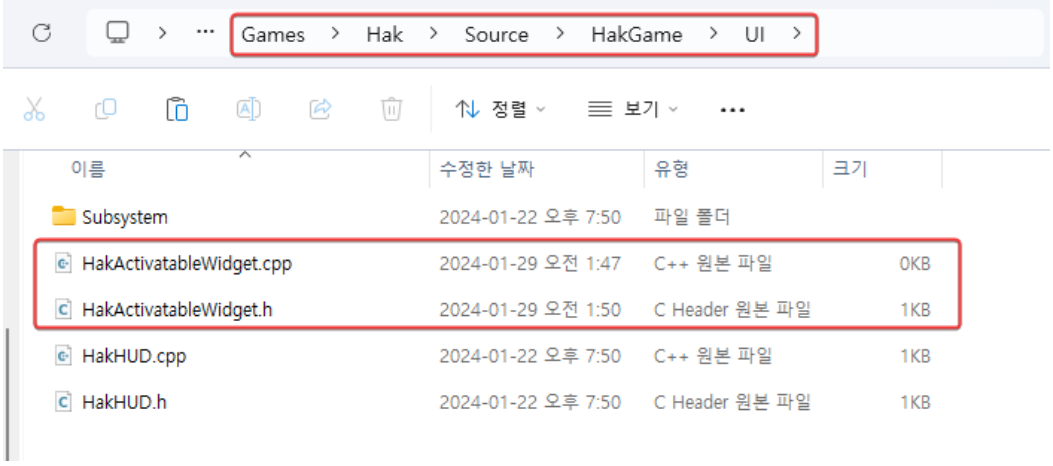
15주차 (2024.01.29)

W_ShooterHUDLayout

▼ **펼치기**

- [ ] HakActivatableWidget 구현:
  - [ ] HakActivatableWidget 파일 생성:

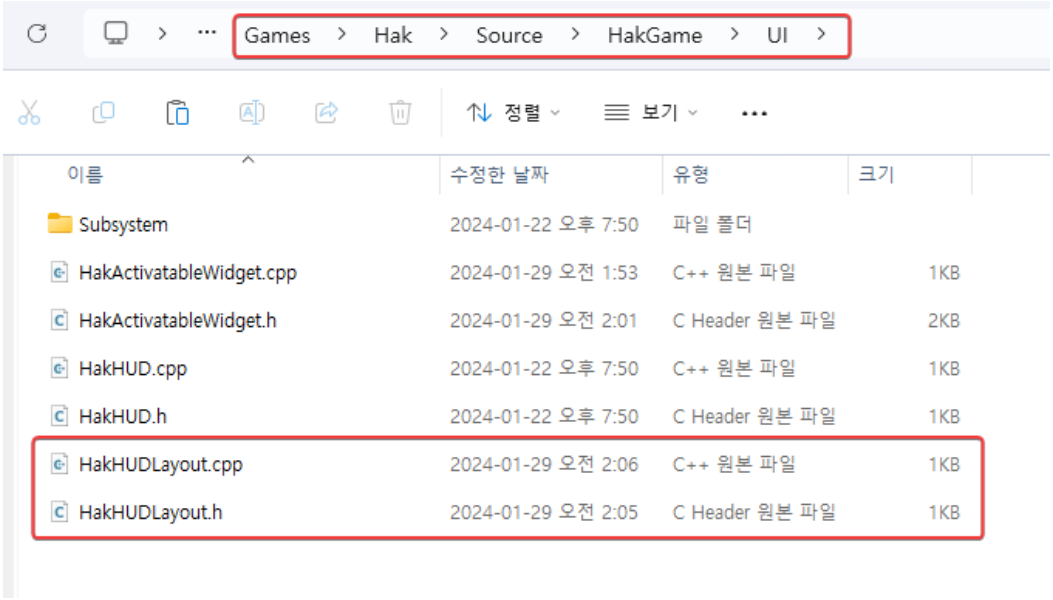


  - [ ] HakActivatableWidget.h/.cpp:

```cpp
#pragma once

#include "CoreMinimal.h"
#include "CommonActivatableWidget.h"
#include "HakActivatableWidget.generated.h"

/**
 * Input 처리 방식 정의
 */
UENUM(BlueprintType)
enum class EHakWidgetInputMode : uint8
{
    Default,
    GameAndMenu,
    Game,
    Menu,
};

/**
 * CommonActivatableWidget의 주석을 보면, 복잡하게 설명되어 있다.
 * 필자가 이해한 CommonActivatableWidget은 두가지 특성을 가지고 있다:
 * 1. Widget Layout과 Widget Instance을 구분하기 위한 용도로 CommonActivatableWidget은 Layout 정의:
 *    - 런타임 Activate/Deactivate
 *    - WidgetTree에서 제거가 아닌 껐다/켰다(== Activate/Deactivate)
 * 2. Input(Mouse or Keyboard etc...) 처리 방법 정의
 */
UCLASS(Abstract, Blueprintable)
class UHakActivatableWidget : public UCommonActivatableWidget
{
    GENERATED_BODY()
public:
    UHakActivatableWidget(const FObjectInitializer& ObjectInitializer);

    /** Input 처리 방식 */
    UPROPERTY(EditDefaultsOnly, Category=Input)
    EHakWidgetInputMode InputConfig = EHakWidgetInputMode::Default;

    /** Mouse 처리 방식 */
    UPROPERTY(EditDefaultsOnly, Category=Input)
    EMouseCaptureMode GameMouseCaptureMode = EMouseCaptureMode::CapturePermanently;
};
```

```cpp
#include "HakActivatableWidget.h"
#include "CommonInputModeTypes.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakActivatableWidget)

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

- [ ] HakHUDLayout 구현:
  - [ ] HakHUDLayout 파일 생성:

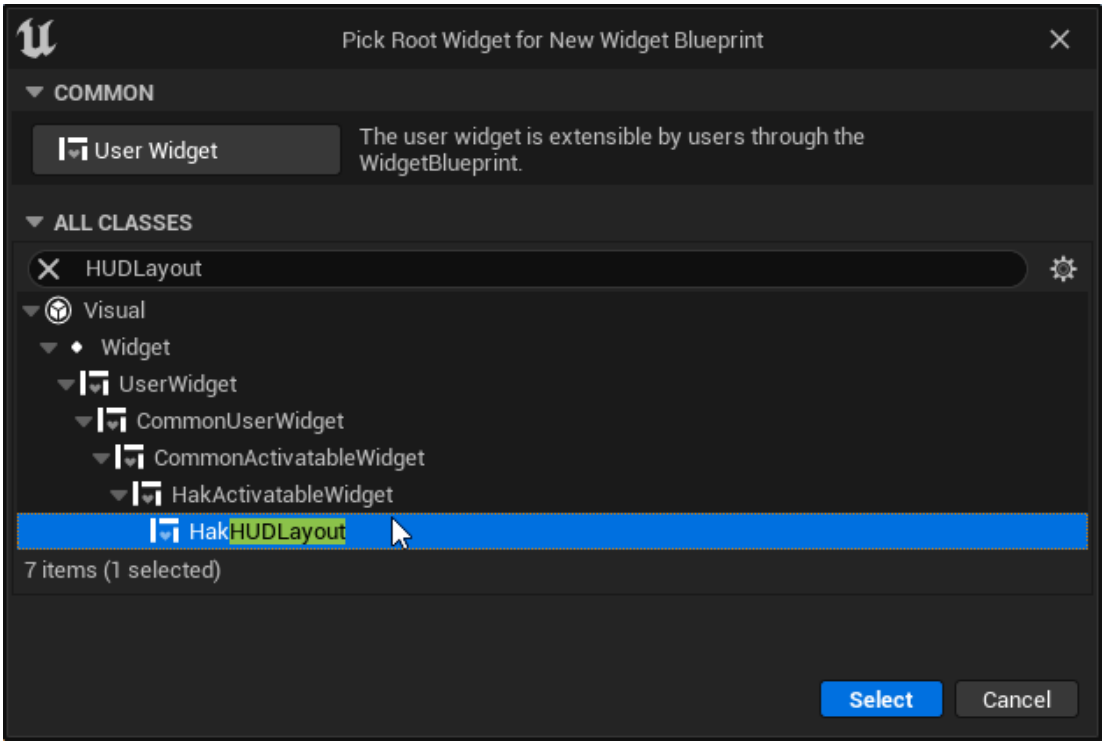


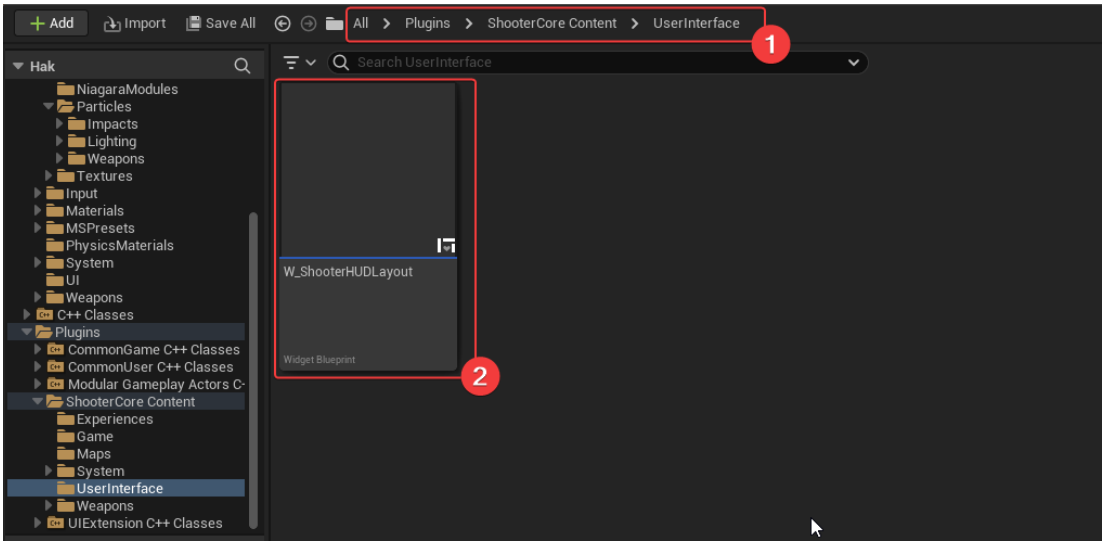
- [ ] HakHUDLayout.h/.cpp:

```h
#pragma once

#include "HakActivatableWidget.h"
#include "HakHUDLayout.generated.h"

/**
 * PrimaryGameLayout의 Layer에 연동할 HUD Layout (CommonActivatableWidget)
 */
UCLASS(Abstract, BlueprintType, Blueprintable, meta=(DisplayName="Hak HUD Layout", Category="Hak|HUD"))
class UHakHUDLayout : public UHakActivatableWidget
{
    GENERATED_BODY()
public:
    UHakHUDLayout(const FObjectInitializer& ObjectInitializer);
};
```

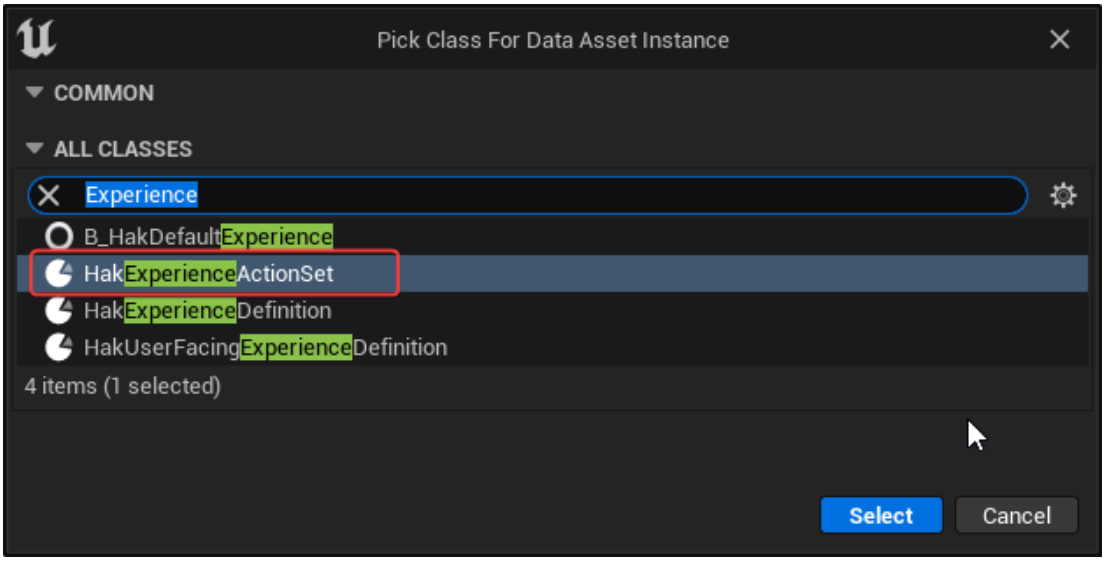
```cpp
#include "HakHUDLayout.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakHUDLayout)

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

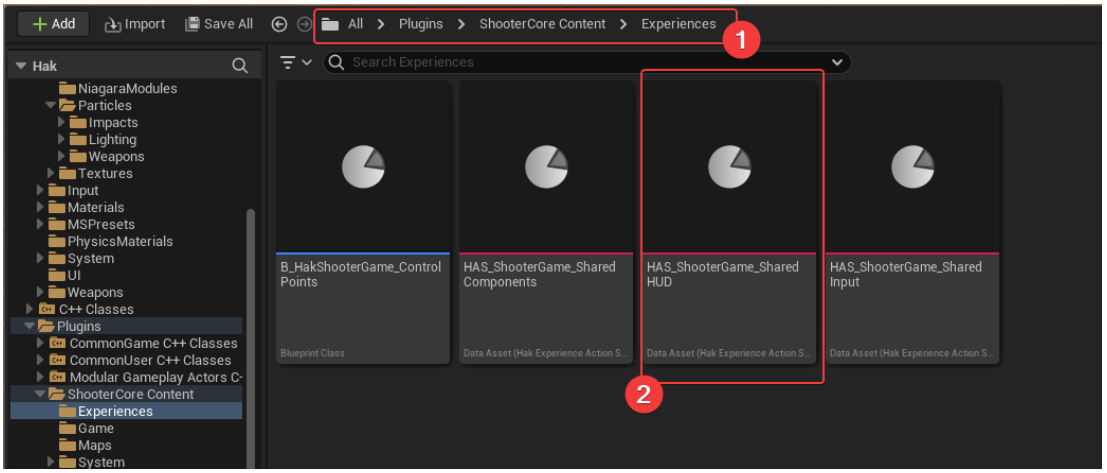
- [ ] W_ShooterHUDLayout:



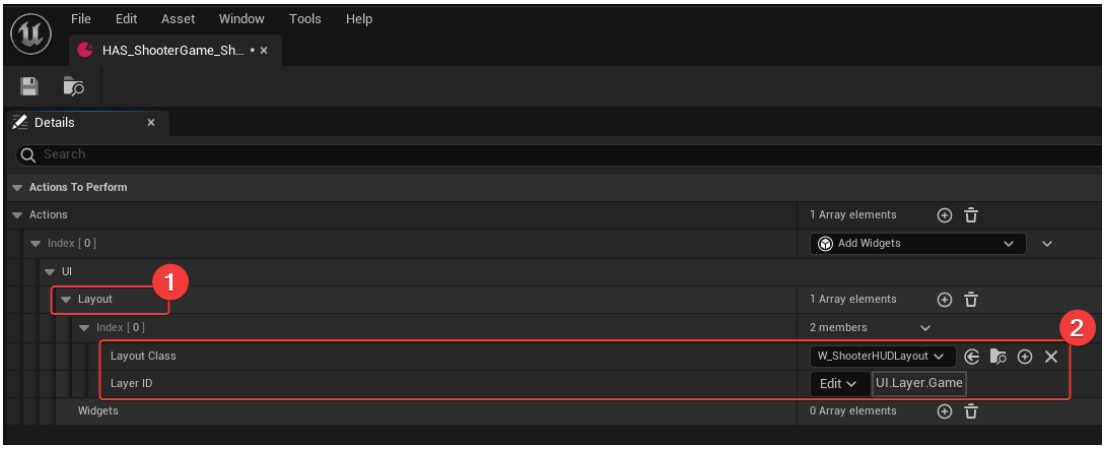


- [ ] HAS_Shooter_StandardHUD 추가:

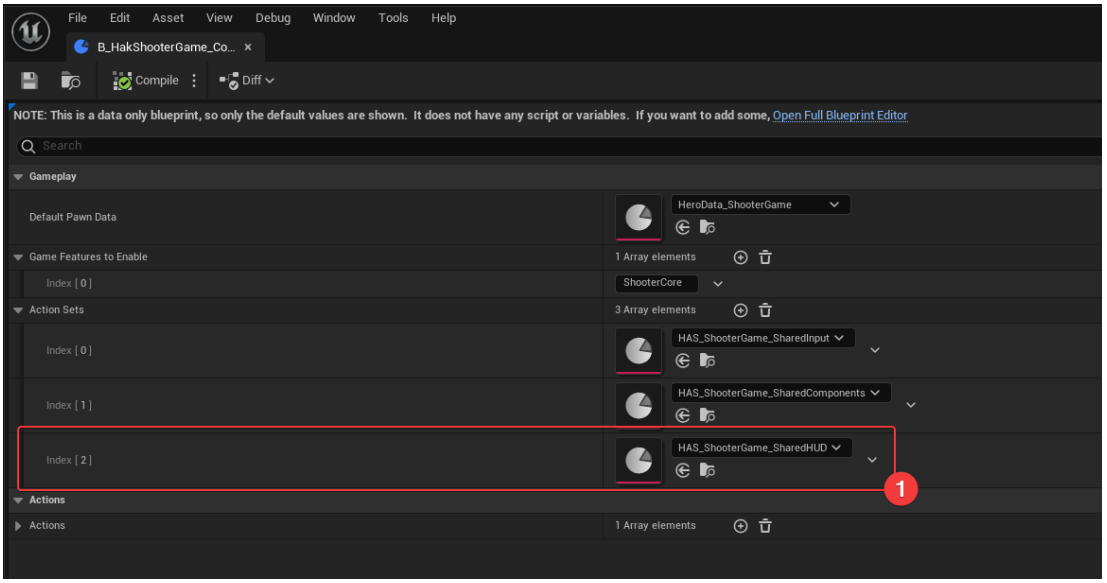




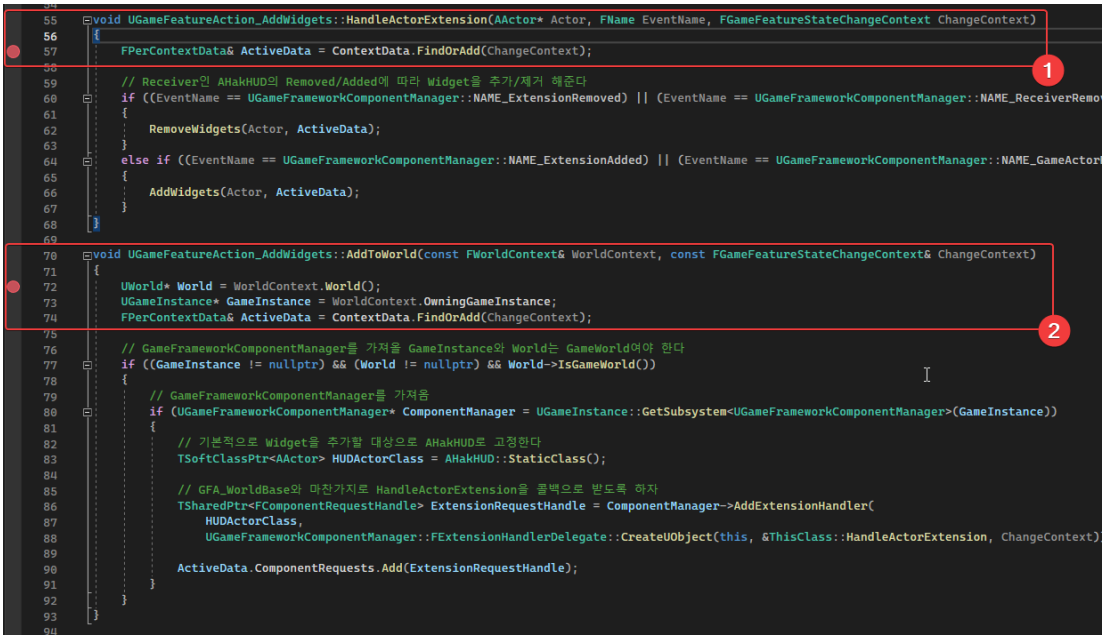
☐ HAS_Shooter_SharedHUD에 GFA(GameFeatureAction)의 AddWidgets 추가:



☐ B_HakShooterGame_ControlPoints에 HAS_Shooter_SharedHUD 추가:



☐ W_ShooterHUDLayout이 GFA의 AddWidgets로 잘 설정되는지 한번 보자:



- 2번은 들어오는데, 1번이 들어오지 않는다....
  - **<u>이는 HakHUD가 기본 클래스로 설정되어 있지 않았다!</u>**

☐ HakGameMode.cpp에 기본 HUD 클래스로 HakHUD로 설정하자:

```cpp
#include "HakGameMode.h"
#include "HakGameState.h"
#include "HakExperienceManagerComponent.h"
#include "HakExperienceDefinition.h"
#include "Kismet/GameplayStatics.h"
#include "HakGame/Player/HakPlayerController.h"
#include "HakGame/Player/HakPlayerState.h"
#include "HakGame/Character/HakCharacter.h"
#include "HakGame/Character/HakPawnData.h"
#include "HakGame/Character/HakPawnExtensionComponent.h"
#include "HakGame/UI/HakHUD.h"
#include "HakGame/HakLogChannels.h"

#include UE_INLINE_GENERATED_CPP_BY_NAME(HakGameMode)

AHakGameMode::AHakGameMode(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
    // 해당 클래스의 Clone 대상이 되는 LyraGameMode를 살펴보자:
    // - 우리는 첫번째로 당장 필요한 GameState, PlayerController, PlayerState와 Character를 구현해보자:
    GameStateClass = AHakGameState::StaticClass();
    PlayerControllerClass = AHakPlayerController::StaticClass();
    PlayerStateClass = AHakPlayerState::StaticClass();
    DefaultPawnClass = AHakCharacter::StaticClass();
    HUDClass = AHakHUD::StaticClass();
}
```

1

2

- 앞서, 들어오지 않았던 Breakpoint가 잘 들어온다:

```cpp
void UGameFeatureAction_AddWidgets::HandleActorExtension(AActor* Actor, FName EventName, FGameFeatureStateChangeContext ChangeContext)
{
    FPerContextData& ActiveData = ContextData.FindOrAdd(ChangeContext);  ≤ 1ms elapsed

    // Receiver인 AHakHUD의 Removed/Added에 따라 Widget을 추가/제거 해준다
    if ((EventName == UGameFrameworkComponentManager::NAME_ExtensionRemoved) || (EventName == UGameFrameworkComponentManager::NAME_ReceiverRemoved))
    {
        RemoveWidgets(Actor, ActiveData);
    }
    else if ((EventName == UGameFrameworkComponentManager::NAME_ExtensionAdded) || (EventName == UGameFrameworkComponentManager::NAME_GameActorReady))
    {
        AddWidgets(Actor, ActiveData);
    }
}
```

☐ 우리가 추가한 W_ShooterHUDLayout이 작동하지 않는다 (**로딩되어 있지 않아서!!**)

```cpp
void UGameFeatureAction_AddWidgets::AddWidgets(AActor* Actor, FPerContextData& ActiveData)
{
    AHakHUD* HUD = CastChecked<AHakHUD>(Actor);

    // HUD를 통해, LocalPlayer을 가져오자
    if (ULocalPlayer* LocalPlayer = Cast<ULocalPlayer>(HUD->GetOwningPlayerController()->Player))
    {
        // Layout의 요청을 순회하자 (보통 하나만 들어가 있긴하다)
        for (const FHakHUDLayoutRequest& Entry : Layout)
        {
            if (TSubclassOf<UCommonActivatableWidget> ConcreteWidgetClass = Entry.LayoutClass.Get())
            {
                ActiveData.LayoutsAdded.Add(UCommonUIExtensions::PushContentToLayer_ForPlayer(LocalPlayer, Entry.LayerID, ConcreteWidgetClass));
            }
        }

        // Widget을 순회하며, UIExtensionSubsystem의 Extension에 추가한다
        UUIExtensionSubsystem* ExtensionSubsystem = HUD->GetWorld()->GetSubsystem<UUIExtensionSubsystem>();  ≤ 4ms elapsed
        for (const FHakHUDElementEntry& Entry : Widgets)
        {
            ActiveData.ExtensionHandles.Add(ExtensionSubsystem->RegisterExtensionAsWidgetForContext(Entry.SlotID, LocalPlayer, Entry.WidgetClass.Get(), -1));
        }
    }
}
```

1 Entry.LayoutClass.Get()이 nullptr를 반환한다.
- StaticFindObject가 실패했다는 것을 의미한다 (ResolveObject 참고)

2 여기로 들어오지 않는다...

