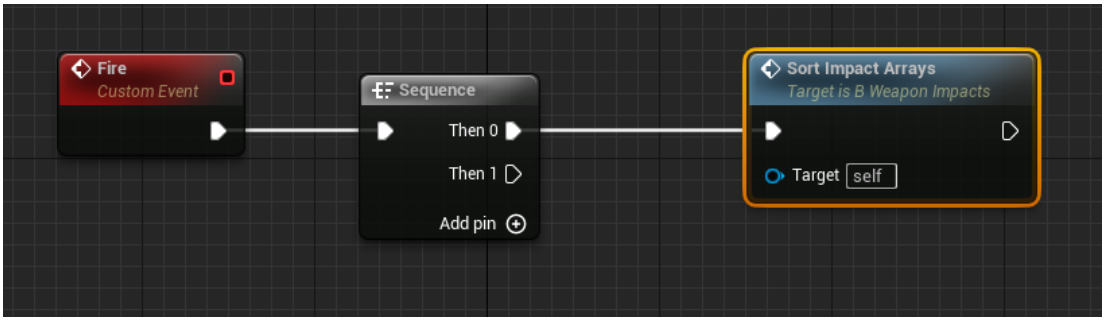
# 14주차 (2024.01.22)

## B_WeaponImpacts - 2

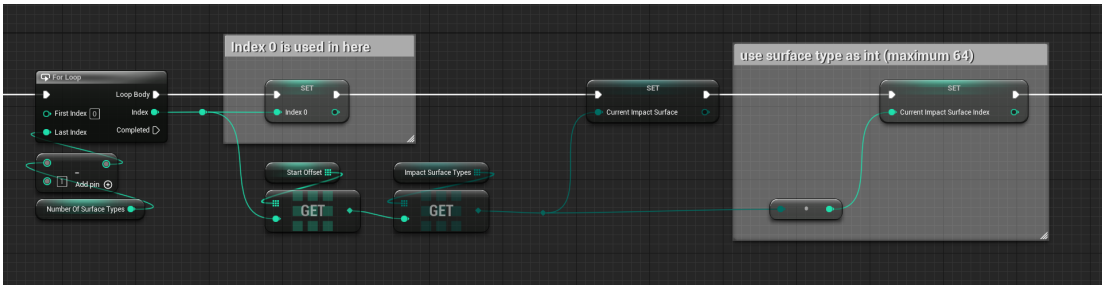
▼ **펼치기**

- [ ] Sort Impact Array를 먼저 호출하자:

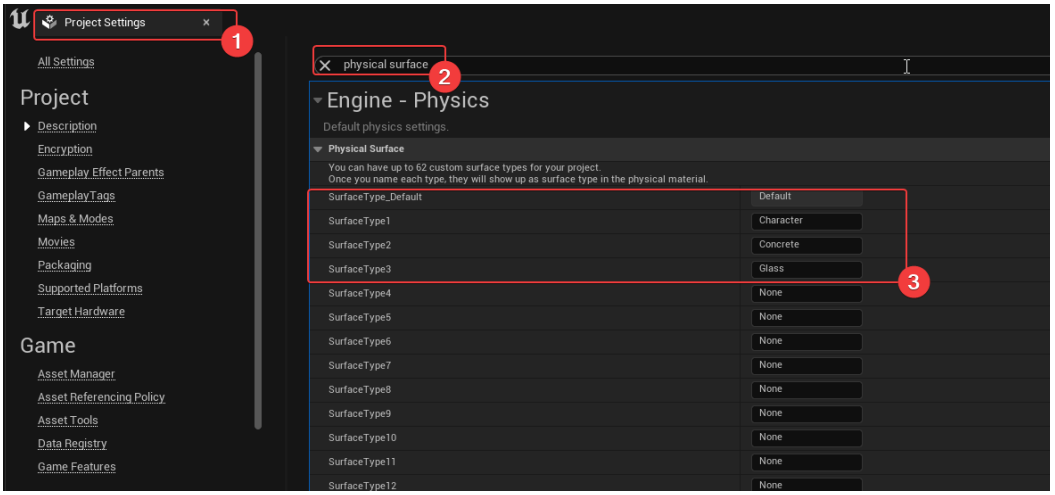



  - 앞서 로직을 다루었지만, Sorting을 하는 이유는 Niagara에 알맞는 데이터를 미리 가공하는 과정으로 생각하면 된다:
    - Niagara에서 Surface Type별로 복수개의 Impact 효과를 처리한다

- [ ] Surface 속성 별로 로직을 처리한다:



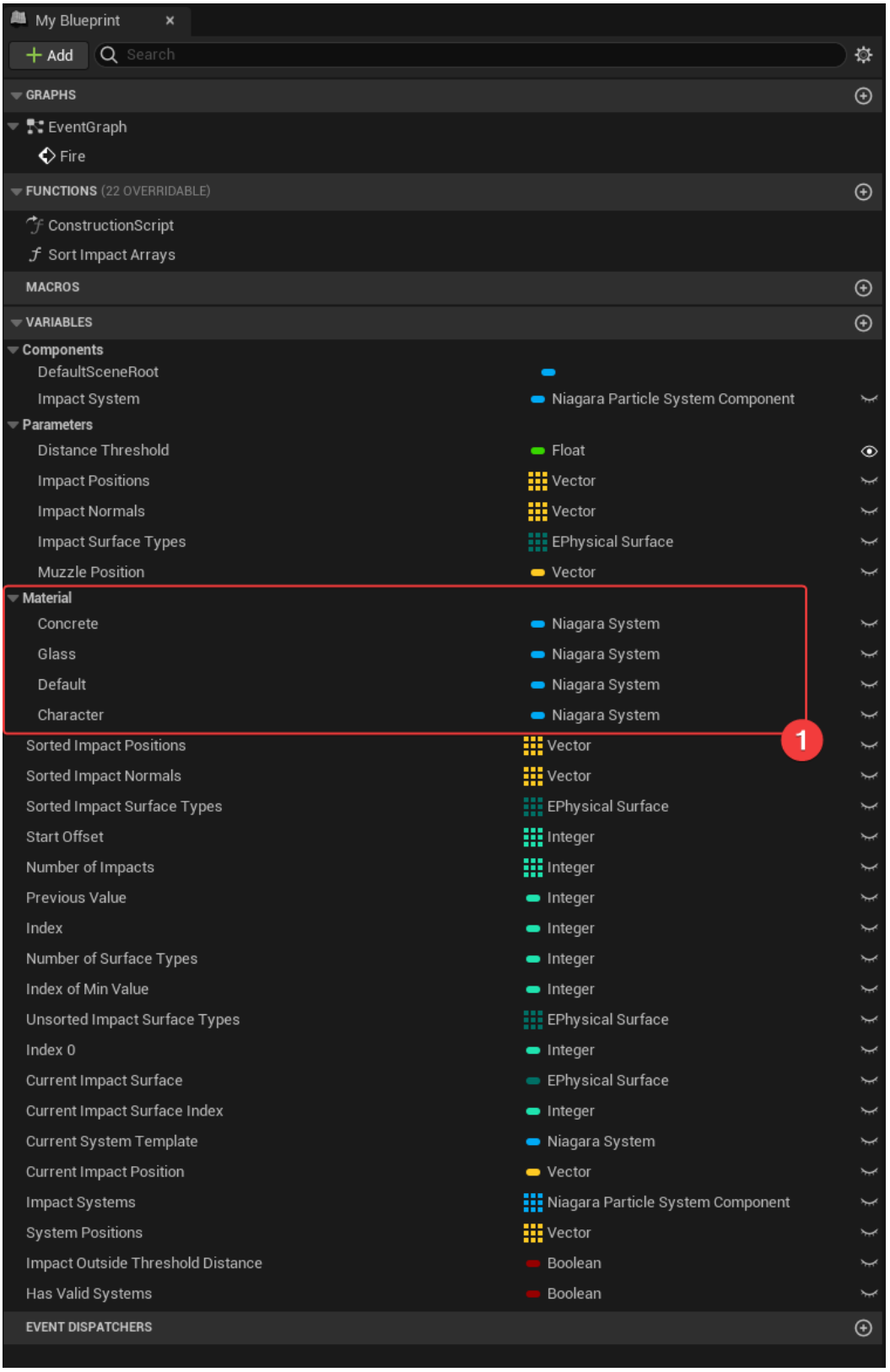

- [ ] 우선 EPhysicalSurface 속성을 정의하자:




- 참고로 최대 63개만 지정 가능하므로, Surface 종류를 추가할 때, 신중하게 정해야 한다. (GameplayTag와 조합으로 얼마든지 갯수를 더 늘리는 것이 가능하다!)

☐ 이어서 계속 로직을 작성해보자:

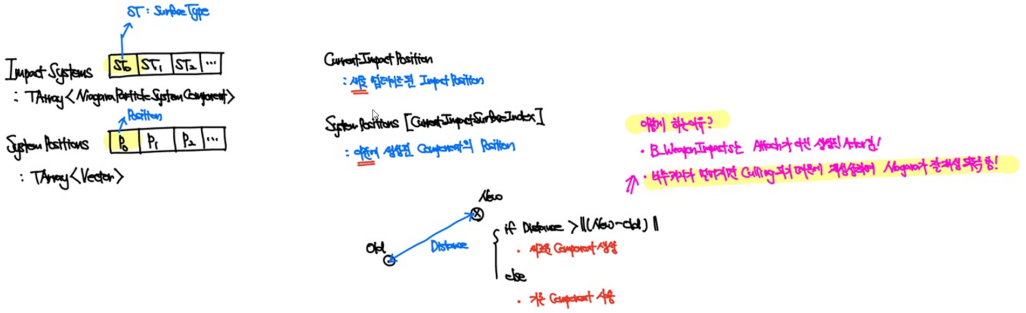
☐ 멤버 변수로, Physical Surface에 매칭되는 Niagara System을 추가하자:

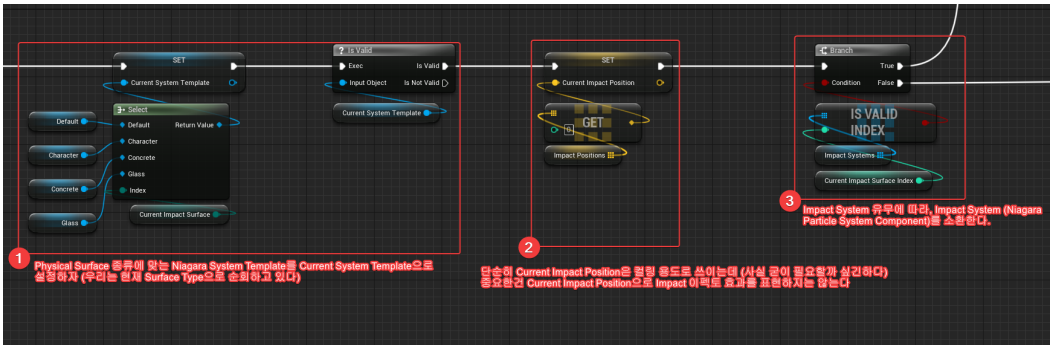


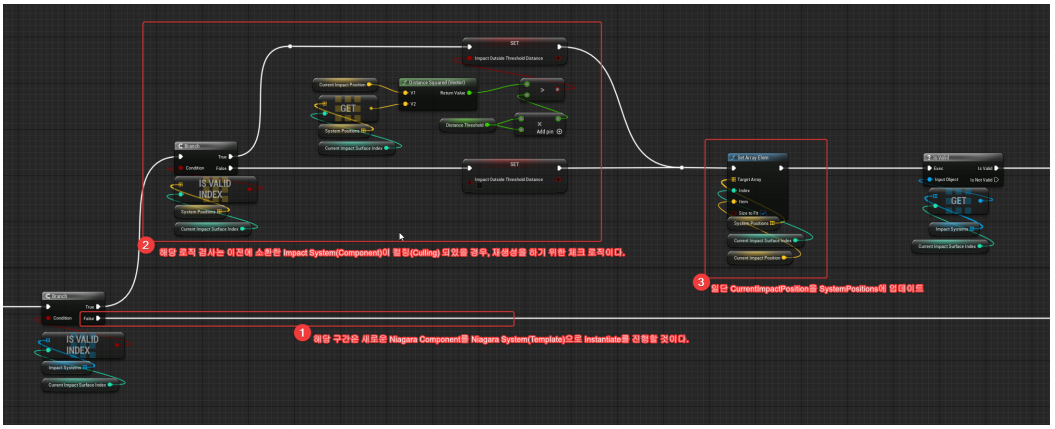

☐ 이에 맞는 Niagara System 기본값 설정 및 임포트하자:

