



10주차 (2023.12.04)

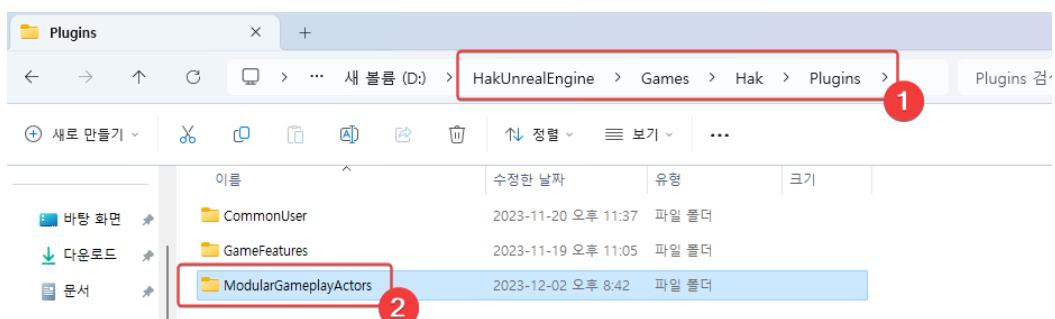
ModularGameplayActors:

▼ 펼치기

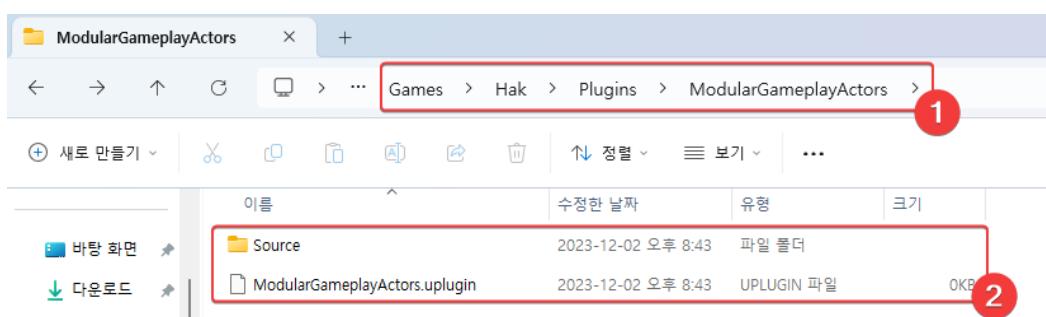
- 우리는 현재 기본 GameFramework에서 제공하는 Actor를 사용하였다:
 - GameFeatureAction을 활용한 동적으로 GameFeature Plugin에 정의된 Component를 Actor에 붙이려면, UGameFeatureFrameworkManager에 Receiver로 등록해야 한다
 - Lyra에서는 이를 용이하게 하기 위해, 따로 ModularGameplayActors의 Plugin을 정의하여 ModularActor와 같은 형태의 클래스로 만들어 Interface를 래핑하였다.

ModularGameplayActors.plugin 생성:

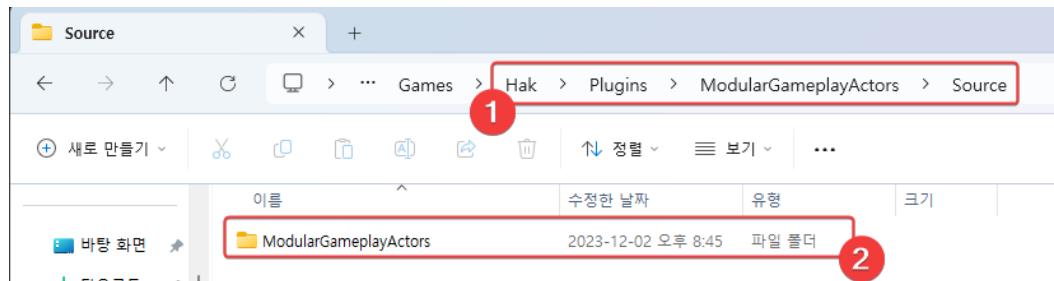
ModularGameplayActors 폴더 생성:



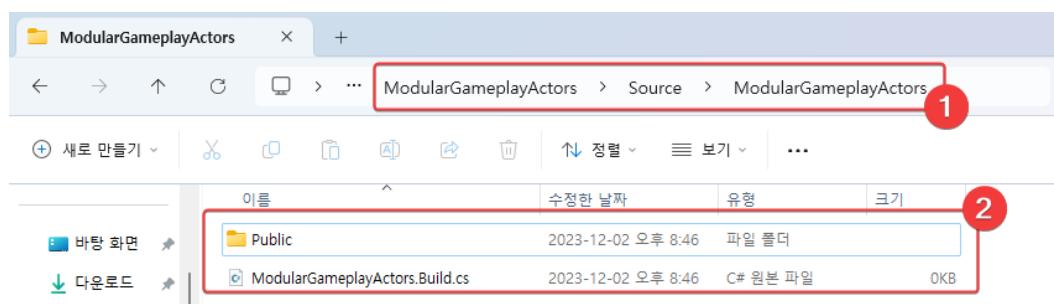
Source 폴더 생성과 ModularGameplayActors.upplugin 생성:



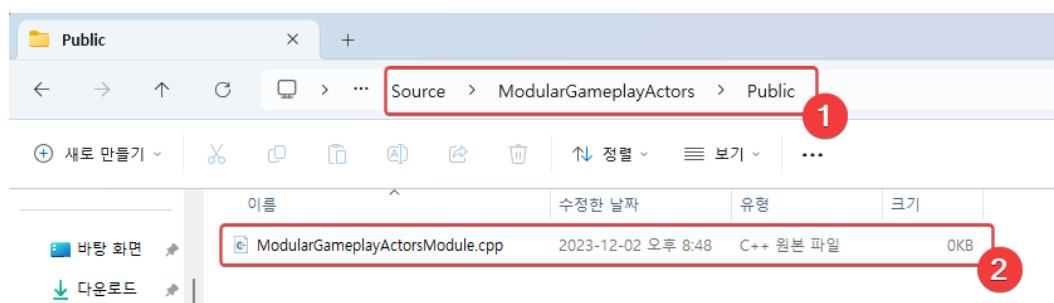
- ModularGameplayActors 생성 → Modular 경로 생성



- ModularGameplayActors의 기본 Module에 필요한 파일 및 폴더 추가:



- 기본 모듈 파일 생성: `ModularGameplayActorsModule.cpp`



- 아래와 같이 각각 파일에 대한 내용 추가:

- `ModuleGameplayActors.uplugin`:

```

1  {
2      "FileVersion": 3,
3      "Version": 1,
4      "VersionName": "1.0",
5      "FriendlyName": "Modular Gameplay Actors",
6      "Description": "Base classes designed to be used with the Modular Gameplay plugin.",
7      "Category": "Gameplay",
8      "CreatedBy": "Epic Games, Inc.",
9      "CreatedbyURL": "http://epicgames.com",
10     "DocsURL": "",
11     "MarketplaceURL": "",
12     "SupportURL": "",
13     "EnabledbyDefault": true,
14     "CanContainContent": false,
15     "IsBetaVersion": false,
16     "Installed": false,
17     "Modules": [
18         {
19             "Name": "ModularGameplayActors",
20             "Type": "Runtime",
21             "LoadingPhase": "Default"
22         }
23     ],
24     "Plugins": [
25         {
26             "Name": "ModularGameplay",
27             "Enabled": true
28         }
29     ]
30 }
31

```

- ModularGameplayActors.Build.cs:

```

1 // Copyright Epic Games, Inc. All Rights Reserved.
2
3 using UnrealBuildTool;
4 using System.IO; // for Path
5
6 public class ModularGameplayActors : ModuleRules
7 {
8     public ModularGameplayActors(ReadOnlyTargetRules Target) : base(Target)
9     {
10         PCHUsage = ModuleRules.PCHUsageMode.UseExplicitOrSharedPCHs;
11
12         PublicIncludePaths.AddRange(
13             new string[] {
14                 // ... add public include paths required here ...
15             }
16         );
17
18         PrivateIncludePaths.AddRange(
19             new string[] {
20                 // ... add other private include paths required here ...
21             }
22         );
23
24         PublicDependencyModuleNames.AddRange(
25             new string[]
26             {
27                 "Core",
28                 "CoreObject",
29                 "Engine",
30                 "ModularGameplay", 2 우리는 ModularGameplay 모듈이 필요하다!
31                 // ... add other public dependencies that you statically link with here ...
32             }
33         );
34
35         PrivateDependencyModuleNames.AddRange(
36             new string[]
37             {
38                 // ... add private dependencies that you statically link with here ...
39             }
40         );
41
42         DynamicallyLoadedModuleNames.AddRange(
43             new string[]
44             {
45                 // ... add any modules that your module loads dynamically here ...
46             }
47         );
48
49     }
50 }
51
52 }
53

```

- ModuleGameplayActorsModule.cpp:

```

ModularGameplayActors.uplugin U ModularGameplayActors.Build.cs U ModularGameplayActorsModule.cpp U
Games > Hak > Plugins > ModularGameplayActors > Source > ModularGameplayActors > Public > ModularGameplayActorsModule.cpp
1 // Copyright Epic Games, Inc. All Rights Reserved.
2
3 #include "Modules/ModuleManager.h"
4
5 IMPLEMENT_MODULE(FDefaultModuleImpl, ModularGameplayActors);
6

```

GenerateProjectFiles.bat를 실행시켜, 해당 Plugin을 생성시키자.

이제 ModularGameplayActors에서 Actor들 중에, ACharacter와 APlayerCharacter를 Modular Actor 타입으로 생성해주자:

우선, ModularCharacter.h/.cpp와 ModularPlayerController.h/.cpp를 생성해 주자:

ModularCharacter.h/.cpp 구현:

```

ModularCharacter.cpp ModularCharacter.h AnimGraphNod...lacement.cpp LyraHeroComponent.h LyraStarterGame.uproject
Hak
1 #pragma once
2
3 #include "GameFramework/Character.h"
4 #include "ModularCharacter.generated.h"
5
6 UCLASS()
7 class MODULARGAMEPLAYACTORS_API AModularCharacter : public ACharacter
8 {
9     GENERATED_BODY()
10    public:
11 };

```

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

```

ModularCharacter.cpp ModularCharacter.h AnimGraphNod...lacement.cpp LyraHeroComponent.h
Hak
1 #include "ModularCharacter.h"
2 #include UE_INLINE_GENERATED_CPP_BY_NAME(ModularCharacter)
3

```

GameFrameworkComponentManager를 활용하여, Character를 모듈화해 주자:

- ModularCharacter.h:

```

1 #pragma once
2
3 #include "Engine/EngineTypes.h"
4 #include "GameFramework/Character.h"
5 #include "ModularCharacter.generated.h"
6
7 UCLASS()
8 class MODULARGAMEPLAYACTORS_API AModularCharacter : public ACharacter
9 {
10     GENERATED_BODY()
11 public:
12     /**
13      * character's interface
14      */
15     virtual void PreInitializeComponents() override;
16     virtual void BeginPlay() override;
17     virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) override;
18 };
19

```

Annotations:

- Annotation 1: A red circle with the number 1 is placed over the line "#include "ModularCharacter.generated.h"".
- Annotation 2: A red circle with the number 2 is placed over the class definition "class MODULARGAMEPLAYACTORS_API AModularCharacter : public ACharacter".
- Annotation 3: A red circle with the number 3 is placed over the closing brace of the class definition "};".

- ModularCharacter.cpp:

```

ModularPlayerController.h ModularPlayerController.cpp ModularCharacter.cpp ModularCharacter.h AnimGraphNodePlacement.cpp LyraHeroComponent.h LyraStarterGame.uasset
1 #include "ModularCharacter.h"
2 #include "Components/GameFrameworkComponentManager.h"
3 #include UE_INLINE_GENERATED_CPP_BY_NAME(ModularCharacter)
4
5 void AModularCharacter::PreInitializeComponents()
6 {
7     Super::PreInitializeComponents();
8     UGameFrameworkComponentManager::AddGameFrameworkComponentReceiver(this);
9 }
10
11 void AModularCharacter::BeginPlay()
12 {
13     UGameFrameworkComponentManager::SendGameFrameworkComponentExtensionEvent(this, UGameFrameworkComponentManager::NAME_GameActorReady);
14     Super::BeginPlay();
15 }
16
17 void AModularCharacter::EndPlay(const EEndPlayReason::Type EndPlayReason)
18 {
19     UGameFrameworkComponentManager::RemoveGameFrameworkComponentReceiver(this);
20     Super::EndPlay(EndPlayReason);
21 }

```

Annotations:

- Annotation 1: A red circle with the number 1 is placed over the line "#include "ModularCharacter.h"".
- Annotation 2: A red circle with the number 2 is placed over the class definition "class AModularCharacter".
- Annotation 3: A red circle with the number 3 is placed over the closing brace of the class definition "};".

□ ModularPlayerController.h/.cpp 구현:

앞서, ModularCharacter랑 똑같이 바로 Interface까지 같이 구현해주자:

- ModularGameplayController.h:

```

ModularPlayerController.h ModularPlayerController.cpp ModularCharacter.cpp ModularCharacter.h AnimGraphNodePlacement.cpp
1 #pragma once
2
3 #include "Engine/EngineTypes.h"
4 #include "GameFramework/PlayerController.h"
5 #include "ModularPlayerController.generated.h"
6
7 UCLASS()
8 class MODULARGAMEPLAYACTORS_API AModularPlayerController : public APlayerController
9 {
10     GENERATED_BODY()
11 public:
12     virtual void PreInitializeComponents() override;
13     virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) override;
14 };

```

Annotations:

- Annotation 1: A red circle with the number 1 is placed over the line "#include "ModularPlayerController.generated.h"".

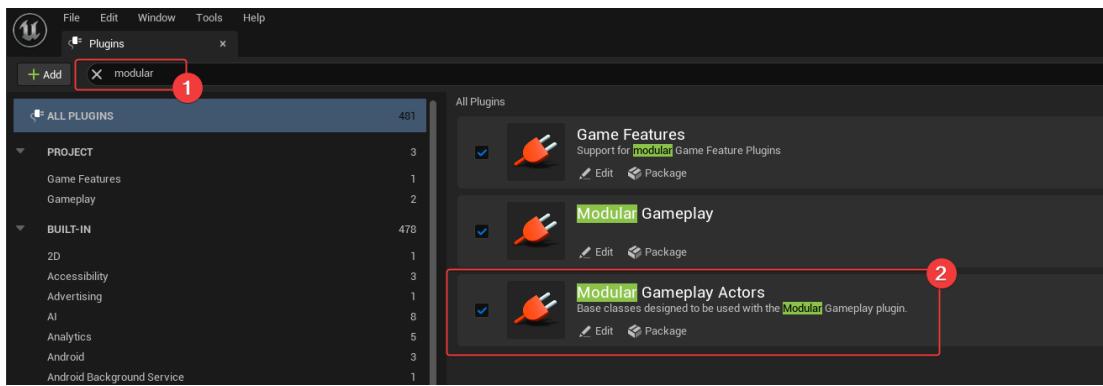
- ModularGameplayController.cpp:

```

ModularPlayerController.h ModularPlayerController.cpp ModularCharacter.cpp ModularCharacter.h AnimGraphNod...
1 #include "ModularPlayerController.h"
2 #include "Components/GameFrameworkComponentManager.h"
3 #include UE_INLINE_GENERATED_CPP_BY_NAME(ModularPlayerController)
4
5 void AModularPlayerController::PreInitializeComponents()
6 {
7     Super::PreInitializeComponents();
8     UGameFrameworkComponentManager::AddGameFrameworkComponentReceiver(this);
9 }
10
11 void AModularPlayerController::EndPlay(const EEndPlayReason::Type EndPlayReason)
12 {
13     UGameFrameworkComponentManager::RemoveGameFrameworkComponentReceiver(this);
14     Super::EndPlay(EndPlayReason);
15 }

```

- ModularGameplayActors을 활성화하자:



ModularGameplayActor에 대해 상속:

▼ 펼치기

- HakGame.Build.cs에서 ModularGameplayActors를 추가하자:

```

1 // Copyright Epic Games, Inc. All Rights Reserved.
2
3 using UnrealBuildTool;
4
5 1 reference
6 public class HakGame : ModuleRules
7 {
8     0 references
9     public HakGame(ReadOnlyTargetRules Target) : base(Target)
10    {
11        PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;
12
13        PublicDependencyModuleNames.AddRange(new string[] {
14            "Core",
15            "CoreObject",
16            "Engine",
17            "InputCore",
18            // GAS
19            "GameplayTags",
20            // Game Features
21            "ModularGameplay",
22            "GameFeatures",
23            "ModularGameplayActors", 2
24            // Input
25            "InputCore",
26            "EnhancedInput",
27            // CommonUser
28            "CommonUser",
29        });
30
31        PrivateDependencyModuleNames.AddRange(new string[] { });
32
33        // Uncomment if you are using Slate UI
34        // PrivateDependencyModuleNames.AddRange(new string[] { "Slate", "SlateCore" });
35
36        // Uncomment if you are using online features
37        // PrivateDependencyModuleNames.Add("OnlineSubsystem");
38
39        // To include OnlineSubsystemSteam, add it to the plugins section in your uproject file with the Enabled attribute
40    }
}

```

HakCharacter를 ModularCharacter 상속으로 교체하자:

```

1 #pragma once
2
3 #include "ModularCharacter.h" 2
4 #include "HakCharacter.generated.h"
5
6 /** forward declaration */
7 class UHakPawnExtensionComponent;
8 class UHakCameraComponent;
9
10 UCCLASS()
11 class AHakCharacter : public AModularCharacter 3
12 {
13     GENERATED_BODY()
14     public:
15         AHakCharacter(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
16
17     /**
18     * ACharacter interfaces
19     */
20     virtual void SetupPlayerInputComponent(UInputComponent* PlayerInputComponent) final;
21
22     UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category="Hak|Character")
23     TObjectPtr<UHakPawnExtensionComponent> PawnExtComponent;
24
25     UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category="Hak|Character")
26     TObjectPtr<UHakCameraComponent> CameraComponent;
27 }

```

HakPlayerController를 ModularPlayerController 상속으로 교체하자:

```

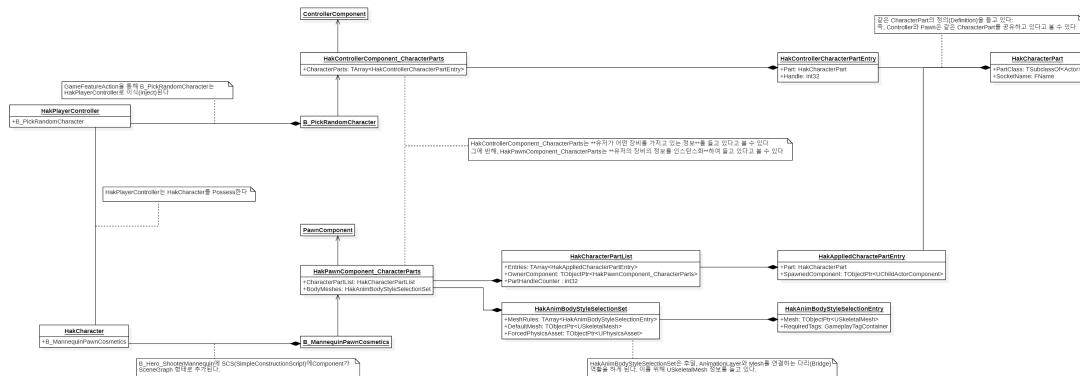
1 #pragma once
2
3 #include "ModularPlayerController.h"
4 #include "HakPlayerController.generated.h"
5
6 /**
7 * * AHakPlayerController
8 * - the base player controller class used by this project
9 */
10 UCLASS()
11 class AHakPlayerController : public AModularPlayerController
12 {
13     GENERATED_BODY()
14 public:
15     AHakPlayerController(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
16 };

```

Cosmetics Classes

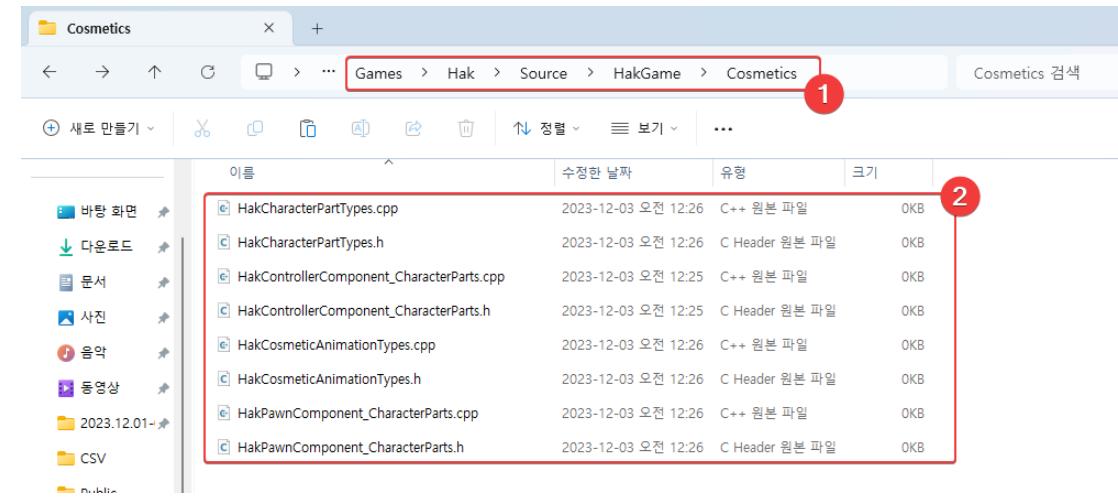
▼ 펼치기

- 전체적인 클래스 구조도 보기:



- 아래의 파일들 생성:

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



□ HakControllerComponent_CharacterParts 구현:

```

1 #pragma once
2
3 #include "Components/ControllerComponent.h"
4 #include "HakControllerComponent_CharacterParts.generated.h"
5
6 /**
7  * Controller가 Pawn을 Possess했을 때, Pawn에 어떤 Cosmetic Actor 생성할지 결정하는 ControllerComponent:
8  * - 필자는 캐릭터 파즈를 유저 관점에서 관리하는 Component로 이해한다
9  */
10 UCCLASS(Meta=(BlueprintSpawnableComponent))
11 UCLASS(HakControllerComponent_CharacterParts : public UControllerComponent
12 {
13     GENERATED_BODY()
14     public:
15         UHakControllerComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
16 }