- 위의 문제를 해결하는 방법은 간단히게 그냥 저 코드에서 LoadSynchronous()를 호출해도 된다. 하지만, 이는 제대로 된 방법은 아니다!
    - 위의 문제를 해결하기 위해 PrimaryDataAsset의 로딩 방식을 이해할 필요가 있다:

- **<u>너무 깊게 다루기 보다, 문제를 어떻게 해결해 나가는지 보면서, 여러분들이 PrimaryDataAsset에 대한 코드를 한번 Deep-Dive 해보길 추천한다</u>**
  (필자는 Deep-Dive하고 왔다~ 🙂)

☐ HakExperienceManagerComponent::StartExperienceLoad()로 가자:

```cpp
void UHakExperienceManagerComponent::StartExperienceLoad()
{
    check(CurrentExperience);
    check(LoadState == EHakExperienceLoadState::Unloaded);

    LoadState = EHakExperienceLoadState::Loading;

    UHakAssetManager& AssetManager = UHakAssetManager::Get();

    TSet<FPrimaryAssetId> BundleAssetList;

    // 이미 앞서, ServerSetCurrentExperience에서 우리는 ExperienceId를 넘겨주었는데, 여기서 CDO를 활용하여, GetPrimaryAssetId를 로딩할 대상으로 넣는다!
    // - 왜 이렇게 하는걸까?
    // - GetPrimaryAssetId를 좀 더 자세히보자:
    // - GetPrimaryAssetId를 살펴봄으로써, 아래의 두가지를 알 수 있다:
    //   1. 우리는 B_HakDefaultExperience를 BP로 만든 이유
    //   2. CDO를 가져와서, GetPrimaryAssetId를 호출한 이유

    // 우리는 앞서 이미 CDO로 로딩하여, CDO를 사용하지 않고 CDO를 사용하여 로딩할 에셋을 지정하여, BundleAssetList에 추가해준다!
    BundleAssetList.Add(CurrentExperience->GetPrimaryAssetId());

    // ExperienceActionSet의 순회하며, BundleAssetList로 추가하자:
    for (const TObjectPtr<UHakExperienceActionSet>& ActionSet : CurrentExperience->ActionSets)
    {
        if (ActionSet)
        {
            // 앞서, 우리가 생성한 HAS_Shooter_SharedHUD가 추가되겠다 (물론 추가적인 HAS_Shooter_XXX)도 추가될거다
            // - BundleAssetList는 Bundle로 등록할 Root의 PrimaryDataAsset를 추가하는 과정이다
            //   (->??? 무슨말인가 싶을건데 ChangeBundleStateForPrimaryAssets)을 살펴보면서 이해하자
            BundleAssetList.Add(ActionSet->GetPrimaryAssetId());
        }
    }

    // load assets associated with the experience
    // 아래는 우리가 후일 GameFeature를 사용하여, Experience에 바인딩된 GameFeature Plugin을 로딩할 Bundle 이름을 추가한다:
    // - Bundle이라는게 후일 우리가 로딩할 에셋의 카테고리 이름이라고 생각하면 된다 (일단 지금은 넘어가자 후일, 또 다룰 것이다!)
    TArray<FName> BundlesToLoad;
    {
        // 여기서 주목해야 할 부분은 OwnerNetMode가 NM_Standalone이면? Client/Server 둘다 로딩에 추가된다!
        const ENetMode OwnerNetMode = GetOwner()->GetNetMode();
        bool bLoadClient = GIsEditor || (OwnerNetMode != NM_DedicatedServer);
        bool bLoadServer = GIsEditor || (OwnerNetMode != NM_Client);

        // 오늘! 아래의 의미를 알게 될 것이다:
        // - LoadStateClient = "Client", LoadStateServer = "Server"
        if (bLoadClient)
        {
            BundlesToLoad.Add(UGameFeaturesSubsystemSettings::LoadStateClient);
        }
        if (bLoadServer)
        {
            BundlesToLoad.Add(UGameFeaturesSubsystemSettings::LoadStateServer);
        }
    }

    FStreamableDelegate OnAssetsLoadedDelegate = FStreamableDelegate::CreateUObject(this, &ThisClass::OnExperienceLoadComplete);

    // 아래도, 후일 Bundle을 우리가 GameFeature에 연동하면서 더 깊게 알아보기로 하고, 지금은 앞서 B_HakDefaultExperience를 로딩해주는 함수로 생각하자
    TSharedPtr<FStreamableHandle> Handle = AssetManager.ChangeBundleStateForPrimaryAssets(
        BundleAssetList.Array(),
        BundlesToLoad,
        {}, false, FStreamableDelegate(), FStreamableManager::AsyncLoadHighPriority);

    if (!Handle.IsValid() || Handle->HasLoadCompleted())
    {
        // 로딩이 완료되었으면, ExecuteDelegate를 통해 OnAssetsLoadedDelegate를 호출하자:
        // - 아래의 함수를 확인해보자:
        FStreamableHandle::ExecuteDelegate(OnAssetsLoadedDelegate);
    }
    else
    {
        Handle->BindCompleteDelegate(OnAssetsLoadedDelegate);
        Handle->BindCancelDelegate(FStreamableDelegate::CreateLambda([OnAssetsLoadedDelegate]()
            {
                OnAssetsLoadedDelegate.ExecuteIfBound();
            }));
    }

    // FrameNumber를 주목해서 보자
    static int32 StartExperienceLoad_FrameNumber = GFrameNumber;
}
```

☐ ChangeBundleStateForPrimaryAssets() 확인:

```cpp
TSharedPtr<FStreamableHandle> UAssetManager::ChangeBundleStateForPrimaryAssets(const TArray<FPrimaryAssetId>& AssetsToChange, const TArray<FName>& AddBundles, const TArray<FName>& RemoveBundles, bool bRem
{
    TArray<TSharedPtr<FStreamableHandle> > NewHandles, ExistingHandles;
    TArray<FPrimaryAssetId> NewAssets;
    TSharedPtr<FStreamableHandle> ReturnHandle;
    int32 MoviePlayerNumAssets = 0;

    // 앞서 BundleAssetList가 AssetsToChange로 전달되었다:
    // - AssetsToChange는 PrimaryDataAsset이다
    for (const FPrimaryAssetId& PrimaryAssetId : AssetsToChange)
    {
        MoviePlayerNumAssets++;
        // Call the blocking tick every 500 assets.
        if (MoviePlayerNumAssets >= 500)
        {
            MoviePlayerNumAssets = 0;
            //UE_LOG(LogTemp, Warning, TEXT("pooo %d"), MoviePlayerNumAssets);
            FMoviePlayerProxy::BlockingTick();
        }

        // --- 여기가 우리가 PrimaryDataAsset으로 등록해도 PrimaryAssetType으로 등록되어 있지 않으면 NameData가 nullptr로 반환된다
        // PrimaryAssetType은 ***AssetMap***에 등록되어있다
        FPrimaryAssetData* NameData = GetNameData(PrimaryAssetId);

        if (NameData)
        {
            FPlatformMisc::PumpEssentialAppMessages();

            // Iterate list of changes, compute new bundle set
            bool bLoadIfNeeded = false;

            // Use pending state if valid
            // PrimaryAssetData인 NameData에 PostLoad 이후, Bundle로 등록된 에셋 리스트를 가지고 있다:
            // - 이는 조금 있다가 또다른 문제를 해결하면서 보게 될 것이다.
            TArray<FName> CurrentBundleState = NameData->PendingState.IsValid() ? NameData->PendingState.BundleNames : NameData->CurrentState.BundleNames;
            TArray<FName> NewBundleState;

            if (!bRemoveAllBundles){ ... }

            for (const FName& AddBundle : AddBundles)
            {
                NewBundleState.AddUnique(AddBundle);
            }

            NewBundleState.Sort(FNameLexicalLess());

            // If the pending state is valid, check if it is different
            if (NameData->PendingState.IsValid()){ ... }
            else if (NameData->CurrentState.IsValid() && NameData->CurrentState.BundleNames == NewBundleState){ ... }

            // Bundles을 포함하여, RootAsset인 NamedData->AssetPtr도 같이해서 로딩할 에셋의 FullPath를 다 담는다
            TSet<FSoftObjectPath> PathsToLoad;

            // Gather asset refs
            const FSoftObjectPath& AssetPath = NameData->AssetPtr.ToSoftObjectPath();

            if (!AssetPath.IsNull())
            {
                // Dynamic types can have no base asset path
                PathsToLoad.Add(AssetPath);
            }

            for (const FName& BundleName : NewBundleState)
            {
                FAssetBundleEntry Entry = GetAssetBundleEntry(PrimaryAssetId, BundleName);

                if (Entry.IsValid())
                {
                    for (const FTopLevelAssetPath & Path : Entry.AssetPaths)
                    {
                        PathsToLoad.Emplace(FSoftObjectPath(Path));
                    }
                }
                else
                {
                    UE_LOG(LogAssetManager, Verbose, TEXT("ChangeBundleStateForPrimaryAssets: No assets for bundle %s::%s"), *PrimaryAssetId.ToString(), *BundleName.ToString());
                }
            }

            if (PathsToLoad.Num() == 0){ ... }

            TSharedPtr<FStreamableHandle> NewHandle;
```

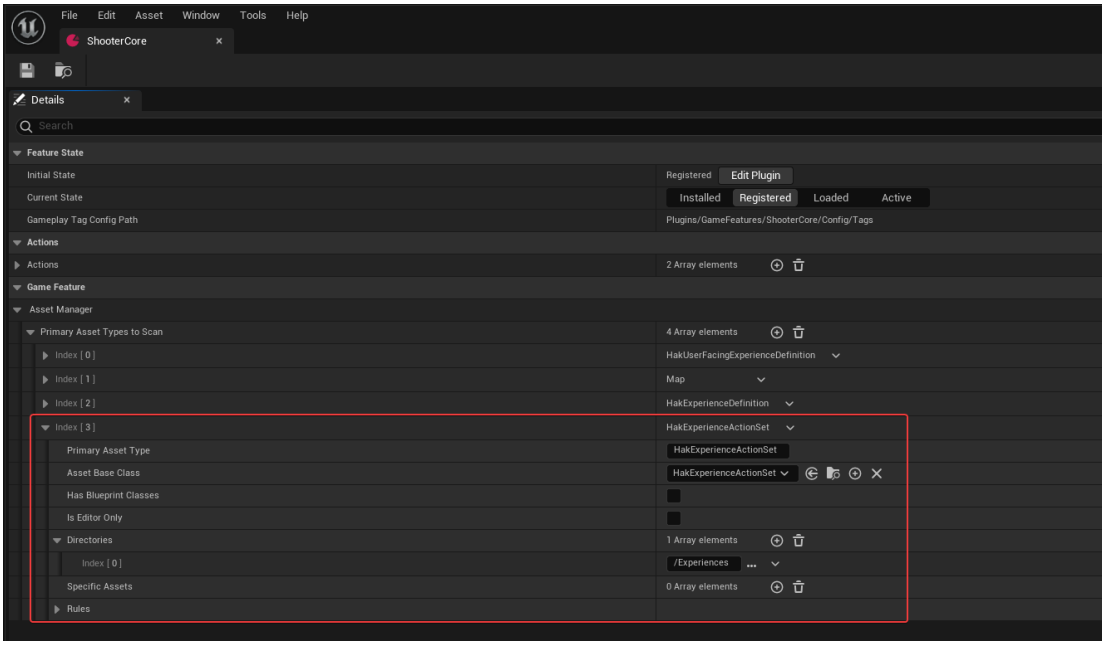
```cpp
            TSharedPtr<FStreamableHandle> NewHandle;

            TStringBuilder<1024> DebugName;
            PrimaryAssetId.AppendString(DebugName);

            if (NewBundleState.Num() > 0){ ... }

            // 여기서 로딩 시작한다!
            NewHandle = LoadAssetList(PathsToLoad.Array(), FStreamableDelegate(), Priority, FString(DebugName.Len(), DebugName.ToString()));

            if (!NewHandle.IsValid())
            {
                // LoadAssetList already throws an error, no need to do it here as well
                continue;
            }

            if (NewHandle->HasLoadCompleted()){ ... }
            else{ ... }

            NewHandles.Add(NewHandle);
            NewAssets.Add(PrimaryAssetId);
        }
        else
        {
            WarnAboutInvalidPrimaryAsset(PrimaryAssetId, TEXT("ChangeBundleStateForPrimaryAssets failed to find NameData"));
        }
    }

    if (NewHandles.Num() > 1 || ExistingHandles.Num() > 0){ ... }
    else if (NewHandles.Num() == 1){ ... }
    else
    {
        // Call completion callback, nothing to do
        FStreamableHandle::ExecuteDelegate(DelegateToCall);
    }

    return ReturnHandle;
}
```

☐ 자, 이제 HAS_Shooter_SharedHUD가 잘 로딩되는지 확인해보자:

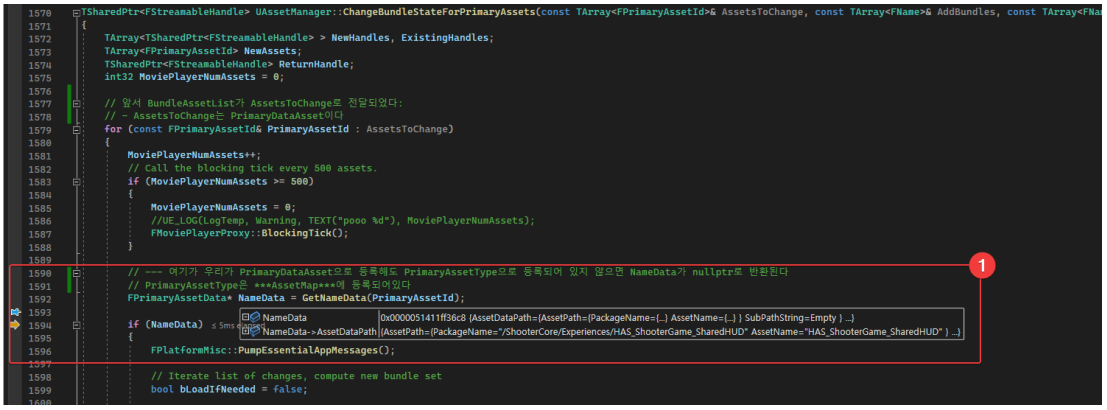
```cpp
TSharedPtr<FStreamableHandle> UAssetManager::ChangeBundleStateForPrimaryAssets(const TArray<FPrimaryAssetId>& AssetsToChar
{
    TArray<TSharedPtr<FStreamableHandle> > NewHandles, ExistingHandles;
    TArray<FPrimaryAssetId> NewAssets;
    TSharedPtr<FStreamableHandle> ReturnHandle;
    int32 MoviePlayerNumAssets = 0;

    // 앞서 BundleAssetList가 AssetsToChange로 전달되었다:
    // - AssetsToChange는 PrimaryDataAsset이다
    for (const FPrimaryAssetId& PrimaryAssetId : AssetsToChange)
    {
        MoviePlayerNumAssets++;
        // Call the blocking tick every 500 assets.
        if (MoviePlayerNumAssets >= 500)
        {
            MoviePlayerNumAssets = 0;
            //UE_LOG(LogTemp, Warning, TEXT("pooo %d"), MoviePlayerNumAssets);
            FMoviePlayerProxy::BlockingTick();
        }

        // --- 여기가 우리가 PrimaryDataAsset으로 등록해도 PrimaryAssetType으로 등록되어 있지 않으면 NameData가 nullptr로 반환된다
        // PrimaryAssetType은 ***AssetMap***에 등록되어있다
        FPrimaryAssetData* NameData = GetNameData(PrimaryAssetId);
                                      NameData  0x0000000000000000 <NULL>
        if (NameData)  ≤ 5ms elapsed
        {
            FPlatformMisc::PumpEssentialAppMessages();
```

- NewData가 nullptr이 반환된다:
    - 주석에 설명했듯이 AssetMap에 PrimaryDataType으로 설정되어있지 않았다!

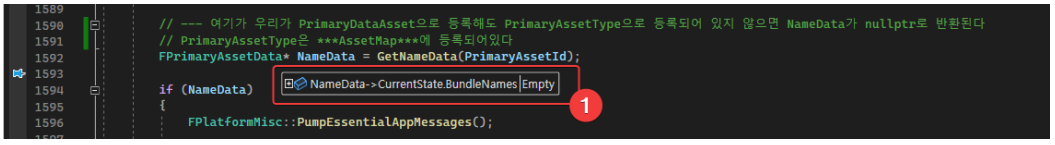
☐ ExperienceActionSet을 ShooterCore의 GameFeatureData에 PrimaryAssetType로 설정:



☐ 잘 로딩되었다:



- [ ] 허나, Bundles이 하나도 없다…



- [ ] Bundles을 추가 시키기 위해, **<u>UpdateAssetBundleData()를 오버로딩 해야 한다:</u>**

  - [ ] UHakExperienceDefinition:

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

#if WITH_EDITORONLY_DATA
    virtual void UpdateAssetBundleData() override;
#endif

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

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

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

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

```cpp
#if WITH_EDITORONLY_DATA
void UHakExperienceDefinition::UpdateAssetBundleData()
{
    // 우리는 UpdateAssetBundleData() 코드를 한번 봐야한다
    Super::UpdateAssetBundleData();

    for (UGameFeatureAction* Action : Actions)
    {
        if (Action)
        {
            // AddAddditionalAssetBundleData()는 UGameFeatureAction의 메서드이다
            // - 우리가 임의적으로 호출을 통해 AssetBundleData에 추가해준다
            Action->AddAdditionalAssetBundleData(AssetBundleData);
        }
    }
}
#endif
```

- [ ] HakExperienceActionSet

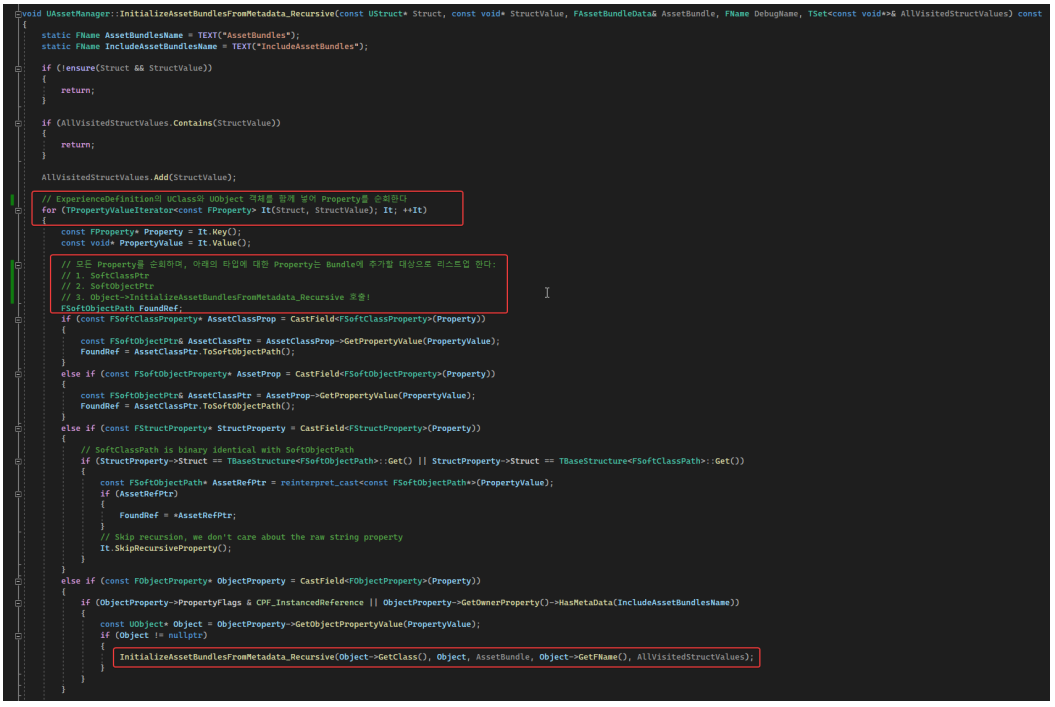
```cpp
#pragma once

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

class UGameFeatureAction;

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

#if WITH_EDITORONLY_DATA
    virtual void UpdateAssetBundleData() override;
#endif

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

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

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

#if WITH_EDITORONLY_DATA
void UHakExperienceActionSet::UpdateAssetBundleData()
{
    Super::UpdateAssetBundleData();

    for (UGameFeatureAction* Action : Actions)
    {
        if (Action)
        {
            Action->AddAdditionalAssetBundleData(AssetBundleData);
        }
    }
}
#endif
```

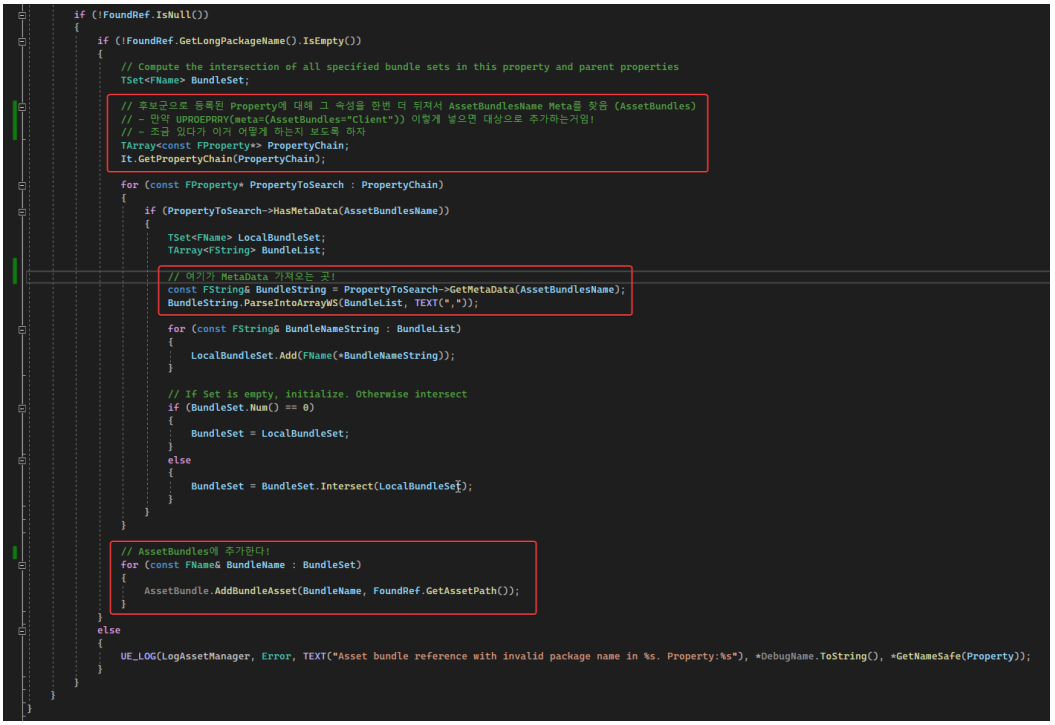
☐ UPrimaryDataAsset::UpdateAssetBundleData()를 한번 보자:

```cpp
void UPrimaryDataAsset::UpdateAssetBundleData()
{
    // By default parse the metadata
    if (UAssetManager::IsValid())
    {
        AssetBundleData.Reset();

        // AssetBundleData를 UE의 Reflection 시스템을 활용하여, 추가시킨다
        // - 자세한 것은 InitializeAssetBundlesFromMetadata를 보자
        UAssetManager::Get().InitializeAssetBundlesFromMetadata(this, AssetBundleData);
    }
}
```

☐ UAssetManager::InitializeAssetBundlesFromMetadata()



```cpp
void UAssetManager::InitializeAssetBundlesFromMetadata_Recursive(const UStruct* Struct, const void* StructValue, FAssetBundleData& AssetBundle, FName DebugName, TSet<const void*>& AllVisitedStructValues) const
{
    static FName AssetBundlesName = TEXT("AssetBundles");
    static FName IncludeAssetBundlesName = TEXT("IncludeAssetBundles");

    if (!ensure(Struct && StructValue))
    {
        return;
    }

    if (AllVisitedStructValues.Contains(StructValue))
    {
        return;
    }

    AllVisitedStructValues.Add(StructValue);

    // ExperienceDefinition의 UClass와 UObject 객체를 함께 넣어 Property를 순회한다
    for (TPropertyValueIterator<const FProperty> It(Struct, StructValue); It; ++It)
    {
        const FProperty* Property = It.Key();
        const void* PropertyValue = It.Value();

        // 모든 Property를 순회하며, 아래의 타입에 대한 Property는 Bundle에 추가할 대상으로 리스트업 한다:
        // 1. SoftClassPtr
        // 2. SoftObjectPtr
        // 3. Object->InitializeAssetBundlesFromMetadata_Recursive 호출!
        FSoftObjectPath FoundRef;
        if (const FSoftClassProperty* AssetClassProp = CastField<FSoftClassProperty>(Property))
        {
            const FSoftObjectPtr& AssetClassPtr = AssetClassProp->GetPropertyValue(PropertyValue);
            FoundRef = AssetClassPtr.ToSoftObjectPath();
        }
        else if (const FSoftObjectProperty* AssetProp = CastField<FSoftObjectProperty>(Property))
        {
            const FSoftObjectPtr& AssetClassPtr = AssetProp->GetPropertyValue(PropertyValue);
            FoundRef = AssetClassPtr.ToSoftObjectPath();
        }
        else if (const FStructProperty* StructProperty = CastField<FStructProperty>(Property))
        {
            // SoftClassPath is binary identical with SoftObjectPath
            if (StructProperty->Struct == TBaseStructure<FSoftObjectPath>::Get() || StructProperty->Struct == TBaseStructure<FSoftClassPath>::Get())
            {
                const FSoftObjectPath* AssetRefPtr = reinterpret_cast<const FSoftObjectPath*>(PropertyValue);
                if (AssetRefPtr)
                {
                    FoundRef = *AssetRefPtr;
                }
                // Skip recursion, we don't care about the raw string property
                It.SkipRecursiveProperty();
            }
        }
        else if (const FObjectProperty* ObjectProperty = CastField<FObjectProperty>(Property))
        {
            if (ObjectProperty->PropertyFlags & CPF_InstancedReference || ObjectProperty->GetOwnerProperty()->HasMetaData(IncludeAssetBundlesName))
            {
                const UObject* Object = ObjectProperty->GetObjectPropertyValue(PropertyValue);
                if (Object != nullptr)
                {
                    InitializeAssetBundlesFromMetadata_Recursive(Object->GetClass(), Object, AssetBundle, Object->GetFName(), AllVisitedStructValues);
                }
            }
        }
```