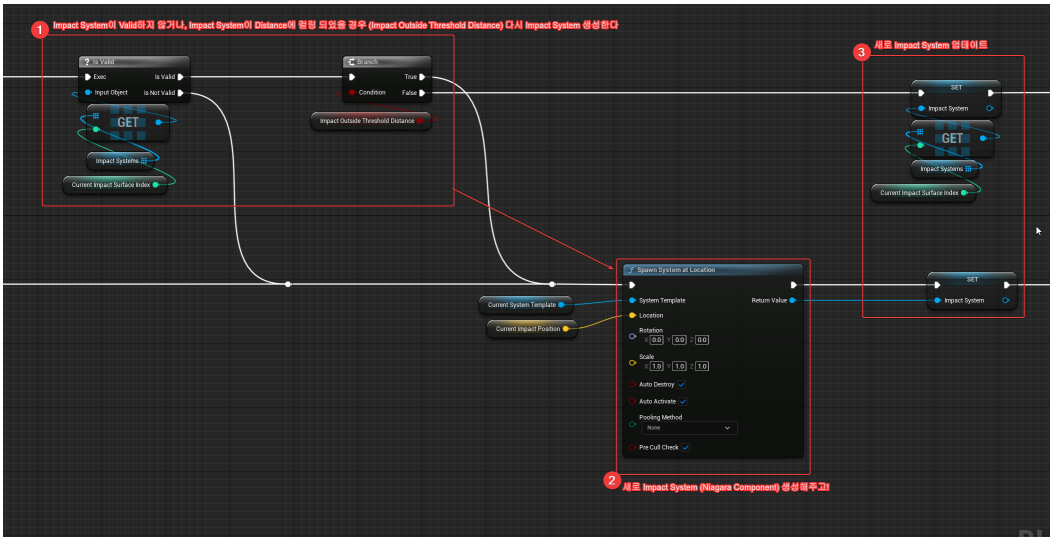
- Concrete: NS_ImpactConcrete
- Glass: NS_ImpactGlass

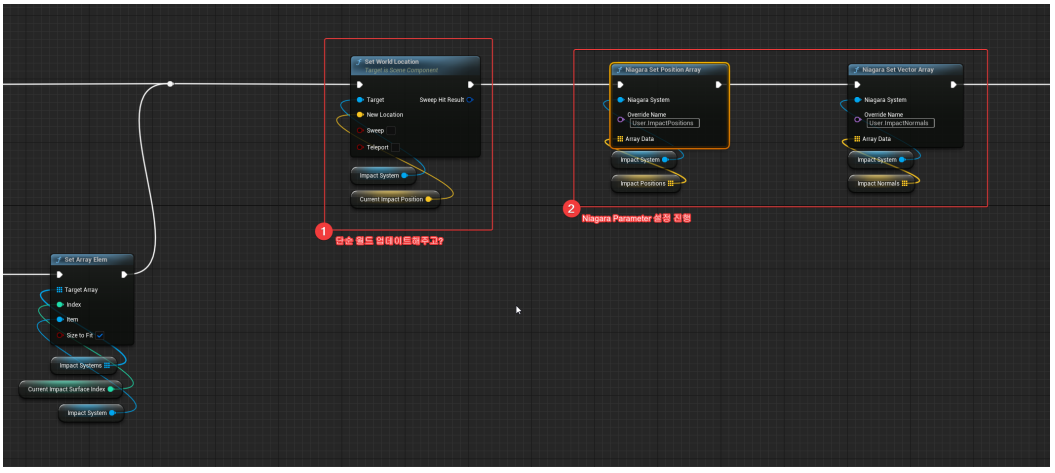
☐ Physical Surface에 맞는 Niagara System을 설정하자:

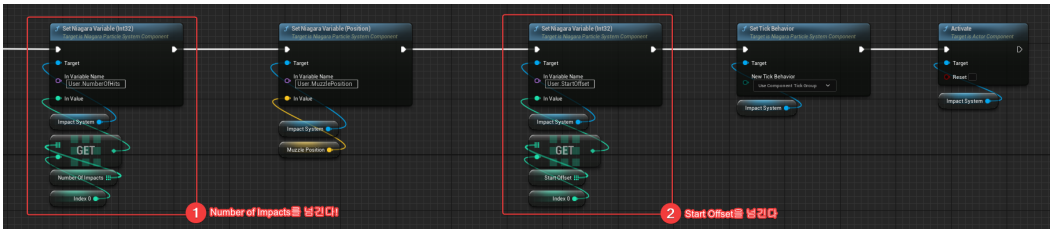


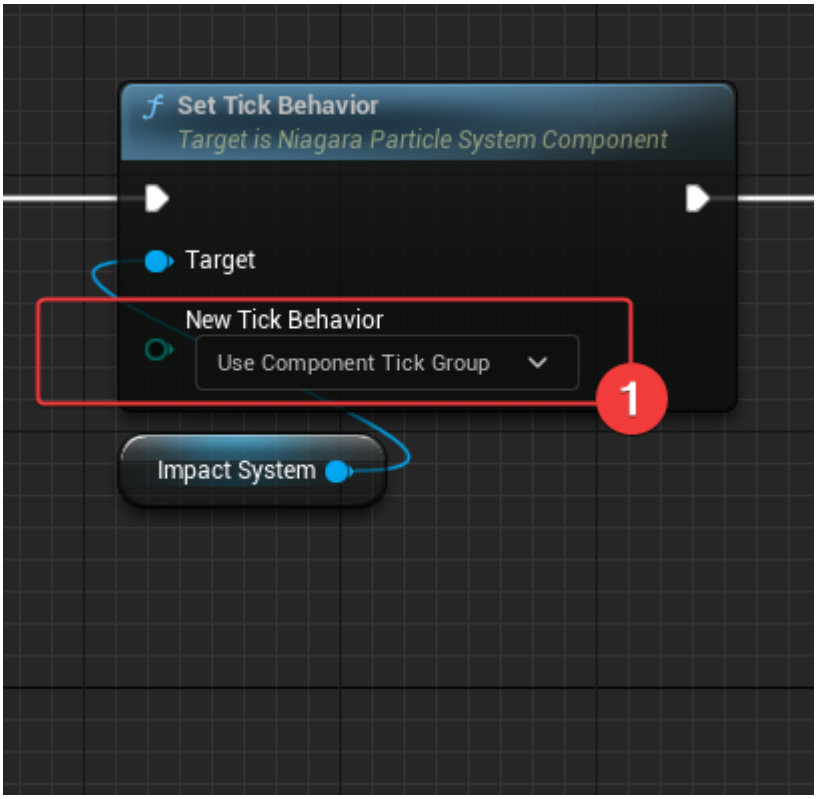





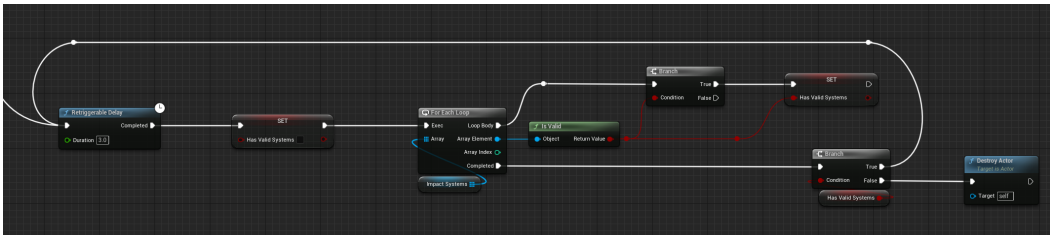



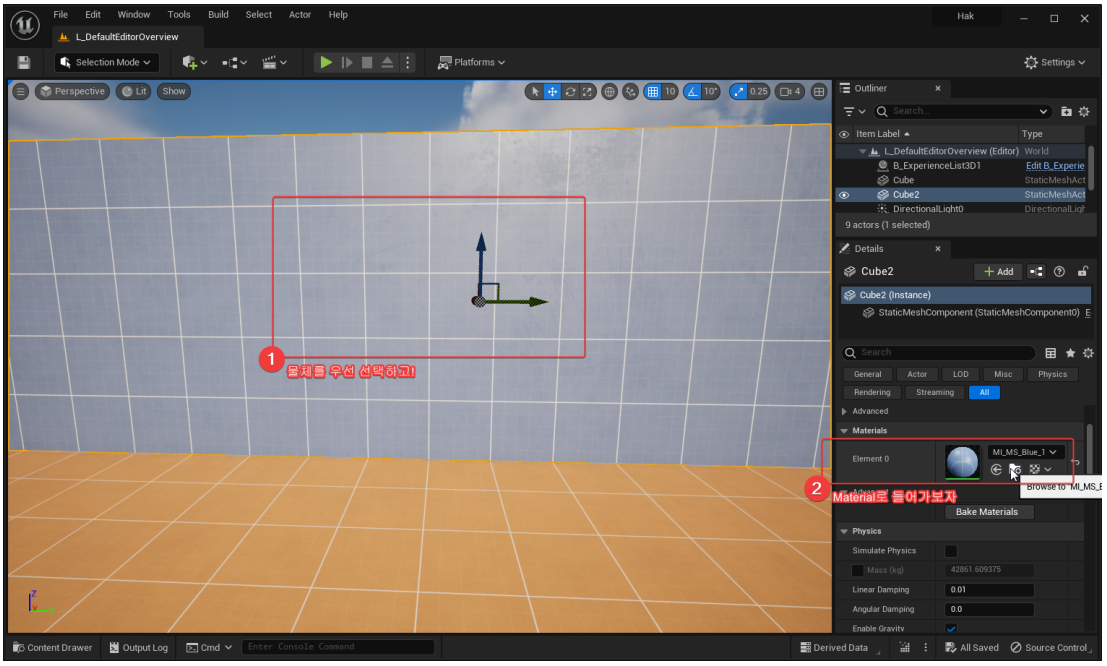

☐ Component Tick Group에 넣넹?

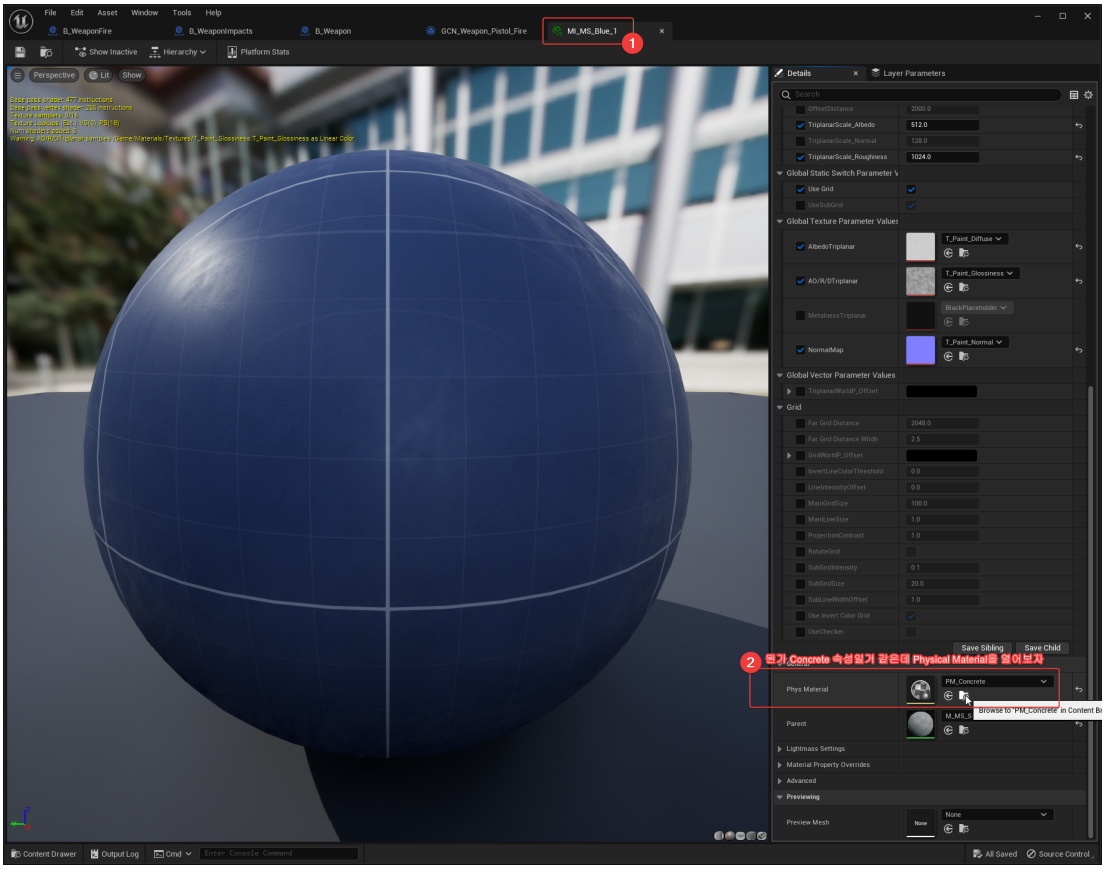


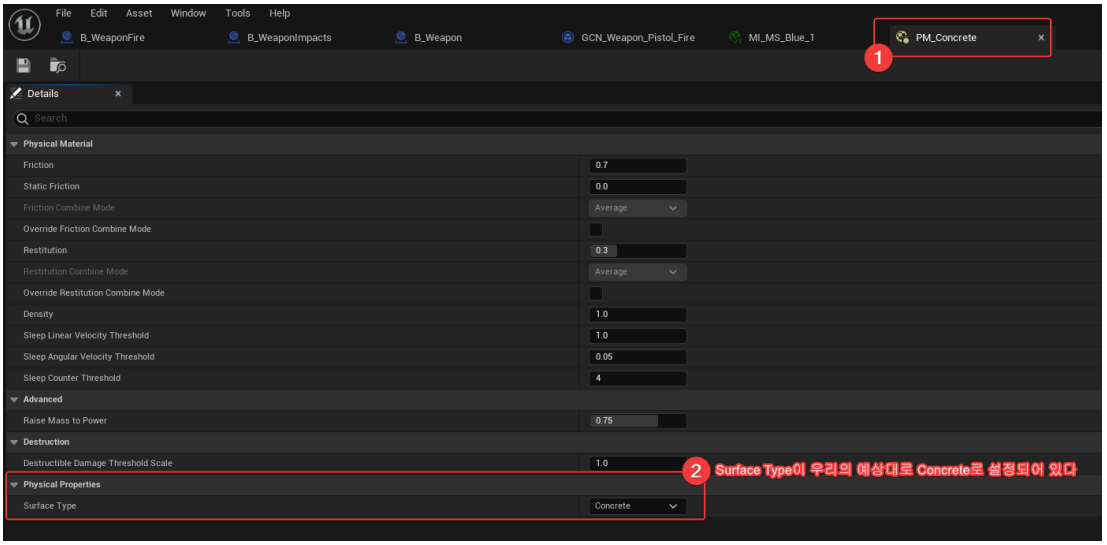

☐ 마지막으로, Niagara Component 전부 파괴되었을 경우, Actor도 자연스럽게 파괴될 수 있도록 처리하는 로직이 앞서 WeaponFire처럼 필요:



☐ 우리는 벽을 대상으로 권총을 쏠 것이다. 벽의 Physical Surface는 제대로 설정되어 있을까:


1
물체를 우선 선택하고!
2
Material로 들어가보자


1
2
원가 Concrete 속성일거 같은데 Physical Material을 들어보자



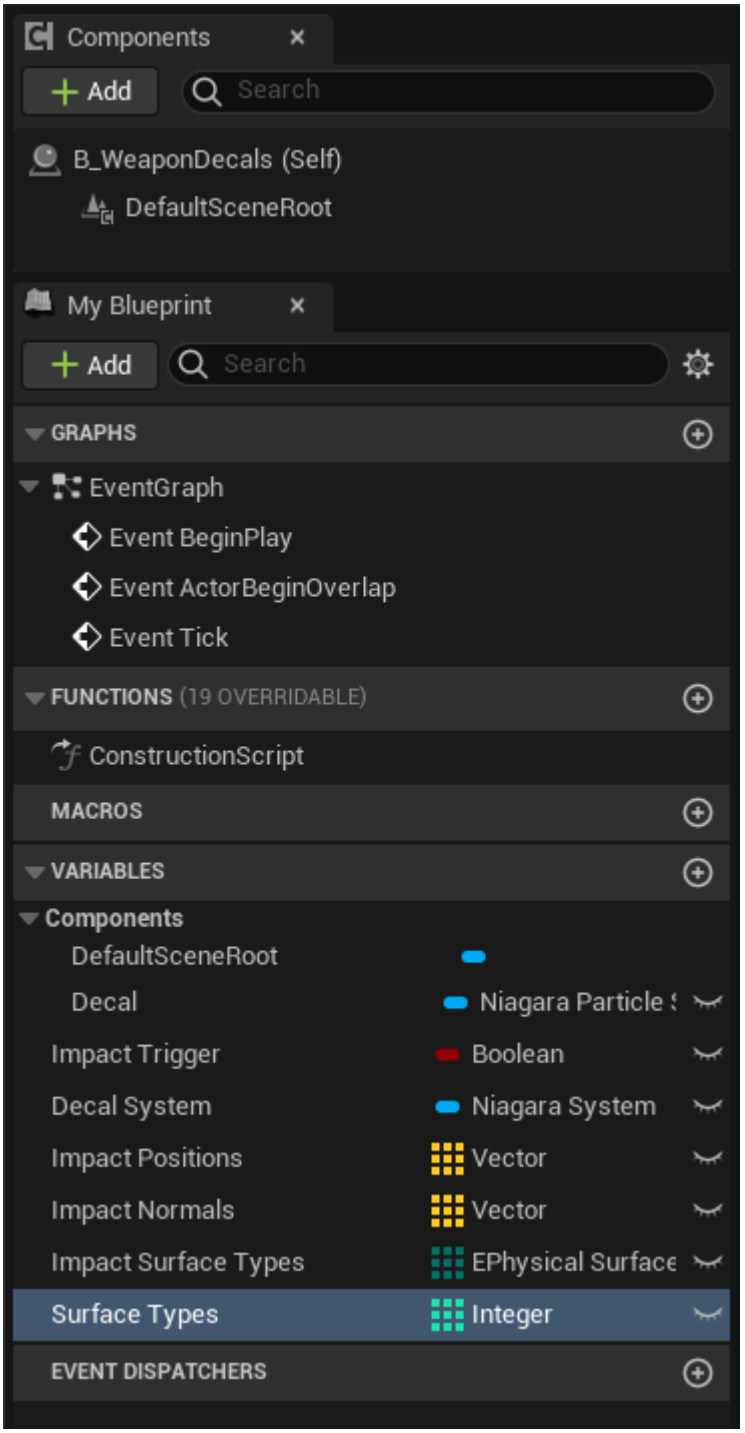

- [ ] 그럼 직접 총을 쏘아보면 아래와 같이 피격 이펙트가 나오는 것을 확인할 수 있다:

https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/7f00c64d-15a5-45e1-8a73-be7146ff0a7a/UnrealEditor_SrMylPjM4Z.mp4

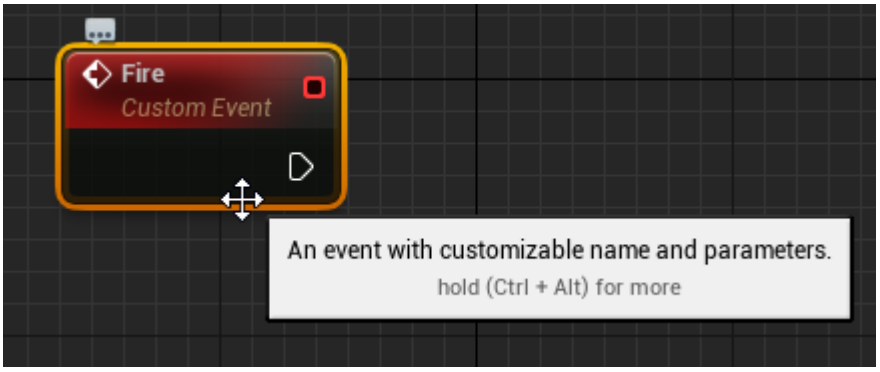
# B_WeaponDecals

▼ **펼치기**

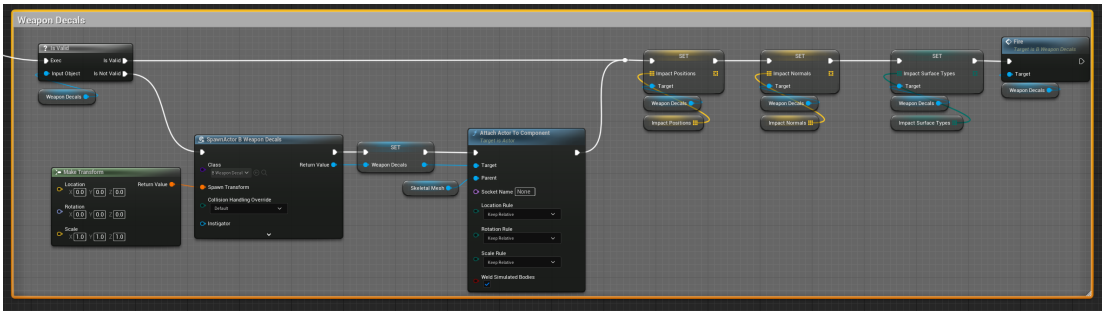
- [ ] 앞서, Fire와 Impacts 처리와 마찬가지로 B_Weapon 로직 작성에 앞서, B_Weapon Decals의 멤버 변수를 정의하자:



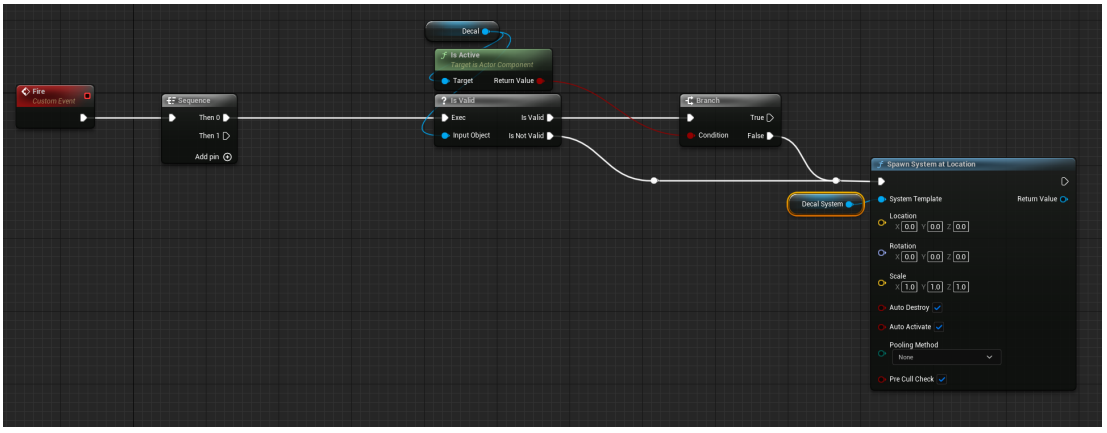

☐ Fire() Custom Event 추가:



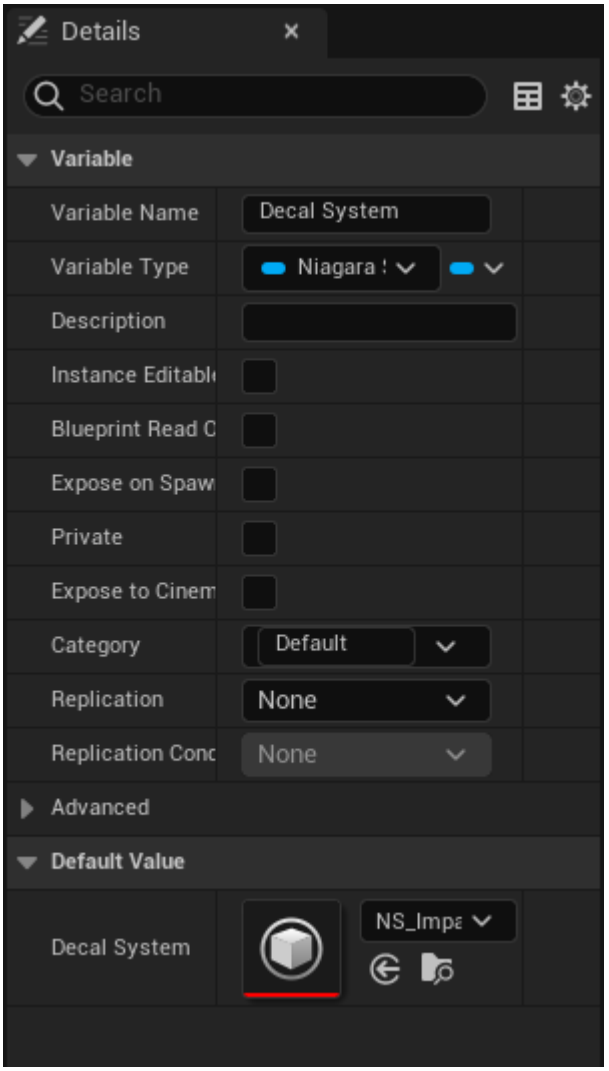

☐ B_Weapon의 Weapon Decals을 구현하자:



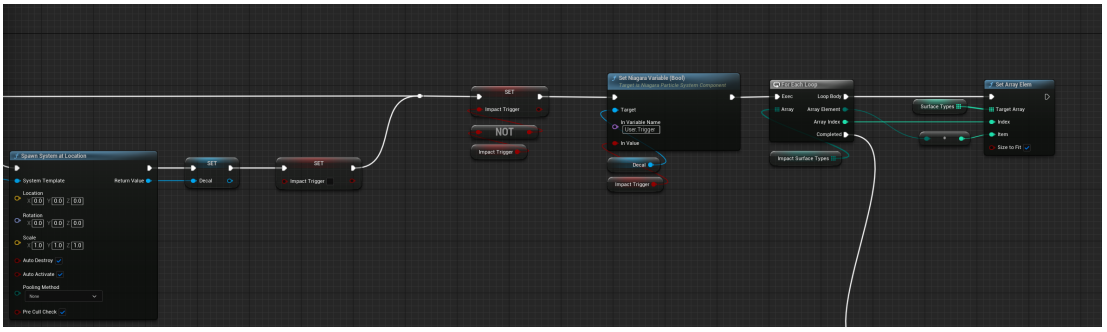

☐ B_WeaponDecals FIre() 로직 완성하자:

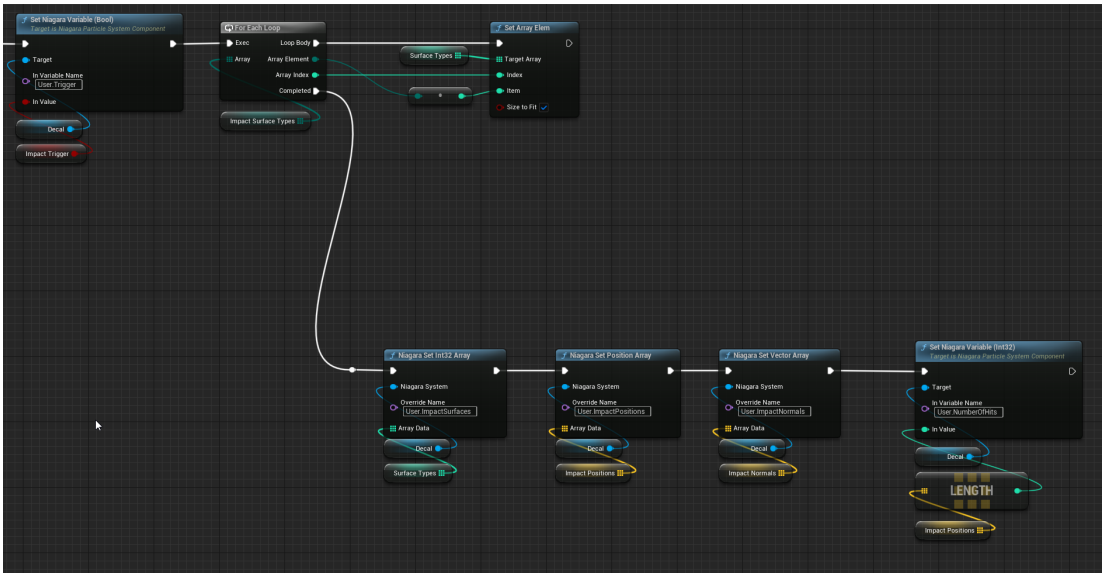



☐ Decal System 기본값 설정 및 임포트:

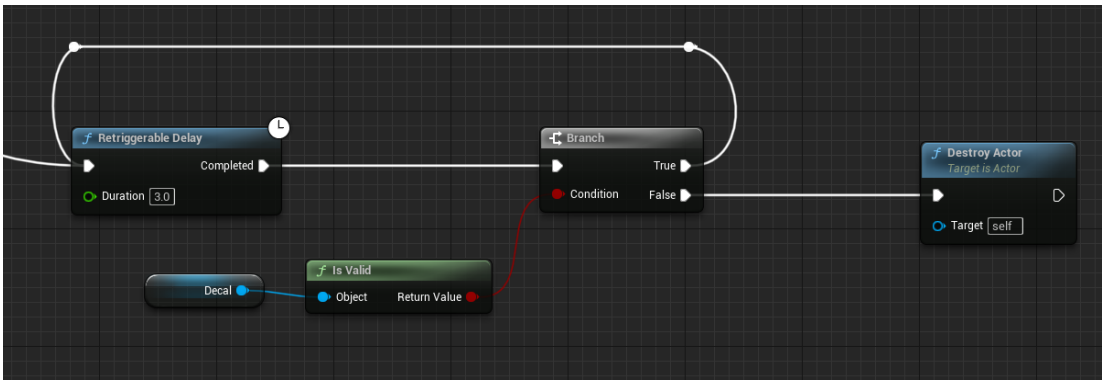



- Decal System: NS_ImpactDecals






- [ ] 마지막 Decal이 파괴되었을 경우, Actor 파괴:




---

- [ ] 아래와 같이 우리가 의도한 최종 결과를 볼 수가 있다:

https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/992d46a1-4934-49da-b842-8be739c5dd0a/UnrealEditor_aOytwAdZqZ.mp4

---

# CommonGame

## ▼ 펼치기

- 우리는 이번 시간과 다음 시간에 걸쳐서, Lyra의 HUD를 클론 해볼 것이다:

- 해당 과정을 통해, **인게임 UI가 Lyra에서는 어떤 구조로 구성되었는지 이해할 수 있다.**

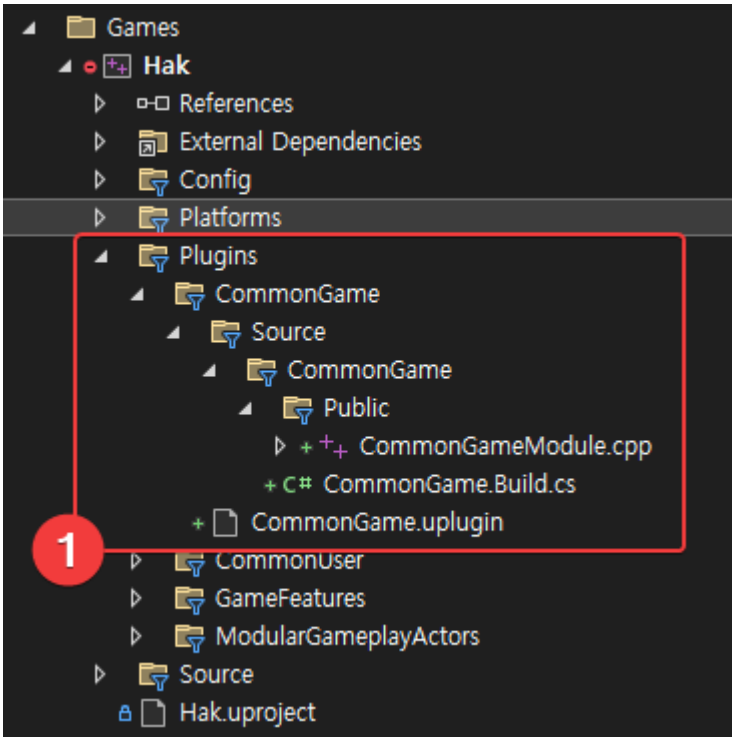
---

- 기본적으로 HUD 및 Lyra의 UI는 Local Player에 기반하고 있다:
  - HUD 및 UI 생성/파괴의 기준을 Local Player에 기반했기 때문에, 우리는 Local Player에 대한 상황 추적을 통해 HUD와 UI 관련 로직을 구성해야 한다
  - 이를 위해, GameInstance, LocalPlayer, PlayerController를 오버라이드해야 한다

---

☐ CommonGame Plugin 생성:

☐ 아래의 구조와 같이 파일과 폴더를 생성해주자:




☐ CommonGame.uplugin:

```
{
    "FileVersion": 3,
    "Version": 1,
    "VersionName": "1.0",
    "FriendlyName": "CommonGame",
    "Description": "",
    "Category": "",
    "CreatedBy": "",
    "CreatedByURL": "",
    "DocsURL": "",
    "MarketplaceURL": "",
    "SupportURL": "",
    "CanContainContent": false,
    "IsBetaVersion": false,
    "Installed": false,
    "Modules": [
        {
            "Name": "CommonGame",
            "Type": "Runtime",
            "LoadingPhase": "Default"
        }
    ],
    "Plugins": [
        {
            "Name": "CommonUI",
            "Enabled": true
        },
        {
            "Name": "CommonUser",
            "Enabled": true
        },
        {
            "Name": "ModularGameplayActors",
            "Enabled": true
        }
    ]
}
```

- [ ] CommonGame.Build.cs:

```
using UnrealBuildTool;

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

        PublicDependencyModuleNames.AddRange(
            new string[]
            {
                "Core",
                "CoreUObject",
                "InputCore",
                "Engine",
                "Slate",
                "SlateCore",
                "UMG",
                "CommonUI",
                "CommonUser",
                "GameplayTags",
                "ModularGameplayActors",
            }
        );

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

        DynamicallyLoadedModuleNames.AddRange(
            new string[]
            {
            }
        );
    }
}
```

- [ ] CommonGameModule.cpp:

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

/**
 * Implements the FCommonGameModule module.
 */
class FCommonGameModule : public IModuleInterface
{
public:
    FCommonGameModule();
    virtual void StartupModule() override;
    virtual void ShutdownModule() override;
private:
};

FCommonGameModule::FCommonGameModule()
{
}

void FCommonGameModule::StartupModule()
{
}

void FCommonGameModule::ShutdownModule()
{
}

IMPLEMENT_MODULE(FCommonGameModule, CommonGame);
```

- 위 세가지 파일이 준비되었으면, GeneratedProjectFiles.bat를 통해 Plugin 생성 가능하다

---

☐ 준비된 CommonGame Plugin을 Hak 프로젝트에 포함하고, HakGame.Build.cs에도 추가하자:

- Hak.uproject:

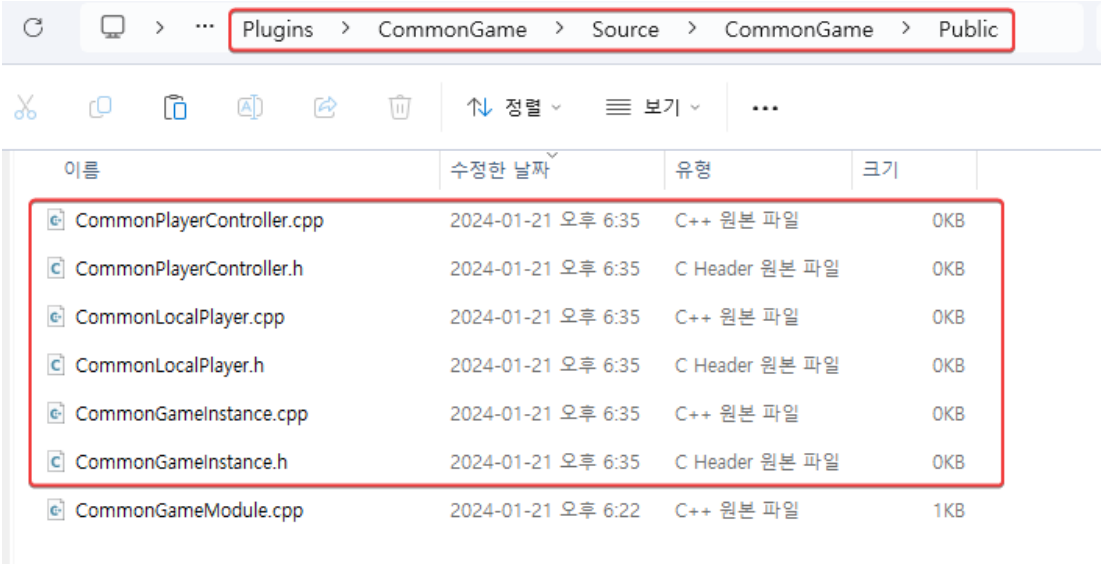

{
"FileVersion": 3,
"EngineAssociation": "",
"Category": "",
"Description": "",
"Modules": [
{
"Name": "HakGame",
"Type": "Runtime",
"LoadingPhase": "Default",
"AdditionalDependencies": [
"Engine"
]
}
],
"Plugins": [
{
"Name": "ModelingToolsEditorMode",
"Enabled": true,
"TargetAllowList": [
"Editor"
]
},
{
"Name": "GameFeatures",
"Enabled": true
},
{
"Name": "ModularGameplay",
"Enabled": true
},
{
"Name": "GameplayAbilities",
"Enabled": true
},
{
"Name": "ShooterCore",
"Enabled": true
},
{
"Name": "AnimationLocomotionLibrary",
"Enabled": true
},
{
"Name": "AnimationWarping",
"Enabled": true
},
{
"Name": "CommonGame",
"Enabled": true
}
]
}

- HakGame.Build.cs:

```
// Copyright Epic Games, Inc. All Rights Reserved.

