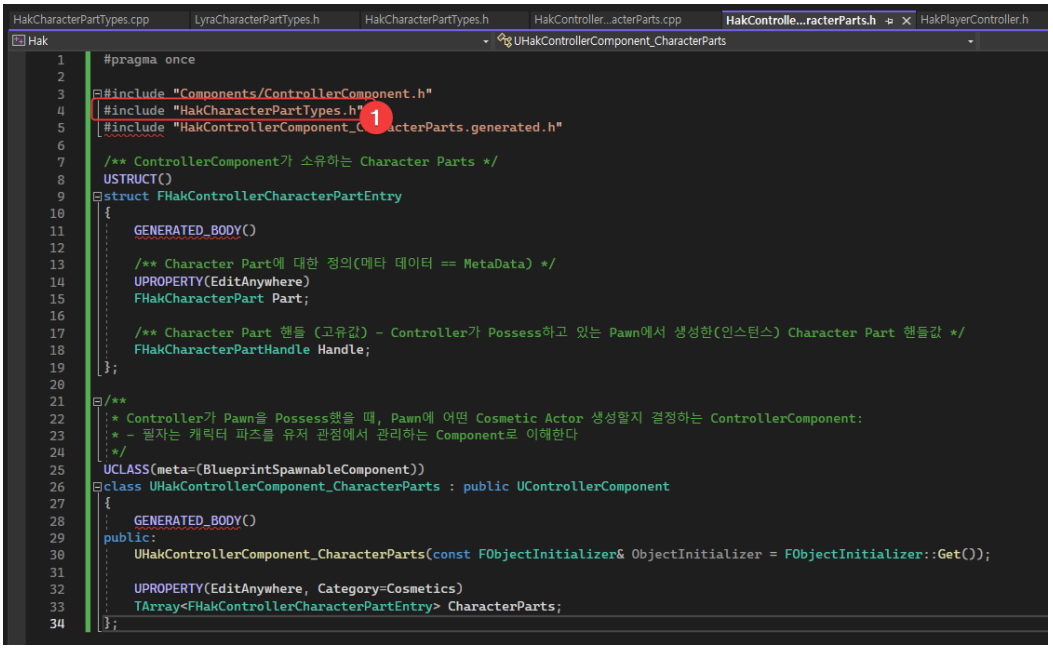
```cpp
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "HakCharacterPartTypes.generated.h"

USTRUCT(BlueprintType)
struct FHakCharacterPartHandle
{
    GENERATED_BODY()

    UPROPERTY()
    int32 PartHandle = INDEX_NONE;
};

USTRUCT(BlueprintType)
struct FHakCharacterPart
{
    GENERATED_BODY()

    /** 파츠로 생성된 Actor의 Class 정보를 가지고 있음 -> Actor BP를 하나의 파츠로 생각하면 됨 */
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TSubclassOf<AActor> PartClass;

    /** Actor를 소환하여 붙이는데 특정 Bone에 붙이기 위한 정보 (따로 정의되어 있지 않으면 그냥 Root에 붙임) */
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FName SocketName;
};
```

HakCharacterPartTypes.cpp | LyraCharacterPartTypes.h | HakCharacterPartTypes.h | HakController...acterI

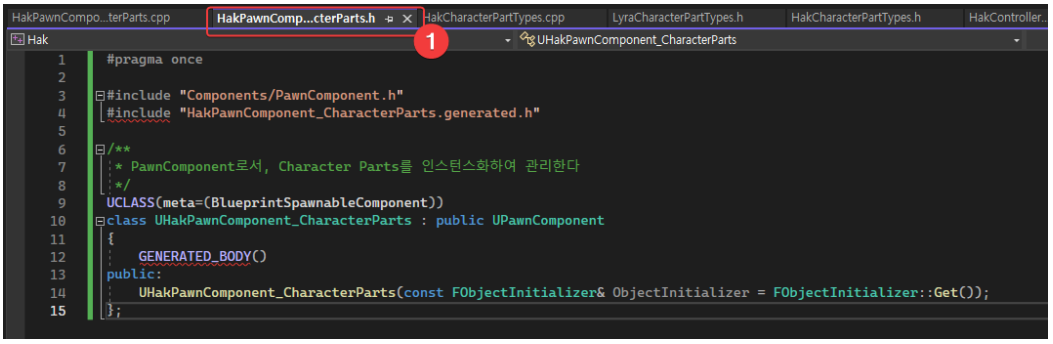
Hak | (Global Scope)

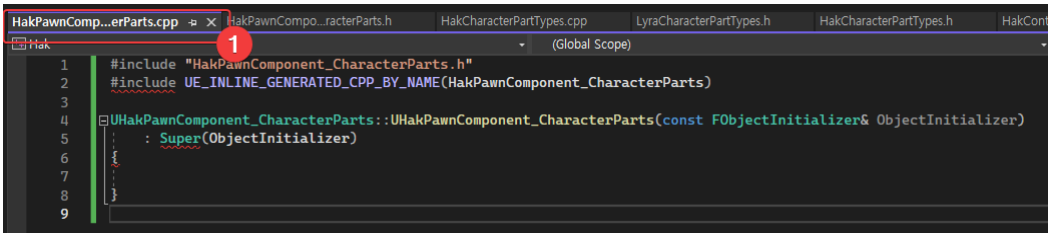
```cpp
#include "HakCharacterPartTypes.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakCharacterPartTypes)
```

☐ HakControllerComponent_CharacterParts.h에 HakCharacterPartTypes.h 추가:



- [ ] HakPawnComponent_CharacterParts 구현:
  - [ ] HakPawnComponent_CharacterParts.h/.cpp

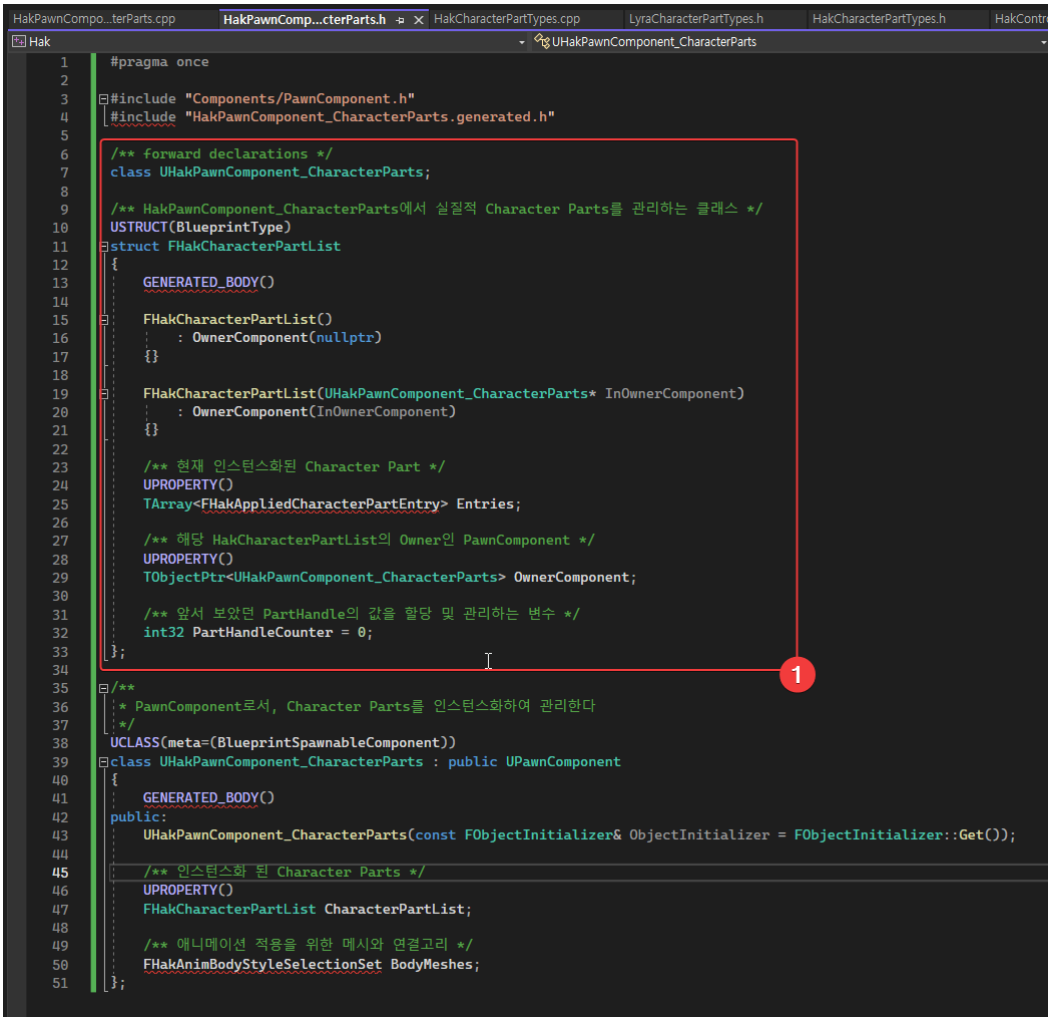




  - [ ] HakPawnComponent_CharacterParts의 멤버 변수들 구현:

```cpp
#pragma once

#include "Components/PawnComponent.h"
#include "HakPawnComponent_CharacterParts.generated.h"

/**
 * PawnComponent로서, Character Parts를 인스턴스화하여 관리한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakPawnComponent_CharacterParts : public UPawnComponent
{
    GENERATED_BODY()
public:
    UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** 인스턴스화 된 Character Parts */
    UPROPERTY()
    FHakCharacterPartList CharacterPartList;

    /** 애니메이션 적용을 위한 메시와 연결고리 */
    FHakAnimBodyStyleSelectionSet BodyMeshes;
};
```

- [ ] HakCharacterPartList



- [ ] HakAppliedCharacterPartEntry

```cpp
#pragma once

#include "HakCharacterPartTypes.h"
#include "Components/PawnComponent.h"
#include "HakPawnComponent_CharacterParts.generated.h"

/** forward declarations */
class UHakPawnComponent_CharacterParts;

/** 인스턴스화 된 Character Part의 단위 */
USTRUCT()
struct FHakAppliedCharacterPartEntry
{
    GENERATED_BODY()

    /** Character Part의 정의(메타 데이터) */
    UPROPERTY()
    FHakCharacterPart Part;

    /** HakCharacterPartList에서 할당 받은 Part 핸들값 (FHakControllerCharacterPartEntry의 Handle 값과 같아야 함 -> 같으면 같은 Part) */
    UPROPERTY()
    int32 PartHandle = INDEX_NONE;

    /** 인스턴스화 된 Character Part용 Actor */
    UPROPERTY()
    TObjectPtr<UChildActorComponent> SpawnedComponent = nullptr;
};

/** HakPawnComponent_CharacterParts에서 실질적 Character Parts를 관리하는 클래스 */
USTRUCT(BlueprintType)
struct FHakCharacterPartList
{
    GENERATED_BODY()

    FHakCharacterPartList()
        : OwnerComponent(nullptr)
    {}

    FHakCharacterPartList(UHakPawnComponent_CharacterParts* InOwnerComponent)
        : OwnerComponent(InOwnerComponent)
    {}

    /** 현재 인스턴스화된 Character Part */
    UPROPERTY()
    TArray<FHakAppliedCharacterPartEntry> Entries;

    /** 해당 HakCharacterPartList의 Owner인 PawnComponent */
    UPROPERTY()
    TObjectPtr<UHakPawnComponent_CharacterParts> OwnerComponent;

    /** 앞서 보았던 PartHandle의 값을 할당 및 관리하는 변수 */
    int32 PartHandleCounter = 0;
};

/**
 * PawnComponent로서, Character Parts를 인스턴스화하여 관리한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakPawnComponent_CharacterParts : public UPawnComponent
{
    GENERATED_BODY()
public:
    UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** 인스턴스화 된 Character Parts */
    UPROPERTY()
    FHakCharacterPartList CharacterPartList;

    /** 애니메이션 적용을 위한 메시와 연결고리 */
    FHakAnimBodyStyleSelectionSet BodyMeshes;
};
```

☐ HakCosmeticAnimationTypes.h/.cpp

```cpp
#pragma once

#include "HakCosmeticAnimationTypes.generated.h"

/** forward declarations */
class USkeletalMesh;
class UPhysicsAsset;

USTRUCT(BlueprintType)
struct FHakAnimBodyStyleSelectionSet
{
    GENERATED_BODY()

    /** AnimLayer 적용할 SkeletalMesh를 들고 있음 -> Animation-Mesh간 Rules을 MeshRules라고 생각하면 됨 */
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TArray<FHakAnimBodyStyleSelectionEntry> MeshRules;

    /** 그냥 디폴트로 적용할 SkeletalMesh */
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TObjectPtr<USkeletalMesh> DefaultMesh = nullptr;

    /** Physics Asset은 하나로 통일함 -> 즉 모든 Animation의 Physics 속성은 공유함 */
    UPROPERTY(EditAnywhere)
    TObjectPtr<UPhysicsAsset> ForcedPhysicsAsset = nullptr;
};
```

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

☐ HakAnimBodyStyleSelectionEntry:

```cpp
#pragma once

#include "GameplayTagContainer.h"
#include "HakCosmeticAnimationTypes.generated.h"

/** forward declarations */
class USkeletalMesh;
class UPhysicsAsset;

USTRUCT(BlueprintType)
struct FHakAnimBodyStyleSelectionEntry
{
    GENERATED_BODY()

    /** AnimLayer를 적용할 대상 SkeletalMesh */
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TObjectPtr<USkeletalMesh> Mesh = nullptr;

    /** Cosmetic Tag라고 생각하면 됨 */
    UPROPERTY(EditAnywhere, BlueprintReadWrite, meta=(Categories="Cosmetic"))
    FGameplayTagContainer RequiredTags;
};

USTRUCT(BlueprintType)
struct FHakAnimBodyStyleSelectionSet
{
    GENERATED_BODY()

    /** AnimLayer 적용할 SkeletalMesh를 들고 있음 -> Animation-Mesh간 Rules을 MeshRules라고 생각하면 됨 */
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TArray<FHakAnimBodyStyleSelectionEntry> MeshRules;

    /** 그냥 디폴트로 적용할 SkeletalMesh */
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TObjectPtr<USkeletalMesh> DefaultMesh = nullptr;

    /** Physics Asset은 하나로 동일함 -> 즉 모든 Animation의 Physics 속성은 공유함 */
    UPROPERTY(EditAnywhere)
    TObjectPtr<UPhysicsAsset> ForcedPhysicsAsset = nullptr;
};
```

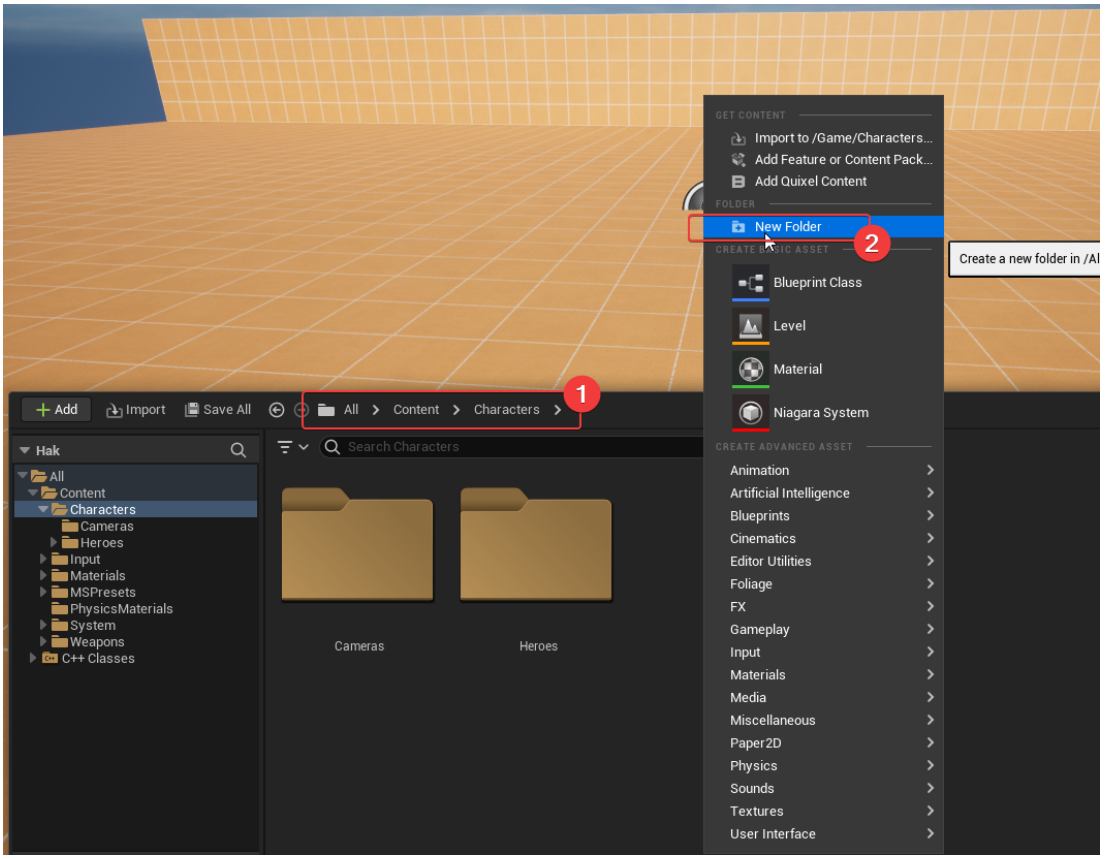
☐ HakPawnComponent_CharacterParts.h에 HakCosmeticAnimationTypes.h 추가:

```cpp
#pragma once

#include "HakCharacterPartTypes.h"
#include "HakCosmeticAnimationTypes.h"
#include "Components/PawnComponent.h"
#include "HakPawnComponent_CharacterParts.generated.h"

/** forward declarations */
class UHakPawnComponent_CharacterParts;

/** 인스턴스화 된 Character Part의 단위 */
USTRUCT()
struct FHakAppliedCharacterPartEntry
{
    GENERATED_BODY()

    /** Character Part의 정의(메타 데이터) */
    UPROPERTY()
    FHakCharacterPart Part;

    /** HakCharacterPartList에서 할당 받은 Part 핸들값 (FHakControllerCharacterPartEntry의 Handle 값과 같아야 함 -> 같으면 같은 Part) */
    UPROPERTY()
    int32 PartHandle = INDEX_NONE;

    /** 인스턴스화 된 Character Part용 Actor */
    UPROPERTY()
    TObjectPtr<UChildActorComponent> SpawnedComponent = nullptr;
};
```

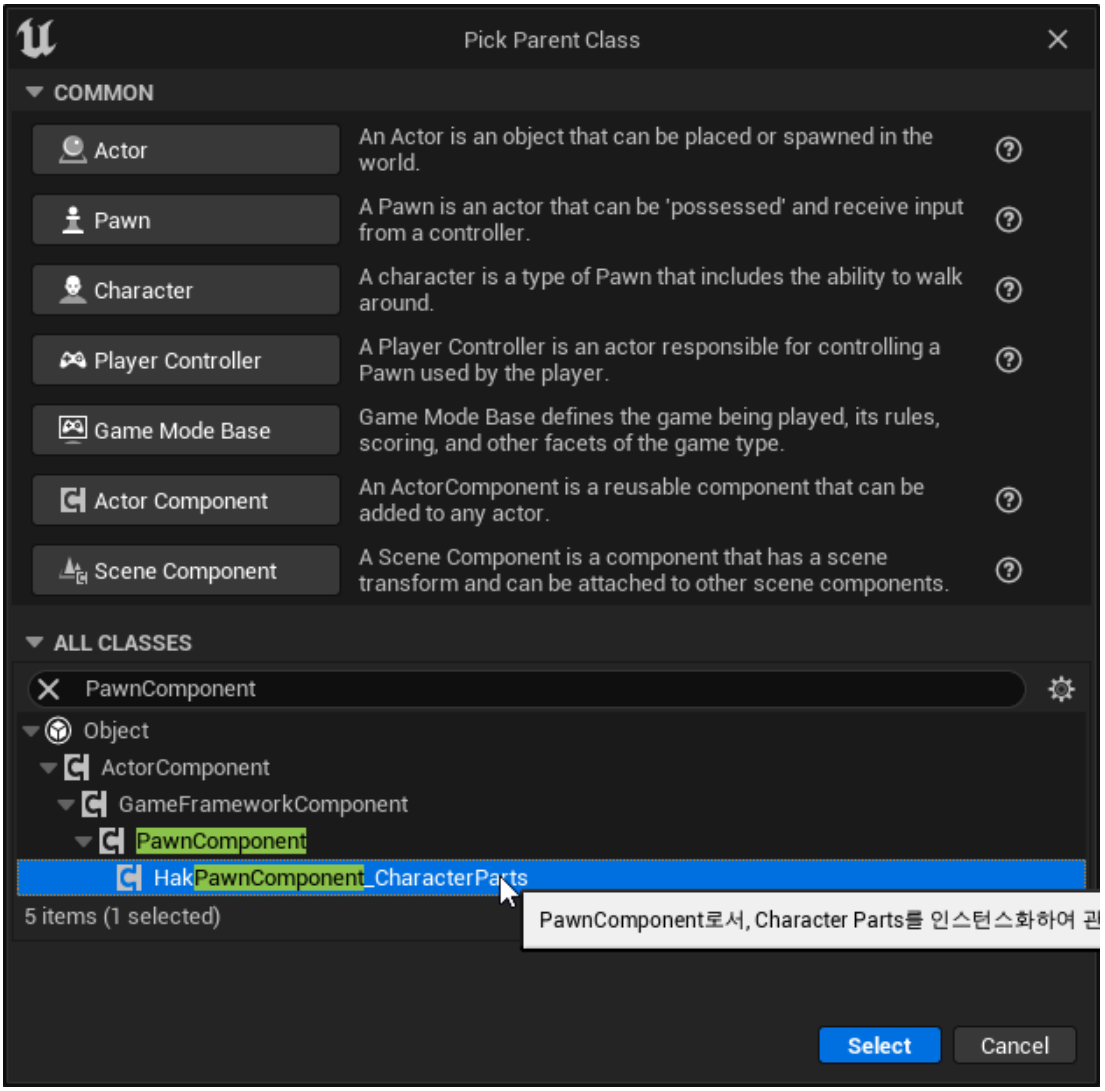
Cosmetics Assets

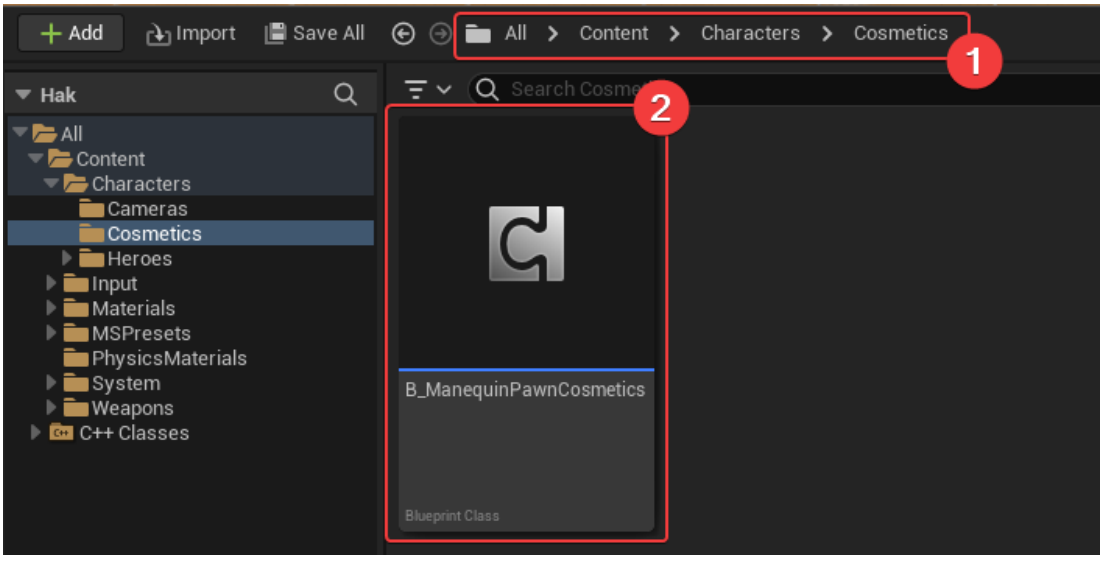
▼ 펼치기

☐ Root Content 폴더에 `Characters/Cosmetics`를 생성해주자:



☐ HakPawnComponent_CharacterParts를 상속 받는 B_ManequinPawnCosmetics를 생성하자:



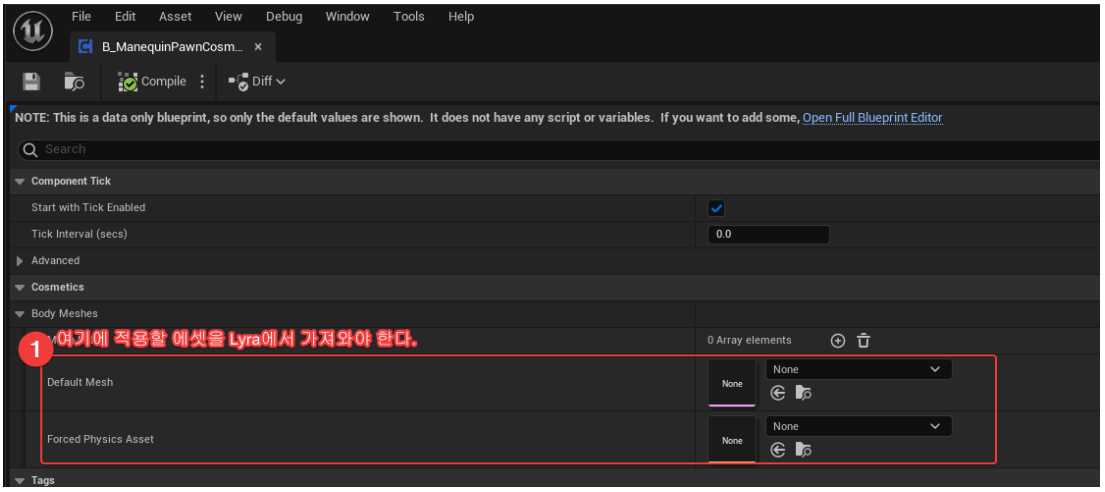


☐ 앞서 코드를 따라온 분들은 아래와 같이 PROPERTY()를 추가해줘야 B_ManequinPawnCosmetics에서 BodyMeshes가 보인다:

```cpp
/**
 * PawnComponent로서, Character Parts를 인스턴스화하여 관리한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakPawnComponent_CharacterParts : public UPawnComponent
{
    GENERATED_BODY()
public:
    UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** 인스턴스화 된 Character Parts */
    UPROPERTY()
    FHakCharacterPartList CharacterPartList;

    /** 애니메이션 적용을 위한 메시와 연결고리 */
    UPROPERTY(EditAnywhere, Category=Cosmetics)
    FHakAnimBodyStyleSelectionSet BodyMeshes;
};
```

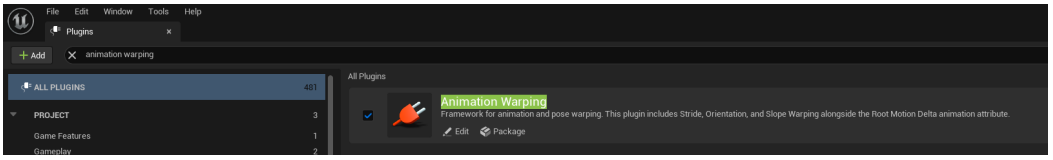
1

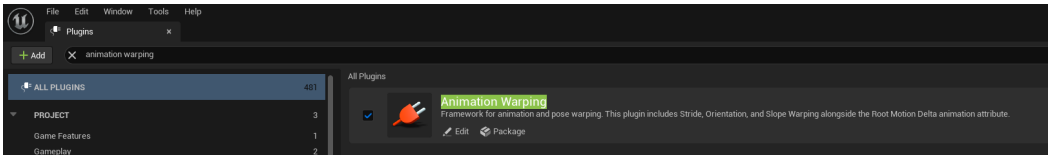
☐ 아래와 같이 적용할 에셋을 Lyra에서 가져와야 한다:



☐ 우선 Lyra에서 에셋을 Import하기 전에 Lyra에서 활용하는 Plugin들을 미리 활성화 해놓자:

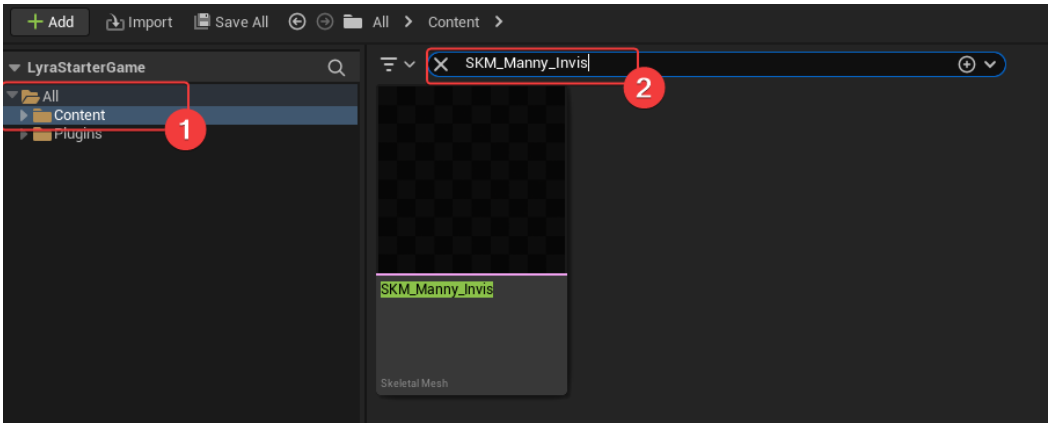
- ☐ `AnimationLocomotionLibrary`와 `AnimationWarping`:

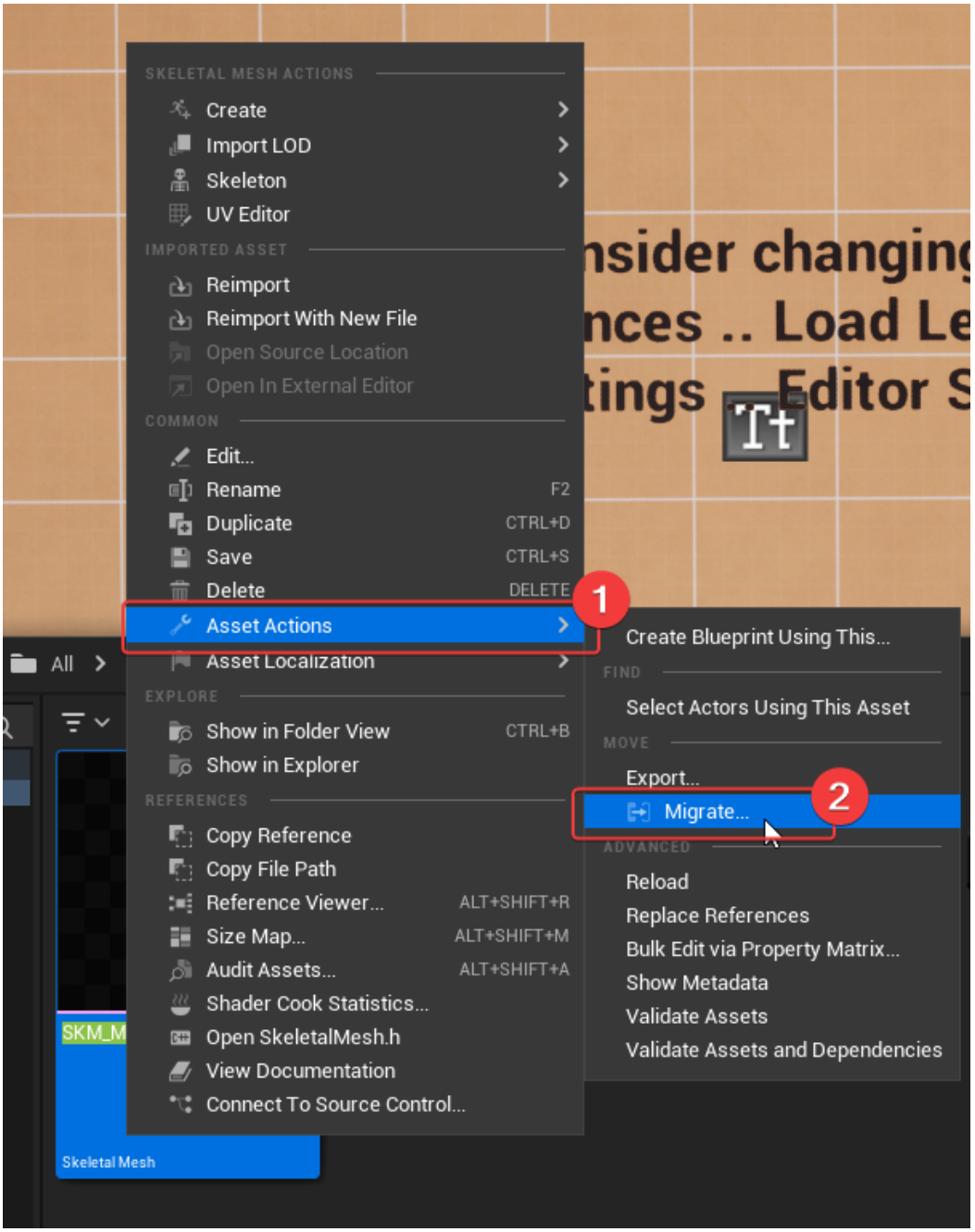




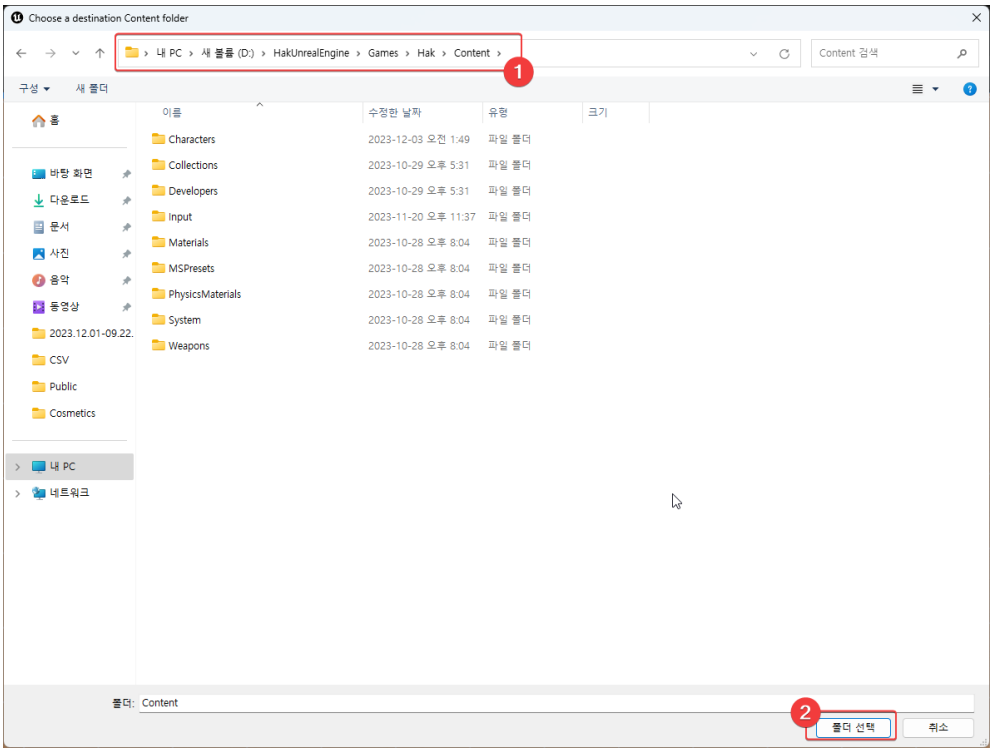
☐ LyraStarterGame을 실행해주자:

- ☐ 우선 SKM_Manny_Invis를 Mirgrate 시켜주자:


Add
Import
Save All
All
Content
LyraStarterGame
All
Content
Plugins
1
SKM_Manny_Invis
2
SKM_Manny_Invis
Skeletal Mesh

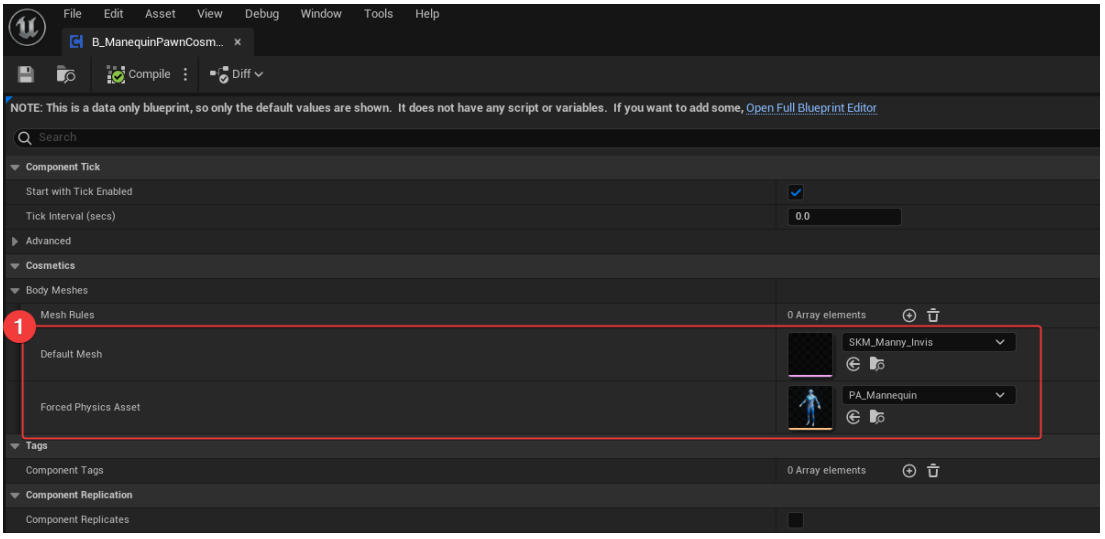

SKELETAL MESH ACTIONS
Create
Import LOD
Skeleton
UV Editor
IMPORTED ASSET
Reimport
Reimport With New File
Open Source Location
Open In External Editor
COMMON
Edit...
Rename F2
Duplicate CTRL+D
Save CTRL+S
Delete DELETE
Asset Actions
Asset Localization
EXPLORE
Show in Folder View CTRL+B
Show in Explorer
REFERENCES
Copy Reference
Copy File Path
Reference Viewer... ALT+SHIFT+R
Size Map... ALT+SHIFT+M
Audit Assets... ALT+SHIFT+A
Shader Cook Statistics...
Open SkeletalMesh.h
View Documentation
Connect To Source Control...
1
Create Blueprint Using This...
FIND
Select Actors Using This Asset
MOVE
Export...
Migrate...
2
ADVANCED
Reload
Replace References
Bulk Edit via Property Matrix...
Show Metadata
Validate Assets
Validate Assets and Dependencies
All
SKM_M
Skeletal Mesh

- 우리의 Root Content에 경로를 설정하면 된다:

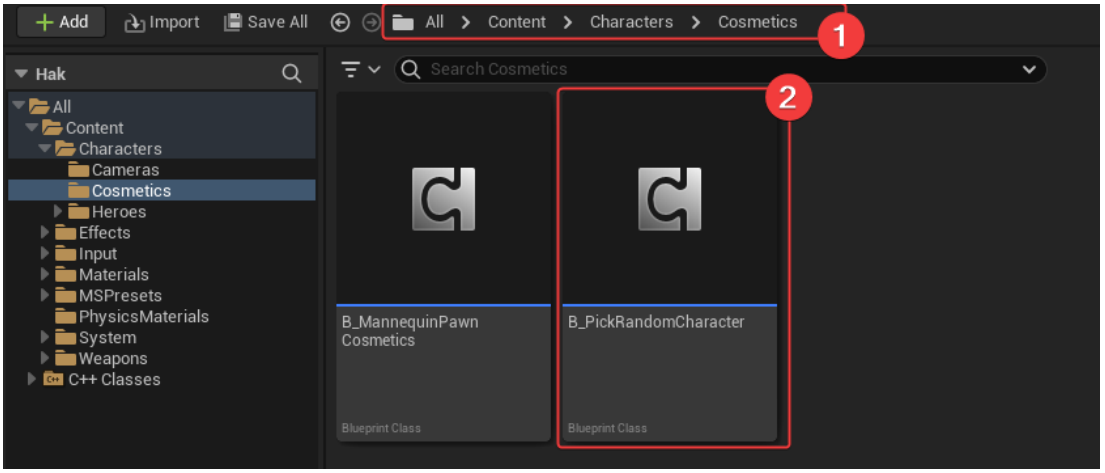


☐ 다음은 PA_Mannequin을 Migrate를 위와 같이 해주자

☐ 아래와 같이 B_MannequinPawnCosmetics를 설정해주자:

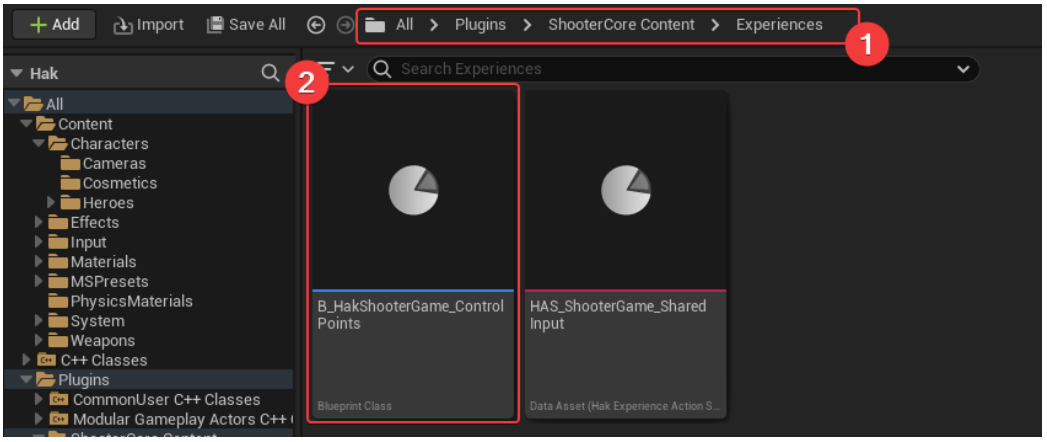


☐ HakControllerComponent_CharacterParts를 상속받은 B_PickRandomCharacter를 생성하자:



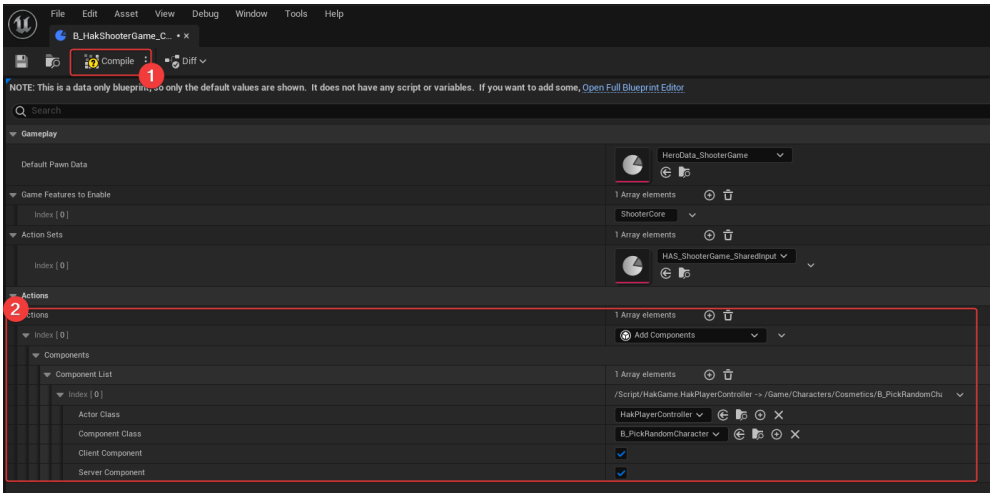
☐ GameFeatureAction을 활용하여, B_PickRandomCharacter를 ShooterCore Plugin이 활성화 될 때, 알맞은 Actor에 붙여주자:

- B_HakShooterGame_ControlPoints에 추가하자:

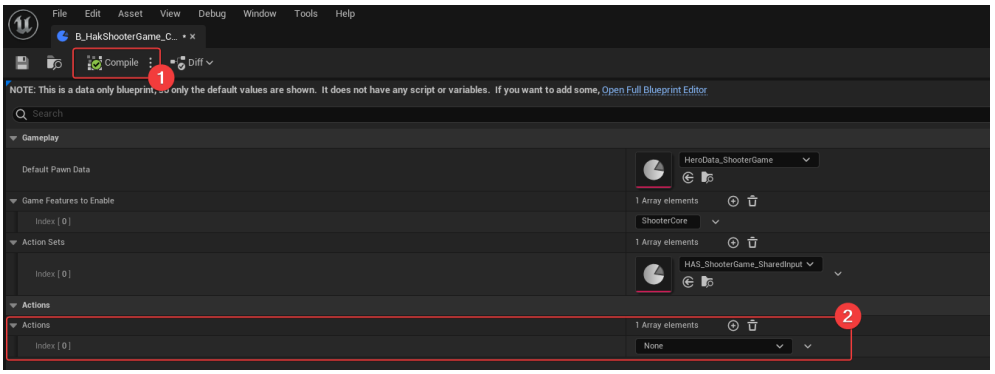


☐ 아래와 같이 세팅한 후, 컴파일하면 없어진다?!