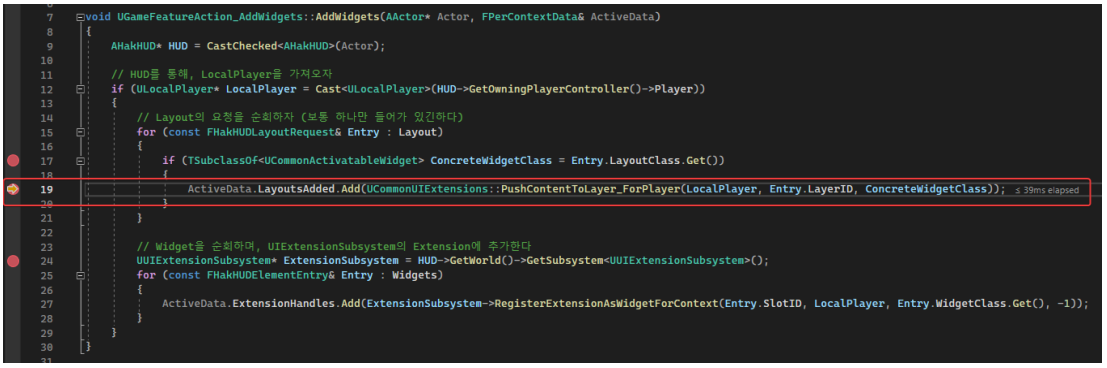


```cpp
        if (!FoundRef.IsNull())
        {
            if (!FoundRef.GetLongPackageName().IsEmpty())
            {
                // Compute the intersection of all specified bundle sets in this property and parent properties
                TSet<FName> BundleSet;

                // 후보군으로 등록된 Property에 대해 그 속성을 한번 더 뒤져서 AssetBundlesName Meta를 찾음 (AssetBundles)
                // - 만약 UPROEPRRY(meta={AssetBundles="Client"}) 이렇게 넣으면 대상으로 추가하는거임!
                // - 조금 있다가 이거 어떻게 하는지 보도록 하자
                TArray<const FProperty*> PropertyChain;
                It.GetPropertyChain(PropertyChain);

                for (const FProperty* PropertyToSearch : PropertyChain)
                {
                    if (PropertyToSearch->HasMetaData(AssetBundlesName))
                    {
                        TSet<FName> LocalBundleSet;
                        TArray<FString> BundleList;

                        // 여기가 MetaData 가져오는 곳!
                        const FString& BundleString = PropertyToSearch->GetMetaData(AssetBundlesName);
                        BundleString.ParseIntoArrayWS(BundleList, TEXT(","));

                        for (const FString& BundleNameString : BundleList)
                        {
                            LocalBundleSet.Add(FName(*BundleNameString));
                        }

                        // If Set is empty, initialize. Otherwise intersect
                        if (BundleSet.Num() == 0)
                        {
                            BundleSet = LocalBundleSet;
                        }
                        else
                        {
                            BundleSet = BundleSet.Intersect(LocalBundleSet);
                        }
                    }
                }

                // AssetBundles에 추가한다!
                for (const FName& BundleName : BundleSet)
                {
                    AssetBundle.AddBundleAsset(BundleName, FoundRef.GetAssetPath());
                }
            }
            else
            {
                UE_LOG(LogAssetManager, Error, TEXT("Asset bundle reference with invalid package name in %s. Property:%s"), *DebugName.ToString(), *GetNameSafe(Property));
            }
        }
    }
}
```

- [ ] 코드에서 보았듯이, AssetBundle metadata를 추가하자:

```cpp
/** HUD의 Layout 요청 */
USTRUCT()
struct FHakHUDLayoutRequest
{
    GENERATED_BODY()

    /** UI의 레이아웃으로 CommonActivatableWidget을 사용 */
    UPROPERTY(EditAnywhere, Category=UI, meta=(AssetBundles="Client"))
    TSoftClassPtr<UCommonActivatableWidget> LayoutClass;

    /** 앞서 보았던 PrimaryGameLayout의 LayerID를 의미 */
    UPROPERTY(EditAnywhere, Category=UI)
    FGameplayTag LayerID;
};
```

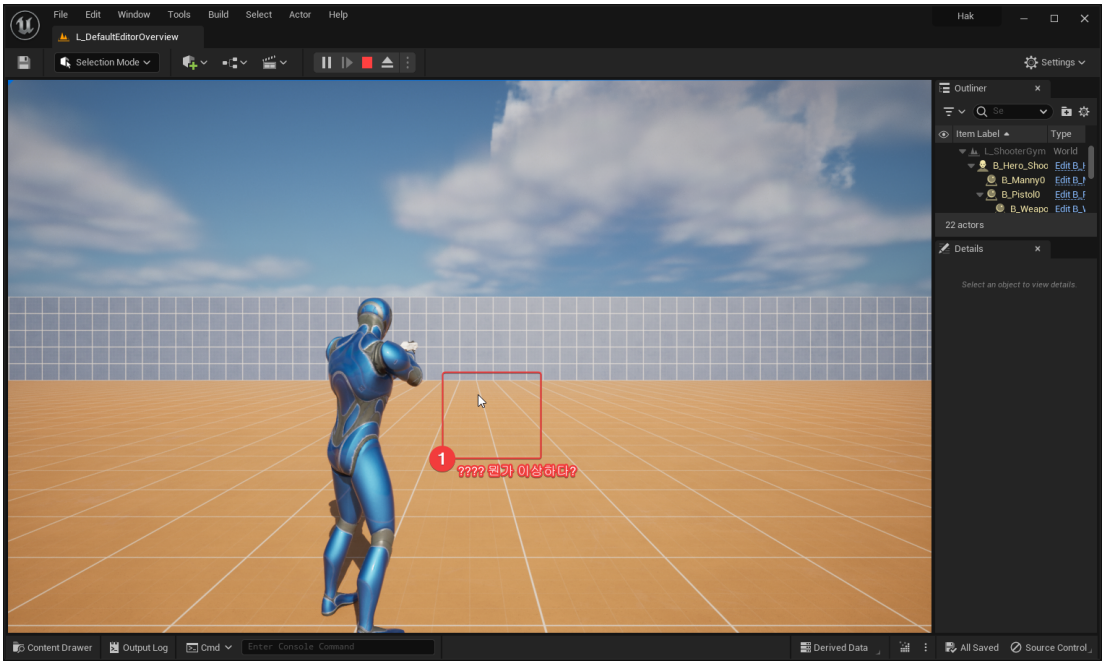
☐ 이제 잘 들어오는 것을 확인할 수 있다:



```cpp
void UGameFeatureAction_AddWidgets::AddWidgets(AActor* Actor, FPerContextData& ActiveData)
{
    AHakHUD* HUD = CastChecked<AHakHUD>(Actor);

    // HUD를 통해, LocalPlayer을 가져오자
    if (ULocalPlayer* LocalPlayer = Cast<ULocalPlayer>(HUD->GetOwningPlayerController()->Player))
    {
        // Layout의 요청을 순회하자 (보통 하나만 들어가 있긴하다)
        for (const FHakHUDLayoutRequest& Entry : Layout)
        {
            if (TSubclassOf<UCommonActivatableWidget> ConcreteWidgetClass = Entry.LayoutClass.Get())
            {
                ActiveData.LayoutsAdded.Add(UCommonUIExtensions::PushContentToLayer_ForPlayer(LocalPlayer, Entry.LayerID, ConcreteWidgetClass)); ≤ 39ms elapsed
            }
        }

        // Widget을 순회하며, UIExtensionSubsystem의 Extension에 추가한다
        UUIExtensionSubsystem* ExtensionSubsystem = HUD->GetWorld()->GetSubsystem<UUIExtensionSubsystem>();
        for (const FHakHUDElementEntry& Entry : Widgets)
        {
            ActiveData.ExtensionHandles.Add(ExtensionSubsystem->RegisterExtensionAsWidgetForContext(Entry.SlotID, LocalPlayer, Entry.WidgetClass.Get(), -1));
        }
    }
}
```

- 앞서 보았던 InitializeAssetBundlesFromMetadata()를 디버깅하면서 잘 작동하는지 스스로 꼭 한번 확인해보자.

☐ 뭔가 아래와 같이 키 입력과 마우스 입력이 이상하다:



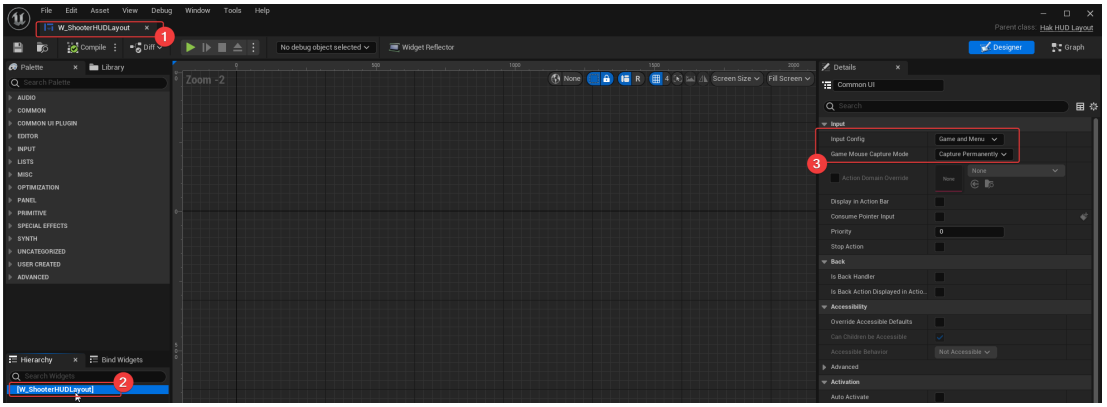
- 이는 앞서 HakActivatableWidget에 Input에 대한 처리가 제대로 되지 않아서 그렇다

☐ UCommonActivatableWidget::GetDesiredInputConfig()를 오버라이드 하자:

```cpp
/**
 * CommonActivatableWidget의 주석을 보면, 복잡하게 설명되어 있다.
 * 필자가 이해한 CommonActivatableWidget은 두가지 특성을 가지고 있다:
 * 1. Widget Layout과 Widget Instance을 구분하기 위한 용도로 CommonActivatableWidget은 Layout 정의:
 *    - 런타임 Activate/Deactivate
 *    - WidgetTree에서 제거가 아닌 껐다/켰다(== Activate/Deactivate)
 * 2. Input(Mouse or Keyboard etc...) 처리 방법 정의
 */
UCLASS(Abstract, Blueprintable)
class UHakActivatableWidget : public UCommonActivatableWidget
{
    GENERATED_BODY()
public:
    UHakActivatableWidget(const FObjectInitializer& ObjectInitializer);

    /**
     * UCommonActivatableWidget's interfaces
     */
    virtual TOptional<FUIInputConfig> GetDesiredInputConfig() const override;

    /** Input 처리 방식 */
    UPROPERTY(EditDefaultsOnly, Category=Input)
    EHakWidgetInputMode InputConfig = EHakWidgetInputMode::Default;

    /** Mouse 처리 방식 */
    UPROPERTY(EditDefaultsOnly, Category=Input)
    EMouseCaptureMode GameMouseCaptureMode = EMouseCaptureMode::CapturePermanently;
};
```

```cpp
TOptional<FUIInputConfig> UHakActivatableWidget::GetDesiredInputConfig() const
{
    // 앞서 정의 했던 WidgetInputMode에 따라, InputConfig를 설정해준다
    // - ECommonInputMode에 따라 Input의 흐름을 Game/Menu 혹은 둘다로 정하는 것이 가능하다
    // - 예로 들어, 우리가 Menu로 들어갔을 경우, 더이상 Game에 Input을 보내고 싶지 않을 경우 매우 용의하다:
    //   - 상상해봐라, Menu 모드인데 Space를 누르거나 MouseClick으로 총알이 나간다면... 우리가 의도한 상황이 아닐 것이다
    switch (InputConfig)
    {
    case EHakWidgetInputMode::GameAndMenu:
        return FUIInputConfig(ECommonInputMode::All, GameMouseCaptureMode);
    case EHakWidgetInputMode::Game:
        return FUIInputConfig(ECommonInputMode::Game, GameMouseCaptureMode);
    case EHakWidgetInputMode::Menu:
        return FUIInputConfig(ECommonInputMode::Menu, GameMouseCaptureMode);
    case EHakWidgetInputMode::Default:
    default:
        return TOptional<FUIInputConfig>();
    }
}
```

☐ W_ShooterHUDLayout의 InputConfig와 GameMouseCaptureMode를 잘 설정해주자:

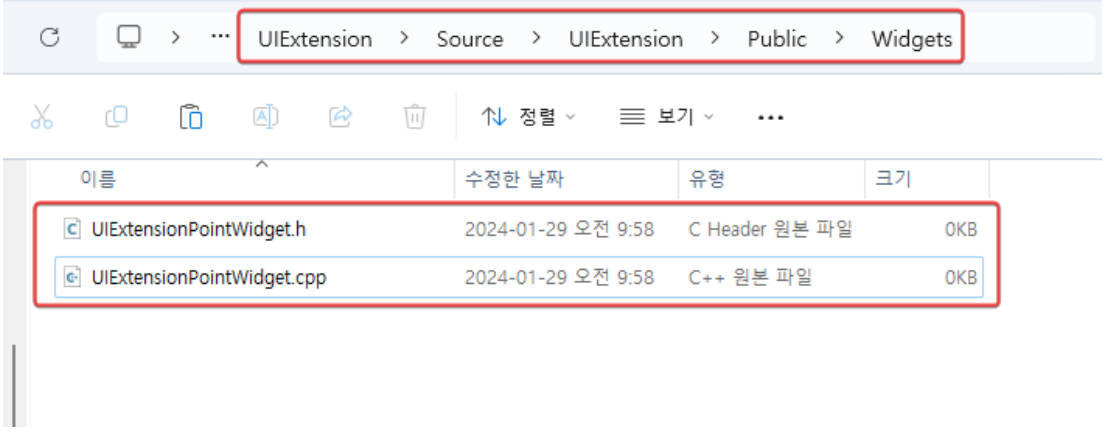


☐ 그럼 이제 원래대로 잘 작동한다!

UIExtensionPointWidget

▼ **펼치기**

☐ UIExtensionPointWidget 파일 추가:



☐ UIExtensionPointWidget.h/.cpp:

```cpp
#include "CoreMinimal.h"
#include "GameplayTagContainer.h"
#include "UIExtensionSystem.h"
#include "Components/DynamicEntryBoxBase.h"
#include "UIExtensionPointWidget.generated.h"

/** forward declarations */
class UCommonLocalPlayer;

/**
 * UUIExtensionPointWidget은 UIExtension 하나 단위로 생각하면 된다:
 * - UIExtension에 결합된 Context별로 관리하는 객체가 UIExtenionPoint(Handle)이다:
 * - 예로 들어, 해당 UIExtension Slot에 PlayerState/PlayerController/LocalPlayer와 같다 다양한 Context로부터 Widge이 결합될 수 있다
 *   이를 분리하여 관리하기 위해 만들어진 객체가 UIExtensionPoint이다
 */
UCLASS()
class UIEXTENSION_API UUIExtensionPointWidget : public UDynamicEntryBoxBase
{
    GENERATED_BODY()
public:
    UUIExtensionPointWidget(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "UI Extension")
    FGameplayTag ExtensionPointTag;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "UI Extension")
    EUIExtensionPointMatch ExtensionPointTagMatch = EUIExtensionPointMatch::ExactMatch;

    /** UIExtensionPointWidget에 결합이 허용된 UWidget Classes */
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "UI Extension")
    TArray<TObjectPtr<UClass>> DataClasses;

    /** UIExtension --- Widget 관계 매핑 */
    UPROPERTY(Transient)
    TMap<FUIExtensionHandle, TObjectPtr<UUserWidget>> ExtensionMapping;

    /** UIExtensionPoint 객체 관리 */
    TArray<FUIExtensionPointHandle> ExtensionPointHandles;
};
```

```cpp
#include "UIExtensionPointWidget.h"
#include "CommonLocalPlayer.h"
#include "GameFramework/PlayerState.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(UIExtensionPointWidget)

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

}
```

☐ UIExtensionSystem.h에 EUIExtensionPointMatch 정의:

```cpp
#pragma once

#include "GameplayTagContainer.h"
#include "Templates/SharedPointer.h"
#include "Subsystems/WorldSubsystem.h"

#include "UIExtensionSystem.generated.h"

/** ExtensionPoint GameplayTag 비교 방법 */
UENUM(BlueprintType)
enum class EUIExtensionPointMatch : uint8
{
    /** A.B는 A.B.C를 ***허용하지*** 않는다 */
    ExactMatch,

    /** A.B는 A.B.C를 ***허용*** 한다 */
    PartialMatch,
};

struct FUIExtension : TSharedFromThis<FUIExtension>
{
    /** UIExtension Widget의 Point Tag이다 (무슨 의미인지 하나씩 구현내가며 이해해보자) */
    FGameplayTag ExtensionPointTag;

    /** Widget Class를 가지고 있으며, UUIExtensionSubsystem에서 AddReferencedObjects에서 GC를 막는다 */
    UObject* Data = nullptr;

    /** 보통 LocalPlayer로 설정된다 */
    TWeakObjectPtr<UObject> ContextObject;

    int32 Priority = INDEX_NONE;
};
```

- [ ] UIExtensionSystem.h에 UIExtensionPoint 정의:

```cpp
USTRUCT(BlueprintType)
struct FUIExtensionRequest
{
    GENERATED_BODY()
public:
    /** UIExtensionPoint와 연동될 Extension */
    UPROPERTY(EditAnywhere, BlueprintReadOnly)
    FUIExtensionHandle ExtenionHandle;

    /** Extension의 Slot GameplayTag */
    UPROPERTY(EditAnywhere, BlueprintReadOnly)
    FGameplayTag ExtensionPointTag;

    /** WidgetClass로 FUIExtension과 같다 */
    UPROPERTY(EditAnywhere, BlueprintReadOnly)
    TObjectPtr<UObject> Data = nullptr;

    /** FUIExtension의 ContextObject를 전달받음 (UCommonLocalPlayer, UModularPlayerState, ... == UIExtension을 실행한 Instigator) */
    UPROPERTY(EditAnyhwere, BlueprintReadOnly)
    TObjectPtr<UObject> ContextObject = nullptr;

    UPROPERTY(EditAnywhere, BlueprintReadOnly)
    int32 Priority = INDEX_NONE;
};

DECLARE_DELEGATE(FExtendExtensionPointDelegate, EUIExtensionAction Action, const FUIExtensionRequest& Request);

struct FUIExtensionPoint : TSharedFromThis<FUIExtensionPoint>
{
public:
    /** UIExtension의 Slot GameplayTag */
    FGameplayTag ExtensionPointTag;

    /** UIExtension을 생성/제거한 Instigator(주체) */
    TWeakObjectPtr<UObject> ContextObject;

    /** UIExtensionPointWidget에 허용된 Widget Class -> UIExtensionPointWidget's DataClasses */
    TArray<UClass*> AllowedDataClasses;

    /** Widget을 ExtensionPointWidget에 연결하기 위한 Callback 함수 */
    FExtendExtensionPointDelegate Callback;
    EUIExtensionPointMatch ExtensionPointTagMatchType = EUIExtensionPointMatch::ExactMatch;
};

USTRUCT(BlueprintType)
struct UIEXTENSION_API FUIExtensionPointHandle
{
    GENERATED_BODY()
public:
    FUIExtensionPointHandle() {}
    FUIExtensionPointHandle(UUIExtensionSubsystem* InExtensionSource, const TSharedPtr<FUIExtensionPoint>& InDataPtr)
        : ExtensionSource(InExtensionSource), DataPtr(InDataPtr)
    {}

    TWeakObjectPtr<UUIExtensionSubsystem> ExtensionSource;
    TSharedPtr<FUIExtensionPoint> DataPtr;
};
```

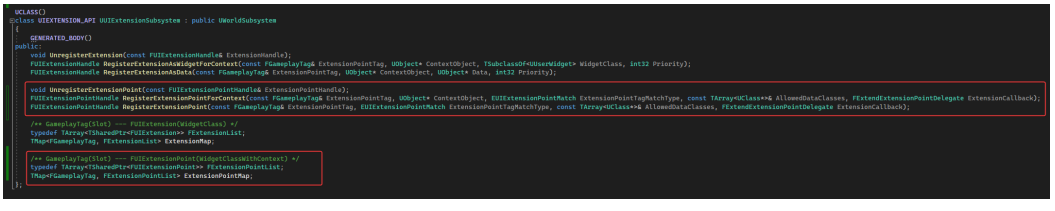
- [ ] UIExtensionPointWidget.h/.cpp:
  - [ ] FUIExtensionHandle의 GetTypeHash()를 추가하자 (TMap 사용)

```cpp
USTRUCT(BlueprintType)
struct UIEXTENSION_API FUIExtensionHandle
{
    GENERATED_BODY()
public:
    FUIExtensionHandle() {}
    FUIExtensionHandle(UUIExtensionSubsystem* InExtensionSource, const TSharedPtr<FUIExtension>& InDataPtr)
        : ExtensionSource(InExtensionSource)
        , DataPtr(InDataPtr)
    {}

    void Unregister();
    bool IsValid() const { return DataPtr.IsValid(); }
    bool operator==(const FUIExtensionHandle& Other) const { return DataPtr == Other.DataPtr; }
    bool operator!=(const FUIExtensionHandle& Other) const { return !operator==(Other); }

    friend FORCEINLINE uint32 GetTypeHash(FUIExtensionHandle Handle)
    {
        return PointerHash(Handle.DataPtr.Get());
    }

    friend class UUIExtensionSubsystem;
    TWeakObjectPtr<UUIExtensionSubsystem> ExtensionSource;
    TSharedPtr<FUIExtension> DataPtr;
};
```

☐ UIExtensionSystem에 UIExtensionPoint 관리 로직 추가:

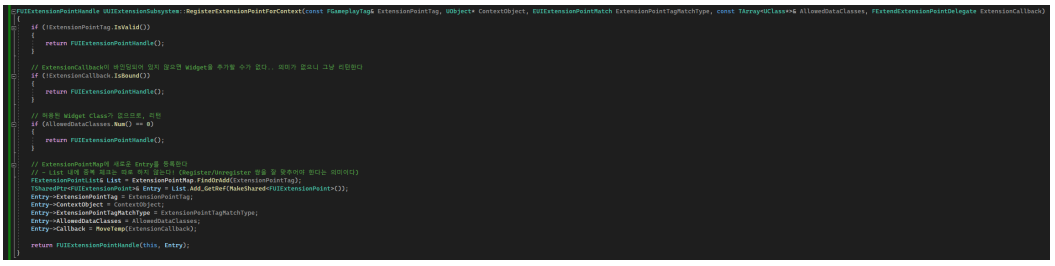
```cpp
UENUM(BlueprintType)
enum class EUIExtensionAction : uint8
{
    Added,
    Removed,
};

USTRUCT(BlueprintType)
struct FUIExtensionRequest
{
    GENERATED_BODY()
public:
    /** UIExtensionPoint와 연동될 Extension */
    UPROPERTY(EditAnywhere, BlueprintReadOnly)
    FUIExtensionHandle ExtenionHandle;

    /** Extension의 Slot GameplayTag */
    UPROPERTY(EditAnywhere, BlueprintReadOnly)
    FGameplayTag ExtensionPointTag;

    /** WidgetClass로 FUIExtension과 같다 */
    UPROPERTY(EditAnywhere, BlueprintReadOnly)
    TObjectPtr<UObject> Data = nullptr;

    /** FUIExtension의 ContextObject를 전달받음 (UCommonLocalPlayer, UModularPlayerState, ... == UIExtension을 실행한 Instigator) */
    UPROPERTY(EditAnywhere, BlueprintReadOnly)
    TObjectPtr<UObject> ContextObject = nullptr;

    UPROPERTY(EditAnywhere, BlueprintReadOnly)
    int32 Priority = INDEX_NONE;
};

DECLARE_DELEGATE_TwoParams(FExtendExtensionPointDelegate, EUIExtensionAction Action, const FUIExtensionRequest& Request);

struct FUIExtensionPoint : TSharedFromThis<FUIExtensionPoint>
{
public:
    /** UIExtension의 Slot GameplayTag */
    FGameplayTag ExtensionPointTag;

    /** UIExtension을 생성/제거한 Instigator(주체) */
    TWeakObjectPtr<UObject> ContextObject;

    /** UIExtensionPointWidget에 허용된 Widget Class -> UIExtensionPointWidget's DataClasses */
    TArray<UClass*> AllowedDataClasses;

    /** Widget을 ExtensionPointWidget에 연결하기 위한 Callback 함수 */
    FExtendExtensionPointDelegate Callback;
    EUIExtensionPointMatch ExtensionPointTagMatchType = EUIExtensionPointMatch::ExactMatch;
};

USTRUCT(BlueprintType)
struct UIEXTENSION_API FUIExtensionPointHandle
{
    GENERATED_BODY()
public:
    FUIExtensionPointHandle() {}
    FUIExtensionPointHandle(UUIExtensionSubsystem* InExtensionSource, const TSharedPtr<FUIExtensionPoint>& InDataPtr)
        : ExtensionSource(InExtensionSource), DataPtr(InDataPtr)
    {}

    TWeakObjectPtr<UUIExtensionSubsystem> ExtensionSource;
    TSharedPtr<FUIExtensionPoint> DataPtr;
};
```