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",
        });

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

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

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

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

☐ 간단히 빌드를 통해 제대로 구현했는지 체크하자

☐ CommonLocalPlayer, CommonGameInstance, CommonPlayerController 클래스 정의를 위해 파일을 생성해주자:




☐ CommonGameInstance 선언:

```h
#pragma once

#include "Engine/GameInstance.h"
#include "CommonGameInstance.generated.h"

UCLASS(Abstract)
class COMMONGAME_API UCommonGameInstance : public UGameInstance
{
    GENERATED_BODY()
public:
    UCommonGameInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
};
```

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

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

- [ ] CommonLocalPlayer 선언:

```h
#pragma once

#include "Engine/LocalPlayer.h"
#include "CommonLocalPlayer.generated.h"

UCLASS()
class COMMONGAME_API UCommonLocalPlayer : public ULocalPlayer
{
    GENERATED_BODY()
public:

    /** 여기서 주목해보면 좋을 점들은 패턴이다: 우리는 여기서 FObjectInitializer::Get()과 같이 넘기지 않았다 */
    UCommonLocalPlayer();
};
```

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

UCommonLocalPlayer::UCommonLocalPlayer()
    // 여기 Super()로 넘기는 과정에서 FObjectInitializer::Get() 통해 넘겼다
    // - 이는 우리가 UCommonLocalPlayer()를 상속받는 LocalPlayer는 굳이 FObjectInitializer를 신경 쓸 필요가 없다.
    // - 만약 상속받는 LocalPlayer에서 FObjectInitializer를 통해 뭔가 추가 SubObject를 정의한다면 이와 같은 구현을 수정해야 한다
    : Super(FObjectInitializer::Get())
{}
```

  - [ ] 주석 내용을 설명하기

- [ ] CommonPlayerController 선언:

```h
#pragma once

#include "ModularPlayerController.h"
#include "CommonPlayerController.generated.h"

UCLASS()
class COMMONGAME_API ACommonPlayerController : public AModularPlayerController
{
    GENERATED_BODY()
public:
    ACommonPlayerController(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
};
```

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

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

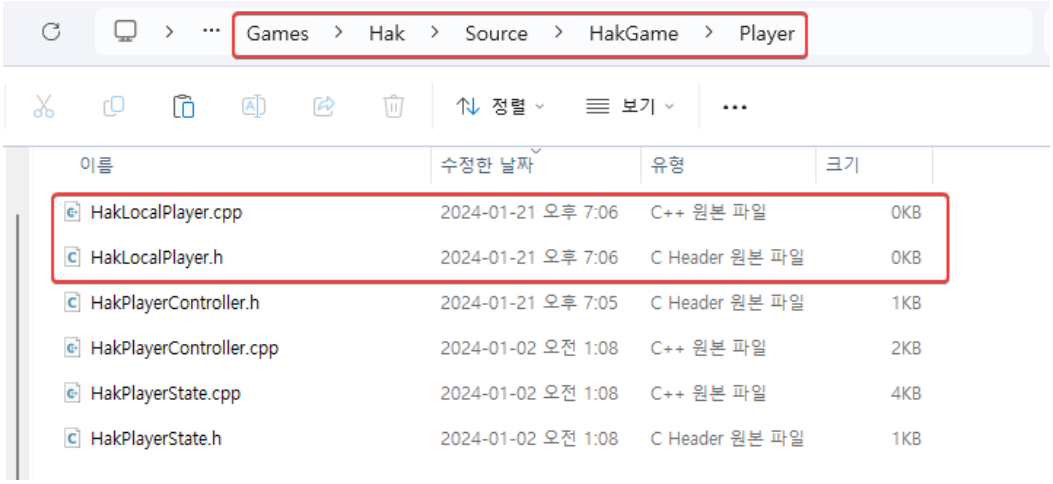
- [ ] HakGameInstance를 CommonGameInstance로 상속 받게 하자:
  - HakGameInstance.h:

```cpp
#pragma once

#include "CommonGameInstance.h"
#include "HakGameInstance.generated.h"

/**
 * GameInstance는 게임 프로세스(.exe)에서 하나만 존재하는 객체로 생각하면 된다 (== High-level Manager Object)
 * - 게임이 켜질때, 만들어지고, 게임이 꺼지기 전까지 살아있다
 * - Editor 상에서는 PIE로 실행 될때마다 하나씩 생성된다: 즉, 에디터에서는 복수개의 GameInstance가 존재 가능하다!
 */
UCLASS()
class UHakGameInstance : public UCommonGameInstance
{
    GENERATED_BODY()
public:
    UHakGameInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * UGameInstance's interfaces
     */
    virtual void Init() override;
    virtual void Shutdown() override;
};
```

- [ ] HakPlayerController를 CommonPlayerController로 상속 받게 하자:
  - HakPlayerController.h:

```cpp
#pragma once

#include "CommonPlayerController.h"
#include "HakPlayerController.generated.h"

/** forward declaration */
class AHakPlayerState;
class UHakAbilitySystemComponent;

/**
 * AHakPlayerController
 * - the base player controller class used by this project
 */
UCLASS()
class AHakPlayerController : public ACommonPlayerController
{
    GENERATED_BODY()
public:
    AHakPlayerController(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * PlayerController interface
     */
    virtual void PostProcessInput(const float DeltaTime, const bool bGamePaused) override;

    /**
     * member methods
     */
    AHakPlayerState* GetHakPlayerState() const;
    UHakAbilitySystemComponent* GetHakAbilitySystemComponent() const;
};
```

- [ ] HakLocalPlayer 정의:
  - [ ] HakLocalPlayer 파일 생성:

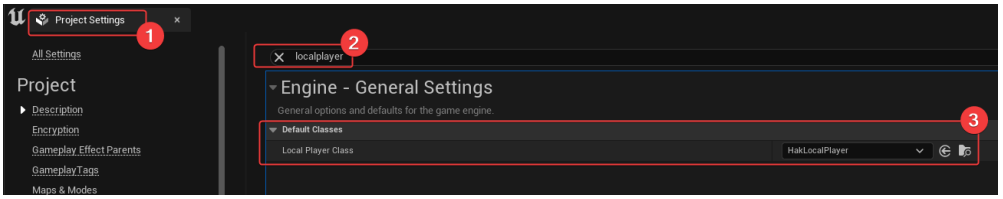



  - [ ] HakLocalPlayer.h/.cpp:

```h
#pragma once

#include "CommonLocalPlayer.h"
#include "HakLocalPlayer.generated.h"

UCLASS()
class UHakLocalPlayer : public UCommonLocalPlayer
{
    GENERATED_BODY()
public:
    UHakLocalPlayer();
};
```

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

UCommonLocalPlayer::UCommonLocalPlayer()
    // 여기 Super()로 넘기는 과정에서 FObjectInitializer::Get() 통해 넘겼다
    // - 이는 우리가 UCommonLocalPlayer()를 상속받는 LocalPlayer는 굳이 FObjectInitializer를 신경 쓸 필요가 없다.
    // - 만약 상속받는 LocalPlayer에서 FObjectInitializer를 통해 뭔가 추가 SubObject를 정의한다면 이와 같은 구현을 수정해야 한다
    : Super(FObjectInitializer::Get())
{}
```

  - [ ] HakLocalPlayer는 HakGameInstance와 HakPlayerController와 달리 우리가 방금 생성했으므로 오버라이드가 필요하다:
    - [ ] 아래와 같이 Project Setting에서 LocalPlayerClass를 HakLocalPlayer로 설정하자:




# GameUIManagerSubsystem

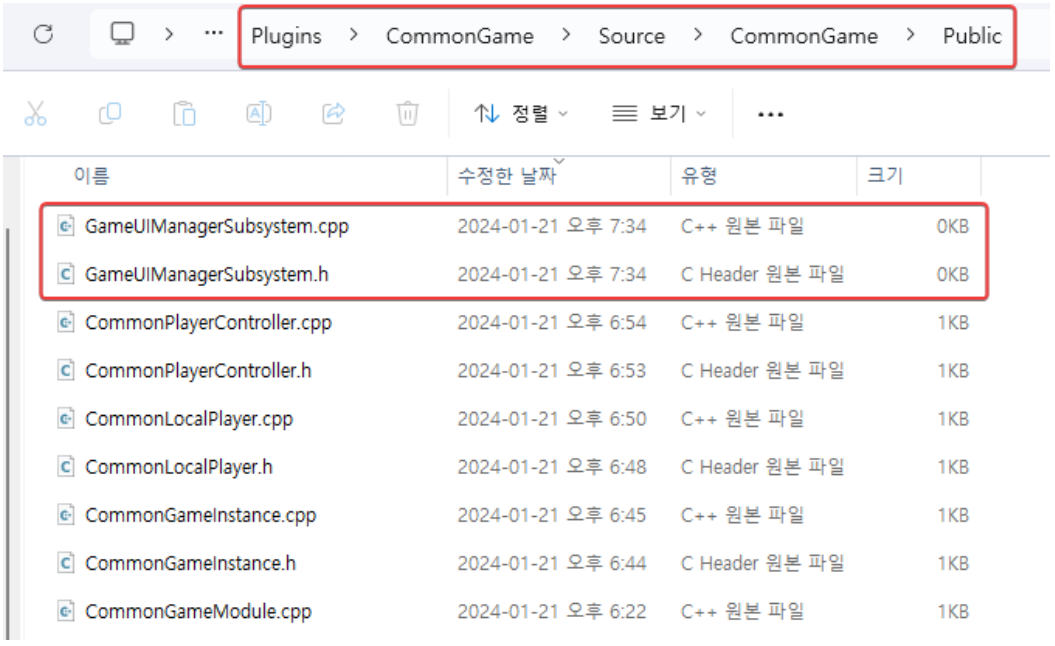
▼ **펼치기**

- Lyra에서 UIManager의 역활을 담당하는 것이 GameUIManagerSubsystem이다.
  - 자세한 것은 하나씩 구현해나가며 이해해보자.
  - 중간 중간에 구현해나가며, 그림으로 컨셉을 이해해보자.

---

☐ GameUIManagerSubsystem.h/.cpp 구현:

☐ CommonGame에 파일을 생성하자:

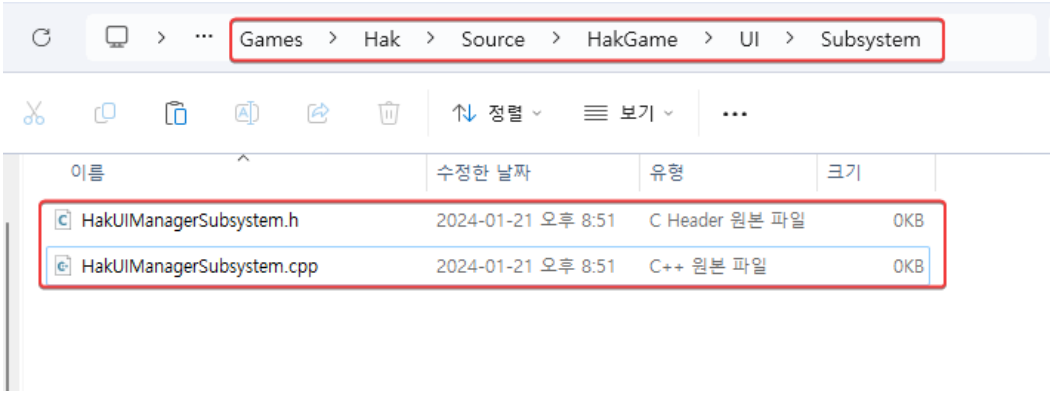



☐ GameUIManagerSubsystem.h/.cpp 구현:

```cpp
#pragma once

#include "Subsystems/GameInstanceSubsystem.h"
#include "GameUIManagerSubsystem.generated.h"

/**
* GameUIManagerSubsystem은 GameInstanceSubsystem의 기반한다
* - 여기서 주목해야 할 점은 UGameUIManagerSubsystem은 UCLASS 속성으로 Abstract라고 정의되어 있다
* - 해당 클래스는 단독으로 사용 불가하며, 누군가 상속받은 Concrete Class로서 활용되야 한다
*/
UCLASS(Abstract, config=Game)
class COMMONGAME_API UGameUIManagerSubsystem : public UGameInstanceSubsystem
{
    GENERATED_BODY()
public:
    UGameUIManagerSubsystem();
};
```

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

UGameUIManagerSubsystem::UGameUIManagerSubsystem()
    : Super()
{}
```

☐ HakUIManagerSubsystem.h/.cpp 생성:




☐ HakUIManagerSubsystem.h/.cpp 구현:

```cpp
#pragma once

#include "GameUIManagerSubsystem.h"
#include "HakUIManagerSubsystem.generated.h"

UCLASS()
class UHakUIManagerSubsystem : public UGameUIManagerSubsystem
{
    GENERATED_BODY()
public:
    UHakUIManagerSubsystem();
};
```

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

UHakUIManagerSubsystem::UHakUIManagerSubsystem()
    : Super()
{}
```

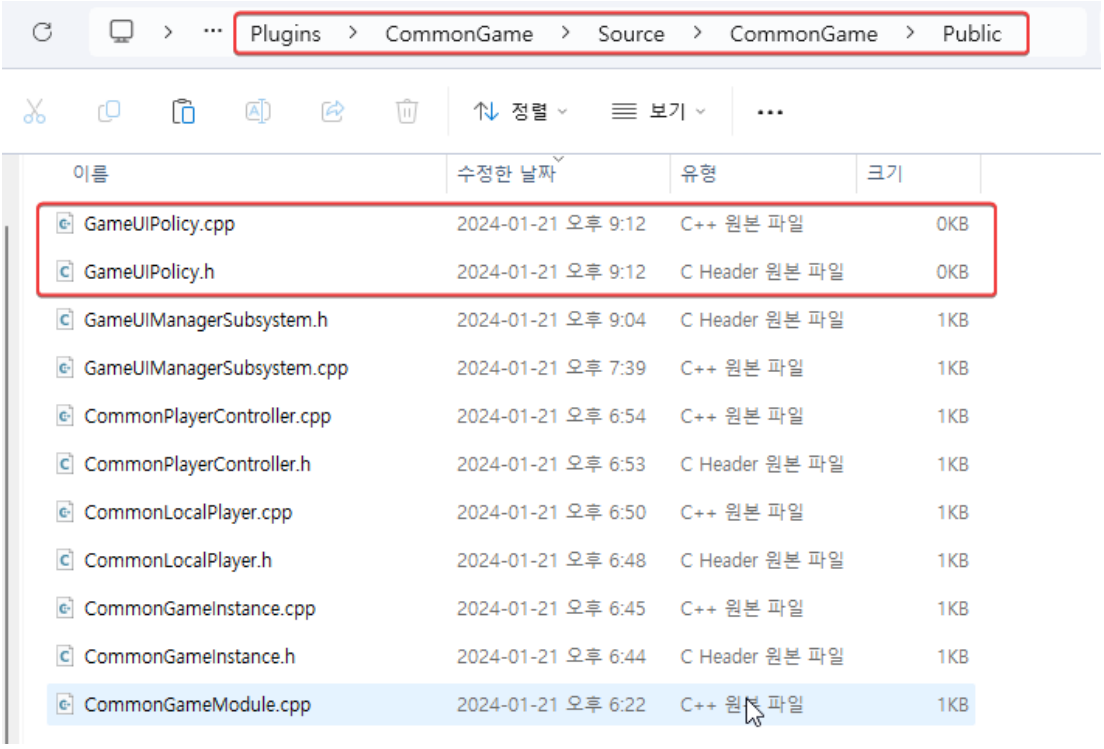
- 이를 통해, HakUIManagerSubsystem가 GameUIManagerSubsystem 상속을 통해 GameUIManagerSubsystem의 Abstract 속성의 문제를 해결하였다