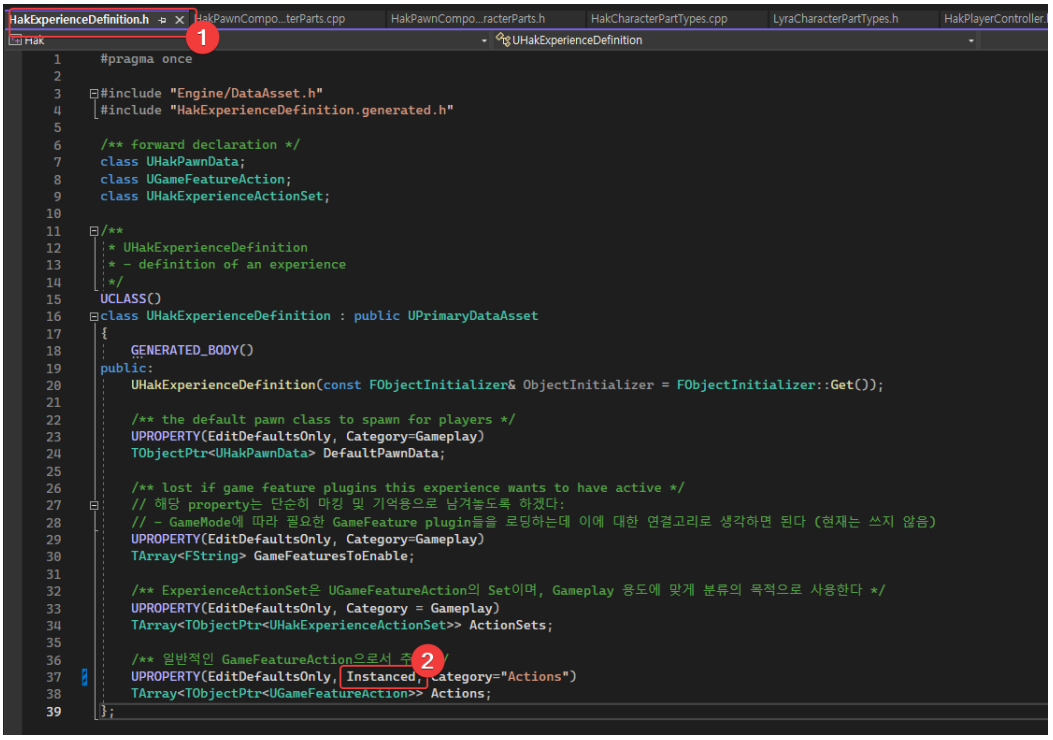
- **<u>컴파일 전</u>**:



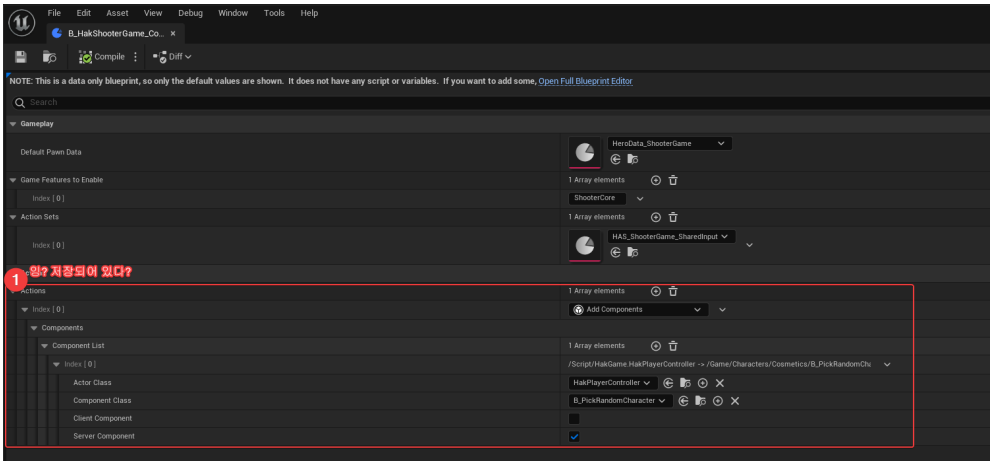
- **컴파일 후**:



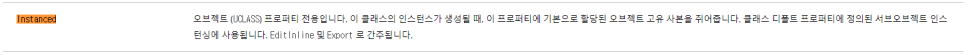
☐ 해당 원인은 UPROPERTY() 속성은 Instanced를 넣지 않아서 그렇다:



- 변경 후, 다시 열어보면?



- 무슨 일이 벌어지고 있는걸까?

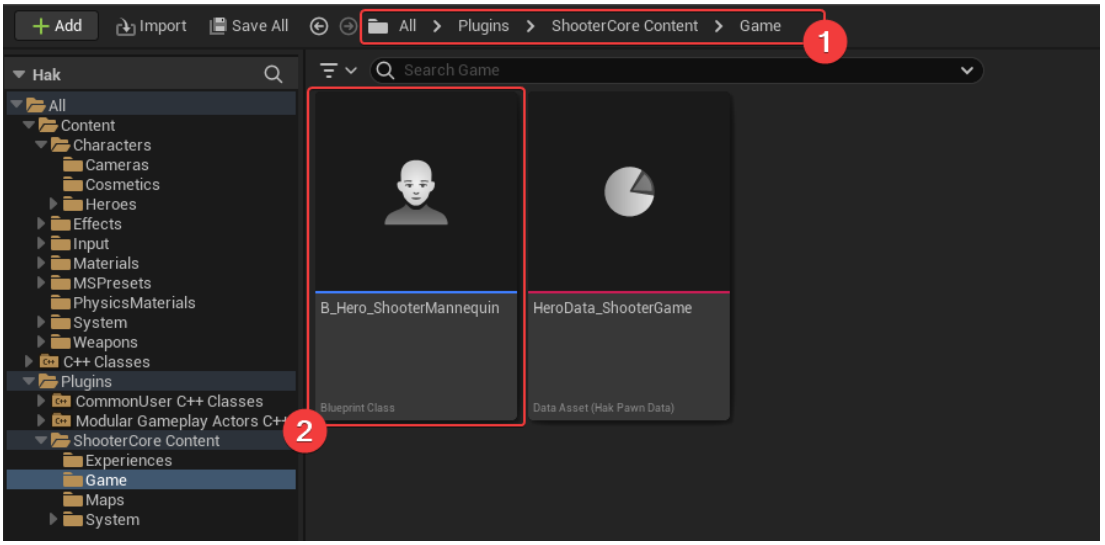


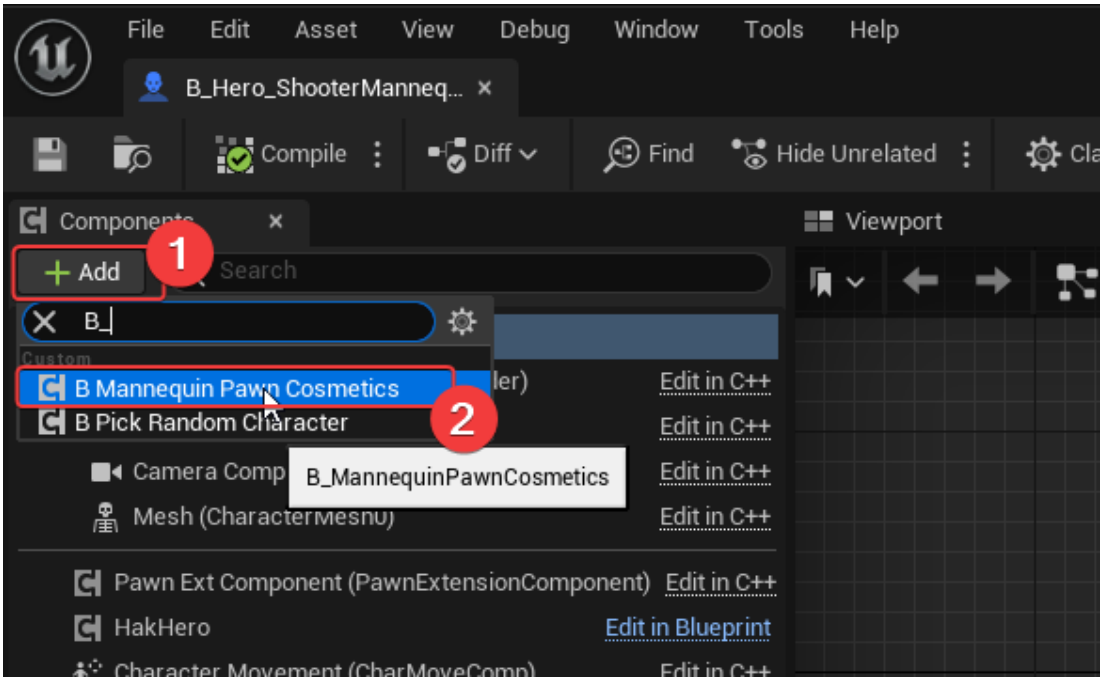
우리가 변경하고자 하는 대상은 BP이다:

- 사실 BP는 클래스 생성하는 객체로 **초기값을 멤버 변수가 가질 수 없다** (항상 리셋된다...)
- 이를 **가능하게 하는 기능은** `Instanced` **라고 생각하면 된다**
- 위의 Unreal 문서에 적혀있듯이, **우리가 설정한 값으로 초기화 값으로 세팅한다는 의미라고 이해할 수 있다**.

> 💡 그러나.. 사실 우리가 앞서 현상에서 보았듯이, 뭔가 엔진 버그에 가까운거 같다. 이미 내부적으로 해당 값을 Serialize해서 가지고 있다... `Instanced` 설정 이후, 값이 보여질 뿐이다... 😅
> (일단 여기서는 그냥 넘어가자...)

☐ B_Hero_ShooterMannequin에 생성한 B_MannequinPawnCosmetics를 추가하자:





- 이름은 `PawnCosmeticsComponent`

AddCharacterPart

▼ **펼치기**

☐ B_PickRandomCharacter의 BeginPlay 노드를 작성하기 앞서, AddCharacterPart를 추가하자:

- HakControllerComponent_CharacterParts:

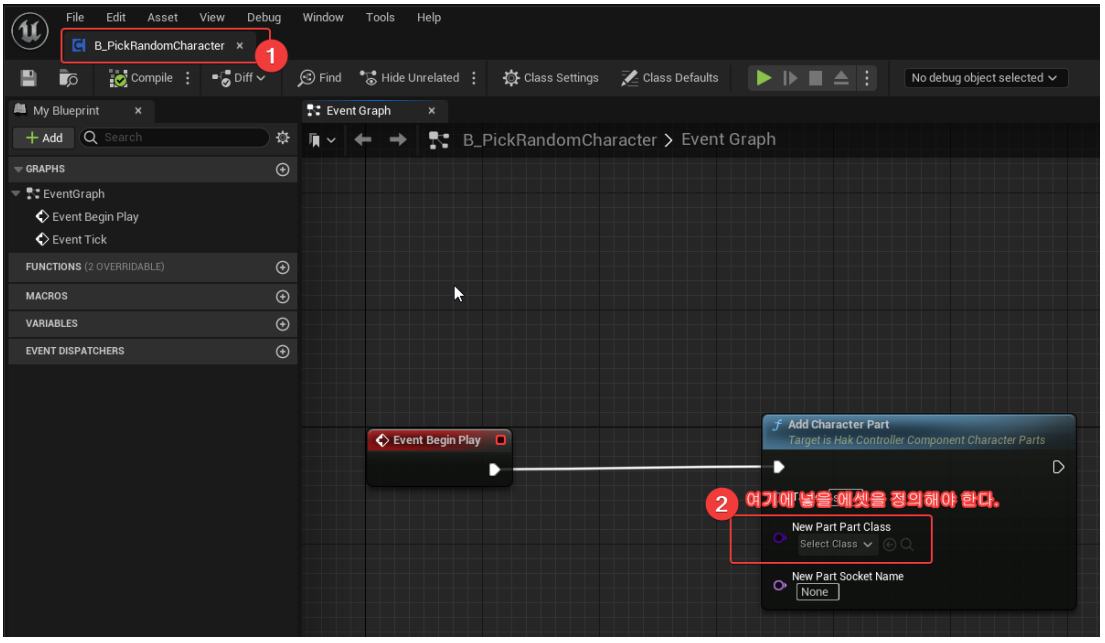
```cpp
/**
 * Controller가 Pawn을 Possess했을 때, Pawn에 어떤 Cosmetic Actor 생성할지 결정하는 ControllerComponent:
 * - 필자는 캐릭터 파츠를 유저 관점에서 관리하는 Component로 이해한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakControllerComponent_CharacterParts : public UControllerComponent
{
    GENERATED_BODY()
public:
    UHakControllerComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UFUNCTION(BlueprintCallable, Category=Cosmetics)
    void AddCharacterPart(const FHakCharacterPart& NewPart);

    UPROPERTY(EditAnywhere, Category=Cosmetics)
    TArray<FHakControllerCharacterPartEntry> CharacterParts;
};
```

1

```cpp
UHakControllerComponent_CharacterParts::UHakControllerComponent_CharacterParts(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
}

void UHakControllerComponent_CharacterParts::AddCharacterPart(const FHakCharacterPart& NewPart)
{
}
```

1 일단 더미로 넣어두자

☐ AddCharacterPart를 사용하여, B_PickRandomCharacter의 BeginPlay 로직을 우선 완성하자:

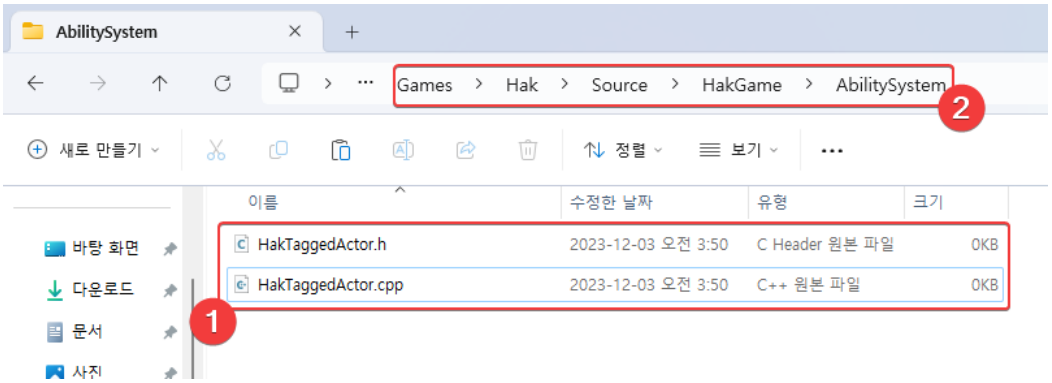


☐ New Part Class에는 B_Manny라는 BP를 생성할 것이다:

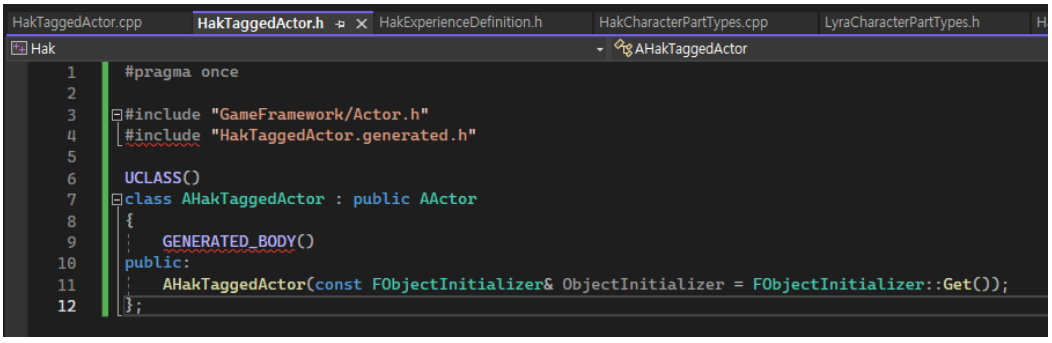
☐ 우선 해당 BP가 상속 받을 클래스를 정의하자: `HakTaggedActor`

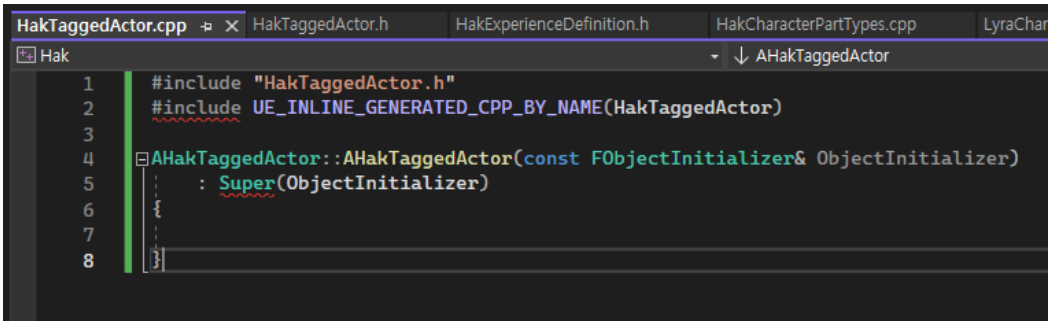
> 💡 사실 B_Manny는 그냥 Actor를 상속 받아도 되지만,
> HakTaggedActor로 Lyra와 똑같이 해주자.

☐ HakTaggedActor.h/.cpp를 아래와 같이 `AbilitySystem` 폴더에 생성하자:

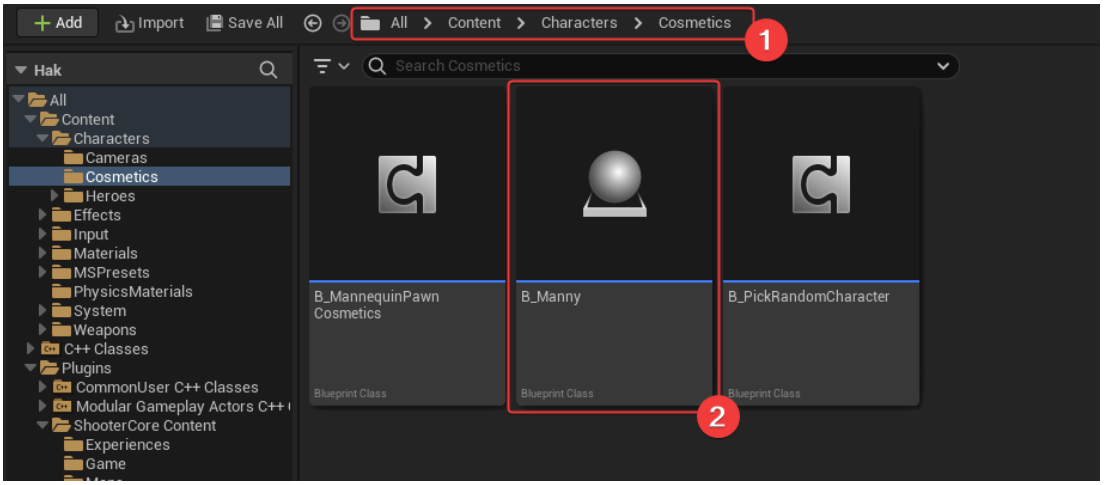


☐ HakTaggedActor를 구현하자:



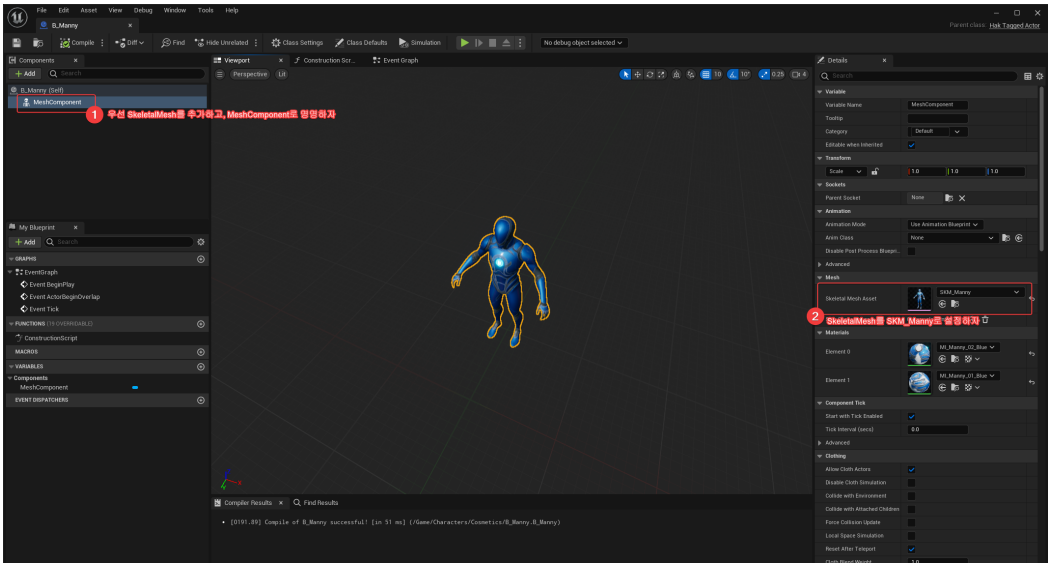


☐ 이제 B_Manny BP를 생성해보자:

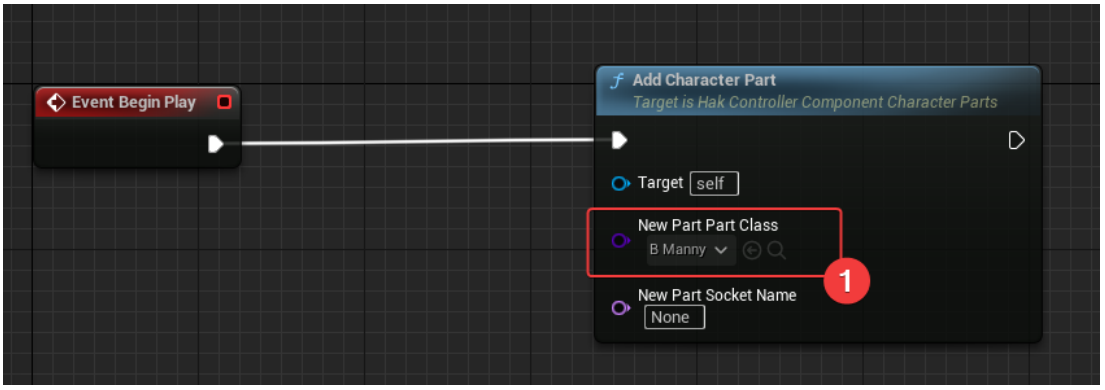


☐ B_Manny에 SkeletalMesh를 추가하자

- 우리의 BaseMesh가 될 대상이다

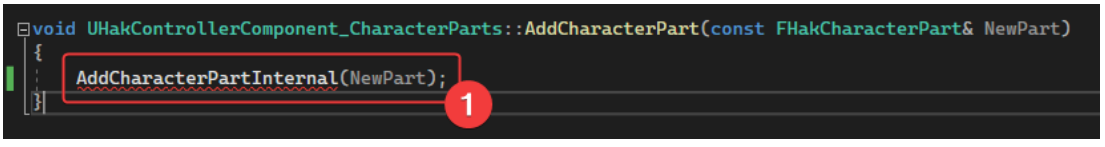


☐ 생성한 B_Manny를 활용하여, B_PickRandomCharacter의 BeginPlay의 로직을 완성하자:



참고로, Lyra에서는 해당 BP는 랜덤으로 B_Manny 혹은 B_Quinny를 선택한다. 우리는 전체적인 구조 파악이므로 해당 로직은 생략한다.