☐ UIExtensionSystem에 UIExtensionPoint 관련 인터페이스와 멤버 변수를 추가하자:



☐ RegisterExtensionPoint()

- [ ] RegisterExtensionPointForContext()



- [ ] UnregisterExternalPoint()

```cpp
void UUIExtensionSubsystem::UnregisterExtensionPoint(const FUIExtensionPointHandle& ExtensionPointHandle)
{
    // 이전 UnregisterExtension()과 거의 흡사하니 스킵하장
    if (ExtensionPointHandle.IsValid())
    {
        check(ExtensionPointHandle.ExtensionSource == this);

        const TSharedPtr<FUIExtensionPoint> ExtensionPoint = ExtensionPointHandle.DataPtr;
        if (FExtensionPointList* ListPtr = ExtensionPointMap.Find(ExtensionPoint->ExtensionPointTag))
        {
            ListPtr->RemoveSwap(ExtensionPoint);
            if (ListPtr->Num() == 0)
            {
                ExtensionPointMap.Remove(ExtensionPoint->ExtensionPointTag);
            }
        }
    }
}
```

- [ ] FUIExtensionPointHandle::IsValid()

```cpp
USTRUCT(BlueprintType)
struct UIEXTENSION_API FUIExtensionPointHandle
{
    GENERATED_BODY()
public:
    FUIExtensionPointHandle() {}
    FUIExtensionPointHandle(UUIExtensionSubsystem* InExtensionSource, const TSharedPtr<FUIExtensionPoint>& InDataPtr)
        : ExtensionSource(InExtensionSource), DataPtr(InDataPtr)
    {}

    bool IsValid() const { return DataPtr.IsValid(); }

    TWeakObjectPtr<UUIExtensionSubsystem> ExtensionSource;
    TSharedPtr<FUIExtensionPoint> DataPtr;
};
```

- [ ] FUIExtensionHandle과 마찬가지로 아래의 메서드를 추가해주자:
  - [ ] operator==, operator≠:

```cpp
USTRUCT(BlueprintType)
struct UIEXTENSION_API FUIExtensionPointHandle
{
    GENERATED_BODY()
public:
    FUIExtensionPointHandle() {}
    FUIExtensionPointHandle(UUIExtensionSubsystem* InExtensionSource, const TSharedPtr<FUIExtensionPoint>& InDataPtr)
        : ExtensionSource(InExtensionSource), DataPtr(InDataPtr)
    {}

    bool IsValid() const { return DataPtr.IsValid(); }
    bool operator==(const FUIExtensionPointHandle& Other) const { return DataPtr == Other.DataPtr; }
    bool operator!=(const FUIExtensionPointHandle& Other) const { return !operator==(Other); }

    TWeakObjectPtr<UUIExtensionSubsystem> ExtensionSource;
    TSharedPtr<FUIExtensionPoint> DataPtr;
};
```

- [ ] Unregister()

```
USTRUCT(BlueprintType)
struct UIEXTENSION_API FUIExtensionPointHandle
{
    GENERATED_BODY()
public:
    FUIExtensionPointHandle() {}
    FUIExtensionPointHandle(UUIExtensionSubsystem* InExtensionSource, const TSharedPtr<FUIExtensionPoint>& InDataPtr)
        : ExtensionSource(InExtensionSource), DataPtr(InDataPtr)
    {}

    void Unregister();
    bool IsValid() const { return DataPtr.IsValid(); }
    bool operator==(const FUIExtensionPointHandle& Other) const { return DataPtr == Other.DataPtr; }
    bool operator!=(const FUIExtensionPointHandle& Other) const { return !operator==(Other); }

    TWeakObjectPtr<UUIExtensionSubsystem> ExtensionSource;
    TSharedPtr<FUIExtensionPoint> DataPtr;
};
```

```
void FUIExtensionPointHandle::Unregister()
{
    if (UUIExtensionSubsystem* ExtensionSourcePtr = ExtensionSource.Get())
    {
        ExtensionSourcePtr->UnregisterExtensionPoint(*this);
    }
}
```

- [ ] UIExtension과 UIExtensionPoint **관리 싱크를 위한** 메서드 추가:
  - [ ] NotifyExtensionPointOfExtensions()

NotifyExtensionPointsOfExtension 정의:

```
UCLASS()
class UIEXTENSION_API UUIExtensionSubsystem : public UWorldSubsystem
{
    GENERATED_BODY()
public:
    void UnregisterExtension(const FUIExtensionHandle& ExtensionHandle);
    FUIExtensionHandle RegisterExtensionAsWidgetForContext(const FGameplayTag& ExtensionPointTag, UObject* ContextObject, TSubclassOf<UUserWidget> W
    FUIExtensionHandle RegisterExtensionAsData(const FGameplayTag& ExtensionPointTag, UObject* ContextObject, UObject* Data, int32 Priority);

    void UnregisterExtensionPoint(const FUIExtensionPointHandle& ExtensionPointHandle);
    FUIExtensionPointHandle RegisterExtensionPointForContext(const FGameplayTag& ExtensionPointTag, UObject* ContextObject, EUIExtensionPointMatch E
    FUIExtensionPointHandle RegisterExtensionPoint(const FGameplayTag& ExtensionPointTag, EUIExtensionPointMatch ExtensionPointTagMatchType, const TA

    /** ExtensionPoint --(Broadcast)--> Extensions [ExtensionPointrk 추가/제거 되었을 경우, Extension에 알림: 참고로 Added만 함] */
    void NotifyExtensionPointOfExtensions(TSharedPtr<FUIExtensionPoint>& Extension);
    /** Extension --(Broadcast) --> ExtensionPoints [Extension이 추가/제거 되었을 경우, Extension Points에 알림] */
    void NotifyExtensionPointsOfExtension(EUIExtensionAction Action, TSharedPtr<FUIExtension>& Extension);

    /** GameplayTag(Slot) --- FUIExtension(WidgetClass) */
    typedef TArray<TSharedPtr<FUIExtension>> FExtensionList;
    TMap<FGameplayTag, FExtensionList> ExtensionMap;

    /** GameplayTag(Slot) --- FUIExtensionPoint(WidgetClassWithContext) */
    typedef TArray<TSharedPtr<FUIExtensionPoint>> FExtensionPointList;
    TMap<FGameplayTag, FExtensionPointList> ExtensionPointMap;
};
```

- [ ] NotifyExtensionPointOfExtensions():

```cpp
void UUIExtensionSubsystem::NotifyExtensionPointOfExtensions(TSharedPtr<FUIExtensionPoint>& ExtensionPoint)
{
    // 헷갈리지 말장: 현재 ExtensionPoint 변화점이 왔다!

    // ExtensionPoint의 ExtensionPointTag(Slot)을 순회하자
    for (FGameplayTag Tag = ExtensionPoint->ExtensionPointTag; Tag.IsValid(); Tag = Tag.RequestDirectParent())
    {
        // ExtensionMap을 통해 ExtensionPointTag에 매칭되는 ExtensionList를 가져오자
        if (const FExtensionList* ListPtr = ExtensionMap.Find(Tag))
        {
            // ExtensionList를 순회하며!
            FExtensionList ExtensionArray(*ListPtr);
            for (const TSharedPtr<FUIExtension>& Extension : ExtensionArray)
            {
                // 현재 ***ExtenionPoint***가 순회하는 Extension에 허용되는지 확인
                if (ExtensionPoint->DoesExtensionPassContract(Extension.Get()))
                {
                    // UIExtensionRequest를 만들고, ExtensionPoint의 UIExtensionPointWidget의 Add를 실행
                    FUIExtensionRequest Request = CreateExtensionRequest(Extension);
                    ExtensionPoint->Callback.ExecuteIfBound(EUIExtensionAction::Added, Request);
                }
            }
        }

        // ExactMatch라면 굳이 GameplayTag의 Parent를 순회할 필요가 없다
        if (ExtensionPoint->ExtensionPointTagMatchType == EUIExtensionPointMatch::ExactMatch)
        {
            break;
        }
    }
}
```

☐ UIExtensionPoint::DoesExtensionPassContract():

```cpp
bool FUIExtensionPoint::DoesExtensionPassContract(const FUIExtension* Extension) const
{
    if (UObject* DataPtr = Extension->Data)
    {
        const bool bMatchContext =
            // ContextObject와 Extension->ContextObject가 둘다 nullptr로 세팅되었는지?
            (ContextObject.IsExplicitlyNull() && Extension->ContextObject.IsExplicitlyNull()) ||
            // 아님 실제 ContextObject와 Extension->ContextObject가 같은지? (LocalPlayer? PlayerState?)
            (ContextObject == Extension->ContextObject);

        if (bMatchContext)
        {
            // DataClass (WidgetClass 추출)
            const UClass* DataClass = DataPtr->IsA(UClass::StaticClass()) ? Cast<UClass>(DataPtr) : DataPtr->GetClass();
            for (const UClass* AllowedDataClass : AllowedDataClasses)
            {
                // AllowedDataClasses를 순회하며 Child/Interface인가 확인
                if (DataClass->IsChildOf(AllowedDataClass) || DataClass->ImplementsInterface(AllowedDataClass))
                {
                    return true;
                }
            }
        }
    }
    return false;
}
```

☐ UIExtensionSubsystem::CreateExtensionRequest():

```cpp
FUIExtensionRequest UUIExtensionSubsystem::CreateExtensionRequest(const TSharedPtr<FUIExtension>& Extension)
{
    FUIExtensionRequest Request;
    Request.ExtensionHandle = FUIExtensionHandle(this, Extension);
    Request.ExtensionPointTag = Extension->ExtensionPointTag;
    Request.Priority = Extension->Priority;
    Request.Data = Extension->Data;
    Request.ContextObject = Extension->ContextObject.Get();
    return Request;
}
```

☐ NotifyExtensionPointOfExtensions() 추가:

☐ NotifyExtensionPointsOfExtension():

```cpp
void UUIExtensionSubsystem::NotifyExtensionPointsOfExtension(EUIExtensionAction Action, TSharedPtr<FUIExtension>& Extension)
{
    // 헷갈리지 말자: 현재 Extension 변경점이 왔다!

    // bOnInitialTag를 활용하는 이유? ExactMatch 로직을 위해서!
    bool bOnInitialTag = true;
    for (FGameplayTag Tag = Extension->ExtensionPointTag; Tag.IsValid(); Tag = Tag.RequestDirectParent())
    {
        if (const FExtensionPointList* ListPtr = ExtensionPointMap.Find(Tag))
        {
            // ExtensionPoint를 순회
            FExtensionPointList ExtensionPointArray(*ListPtr);
            for (const TSharedPtr<FUIExtensionPoint>& ExtensionPoint : ExtensionPointArray)
            {
                /** ExactMatch용 + PartialMatch용 */
                if (bOnInitialTag ||
                    (ExtensionPoint->ExtensionPointTagMatchType == EUIExtensionPointMatch::PartialMatch))
                {
                    if (ExtensionPoint->DoesExtensionPassContract(Extension.Get()))
                    {
                        FUIExtensionRequest Request = CreateExtensionRequest(Extension);
                        ExtensionPoint->Callback.ExecuteIfBound(Action, Request);
                    }
                }
            }
        }

        // 흠.. 근데 왜 다 순회할까?
        // - ExtensionPointTagMatchType이 UIExtensionPoint 안에 있으므로!
        bOnInitialTag = false;
    }
}
```

☐ NotifyExtensionPointsOfExtension() 추가:

- Added:

```cpp
FUIExtensionHandle UUIExtensionSubsystem::RegisterExtensionAsData(const FGameplayTag& ExtensionPointTag, UObject* ContextObject, UObject* Data, int32 Priority)
{
    // ExtensionPointTag(Slot)이 Invalid
    if (!ExtensionPointTag.IsValid())
    {
        return FUIExtensionHandle();
    }

    // WidgetClass가 없으면
    if (!Data)
    {
        return FUIExtensionHandle();
    }

    // ExtensionPointTag를 기반하여, ExtensionList를 추출
    FExtensionList& List = ExtensionMap.FindOrAdd(ExtensionPointTag);

    // 새로운 UIExtension을 추가 진행
    TSharedPtr<FUIExtension>& Entry = List.Add_GetRef(MakeShared<FUIExtension>());
    Entry->ExtensionPointTag = ExtensionPointTag;
    Entry->ContextObject = ContextObject;
    Entry->Data = Data;
    Entry->Priority = Priority;

    // Extension이 추가되었으니 Added로 NotifyExtensionPointsOfExtension 실행
    NotifyExtensionPointsOfExtension(EUIExtensionAction::Added, Entry);

    return FUIExtensionHandle(this, Entry);
}
```

- Removed:

```cpp
void UUIExtensionSubsystem::UnregisterExtension(const FUIExtensionHandle& ExtensionHandle)
{
    if (ExtensionHandle.IsValid())
    {
        // 반드시 해당 ExtensionHandle이 UUIExtensionSubsystem과 같은지 확인해야 함!
        checkf(ExtensionHandle.ExtensionSource == this, TEXT("Trying to unregister an extension that's not from this extension subsystem"));

        TSharedPtr<FUIExtension> Extension = ExtensionHandle.DataPtr;

        // Extension의 PointTag를 통해 ExtensionMap에서 해당 Slot에 있는지 찾아서 제거
        if (FExtensionList* ListPtr = ExtensionMap.Find(Extension->ExtensionPointTag))
        {
            // Extension 제거, NotifyExtensionPointsOfExtension 호출
            NotifyExtensionPointsOfExtension(EUIExtensionAction::Removed, Extension);

            ListPtr->RemoveSwap(Extension);
            if (ListPtr->Num() == 0)
            {
                // Num() == 0이면 Map에서도 제거 진행
                ExtensionMap.Remove(Extension->ExtensionPointTag);
            }
        }
    }
}
```

- [ ] UUIExtensionPointWidget::RebuildWidget():
  - [ ] ResetExtensionPoint():

```cpp
void UUIExtensionPointWidget::ResetExtensionPoint()
{
    // UDynamicEntryBoxBase::ResetInternal() 호출
    ResetInternal();

    // UUserWidget은 알아서 GC 될 것이니 그냥 컨테이너만 Reset()
    ExtensionMapping.Reset();

    // ExtensionPointHandle을 순회하며, UUIExtensionSystem에서 제거(Unregister)
    for (FUIExtensionPointHandle& Handle : ExtensionPointHandles)
    {
        Handle.Unregister();
    }
    ExtensionPointHandles.Reset();
}
```

  - [ ] RegisterExternalPoint():

```cpp
void UUIExtensionPointWidget::RegisterExtensionPoint()
{
    if (UUIExtensionSubsystem* ExtensionSubsystem = GetWorld()->GetSubsystem<UUIExtensionSubsystem>())
    {
        // UUserWidget을 포함하여, AllowedDataClasses를 생성
        TArray<UClass*> AllowedDataClasses;
        AllowedDataClasses.Add(UUserWidget::StaticClass());
        AllowedDataClasses.Append(DataClasses);

        // nullptr용 (ContextObject) ExtensionPoint 생성
        ExtensionPointHandles.Add(ExtensionSubsystem->RegisterExtensionPoint(
            ExtensionPointTag, ExtensionPointTagMatch, AllowedDataClasses,
            FExtendExtensionPointDelegate::CreateUObject(this, &ThisClass::OnAddOrRemoveExtension)
        ));

        // LocalPlayer용 (ContextObject) ExtensionPoint 생성
        ExtensionPointHandles.Add(ExtensionSubsystem->RegisterExtensionPointForContext(
            ExtensionPointTag, GetOwningLocalPlayer(), ExtensionPointTagMatch, AllowedDataClasses,
            FExtendExtensionPointDelegate::CreateUObject(this, &ThisClass::OnAddOrRemoveExtension)
        ));
    }
}
```

  - [ ] RegisterExtensionPointForPlayerState():

```cpp
void UUIExtensionPointWidget::RegisterExtensionPointForPlayerState(UCommonLocalPlayer* LocalPlayer, APlayerState* PlayerState)
{
    // RegisterExtensionPoint랑 거의 같으니 생략
    if (UUIExtensionSubsystem* ExtensionSubsystem = GetWorld()->GetSubsystem<UUIExtensionSubsystem>())
    {
        TArray<UClass*> AllowedDataClasses;
        AllowedDataClasses.Add(UUserWidget::StaticClass());
        AllowedDataClasses.Append(DataClasses);

        ExtensionPointHandles.Add(ExtensionSubsystem->RegisterExtensionPointForContext(
            ExtensionPointTag, PlayerState, ExtensionPointTagMatch, AllowedDataClasses,
            FExtendExtensionPointDelegate::CreateUObject(this, &ThisClass::OnAddOrRemoveExtension)
        ));
    }
}
```

☐ UCommonLocalPlayer::OnPlayerStateSet 추가:

```cpp
UCLASS()
class COMMONGAME_API UCommonLocalPlayer : public ULocalPlayer
{
    GENERATED_BODY()
public:
    /** 여기서 주목해보면 좋을 점들은 패턴이다: 우리는 여기서 FObjectInitializer::Get()과 같이 넘기지 않았다 */
    UCommonLocalPlayer();

    /** player controller가 local player에 할당(assign)되었을 경우 실행할 Delegate */
    DECLARE_MULTICAST_DELEGATE_TwoParams(FPlayerControllerSetDelegate, UCommonLocalPlayer* LocalPlayer, APlayerController* PlayerController)
    FPlayerControllerSetDelegate OnPlayerControllerSet;

    /** player state가 local player에 할당(assign)되었을 경우 실행할 Delegate */
    DECLARE_MULTICAST_DELEGATE_TwoParams(FPlayerStateSetDelegate, UCommonLocalPlayer* LocalPlayer, APlayerState* PlayerState)
    FPlayerStateSetDelegate OnPlayerStateSet;

    FDelegateHandle CallAndRegister_OnPlayerStateSet(FPlayerStateSetDelegate::FDelegate Delegate);
};
```

```cpp
FDelegateHandle UCommonLocalPlayer::CallAndRegister_OnPlayerStateSet(FPlayerStateSetDelegate::FDelegate Delegate)
{
    APlayerController* PC = GetPlayerController(GetWorld());
    APlayerState* PlayerState = PC ? PC->PlayerState : nullptr;
    if (PlayerState)
    {
        // PlayerState가 설정되어 있으면 바로 호출
        Delegate.Execute(this, PlayerState);
    }
    // OnPlayerStateSet에 등록
    return OnPlayerStateSet.Add(Delegate);
}
```

☐ ACommonPlayerController::ReceivePlayer():

```cpp
void ACommonPlayerController::ReceivedPlayer()
{
    Super::ReceivedPlayer();

    // PlayerController가 LocalPlayer에 붙으면 활성화되는 이벤트가 ReceivedPlayer()이다
    if (UCommonLocalPlayer* LocalPlayer = Cast<UCommonLocalPlayer>(Player))
    {
        LocalPlayer->OnPlayerControllerSet.Broadcast(LocalPlayer, this);
        if (PlayerState)
        {
            LocalPlayer->OnPlayerStateSet.Broadcast(LocalPlayer, PlayerState);
        }
    }
}
```

☐ RebuildWIdget():

```cpp
TSharedRef<SWidget> UUIExtensionPointWidget::RebuildWidget()
{
    // 실제 InGame 랜더링될 때, 실행된다
    if (!IsDesignTime() && ExtensionPointTag.IsValid())
    {
        // UIExtensionPointWidget 내부 UDynamicEntryBoxBase::Child 모두 제거
        ResetExtensionPoint();

        // 다시 ExtensionPointWidget을 등록하기 위한 Delegate 및 Handle 추가
        RegisterExtensionPoint();

        // PlayerState 설정에 대한 Delegate를 통해 ExtensionPointWidget 추가를 위한 호출 확보
        FDelegateHandle Handle = GetOwningLocalPlayer<UCommonLocalPlayer>()->CallAndRegister_OnPlayerStateSet(
            UCommonLocalPlayer::FPlayerStateSetDelegate::FDelegate::CreateUObject(this, &UUIExtensionPointWidget::RegisterExtensionPointForPlayerState)
        );
    }

    // 여기는 UMG Editor에서 실행될때 실행되는 구간이다 (이를 IsDesignTime()으로 구분 가능)
    if (IsDesignTime())
    {
        auto GetExtensionPointText = [this]()
        {
            // ExtensionPoint
            // [ExtensionPointTag]
            // - 위와 같은 형태로 표현될거다
            return FText::Format(LOCTEXT("DesignTime_ExtensionPointLabel", "ExtensionPoint\n{0}"), FText::FromName(ExtensionPointTag.GetTagName()));
        };

        // MessageBox라고 하지만, 그냥 Editor HUD라고 생각하면 된다 (SOverlay)
        TSharedRef<SOverlay> MessageBox = SNew(SOverlay);
        MessageBox->AddSlot()
            .Padding(0.5f)
            .HAlign(HAlign_Center)
            .VAlign(VAlign_Center)
            [
                // 가운데 정렬로 GetExtensionPointText 넣어준다
                SNew(STextBlock)
                .Justification(ETextJustify::Center)
                .Text_Lambda(GetExtensionPointText)
            ];
        return MessageBox;
    }
    else
    {
        return Super::RebuildWidget();
    }
}
```

☐ UUIExtensionPointWidget::OnAddOrRemoveExtension():

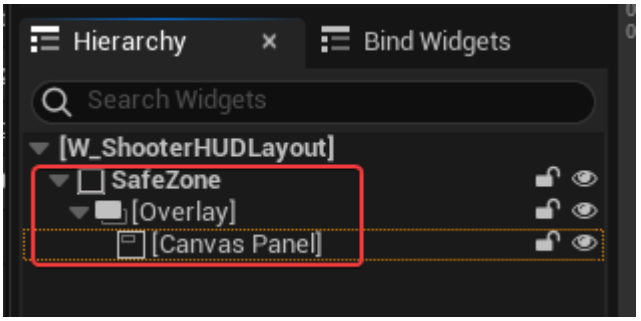
```cpp
void UUIExtensionPointWidget::OnAddOrRemoveExtension(EUIExtensionAction Action, const FUIExtensionRequest& Request)
{
    if (Action == EUIExtensionAction::Added)
    {
        UObject* Data = Request.Data;

        // Data는 앞서 이야기했듯이 WidgetClass이다
        TSubclassOf<UUserWidget> WidgetClass(Cast<UClass>(Data));
        if (WidgetClass)
        {
            // WidgetClass를 활용하여, UDynamicEntryBoxBase::CreateEntryInternal 호출로 Child Widget을 만든다
            UUserWidget* Widget = CreateEntryInternal(WidgetClass);

            // 제거할 경우, Tracking을 위해 ExtensionMapping 추가
            ExtensionMapping.Add(Request.ExtensionHandle, Widget);
        }
        // DataClasses 처리 (생략)
    }
    else
    {
        // ExtensionMapping을 활용하여, 안정하게 UDynamicEntryBoxBase::RemoveEntryInternal로 제거
        if (UUserWidget* Extension = ExtensionMapping.FindRef(Request.ExtensionHandle))
        {
            RemoveEntryInternal(Extension);
            ExtensionMapping.Remove(Request.ExtensionHandle);
        }
    }
}
```

- 해당 로직은 앞서 UUIExtensionSystem에서 보았듯이, NotifyExtensionPointOfExtensions()와 NotifyExtensionPointsOfExtension()에서 Callback으로 저장된 Delegate를 호출한다.