---

☐ UGameUIManagerSubsystem 멤버 변수 구현하자:

```cpp
#pragma once

#include "Subsystems/GameInstanceSubsystem.h"
#include "GameUIManagerSubsystem.generated.h"

/** forward declarations */
class UGameUIPolicy;

/**
 * GameUIManagerSubsystem은 GameInstanceSubsystem의 기반한다
 * - 여기서 주목해야 할 점은 UGameUIManagerSubsystem은 UCLASS 속성으로 Abstract라고 정의되어 있다
 * - 해당 클래스는 단독으로 사용 불가하며, 누군가 상속받은 Concrete Class로서 활용되야 한다
 */
UCLASS(Abstract, config=Game)
class COMMONGAME_API UGameUIManagerSubsystem : public UGameInstanceSubsystem
{
    GENERATED_BODY()
public:
    UGameUIManagerSubsystem();

    UPROPERTY(Transient)
    TObjectPtr<UGameUIPolicy> CurrentPolicy = nullptr;

    /**
     * default UI policy를 생성할 class
     * - 우리는 해당 클래스는 B_BttGameUIPolicy로 지정할 것이다
     */
    UPROPERTY(Config, EditAnywhere)
    TSoftClassPtr<UGameUIPolicy> DefaultUIPolicyClass;
};
```

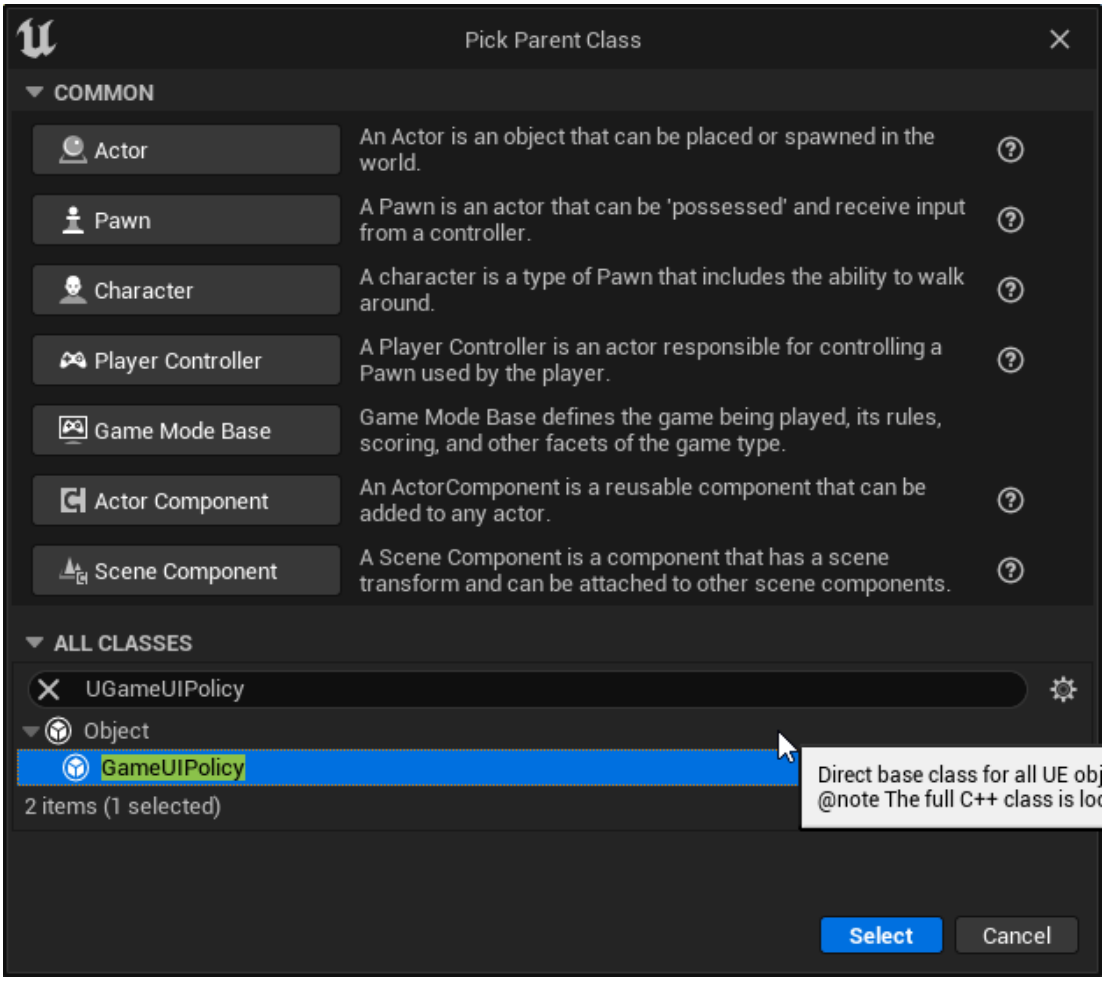
☐ GameUIPolicy.h/.cpp 생성 및 구현:

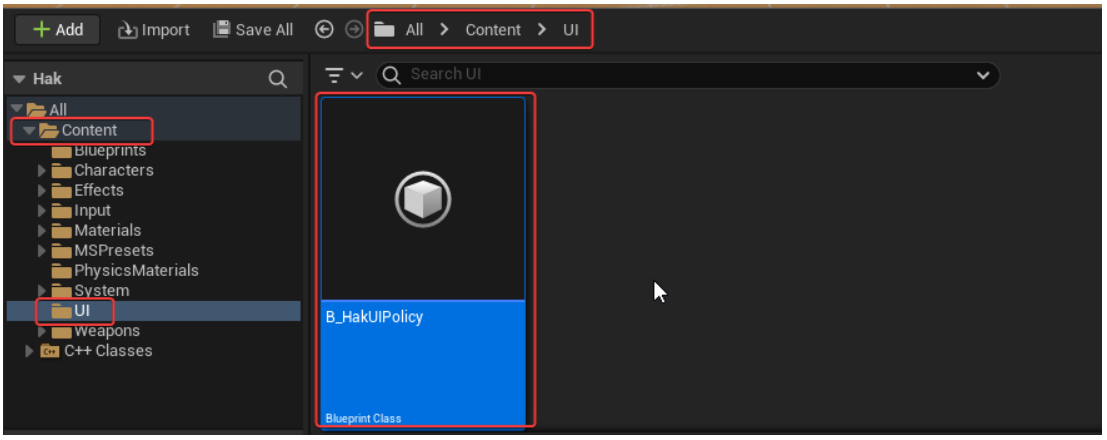



```cpp
#pragma once

#include "UObject/Object.h"
#include "UObject/UObjectGlobals.h"

#include "GameUIPolicy.generated.h"

/**
 * UGameUIPolicy가 Abstract라는 것을 기억하자
 */
UCLASS(Abstract, Blueprintable)
class COMMONGAME_API UGameUIPolicy : public UObject
{
    GENERATED_BODY()
public:
};
```

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

☐ UGameUIPolicy를 상속 받는 B_HakUIPolicy를 생성하자:




☐ B_HakUIPolicy를 HakUIManagerSubsystem의 DefaultUIPolicyClass로 DefaultGame.ini를 통해 설정하자:

```
[/Script/EngineSettings.GeneralProjectSettings]
ProjectID=0BAF38A448591EAFD8CD9891E38E980D

[/Script/GameplayAbilities.AbilitySystemGlobals]
GlobalGameplayCueManagerClass=/Script/HakGame.HakGameplayCueManager

[/Script/Engine.AssetManagerSettings]
-PrimaryAssetTypesToScan=(PrimaryAssetType="Map",AssetBaseClass=/Script/Engine.World,bHasBlueprintClasses=False,bIsEditorOnly=Tr
-PrimaryAssetTypesToScan=(PrimaryAssetType="PrimaryAssetLabel",AssetBaseClass=/Script/Engine.PrimaryAssetLabel,bHasBlueprintClas
+PrimaryAssetTypesToScan=(PrimaryAssetType="Map",AssetBaseClass="/Script/Engine.World",bHasBlueprintClasses=False,bIsEditorOnly=
+PrimaryAssetTypesToScan=(PrimaryAssetType="PrimaryAssetLabel",AssetBaseClass="/Script/Engine.PrimaryAssetLabel",bHasBlueprintCl
+PrimaryAssetTypesToScan=(PrimaryAssetType="HakUserFacingExperienceDefinition",AssetBaseClass="/Script/HakGame.HakUserFacingExpe
+PrimaryAssetTypesToScan=(PrimaryAssetType="HakExperienceDefinition",AssetBaseClass="/Script/HakGame.HakExperienceDefinition",bH
bOnlyCookProductionAssets=False
bShouldManagerDetermineTypeAndName=False
bShouldGuessTypeAndNameInEditor=True
bShouldAcquireMissingChunksOnLoad=False
bShouldWarnAboutInvalidAssets=True
MetaDataTagsForAssetRegistry=()

[/Script/GameFeatures.GameFeaturesSubsystemSettings]
GameFeaturesManagerClassName=/Script/HakGame.HakGameplayFeaturePolicy

[/Script/HakGame.HakUIManagerSubsystem]
DefaultUIPolicyClass=/Game/UI/B_HakUIPolicy.B_HakUIPolicy_C
```

1

☐ GameUIManagerSubsystem의 GameInstanceSubsystem의 interfaces를 구현하자:

```cpp
#pragma once

#include "Subsystems/GameInstanceSubsystem.h"
#include "GameUIManagerSubsystem.generated.h"

/** forward declarations */
class UGameUIPolicy;

/**
 * GameUIManagerSubsystem은 GameInstanceSubsystem의 기반한다
 * - 여기서 주목해야 할 점은 UGameUIManagerSubsystem은 UCLASS 속성으로 Abstract라고 정의되어 있다
 * - 해당 클래스는 단독으로 사용 불가하며, 누군가 상속받은 Concrete Class로서 활용되야 한다
 */
UCLASS(Abstract, config=Game)
class COMMONGAME_API UGameUIManagerSubsystem : public UGameInstanceSubsystem
{
    GENERATED_BODY()
public:
    UGameUIManagerSubsystem();

    /**
     * UGameInstanceSubsystem's interfaces
     */
    virtual void Initialize(FSubsystemCollectionBase& Collection) override;
    virtual void Deinitialize() override;
    virtual bool ShouldCreateSubsystem(UObject* Outer) const override;

    UPROPERTY(Transient)
    TObjectPtr<UGameUIPolicy> CurrentPolicy = nullptr;

    /**
     * default UI policy를 생성할 class
     * - 우리는 해당 클래스는 B_BttGameUIPolicy로 지정할 것이다
     */
    UPROPERTY(Config, EditAnywhere)
    TSoftClassPtr<UGameUIPolicy> DefaultUIPolicyClass;
};
```

☐ Initialize():

- ☐ SwitchToPolicy():

```cpp
/**
 * GameUIManagerSubsystem은 GameInstanceSubsystem의 기반한다
 * - 여기서 주목해야 할 점은 UGameUIManagerSubsystem은 UCLASS 속성으로 Abstract라고 정의되어 있다
 * - 해당 클래스는 단독으로 사용 불가하며, 누군가 상속받은 Concrete Class로서 활용되야 한다
 */
UCLASS(Abstract, config=Game)
class COMMONGAME_API UGameUIManagerSubsystem : public UGameInstanceSubsystem
{
    GENERATED_BODY()
public:
    UGameUIManagerSubsystem();

    void SwitchToPolicy(UGameUIPolicy* InPolicy);

    /**
     * UGameInstanceSubsystem's interfaces
     */
    virtual void Initialize(FSubsystemCollectionBase& Collection) override;
    virtual void Deinitialize() override;
    virtual bool ShouldCreateSubsystem(UObject* Outer) const override;

    UPROPERTY(Transient)
    TObjectPtr<UGameUIPolicy> CurrentPolicy = nullptr;

    /**
     * default UI policy를 생성할 class
     * - 우리는 해당 클래스는 B_BttGameUIPolicy로 지정할 것이다
     */
    UPROPERTY(Config, EditAnywhere)
    TSoftClassPtr<UGameUIPolicy> DefaultUIPolicyClass;
};
```

```cpp
#include "GameUIManagerSubsystem.h"
#include "GameUIPolicy.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(GameUIManagerSubsystem)

UGameUIManagerSubsystem::UGameUIManagerSubsystem()
    : Super()
{}

void UGameUIManagerSubsystem::SwitchToPolicy(UGameUIPolicy* InPolicy)
{
    if (CurrentPolicy != InPolicy)
    {
        CurrentPolicy = InPolicy;
    }
}
```