```

```

1 #include "HakControllerComponent_CharacterParts.h"
2 #include "UE_INLINE_GENERATED_CPP_BY_NAME(HakControllerComponent_CharacterParts)
3
4 UHakControllerComponent_CharacterParts::UHakControllerComponent_CharacterParts(const FObjectInitializer& ObjectInitializer)
5     : Super(ObjectInitializer)
6 {
7 }

```

□ HakControllerComponent_CharacterParts의 멤버 변수를 구현하자:

```

1 #pragma once
2
3 #include "Components/ControllerComponent.h"
4 #include "HakControllerComponent_CharacterParts.generated.h"
5
6 /**
7  * Controller가 Pawn을 Possess했을 때, Pawn에 어떤 Cosmetic Actor 생성할지 결정하는 ControllerComponent:
8  * - 필자는 캐릭터 파즈를 유저 관점에서 관리하는 Component로 이해한다
9  */
10 UCCLASS(Meta=(BlueprintSpawnableComponent))
11 UCLASS(HakControllerComponent_CharacterParts : public UControllerComponent
12 {
13     GENERATED_BODY()
14     public:
15         UHakControllerComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
16
17         UPROPERTY(EditAnywhere, Category=Cosmetics)
18         TArray<FHakControllerCharacterPartEntry> CharacterParts;
19 }

```

□ HakControllerCharacterPartEntry

```

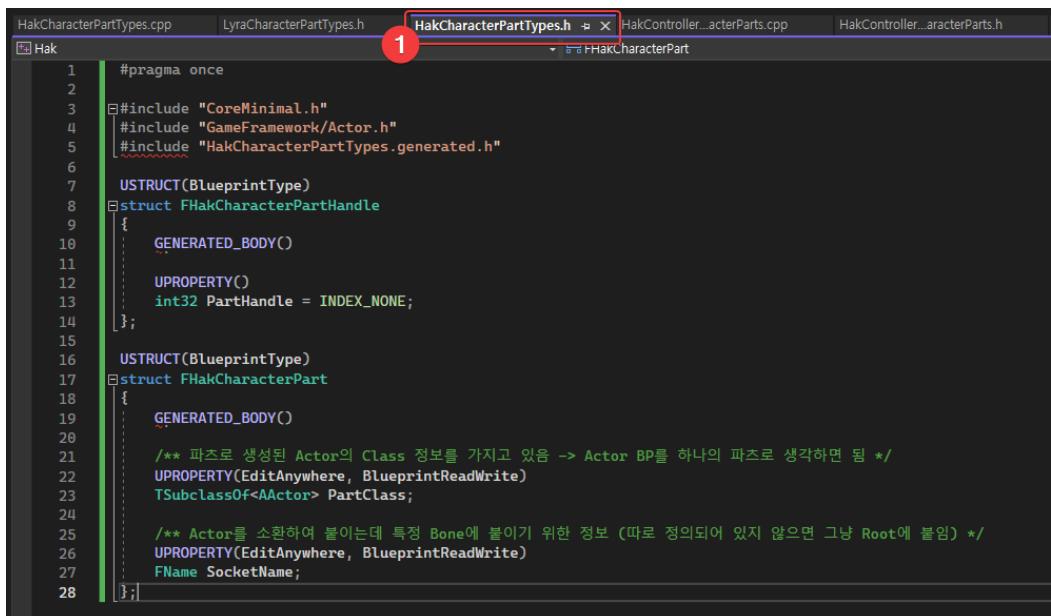
/** ControllerComponent가 소유하는 Character Parts */
USTRUCT()
struct FHakControllerCharacterPartEntry
{
    GENERATED_BODY()

    /** 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;
};

```

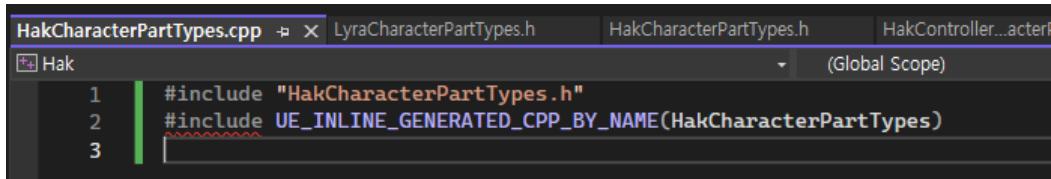
□ HakCharacterPartTypes.h/.cpp 구현:



```

1 #pragma once
2
3 #include "CoreMinimal.h"
4 #include "GameFramework/Actor.h"
5 #include "HakCharacterPartTypes.generated.h"
6
7 USTRUCT(BlueprintType)
8 struct FHakCharacterPartHandle
9 {
10     GENERATED_BODY()
11
12     UPROPERTY()
13     int32 PartHandle = INDEX_NONE;
14 };
15
16 USTRUCT(BlueprintType)
17 struct FHakCharacterPart
18 {
19     GENERATED_BODY()
20
21     /** 파즈로 생성된 Actor의 Class 정보를 가지고 있음 -> Actor BP를 하나의 파즈로 생각하면 됨 */
22     UPROPERTY(EditAnywhere, BlueprintReadWrite)
23     TSubclassOf<AActor> PartClass;
24
25     /** Actor를 소환하여 붙이는데 특정 Bone에 붙이기 위한 정보 (따로 정의되어 있지 않으면 그냥 Root에 붙임) */
26     UPROPERTY(EditAnywhere, BlueprintReadWrite)
27     FName SocketName;
28 };

```



```

1 #include "HakCharacterPartTypes.h"
2 #include UE_INLINE_GENERATED_CPP_BY_NAME(HakCharacterPartTypes)
3

```

□ HakControllerComponent_CharacterParts.h에 HakCharacterPartTypes.h 추가:

```

1 #pragma once
2
3 #include "Components/ControllerComponent.h"
4 #include "HakCharacterPartTypes.h" ①
5 #include "HakControllerComponent_CharacterParts.generated.h"
6
7 /** ControllerComponent가 소유하는 Character Parts */
8 USTRUCT()
9 struct FHakControllerCharacterPartEntry
10 {
11     GENERATED_BODY()
12
13     /** Character Part에 대한 정의(메타 데이터 == MetaData) */
14     UPROPERTY(EditAnywhere)
15     FHakCharacterPart Part;
16
17     /** Character Part 핸들 (고유값) - Controller가 Possess하고 있는 Pawn에서 생성한(인스턴스) Character Part 핸들값 */
18     FHakCharacterPartHandle Handle;
19 };
20
21 /**
22 * Controller가 Pawn을 Possess했을 때, Pawn에 어떤 Cosmetic Actor 생성할지 결정하는 ControllerComponent:
23 * - 필자는 캐릭터 파츠를 먼저 관리하는 Component로 이해한다
24 */
25 UCLASS(meta=(BlueprintSpawnableComponent))
26 class UHakControllerComponent_CharacterParts : public UControllerComponent
27 {
28     GENERATED_BODY()
29     public:
30         UHakControllerComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
31
32         UPROPERTY(EditAnywhere, Category=Cosmetics)
33         TArray<FHakControllerCharacterPartEntry> CharacterParts;
34 };

```

□ HakPawnComponent_CharacterParts 구현:

□ HakPawnComponent_CharacterParts.h/.cpp

```

1 #pragma once
2
3 #include "Components/PawnComponent.h"
4 #include "HakPawnComponent_CharacterParts.generated.h" ①
5
6 /**
7 * PawnComponent로서, Character Parts를 인스턴스화하여 관리한다
8 */
9 UCLASS(meta=(BlueprintSpawnableComponent))
10 class UHakPawnComponent_CharacterParts : public UPawnComponent
11 {
12     GENERATED_BODY()
13     public:
14         UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
15 };

```

```

1 #include "HakPawnComponent_CharacterParts.h"
2 #include UE_INLINE_GENERATED_CPP_BY_NAME(HakPawnComponent_CharacterParts)
3
4 UHakPawnComponent_CharacterParts::UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer)
5     : Super(ObjectInitializer)
6 {
7 }
8
9

```

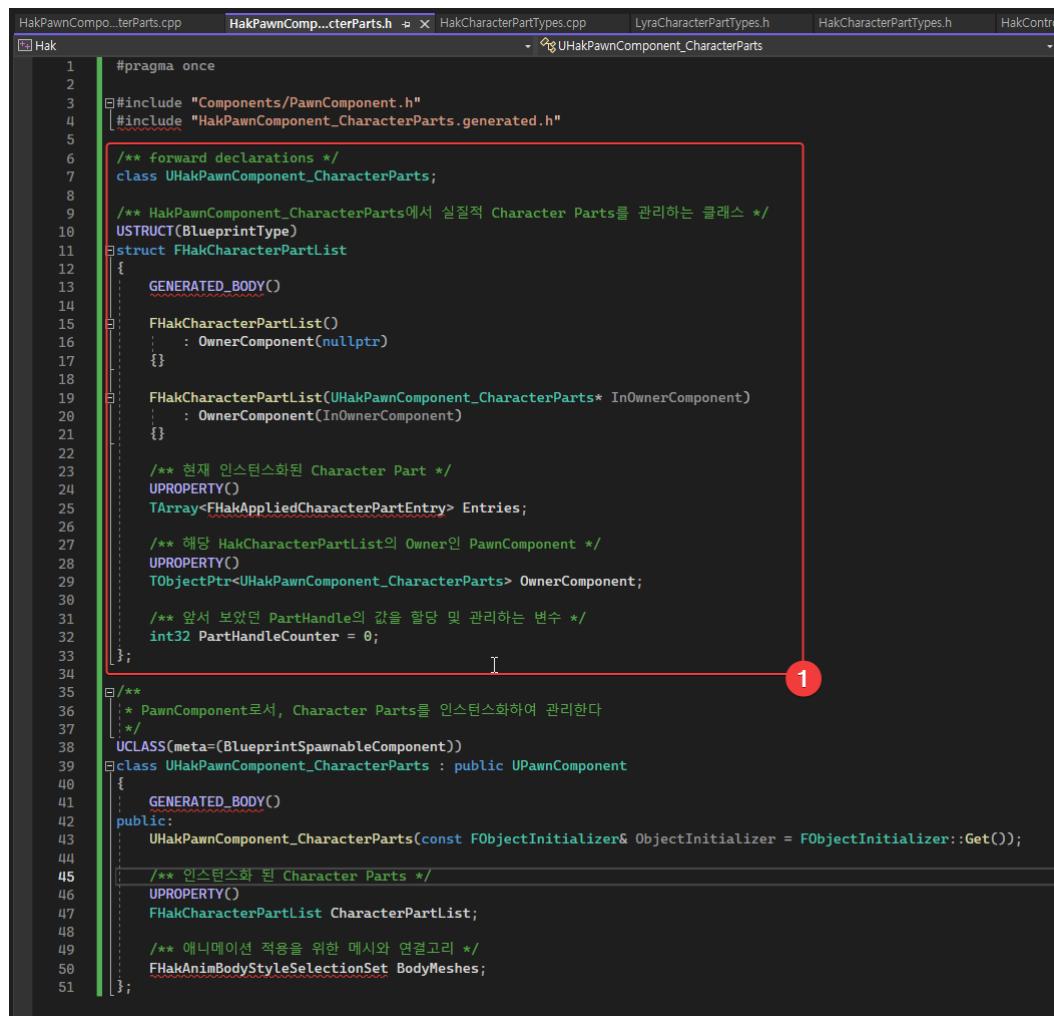
□ HakPawnComponent_CharacterParts의 멤버 변수들 구현:

```

1 #pragma once
2
3 #include "Components/PawnComponent.h"
4 #include "HakPawnComponent_CharacterParts.generated.h"
5
6 /**
7 * PawnComponent로서, Character Parts를 인스턴스화하여 관리한다
8 */
9 UCLASS(meta=(BlueprintSpawnableComponent))
10 class UHakPawnComponent_CharacterParts : public UPawnComponent
11 {
12     GENERATED_BODY()
13 public:
14     UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
15
16     /** 인스턴스화 된 Character Parts */
17     UPROPERTY()
18     FHakCharacterPartList CharacterPartList;
19
20     /** 애니메이션 적용을 위한 메시와 연결고리 */
21     FHakAnimBodyStyleSelectionSet BodyMeshes;
22 };

```

□ HakCharacterPartList



```

HakPawnCompo...terParts.cpp   HakPawnCom...terParts.h   HakCharacterPartTypes.cpp   Lyr...CharacterPartTypes.h   HakCharacterPartTypes.h   HakContr...
Hak
1 #pragma once
2
3 #include "Components/PawnComponent.h"
4 #include "HakPawnComponent_CharacterParts.generated.h"
5
6 /**
7 * forward declarations */
8 class UHakPawnComponent_CharacterParts;
9
10 /**
11 * HakPawnComponent_CharacterParts에서 실질적 Character Parts를 관리하는 클래스 */
12 USTRUCT(BlueprintType)
13 struct FHakCharacterPartList
14 {
15     GENERATED_BODY()
16
17     FHakCharacterPartList()
18     : OwnerComponent(nullptr)
19     {}
20
21     FHakCharacterPartList(UHakPawnComponent_CharacterParts* InOwnerComponent)
22     : OwnerComponent(InOwnerComponent)
23     {}
24
25     /** 현재 인스턴스화된 Character Part */
26     UPROPERTY()
27     TArray<FHakAppliedCharacterPartEntry> Entries;
28
29     /** 해당 HakCharacterPartList의 Owner의 PawnComponent */
30     UPROPERTY()
31     TObjectPtr<UHakPawnComponent_CharacterParts> OwnerComponent;
32
33     /** 앞서 보았던 PartHandle의 값을 할당 및 관리하는 변수 */
34     int32 PartHandleCounter = 0;
35 };
36
37 /**
38 * PawnComponent로서, Character Parts를 인스턴스화하여 관리한다
39 */
40 UCLASS(meta=(BlueprintSpawnableComponent))
41 class UHakPawnComponent_CharacterParts : public UPawnComponent
42 {
43     GENERATED_BODY()
44 public:
45     UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
46
47     /** 인스턴스화 된 Character Parts */
48     UPROPERTY()
49     FHakCharacterPartList CharacterPartList;
50
51 };

```

□ HakAppliedCharacterPartEntry

```

1 #pragma once
2
3 #include "HakCharacterPartTypes.h" ①
4 #include "Components/PawnComponent.h"
5 #include "HakPawnComponent_CharacterParts.generated.h"
6
7 /** forward declarations */
8 class UHakPawnComponent_CharacterParts;
9
10 USTRUCT()
11 struct FHakAppliedCharacterPartEntry
12 {
13     GENERATED_BODY()
14
15     /** Character Part의 정의(메타 데이터) */
16     UPROPERTY()
17     FHakCharacterPart Part;
18
19     /** HakCharacterPartList에서 할당 받은 Part 핸들(FHakControllerCharacterPartEntry의 Handle 값과 같아야 함 -> 같은 Part) */
20     UPROPERTY()
21     int32 PartHandle = INDEX_NONE;
22
23     /** 인스턴스화 된 Character Part용 Actor */
24     UPROPERTY()
25     TObjectPtr<UChildActorComponent> SpawnerComponent = nullptr;
26 };
27
28
29 /** HakPawnComponent_CharacterParts에서 실질적 Character Parts를 관리하는 클래스 */
30 USTRUCT(BlueprintType)
31 struct FHakCharacterPartList
32 {
33     GENERATED_BODY()
34
35     FHakCharacterPartList() : OwnerComponent(nullptr)
36     {}
37
38     FHakCharacterPartList(UHakPawnComponent_CharacterParts* InOwnerComponent) : OwnerComponent(InOwnerComponent)
39     {}
40
41
42     /** 현재 인스턴스화된 Character Part */
43     UPROPERTY()
44     TArray<FHakAppliedCharacterPartEntry> Entries;
45
46     /** 해당 HakCharacterPartList의 Owner인 PawnComponent */
47     UPROPERTY()
48     TObjectPtr<UHakPawnComponent_CharacterParts> OwnerComponent;
49
50     /** 앞서 보았던 PartHandle의 값을 할당 및 관리하는 변수 */
51     int32 PartHandleCounter = 0;
52 };
53
54
55 /**
56  * PawnComponent로서, Character Parts를 인스턴스화하여 관리한다
57 */
58 UCLASS(meta=(BlueprintSpawnableComponent))
59 class UHakPawnComponent_CharacterParts : public UPawnComponent
60 {
61     GENERATED_BODY()
62     public:
63     UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
64
65     /** 인스턴스화 된 Character Parts */
66     UPROPERTY()
67     FHakCharacterPartList CharacterPartList;
68
69     /** 애니메이션 적용을 위한 메시와 연결고리 */
70     FHakAnimBodyStyleSelectionSet BodyMeshes;
71 };

```

□ HakCosmeticAnimationTypes.h/.cpp

```

1 #pragma once
2
3 #include "HakCosmeticAnimationTypes.generated.h"
4
5 /** forward declarations */
6 class USkeletalMesh;
7 class UPhysicsAsset;
8
9 USTRUCT(BlueprintType)
10 struct FHakAnimBodyStyleSelectionSet
11 {
12     GENERATED_BODY()
13
14     /** AnimLayer 적용할 SkeletalMesh를 들고 있음 -> Animation-Mesh간 Rules를 MeshRules라고 생각하면 됨 */
15     UPROPERTY(EditAnywhere, BlueprintReadWrite)
16     TArray<FHakAnimBodyStyleSelectionEntry> MeshRules;
17
18     /** 그냥 디폴트로 적용할 SkeletalMesh */
19     UPROPERTY(EditAnywhere, BlueprintReadWrite)
20     TObjectPtr<USkeletalMesh> DefaultMesh = nullptr;
21
22     /** Physics Asset은 하나로 등일함 -> 즉 모든 Animation의 Physics 속성은 공유함 */
23     UPROPERTY(EditAnywhere)
24     TObjectPtr<UPhysicsAsset> ForcedPhysicsAsset = nullptr;
25 };

```

```

1 #include "HakCosmeticAnimationTypes.h"
2 #include UE_INLINE_GENERATED_CPP_BY_NAME(HakCosmeticAnimationTypes)
3

```

□ HakAnimBodyStyleSelectionEntry:

```

1 #pragma once
2
3 #include "GameplayTagContainer.h" 1
4 #include "HakCosmeticAnimationTypes.generated.h"
5
6 /** forward declarations */
7 class USkeletalMesh;
8 class UPhysicsAsset;
9
10 USTRUCT(BlueprintType)
11 struct FHakAnimBodyStyleSelectionEntry
12 {
13     GENERATED_BODY()
14
15     /** AnimLayer를 적용할 대상 SkeletalMesh */
16     UPROPERTY(EditAnywhere, BlueprintReadWrite)
17     TObjectPtr<USkeletalMesh> Mesh = nullptr;
18
19     /** Cosmetic Tag라고 생각하면 됨 */
20     UPROPERTY(EditAnywhere, BlueprintReadWrite, meta=(Categories="Cosmetic"))
21     FGameplayTagContainer RequiredTags;
22 };
23
24 USTRUCT(BlueprintType)
25 struct FHakAnimBodyStyleSelectionSet
26 {
27     GENERATED_BODY()
28
29     /** AnimLayer 적용할 SkeletalMesh를 들고 있음 -> Animation-Mesh간 Rules을 MeshRules라고 생각하면 됨 */
30     UPROPERTY(EditAnywhere, BlueprintReadWrite)
31     TArray<FHakAnimBodyStyleSelectionEntry> MeshRules;
32
33     /** 그냥 디폴트로 적용할 SkeletalMesh */
34     UPROPERTY(EditAnywhere, BlueprintReadWrite)
35     TObjectPtr<USkeletalMesh> DefaultMesh = nullptr;
36
37     /** Physics Asset은 하나로 동일함 -> 즉 모든 Animation의 Physics 속성은 공유함 */
38     UPROPERTY(EditAnywhere)
39     TObjectPtr<UPhysicsAsset> ForcedPhysicsAsset = nullptr;
40 };