☐ AddCharacterPart() 로직을 C++ 구현하자:



☐ AddCharacterPartInternal()

```cpp
/**
 * Controller가 Pawn을 Possess했을 때, Pawn에 어떤 Cosmetic Actor 생성할지 결정하는 ControllerComponent:
 * - 필자는 캐릭터 파츠를 유저 관점에서 관리하는 Component로 이해한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakControllerComponent_CharacterParts : public UControllerComponent
{
    GENERATED_BODY()
public:
    UHakControllerComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    void AddCharacterPartInternal(const FHakCharacterPart& NewPart);

    UFUNCTION(BlueprintCallable, Category=Cosmetics)
    void AddCharacterPart(const FHakCharacterPart& NewPart);

    UPROPERTY(EditAnywhere, Category=Cosmetics)
    TArray<FHakControllerCharacterPartEntry> CharacterParts;
};
```

1

```cpp
#include "HakControllerComponent_CharacterParts.h"
#include "HakPawnComponent_CharacterParts.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakControllerComponent_CharacterParts)

UHakControllerComponent_CharacterParts::UHakControllerComponent_CharacterParts(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
}

void UHakControllerComponent_CharacterParts::AddCharacterPartInternal(const FHakCharacterPart& NewPart)
{
    FHakControllerCharacterPartEntry& NewEntry = CharacterParts.AddDefaulted_GetRef();
    NewEntry.Part = NewPart;

    if (UHakPawnComponent_CharacterParts* PawnCustomizer = GetPawnCustomizer())
    {
        NewEntry.Handle = PawnCustomizer->AddCharacterPart(NewPart);
    }
}

void UHakControllerComponent_CharacterParts::AddCharacterPart(const FHakCharacterPart& NewPart)
{
    AddCharacterPartInternal(NewPart);
}
```

1

2

- 우리는 아래 두 가지를 구현해야 한다:
  1. GetPawnCustomizer()
  2. HakPawnComponent_CharacterParts::AddCharacterPart()

☐ GetPawnCustomizer()

```cpp
/**
 * Controller가 Pawn을 Possess했을 때, Pawn에 어떤 Cosmetic Actor 생성할지 결정하는 ControllerComponent:
 * - 필자는 캐릭터 파츠를 유저 관점에서 관리하는 Component로 이해한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakControllerComponent_CharacterParts : public UControllerComponent
{
    GENERATED_BODY()
public:
    UHakControllerComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UHakPawnComponent_CharacterParts* GetPawnCustomizer() const;
    void AddCharacterPartInternal(const FHakCharacterPart& NewPart);

    UFUNCTION(BlueprintCallable, Category=Cosmetics)
    void AddCharacterPart(const FHakCharacterPart& NewPart);

    UPROPERTY(EditAnywhere, Category=Cosmetics)
    TArray<FHakControllerCharacterPartEntry> CharacterParts;
};
```

1

```cpp
#include "HakControllerComponent_CharacterParts.h"
#include "HakPawnComponent_CharacterParts.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakControllerComponent_CharacterParts)

UHakControllerComponent_CharacterParts::UHakControllerComponent_CharacterParts(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
}

UHakPawnComponent_CharacterParts* UHakControllerComponent_CharacterParts::GetPawnCustomizer() const
{
    if (APawn* ControlledPawn = GetPawn<APawn>())
    {
        // 생각해보면, 우리는 앞서 HakPawnComponent_CharacterParts를 상속받는 B_MannequinPawnCosmetics를 이미 B_Hero_ShooterMannequin에 추가하였다.
        // B_MannequinPawnCosmetics를 반환되길 기대한다
        return ControlledPawn->FindComponentByClass<UHakPawnComponent_CharacterParts>();
    }
    return nullptr;
}
```

- [ ] HakPawnComponent_CharacterParts::AddCharacterPart()

```cpp
/**
 * PawnComponent로서, Character Parts를 인스턴스화하여 관리한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakPawnComponent_CharacterParts : public UPawnComponent
{
    GENERATED_BODY()
public:
    UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    void AddCharacterPart(const FHakCharacterPart& NewPart);

    /** 인스턴스화 된 Character Parts */
    UPROPERTY()
    FHakCharacterPartList CharacterPartList;

    /** 애니메이션 적용을 위한 메시와 연결고리 */
    UPROPERTY(EditAnywhere, Category=Cosmetics)
    FHakAnimBodyStyleSelectionSet BodyMeshes;
};
```

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

UHakPawnComponent_CharacterParts::UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
}

void UHakPawnComponent_CharacterParts::AddCharacterPart(const FHakCharacterPart& NewPart)
{
    return CharacterPartList.AddEntry(NewPart);
}
```

- [ ] FHakCharacterPartList::AddEntry()

```cpp
/** HakPawnComponent_CharacterParts에서 실질적 Character Parts를 관리하는 클래스 */
USTRUCT(BlueprintType)
struct FHakCharacterPartList
{
    GENERATED_BODY()

    FHakCharacterPartList()
        : OwnerComponent(nullptr)
    {}

    FHakCharacterPartList(UHakPawnComponent_CharacterParts* InOwnerComponent)
        : OwnerComponent(InOwnerComponent)
    {}

    FHakCharacterPartHandle AddEntry(FHakCharacterPart NewPart);

    /** 현재 인스턴스화된 Character Part */
    UPROPERTY()
    TArray<FHakAppliedCharacterPartEntry> Entries;

    /** 해당 HakCharacterPartList의 Owner인 PawnComponent */
    UPROPERTY()
    TObjectPtr<UHakPawnComponent_CharacterParts> OwnerComponent;

    /** 앞서 보았던 PartHandle의 값을 할당 및 관리하는 변수 */
    int32 PartHandleCounter = 0;
};
```

1

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

FHakCharacterPartHandle FHakCharacterPartList::AddEntry(FHakCharacterPart NewPart)
{
    // PawnComponent의 CharacterPartList가 PartHandle를 관리하고, 이를 ControllerComponent_CharacterParts에 전달한다
    FHakCharacterPartHandle Result;
    Result.PartHandle = PartHandleCounter++;

    // Authority가 있다면, AppliedCharacterPartEntry를 Entries에 추가한다
    if (ensure(OwnerComponent && OwnerComponent->GetOwner() && OwnerComponent->GetOwner()->HasAuthority()))
    {
        FHakAppliedCharacterPartEntry& NewEntry = Entries.AddDefaulted_GetRef();
        NewEntry.Part = NewPart;
        NewEntry.PartHandle = Result.PartHandle;

        // 여기서 실제 Actor를 생성하고, OwnerComponent의 Owner Actor에 Actor끼리 RootComponent로 Attach 시킨다
        if (SpawnActorForEntry(NewEntry))
        {
            OwnerComponent->BroadcastChanged();
        }
    }
}

UHakPawnComponent_CharacterParts::UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
}
```

1

- [ ] SpawnActorForEntry()

```cpp
#include "HakPawnComponent_CharacterParts.h"
#include "GameFramework/Actor.h"
#include "Components/ChildActorComponent.h"
#include "Components/SceneComponent.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakPawnComponent_CharacterParts)

bool FHakCharacterPartList::SpawnActorForEntry(FHakAppliedCharacterPartEntry& Entry)
{
    bool bCreatedAnyActor = false;
    // 전달된 AppliedCharacterPartEntry의 Part Class가 제대로 세팅되어 있다면
    if (Entry.Part.PartClass != nullptr)
    {
        // OwnerComponent의 Owner에 속한 World를 반환
        UWorld* World = OwnerComponent->GetWorld();

        // HakPawnComponent_CharacterParts에 어느 Component에 붙일 것인지 결정한다:
        // - GetSceneComponentToAttachTo
        if (USceneComponent* ComponentToAttachTo = OwnerComponent->GetSceneComponentToAttachTo())
        {
            // 붙일 Component인 ComponentToAttachTo의 Bone 혹은 SocketName을 통해 어디에 붙일지 Transform을 계산한다
            const FTransform SpawnTransform = ComponentToAttachTo->GetSocketTransform(Entry.Part.SocketName);

            // 우리는 Actor-Actor의 결합이므로, ChildActorComponent를 활용한다
            UChildActorComponent* PartComponent = NewObject<UChildActorComponent>(OwnerComponent->GetOwner());
            PartComponent->SetupAttachment(ComponentToAttachTo, Entry.Part.SocketName);
            PartComponent->SetChildActorClass(Entry.Part.PartClass);
            // 참고로 RegisterComponent를 통해 마지막으로 RenderWorld인 FScene에 변경 내용을 전달한다 (혹은 생성한다)
            PartComponent->RegisterComponent();

            // ChildActorComponent에서 생성한 Actor를 반환하여
            if (AActor* SpawnedActor = PartComponent->GetChildActor())
            {
                // 해당 Actor가 Parent인 HakPawnComponent_CharacterParts의 Owner Actor보다 먼저 Tick이 실행되지 않도록 선행조건을 붙인다
                if (USceneComponent* SpawnedRootComponent = SpawnedActor->GetRootComponent())
                {
                    SpawnedRootComponent->AddTickPrerequisiteComponent(ComponentToAttachTo);
                }
            }
            Entry.SpawnedComponent = PartComponent;
            bCreatedAnyActor = true;
        }
    }

    return bCreatedAnyActor;
}
```

```cpp
/** HakPawnComponent_CharacterParts에서 실질적 Character Parts를 관리하는 클래스 */
USTRUCT(BlueprintType)
struct FHakCharacterPartList
{
    GENERATED_BODY()

    FHakCharacterPartList()
        : OwnerComponent(nullptr)
    {}

    FHakCharacterPartList(UHakPawnComponent_CharacterParts* InOwnerComponent)
        : OwnerComponent(InOwnerComponent)
    {}

    bool SpawnActorForEntry(FHakAppliedCharacterPartEntry& Entry);

    FHakCharacterPartHandle AddEntry(FHakCharacterPart NewPart);

    /** 현재 인스턴스화된 Character Part */
    UPROPERTY()
    TArray<FHakAppliedCharacterPartEntry> Entries;

    /** 해당 HakCharacterPartList의 Owner인 PawnComponent */
    UPROPERTY()
    TObjectPtr<UHakPawnComponent_CharacterParts> OwnerComponent;

    /** 앞서 보았던 PartHandle의 값을 할당 및 관리하는 변수 */
    int32 PartHandleCounter = 0;
};
```

- [ ] GetSceneComponentToAttachTo()

```cpp
/**
 * PawnComponent로서, Character Parts를 인스턴스화하여 관리한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakPawnComponent_CharacterParts : public UPawnComponent
{
    GENERATED_BODY()
public:
    UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    USkeletalMeshComponent* GetParentMeshComponent() const;
    USceneComponent* GetSceneComponentToAttachTo() const;

    FHakCharacterPartHandle AddCharacterPart(const FHakCharacterPart& NewPart);

    /** 인스턴스화 된 Character Parts */
    UPROPERTY()
    FHakCharacterPartList CharacterPartList;

    /** 애니메이션 적용을 위한 메시와 연결고리 */
    UPROPERTY(EditAnywhere, Category=Cosmetics)
    FHakAnimBodyStyleSelectionSet BodyMeshes;
};
```

```cpp
USkeletalMeshComponent* UHakPawnComponent_CharacterParts::GetParentMeshComponent() const
{
    // Character를 활용하여, 최상위 SkeletalMesh를 반환한다
    if (AActor* OwnerActor = GetOwner())
    {
        if (ACharacter* OwningCharacter = Cast<ACharacter>(OwnerActor))
        {
            if (USkeletalMeshComponent* MeshComponent = OwningCharacter->GetMesh())
            {
                return MeshComponent;
            }
        }
    }
    return nullptr
}

USceneComponent* UHakPawnComponent_CharacterParts::GetSceneComponentToAttachTo() const
{
    // Parent에 SkeletalMeshComponent가 있으면 반환하고
    if (USkeletalMeshComponent* MeshComponent = GetParentMeshComponent())
    {
        return MeshComponent;
    }
    // 그리고 RootComponent도 확인하고
    else if (AActor* OwnerActor = GetOwner())
    {
        return OwnerActor->GetRootComponent();
    }
    // 그냥 nullptr 반환
    return nullptr;
}
```

☐ BroadcastChanged()

```cpp
/**
 * PawnComponent로서, Character Parts를 인스턴스화하여 관리한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakPawnComponent_CharacterParts : public UPawnComponent
{
    GENERATED_BODY()
public:
    UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    USkeletalMeshComponent* GetParentMeshComponent() const;
    USceneComponent* GetSceneComponentToAttachTo() const;
    void BroadcastChanged();
    FHakCharacterPartHandle AddCharacterPart(const FHakCharacterPart& NewPart);

    /** 인스턴스화 된 Character Parts */
    UPROPERTY()
    FHakCharacterPartList CharacterPartList;

    /** 애니메이션 적용을 위한 메시와 연결고리 */
    UPROPERTY(EditAnywhere, Category=Cosmetics)
    FHakAnimBodyStyleSelectionSet BodyMeshes;
};
```

```cpp
void UHakPawnComponent_CharacterParts::BroadcastChanged()
{
    const bool bReinitPose = true;

    // 현재 Owner의 SkeletalMeshComponent를 반환한다
    if (USkeletalMeshComponent* MeshComponent = GetParentMeshComponent())
    {
        // BodyMeshes를 통해, GameplayTag를 활용하여, 알맞은 SkeletalMesh로 재설정해준다
        const FGameplayTagContainer MergedTags = GetCombinedTags(FGameplayTag());
        USkeletalMesh* DesiredMesh = BodyMeshes.SelectBestBodyStyle(MergedTags);

        // SkeletalMesh를 초기화 및 Animation 초기화 시켜준다
        MeshComponent->SetSkeletalMesh(DesiredMesh, bReinitPose);

        // PhysicsAsset을 초기화해준다
        if (UPhysicsAsset* PhysicsAsset = BodyMeshes.ForcedPhysicsAsset)
        {
            MeshComponent->SetPhysicsAsset(PhysicsAsset, bReinitPose);
        }
    }
}
```

☐ GetCombinedTags() 구현:

```cpp
/**
 * PawnComponent로서, Character Parts를 인스턴스화하여 관리한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakPawnComponent_CharacterParts : public UPawnComponent
{
    GENERATED_BODY()
public:
    UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    USkeletalMeshComponent* GetParentMeshComponent() const;
    USceneComponent* GetSceneComponentToAttachTo() const;
    FGameplayTagContainer GetCombinedTags(FGameplayTag RequiredPrefix) const;
    void BroadcastChanged();
    FHakCharacterPartHandle AddCharacterPart(const FHakCharacterPart& NewPart);

    /** 인스턴스화 된 Character Parts */
    UPROPERTY()
    FHakCharacterPartList CharacterPartList;

    /** 애니메이션 적용을 위한 메시와 연결고리 */
    UPROPERTY(EditAnywhere, Category=Cosmetics)
    FHakAnimBodyStyleSelectionSet BodyMeshes;
};
```

1

```cpp
FGameplayTagContainer UHakPawnComponent_CharacterParts::GetCombinedTags(FGameplayTag RequiredPrefix) const
{
    // 현재 장착된 CharacterPartList의 Merge된 Tags를 반환한다
    FGameplayTagContainer Result = CharacterPartList.CollectCombinedTags();
    if (RequiredPrefix.IsValid())
    {
        // 만약 GameplayTag를 통해 필터링할 경우, 필터링해서 진행한다
        return Result.Filter(FGameplayTagContainer(RequiredPrefix));
    }
    else
    {
        // 필터링할 GameplayTag가 없으면 그냥 반환
        return Result;
    }
}
```

☐ HakCharacterPartList::CollectCombinedTags()

```cpp
/** HakPawnComponent_CharacterParts에서 실질적 Character Parts를 관리하는 클래스 */
USTRUCT(BlueprintType)
struct FHakCharacterPartList
{
    GENERATED_BODY()

    FHakCharacterPartList()
        : OwnerComponent(nullptr)
    {}

    FHakCharacterPartList(UHakPawnComponent_CharacterParts* InOwnerComponent)
        : OwnerComponent(InOwnerComponent)
    {}

    bool SpawnActorForEntry(FHakAppliedCharacterPartEntry& Entry);

    FHakCharacterPartHandle AddEntry(FHakCharacterPart NewPart);

    FGameplayTagContainer CollectCombinedTags() const;

    /** 현재 인스턴스화된 Character Part */
    UPROPERTY()
    TArray<FHakAppliedCharacterPartEntry> Entries;

    /** 해당 HakCharacterPartList의 Owner인 PawnComponent */
    UPROPERTY()
    TObjectPtr<UHakPawnComponent_CharacterParts> OwnerComponent;

    /** 앞서 보았던 PartHandle의 값을 할당 및 관리하는 변수 */
    int32 PartHandleCounter = 0;
};
```

1

```cpp
FGameplayTagContainer FHakCharacterPartList::CollectCombinedTags() const
{
    FGameplayTagContainer Result;

    // Entries를 순회하며,
    for (const FHakAppliedCharacterPartEntry& Entry : Entries)
    {
        // Part Actor가 생성되어 SpawnedComponent에 캐싱되어 있으면
        if (Entry.SpawnedComponent)
        {
            // 해당 Actor의 IGameplayTagAssetInterface를 통해 GameplayTag를 검색한다:
            // - 현재 우리의 TaggedActor는 IGameplayTagAssetInterface를 상속받지 않으므로 그냥 넘어갈 것이다
            // - 후일 여러분들이 각 Part에 대해 GameplayTag를 넣고 싶다면 이걸 상속받아 정의해야 한다:
            //   - 예로 들어, 특정 Lv100이상 장착 가능한 장비를 만들고 싶다면 넣어야겠다
            if (IGameplayTagAssetInterface* TagInterface = Cast<IGameplayTagAssetInterface>(Entry.SpawnedComponent->GetChildActor()))
            {
                TagInterface->GetOwnedGameplayTags(Result);
            }
        }
    }
    return Result;
}
```

- [ ] HakAnimBodyStyleSelectionSet::SelectBestBodyStyle()

```cpp
USTRUCT(BlueprintType)
struct FHakAnimBodyStyleSelectionSet
{
    GENERATED_BODY()

    /** GameplayTag를 통해 (CosmeticTags), Mesh Rules에 따라 알맞은 BodyStyle를 변환한다 */
    USkeletalMesh* SelectBestBodyStyle(const FGameplayTagContainer& CosmeticTags) const;

    /** AnimLayer 적용할 SkeletalMesh를 들고 있음 -> Animation-Mesh간 Rules을 MeshRules라고 생각하면 됨 */
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TArray<FHakAnimBodyStyleSelectionEntry> MeshRules;

    /** 그냥 디폴트로 적용할 SkeletalMesh */
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TObjectPtr<USkeletalMesh> DefaultMesh = nullptr;

    /** Physics Asset은 하나로 동일함 -> 즉 모든 Animation의 Physics 속성은 공유함 */
    UPROPERTY(EditAnywhere)
    TObjectPtr<UPhysicsAsset> ForcedPhysicsAsset = nullptr;
};
```

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

USkeletalMesh* FHakAnimBodyStyleSelectionSet::SelectBestBodyStyle(const FGameplayTagContainer& CosmeticTags) const
{
    // MeshRule을 순회하며, CosmeticTags 요구 조건에 맞는 MeshRule을 찾아 SkeletalMesh를 반환한다
    for (const FHakAnimBodyStyleSelectionEntry& Rule : MeshRules)
    {
        if ((Rule.Mesh) && CosmeticTags.HasAll(Rule.RequiredTags))
        {
            return Rule.Mesh;
        }
    }
    return DefaultMesh;
}
```

이렇게 복잡한 BodyStyle을 구성하는 이유는 Animation에 대한 정보가 장착 파츠에 의해 Animation Layer로 결정되기 때문이다. 총을 장착하였다면, 총 장착 Animation Layer로 재생되야 하는데 이를 위해 알맞는 SkeletalMesh를 설정이 필요하고 , 이를 위해 MeshRule을 가지고 있는 AnimBodyStyleSelectionSet가 필요하다:

약간 HTML과 CSS의 관계를 생각해보면 좋을거 같다:
- HTML은 뼈대이지만, CSS에 따라 웹페이지의 외형이 달라진다 (애니메이션도!)

- 이제 Add에 집중하였다, Remove를 구현하며, 한 번 더 역으로 이해해보자:
  - **<u>당연히 구현은 Add가 있으면 Remove가 있어야 한다!</u>**

☐ HakPawnComponent_CharacterParts::RemoveCharacterPart

```cpp
/**
 * PawnComponent로서, Character Parts를 인스턴스화하여 관리한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakPawnComponent_CharacterParts : public UPawnComponent
{
    GENERATED_BODY()
public:
    UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    USkeletalMeshComponent* GetParentMeshComponent() const;
    USceneComponent* GetSceneComponentToAttachTo() const;
    FGameplayTagContainer GetCombinedTags(FGameplayTag RequiredPrefix) const;
    void BroadcastChanged();

    FHakCharacterPartHandle AddCharacterPart(const FHakCharacterPart& NewPart);
    void RemoveCharacterPart(FHakCharacterPartHandle Handle);

    /** 인스턴스화 된 Character Parts */
    UPROPERTY()
    FHakCharacterPartList CharacterPartList;

    /** 애니메이션 적용을 위한 메시와 연결고리 */
    UPROPERTY(EditAnywhere, Category=Cosmetics)
    FHakAnimBodyStyleSelectionSet BodyMeshes;
};
```

1

```cpp
void UHakPawnComponent_CharacterParts::RemoveCharacterPart(FHakCharacterPartHandle Handle)
{
    CharacterPartList.RemoveEntry(Handle);
}
```

- [ ] HakCharacterPartList::RemoveEntry()