- [ ] 로직 완성:

```cpp
void UGameUIManagerSubsystem::Initialize(FSubsystemCollectionBase& Collection)
{
    Super::Initialize(Collection);

    // CurrentPolicy가 설정되어 있지 않고, DefaultUIPolicyClass가 제대로 설정되어 있을 경우 (우리는 HakUIPolicy겠지?)
    if (!CurrentPolicy && !DefaultUIPolicyClass.IsNull())
    {
        // UIPolicyClass는 BP에셋이기에, 로딩해야 함
        TSubclassOf<UGameUIPolicy> PolicyClass = DefaultUIPolicyClass.LoadSynchronous();

        // UIPolicyClass를 통해 NewObject로 인스턴싱해서 CurrentPolicy에 설정
        SwitchToPolicy(NewObject<UGameUIPolicy>(this, PolicyClass));
    }
}
```

- [ ] Deinitialize():

```cpp
void UGameUIManagerSubsystem::Deinitialize()
{
    Super::Deinitialize();
    SwitchToPolicy(nullptr);
}
```

- [ ] ShouldCreateSubsystem()

```cpp
bool UGameUIManagerSubsystem::ShouldCreateSubsystem(UObject* Outer) const
{
    // 우선 DedicatedServer의 경우, GameUIManagerSubsystem을 활성화하지 않음
    if (!CastChecked<UGameInstance>(Outer)->IsDedicatedServerInstance())
    {
        // 만약 어떠한 Subsystem도 UGameUIManagerSubsystem을 상속받지 않았다면, 해당 GameInstanceSubsystem은 활성화 안함
        TArray<UClass*> ChildClasses;
        GetDerivedClasses(GetClass(), ChildClasses, false);

        return ChildClasses.Num() == 0;
    }

    return false;
}
```

- [ ] FSubsystemCollectionBase::AddAndInitializeSubsystem()

```cpp
USubsystem* FSubsystemCollectionBase::AddAndInitializeSubsystem(UClass* SubsystemClass)
{
    TGuardValue<bool> PopulatingGuard(bPopulating, true);

    if (!SubsystemMap.Contains(SubsystemClass))
    {
        // Only add instances for non abstract Subsystems
        if (SubsystemClass && !SubsystemClass->HasAllClassFlags(CLASS_Abstract))
        {
            // Catch any attempt to add a subsystem of the wrong type
            checkf(SubsystemClass->IsChildOf(BaseType), TEXT("ClassType (%s) must be a subclass of BaseType(%s)."), *SubsystemClass->GetName(), *BaseType->GetName());

            // Do not create instances of classes aren't authoritative
            if (SubsystemClass->GetAuthoritativeClass() != SubsystemClass)
            {
                return nullptr;
            }

            UE_SCOPED_ENGINE_ACTIVITY(TEXT("Initializing Subsystem %s"), *SubsystemClass->GetName());

            const USubsystem* CDO = SubsystemClass->GetDefaultObject<USubsystem>();
            if (CDO->ShouldCreateSubsystem(Outer))
            {
                USubsystem* Subsystem = NewObject<USubsystem>(Outer, SubsystemClass);
                SubsystemMap.Add(SubsystemClass,Subsystem);
                Subsystem->InternalOwningSubsystem = this;
                Subsystem->Initialize(*this);
                return Subsystem;
            }

            UE_LOG(LogSubsystemCollection, VeryVerbose, TEXT("Subsystem does not exist, but CDO choose to not create (%s)"), *SubsystemClass->GetName());
        }
        return nullptr;
    }

    UE_LOG(LogSubsystemCollection, VeryVerbose, TEXT("Subsystem already exists (%s)"), *SubsystemClass->GetName());
    return SubsystemMap.FindRef(SubsystemClass);
}
```

- [ ] 우리가 의도한 대로 잘 작동하는지 확인해보자:

```cpp
17  void UGameUIManagerSubsystem::Initialize(FSubsystemCollectionBase& Collection)
18  {
19      Super::Initialize(Collection);
20
21      // CurrentPolicy가 설정되어 있지 않고, DefaultUIPolicyClass가 제대로 설정되어 있을 경우 (우리는 HakUIPolicy겠지?)
22      if (!CurrentPolicy && !DefaultUIPolicyClass.IsNull())
23      {
24          // UIPolicyClass는 BP에셋이기에, 로딩해야 함
25          TSubclassOf<UGameUIPolicy> PolicyClass = DefaultUIPolicyClass.LoadSynchronous();
26
27          // UIPolicyClass를 통해 NewObject로 인스턴싱해서 CurrentPolicy에 설정
28          SwitchToPolicy(NewObject<UGameUIPolicy>(this, PolicyClass));
29      }
30  }
```

1

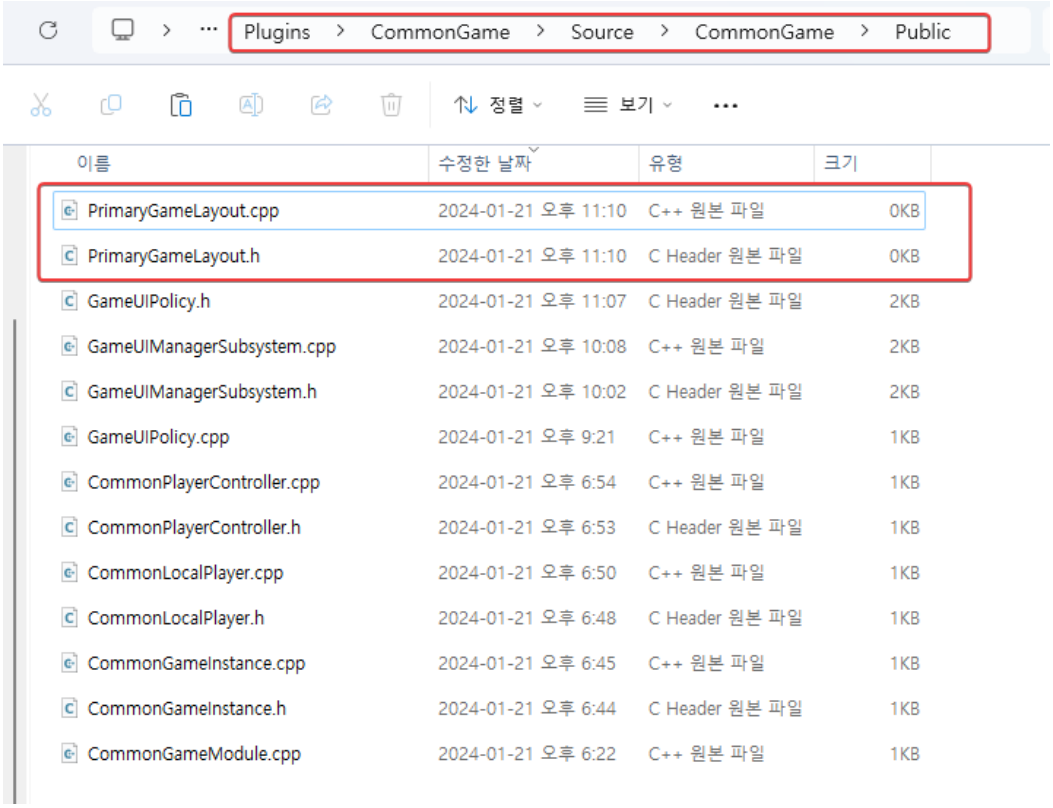
# GameUIPolicy

▼ **펼치기**

- [ ] GameUIPolicy 멤벼 변수:

```cpp
#pragma once

#include "UObject/Object.h"
#include "UObject/UObjectGlobals.h"

#include "GameUIPolicy.generated.h"

/** forward declarations */
class UPrimaryGameLayout;

USTRUCT()
struct FRootViewportLayoutInfo
{
    GENERATED_BODY()
public:
    UPROPERTY(Transient)
    TObjectPtr<ULocalPlayer> LocalPlayer = nullptr;

    UPROPERTY(Transient)
    TObjectPtr<UPrimaryGameLayout> RootLayout = nullptr;

    UPROPERTY(Transient)
    bool bAddedToViewport = false;
};

/**
 * UGameUIPolicy가 Abstract라는 것을 기억하자
 */
UCLASS(Abstract, Blueprintable)
class COMMONGAME_API UGameUIPolicy : public UObject
{
    GENERATED_BODY()
public:
    /** LocalPlayer에 바인딩된 UI의 Layout으로 생각하면 된다 (아직 의미가 모호할 수 있는데, 하나씩 구현해보면서, 어떤 느낌인가 더 와닫을 것이다) */
    UPROPERTY(EditAnywhere)
    TSoftClassPtr<UPrimaryGameLayout> LayoutClass;

    /** 보통 싱글 게임에서는 LocalPlayer-PrimaryGameLayout 하나만 있겠지만, 멀티플레이의 경우, 복수개 가능하다 (리플레이를 생각해보자) */
    UPROPERTY(Transient)
    TArray<FRootViewportLayoutInfo> RootViewportLayouts;
};
```

- [ ] PrimaryGameLayout 구현:
  - [ ] PrimaryGameLayout 파일 생성:

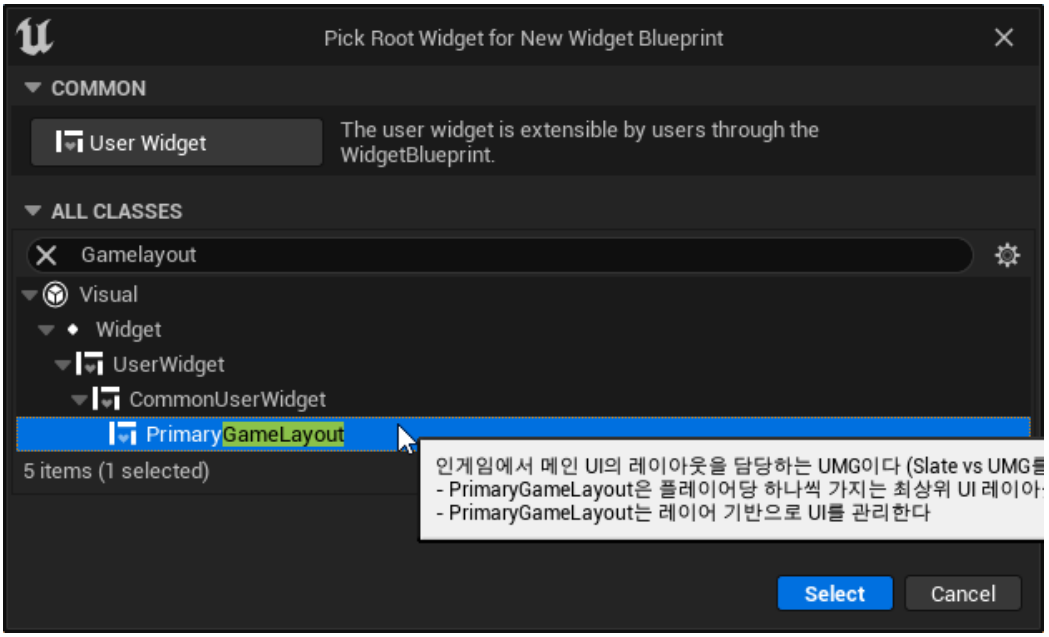



- [ ] PrimaryGameLayout.h/.cpp:

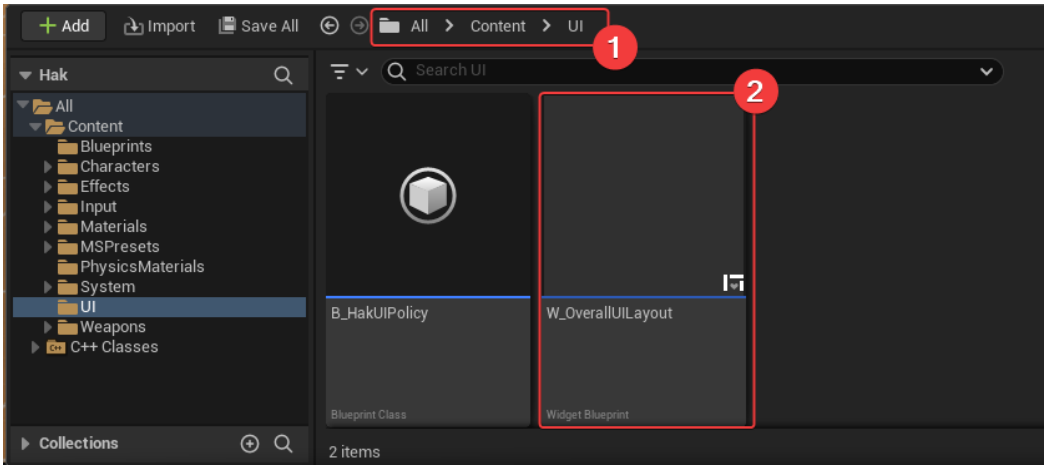
```cpp
#pragma once

#include "CommonUserWidget.h"
#include "PrimaryGameLayout.generated.h"

/**
 * 인게임에서 메인 UI의 레이아웃을 담당하는 UMG이다 (Slate vs UMG를 이해하장 - 간단하게 UObject을 기반하는가 아닌가 차이)
 * - PrimaryGameLayout은 플레이어당 하나씩 가지는 최상위 UI 레이아웃으로 이해하면 된다
 * - PrimaryGameLayout는 레이어 기반으로 UI를 관리한다
 */
UCLASS(Abstract)
class COMMONGAME_API UPrimaryGameLayout : public UCommonUserWidget
{
    GENERATED_BODY()
public:
    /** 해당 클래스는 Abstract로 설정되었으므로 굳이 FObjectInitializer::Get()할 필요는 없다 */
    UPrimaryGameLayout(const FObjectInitializer& ObjectInitializer);
};
```