```

□ HakPawnComponent_CharacterParts.h에
HakCosmeticAnimationTypes.h 추가:

```

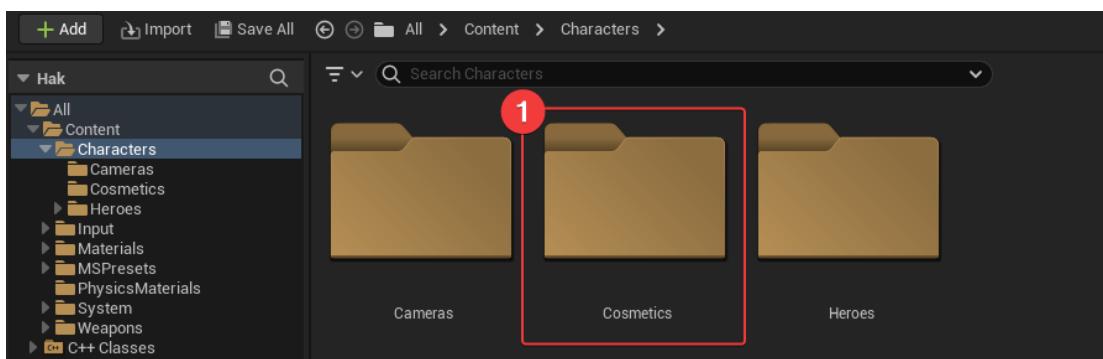
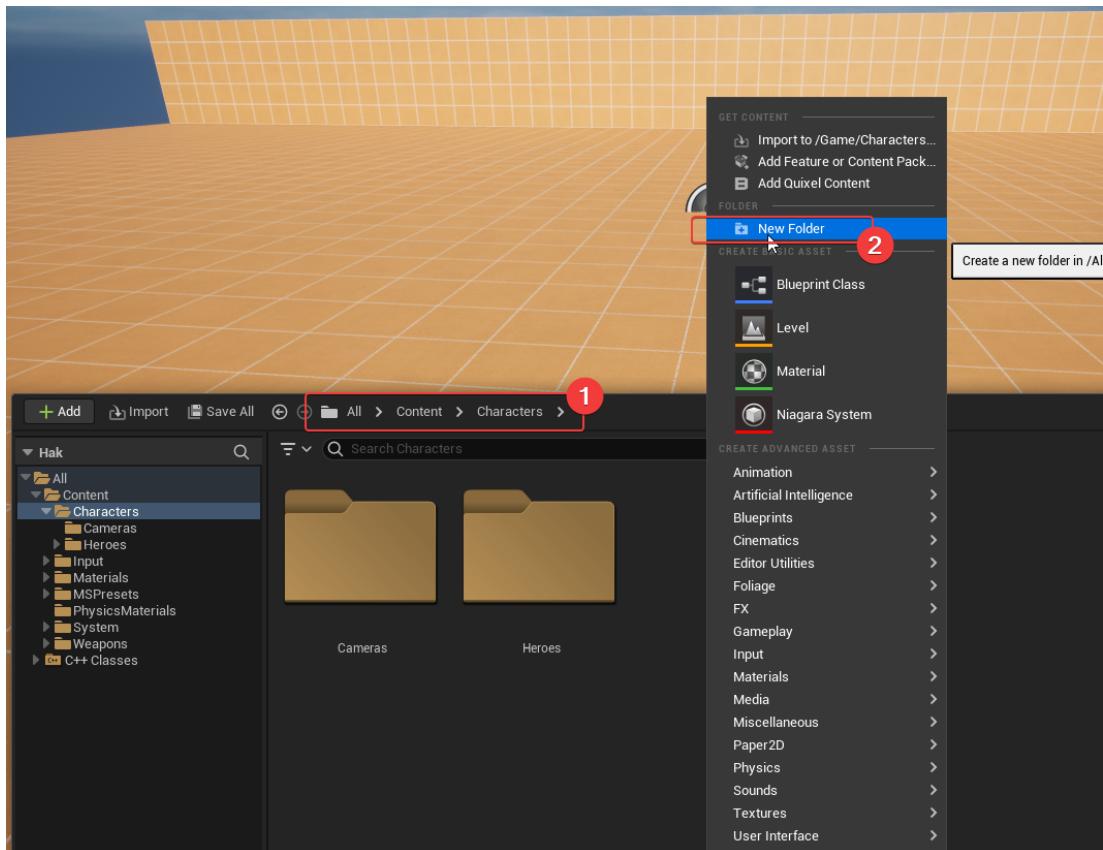
1 #pragma once
2
3 #include "HakCharacterPartTypes.h" 1
4 #include "HakCosmeticAnimationTypes.h"
5 #include "Components/PawnComponent.h"
6 #include "HakPawnComponent_CharacterParts.generated.h"
7
8 /** forward declarations */
9 class UHakPawnComponent_CharacterParts;
10
11 /** 인스턴스화 된 Character Part의 단위 */
12 USTRUCT()
13 struct FHakAppliedCharacterPartEntry
14 {
15     GENERATED_BODY()
16
17     /** Character Part의 정의(메타 데이터) */
18     UPROPERTY()
19     FHakCharacterPart Part;
20
21     /** HakCharacterPartList에서 할당 받은 Part 헤더값 (FHakControllerCharacterPartEntry의 Handle 값과 같아야 함 -> 같으면 같은 Part) */
22     UPROPERTY()
23     int32 PartHandle = INDEX_NONE;
24
25     /** 인스턴스화 된 Character Part용 Actor */
26     UPROPERTY()
27     TObjectPtr<UChildActorComponent> SpawnerComponent = nullptr;
28 }
29

```

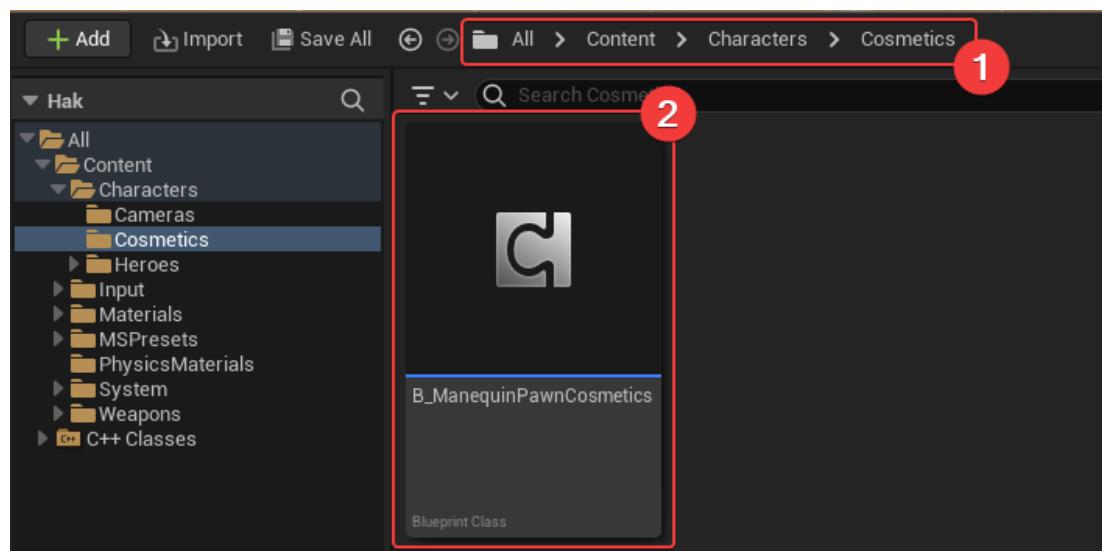
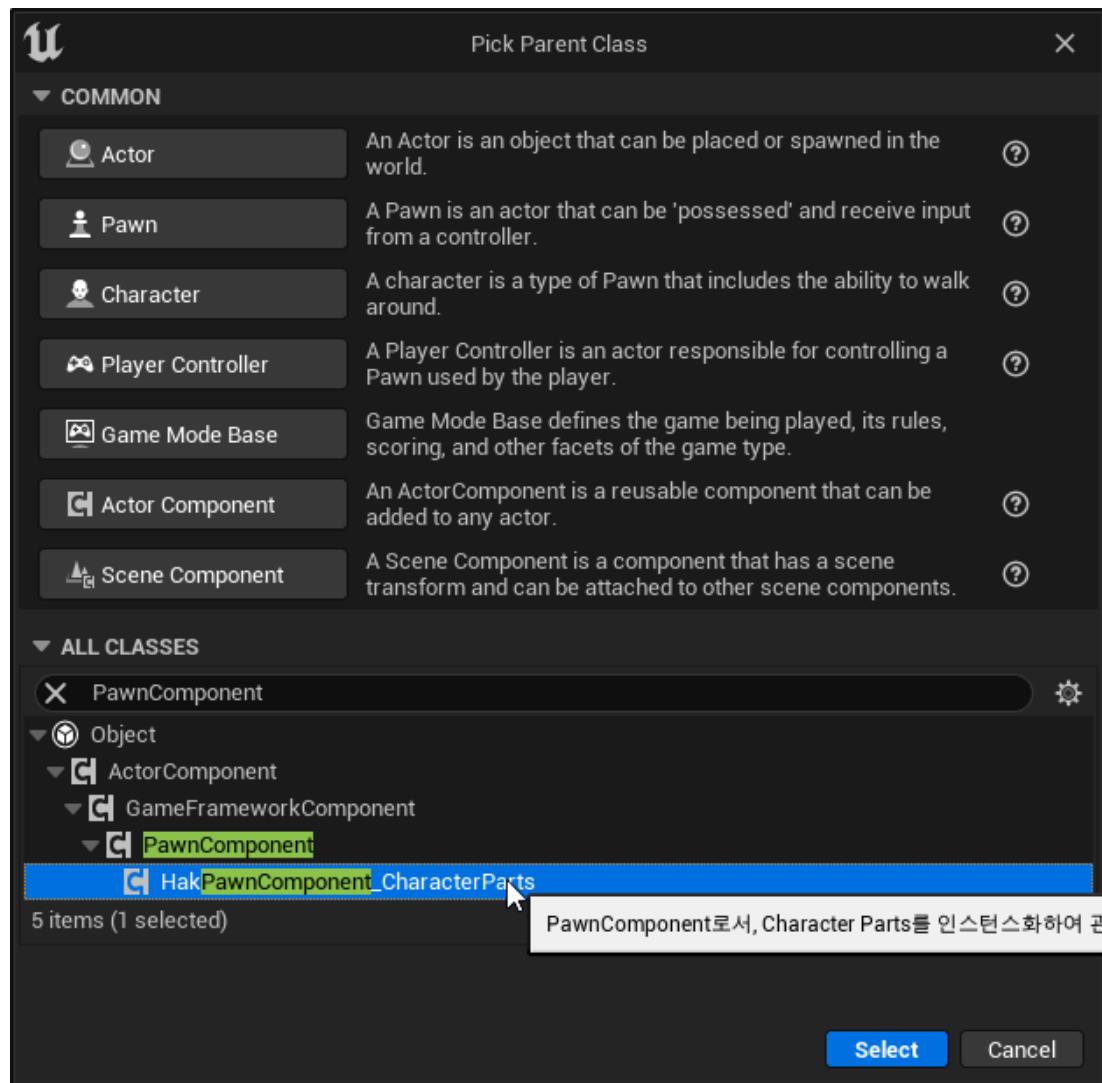
Cosmetics Assets

▼ 편집기

- Root Content 폴더에 `characters/Cosmetics` 를 생성해주자:



- `HakPawnComponent_CharacterParts`를 상속 받는 `B_ManequinPawnCosmetics`를 생성하자:



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

```

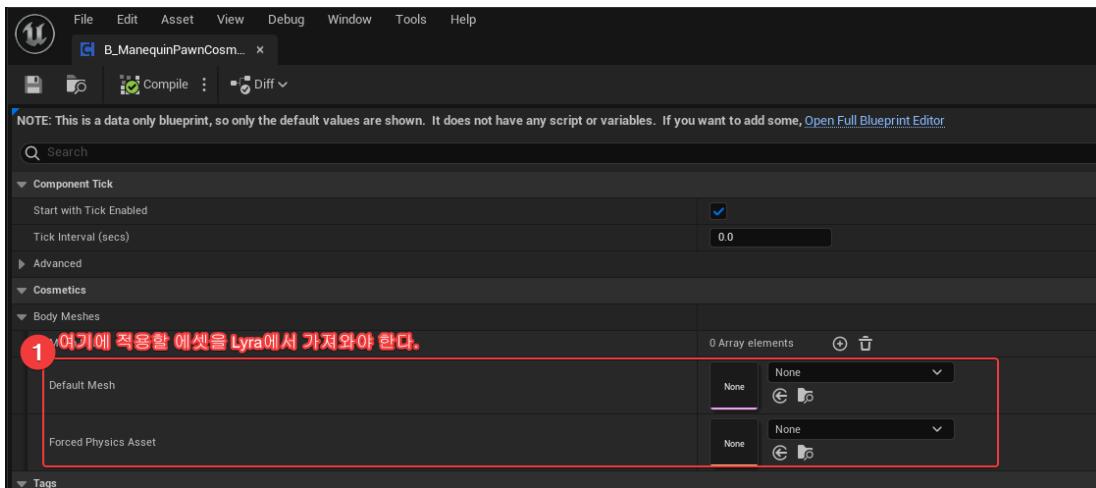

    /**
     * PawnComponent로서, Character Parts를 인스턴스화하여 관리한다
     */
    UCLASS(meta=(BlueprintSpawnableComponent))
    class UHakPawnComponent_CharacterParts : public UPawnComponent
    {
        GENERATED_BODY()
    public:
        UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer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

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

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

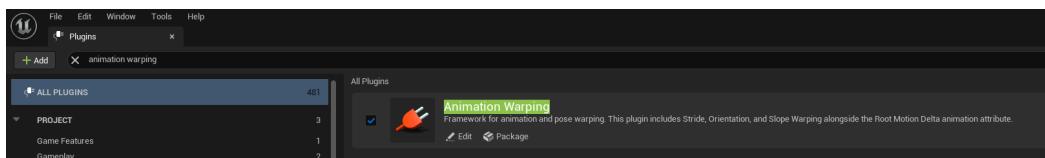
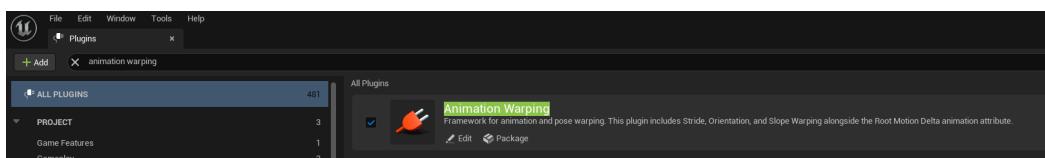

```

- 아래와 같이 적용할 에셋을 Lyra에서 가져와야 한다:



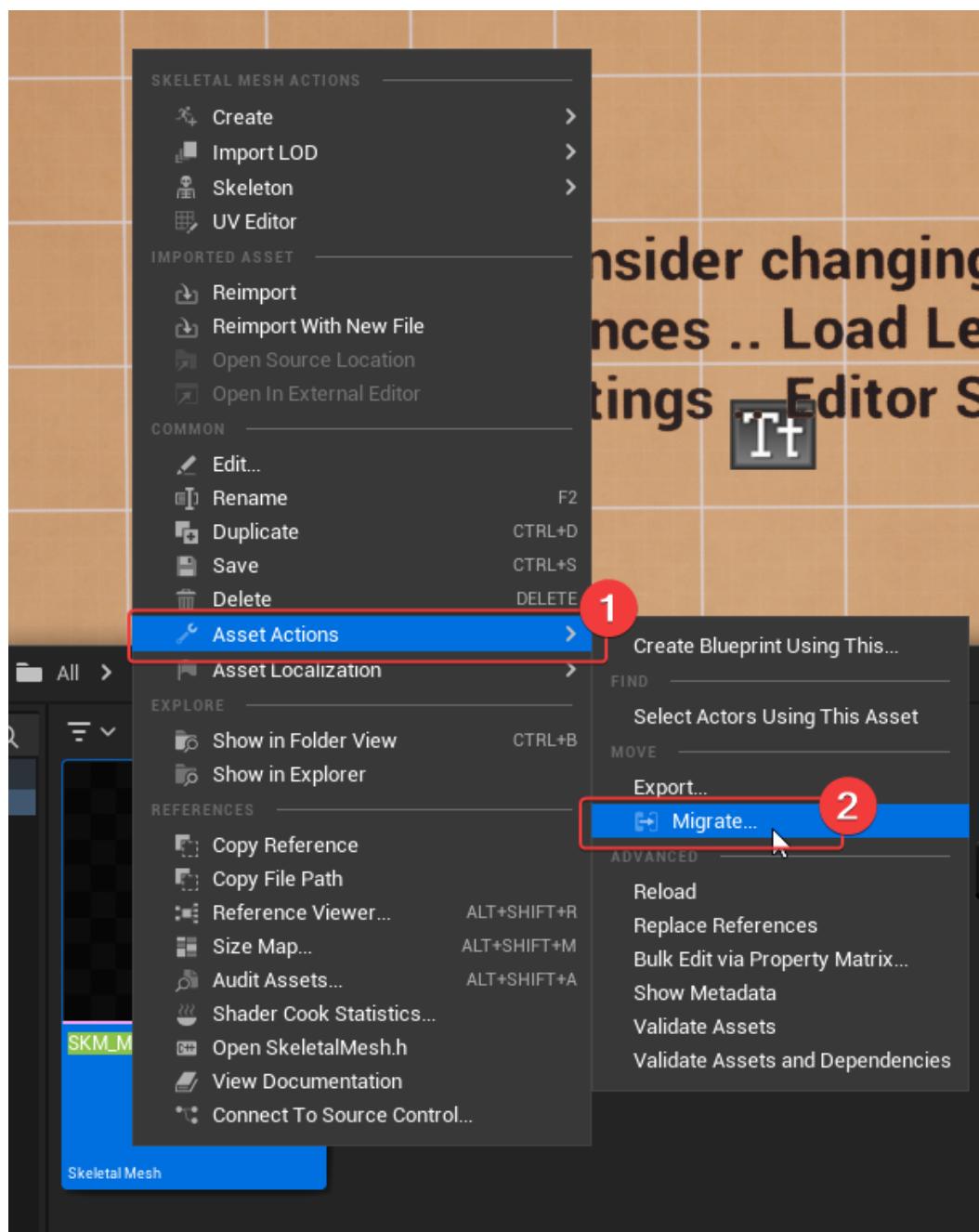
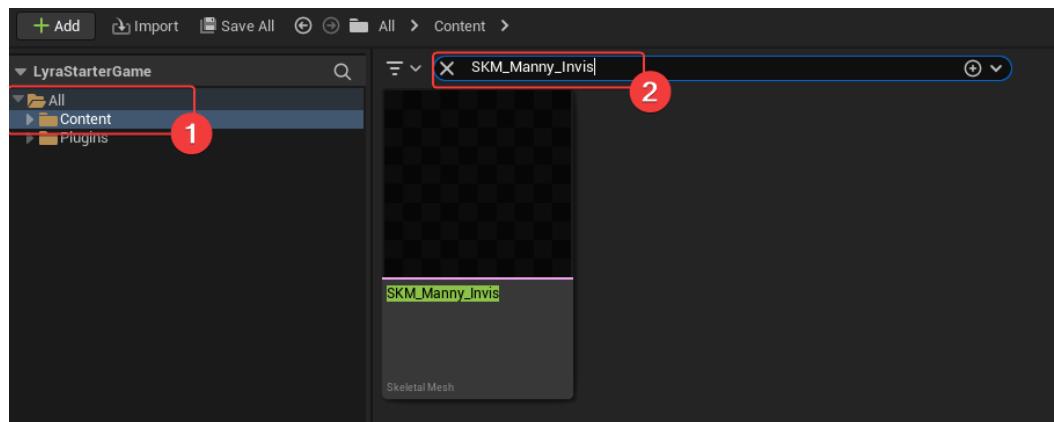
- 우선 Lyra에서 에셋을 Import하기 전에 Lyra에서 활용하는 Plugin들을 미리 활성화 해놓자:

- `AnimationLocomotionLibrary` 와 `AnimationWarping`:

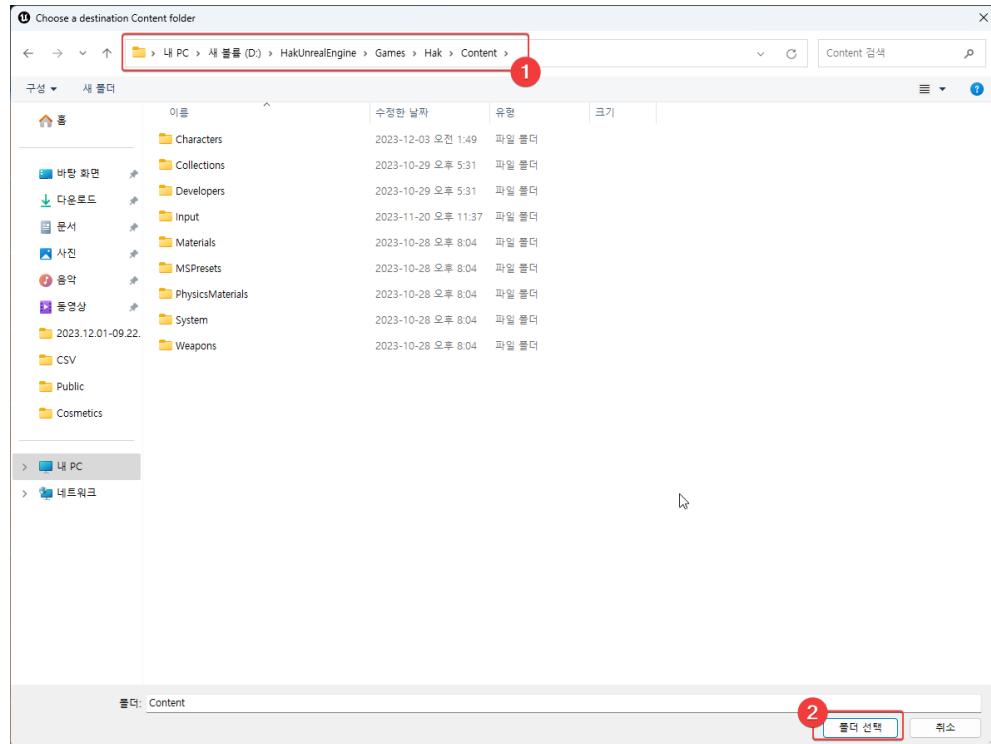


- LyraStarterGame을 실행해주자:

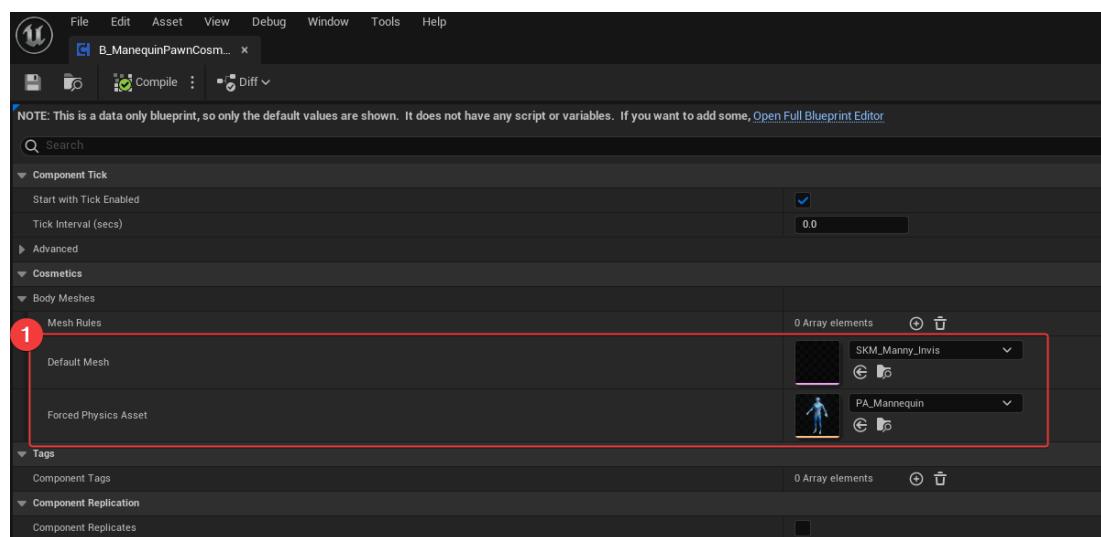
- 우선 SKM_Manny_Invis를 Mirgrate 시켜주자:



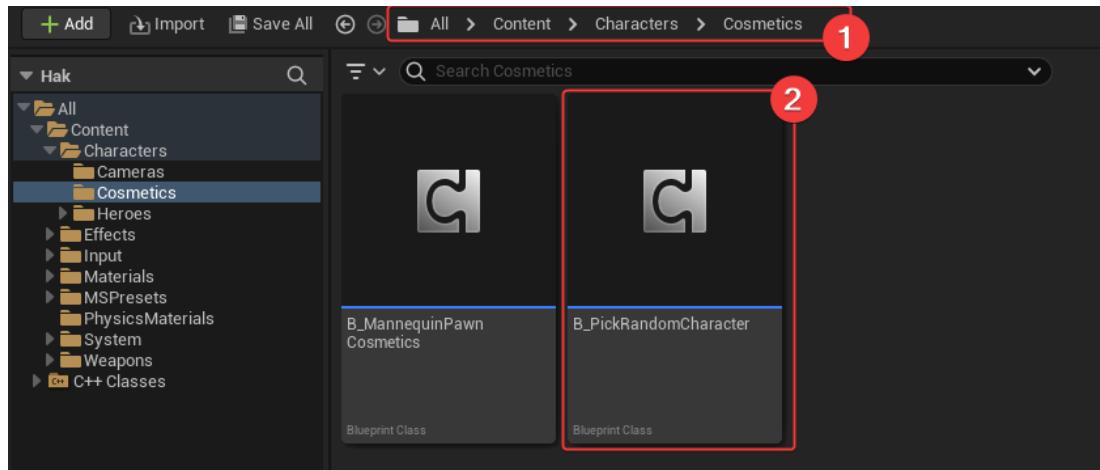
- 우리의 Root Content에 경로를 설정하면 된다:



- 다음은 PA_Mannequin을 Migrate를 위와 같이 해주자
- 아래와 같이 B_MannequinPawnCosmetics를 설정해주자:

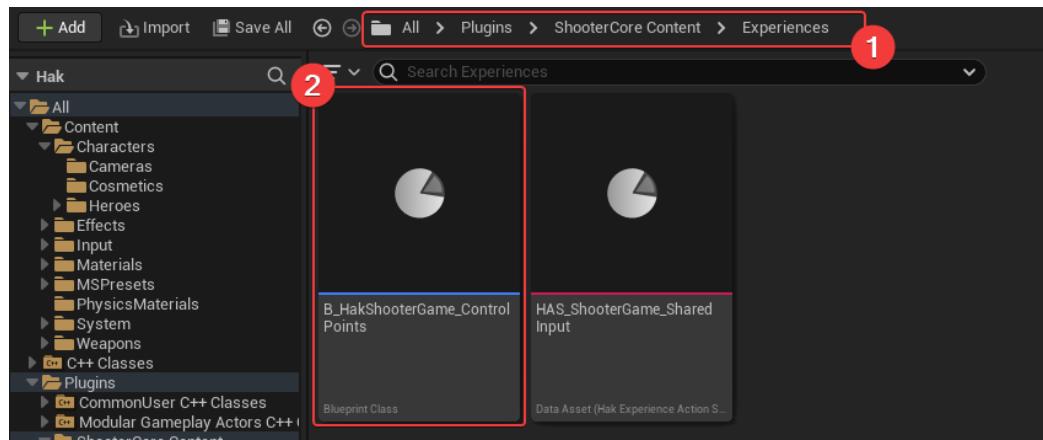


- HakControllerComponent_CharacterParts를 상속받은 B_PickRandomCharacter를 생성하자:



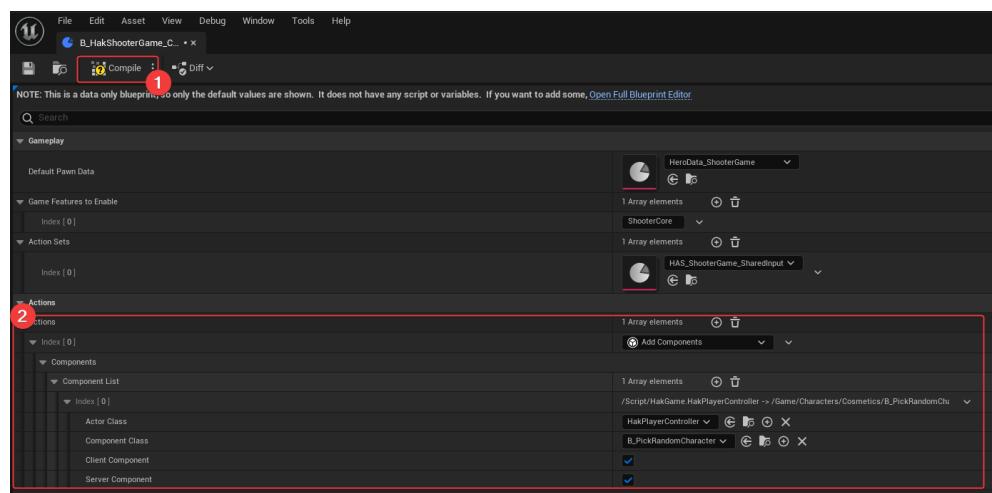
□ GameFeatureAction을 활용하여, B_PickRandomCharacter를 ShooterCore Plugin이 활성화 될 때, 알맞은 Actor에 붙여주자:

- B_HakShooterGame_ControlPoints에 추가하자:

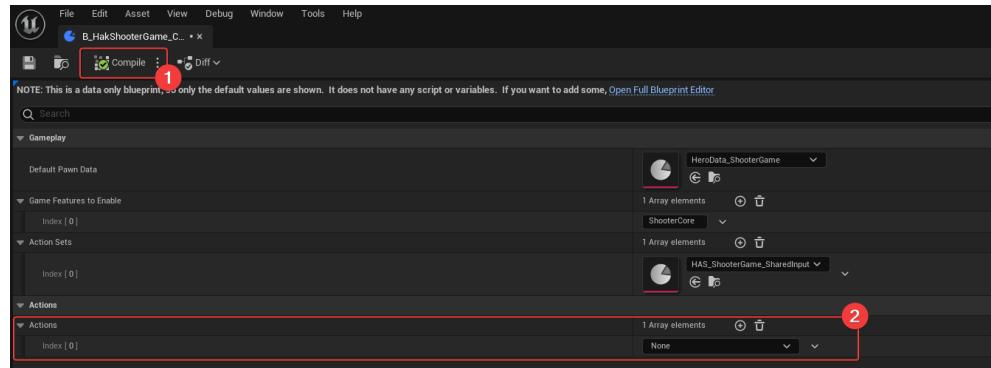


□ 아래와 같이 세팅한 후, 컴파일하면 없어진다?!

- 컴파일 전:



- 컴파일 후:



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

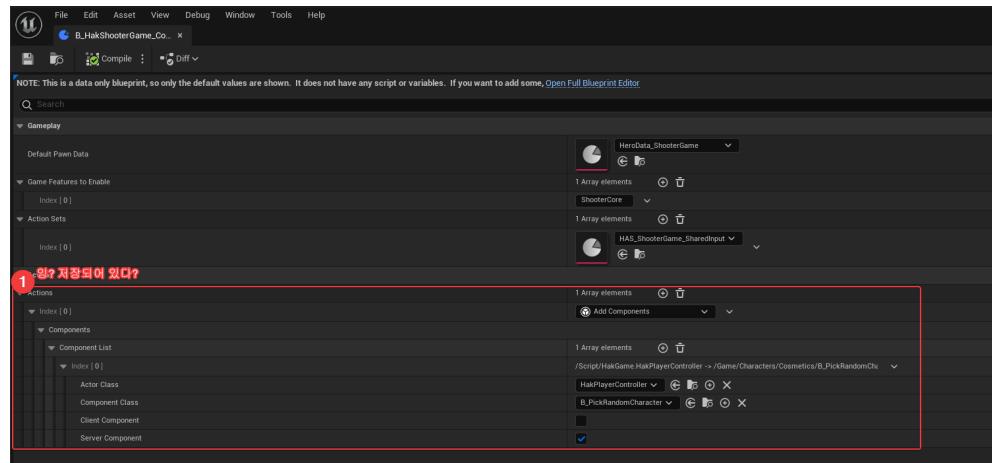
```

HakExperienceDefinition.h  HakPawnCompo...terParts.cpp  HakPawnCompo...acterParts.h  HakCharacterPartTypes.cpp  LyraCharacterPartTypes.h  HakPlayerController...
mak  1

1  #pragma once
2
3  #include "Engine/DataAsset.h"
4  #include "HakExperienceDefinition.generated.h"
5
6  /** forward declaration */
7  class UHakPawnData;
8  class UGameFeatureAction;
9  class UHakExperienceActionSet;
10
11 /**
12  * UHakExperienceDefinition
13  * - definition of an experience
14  */
15 UCCLASS()
16 class UHakExperienceDefinition : public UPrimaryDataAsset
17 {
18     GENERATED_BODY()
19 public:
20     UHakExperienceDefinition(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
21
22     /** the default pawn class to spawn for players */
23     UPROPERTY(EditDefaultsOnly, Category=Gameplay)
24     TObjectPtr<UHakPawnData> DefaultPawnData;
25
26     /** lost if game feature plugins this experience wants to have active */
27     // 해당 property는 단순히 마킹 및 기억용으로 남겨놓도록 하겠다:
28     // - GameMode에 따라 필요한 GameFeature plugin들을 로딩하는데 이에 대한 연결고리로 생각하면 된다 (현재는 쓰지 않음)
29     UPROPERTY(EditDefaultsOnly, Category=Gameplay)
30     TArray< FString > GameFeaturesToEnable;
31
32     /** ExperienceActionSet은 UGameFeatureAction의 Set이며, Gameplay 송도에 맞게 분류의 목적으로 사용한다 */
33     UPROPERTY(EditDefaultsOnly, Category = Gameplay)
34     TArray< TObjectPtr<UHakExperienceActionSet>> ActionSets;
35
36     /** 일반적인 GameFeatureAction으로서 */
37     UPROPERTY(EditDefaultsOnly, [Instanced], Category="Actions")
38     TArray< TObjectPtr< UGameFeatureAction >> Actions;
39 };

```

- 변경 후, 다시 열어보면?



- 무슨 일이 벌어지고 있는걸까?

Instances 오브젝트 (UCLASS) 프로퍼티 전용입니다. 이 클래스의 인스턴스가 생성될 때, 이 프로퍼티에 기본으로 할당된 오브젝트 고유 사본을 쿠어줍니다. 클래스 디플트 프로퍼티에 정의된 서보오브젝트 인스턴싱에 사용됩니다. EditInline 및 Export 를 강조합니다.

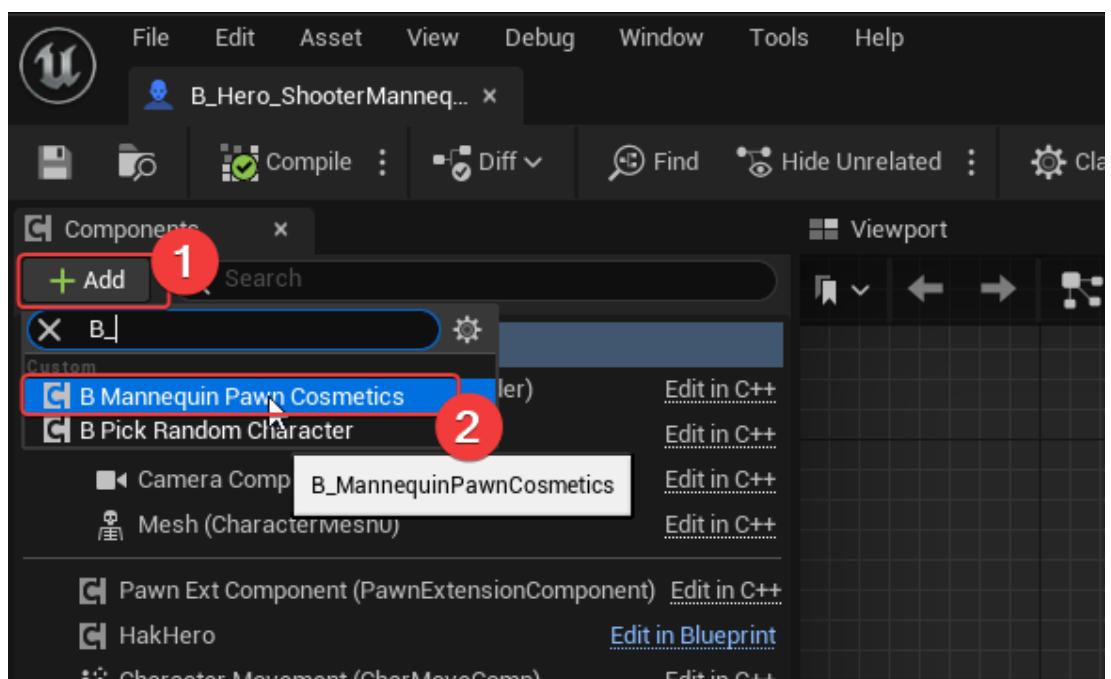
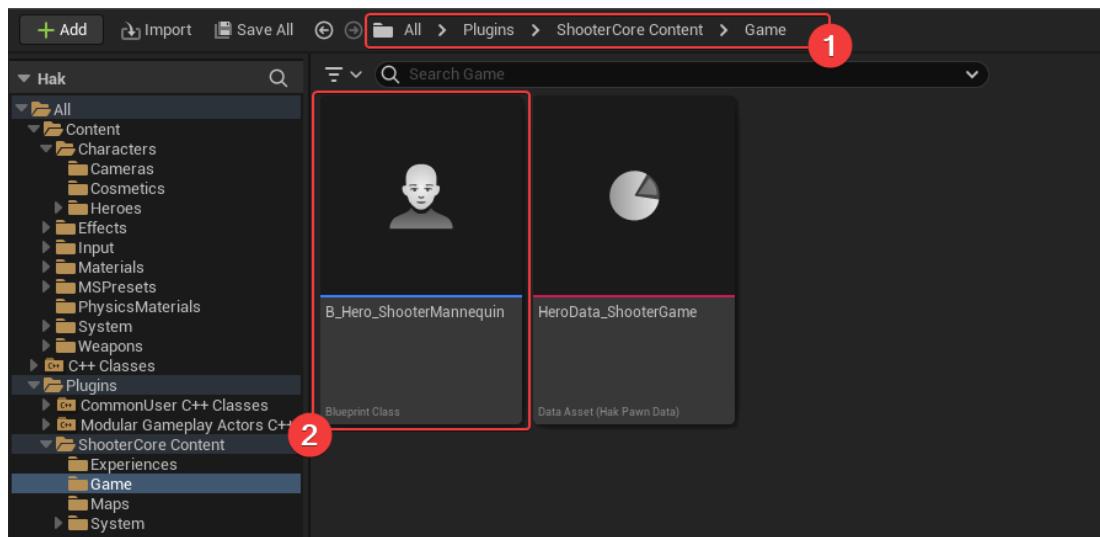
우리가 변경하고자 하는 대상은 BP이다:

- 사실 BP는 클래스 생성하는 객체로 초기값을 멤버 변수가 가질 수 없다 (항상 리셋된다...)
- 이를 가능하게 하는 기능은 Instanced라고 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 가지 고 있다... Instanced 설정 이후, 값이 보여질 뿐이다... 😅 (일단 여기서는 그냥 넘어가자...)

- B_Hero_ShooterMannequin에 생성한 B_MannequinPawnCosmetics를 추가하자:



- 이름은 `PawnCosmeticsComponent`

AddCharacterPart

▼ 펼치기

- `B_PickRandomCharacter`의 `BeginPlay` 노드를 작성하기 앞서, `AddCharacterPart`을 추가하자:

- `HakControllerComponent_CharacterParts`:

```

1 /**
 * Controller가 Pawn을 Possess했을 때, Pawn에 어떤 Cosmetic Actor 생성할지 결정하는 ControllerComponent:
 * - 필자는 캐릭터 피즈를 유저 관점에서 관리하는 Component로 이해한다
 */
UCLASS(meta=(BlueprintSpawnableComponent))
class UHakControllerComponent_CharacterParts : public UControllerComponent
{
    GENERATED_BODY()
public:
    UFUNCTION(BlueprintCallable, Category=Cosmetics)
    void AddCharacterPart(const FHakCharacterPart& NewPart); ①

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

```

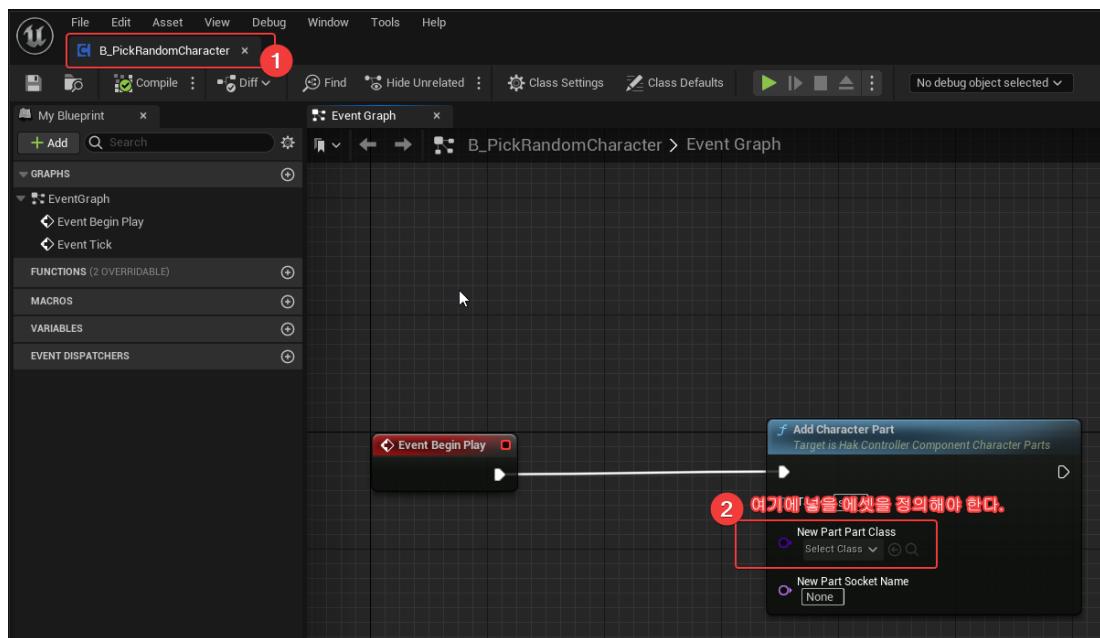
```

3
4     : Super(ObjectInitializer)
5
6     {
7     }
8
9     void UHakControllerComponent_CharacterParts::AddCharacterPart(const FHakCharacterPart& NewPart) ①
10    {
11    }
12

```

1 일반 더미로 넣어두자

- AddCharacterPart를 사용하여, B_PickRandomCharacter의 BeginPlay 로직을 우선 완성하자:



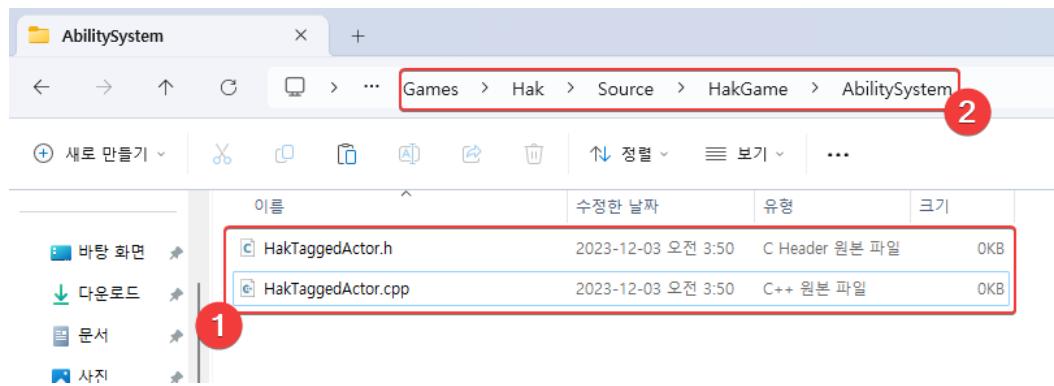
- New Part Class에는 B_Manny라는 BP를 생성할 것이다:

- 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:

```

HakTaggedActor.cpp   HakTaggedActor.h  HakExperienceDefinition.h  HakCharacterPartTypes.cpp  LyraCharacterPartTypes.h
Hak
1 #pragma once
2
3 #include "GameFramework/Actor.h"
4 #include "HakTaggedActor.generated.h"
5
6 UCLASS()
7 class AHakTaggedActor : public AActor
8 {
9     GENERATED_BODY()
10 public:
11     AHakTaggedActor(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
12 };

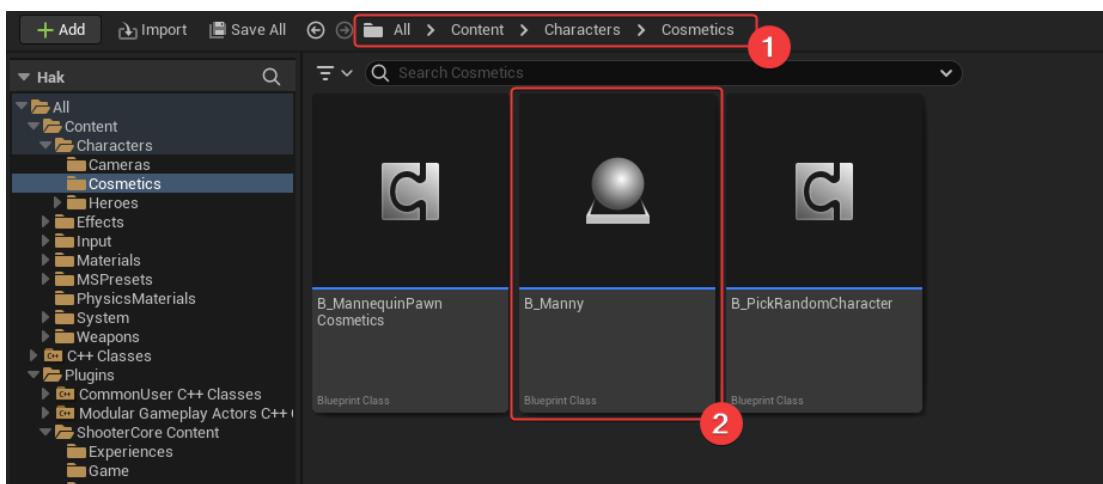
```

```

HakTaggedActor.cpp  HakTaggedActor.h  HakExperienceDefinition.h  HakCharacterPartTypes.cpp  LyraCharacterPartTypes.h
Hak
1 #include "HakTaggedActor.h"
2 #include UE_INLINE_GENERATED_CPP_BY_NAME(HakTaggedActor)
3
4 AHakTaggedActor::AHakTaggedActor(const FObjectInitializer& ObjectInitializer)
5     : Super(ObjectInitializer)
6 {
7 }
8

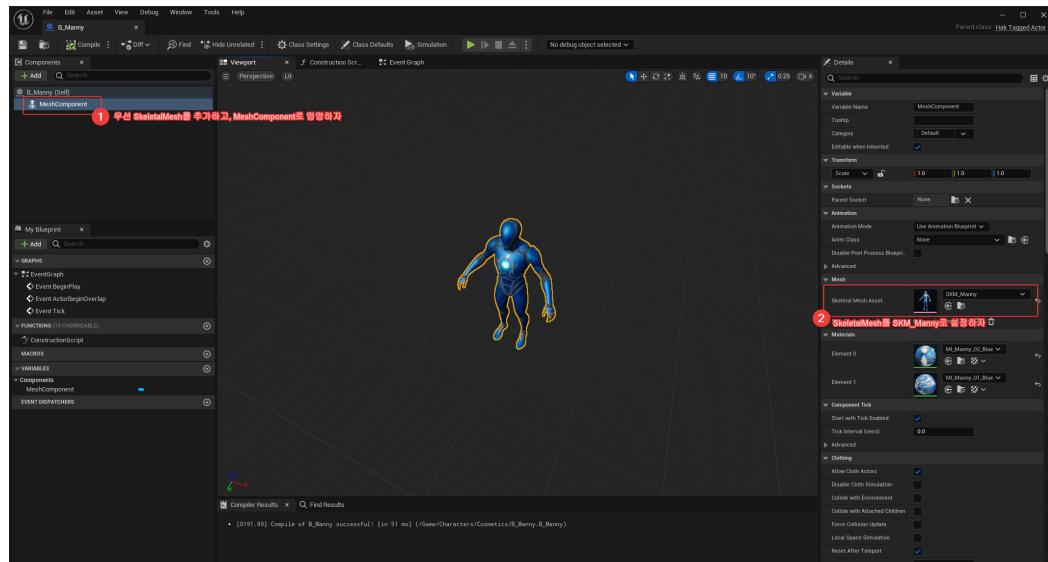
```

□ 이제 B_Manny BP를 생성해보자:

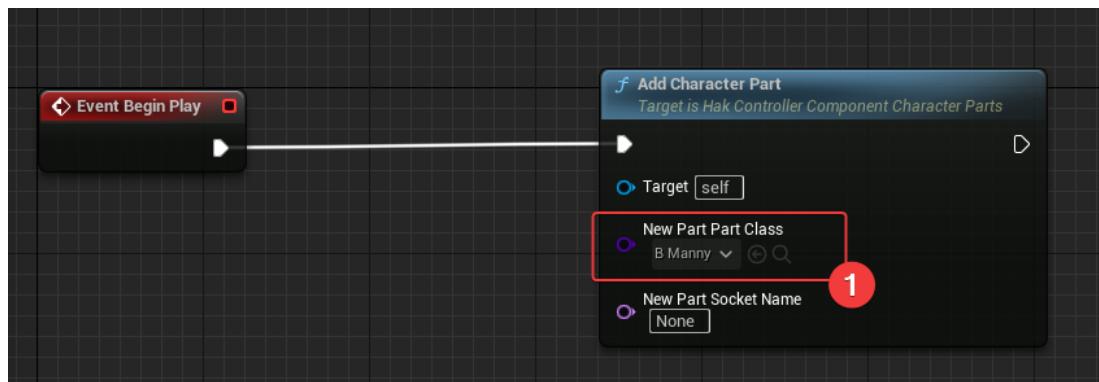


□ B_Manny에 SkeletalMesh를 추가하자

- 우리의 BaseMesh가 될 대상이다

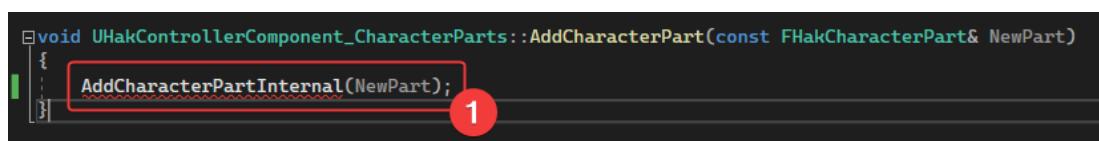


□ 생성한 B_Manny를 활용하여, B_PickRandomCharacter의 BeginPlay의 로직을 완성하자:



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

□ AddCharacterPart() 로직을 C++ 구현하자:



□ AddCharacterPartInternal()

```
/**  
 * Controller가 Pawn을 Possess했을 때, Pawn에 어떤 Cosmetic Actor 생성할지 결정하는 ControllerComponent:  
 * - 필자는 캐릭터 파츠를 유지 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); 1  
  
    UFUNCTION(BlueprintCallable, Category=Cosmetics)  
    void AddCharacterPart(const FHakCharacterPart& NewPart);  
  
    UPROPERTY(EditAnywhere, Category=Cosmetics)  
    TArray<FHakControllerCharacterPartEntry> CharacterParts;  
};
```

```
#include "HakControllerComponent_CharacterParts.h"  
#include "HakPawnComponent_CharacterParts.h" 1  
#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);  
        }  
    } 2  
  
    void UHakControllerComponent_CharacterParts::AddCharacterPart(const FHakCharacterPart& NewPart)  
    {  
        AddCharacterPartInternal(NewPart);  
    }  
}
```

- 우리는 아래 두 가지를 구현해야 한다:

1. GetPawnCustomizer()
2. HakPawnComponent_CharacterParts::AddCharacterPart()

□ GetPawnCustomizer()

```
/**  
 * Controller가 Pawn을 Possess했을 때, Pawn에 어떤 Cosmetic Actor 생성할지 결정하는 ControllerComponent:  
 * - 필자는 캐릭터 파츠를 유지 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; 1  
    void AddCharacterPartInternal(const FHakCharacterPart& NewPart);  
  
    UFUNCTION(BlueprintCallable, Category=Cosmetics)  
    void AddCharacterPart(const FHakCharacterPart& NewPart);  
  
    UPROPERTY(EditAnywhere, Category=Cosmetics)  
    TArray<FHakControllerCharacterPartEntry> CharacterParts;  
};
```

```

#include "HakControllerComponent_CharacterParts.h"
#include "HakPawnComponent_CharacterParts.h" 1
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakControllerComponent_CharacterParts)

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

UHakControllerComponent_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()

```


/**
 * 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); 1
    /** 인스턴스화 된 Character Parts */
    UPROPERTY()
    FHakCharacterPartList CharacterPartList;

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


```

```

#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); 1
}

```

□ FHakCharacterPartList::AddEntry()

```

/** 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); 1

    /** 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;
};
```

```

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

FHakCharacterPartHandle FHakCharacterPartList::AddEntry(FHakCharacterPart NewPart)
{
    // PawnComponent의 CharacterPartList가 PartHandle을 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)
```

□ SpawnActorForEntry()

```

1 #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)

2 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 볼일 것인지 결정한다:
        // - GetSceneComponentToAttachTo
        if (USceneComponent* ComponentToAttachTo = OwnerComponent->GetSceneComponentToAttachTo())
        {
            // 볼일 Component인 ComponentToAttachTo의 Bone 혹은 SocketName을 통해 어디에 볼일지 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 RenderWorld의 FScene에 변경 내용을 전달한다 (혹은 생성한다)
            PartComponent->RegisterComponent();

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

    return bCreatedAnyActor;
}

```

```

/** 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);
    1 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()

```

    /**
     * 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; 1

        FHakCharacterPartHandle AddCharacterPart(const FHakCharacterPart& NewPart);

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

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

```

```

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()

```

    /**
     * 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;
1        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;
    };

```

```

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:

```

/** 
 * 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; ①
    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;
};

```

```

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()

```


/** 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; 1

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

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

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


```

```


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

    // Entries를 순회하며,
    for (const FHakAppliedCharacterPartEntry& Entry : Entries)
    {
        // Part Actor가 생성되어 SpawnerComponent에 개설되어 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()

```

USTRUCT(BlueprintType)
struct FHakAnimBodyStyleSelectionSet
{
    GENERATED_BODY()

    /** GameplayTag를 통해 (CosmeticTags), Mesh Rules에 따라 알맞은 BodyStyle을 반환한다 */
    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;
};

```

1

```

#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:

- 당연히 구현은 Add가 있으면 Remove가 있어야 한다!

HakPawnComponent_CharacterParts::RemoveCharacterPart

```


    /**
     * 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); 1
    /** 인스턴스화 된 Character Parts */
    UPROPERTY()
    FHakCharacterPartList CharacterPartList;
    /** 애니메이션 적용을 위한 메시와 연결고리 */
    UPROPERTY(EditAnywhere, Category=Cosmetics)
    FHakAnimBodyStyleSelectionSet BodyMeshes;
    };


```

```


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


```

HakCharacterPartList::RemoveEntry()

```


/** 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); 1
    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;
};


```

```

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()

```

/** 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); 1

    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;
};

```

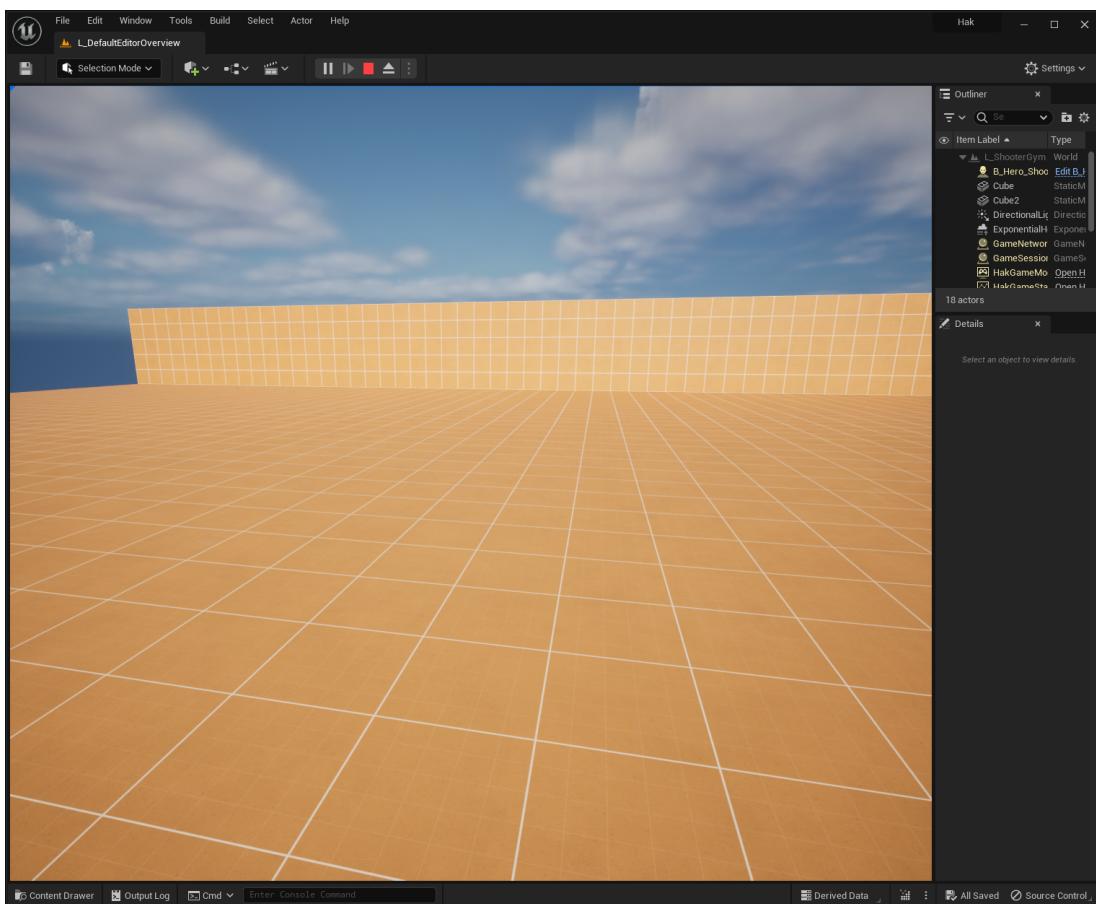
```

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

```

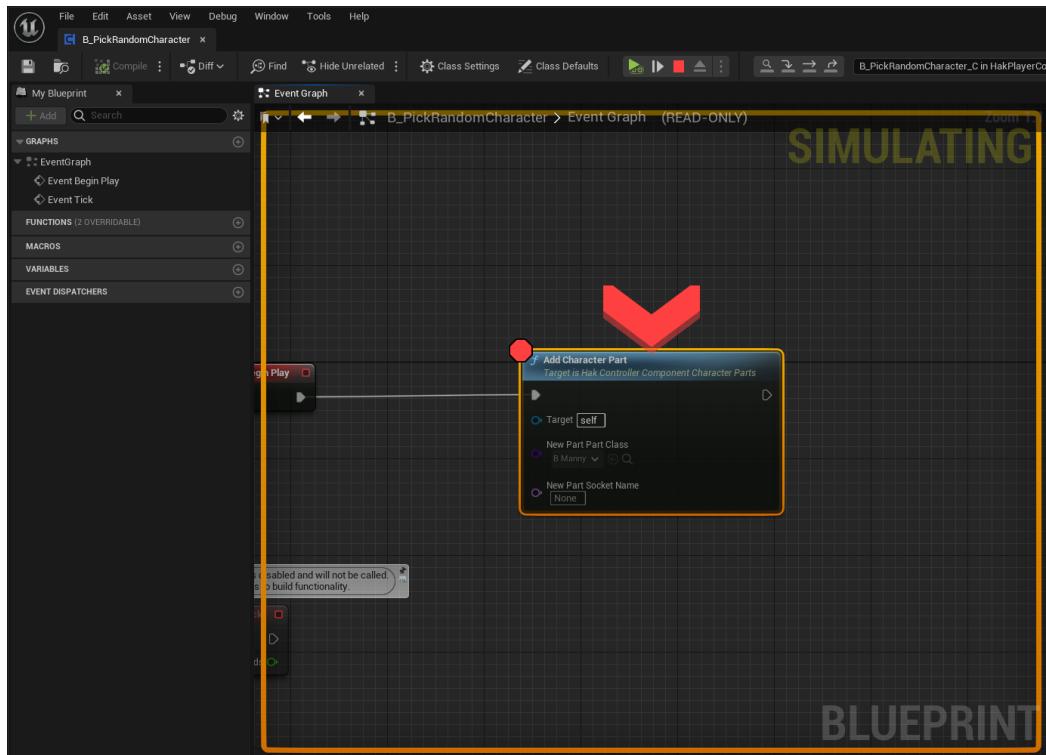
- ControllerComponent_CharacterParts는 유저가 조정하는 Pawn이 변경되었을 경우, Character Part를 유지하고 싶다면 재적용을 위해 Pawn에 있는 Parts를 다시 재적용한다:
 - 반대로, Character Part를 다 없애고 싶다면, ControllerComponent_CharacterParts의 상태도 비워줘야한다
 - 이 경우, Remove가 필요하다, 이거는 여러분들에게 맡겨보도록 할 것이다 (후일 구현할지도?)

□ 흠.. 아무 변화가 없다...



□ 우선 앞서, B_PickRandomCharacter에 AddCharacterPart에 넣었다:

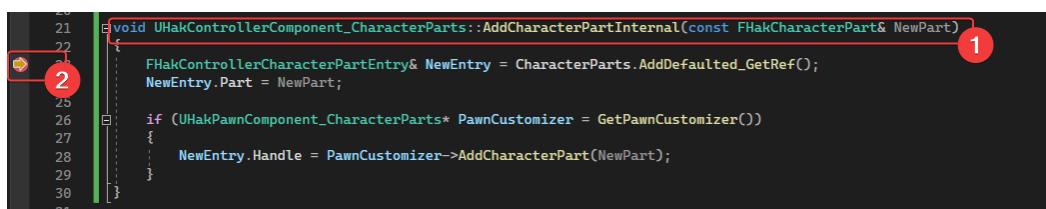
- 여기에 Breakpoint를 넣어서 잡히는지 확인해보자:



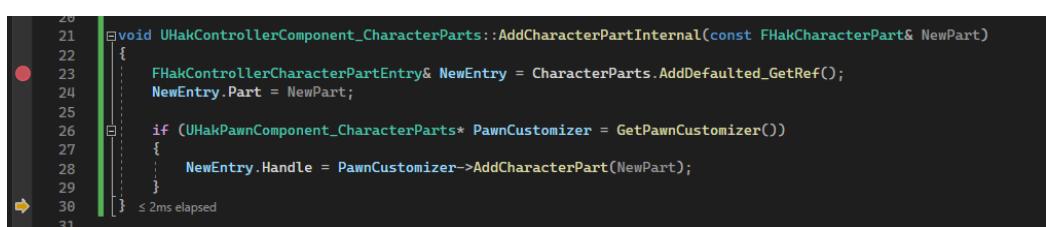
- 잡한다! → 이는 B_PickRandomCharacter가 HakPlayerController에 잘 이식되어 BeginPlay가 불렸다는 것이다 (GameFeatureAction은 잘 작동했다는 것을 의미한다)

□ 무슨 상황이 벌어지고 있는지 ControllerComponent_CharacterParts의 AddCharacterPart()를 디버깅해보자:

- 아래와 같이 Debug Breakpoint를 넣어주자:



- 임? 넘어간다!



- 그럼 GetPawnCustomizer가 실패했다는 것을 의미한다:

- Breakpoint가 안 잡히니 아래와 같이, PRAGMA_DISABLE_OPTIMIZATION을 활용하자:

```

10  PRAGMA_DISABLE_OPTIMIZATION
11  UHakPawnComponent_CharacterParts* UHakControllerComponent_CharacterParts::GetPawnCustomizer() const
12  {
13      if (APawn* ControlledPawn = GetPawn<APawn>())
14      {
15          // 생각해보면, 우리는 앞서 HakPawnComponent_CharacterParts를 상속받는 B_MannequinPawnCosmetics를 이미 B_Hero_ShooterMannequin에 추가하였다.
16          // B_MannequinPawnCosmetics를 반환되길 기대한다
17          return ControlledPawn->FindComponentByClass<UHakPawnComponent_CharacterParts>();
18      }
19      return nullptr;
20  }
21  PRAGMA_ENABLE_OPTIMIZATION

```

- Controller에 Possess가 된 Pawn이 없다는 것이다:

```

10  PRAGMA_DISABLE_OPTIMIZATION
11  UHakPawnComponent_CharacterParts* UHakControllerComponent_CharacterParts::GetPawnCustomizer() const
12  {
13      if (APawn* ControlledPawn = GetPawn<APawn>())
14      {
15          // 생각해보면, 우리는 앞서 HakPawnComponent_CharacterParts를 상속받는 B_MannequinPawnCosmetics를 이미 B_Hero_ShooterMannequin에 추가하였다.
16          // B_MannequinPawnCosmetics를 반환되길 기대한다
17          return ControlledPawn->FindComponentByClass<UHakPawnComponent_CharacterParts>();
18      }
19      return nullptr; // ≤ 5ms elapsed
20  }
21  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:

```

/* I */
/* Controller가 Pawn을 Possess했을 때, Pawn에 어떤 Cosmetic Actor 생성할지 결정하는 ControllerComponent:
 * - 필자는 캐릭터 파츠를 유지 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); // 1
    UPROPERTY(EditAnywhere, Category=Cosmetics)
    TArray<FHakControllerCharacterPartEntry> CharacterParts;
};

```

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

□ AController::OnPossessedPawnChanged의 Delegate로 연결하기:

- BeginPlay()와 EndPlay()을 쌍을 맞추어 정의해주자:

```
/**  
 * Controller가 Pawn을 Possess했을 때, Pawn에 어떤 Cosmetic Actor 생성할지 결정하는 ControllerComponent:  
 * - 필자는 캐릭터 피즈를 뮤저 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; 1  
    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;  
};
```

```
void UHakControllerComponent_CharacterParts::BeginPlay()  
{  
    Super::BeginPlay();  
  
    if (HasAuthority())  
    {  
        if (AController* OwningController = GetController<AController>())  
        {  
            OwningController->OnPossessedPawnChanged.AddDynamic(this, &ThisClass::OnPossessedPawnChanged); 1  
            // 우리는 Controller에 우리의 OnPossessedPawnChanged를 Delegate에 연결해주었다  
        }  
    }  
  
    void UHakControllerComponent_CharacterParts::EndPlay(const EEndPlayReason::Type EndPlayReason)  
    {  
        RemoveAllCharacterParts();  
        Super::EndPlay(EndPlayReason);  
    }  
}
```

□ RemoveAllCharacterParts()

```


    /**
     * 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(); 1
        /* UFUNCTION으로 정의하지 않으면 AddDynamic이 되지 않는다! */
        UFUNCTION()
        void OnPossessedPawnChanged(APawn* OldPawn, APawn* NewPawn);

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


```

```


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


```

□ OnPossessedPawnChanged 구현:

```


void UHakControllerComponent_CharacterParts::OnPossessedPawnChanged(APawn* OldPawn, APawn* NewPawn)
{
    // 이전 OldPawn에 대해서는 Character Parts를 전부 제거해준다
    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 주기해준다
    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?

```

    FHakCharacterPartHandle FHakCharacterPartList::AddEntry(FHakCharacterPart NewPart)
    {
        // PawnComponent의 CharacterPartList가 PartHandle을 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.Add();
            NewEntry.Part = NewPart;
            NewEntry.PartHandle = Result.PartHandle;

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

        return Result;
    }

```

- 아래와 같이 우리가 OwnerComponent를 설정하지 않았음:

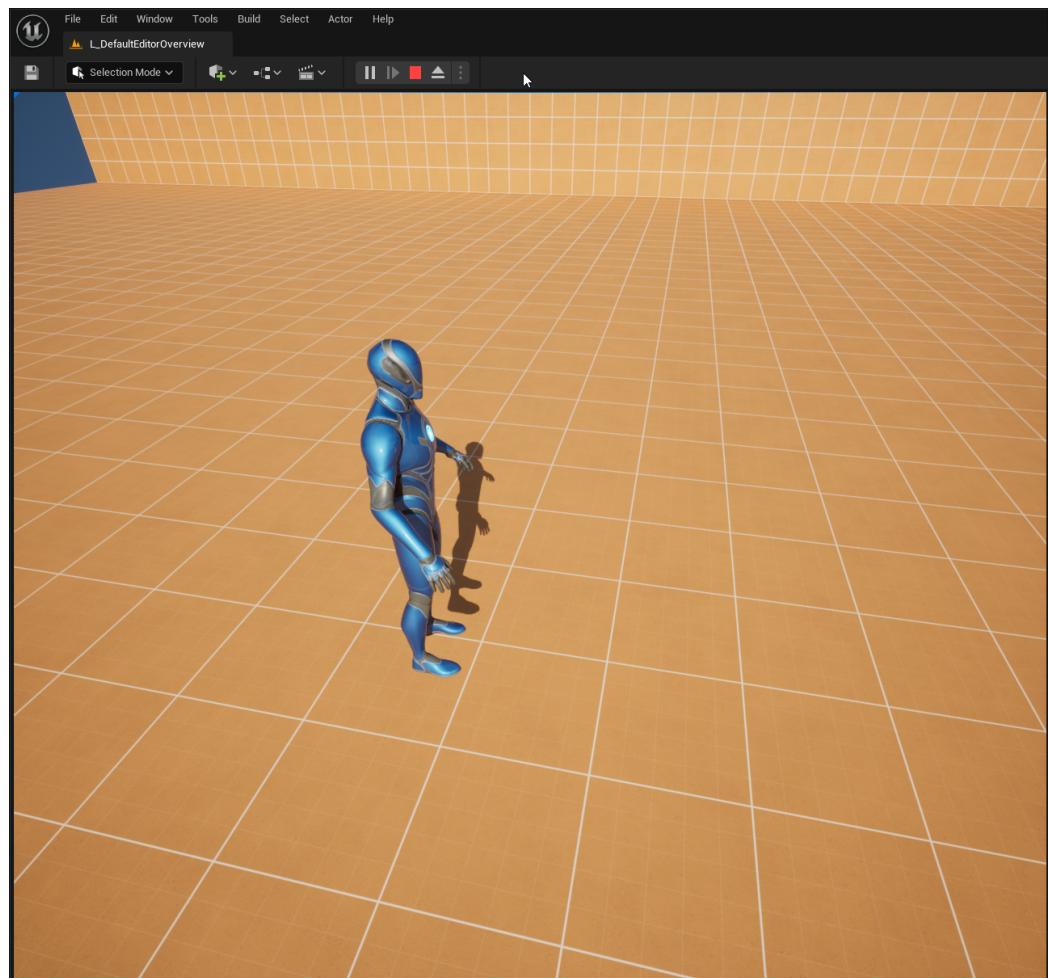
```

UHakPawnComponent_CharacterParts::UHakPawnComponent_CharacterParts(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
    , CharacterPartList(this) ①
{
}

```

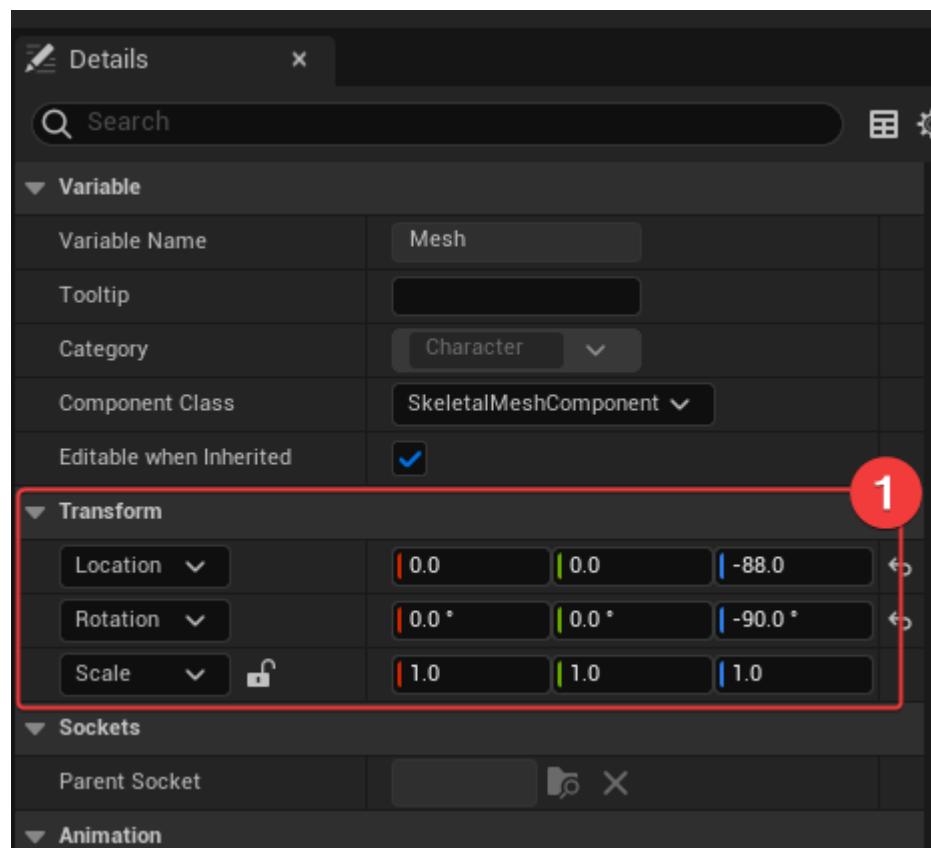
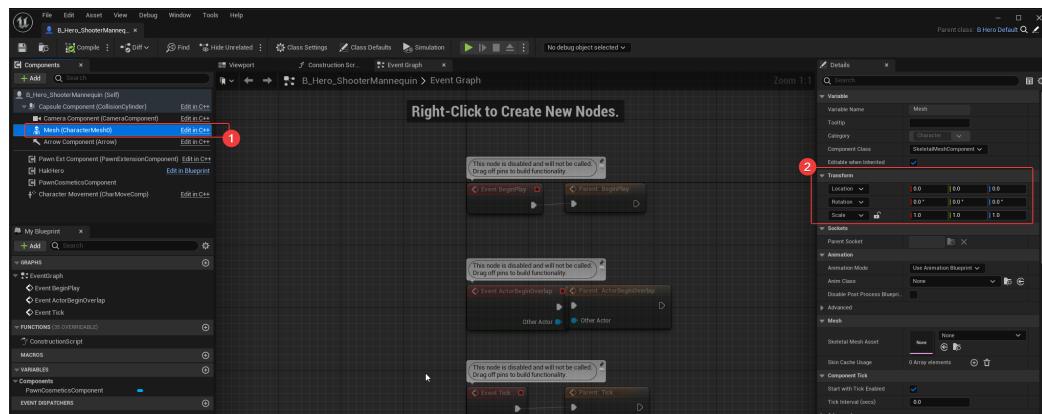
- 아래와 같이 드디어 캐릭터가 나왔다:

- 그러나 아직 뭔가 캐릭터 Transform이 이상한거 같다...

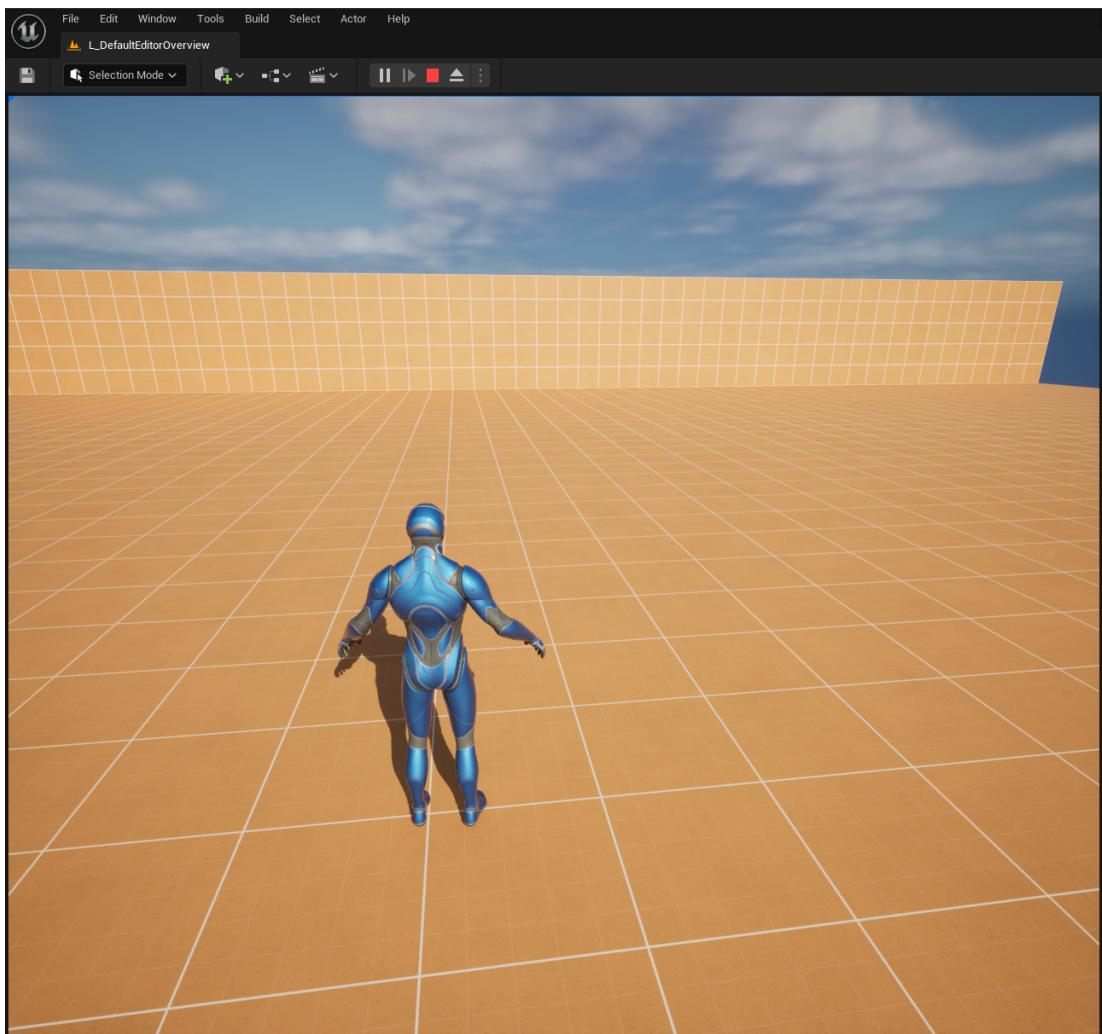


□ B_Hero_ShooterMannequin을 확인해보자:

- 아래의 Mesh의 Transform을 수정하자:



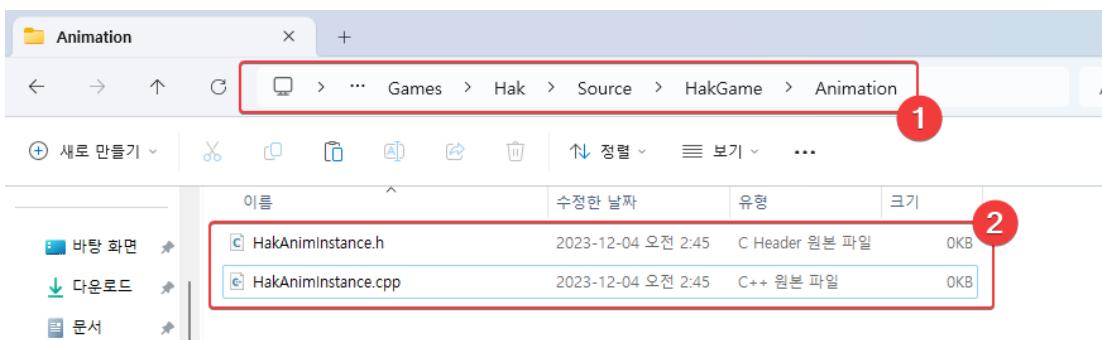
□ 이제 우리가 의도한 결과대로 나오는거 같다:



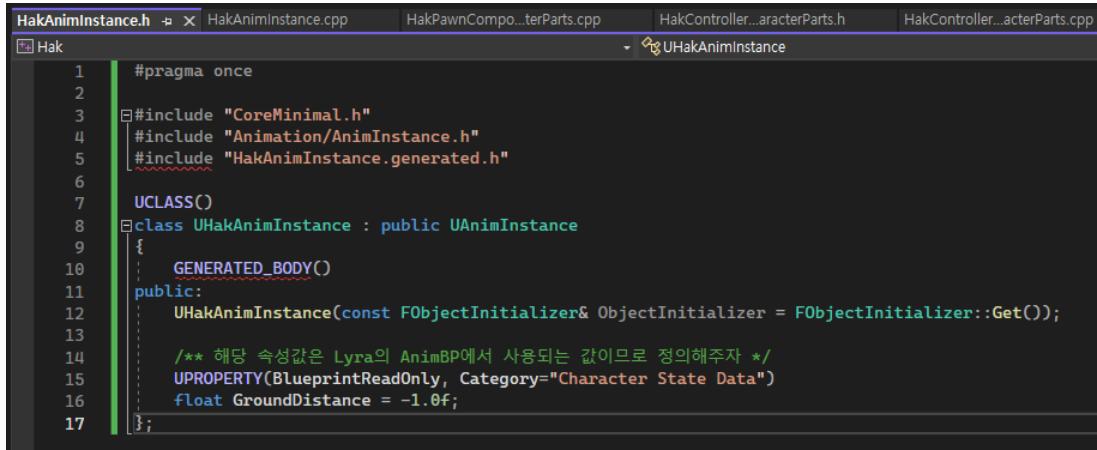
HakAnimInstance

▼ 펼치기

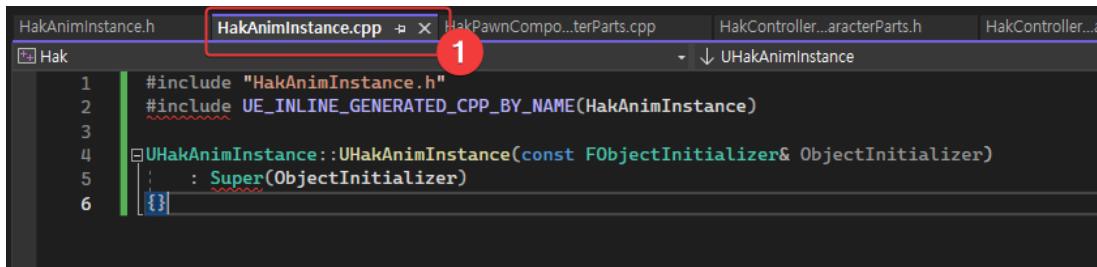
- Lyra의 Animation을 Migrate를 시키기 위해서, 우리는 AnimInstance를 정의하여 Override 할 필요성이 있다.
- HakAnimInstance.h/.cpp 생성:



□ HakAnimInstance:



```
HakAnimInstance.h  HakAnimInstance.cpp  HakPawnCompo...terParts.cpp  HakController...aracterParts.h  HakController...acterParts.cpp
[Hak] UHakAnimInstance
1  #pragma once
2
3  #include "CoreMinimal.h"
4  #include "Animation/AnimInstance.h"
5  #include "HakAnimInstance.generated.h"
6
7  UCLASS()
8  class UHakAnimInstance : public UAnimInstance
9  {
10  public:
11     GENERATED_BODY()
12     UHakAnimInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
13
14     /** 해당 속성값은 Lyra의 AnimBP에서 사용되는 값으로 정의해주자 */
15     UPROPERTY(BlueprintReadOnly, Category="Character State Data")
16     float GroundDistance = -1.0f;
17 }
```



```
HakAnimInstance.h  HakAnimInstance.cpp  HakPawnCompo...terParts.cpp  HakController...aracterParts.h  HakController...acterParts.cpp
[Hak] UHakAnimInstance
1  #include "HakAnimInstance.h"
2  #include UE_INLINE_GENERATED_CPP_BY_NAME(HakAnimInstance)
3
4  UHakAnimInstance::UHakAnimInstance(const FObjectInitializer& ObjectInitializer)
5      : Super(ObjectInitializer)
6  { }
```

□ LyraAnimInstance를 HakAnimInstance로 Class Redirect를 해주자:

□ DefaultEngine.ini 파일을 열어주자:

- 당연히 저의 경우, Hak 프로젝트의 DefaultEngine.ini일 것이다. 여러분들도 여러분에 알맞는 프로젝트의 DefaultEngine.ini를 수정하자:

1

```

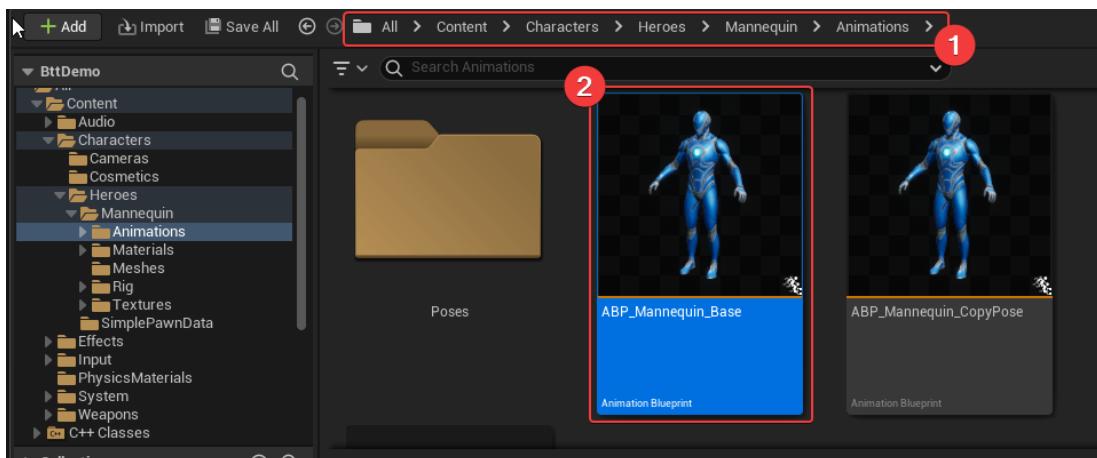
1 [/Script/EngineSettings.GameMapsSettings]
2 GameDefaultMap=/Engine/Maps/Templates/OpenWorld
3 EditorStartupMap=/Game/System/DefaultEditorMap/L_DefaultEditorOverview.L_DefaultEditorOverview
4 GlobalDefaultGameMode=/Script/HakGame.HakGameMode
5 GameInstanceClass=/Script/HakGame.HakGameInstance
6
7 [/Script/WindowsTargetPlatform.WindowsTargetSettings]
8 DefaultGraphicsRHIDefaultGraphicsRHI=D3D12
9 -D3D12TargetedShaderFormats=PCD3D_SMS
10 +D3D12TargetedShaderFormats=PCD3D_SM6
11 -D3D11TargetedShaderFormats=PCD3D_SMS
12 +D3D11TargetedShaderFormats=PCD3D_SMS
13
14 CompilerDefault
15 AudioSampleRate=48000
16 AudioCallbackBufferSize=1024
17 AudioNumBuffersToEnqueue=1
18 AudioMaxChannels=8
19 AudioNumSourceWorkers=4
20 SpatializationPlugin=
21 SourceDataOverridePlugin=
22 ReverbPlugin=
23 OcclusionPlugin=
24 CompressionOverrides=(bOverrideCompressionTimes=False, DurationThreshold=5.000000, MaxNumRandomBranches=0, SoundCueQualityIndex=0)
25 CacheSizekB=65536
26 MaxChunkSizeOverridekB=0
27 bResampleForDevice=False
28 MaxSampleRate=48000.000000
29 HighSampleRate=32000.000000
30 MedSampleRate=24000.000000
31 LowSampleRate=12000.000000
32 MinSampleRate=8000.000000
33 CompressionQualityModifier=1.000000
34 AutoStreamingThreshold=0.000000
35 SoundCueCookQualityIndex=-1
36
37 [/Script/HardwareTargeting.HardwareTargetingSettings]
38 TargetedHardwareClass=Desktop
39 AppliedTargetedHardwareClass=Desktop
40 DefaultGraphicsPerformance=Maximum
41 AppliedDefaultGraphicsPerformance=Maximum
42
43 [/Script/Engine.RendererSettings]
44 r.GenerateMeshDistanceFields=True
45 r.DynamicGlobalIlluminationMethod=1
46 r.ReflectionMethod=1
47 r.Shadow.Virtual.Enable=1
48 r.DefaultFeature.AutoExposure.ExtendDefaultLuminanceRange=True
49
50 [/Script/WorldPartitionEditor.WorldPartitionEditorSettings]
51 CommandletClass='Script/UnrealEd.WorldPartitionConvertCommandlet'
52
53 [/Script/Engine.Engine]
54 +ActiveGameNameRedirects=(OldGameName="TP_Blank", NewGameName="/Script/Hak")
55 +ActiveGameNameRedirects=(OldGameName="/Script/TP_Blank", NewGameName="/Script/Hak")
56 +ActiveClassNameRedirects=(OldClassName="TP_BlankGameModeBase", NewClassName="HakGameModeBase")
57 AssetManagerClassName=/Script/HakGame.HakAssetManager
58
59 [/Script/AndroidFileServerEditor.AndroidFileServerRuntimeSettings]
60 bEnablePlugin=True
61 bAllowNetworkConnection=True
62 SecurityToken=3FDE1A264C0EBA42CA258FA84D1832F6
63 bIncludeInShipping=False
64 bAllowExternalStartInShipping=False
65 bCompileAPSPackage=False
66 bUseCompression=False
67 bLogFiles=False
68 bReportStats=False
69 ConnectionType=USBOnly
70 bUseManualIPAddress=False
71 ManualIPAddress=
72
73 [CoreRedirects]
74 +ClassRedirects=(OldName="LyraAnimInstance", NewName="/Script/HakGame.HakAnimInstance")
75