```cpp
/** HakPawnComponent_CharacterParts에서 실질적 Character Parts를 관리하는 클래스 */
USTRUCT(BlueprintType)
struct FHakCharacterPartList
{
    GENERATED_BODY()

    FHakCharacterPartList()
        : OwnerComponent(nullptr)
    {}

    FHakCharacterPartList(UHakPawnComponent_CharacterParts* InOwnerComponent)
        : OwnerComponent(InOwnerComponent)
    {}

    bool SpawnActorForEntry(FHakAppliedCharacterPartEntry& Entry);

    FHakCharacterPartHandle AddEntry(FHakCharacterPart NewPart);
    void RemoveEntry(FHakCharacterPartHandle Handle);

    FGameplayTagContainer CollectCombinedTags() const;

    /** 현재 인스턴스화된 Character Part */
    UPROPERTY()
    TArray<FHakAppliedCharacterPartEntry> Entries;

    /** 해당 HakCharacterPartList의 Owner인 PawnComponent */
    UPROPERTY()
    TObjectPtr<UHakPawnComponent_CharacterParts> OwnerComponent;

    /** 앞서 보았던 PartHandle의 값을 할당 및 관리하는 변수 */
    int32 PartHandleCounter = 0;
};
```

1

```cpp
void FHakCharacterPartList::RemoveEntry(FHakCharacterPartHandle Handle)
{
    for (auto EntryIt = Entries.CreateIterator(); EntryIt; ++EntryIt)
    {
        FHakAppliedCharacterPartEntry& Entry = *EntryIt;

        // 제거할 경우, PartHandle을 활용한다
        if (Entry.PartHandle == Handle.PartHandle)
        {
            DestroyActorForEntry(Entry);
        }
    }
}
```

☐ HakCharacterPartList::DestroyActorForEntry()

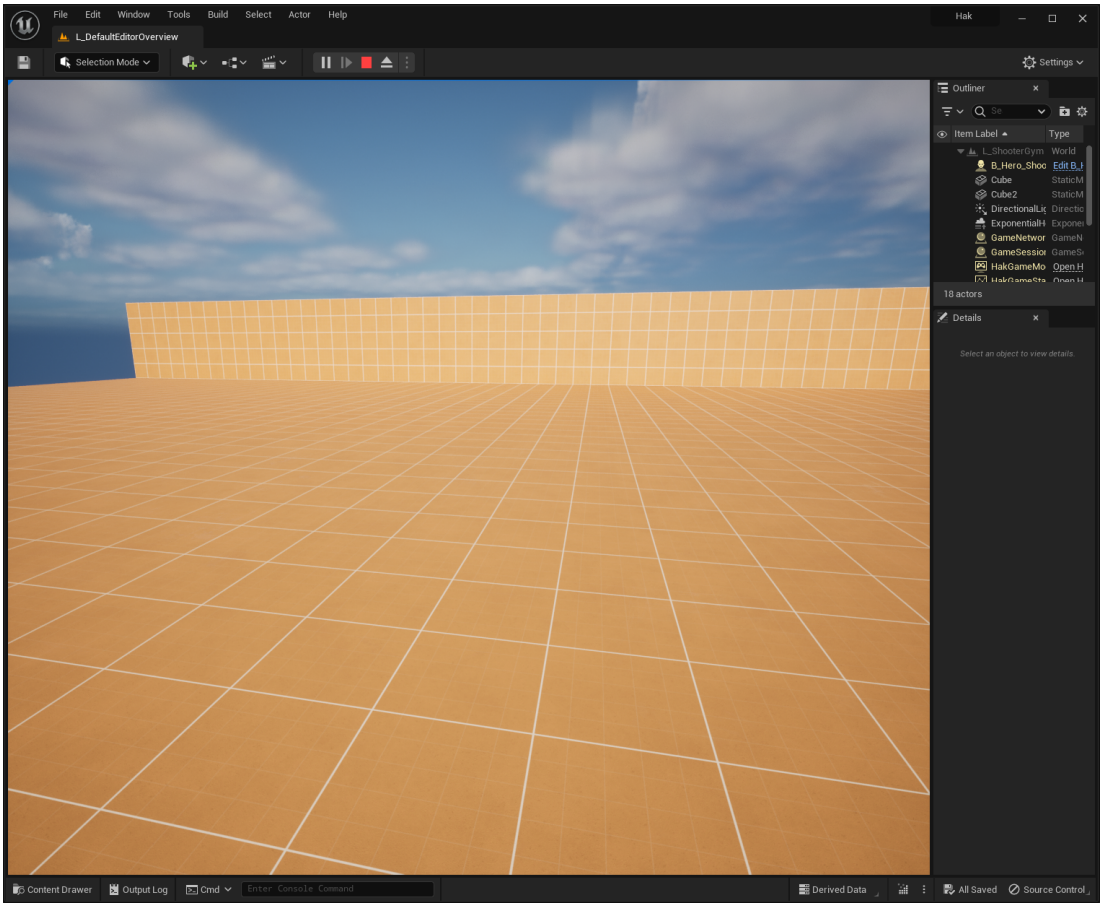
```cpp
/** HakPawnComponent_CharacterParts에서 실질적 Character Parts를 관리하는 클래스 */
USTRUCT(BlueprintType)
struct FHakCharacterPartList
{
    GENERATED_BODY()

    FHakCharacterPartList()
        : OwnerComponent(nullptr)
    {}

    FHakCharacterPartList(UHakPawnComponent_CharacterParts* InOwnerComponent)
        : OwnerComponent(InOwnerComponent)
    {}

    bool SpawnActorForEntry(FHakAppliedCharacterPartEntry& Entry);
    void DestroyActorForEntry(FHakAppliedCharacterPartEntry& Entry);

    FHakCharacterPartHandle AddEntry(FHakCharacterPart NewPart);
    void RemoveEntry(FHakCharacterPartHandle Handle);

    FGameplayTagContainer CollectCombinedTags() const;

    /** 현재 인스턴스화된 Character Part */
    UPROPERTY()
    TArray<FHakAppliedCharacterPartEntry> Entries;

    /** 해당 HakCharacterPartList의 Owner인 PawnComponent */
    UPROPERTY()
    TObjectPtr<UHakPawnComponent_CharacterParts> OwnerComponent;

    /** 앞서 보았던 PartHandle의 값을 할당 및 관리하는 변수 */
    int32 PartHandleCounter = 0;
};
```

1

```cpp
void FHakCharacterPartList::DestroyActorForEntry(FHakAppliedCharacterPartEntry& Entry)
{
    if (Entry.SpawnedComponent)
    {
        Entry.SpawnedComponent->DestroyComponent();
        Entry.SpawnedComponent = nullptr;
    }
}
```

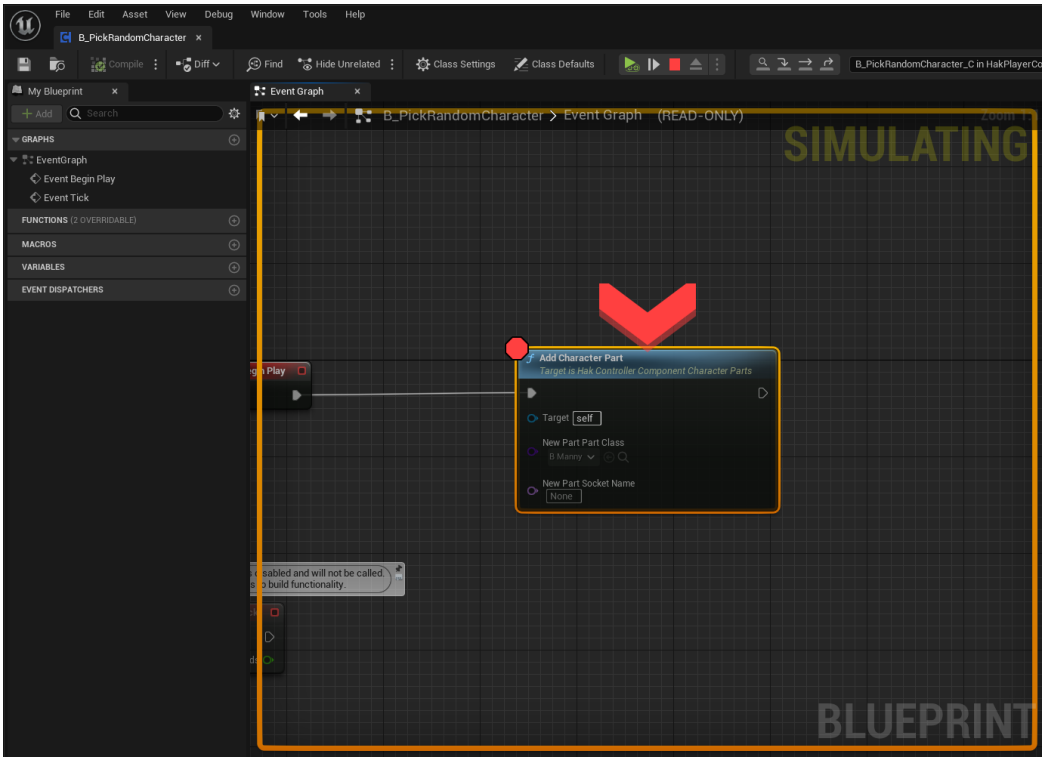
- ControllerComponent_CharacterParts는 유저가 조정하는 Pawn이 변경되었을 경우, Character Part를 유지하고 싶다면 재적용을 위해 Pawn에 있는 Parts를 다시 재적용한다:
    - 반대로, Character Part를 다 없애고 싶다면, ControllerComponent_CharacterParts의 상태도 비워줘야한다
    - 이 경우, Remove가 필요하다, 이거는 여러분들에게 맡겨보도록 할 것이다 (후일 구현할지도?)

☐ 흠.. 아무 변화가 없다…



☐ 우선 앞서, B_PickRandomCharacter에 AddCharacterPart에 넣었다:

- 여기에 Breakpoint를 넣어서 잡히는지 확인해보자:



- 잡힌다! → **<u>이는 B_PickRandomCharacter가 HakPlayerController에 잘 이식되어 BeginPlay가 불렸다는 것</u>**이다 (GameFeatrueAction은 잘 작동했다는 것을 의미한다)

☐ 무슨 상황이 벌이지고 있는지 ControllerComponent_CharacterParts의 AddCharacterPart()를 디버깅해보자:

- 아래와 같이 Debug Breakpoint를 넣어주자:

```
21  void UHakControllerComponent_CharacterParts::AddCharacterPartInternal(const FHakCharacterPart& NewPart)
22  {
23      FHakControllerCharacterPartEntry& NewEntry = CharacterParts.AddDefaulted_GetRef();
24      NewEntry.Part = NewPart;
25
26      if (UHakPawnComponent_CharacterParts* PawnCustomizer = GetPawnCustomizer())
27      {
28          NewEntry.Handle = PawnCustomizer->AddCharacterPart(NewPart);
29      }
30  }
```

- 잉? 넘어간다!

```
21  void UHakControllerComponent_CharacterParts::AddCharacterPartInternal(const FHakCharacterPart& NewPart)
22  {
23      FHakControllerCharacterPartEntry& NewEntry = CharacterParts.AddDefaulted_GetRef();
24      NewEntry.Part = NewPart;
25
26      if (UHakPawnComponent_CharacterParts* PawnCustomizer = GetPawnCustomizer())
27      {
28          NewEntry.Handle = PawnCustomizer->AddCharacterPart(NewPart);
29      }
30  } ≤ 2ms elapsed
```

- 그럼 GetPawnCustomizer가 실패했다는 것을 의미한다:

- Breakpoint가 안 잡히니 아래와 같이,
  PRAGMA_DISABLE_OPTMIZATION을 활용하자:

```cpp
PRAGMA_DISABLE_OPTIMIZATION
UHakPawnComponent_CharacterParts* UHakControllerComponent_CharacterParts::GetPawnCustomizer() const
{
    if (APawn* ControlledPawn = GetPawn<APawn>())
    {
        // 생각해보면, 우리는 앞서 HakPawnComponent_CharacterParts를 상속받는 B_MannequinPawnCosmetics를 이미 B_Hero_ShooterMannequin에 추가하였다.
        // B_MannequinPawnCosmetics를 반환되길 기대한다
        return ControlledPawn->FindComponentByClass<UHakPawnComponent_CharacterParts>();
    }
    return nullptr;
}
PRAGMA_ENABLE_OPTIMIZATION
```

- Controller에 Possess가 된 Pawn이 없다는 것이다:

```cpp
PRAGMA_DISABLE_OPTIMIZATION
UHakPawnComponent_CharacterParts* UHakControllerComponent_CharacterParts::GetPawnCustomizer() const
{
    if (APawn* ControlledPawn = GetPawn<APawn>())
    {
        // 생각해보면, 우리는 앞서 HakPawnComponent_CharacterParts를 상속받는 B_MannequinPawnCosmetics를 이미 B_Hero_ShooterMannequin에 추가하였다.
        // B_MannequinPawnCosmetics를 반환되길 기대한다
        return ControlledPawn->FindComponentByClass<UHakPawnComponent_CharacterParts>();
    }
    return nullptr;  ≤ 5ms elapsed
}
PRAGMA_ENABLE_OPTIMIZATION
```

- **뭔가 우리에게 다른 방법이 필요하다는 것을 의미한다...**

OnPossessedPawnChanged

▼ **펼치기**

- 현재 PlayerController가 Pawn을 Possess할 시점에 맞추어, AddCharacterPart를 진행해야하는데, 이에 대한 어려움을 우리가 겪고 있다:
  - **만약 OnPossess하는 시점에 우리가 Delegate를 걸어, AddCharacterPart를 실행하면 어떨까?**

☐ OnPossessedPawnChanged 정의:

```cpp
/**
 * Controller가 Pawn을 Possess했을 때, Pawn에 어떤 Cosmetic Actor 생성할지 결정하는 ControllerComponent:
 * - 필자는 캐릭터 파츠를 유저 관점에서 관리하는 Component로 이해한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakControllerComponent_CharacterParts : public UControllerComponent
{
    GENERATED_BODY()
public:
    UHakControllerComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UHakPawnComponent_CharacterParts* GetPawnCustomizer() const;
    void AddCharacterPartInternal(const FHakCharacterPart& NewPart);

    UFUNCTION(BlueprintCallable, Category=Cosmetics)
    void AddCharacterPart(const FHakCharacterPart& NewPart);

    /** UFUNCTION으로 정의하지 않으면 AddDynamic이 되지 않는다! */
    UFUNCTION()
    void OnPossessedPawnChanged(APawn* OldPawn, APawn* NewPawn);

    UPROPERTY(EditAnywhere, Category=Cosmetics)
    TArray<FHakControllerCharacterPartEntry> CharacterParts;
};
```

```cpp
void UHakControllerComponent_CharacterParts::OnPossessedPawnChanged(APawn* OldPawn, APawn* NewPawn)
{

}
```

- [ ] AController::OnPossessedPawnChanged의 Delegate로 연결하기:
  - BeginPlay()와 EndPlay()을 쌍을 맞추어 정의해주자:

```cpp
/**
 * Controller가 Pawn을 Possess했을 때, Pawn에 어떤 Cosmetic Actor 생성할지 결정하는 ControllerComponent:
 * - 필자는 캐릭터 파츠를 유저 관점에서 관리하는 Component로 이해한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakControllerComponent_CharacterParts : public UControllerComponent
{
    GENERATED_BODY()
public:
    UHakControllerComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    virtual void BeginPlay() override;
    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) override;

    UHakPawnComponent_CharacterParts* GetPawnCustomizer() const;
    void AddCharacterPartInternal(const FHakCharacterPart& NewPart);

    UFUNCTION(BlueprintCallable, Category=Cosmetics)
    void AddCharacterPart(const FHakCharacterPart& NewPart);

    /** UFUNCTION으로 정의하지 않으면 AddDynamic이 되지 않는다! */
    UFUNCTION()
    void OnPossessedPawnChanged(APawn* OldPawn, APawn* NewPawn);

    UPROPERTY(EditAnywhere, Category=Cosmetics)
    TArray<FHakControllerCharacterPartEntry> CharacterParts;
};
```

1

```cpp
void UHakControllerComponent_CharacterParts::BeginPlay()
{
    Super::BeginPlay();

    if (HasAuthority())
    {
        if (AController* OwningController = GetController<AController>())
        {
            OwningController->OnPossessedPawnChanged.AddDynamic(this, &ThisClass::OnPossessedPawnChanged);
        }
    }
}

void UHakControllerComponent_CharacterParts::EndPlay(const EEndPlayReason::Type EndPlayReason)
{
    RemoveAllCharacterParts();
    Super::EndPlay(EndPlayReason);
}
```

1 우리는 Controller에 우리의 OnPossessedPawnChanged를 Delegate에 연결해주었다

- [ ] RemoveAllCharacterParts()

```cpp
/**
 * Controller가 Pawn을 Possess했을 때, Pawn에 어떤 Cosmetic Actor 생성할지 결정하는 ControllerComponent:
 * - 필자는 캐릭터 파츠를 유저 관점에서 관리하는 Component로 이해한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakControllerComponent_CharacterParts : public UControllerComponent
{
    GENERATED_BODY()
public:
    UHakControllerComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    virtual void BeginPlay() override;
    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) override;

    UHakPawnComponent_CharacterParts* GetPawnCustomizer() const;
    void AddCharacterPartInternal(const FHakCharacterPart& NewPart);

    UFUNCTION(BlueprintCallable, Category=Cosmetics)
    void AddCharacterPart(const FHakCharacterPart& NewPart);

    void RemoveAllCharacterParts();

    /** UFUNCTION으로 정의하지 않으면 AddDynamic이 되지 않는다! */
    UFUNCTION()
    void OnPossessedPawnChanged(APawn* OldPawn, APawn* NewPawn);

    UPROPERTY(EditAnywhere, Category=Cosmetics)
    TArray<FHakControllerCharacterPartEntry> CharacterParts;
};
```

```cpp
void UHakControllerComponent_CharacterParts::RemoveAllCharacterParts()
{
    if (UHakPawnComponent_CharacterParts* PawnCustomizer = GetPawnCustomizer())
    {
        for (FHakControllerCharacterPartEntry& Entry : CharacterParts)
        {
            PawnCustomizer->RemoveCharacterPart(Entry.Handle);
        }
    }
    CharacterParts.Reset();
}
```

☐ OnPossessedPawnChanged 구현:

```cpp
void UHakControllerComponent_CharacterParts::OnPossessedPawnChanged(APawn* OldPawn, APawn* NewPawn)
{
    // 이전 OldPawn에 대해서는 Character Parts를 전부 제거해주자
    if (UHakPawnComponent_CharacterParts* OldCustomizer = OldPawn ? OldPawn->FindComponentByClass<UHakPawnComponent_CharacterParts>() : nullptr)
    {
        for (FHakControllerCharacterPartEntry& Entry : CharacterParts)
        {
            OldCustomizer->RemoveCharacterPart(Entry.Handle);
            Entry.Handle.Reset();
        }
    }

    // 새로운 Pawn에 대해서 기존 Controller가 가지고 있는 Character Parts를 추가해주자
    if (UHakPawnComponent_CharacterParts* NewCustomizer = NewPawn ? NewPawn->FindComponentByClass<UHakPawnComponent_CharacterParts>() : nullptr)
    {
        for (FHakControllerCharacterPartEntry& Entry : CharacterParts)
        {
            check(!Entry.Handle.IsValid());
            Entry.Handle = NewCustomizer->AddCharacterPart(Entry.Part);
        }
    }
}
```

이를 통해 우리는 **이제 자동적으로 Controller에서 Pawn을 변경할 때마다, CharacterParts가 Pawn에 자동적으로 업데이트할 준비가 되었다.**

☐ 아래와 같이 크래시가 발생하면?