ExtensionPoint_Reticle

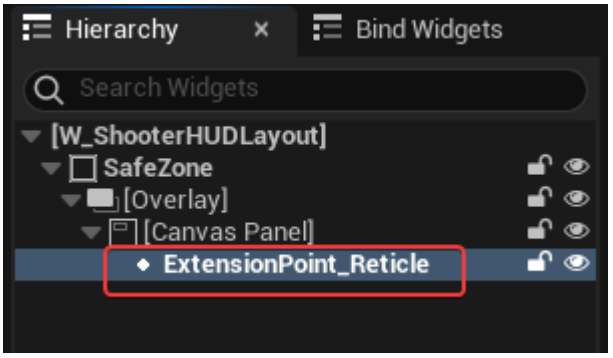
▼ **펼치기**

☐ W_ShooterHUDLayout 뼈대를 완성하자 (Reticle 위주로)

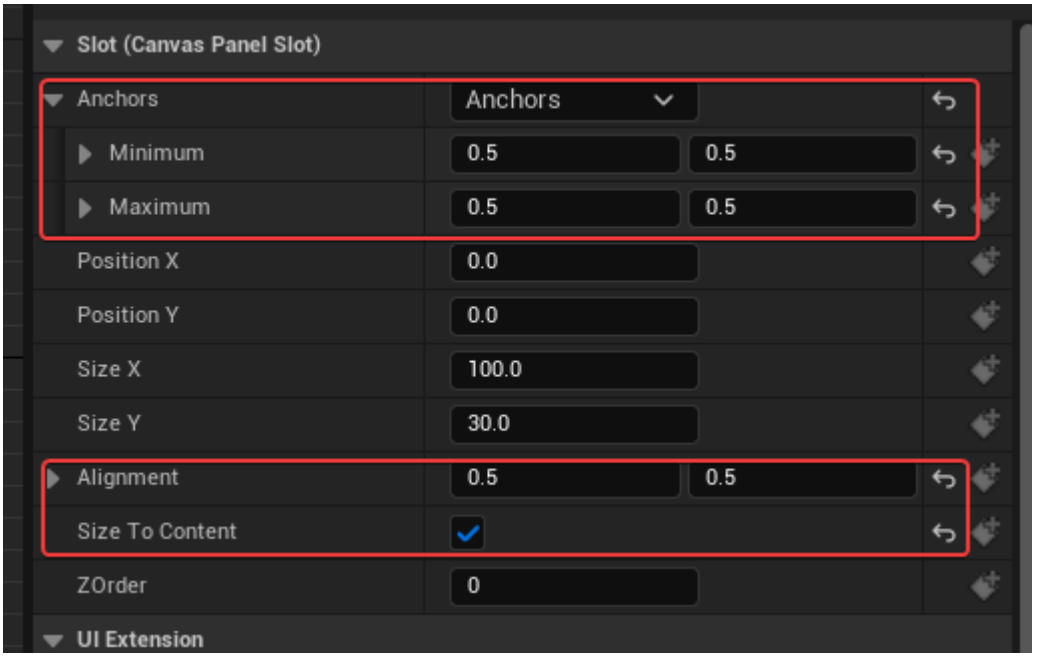


☐ Canvas Panel의 기본값 업데이트하자:

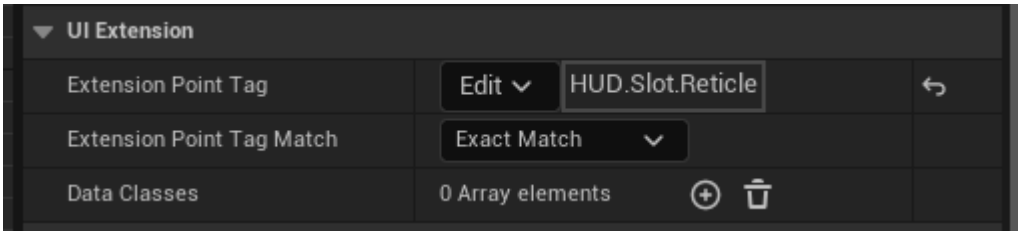
☐ UIExtension Point Widget 추가하자:



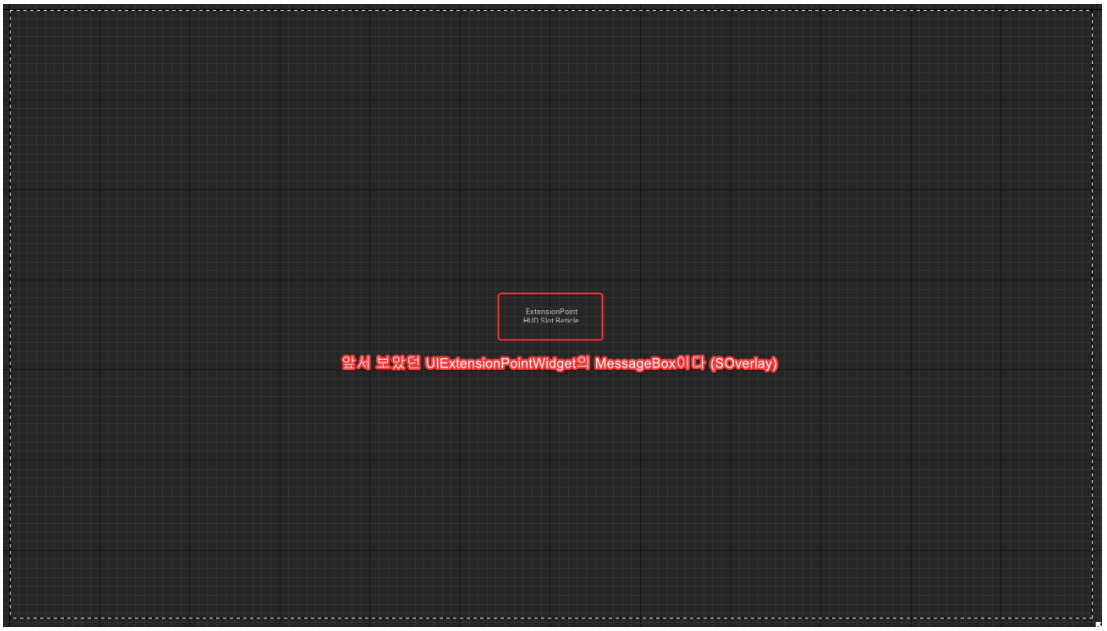
☐ 기본값 업데이트:



- [ ] UIExtensionPointTag 설정:



- [ ] 아래와 같은 형태가 된다:



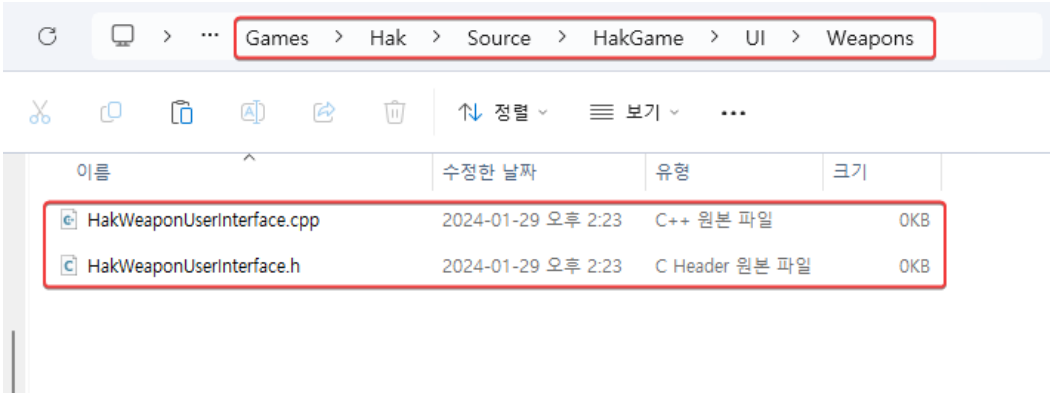
W_WeaponReticleHost

▼ 펼치기

- HUD.Slot.Reticle에 바인딩 할 Widget을 정의해야 한다:
  - 해당 Widget Asset은 W_WeaponReticleHost로 UHakWeaponUserInterface를 상속한다

☐ UHakWeaponUserInterface 정의:

☐ HakWeaponUserInterface 파일 생성:



☐ HakWeaponUserInterface.h/.cpp:

```h
#pragma once

#include "CommonUserWidget.h"
#include "UObject/ObjectPtr.h"
#include "UObject/UObjectGlobals.h"
#include "HakWeaponUserInterface.generated.h"

/** forward declaration */
class UHakWeaponInstance;

UCLASS()
class UHakWeaponUserInterface : public UCommonUserWidget
{
    GENERATED_BODY()
public:
    UHakWeaponUserInterface(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** 현재 장착된 WeaponInstance를 추적한다 (NativeTick을 활용하여 주기적 업데이트한다) */
    UPROPERTY(Transient)
    TObjectPtr<UHakWeaponInstance> CurrentInstance;
};
```

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

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

☐ HakWeaponUserInterface::NativeTick():

- Weapon의 변화를 Widget의 NativeTick()을 통해 주기적으로 체크한다

☐ OnWeaponChanged() 정의:

```cpp
#pragma once

#include "CommonUserWidget.h"
#include "UObject/ObjectPtr.h"
#include "UObject/UObjectGlobals.h"
#include "HakWeaponUserInterface.generated.h"

/** forward declaration */
class UHakWeaponInstance;

/**
 * 무기 특화 UI
 */
UCLASS()
class UHakWeaponUserInterface : public UCommonUserWidget
{
    GENERATED_BODY()
public:
    UHakWeaponUserInterface(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** Weapon 변경에 따른 BP Event */
    UFUNCTION(BlueprintImplementableEvent)
    void OnWeaponChanged(UHakWeaponInstance* OldWeapon, UHakWeaponInstance* NewWeapon);

    /**
     * UUserWidget's interface
     */
    virtual void NativeTick(const FGeometry& MyGeometry, float InDeltaTime) override;

    /** 현재 장착된 WeaponInstance를 추적한다 (NativeTick을 활용하여 주기적 업데이트한다) */
    UPROPERTY(Transient)
    TObjectPtr<UHakWeaponInstance> CurrentInstance;
};
```

☐ NativeTick():

```cpp
#include "HakWeaponUserInterface.h"
#include "HakGame/Equipment/HakEquipmentManagerComponent.h"
#include "HakGame/Weapons/HakWeaponInstance.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakWeaponUserInterface)

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

void UHakWeaponUserInterface::NativeTick(const FGeometry& MyGeometry, float InDeltaTime)
{
    Super::NativeTick(MyGeometry, InDeltaTime);

    // Pawn을 가져오고
    if (APawn* Pawn = GetOwningPlayerPawn())
    {
        // EquipmentManaterComponent를 활용하여, WeaponInstance를 가져오자
        if (UHakEquipmentManagerComponent* EquipmentManager = Pawn->FindComponentByClass<UHakEquipmentManagerComponent>())
        {
            if (UHakWeaponInstance* NewInstance = EquipmentManager->GetFirstInstanceOfType<UHakWeaponInstance>())
            {
                if (NewInstance != CurrentInstance && NewInstance->GetInstigator() != nullptr)
                {
                    // 새로 업데이트해주고, OnWeaponChanged 호출 진행
                    UHakWeaponInstance* OldWeapon = CurrentInstance;
                    CurrentInstance = NewInstance;
                    OnWeaponChanged(OldWeapon, CurrentInstance);
                }
            }
        }
    }
}
```

☐ HakEquipmentManagerComponent::GetFirstInstanceOfType():

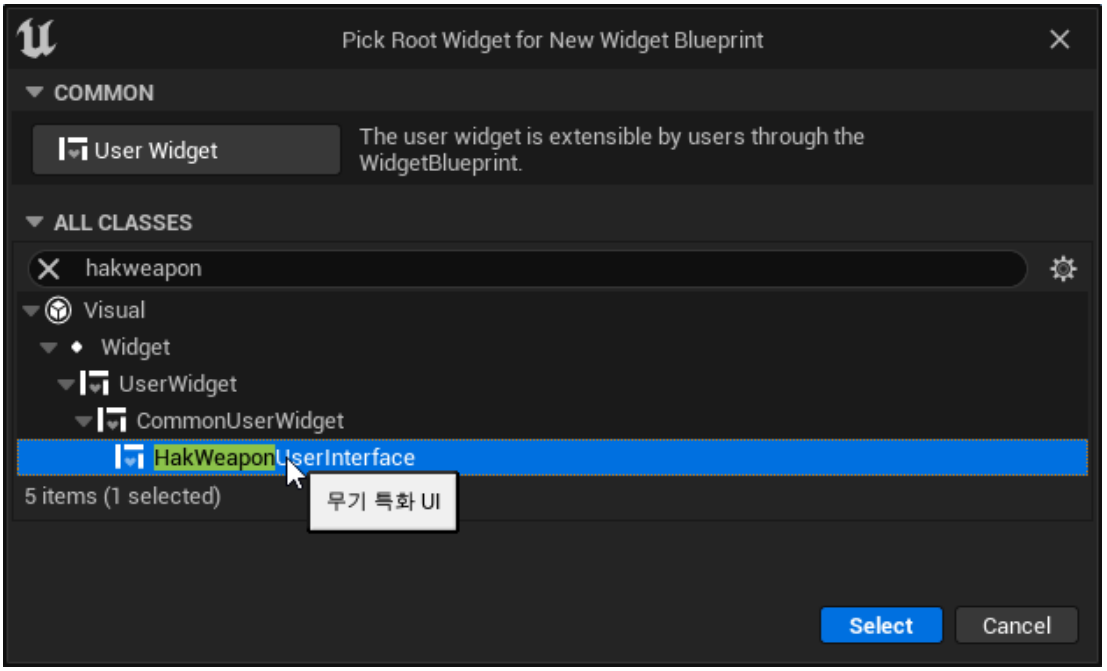
```cpp
/**
 * Pawn의 Component로서 장착물에 대한 관리를 담당한다
 */
UCLASS(BlueprintType)
class UHakEquipmentManagerComponent : public UPawnComponent
{
    GENERATED_BODY()
public:
    UHakEquipmentManagerComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UHakEquipmentInstance* EquipItem(TSubclassOf<UHakEquipmentDefinition> EquipmentDefinition);
    void UnequipItem(UHakEquipmentInstance* ItemInstance);

    UFUNCTION(BlueprintCallable)
    TArray<UHakEquipmentInstance*> GetEquipmentInstancesOfType(TSubclassOf<UHakEquipmentInstance> InstanceType) const;

    /** 장착물 중 처음 것을 반환 없으면 NULL */
    UHakEquipmentInstance* GetFirstInstanceOfType(TSubclassOf<UHakEquipmentInstance> InstanceType);

    template <typename T>
    T* GetFirstInstanceOfType()
    {
        return (T*)GetFirstInstanceOfType(T::StaticClass());
    }

    UPROPERTY()
    FHakEquipmentList EquipmentList;
};
```

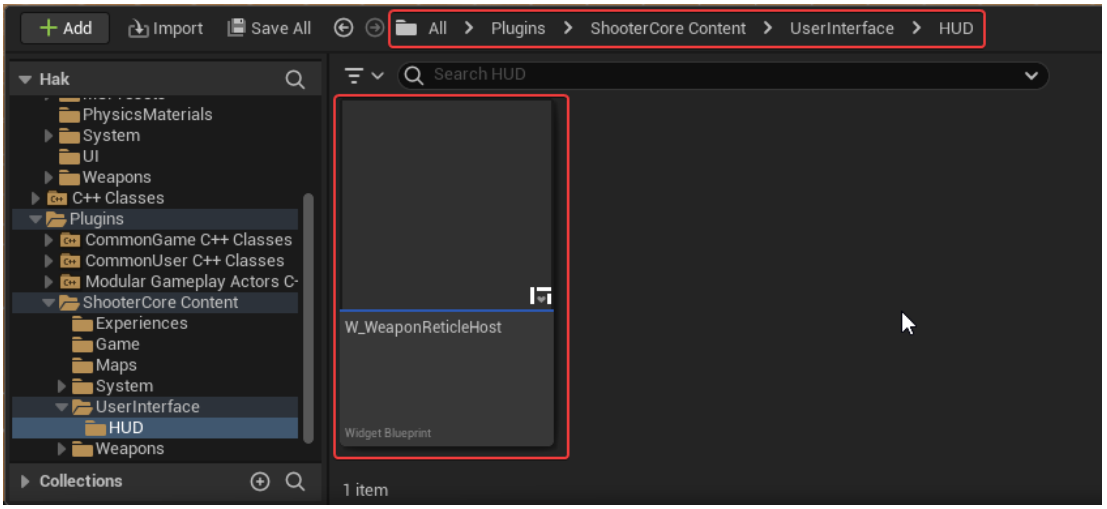
```cpp
UHakEquipmentInstance* UHakEquipmentManagerComponent::GetFirstInstanceOfType(TSubclassOf<UHakEquipmentInstance> InstanceType)
{
    for (FHakAppliedEquipmentEntry& Entry : EquipmentList.Entries)
    {
        if (UHakEquipmentInstance* Instance = Entry.Instance)
        {
            if (Instance->IsA(InstanceType))
            {
                return Instance;
            }
        }
    }
    return nullptr;
}
```

☐ HakEquipmentInstance::GetInstigator():

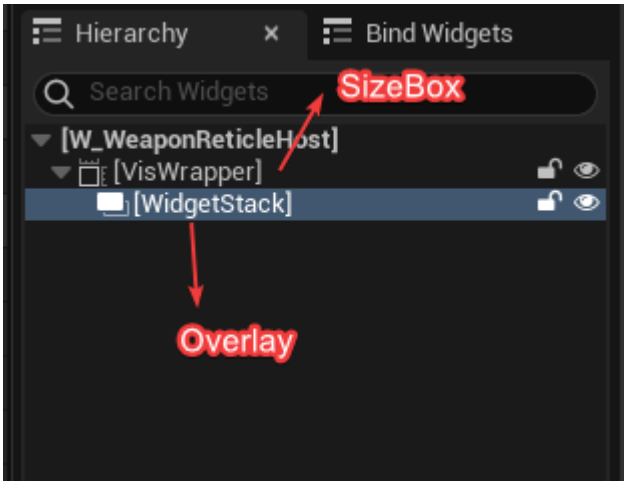
```cpp
UCLASS(BlueprintType, Blueprintable)
class UHakEquipmentInstance : public UObject
{
    GENERATED_BODY()
public:
    UHakEquipmentInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * Blueprint 정의를 위한 Equip/Unequip 함수
     */
    UFUNCTION(BlueprintImplementableEvent, Category = Equipment, meta = (DisplayName = "OnEquipped"))
    void K2_OnEquipped();

    UFUNCTION(BlueprintImplementableEvent, Category = Equipment, meta = (DisplayName = "OnUnequipped"))
    void K2_OnUnequipped();

    UFUNCTION(BlueprintPure, Category=Equipment)
    APawn* GetPawn() const;

    UFUNCTION(BlueprintPure, Category=Equipment)
    TArray<AActor*> GetSpawnedActors() const { return SpawnedActors; }

    UFUNCTION(BlueprintPure, Category = Equipment)
    UObject* GetInstigator() const { return Instigator; }

    /**
     * DeterminesOutputType은 C++ 정의에는 APawn* 반환하지만, BP에서는 PawnType에 따라 OutputType이 결정되도록 리다이렉트(Redirect)한다
     */
    UFUNCTION(BlueprintPure, Category=Equipment, meta=(DeterminesOutputType=PawnType))
    APawn* GetTypedPawn(TSubclassOf<APawn> PawnType) const;

    void SpawnEquipmentActors(const TArray<FHakEquipmentActorToSpawn>& ActorsToSpawn);
    void DestroyEquipmentActors();

    /**
     * interfaces
     */
    virtual void OnEquipped();
    virtual void OnUnequipped();

    /** 어떤 InventoryItemInstance에 의해 활성화되었는지 (추후, QuickBarComponent에서 보게 될것이다) */
    UPROPERTY()
    TObjectPtr<UObject> Instigator;

    /** HakEquipementDefinition에 맞게 Spawn된 Actor Instance들 */
    UPROPERTY()
    TArray<TObjectPtr<AActor>> SpawnedActors;
};
```

☐ W_WeaponReticleHost:

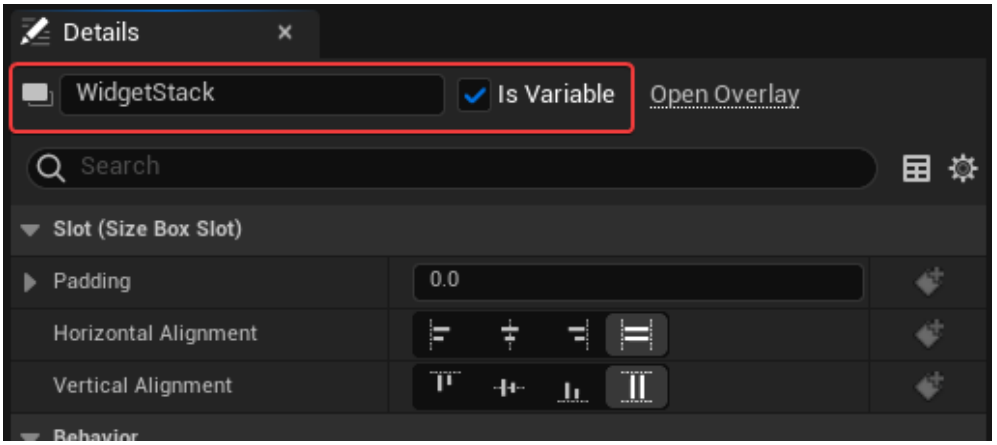




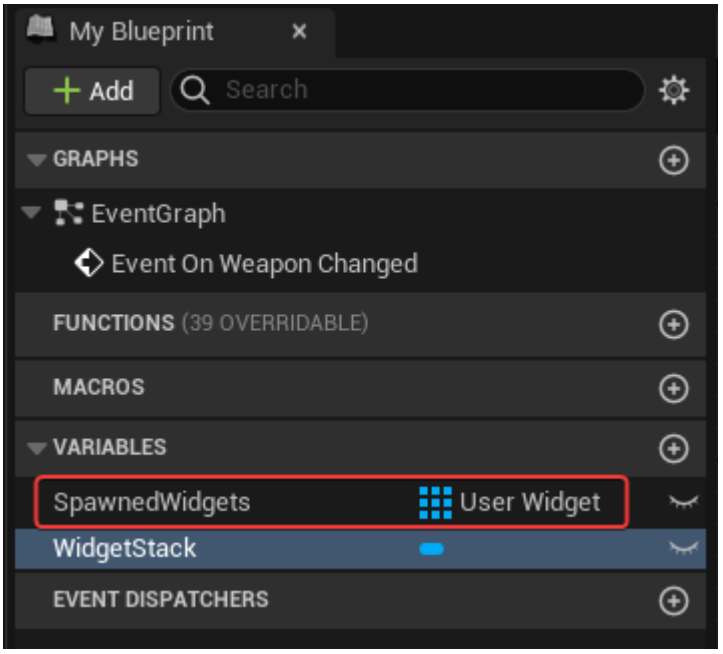
☐ W_WeaponReticleHost UI 레이아웃 완성:



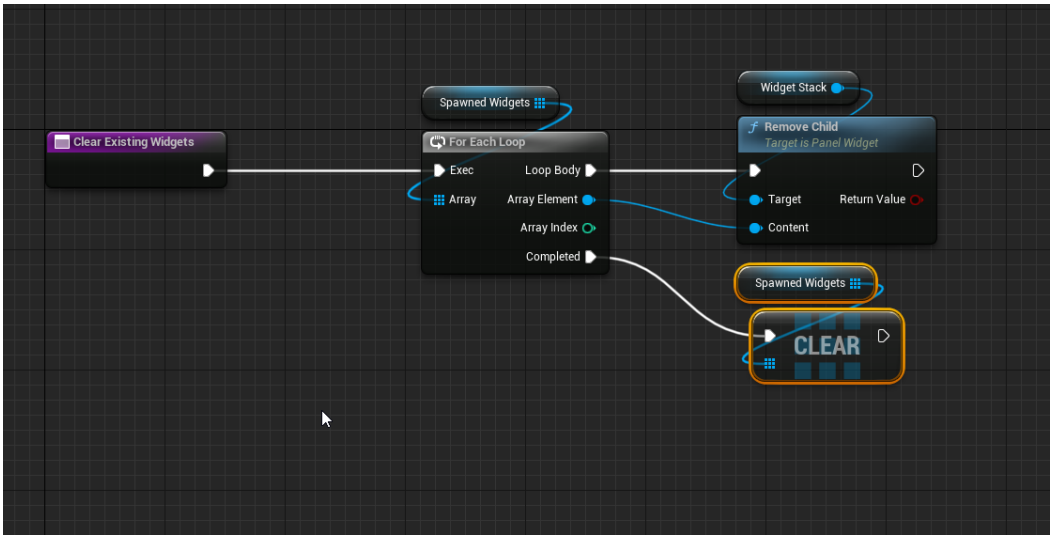
- [ ] WeaponUserInterface::OnWeaponChanged() BP 구현:
  - [ ] 변수 선언:
    - WidgetStack:



    - SpawnedWidgets:



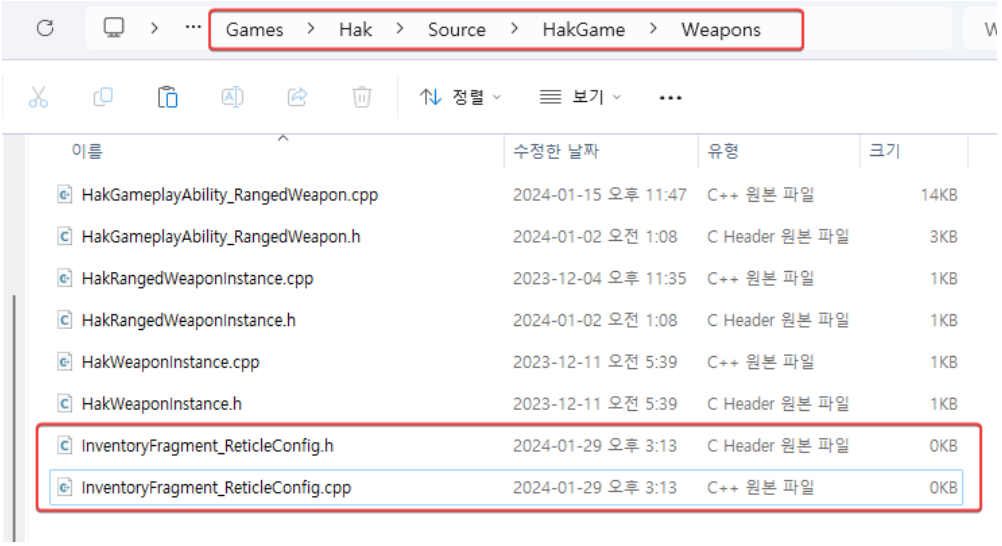
  - [ ] ClearExistingWidgets BP Function:



- [ ] HakInventoryItemInstance::FIndFragmentByClass BP화:

```cpp
/**
 * 해당 클래스는 Inventory Item의 인스턴스로 볼 수 있다
 */
UCLASS(BlueprintType)
class UHakInventoryItemInstance : public UObject
{
    GENERATED_BODY()
public:
    UHakInventoryItemInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UFUNCTION(BlueprintCallable, BlueprintPure=false, meta=(DeterminesOutputType=FragmentClass))
    const UHakInventoryItemFragment* FindFragmentByClass(TSubclassOf<UHakInventoryItemFragment> FragmentClass) const;

    template <typename ResultClass>
    const ResultClass* FindFragmentByClass() const
    {
        return (ResultClass*)FindFragmentByClass(ResultClass::StaticClass());
    }

    /** Inventory Item의 인스턴스에는 무엇으로 정의되었는지 메타 클래스인 HakInventoryItemDefinition을 들고 있다 */
    UPROPERTY()
    TSubclassOf<UHakInventoryItemDefinition> ItemDef;
};
```

- [ ] InventoryFragment_ReticleConfig 정의:
  - [ ] InventoryFragment_ReticleConfig 파일 생성:

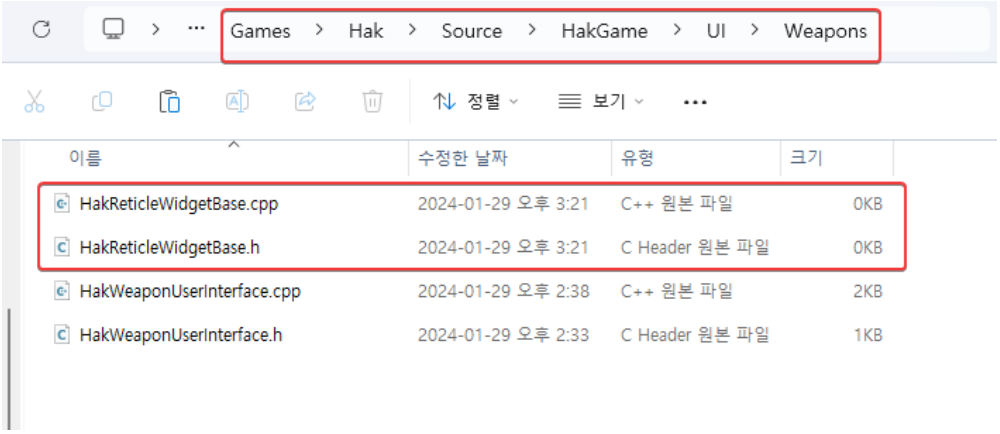


- [ ] InventoryFragment_Reticle.h/.cpp:

```cpp
#pragma once

#include "Containers/Array.h"
#include "Templates/SubclassOf.h"
#include "HakGame/Inventory/HakInventoryItemDefinition.h"
#include "InventoryFragment_ReticleConfig.generated.h"

/** forward declaration */
class UHakReticleWidgetBase;

UCLASS()
class UHakInventoryFragment_ReticleConfig : public UHakInventoryItemFragment
{
    GENERATED_BODY()
public:
    /** 무기에 결합된 ReticleWidget 정보를 가지고 있는 Fragment */
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Reticle)
    TArray<TSubclassOf<UHakReticleWidgetBase>> ReticleWidgets;
};
```

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

- [ ] HakReticleWidgetBase 정의:
  - [ ] HakReticleWidgetBase 파일 생성:



☐ HakReticleWidgetBase.h/.cpp:

```cpp
#pragma once

#include "CommonUserWidget.h"
#include "UObject/ObjectPtr.h"
#include "UObject/UObjectGlobals.h"
#include "HakReticleWidgetBase.generated.h"

/** forward declarations */
class UHakWeaponInstance;
class UHakInventoryItemInstance;

UCLASS(Abstract)
class UHakReticleWidgetBase : public UCommonUserWidget
{
    GENERATED_BODY()
public:
    UHakReticleWidgetBase(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * WeaponInstance/InventoryInstance를 상태 추적용으로 캐싱 목적
     */
    UPROPERTY(BlueprintReadOnly)
    TObjectPtr<UHakWeaponInstance> WeaponInstance;

    UPROPERTY(BlueprintReadOnly)
    TObjectPtr<UHakInventoryItemInstance> InventoryInstance;
};
```

```cpp
#include "HakReticleWidgetBase.h"
#include "HakGame/Weapons/HakWeaponInstance.h"
#include "HakGame/Inventory/HakInventoryItemInstance.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakReticleWidgetBase)

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

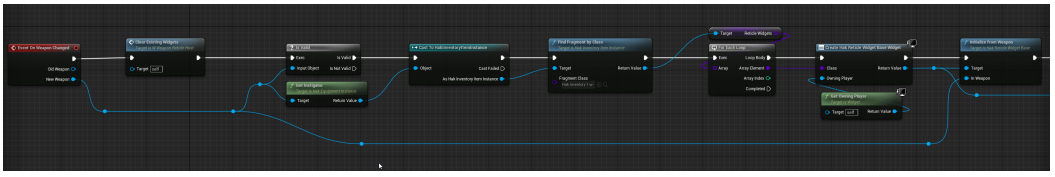
☐ InitializeFromWeapon()

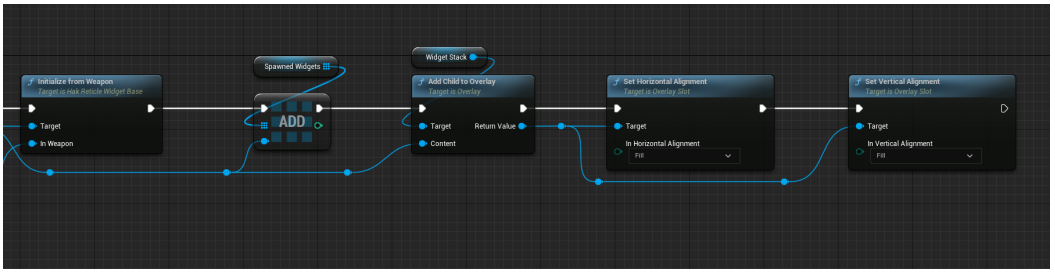
```cpp
UCLASS(Abstract)
class UHakReticleWidgetBase : public UCommonUserWidget
{
    GENERATED_BODY()
public:
    UHakReticleWidgetBase(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UFUNCTION(BlueprintCallable)
    void InitializeFromWeapon(UHakWeaponInstance* InWeapon);

    /**
     * WeaponInstance/InventoryInstance를 상태 추적용으로 캐싱 목적
     */
    UPROPERTY(BlueprintReadOnly)
    TObjectPtr<UHakWeaponInstance> WeaponInstance;