```

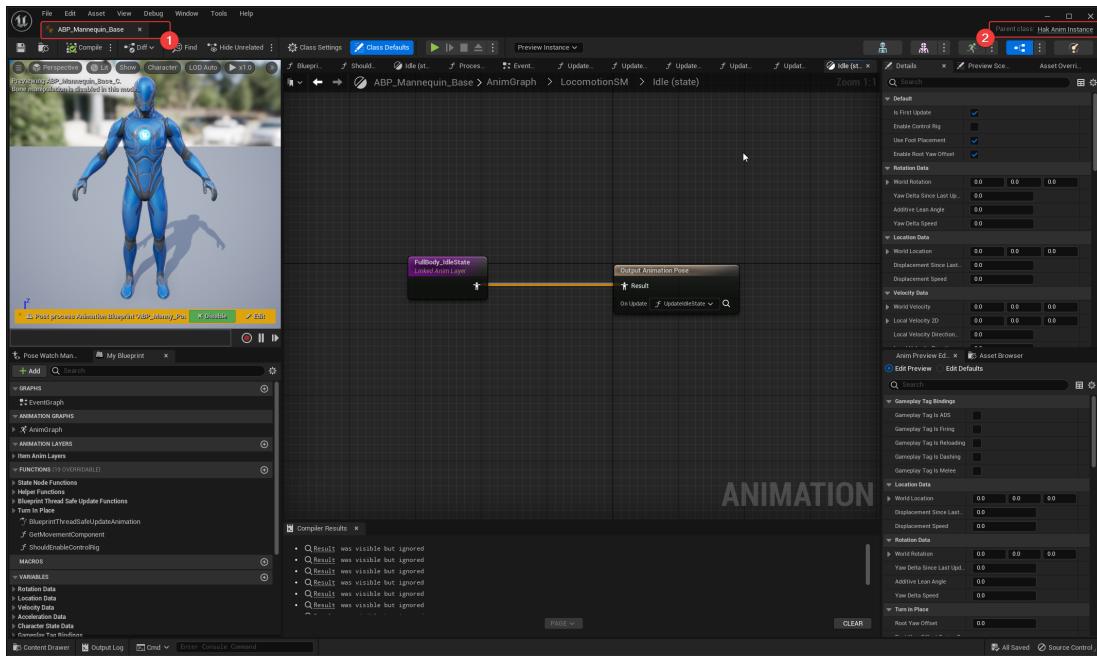
2 이부분을 통해 알맞는 DefaultEngine.ini를 찾았는지 확인할 수 있다

3 LyraAnimInstance → HakAnimInstance로 Redirect!!

□ 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) 1
{
    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) > B.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,"), *OldObjectName);
                        }
                    }
                    else
                    {
                        // Set value redirects for processing outside
                    }
                }
            }
        }
    }
}
```

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

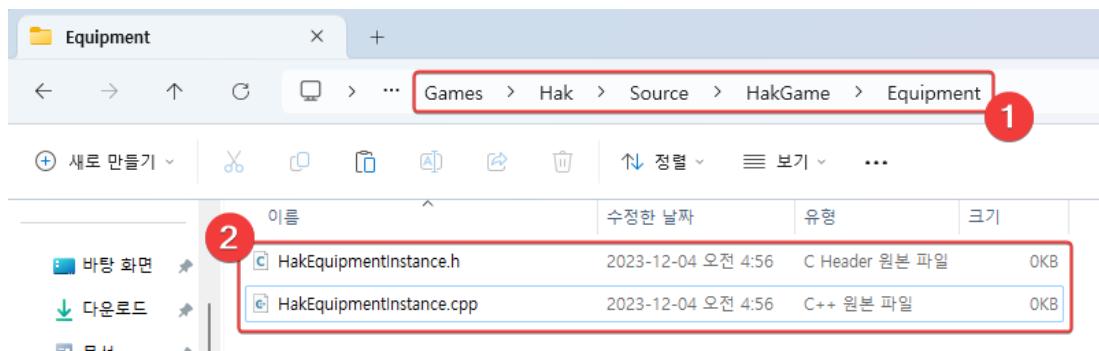
원리는 StaticConstructObject하는 과정에서 UClass 이름을 ResolveName하는 과정이 있는데 이 과정에서 Redirect를 진행하여, LyraAnimInstance는 HakAnimInstance로 변환하는 과정이 일어난다.

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:

```
#pragma once
#include "UObject/Object.h"
#include "HakEquipmentInstance.generated.h"

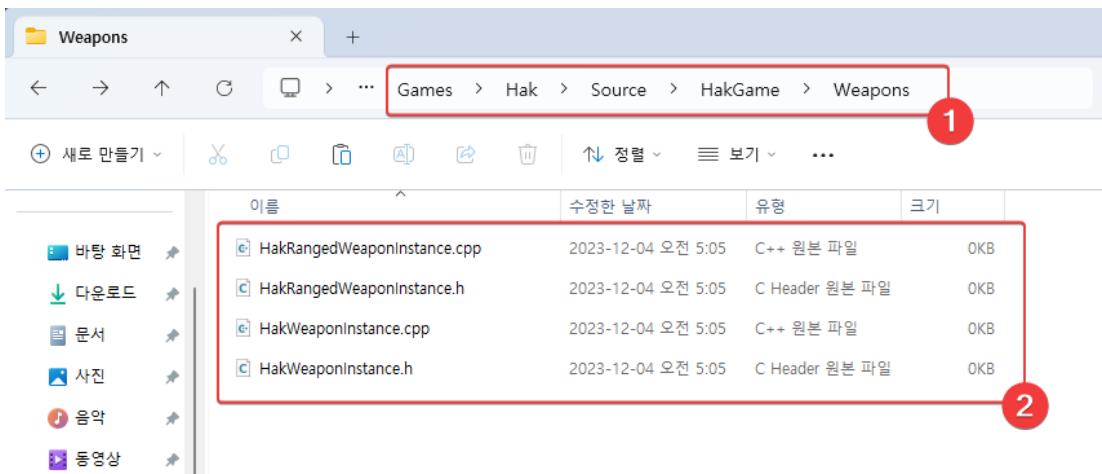
UCLASS(BlueprintType)
class UHakEquipmentInstance : public UObject
{
GENERATED_BODY()
public:
    UHakEquipmentInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
};
```

```

1 #include "HakEquipmentInstance.h"
2 #include UE_INLINE_GENERATED_CPP_BY_NAME(HakEquipmentInstance)
3
4 UHakEquipmentInstance::UHakEquipmentInstance(const FObjectInitializer& ObjectInitializer)
5     : Super(ObjectInitializer)
6

```

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



□ HakWeaponInstance 구현:

```

1 #pragma once
2
3 #ifndef HAK_WEAPON_INSTANCE_H_
4 #define HAK_WEAPON_INSTANCE_H_
5
6 UCLASS()
7 class UHakWeaponInstance : public UHakEquipmentInstance
8 {
9     GENERATED_BODY()
10 public:
11     UHakWeaponInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
12 }

```

```

1 #include "HakWeaponInstance.h"
2 #include UE_INLINE_GENERATED_CPP_BY_NAME(HakWeaponInstance)
3
4 UHakEquipmentInstance::UHakEquipmentInstance(const FObjectInitializer& ObjectInitializer)
5     : Super(ObjectInitializer)
6

```

□ HakRangedWeaponInstance 구현:

```

1 #pragma once
2
3 #include "HakWeaponInstance.h"
4 #include "HakRangedWeaponInstance.generated.h"
5
6 UCLASS()
7 class UHakRangedWeaponInstance : public UHakWeaponInstance
8 {
9     GENERATED_BODY()
10    public:
11        UHakRangedWeaponInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
12 }

```

```

1 #include "HakRangedWeaponInstance.h"
2 #include UE_INLINE_GENERATED_CPP_BY_NAME(HakRangedWeaponInstance)
3
4 UHakRangedWeaponInstance::UHakRangedWeaponInstance(const FObjectInitializer& ObjectInitializer)
5     : Super(ObjectInitializer)
6 {
}

```

□ HakWeaponInstance 멤버 변수 선언:

```

1 #pragma once
2
3 #include "HakGame/Equipment/HakEquipmentInstance.h"
4 #include "HakWeaponInstance.generated.h"
5
6 UCLASS()
7 class UHakWeaponInstance : public UHakEquipmentInstance
8 {
9     GENERATED_BODY()
10    public:
11        UHakWeaponInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
12
13        /** Weapon에 Equip/Unequip에 대한 Animation Set 정보를 들고 있다 */
14        UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Animation)
15        FHakAnimLayerSelectionSet EquippedAnimSet;
16
17        UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Animation)
18        FHakAnimLayerSelectionSet UnequippedAnimSet;
19 }

```

□ FHakAnimLayerSelectionSet:

```

1 USTRUCT(BlueprintType)
2 struct FHakAnimLayerSelectionSet
3 {
4     GENERATED_BODY()
5
6     /** 앞서 보았던 HakAnimBodyStyleSelection의 MeshRule과 같이 AnimInstance의 Rule을 가진 LayerRules로 생각하면 됨 */
7     UPROPERTY(EditAnywhere, BlueprintReadWrite)
8     TArray<FHakAnimLayerSelectionEntry> LayerRules;
9
10    /** 디폴트 Layer */
11    UPROPERTY(EditAnywhere, BlueprintReadWrite)
12    TSubclassOf<UAnimInstance> DefaultLayer;
13 }

```

□ FHakAnimLayerSelectionEntry:

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

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

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

□ 해더 추가:

```
#pragma once

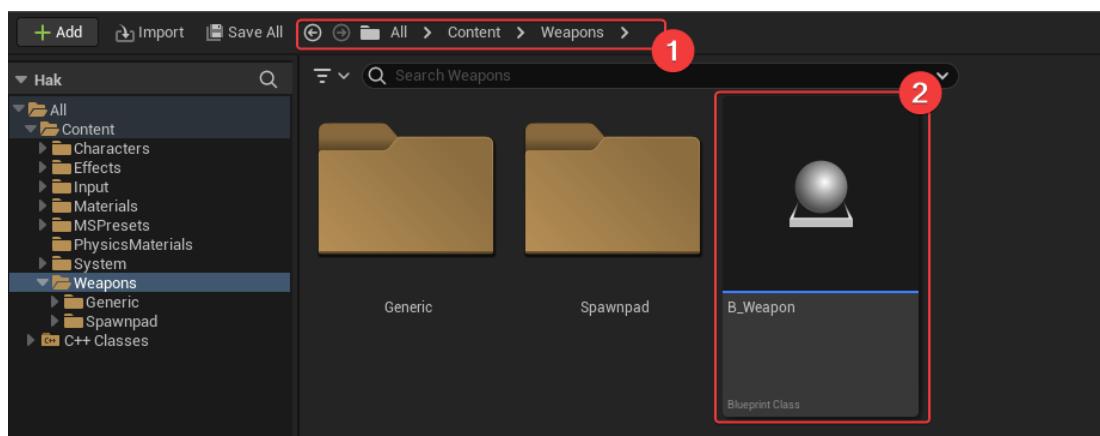
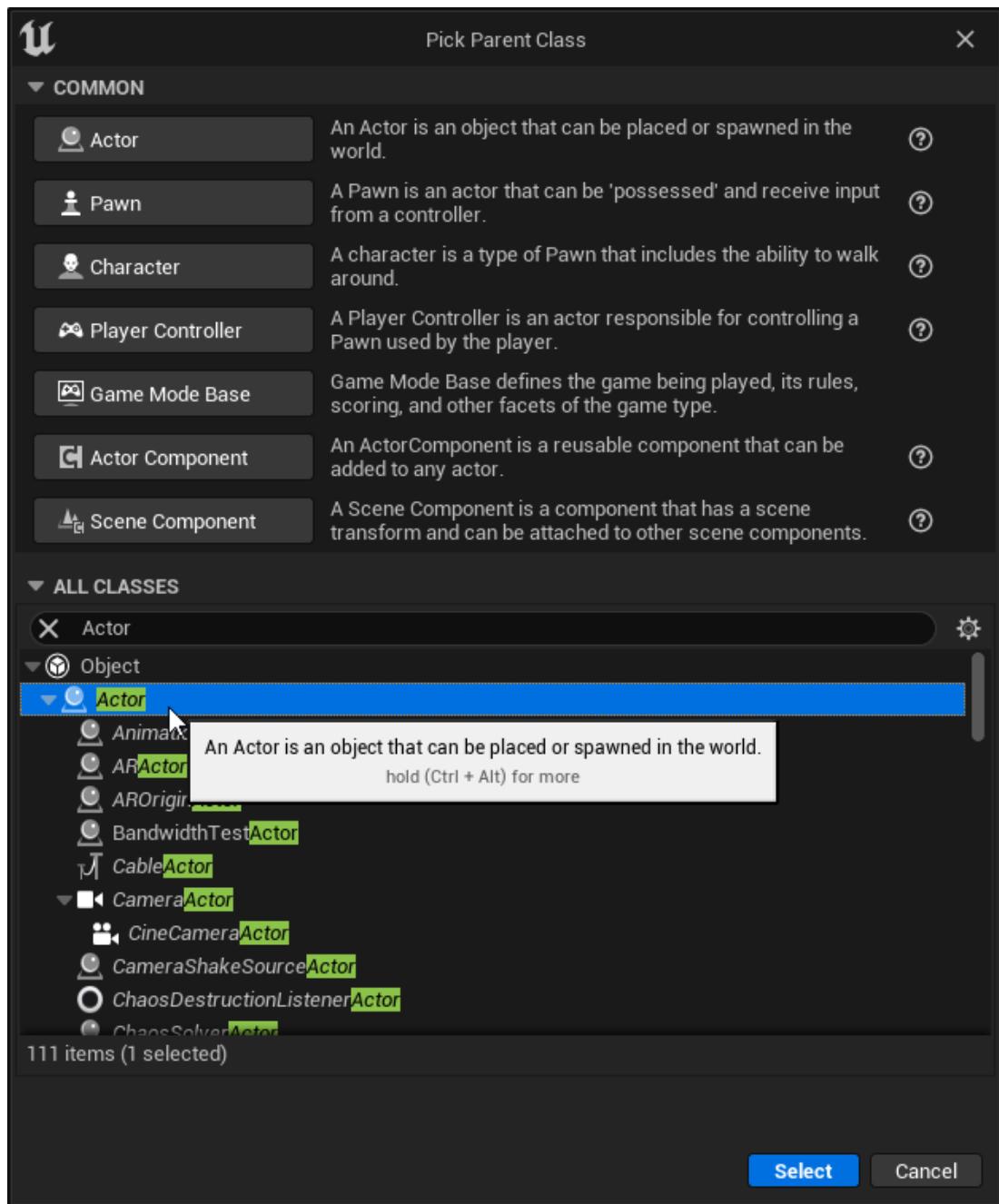
#include "HakGame/Equipment/HakEquipmentInstance.h"
#include "HakGame/Cosmetics/HakCosmeticAnimationTypes.h"#include "HakWeaponInstance.generated.h" 1

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

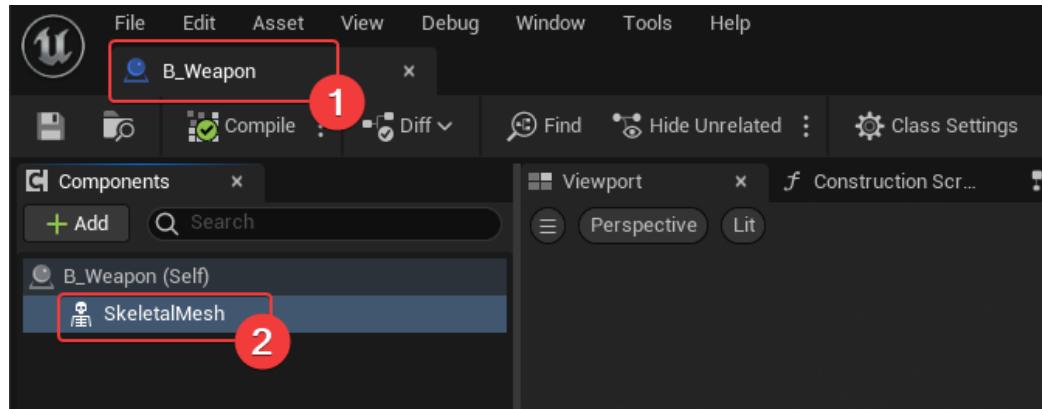
    /** Weapon이 Equip/Unequip에 대한 Animation Set 정보를 들고 있다 */
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Animation)
    FHakAnimLayerSelectionSet EquippedAnimSet;

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

□ B_Weapon 생성:

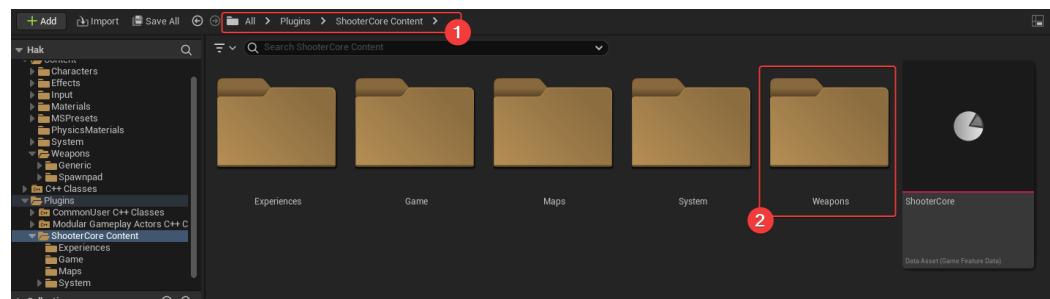


- SkeletalMesh 생성

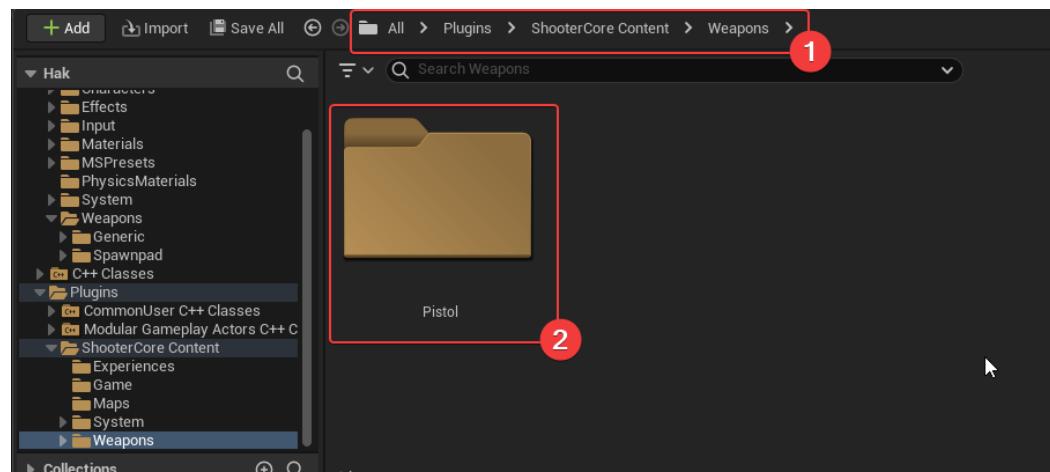


B_Pistol 생성:

ShooterCore/Weapons 생성:



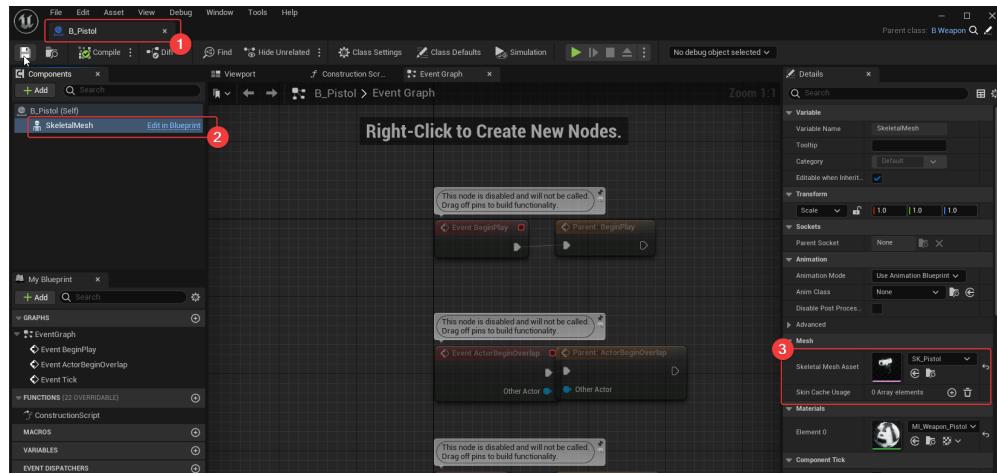
Pistol 폴더 생성:



B_Weapon을 상속 받는 B_Pistol 생성:

SK_Pistol Migrate:

SK_Pistol을 B_Pistol의 SkeletalMesh에 설정:



B_WeaponInstance_Base 생성:

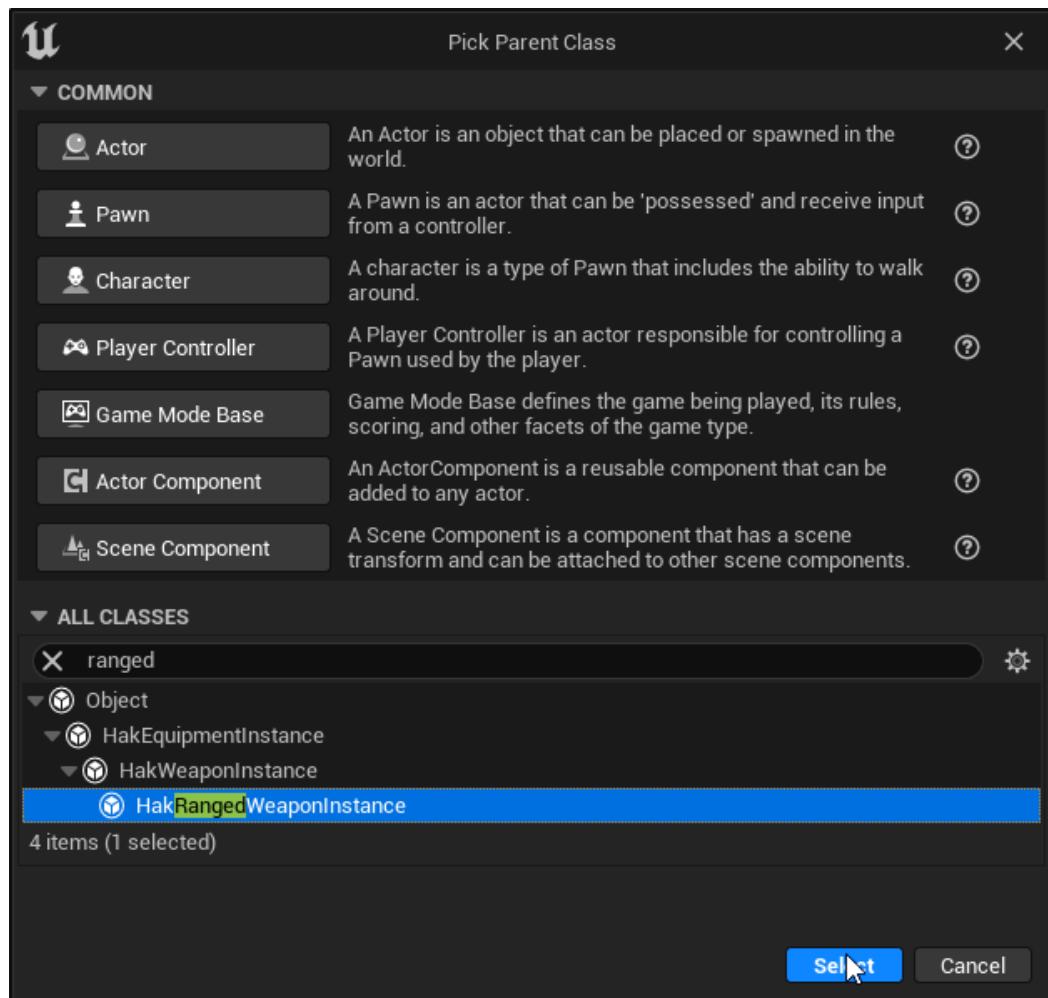
- HakEquipmentInstance에 PROPERTY 속성 추가해주자:

```
#pragma once

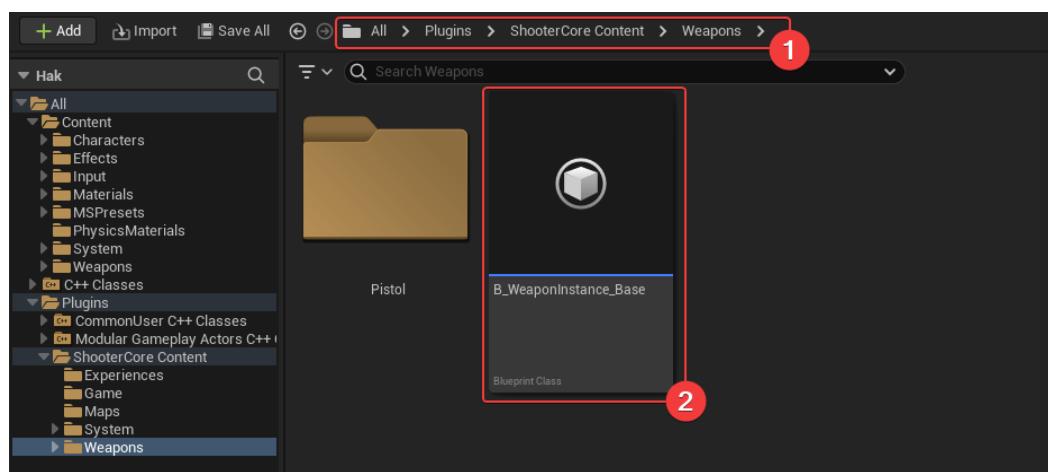
#include "UObject/Object.h"
#include "HakEquipmentInstance.generated.h" 1

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

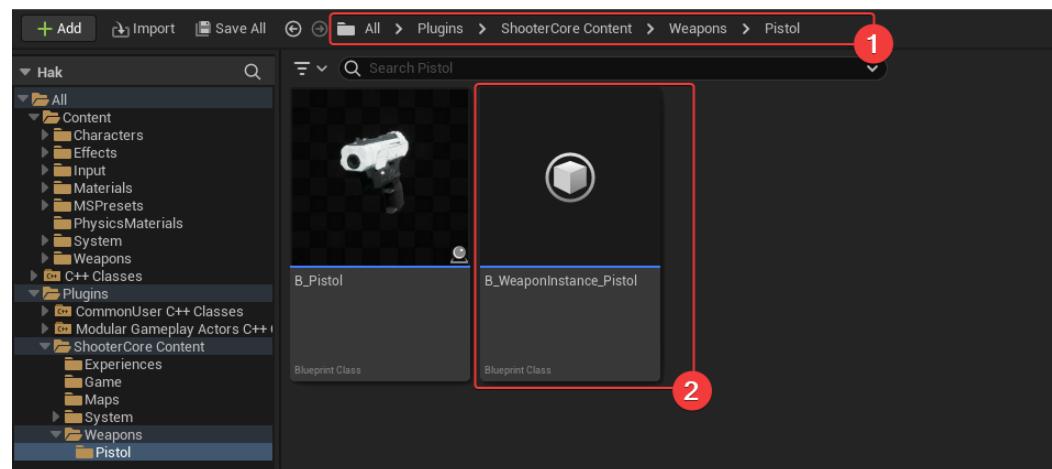
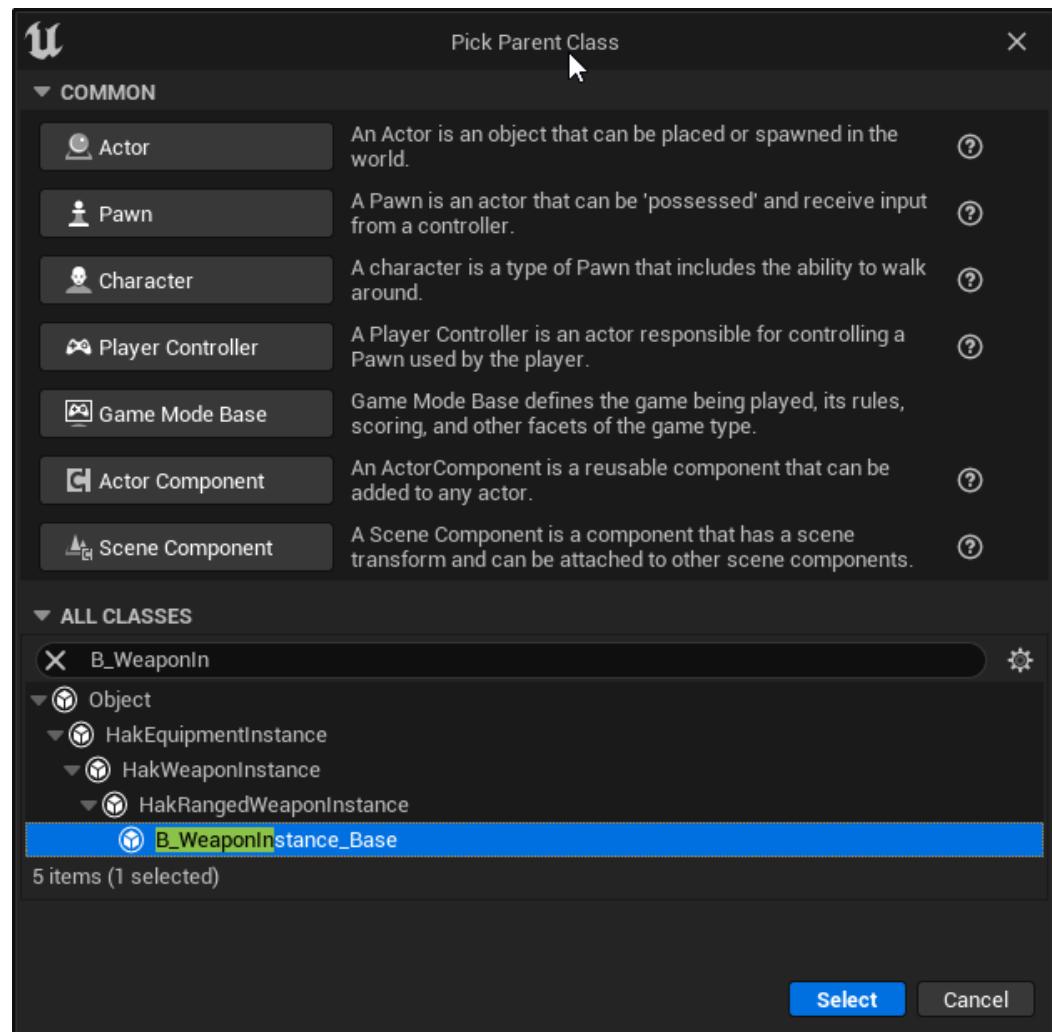
HakRangedWeaponInstance를 상속받자:



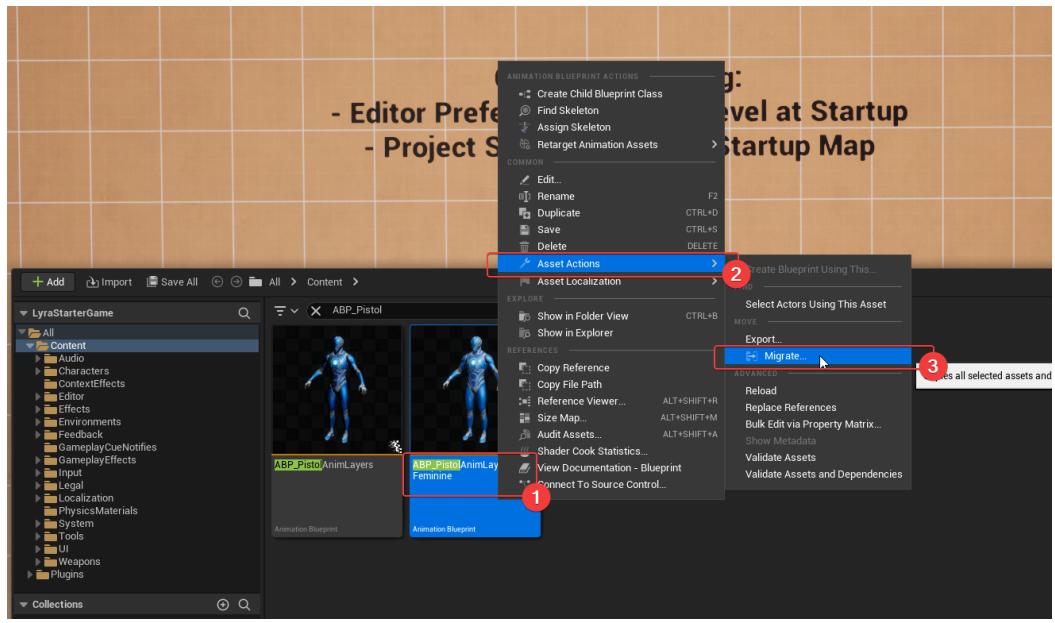
□ B_WeaponInstance_Base 생성:



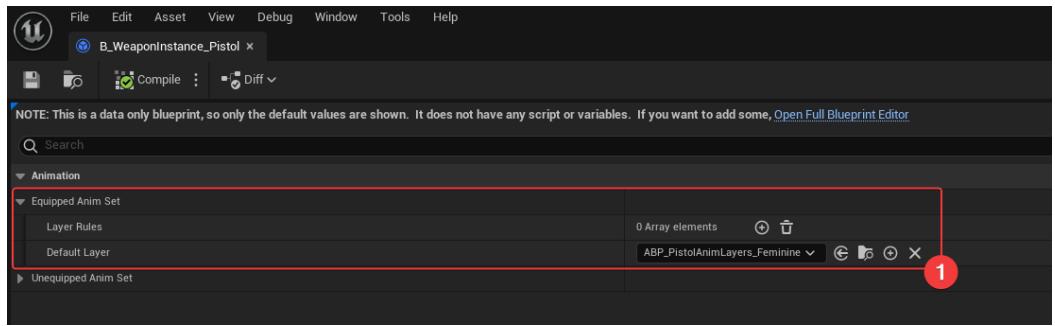
□ B_WeaponInstance_Pistol 생성:



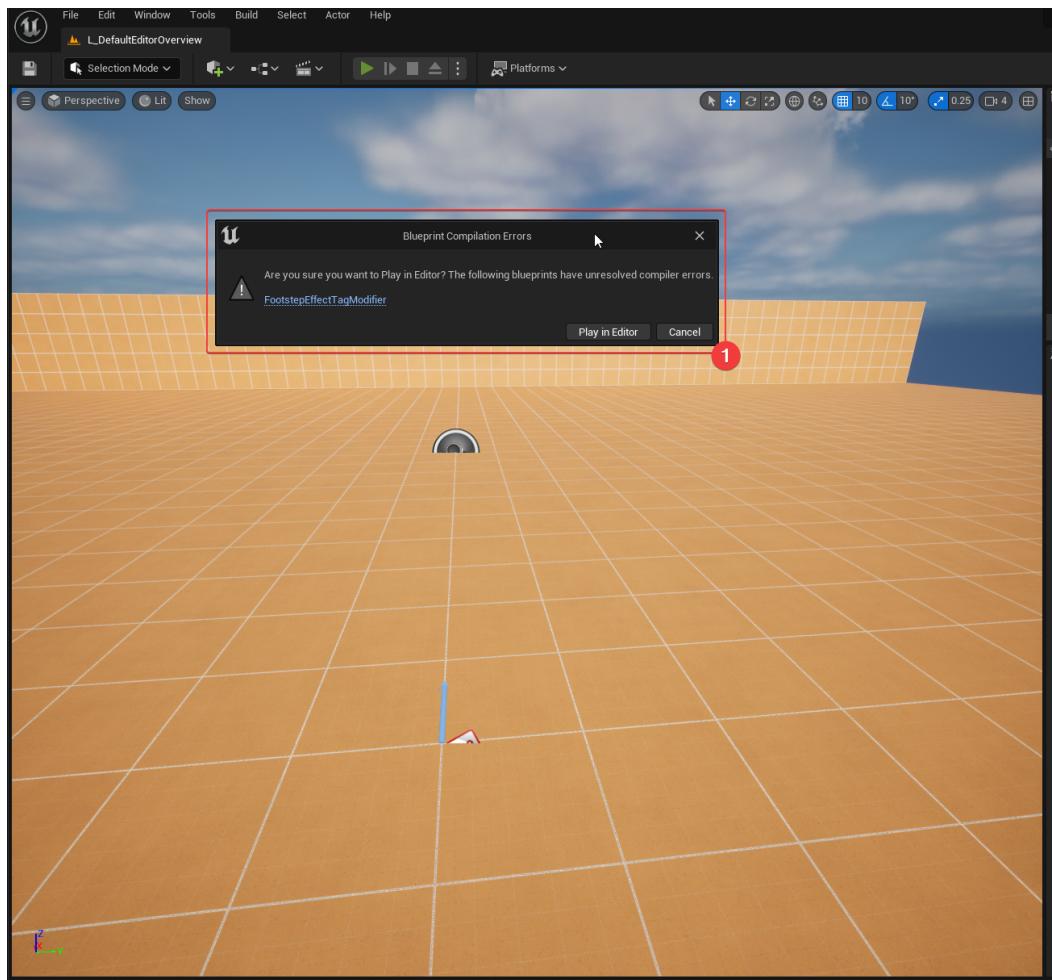
ABP_PistolAnimLayers_Feminine Migrate 해주자:



□ EquippedAnimSet에 설정해주자:



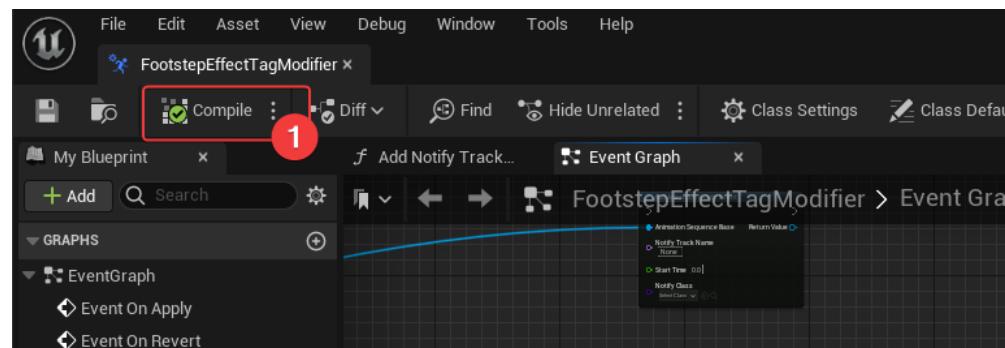
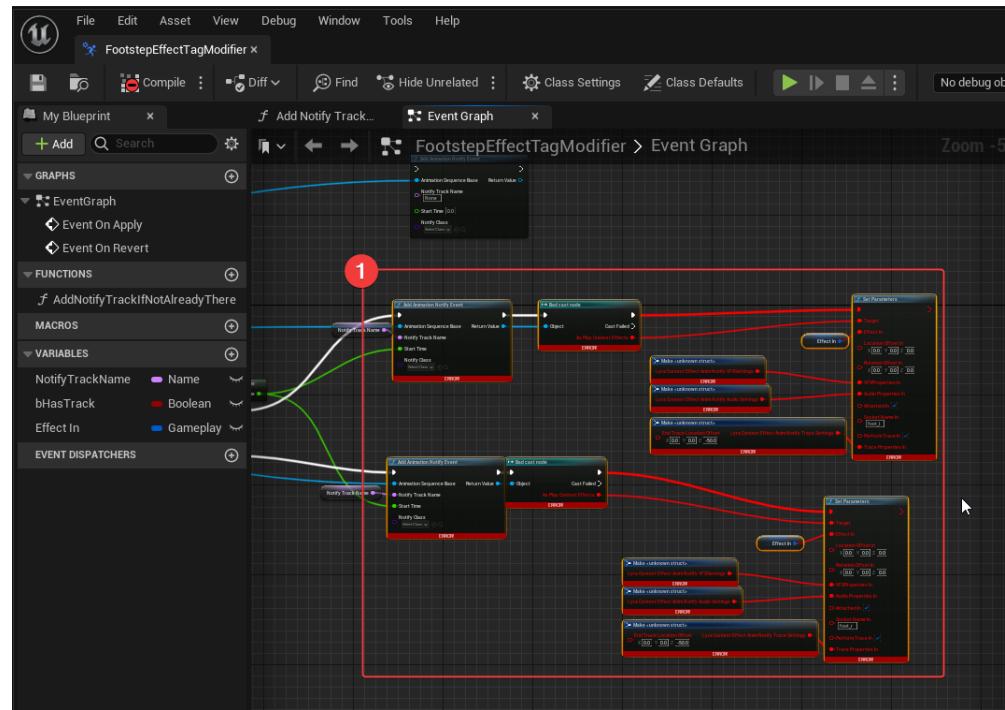
□ PIE 재생이 안된다:



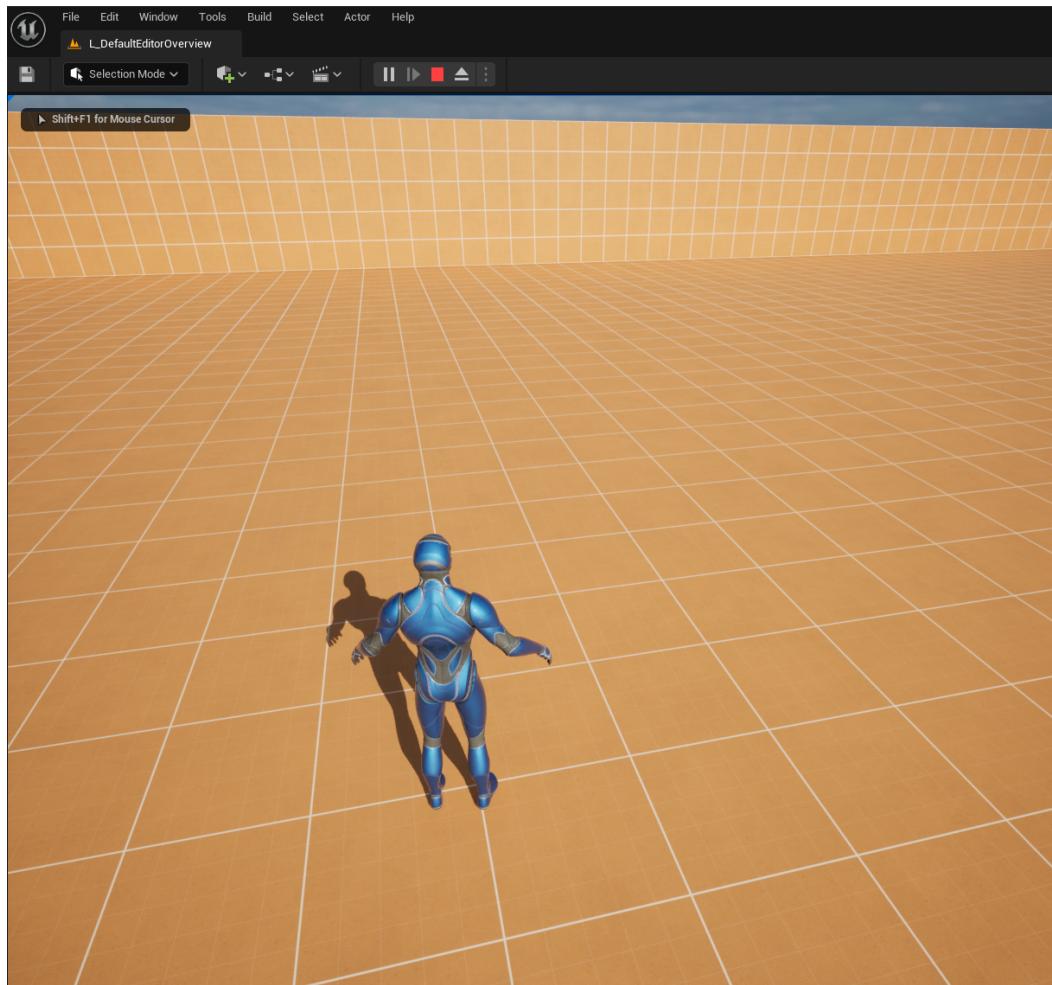
□ 다음과 같이 FootstepEffectTagModifier를 눌러주자:



□ 일단 그냥 지우자...



아직 아무 변화가 없다...



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