```cpp
FHakCharacterPartHandle FHakCharacterPartList::AddEntry(FHakCharacterPart NewPart)
{
    // PawnComponent의 CharacterPartList가 PartHandle를 관리하고, 이를 ControllerComponent_CharacterParts에 전달한다
    FHakCharacterPartHandle Result;
    Result.PartHandle = PartHandleCounter++;

    // Authority가 있다면, AppliedCharacterPartEntry를 Entries에 추가한다
    if (ensure(OwnerComponent && OwnerComponent->GetOwner() && OwnerComponent->GetOwner()->HasAuthority()))
    {
        FHakAppliedCharacterPartEntry& NewEntry = Entries.A
        NewEntry.Part = NewPart;
        NewEntry.PartHandle = Result.PartHandle;

        // 여기서 실제 Actor를 생성하고, OwnerComponent의 Owne
        if (SpawnActorForEntry(NewEntry))
        {
            // BroadcastChanged를 통해, OwnerComponent에서 Owner의 SkeletalMeshComponent를 활용하여, Animation 및 Physics를 Re-initialize해준다
            OwnerComponent->BroadcastChanged();
        }
    }

    return Result;
}
```

Breakpoint Instruction Executed

A breakpoint instruction (__debugbreak() statement or a similar call) was executed in UnrealEditor.exe.

Show Call Stack | Copy Details | Start Live Share session...

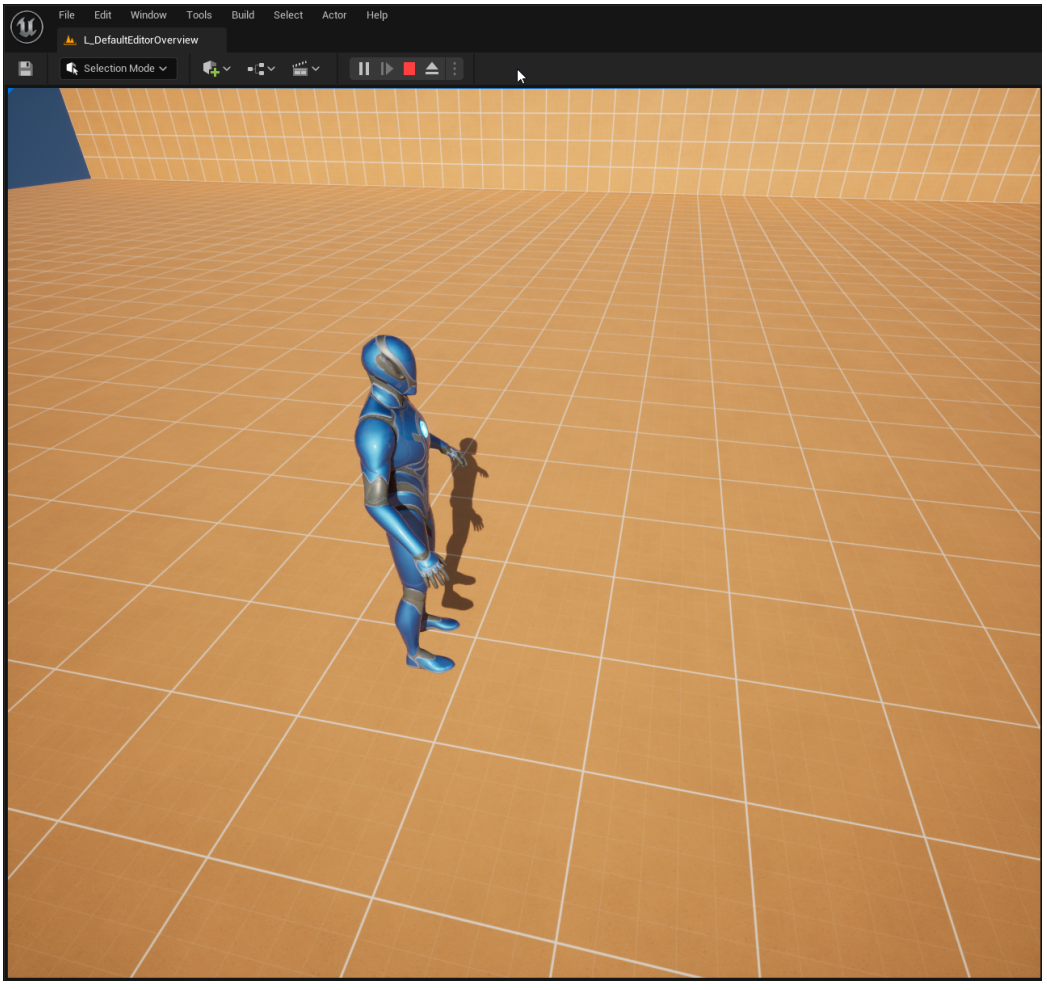
- 아래와 같이 우리가 OwnerComponent를 설정하지 않았음:

```cpp
UHakPawnComponent_CharacterParts::UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
    , CharacterPartList(this)
{
}
```

1

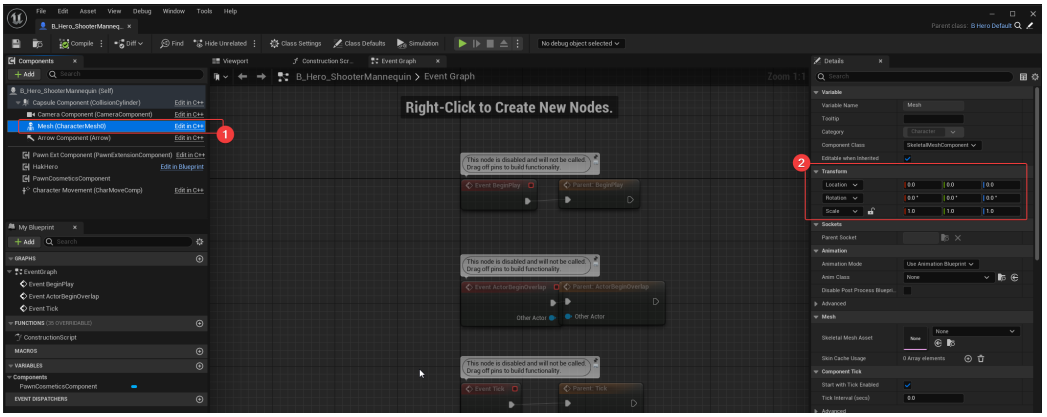
☐ 아래와 같이 드디어 캐릭터가 나왔다:

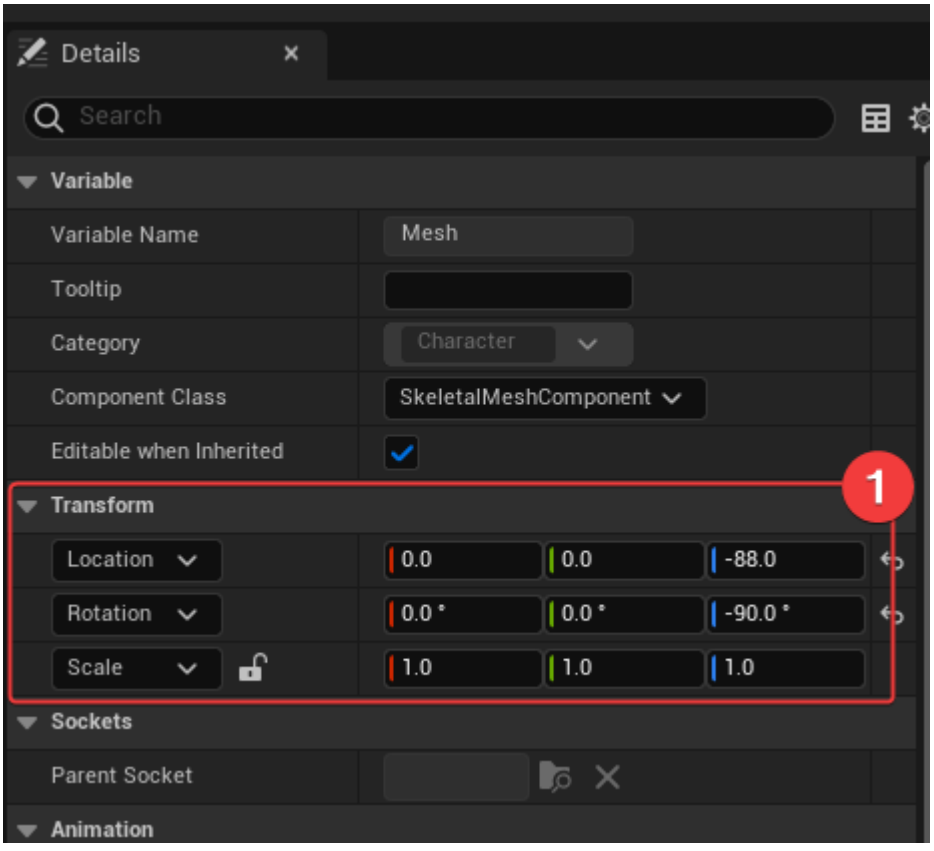
- 허나 아직 뭔가 캐릭터 Transform이 이상한거 같다…



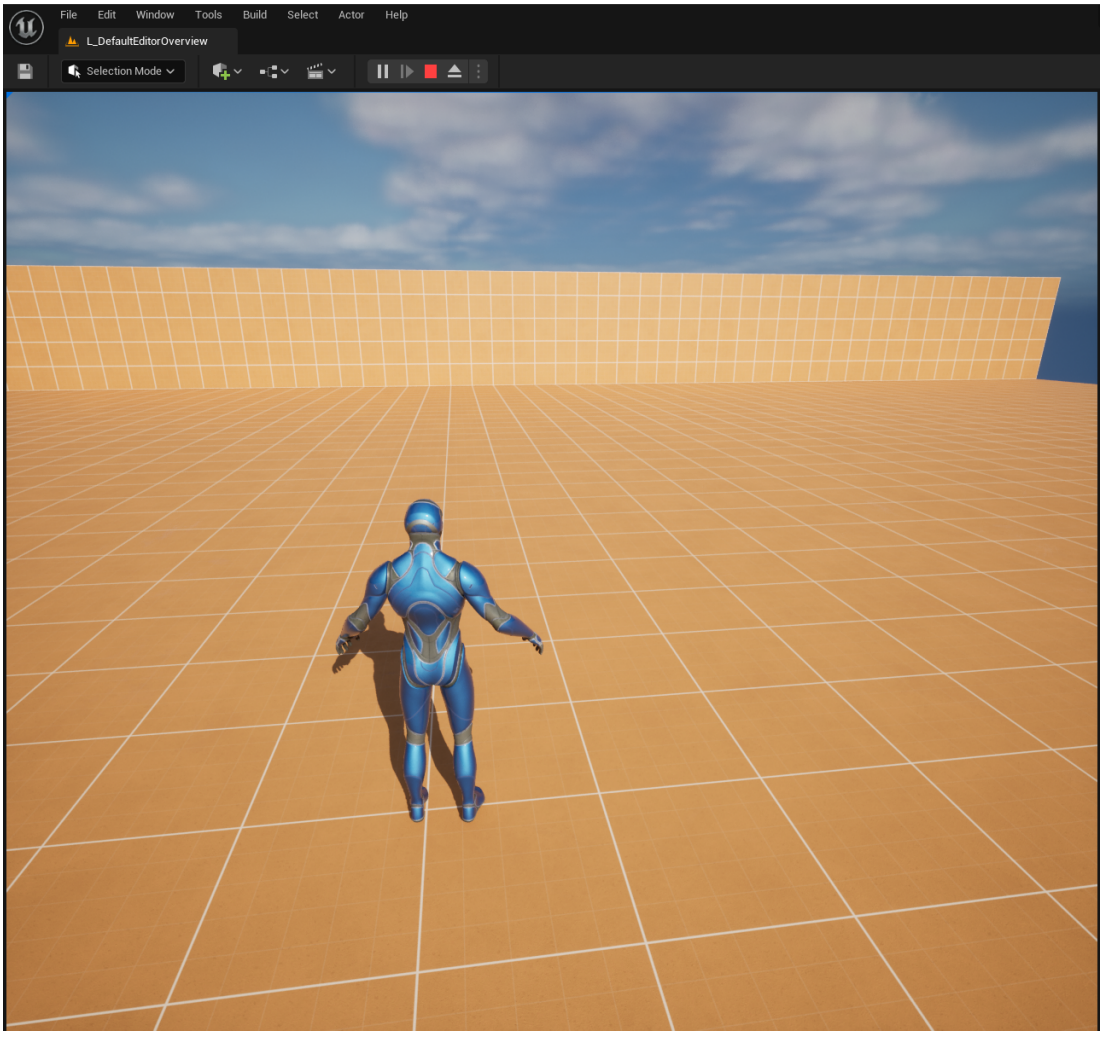
☐ B_Hero_ShooterMannequin을 확인해보자:

- 아래의 Mesh의 Transform을 수정하자:





☐ 이제 우리가 의도한 결과대로 나오는거 같다:

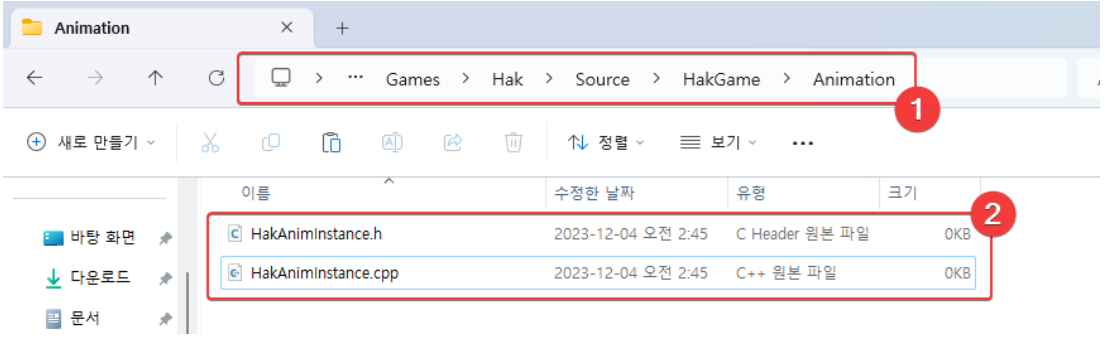


HakAnimInstance

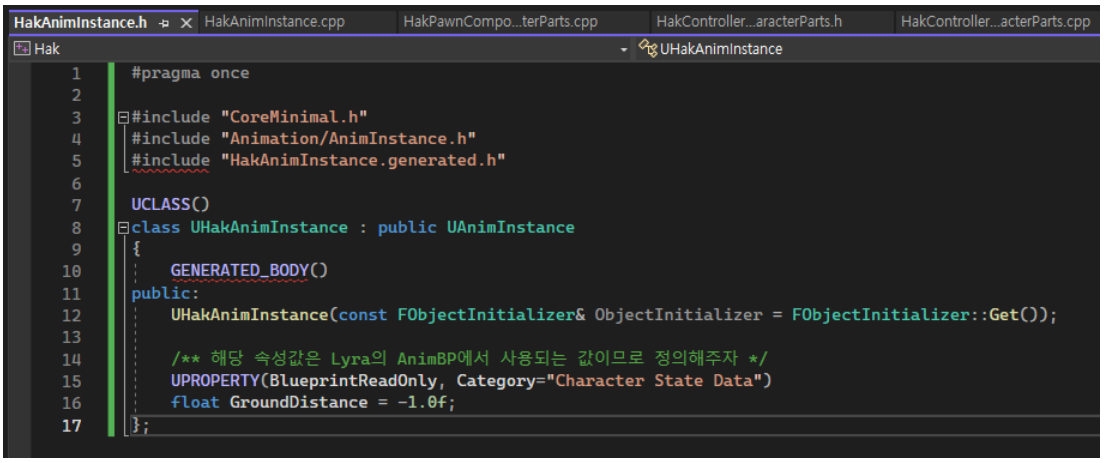
▼ **펼치기**

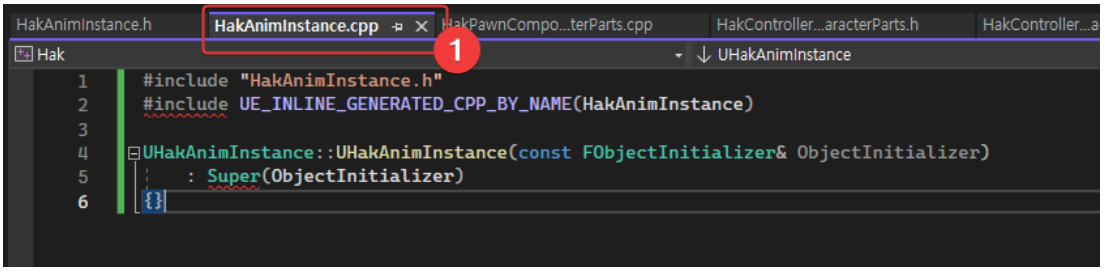
- Lyra의 Animation을 Migrate를 시키기 위해서, 우리는 AnimInstance를 정의하여 Override 할 필요성이 있다.

☐ HakAnimInstance.h/.cpp 생성:



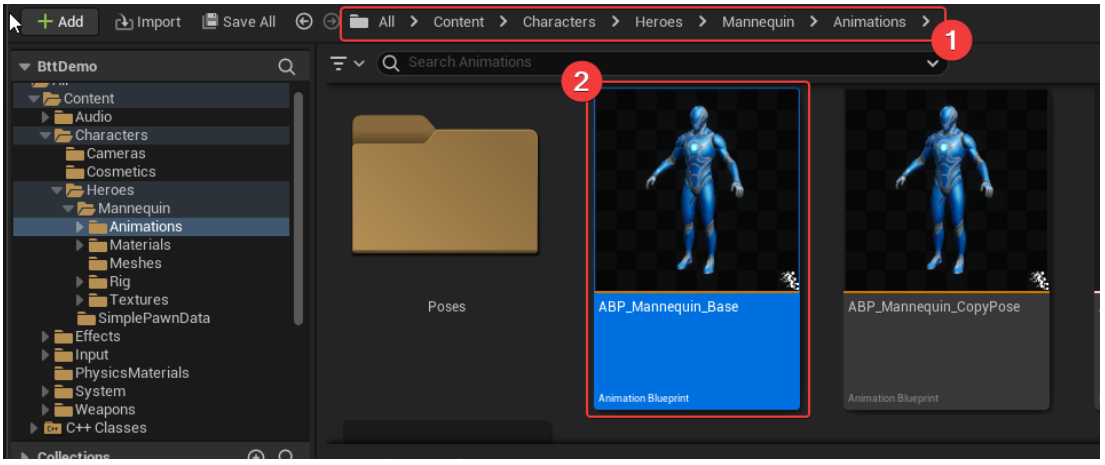
- [ ] HakAnimInstance:



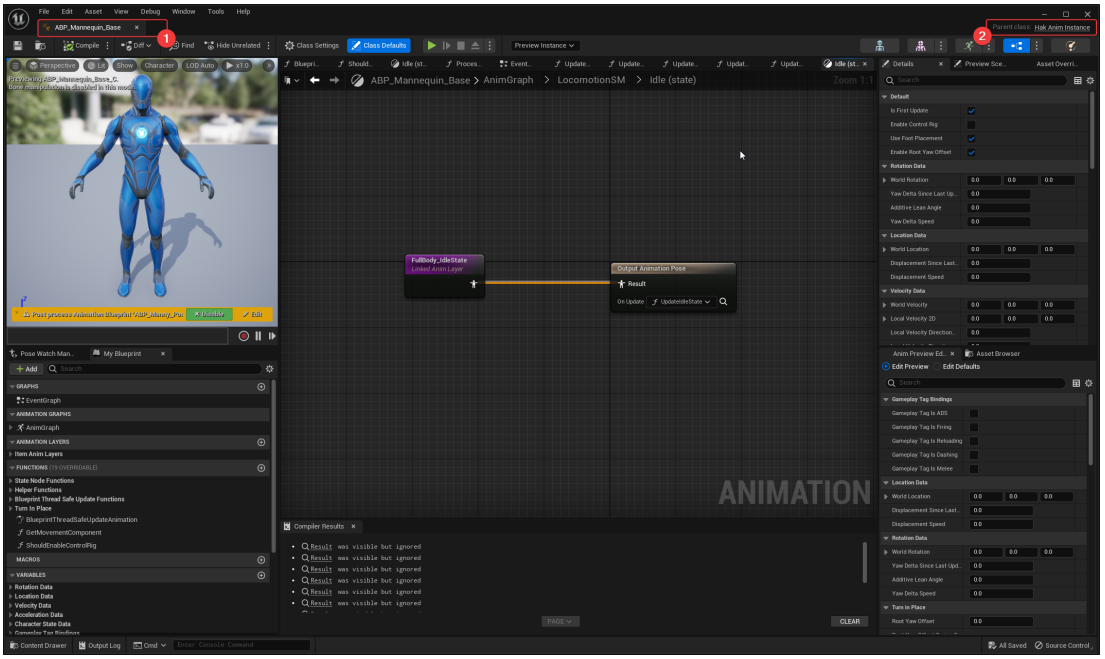


- [ ] LyraAnimInstance를 HakAnimInstance로 Class Redirect를 해주자:
  - [ ] DefaultEngine.ini 파일을 열어주자:
    - 당연히 저의 경우, Hak 프로젝트의 DefaultEngine.ini일 것이다. 여러분들도 여러분에 알맞는 프로젝트의 DefaultEngine.ini를 수정하자:

☐ 이제 간단한 AnimBP를 하나 Migrate 해보자:



☐ ABP_Mannequin_Base가 잘 열려, HakAnimInstance로 Redirect된 것을 확인할 수 있다:



☐ 여러분들이 CoreRedirects가 어떻게 일어나는지 궁금하면, 아래의 코드를 디버깅하면 된다:

```
bool FCoreRedirects::RedirectNameAndValues(ECoreRedirectFlags Type, const FCoreRedirectObjectName& OldObjectName,
    FCoreRedirectObjectName& NewObjectName, const FCoreRedirect** FoundValueRedirect, ECoreRedirectMatchFlags MatchFlags)
{
    NewObjectName = OldObjectName;
    TArray<const FCoreRedirect*> FoundRedirects;

    if (GetMatchingRedirects(Type, OldObjectName, FoundRedirects, MatchFlags))
    {
        // Sort them based on match
        FoundRedirects.Sort([&OldObjectName](const FCoreRedirect& A, const FCoreRedirect& B) { return A.OldName.MatchScore(OldObjectName

        // Apply in order
        for (int32 i = 0; i < FoundRedirects.Num(); i++)
        {
            const FCoreRedirect* Redirect = FoundRedirects[i];