    UPROPERTY(BlueprintReadOnly)
    TObjectPtr<UHakInventoryItemInsta
};
```

(field) TObjectPtr<UHakWeaponInstance> UHakReticleWidgetBase::WeaponInstance
Search Online

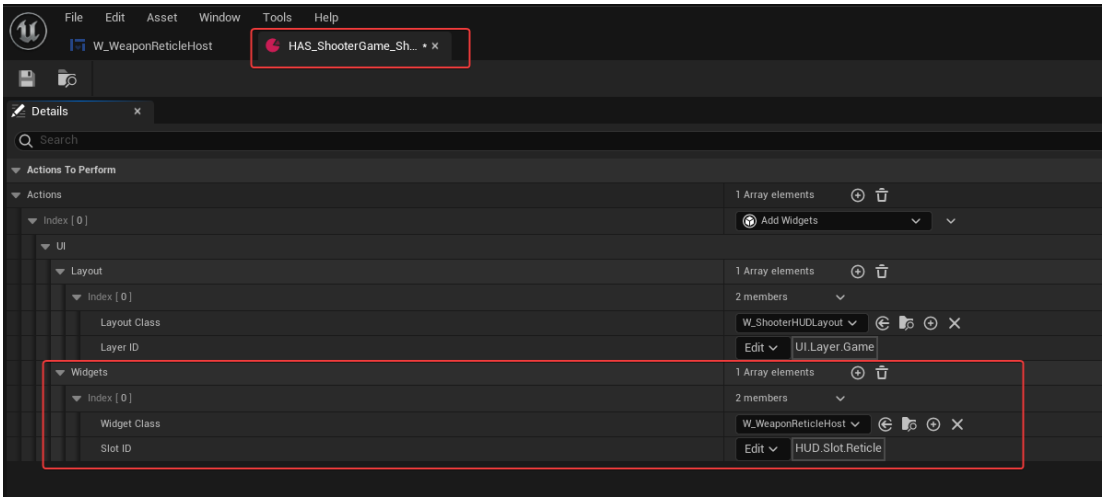
```cpp
void UHakReticleWidgetBase::InitializeFromWeapon(UHakWeaponInstance* InWeapon)
{
    WeaponInstance = InWeapon;
    InventoryInstance = nullptr;
    if (WeaponInstance)
    {
        InventoryInstance = Cast<UHakInventoryItemInstance>(WeaponInstance->GetInstigator());
    }
}
```

- [ ] OnWeaponChanged():





- [ ] GFA_AddWidgets()에 W_WeaponReticleHost를 HUD.Slot.Reticle에 넣기:

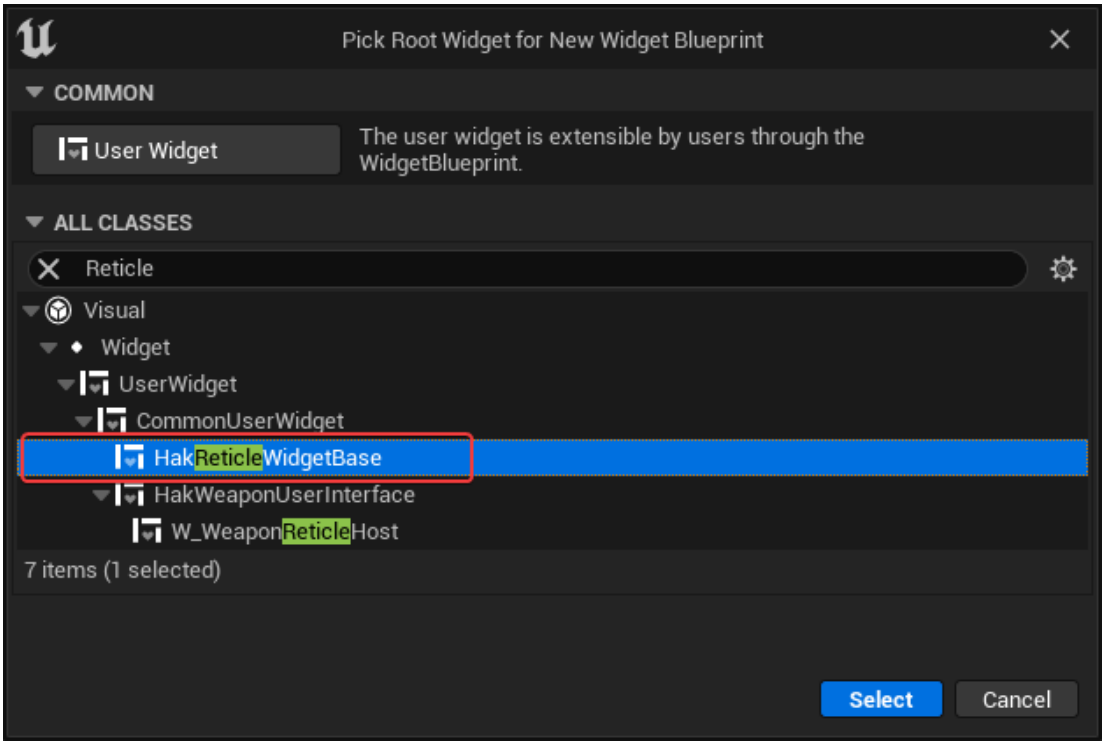


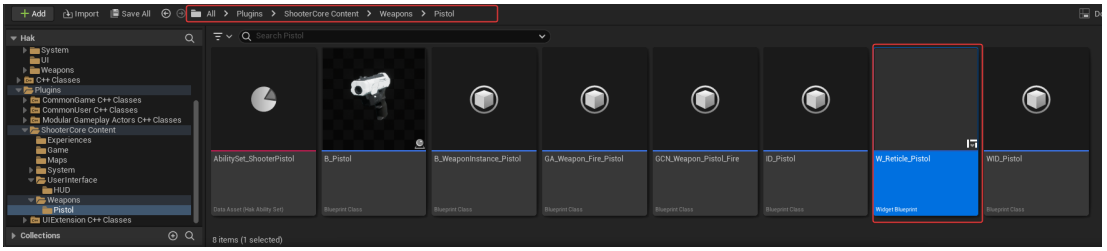
W_Reticle_Pistol

▼ **펼치기**

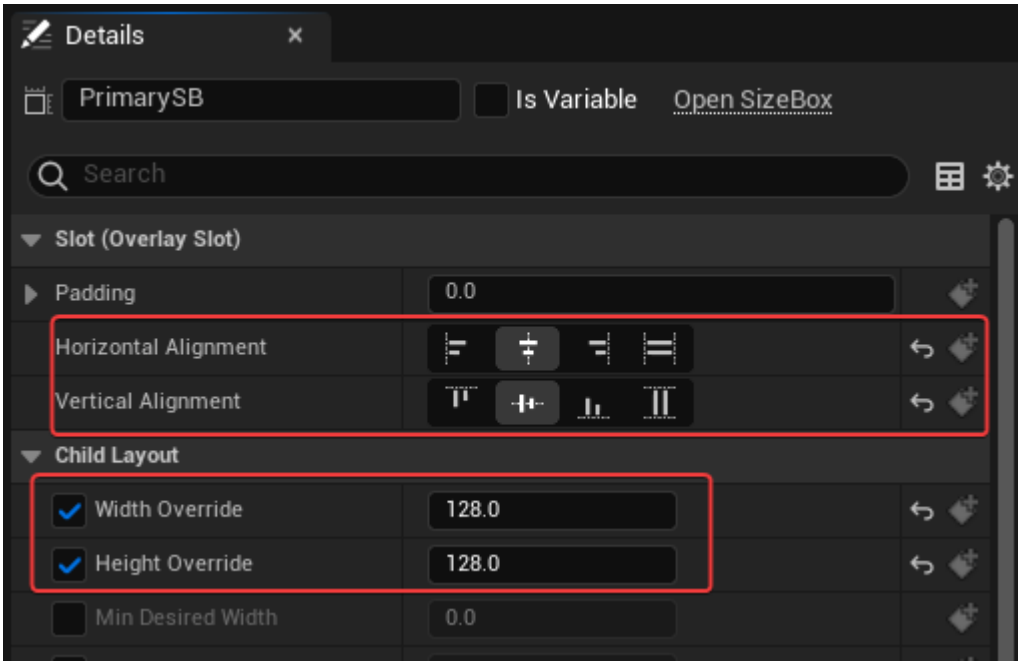
- InventoryFragment_ReticleConfig에 담을 ReticleWidget인 W_Reticle_Pistol 정의하자

☐ W_Reticle_Pistol 생성:

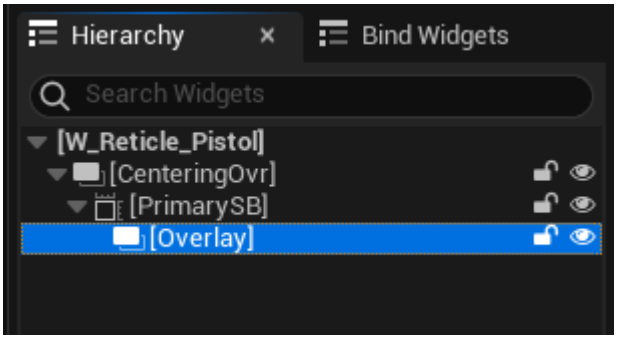




- [ ] W_Reticle_Pistol UI 레이아웃 정의:
  - [ ] Overlay: CenteringOvr
  - [ ] SizeBox: PrimarySB:

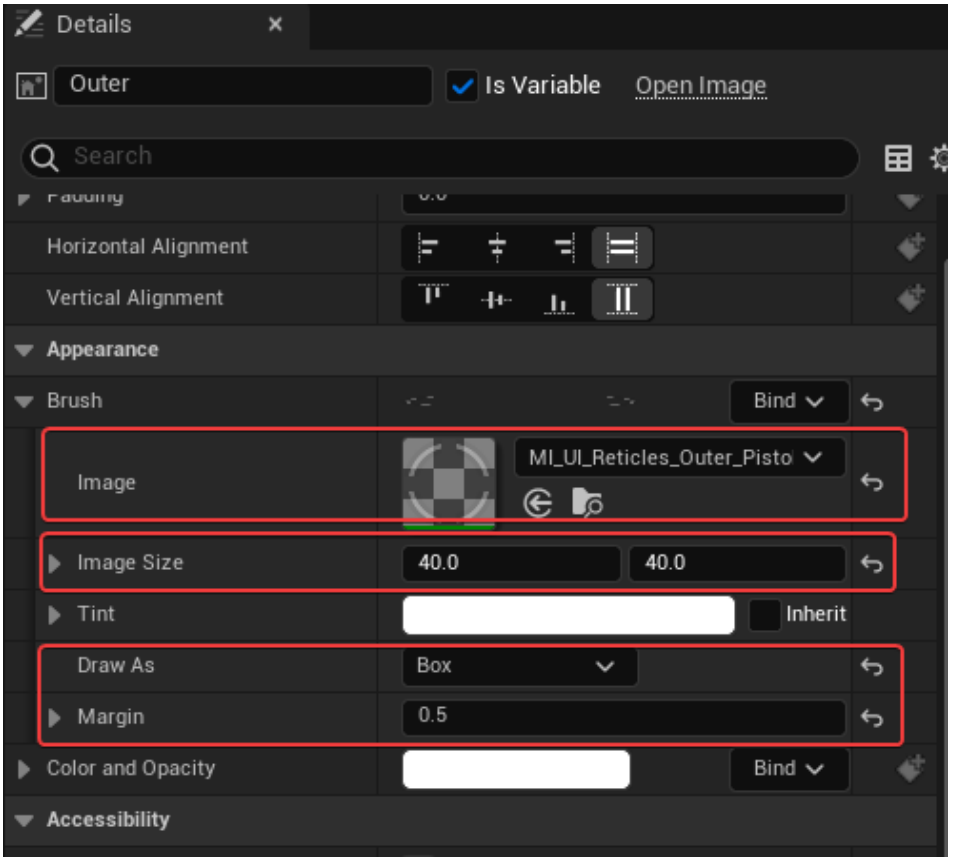


  - [ ] Overlay:

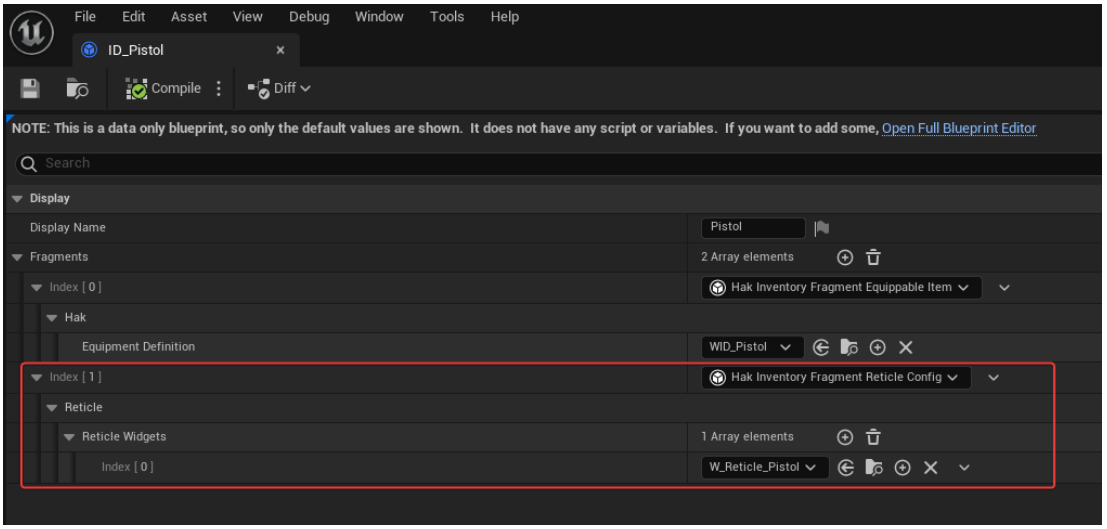


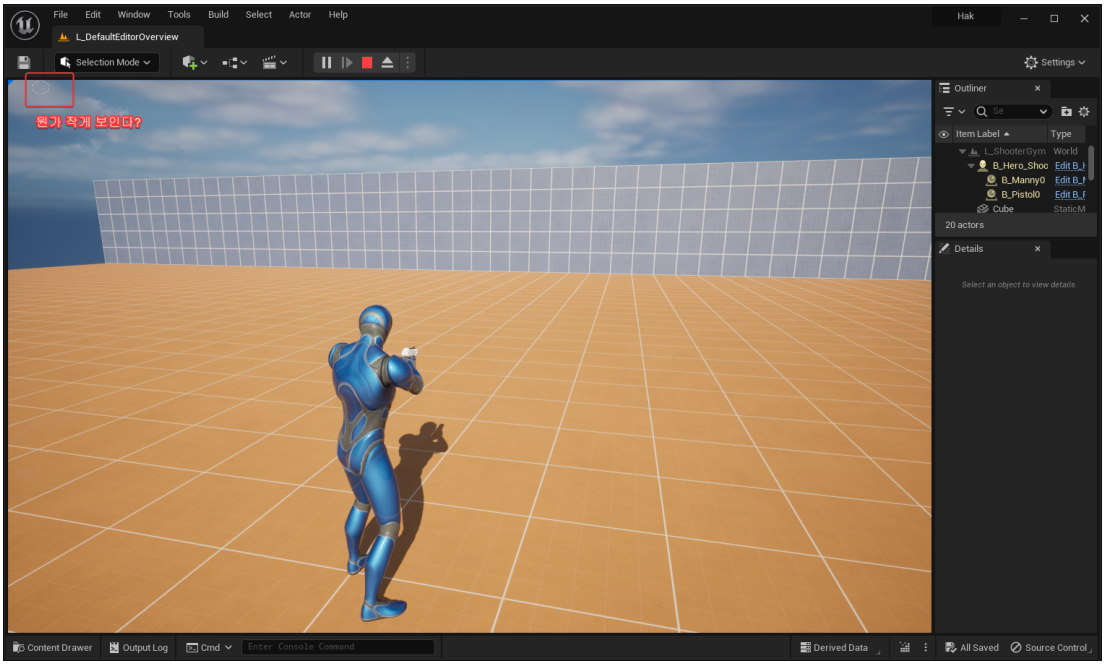
  - [ ] SizeBox: SBOuterReticle

- [ ] Image: Outer
  - [ ] Migrate `MI_UI_Reticles_Outer_Pistol`

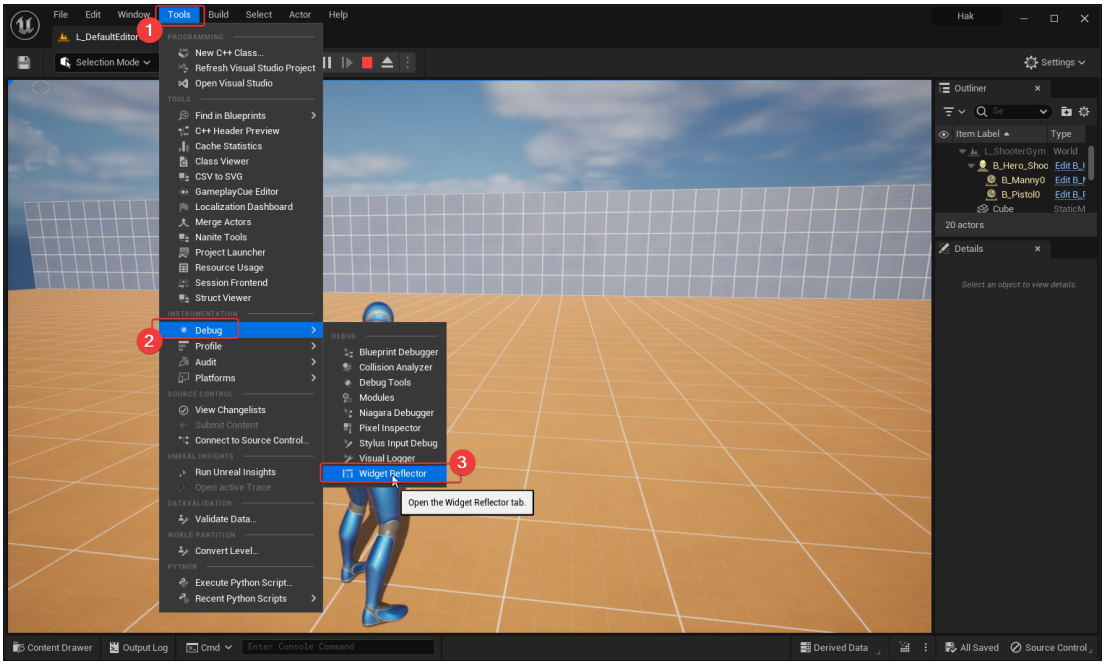


- [ ] W_Reticle_Pistol을 ID_Pistol에 넣어 Fragment Reticle Config를 Pistol에 설정해주자:

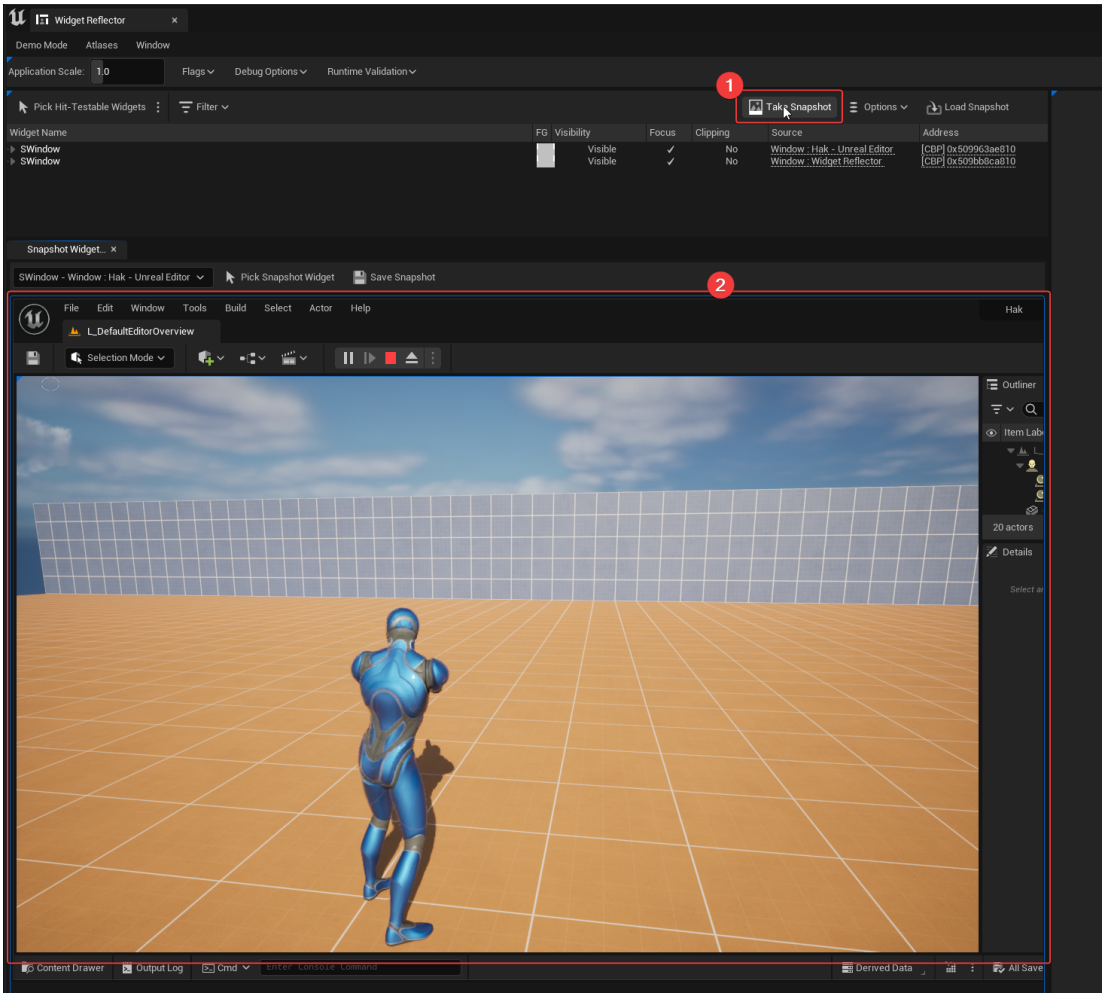




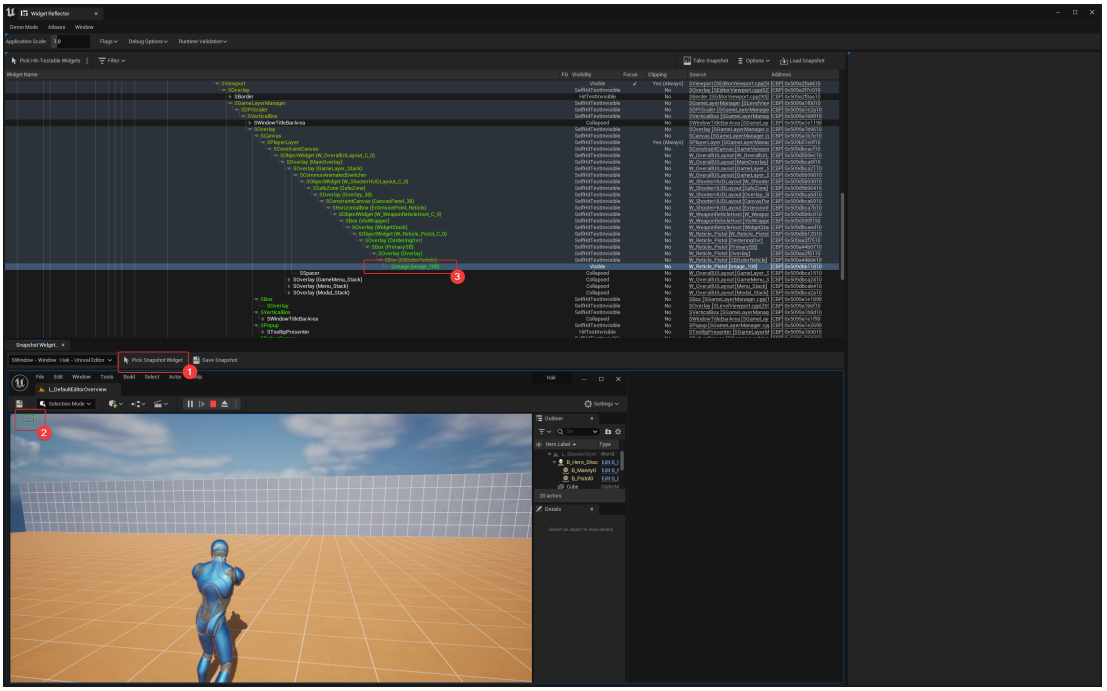
- [ ] 디버깅을 위해 WidgetReflectors를 활용해보자:



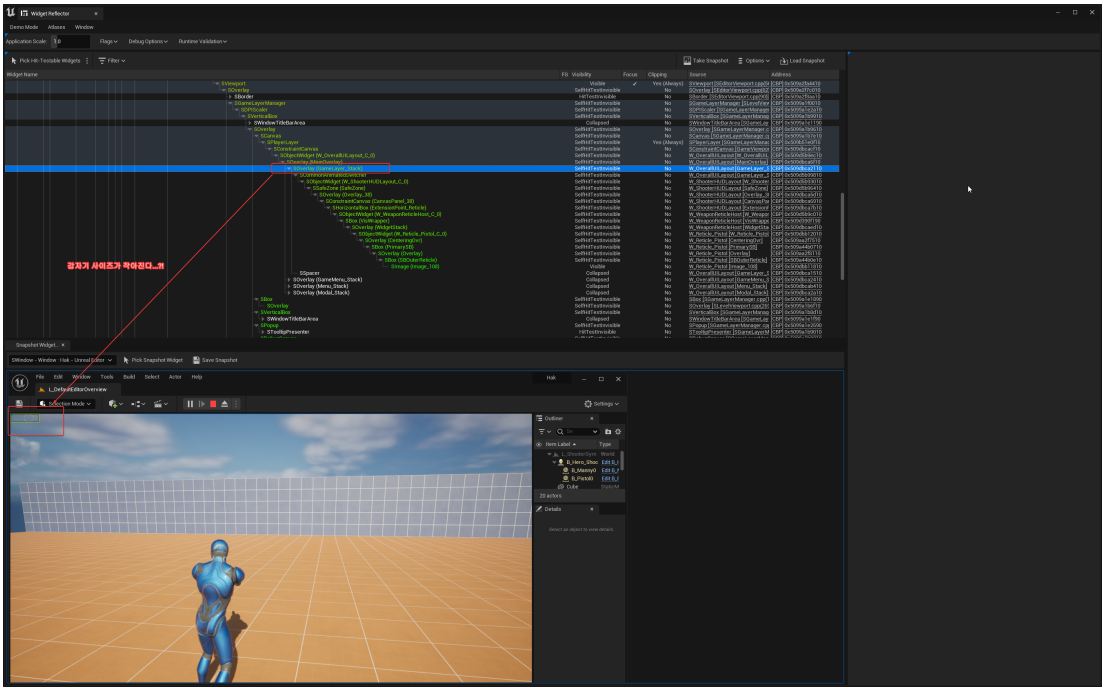
☐ TakeSnapshot으로 찍자:



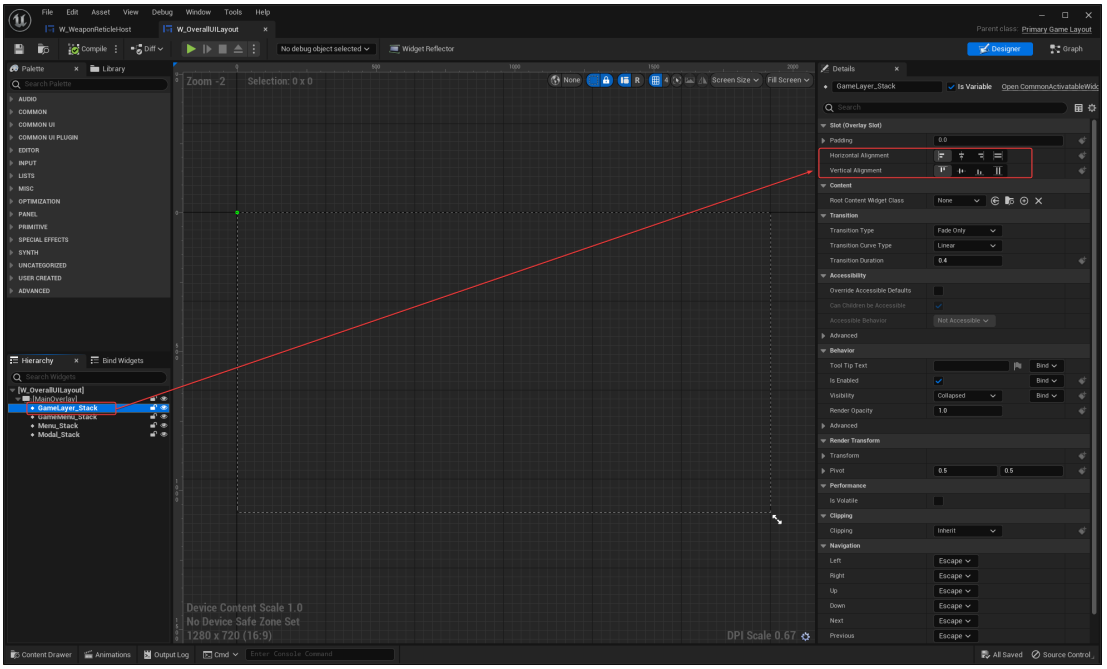
☐ Pick Snapshot Widget을 활용해 디버깅하면 된다:



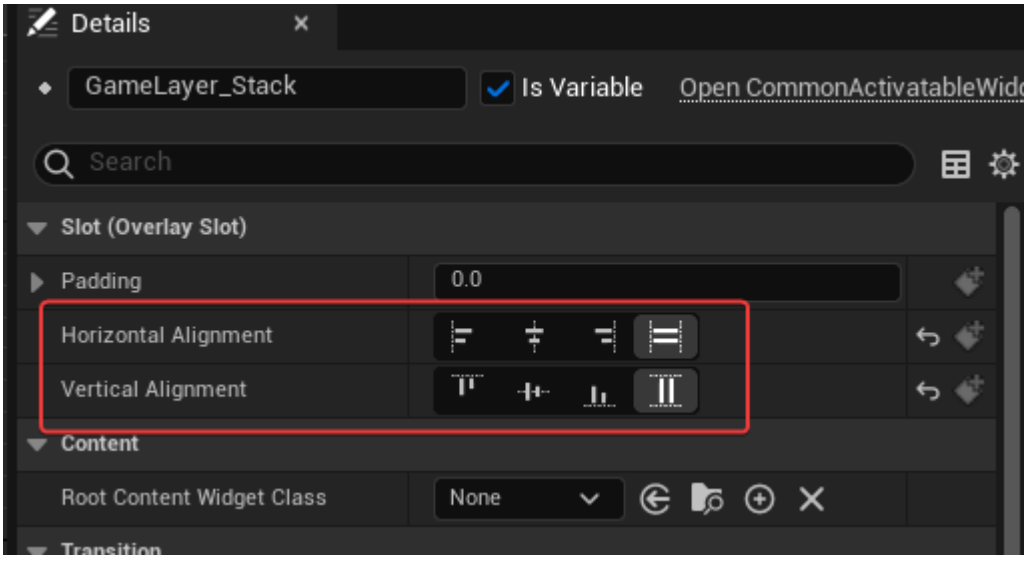
☐ GameLayer_Stack이 이상하다:



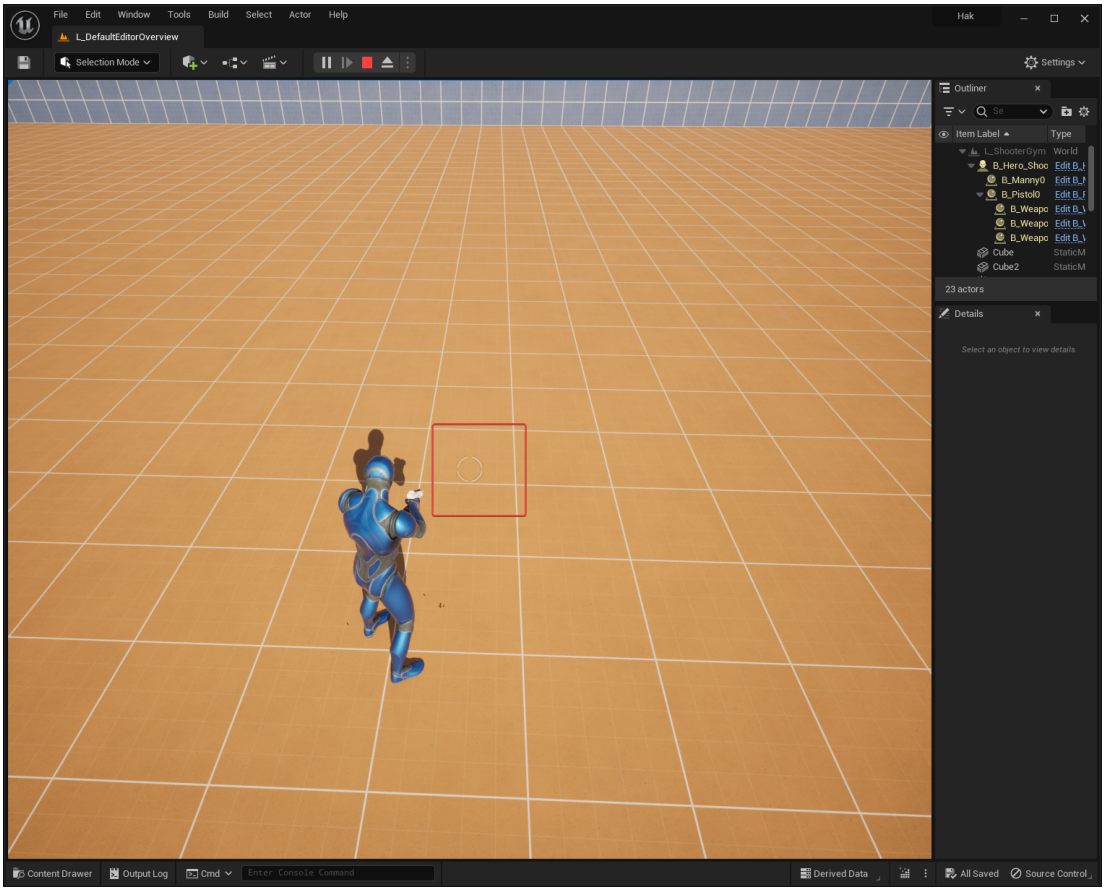
☐ W_OverallUILayout의 각 CommonActivatableWidget의 Alignment 설정이 잘못 되었다:



☐ 각 Layer에 대해 모두 아래와 같이 변경하자:



☐ 그럼 아래와 같이 잘 나온다:



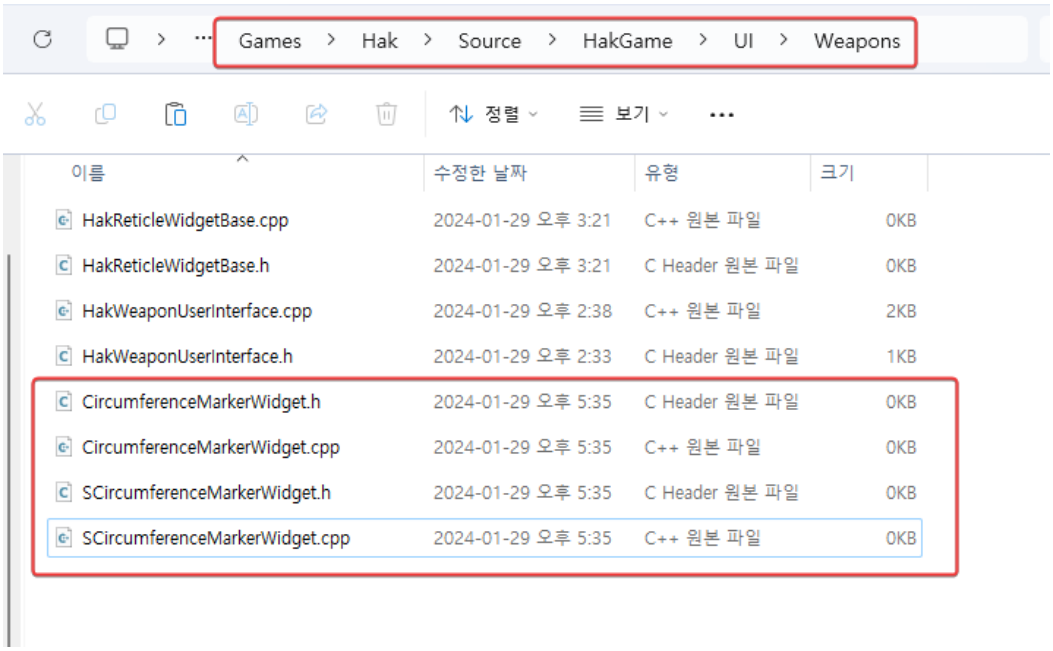
CircumferenceMarkerWidget

▼ **펼치기**

- **십자가 표시 과녁을 렌더링** 해보자

☐ CirumferenceMarkerWidget/SCircumferenceMarekrWidget:

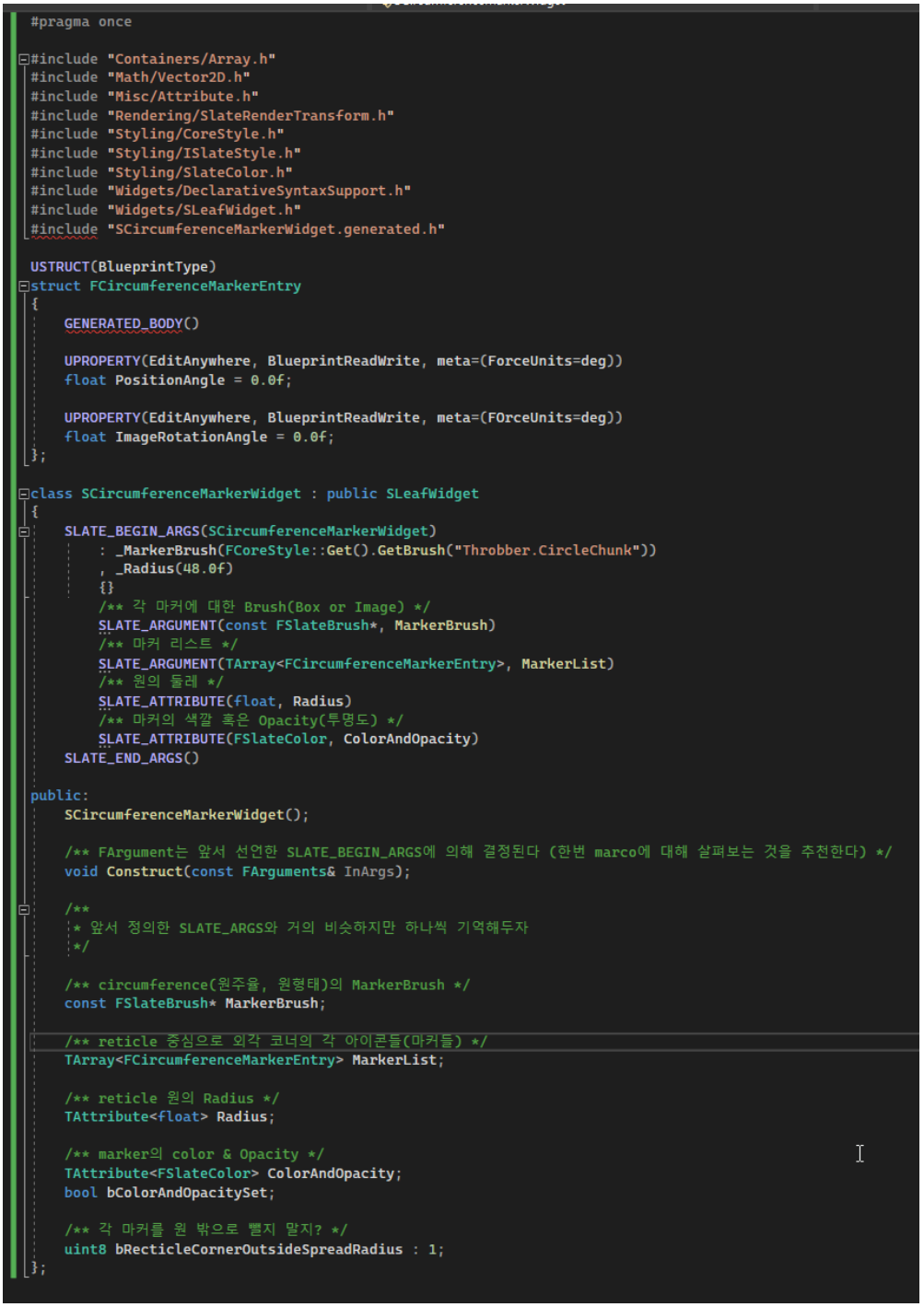
☐ 파일 추가:



- [ ] SCircumferenceMarkerWidget.h/.cpp:
  - [ ] Slate/SlateCore를 HakGame.Build.cs에 포함시키자:

```
using UnrealBuildTool;

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

        PublicDependencyModuleNames.AddRange(new string[] {
            "Core",
            "CoreUObject",
            "Engine",
            "InputCore",
            // GAS
            "GameplayTags",
            "GameplayTasks",
            "GameplayAbilities",
            // Game Features
            "ModularGameplay",
            "GameFeatures",
            "ModularGameplayActors",
            // Input
            "InputCore",
            "EnhancedInput",
            // CommonUser
            "CommonUser",
            // CommonGame
            "CommonGame",
            // CommonUI
            "CommonUI",
            // UMG
            "UMG",
            // UIExtension
            "UIExtension",
            // Slate
            "Slate",
            "SlateCore",
        });