            if (!Redirect)
            {
                continue;
            }

            // Only apply if name match is still valid, if it already renamed part of it it may not apply any more. Don't want to check
            if (Redirect->Matches(NewObjectName, MatchFlags))
            {
                if (FoundValueRedirect && (Redirect->HasValueChanges() || Redirect->OverrideClassName.IsValid()))
                {
                    if (*FoundValueRedirect)
                    {
                        if ((*FoundValueRedirect)->ValueChanges.OrderIndependentCompareEqual(Redirect->ValueChanges) == false)
                        {
                            UE_LOG(LogCoreRedirects, Error, TEXT("RedirectNameAndValues(%s) found multiple conflicting value redirects,
                        }
                    }
                    else
                    {
                        // Set value redirects for processing outside
```

- 허나 여길 디버깅하기 위해, 이름을 넣으면 Full-Compile이므로 생략하도록 하겠다

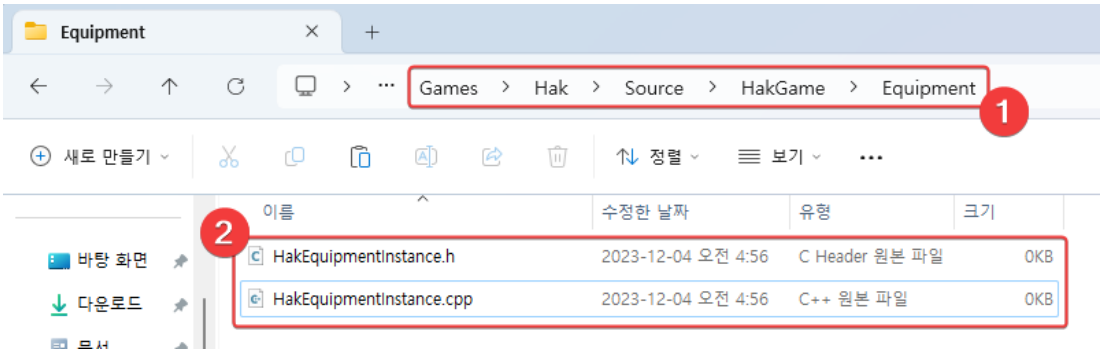
원리는 StaticConstructObject하는 과정에서 UClass 이름을 ResolveName하는 과정이 있는데 이 과정에서 Redirect를 진행하여, LyraAnimInstance는 HakAnimInstance로 변환하는 과정이 일어난다.

WeaponInstance

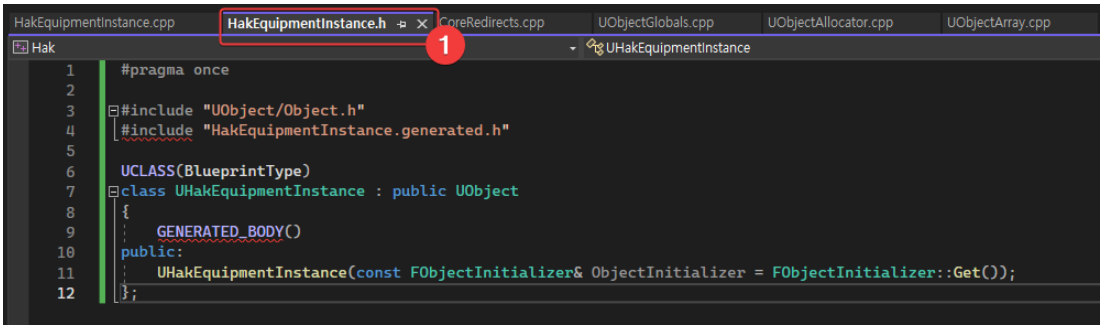
▼ **펼치기**

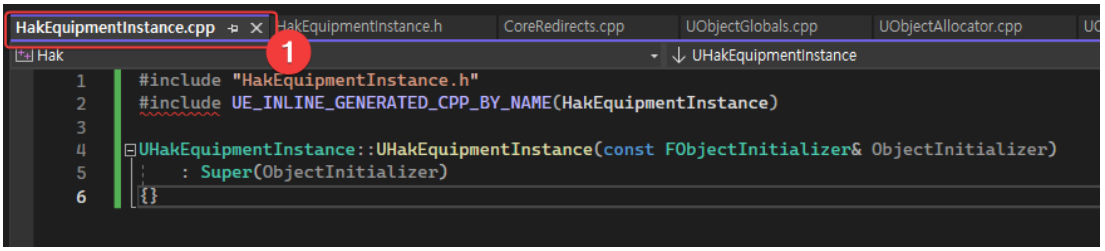
- Lyra는 Animation이 사실상 Equipment 혹은 Weapon에 정의한다:
  - 이를 **Animation 재생에 Equipment 또는 Weapon이 필요하다는 것을 의미**한다.

☐ HakEquipmentInstance.h/.cpp 생성:

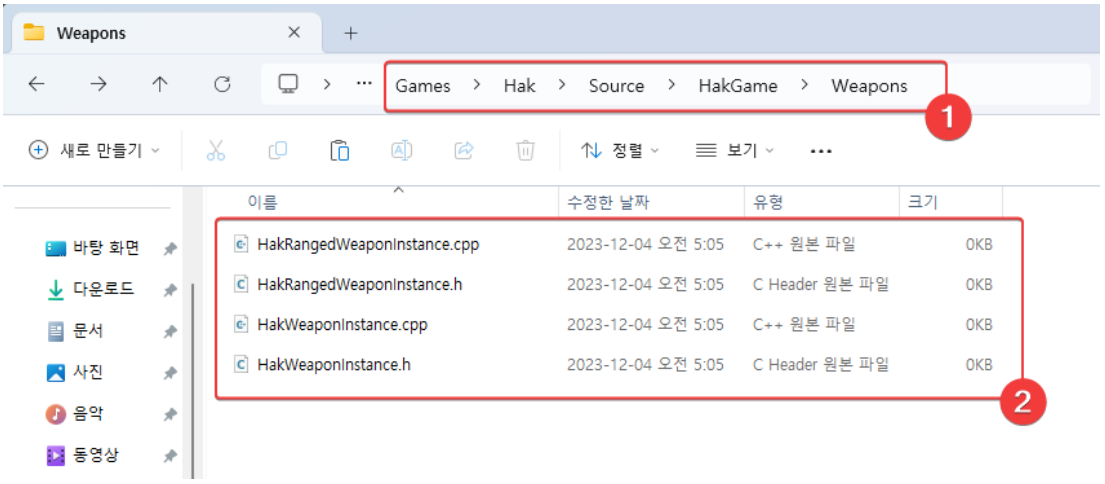


☐ HakEquipmentInstance 구현:

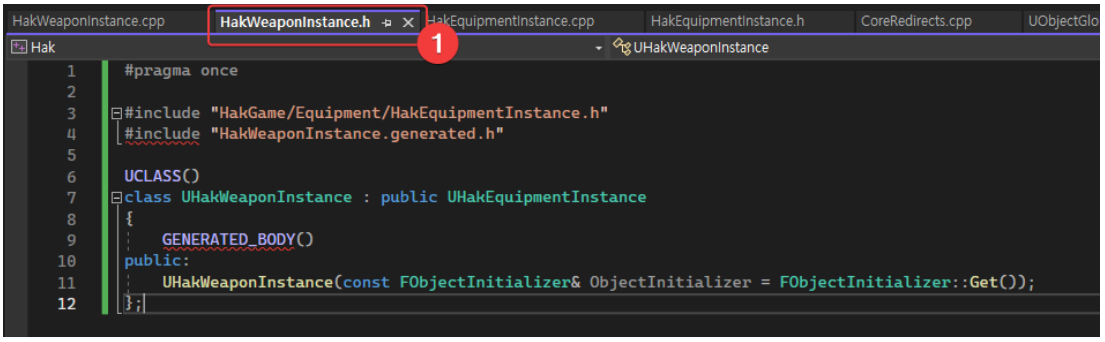


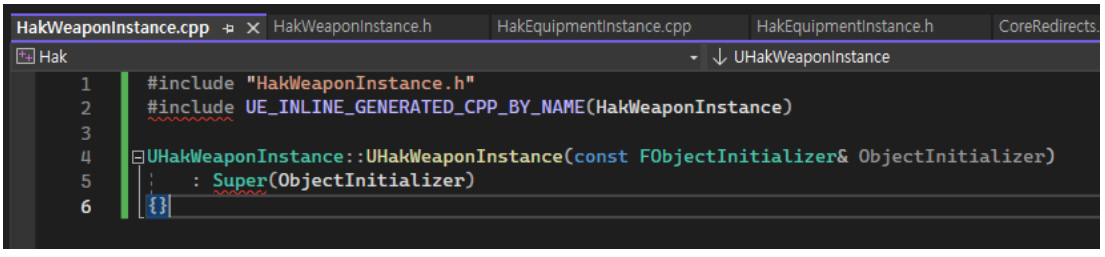


☐ HakWeaponInstance.h/.cpp, HakRangedWeaponInstance.h/.cpp 생성:



☐ HakWeaponInstance 구현:





☐ HakRangedWeaponInstance 구현:

```h
#pragma once

#include "HakWeaponInstance.h"
#include "HakRangedWeaponInstance.generated.h"

UCLASS()
class UHakRangedWeaponInstance : public UHakWeaponInstance
{
    GENERATED_BODY()
public:
    UHakRangedWeaponInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
};
```

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

UHakRangedWeaponInstance::UHakRangedWeaponInstance(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{}
```

☐ HakWeaponInstance 멤버 변수 선언:

```h
#pragma once

#include "HakGame/Equipment/HakEquipmentInstance.h"
#include "HakWeaponInstance.generated.h"

UCLASS()
class UHakWeaponInstance : public UHakEquipmentInstance
{
    GENERATED_BODY()
public:
    UHakWeaponInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** Weapon에 Equip/Unequip에 대한 Animation Set 정보를 들고 있다 */
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Animation)
    FHakAnimLayerSelectionSet EquippedAnimSet;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Animation)
    FHakAnimLayerSelectionSet UnequippedAnimSet;
};
```

☐ FHakAnimLayerSelectionSet:

```h
USTRUCT(BlueprintType)
struct FHakAnimLayerSelectionSet
{
    GENERATED_BODY()

    /** 앞서 보았던 HakAnimBodyStyleSelection의 MeshRule과 같이 AnimInstance의 Rule을 가진 LayerRules로 생각하면 됨 */
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TArray<FHakAnimLayerSelectionEntry> LayerRules;

    /** 디폴트 Layer */
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TSubclassOf<UAnimInstance> DefaultLayer;
};
```

☐ FHakAnimLayerSelectionEntry:

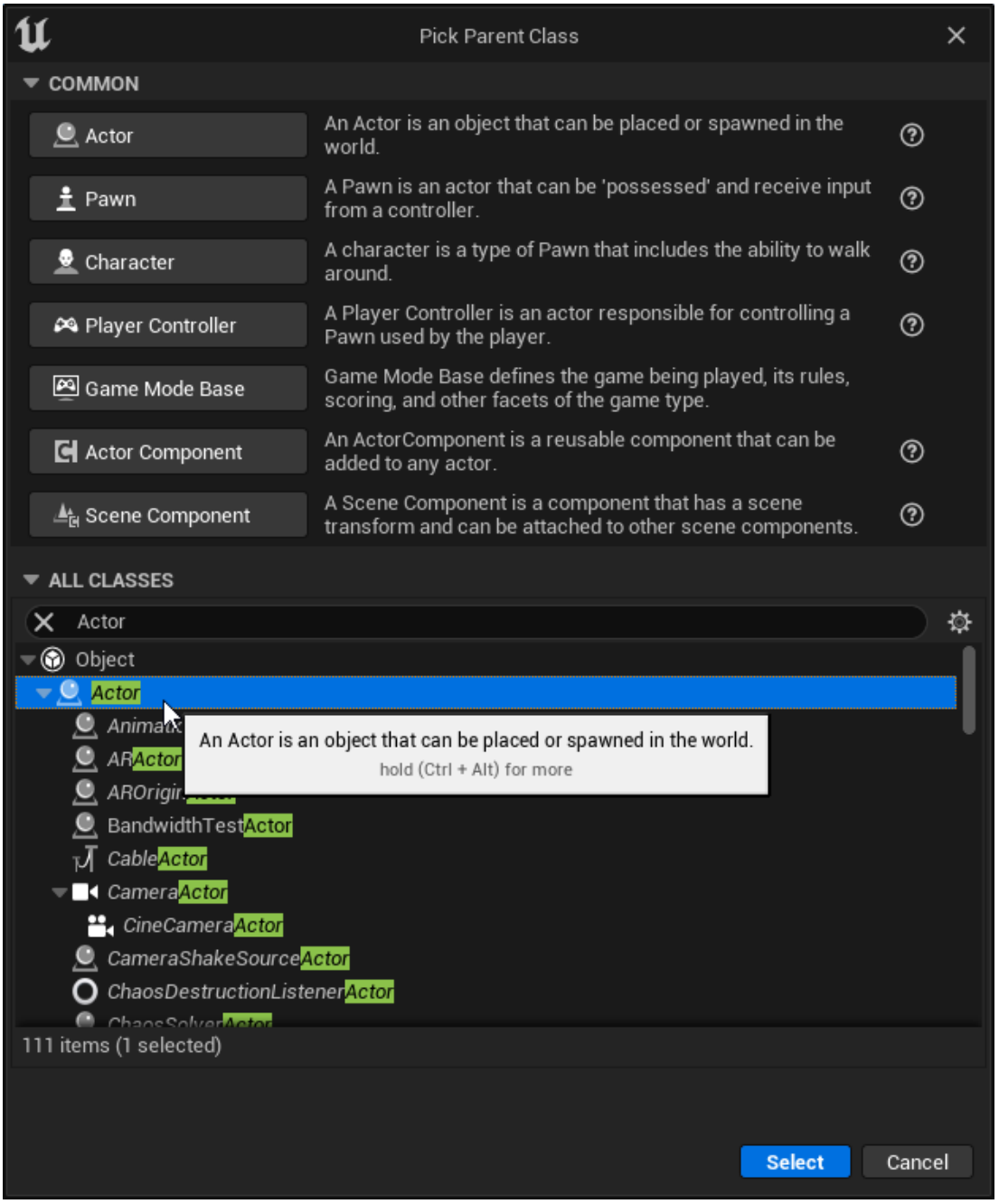
```cpp
USTRUCT(BlueprintType)
struct FHakAnimLayerSelectionEntry
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TSubclassOf<UAnimInstance> Layer;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FGameplayTagContainer RequiredTags;
};
```

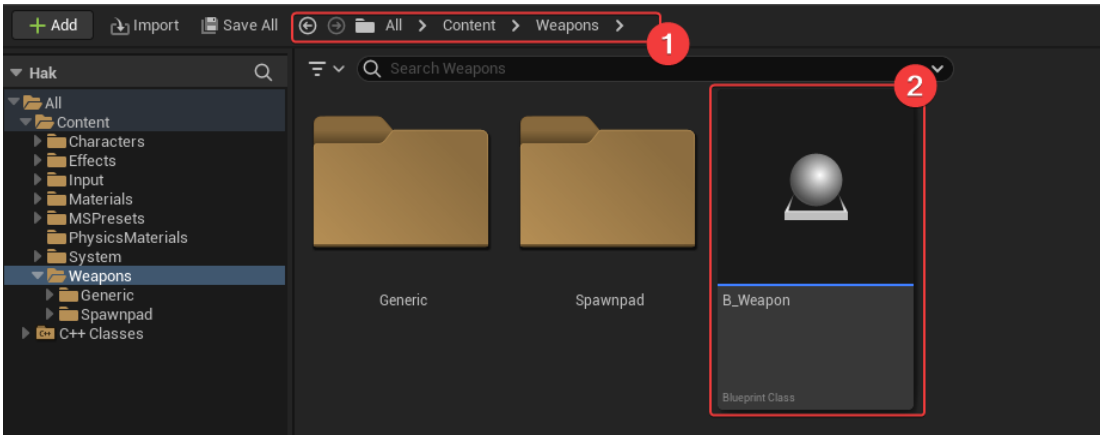
☐ 헤더 추가:

```cpp
#pragma once

#include "HakGame/Equipment/HakEquipmentInstance.h"
#include "HakGame/Cosmetics/HakCosmeticAnimationTypes.h"
#include "HakWeaponInstance.generated.h"

UCLASS()
class UHakWeaponInstance : public UHakEquipmentInstance
{
    GENERATED_BODY()
public:
    UHakWeaponInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** Weapon에 Equip/Unequip에 대한 Animation Set 정보를 들고 있다 */
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Animation)
    FHakAnimLayerSelectionSet EquippedAnimSet;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Animation)
    FHakAnimLayerSelectionSet UnequippedAnimSet;
};
```

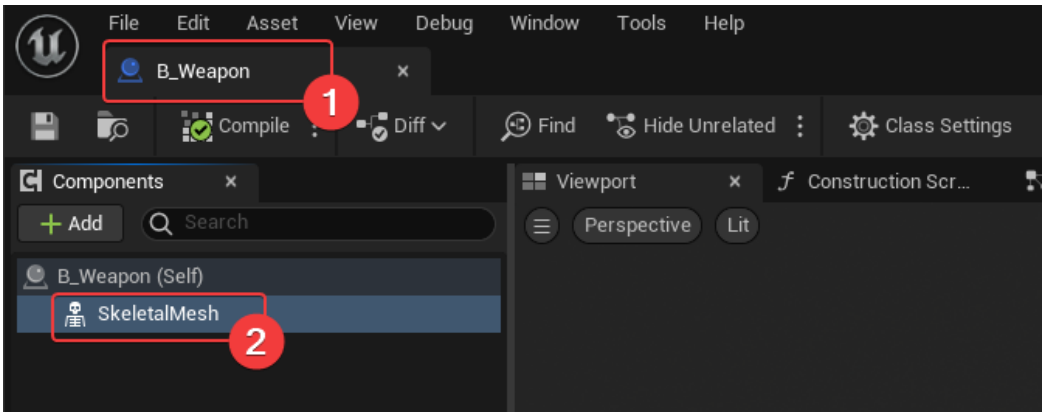
1

☐ B_Weapon 생성:

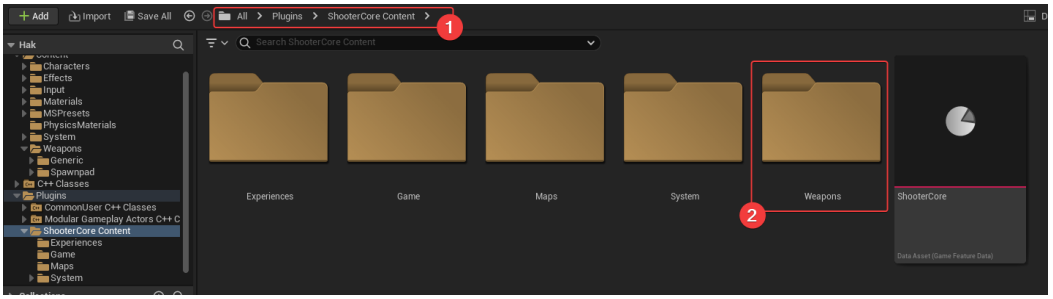

Pick Parent Class
COMMON
Actor
An Actor is an object that can be placed or spawned in the world.
Pawn
A Pawn is an actor that can be 'possessed' and receive input from a controller.
Character
A character is a type of Pawn that includes the ability to walk around.
Player Controller
A Player Controller is an actor responsible for controlling a Pawn used by the player.
Game Mode Base
Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.
Actor Component
An ActorComponent is a reusable component that can be added to any actor.
Scene Component
A Scene Component is a component that has a scene transform and can be attached to other scene components.
ALL CLASSES
Actor
Object
Actor
An Actor is an object that can be placed or spawned in the world.
hold (Ctrl + Alt) for more
BandwidthTestActor
CableActor
CameraActor
CineCameraActor
CameraShakeSourceActor
ChaosDestructionListenerActor
111 items (1 selected)
Select
Cancel


Add
Import
Save All
All
Content
Weapons
1
Hak
Search Weapons
2
All
Content
Characters
Effects
Input
Materials
MSPresets
PhysicsMaterials
System
Weapons
Generic
Spawnpad
C++ Classes
Generic
Spawnpad
B_Weapon
Blueprint Class

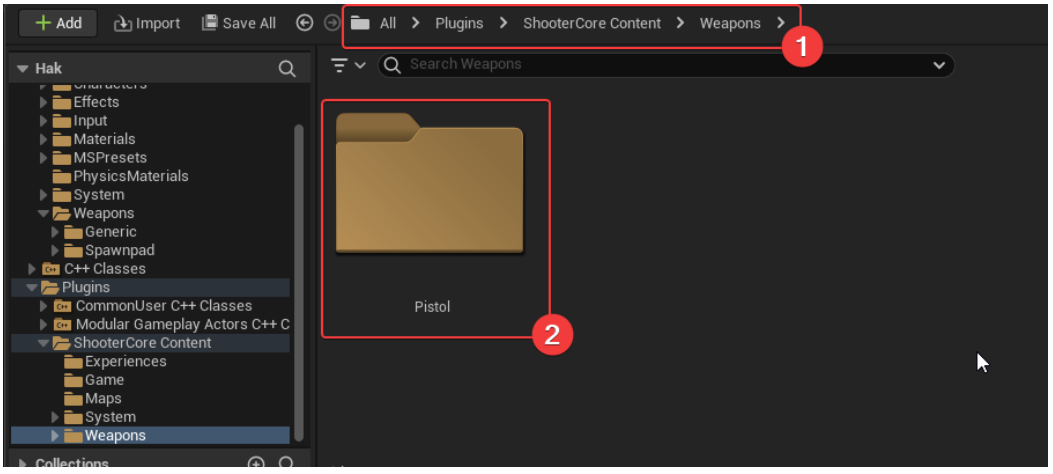
- SkeletalMesh 생성



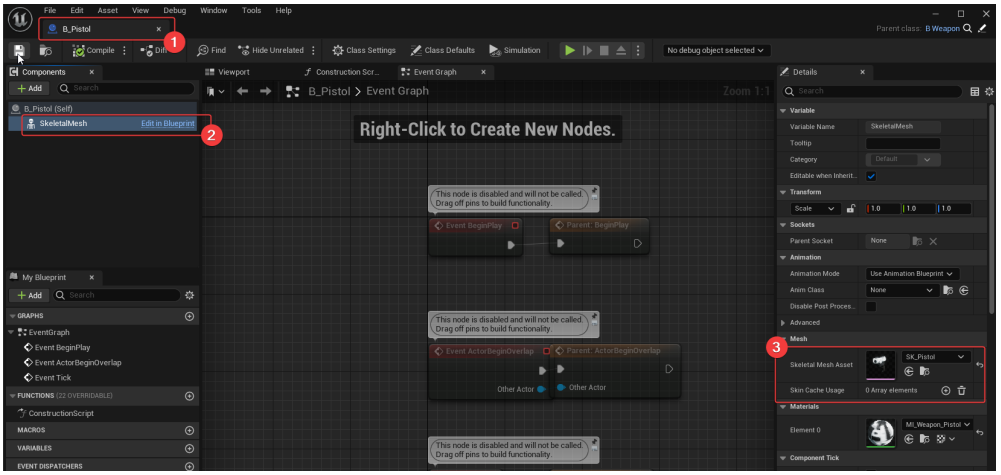
- [ ] B_Pistol 생성:
  - [ ] ShooterCore/Weapons 생성:



  - [ ] Pistol 폴더 생성:



  - [ ] B_Weapon을 상속 받는 B_Pistol 생성:
    - [ ] SK_Pistol Migrate:
    - [ ] SK_Pistol을 B_Pistol의 SkeletalMesh에 설정:

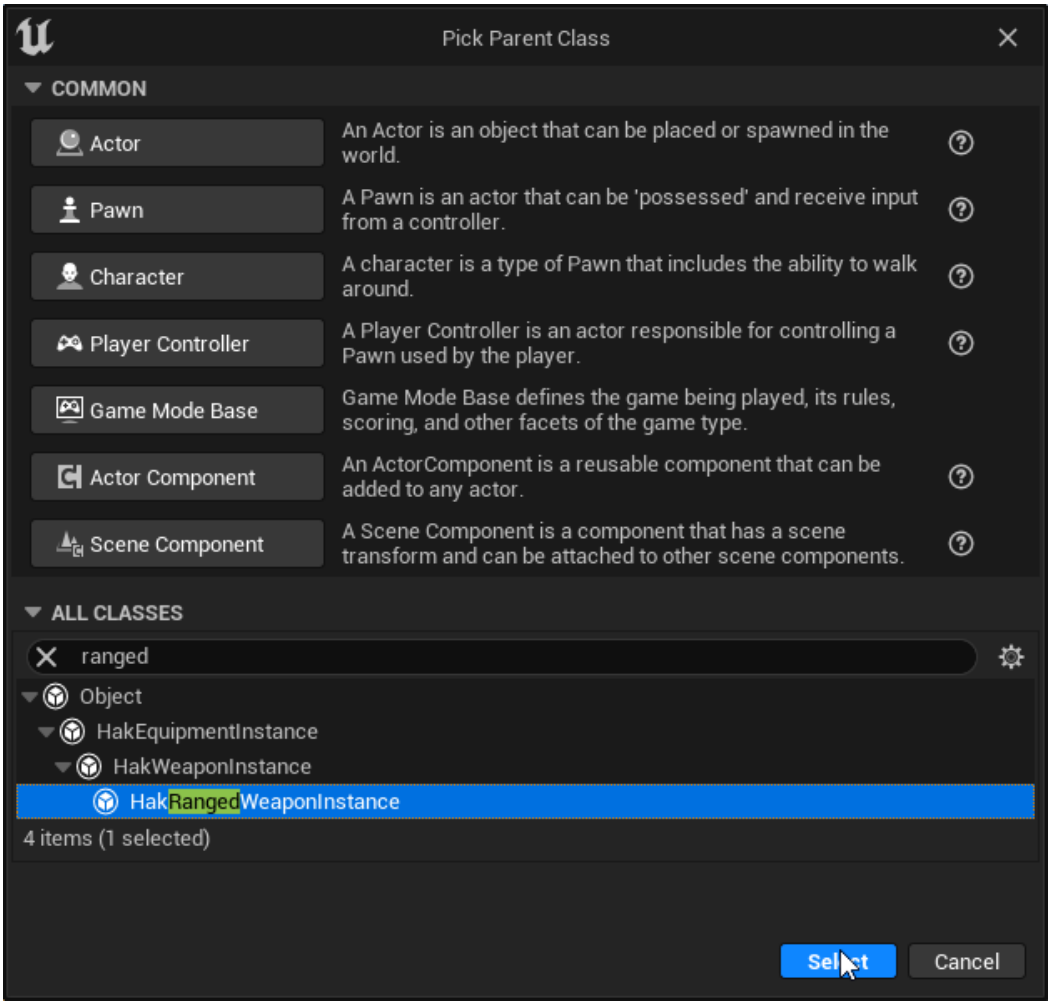


☐ B_WeaponInstance_Base 생성:

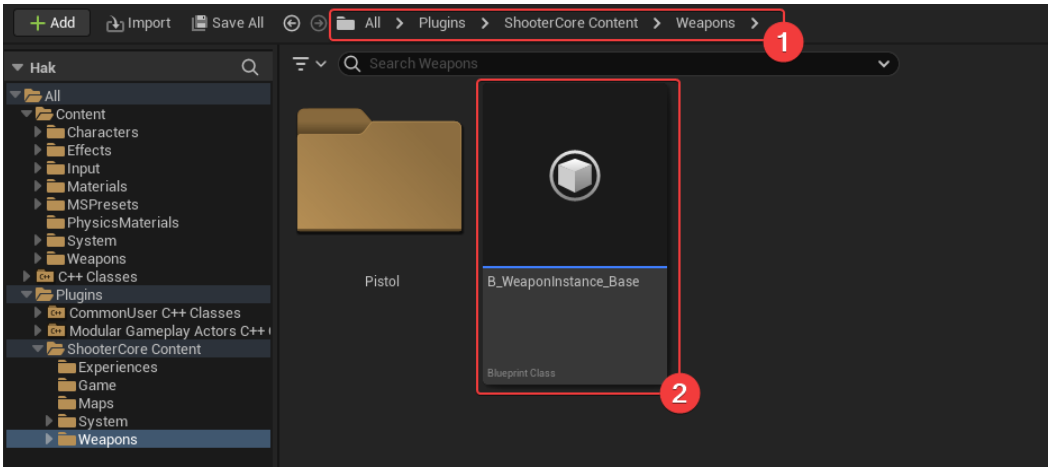
- HakEquipmentInstance에 PROPERTY 속성 추가해주자:

```
#pragma once

#include "UObject/Object.h"
#include "HakEquipmentInstance.generated.h"

UCLASS(BlueprintType, Blueprintable)
class UHakEquipmentInstance : public UObject
{
    GENERATED_BODY()
public:
    UHakEquipmentInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
};
```

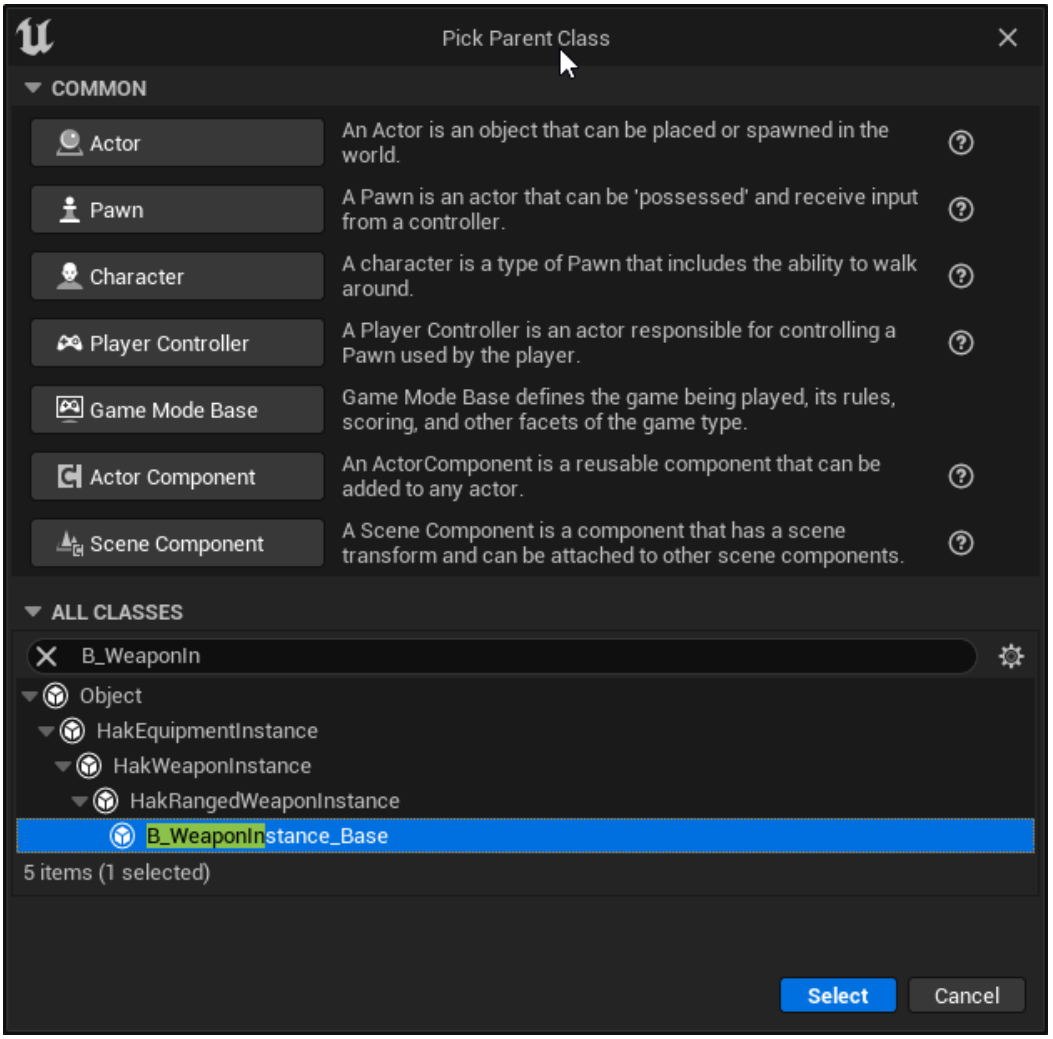
☐ HakRangedWeaponInstance를 상속받자:

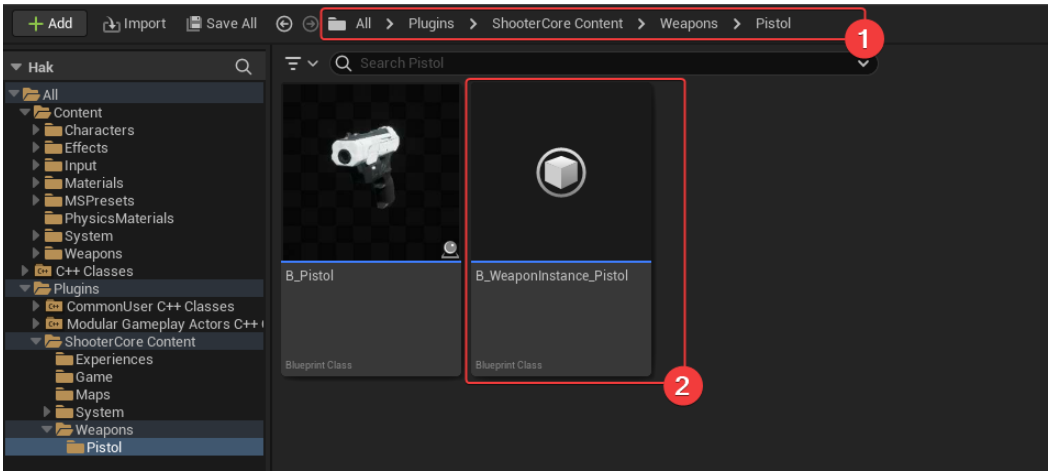


☐ B_WeaponInstance_Base 생성:

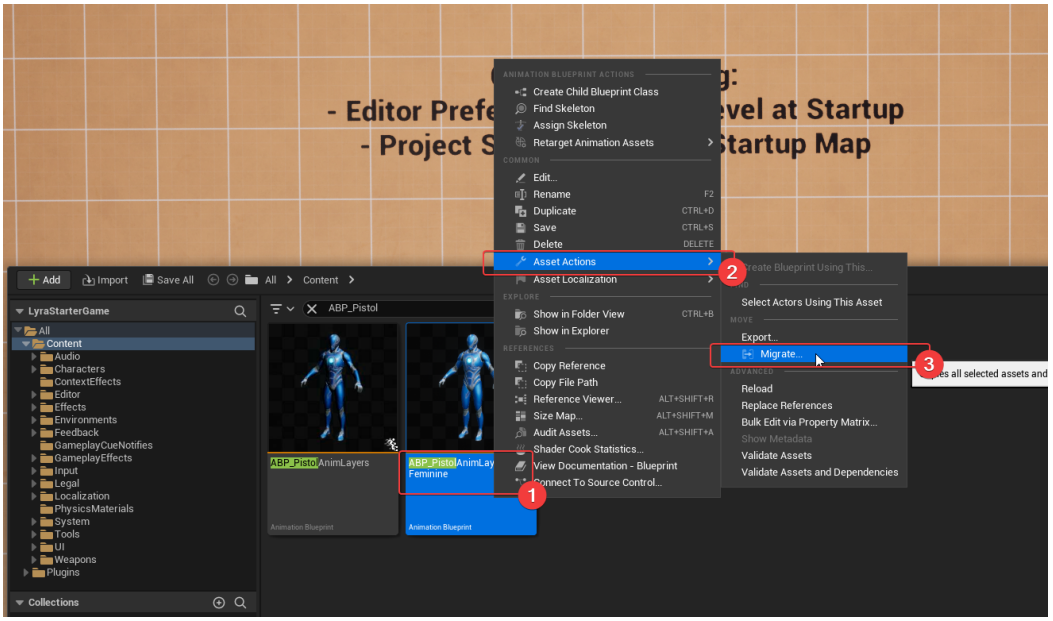


☐ B_WeaponInstance_Pistol 생성:

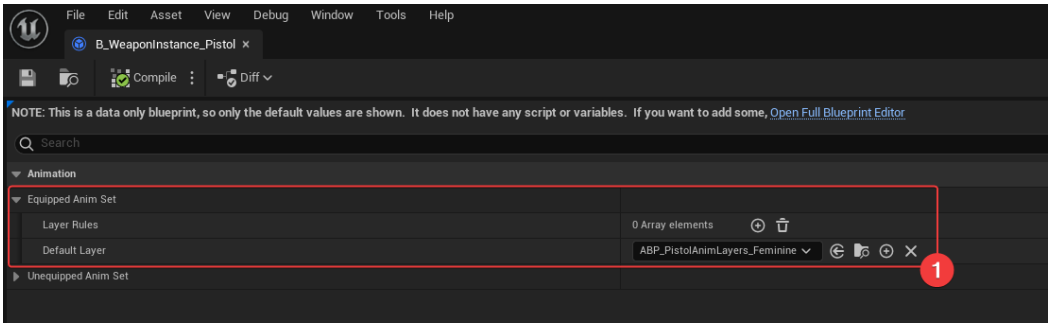




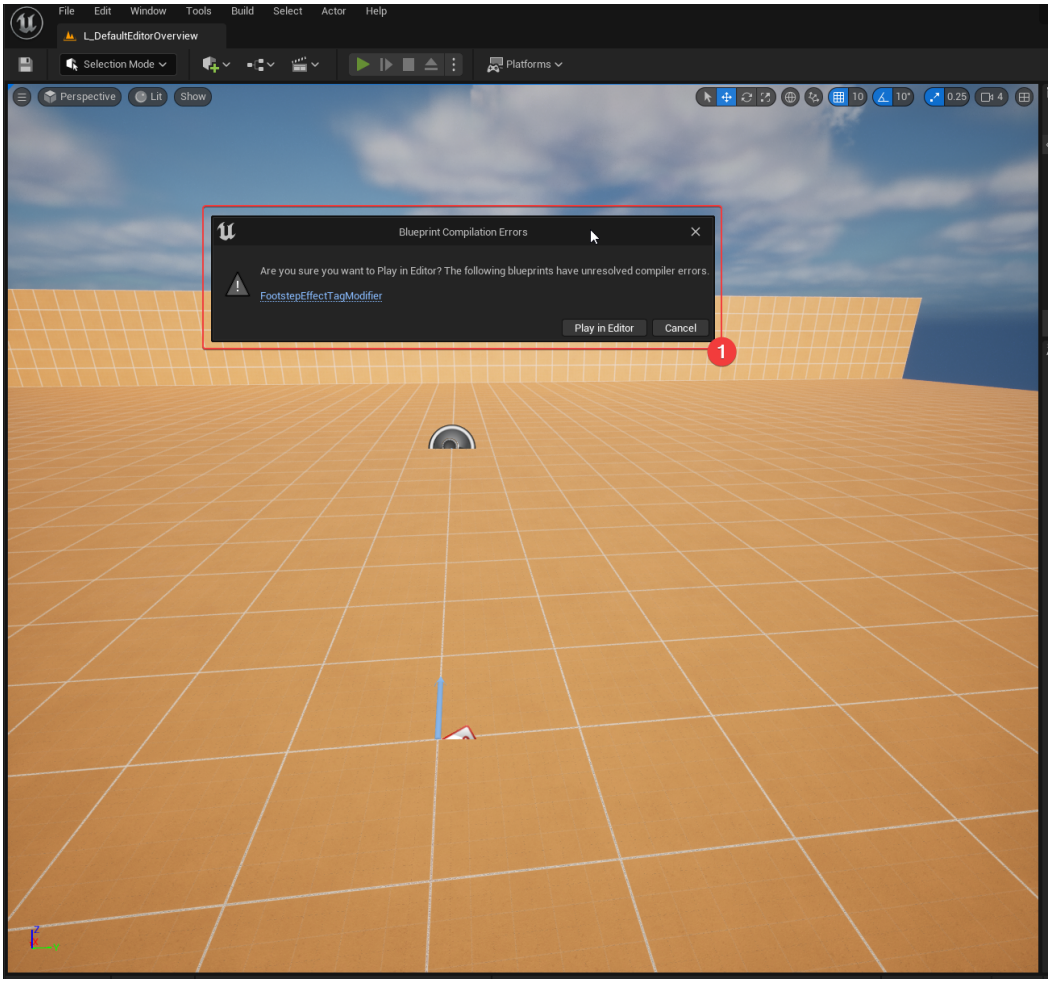
☐ ABP_PistolAnimLayers_Feminine Migrate 해주자:



☐ EquippedAnimSet에 설정해주자:

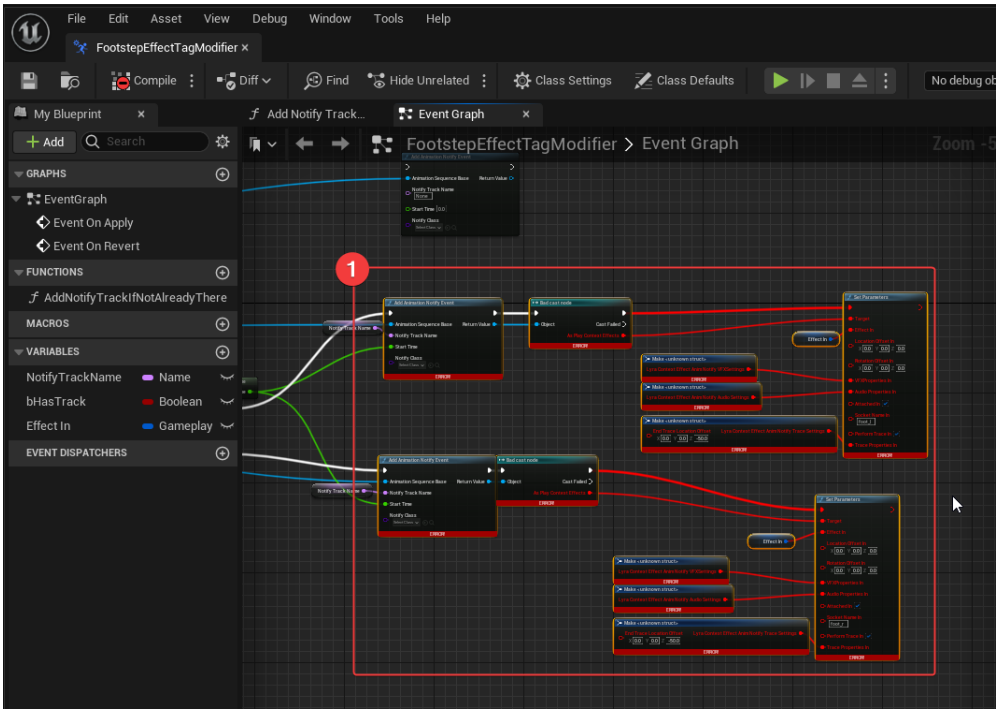


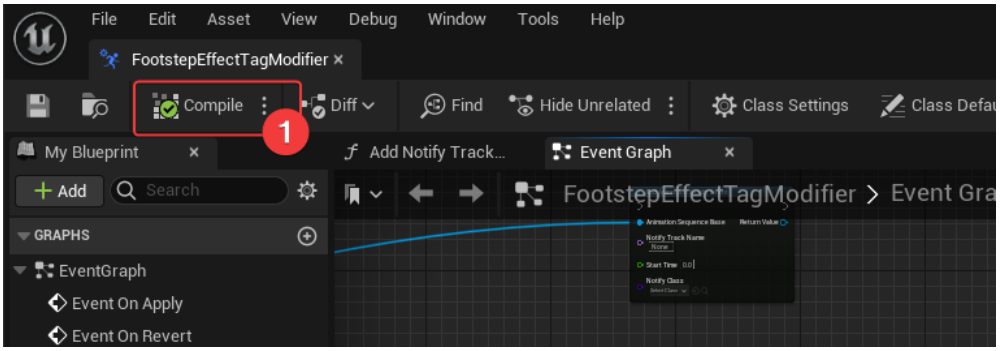
☐ PIE 재생이 안된다:



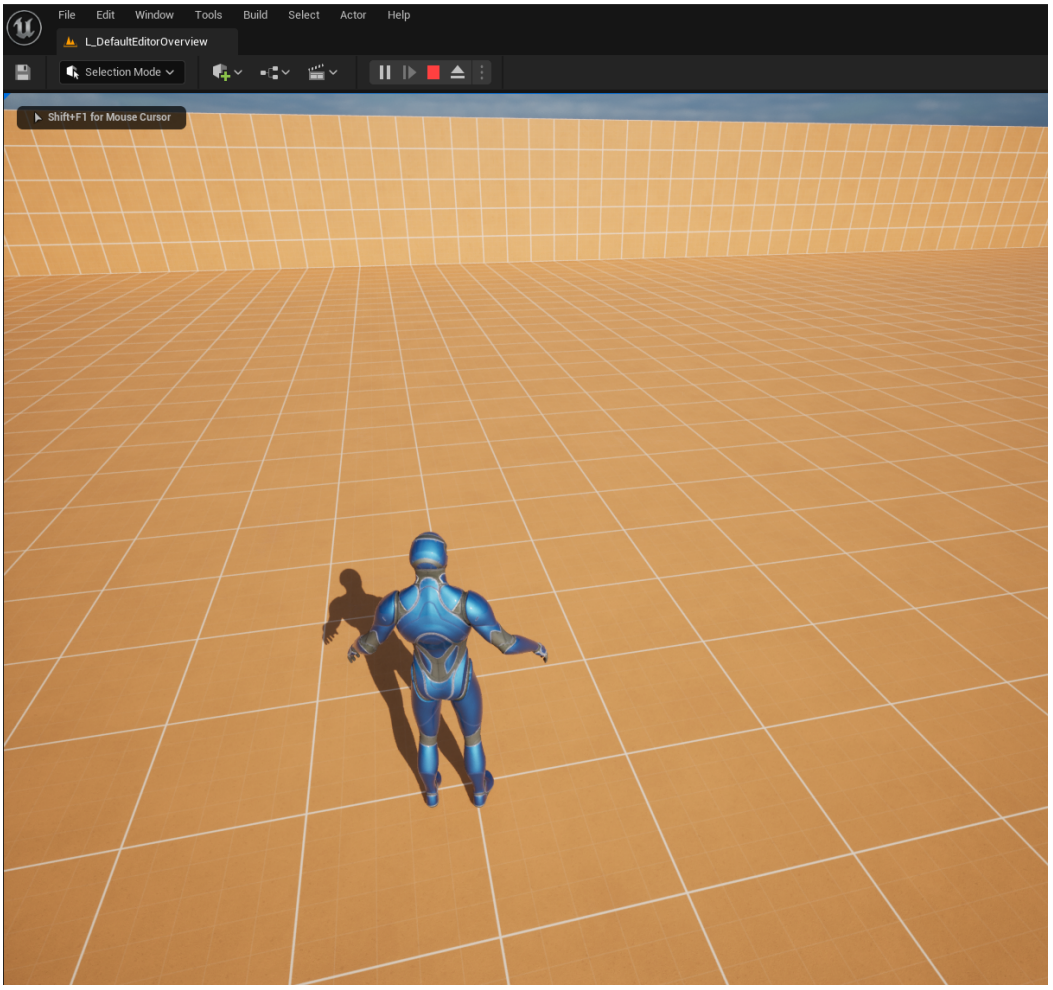
☐ 다음과 같이 FootstepEffectTagModifier를 눌러주자:

☐ 일단 그냥 지우자...


File
Edit
Asset
View
Debug
Window
Tools
Help
FootstepEffectTagModifier
Compile
Diff
Find
Hide Unrelated
Class Settings
Class Defaults
My Blueprint
Add
Search
GRAPHS
EventGraph
Event On Apply
Event On Revert
FUNCTIONS
AddNotifyTrackIfNotAlreadyThere
MACROS
VARIABLES
NotifyTrackName
Name
bHasTrack
Boolean
Effect In
Gameplay
EVENT DISPATCHERS
Add Notify Track...
Event Graph
FootstepEffectTagModifier > Event Graph
1


File
Edit
Asset
View
Debug
Window
Tools
Help
FootstepEffectTagModifier
Compile
1
Diff
Find
Hide Unrelated
Class Settings
My Blueprint
Add
Search
GRAPHS
EventGraph
Event On Apply
Event On Revert
Add Notify Track...
Event Graph
FootstepEffectTagModifier > Event Gra

☐ 아직 아무 변화가 없다...



- 당연히 그러하다:
  - 우리는 아직 AnimInstance를 활성화하지 않았기 때문이다
  - 이는 다음 시간에 Equipment, Inventory를 통해 Weapon을 장착하며, Animation을 활성화해보자!