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

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

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

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

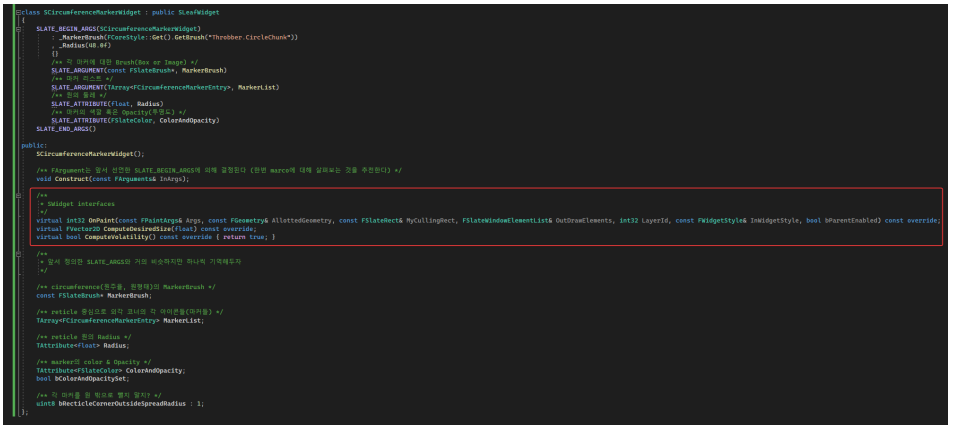
  - [ ] SCircumferenceMarkerWidget:



```h
#pragma once

#include "Containers/Array.h"
#include "Math/Vector2D.h"
#include "Misc/Attribute.h"
#include "Rendering/SlateRenderTransform.h"
#include "Styling/CoreStyle.h"
#include "Styling/ISlateStyle.h"
#include "Styling/SlateColor.h"
#include "Widgets/DeclarativeSyntaxSupport.h"
#include "Widgets/SLeafWidget.h"
#include "SCircumferenceMarkerWidget.generated.h"

USTRUCT(BlueprintType)
struct FCircumferenceMarkerEntry
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere, BlueprintReadWrite, meta=(ForceUnits=deg))
    float PositionAngle = 0.0f;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, meta=(FOrceUnits=deg))
    float ImageRotationAngle = 0.0f;
};

class SCircumferenceMarkerWidget : public SLeafWidget
{
    SLATE_BEGIN_ARGS(SCircumferenceMarkerWidget)
        : _MarkerBrush(FCoreStyle::Get().GetBrush("Throbber.CircleChunk"))
        , _Radius(48.0f)
        {}
        /** 각 마커에 대한 Brush(Box or Image) */
        SLATE_ARGUMENT(const FSlateBrush*, MarkerBrush)
        /** 마커 리스트 */
        SLATE_ARGUMENT(TArray<FCircumferenceMarkerEntry>, MarkerList)
        /** 원의 둘레 */
        SLATE_ATTRIBUTE(float, Radius)
        /** 마커의 색깔 혹은 Opacity(투명도) */
        SLATE_ATTRIBUTE(FSlateColor, ColorAndOpacity)
    SLATE_END_ARGS()

public:
    SCircumferenceMarkerWidget();

    /** FArgument는 앞서 선언한 SLATE_BEGIN_ARGS에 의해 결정된다 (한번 marco에 대해 살펴보는 것을 추천한다) */
    void Construct(const FArguments& InArgs);

    /**
    * 앞서 정의한 SLATE_ARGS와 거의 비슷하지만 하나씩 기억해두자
    */

    /** circumference(원주율, 원형태)의 MarkerBrush */
    const FSlateBrush* MarkerBrush;

    /** reticle 중심으로 외각 코너의 각 아이콘들(마커들) */
    TArray<FCircumferenceMarkerEntry> MarkerList;

    /** reticle 원의 Radius */
    TAttribute<float> Radius;

    /** marker의 color & Opacity */
    TAttribute<FSlateColor> ColorAndOpacity;
    bool bColorAndOpacitySet;

    /** 각 마커를 원 밖으로 뺄지 말지? */
    uint8 bRecticleCornerOutsideSpreadRadius : 1;
};
```

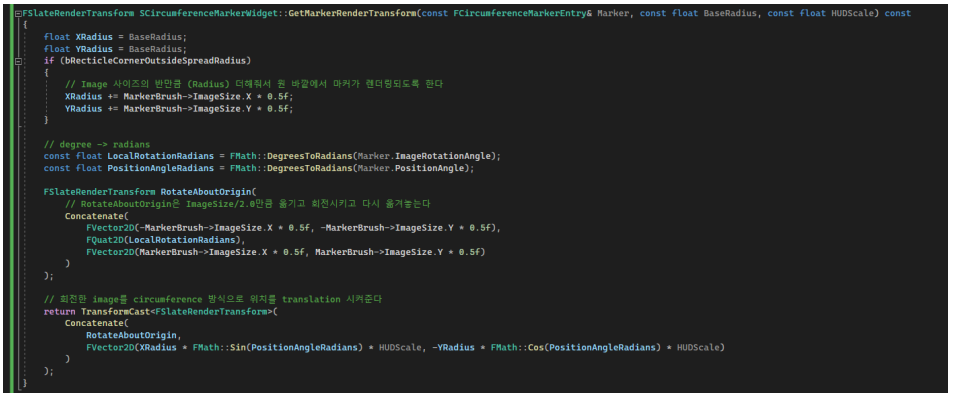
```cpp
#include "SCircumferenceMarkerWidget.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(SCircumferenceMarkerWidget)

SCircumferenceMarkerWidget::SCircumferenceMarkerWidget()
{

}

void SCircumferenceMarkerWidget::Construct(const FArguments& InArgs)
{
    MarkerBrush = InArgs._MarkerBrush;
    MarkerList = InArgs._MarkerList;
    Radius = InArgs._Radius;
    bColorAndOpacitySet = InArgs._ColorAndOpacity.IsSet();
    ColorAndOpacity = InArgs._ColorAndOpacity;
}
```

- [ ] SWidget Interfaces:
  - [ ] SWidget 인터페이스 선언:



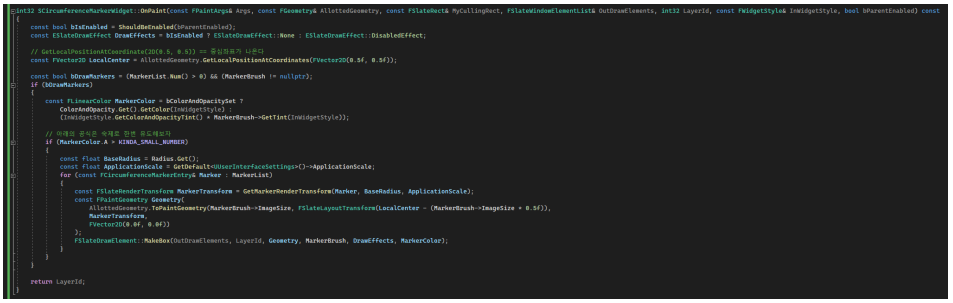
```cpp
class SCircumferenceMarkerWidget : public SLeafWidget
{
    SLATE_BEGIN_ARGS(SCircumferenceMarkerWidget)
        : _MarkerBrush(FCoreStyle::Get().GetBrush("Throbber.CircleChunk"))
        , _Radius(50.0f)
        {}
        /** 각 마커에 대한 Brush(Box or Image) */
        SLATE_ARGUMENT(const FSlateBrush*, MarkerBrush)
        /** 마커 리스트 */
        SLATE_ARGUMENT(TArray<FCircumferenceMarkerEntry>, MarkerList)
        /** 원의 반지름 */
        SLATE_ATTRIBUTE(float, Radius)
        /** 마커와 색상 혹은 Opacity(투명도) */
        SLATE_ATTRIBUTE(FSlateColor, ColorAndOpacity)
    SLATE_END_ARGS()

public:
    SCircumferenceMarkerWidget();

    /** FArgument는 앞서 선언한 SLATE_BEGIN_ARGS에 의해 결정된다 (한번 marco에 대해 살펴보는 것을 추천한다) */
    void Construct(const FArguments& InArgs);

    /**
     * Swidget interfaces
     */
    virtual int32 OnPaint(const FPaintArgs& Args, const FGeometry& AllottedGeometry, const FSlateRect& MyCullingRect, FSlateWindowElementList& OutDrawElements, int32 LayerId, const FWidgetStyle& InWidgetStyle, bool bParentEnabled) const override;
    virtual FVector2D ComputeDesiredSize(float) const override;
    virtual bool ComputeVolatility() const override { return true; }

    /**
     * 앞서 정의한 SLATE_ARGS와 거의 비슷하지만 하나씩 기억해두자
     */

    /** circumference(원주율, 파형대)의 MarkerBrush */
    const FSlateBrush* MarkerBrush;

    /** reticle 중심으로 외각 고니의 각 아이콘들(마커들) */
    TArray<FCircumferenceMarkerEntry> MarkerList;

    /** reticle 원의 Radius */
    TAttribute<float> Radius;

    /** marker의 color & Opacity */
    TAttribute<FSlateColor> ColorAndOpacity;
    bool bColorAndOpacitySet;

    /** 각 마커를 원 형으로 펼쳐 놓기? */
    uint8 bRecticleCornerOutsideSpreadRadius : 1;
};
```

  - [ ] GetMarkerRenderTransform():



```cpp
FSlateRenderTransform SCircumferenceMarkerWidget::GetMarkerRenderTransform(const FCircumferenceMarkerEntry& Marker, const float BaseRadius, const float HUDScale) const
{
    float XRadius = BaseRadius;
    float YRadius = BaseRadius;
    if (bRecticleCornerOutsideSpreadRadius)
    {
        // Image 사이즈의 반만큼 (Radius) 더해줘서 원 바깥에서 마커가 랜더링되도록 한다
        XRadius += MarkerBrush->ImageSize.X * 0.5f;
        YRadius += MarkerBrush->ImageSize.Y * 0.5f;
    }

    // degree -> radians
    const float LocalRotationRadians = FMath::DegreesToRadians(Marker.ImageRotationAngle);
    const float PositionAngleRadians = FMath::DegreesToRadians(Marker.PositionAngle);

    FSlateRenderTransform RotateAboutOrigin(
        // RotateAboutOrigin은 ImageSize/2.0만큼 옮기고 회전시키고 다시 옮겨놓는다
        Concatenate(
            FVector2D(-MarkerBrush->ImageSize.X * 0.5f, -MarkerBrush->ImageSize.Y * 0.5f),
            FQuat2D(LocalRotationRadians),
            FVector2D(MarkerBrush->ImageSize.X * 0.5f, MarkerBrush->ImageSize.Y * 0.5f)
        )
    );

    // 회전한 image를 circumference 방식으로 위치를 translation 시켜준다
    return TransformCast<FSlateRenderTransform>(
        Concatenate(
            RotateAboutOrigin,
            FVector2D(XRadius * FMath::Sin(PositionAngleRadians) * HUDScale, -YRadius * FMath::Cos(PositionAngleRadians) * HUDScale)
        )
    );
}
```

  - [ ] OnPaint():



- [ ] CircumferenceMarkerWidget.h/.cpp

```h
#pragma once

#include "Components/Widget.h"
#include "SCircumferenceMarkerWidget.h"
#include "CircumferenceMarkerWidget.generated.h"

UCLASS()
class UCircumferenceMarkerWidget : public UWidget
{
    GENERATED_BODY()
public:
    UCircumferenceMarkerWidget(const FObjectInitializer& ObjectInitializer);

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Appearance)
    TArray<FCircumferenceMarkerEntry> MarkerList;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Appearance)
    float Radius = 48.0f;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Appearance)
    FSlateBrush MarkerImage;

    UPROPERTY(EditAnywhere, Category = Corner)
    uint8 bReticleCornerOutsideSpreadRadius : 1;

    /** UMG의 CircumferenceMarkerWidget에 대응되는 SWidget */
    TSharedPtr<SCircumferenceMarkerWidget> MyMarkerWidget;
};
```

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

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

- [ ] ReleaseSlateResources():

```cpp
UCLASS()
class UCircumferenceMarkerWidget : public UWidget
{
    GENERATED_BODY()
public:
    UCircumferenceMarkerWidget(const FObjectInitializer& ObjectInitializer);

    /**
     * UVisual interfaces
     */
    /** 해당 UWidget은 MyMarkerWidget의 SWidget을 가지고 있으므로 잘 해제해야 한다 */
    virtual void ReleaseSlateResources(bool bReleaseChildren) override;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Appearance)
    TArray<FCircumferenceMarkerEntry> MarkerList;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Appearance)
    float Radius = 48.0f;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Appearance)
    FSlateBrush MarkerImage;

    UPROPERTY(EditAnywhere, Category = Corner)
    uint8 bReticleCornerOutsideSpreadRadius : 1;

    /** UMG의 CircumferenceMarkerWidget에 대응되는 SWidget */
    TSharedPtr<SCircumferenceMarkerWidget> MyMarkerWidget;
};
```

```cpp
void UCircumferenceMarkerWidget::ReleaseSlateResources(bool bReleaseChildren)
{
    Super::ReleaseSlateResources(bReleaseChildren);
    MyMarkerWidget.Reset();
}
```

- [ ] UWidget interfaces:
  - [ ] Interface 선언:

```cpp
UCLASS()
class UCircumferenceMarkerWidget : public UWidget
{
    GENERATED_BODY()
public:
    UCircumferenceMarkerWidget(const FObjectInitializer& ObjectInitializer);

    /**
     * UWidget interfaces
     */
    /** SWidget과 UWidget간 데이터 싱크를 맞추기 위한 메서드 */
    virtual void SynchronizeProperties() override;
    /** Widget 재생성 */
    virtual TSharedRef<SWidget> RebuildWidget() override;

    /**
     * UVisual interfaces
     */
    /** 해당 UWidget은 MyMarkerWidget의 SWidget을 가지고 있으므로 잘 해제해야 한다 */
    virtual void ReleaseSlateResources(bool bReleaseChildren) override;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Appearance)
    TArray<FCircumferenceMarkerEntry> MarkerList;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Appearance)
    float Radius = 48.0f;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Appearance)
    FSlateBrush MarkerImage;

    UPROPERTY(EditAnywhere, Category = Corner)
    uint8 bReticleCornerOutsideSpreadRadius : 1;

    /** UMG의 CircumferenceMarkerWidget에 대응되는 SWidget */
    TSharedPtr<SCircumferenceMarkerWidget> MyMarkerWidget;
};
```

- [ ] SynchronizeProperties():

```cpp
void UCircumferenceMarkerWidget::SynchronizeProperties()
{
    Super::SynchronizeProperties();
    MyMarkerWidget->SetRadius(Radius);
    MyMarkerWidget->SetMarkerList(MarkerList);
}
```

  - [ ] SCircumferenceMarkerWidget::SetRadius()

```cpp
void SCircumferenceMarkerWidget::SetRadius(float NewRadius)
{
    if (Radius.IsBound() || (Radius.Get() != NewRadius))
    {
        Radius = NewRadius;
        Invalidate(EInvalidateWidgetReason::Layout);
    }
}
```

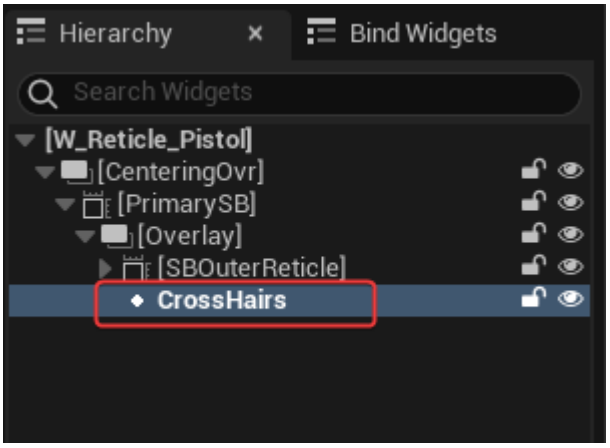
  - [ ] SCircumferenceMarkerWidget::SetMarkerList()

```cpp
void SCircumferenceMarkerWidget::SetMarkerList(TArray<FCircumferenceMarkerEntry>& NewMarkerList)
{
    MarkerList = NewMarkerList;
}
```

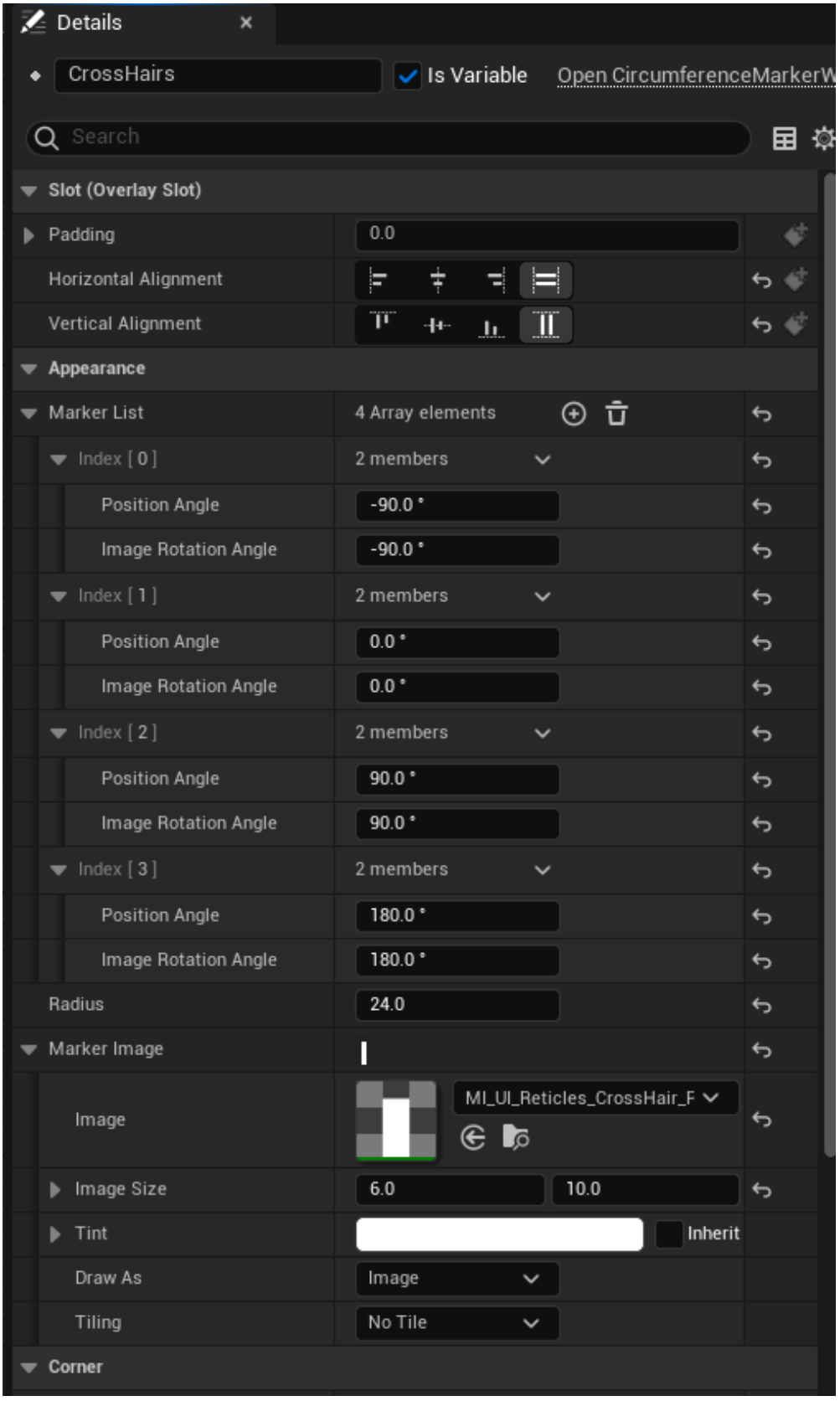
- [ ] RebuildWidget():

```cpp
TSharedRef<SWidget> UCircumferenceMarkerWidget::RebuildWidget()
{
    MyMarkerWidget = SNew(SCircumferenceMarkerWidget)
        .MarkerBrush(&MarkerImage)
        .Radius(this->Radius)
        .MarkerList(this->MarkerList);
    return MyMarkerWidget.ToSharedRef();
}
```

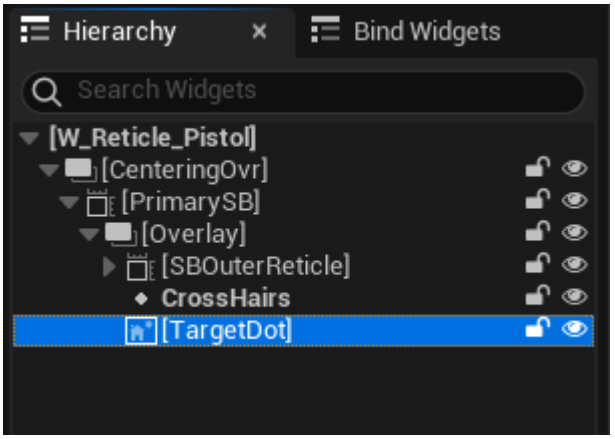
- [ ] CircumferenceMarkerWidget 추가:



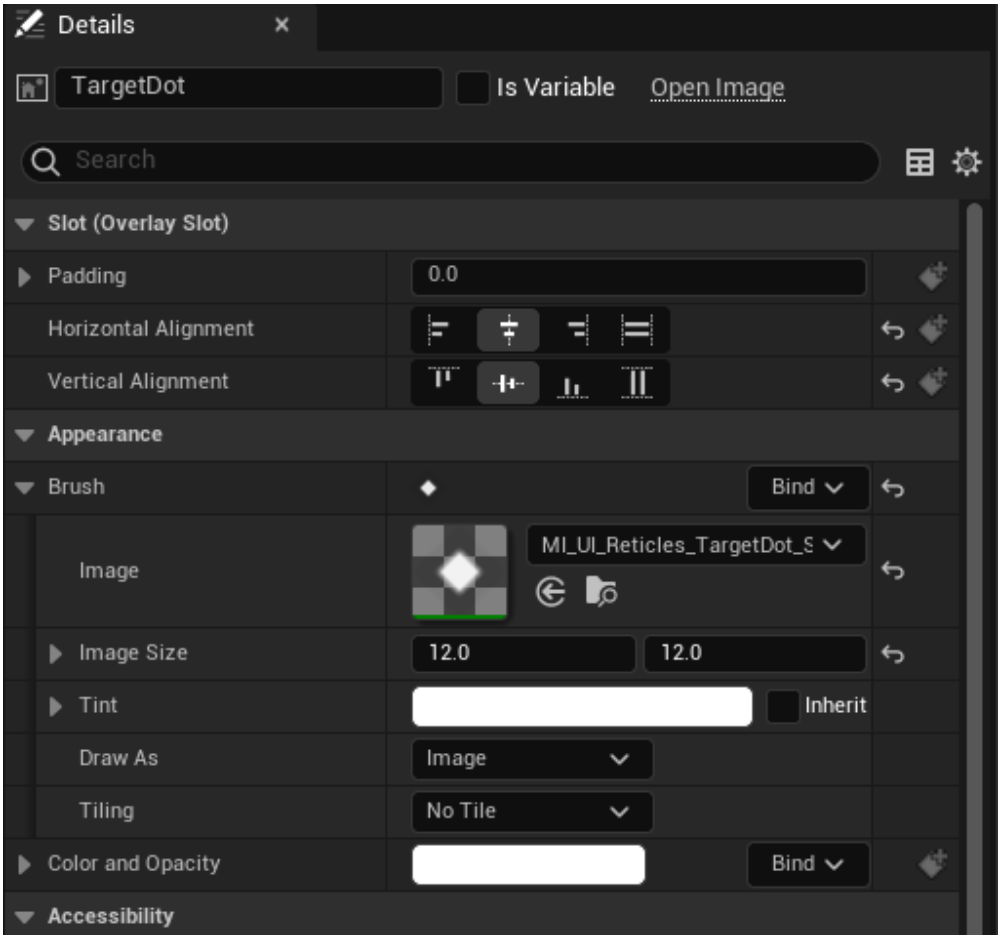
- [ ] `MI_UI_Reticle_CrossHair_Pistol` Migrate:
- [ ] 기본값 변경:



- [ ] TargetDot 넣기:
  - [ ] Image: TargetDot



- [ ] `MI_UI_Reticles_TargetDot_Single` Migrate
- [ ] 기본값 변경:



- [ ] 뭔가 HUD가 로딩 안되면 아래를 확인하자:

```cpp
/** HUD의 Layout 요청 */
USTRUCT()
struct FHakHUDLayoutRequest
{
    GENERATED_BODY()

    /** UI의 레이아웃으로 CommonActivatableWidget을 사용 */
    UPROPERTY(EditAnywhere, Category=UI, meta=(AssetBundles="Client"))
    TSoftClassPtr<UCommonActivatableWidget> LayoutClass;

    /** 앞서 보았던 PrimaryGameLayout의 LayerID를 의미 */
    UPROPERTY(EditAnywhere, Category=UI)
    FGameplayTag LayerID;
};

USTRUCT()
struct FHakHUDElementEntry
{
    GENERATED_BODY()

    /** HakHUDLayout 위에 올릴 대상이 되는 Widget Class */
    UPROPERTY(EditAnywhere, Category=UI, meta=(AssetBundles="Client"))
    TSoftClassPtr<UUserWidget> WidgetClass;

    /** SlotID는 HakHUDLayoutRequest에 올린 LayoutClass에 정의된 Slot(GameplayTag)를 의미 */
    UPROPERTY(EditAnywhere, Category=UI)
    FGameplayTag SlotID;
};
```

☐ 아래와 같이 완성되었다:

https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/c863237c-a0f0-4dc5-a343-ac8c2c5f9491/UnrealEditor_wAGqu6L7SS.mp4