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

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

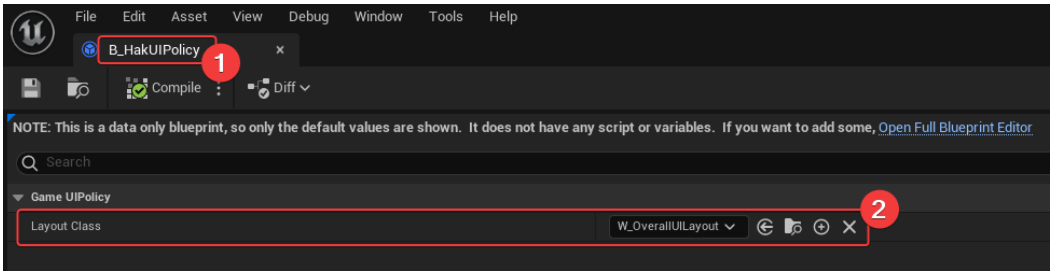
- [ ] W_OverallUILayout 생성:
  - Widget Blueprint 생성하자:




- W_OverallUILayout 생성하자:




☐ B_HakUIPolicy에 LayoutClass를 방금 생성한 W_OverallUILayout 설정하자:



# PrimaryGameLayout과 LocalPlayer 연동:

▼ **펼치기**

☐ CommonGameInstance를 통해 AddLocalPlayer/RemoveLocalPlayer를 Hookup하자:

```cpp
#pragma once

#include "Engine/GameInstance.h"
#include "CommonGameInstance.generated.h"

UCLASS(Abstract)
class COMMONGAME_API UCommonGameInstance : public UGameInstance
{
    GENERATED_BODY()
public:
    UCommonGameInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * GameInstance's interfaces
     */
    virtual int32 AddLocalPlayer(ULocalPlayer* NewPlayer, FPlatformUserId UserId) override;
    virtual bool RemoveLocalPlayer(ULocalPlayer* ExistingPlayer) override;

    /** 메인 로컬 플레이어를 캐싱한다 */
    TWeakObjectPtr<ULocalPlayer> PrimaryPlayer;
};
```

☐ AddLocalPlayer():

```cpp
int32 UCommonGameInstance::AddLocalPlayer(ULocalPlayer* NewPlayer, FPlatformUserId UserId)
{
    // GameInstance에 관리하는 Player 컨테이너에 추가한다 (ReturnVal은 참고로 새로 추가된 Index이다)
    int32 ReturnVal = Super::AddLocalPlayer(NewPlayer, UserId);
    if (ReturnVal != INDEX_NONE)
    {
        // PrimaryPlayer는 처음만 캐싱하는듯하다 (무조건 처음 세팅되면 메인인가 Index==0을 메인으로 잡긴하던데?)
        if (!PrimaryPlayer.IsValid())
        {
            PrimaryPlayer = NewPlayer;
        }

        // GameUIManagerSubsystem를 통해 NotifyPlayerAdded() 호출로 GameLayout을 추가한다
        GetSubsystem<UGameUIManagerSubsystem>()->NotifyPlayerAdded(Cast<UCommonLocalPlayer>(NewPlayer));
    }
    return ReturnVal;
}
```

☐ UGameUIManagerSubsystem::NotifyPlayerAdded():

```h
#pragma once

#include "Subsystems/GameInstanceSubsystem.h"
#include "GameUIManagerSubsystem.generated.h"

/** forward declarations */
class UGameUIPolicy;
class UCommonLocalPlayer;

/**
 * GameUIManagerSubsystem은 GameInstanceSubsystem의 기반한다
 * - 여기서 주목해야 할 점은 UGameUIManagerSubsystem은 UCLASS 속성으로 Abstract라고 정의되어 있다
 * - 해당 클래스는 단독으로 사용 불가하며, 누군가 상속받은 Concrete Class로서 활용되야 한다
 */
UCLASS(Abstract, config=Game)
class COMMONGAME_API UGameUIManagerSubsystem : public UGameInstanceSubsystem
{
    GENERATED_BODY()
public:
    UGameUIManagerSubsystem();

    void SwitchToPolicy(UGameUIPolicy* InPolicy);

    /**
     * UGameInstanceSubsystem's interfaces
     */
    virtual void Initialize(FSubsystemCollectionBase& Collection) override;
    virtual void Deinitialize() override;
    virtual bool ShouldCreateSubsystem(UObject* Outer) const override;

    /**
     * UGameUIManagerSubsystem's interfaces
     */
    virtual void NotifyPlayerAdded(UCommonLocalPlayer* LocalPlayer);
    virtual void NotifyPlayerRemoved(UCommonLocalPlayer* LocalPlayer);
    virtual void NotifyPlayerDestroyed(UCommonLocalPlayer* LocalPlayer);

    UPROPERTY(Transient)
    TObjectPtr<UGameUIPolicy> CurrentPolicy = nullptr;

    /**
     * default UI policy를 생성할 class
     * - 우리는 해당 클래스는 B_BttGameUIPolicy로 지정할 것이다
     */
    UPROPERTY(Config, EditAnywhere)
    TSoftClassPtr<UGameUIPolicy> DefaultUIPolicyClass;
};
```

- [ ] NotifyPlayerAdded():

```cpp
void UGameUIManagerSubsystem::NotifyPlayerAdded(UCommonLocalPlayer* LocalPlayer)
{
    if (ensure(LocalPlayer) && CurrentPolicy)
    {
        CurrentPolicy->NotifyPlayerAdded(LocalPlayer);
    }
}
```

- [ ] UGameUIPolicy::NotifyPlayerAdded():
  - [ ] UGameUIPolicy의 Interface 추가:

```cpp
#pragma once

#include "UObject/Object.h"
#include "UObject/UObjectGlobals.h"

#include "GameUIPolicy.generated.h"

/** forward declarations */
class UPrimaryGameLayout;
class UCommonLocalPlayer;

USTRUCT()
struct FRootViewportLayoutInfo
{
    GENERATED_BODY()
public:
    UPROPERTY(Transient)
    TObjectPtr<ULocalPlayer> LocalPlayer = nullptr;

    UPROPERTY(Transient)
    TObjectPtr<UPrimaryGameLayout> RootLayout = nullptr;

    UPROPERTY(Transient)
    bool bAddedToViewport = false;
};

/**
 * UGameUIPolicy가 Abstract라는 것을 기억하자
 */
UCLASS(Abstract, Blueprintable)
class COMMONGAME_API UGameUIPolicy : public UObject
{
    GENERATED_BODY()
public:
    void NotifyPlayerAdded(UCommonLocalPlayer* LocalPlayer);
    void NotifyPlayerRemoved(UCommonLocalPlayer* LocalPlayer);
    void NotifyPlayerDestroyed(UCommonLocalPlayer* LocalPlayer);

    /** LocalPlayer에 바인딩된 UI의 Layout으로 생각하면 된다 (아직 의미가 모호할 수 있는데, 하나씩 구현해보면서, 어떤 느낌인가 더 와닫을 것이다) */
    UPROPERTY(EditAnywhere)
    TSoftClassPtr<UPrimaryGameLayout> LayoutClass;

    /** 보통 싱글 게임에서는 LocalPlayer-PrimaryGameLayout 하나만 있겠지만, 멀티플레이의 경우, 복수개 가능하다 (리플레이를 생각해보자) */
    UPROPERTY(Transient)
    TArray<FRootViewportLayoutInfo> RootViewportLayouts;
};
```

- [ ] UGameUIPolicy::NotifyPlayerAdded():
  - [ ] OnPlayerControllerSet 추가:

```cpp
#pragma once

#include "Engine/LocalPlayer.h"
#include "Delegates/Delegate.h"
#include "CommonLocalPlayer.generated.h"

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;
};
```

  - [ ] NotifyPlayerRemoved():

```cpp
void UGameUIPolicy::NotifyPlayerRemoved(UCommonLocalPlayer* LocalPlayer)
{
    if (FRootViewportLayoutInfo* LayoutInfo = RootViewportLayouts.FindByKey(LocalPlayer))
    {
        RemoveLayoutFromViewport(LocalPlayer, LayoutInfo->RootLayout);
        LayoutInfo->bAddedToViewport = false;
    }
}
```

  - [ ] Add/RemoveLayoutFromViewport() 추가:

```cpp
/**
 * UGameUIPolicy가 Abstract라는 것을 기억하자
 */
UCLASS(Abstract, Blueprintable)
class COMMONGAME_API UGameUIPolicy : public UObject
{
    GENERATED_BODY()
public:
    void AddLayoutToViewport(UCommonLocalPlayer* LocalPlayer, UPrimaryGameLayout* Layout);
    void RemoveLayoutFromViewport(UCommonLocalPlayer* LocalPlayer, UPrimaryGameLayout* Layout);

    void NotifyPlayerAdded(UCommonLocalPlayer* LocalPlayer);
    void NotifyPlayerRemoved(UCommonLocalPlayer* LocalPlayer);
    void NotifyPlayerDestroyed(UCommonLocalPlayer* LocalPlayer);

    /** LocalPlayer에 바인딩된 UI의 Layout으로 생각하면 된다 (아직 의미가 모호할 수 있는데, 하나씩 구현해보면서, 어떤 느낌인가 더 와닫을 것이다) */
    UPROPERTY(EditAnywhere)
    TSoftClassPtr<UPrimaryGameLayout> LayoutClass;

    /** 보통 싱글 게임에서는 LocalPlayer-PrimaryGameLayout 하나만 있겠지만, 멀티플레이의 경우, 복수개 가능하다 (리플레이를 생각해보자) */
    UPROPERTY(Transient)
    TArray<FRootViewportLayoutInfo> RootViewportLayouts;
};
```

- [ ] RemoveLayoutFromViewport():

```cpp
#include "GameUIPolicy.h"
#include "CommonLocalPlayer.h"
#include "PrimaryGameLayout.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(GameUIPolicy)

void UGameUIPolicy::AddLayoutToViewport(UCommonLocalPlayer* LocalPlayer, UPrimaryGameLayout* Layout)
{

}

void UGameUIPolicy::RemoveLayoutFromViewport(UCommonLocalPlayer* LocalPlayer, UPrimaryGameLayout* Layout)
{
    // UCommonUserWidget의 SlateWidget을 가져와서 Parent와 연결고리를 끊어 놓는다
    TWeakPtr<SWidget> LayoutSlateWidget = Layout->GetCachedWidget();
    if (LayoutSlateWidget.IsValid())
    {
        Layout->RemoveFromParent();
    }
}
```

- [ ] AddLayoutToViewport():

```cpp
void UGameUIPolicy::AddLayoutToViewport(UCommonLocalPlayer* LocalPlayer, UPrimaryGameLayout* Layout)
{
    // CommonUserWidget에 PlayerContext를 설정
    Layout->SetPlayerContext(FLocalPlayerContext(LocalPlayer));

    // 해당 Layout에 우선순위를 높게 설정한다 (1000)
    // - AddToPlayerScreen를 통해 Widget을 붙인다
    Layout->AddToPlayerScreen(1000);
}
```

- [ ] CreateLayoutWidget():
  - [ ] FRootViewportLayoutInfo Constructor 정의하자:

```cpp
USTRUCT()
struct FRootViewportLayoutInfo
{
    GENERATED_BODY()
public:
    UPROPERTY(Transient)
    TObjectPtr<ULocalPlayer> LocalPlayer = nullptr;

    UPROPERTY(Transient)
    TObjectPtr<UPrimaryGameLayout> RootLayout = nullptr;

    UPROPERTY(Transient)
    bool bAddedToViewport = false;

    FRootViewportLayoutInfo() {}
    FRootViewportLayoutInfo(ULocalPlayer* InLocalPlayer, UPrimaryGameLayout* InRootLayout, bool bIsInViewport)
        : LocalPlayer(InLocalPlayer)
        , RootLayout(InRootLayout)
        , bAddedToViewport(bIsInViewport)
    {}
};
```

  - [ ] GetLayoutWidgetClass():

```cpp
TSubclassOf<UPrimaryGameLayout> UGameUIPolicy::GetLayoutWidgetClass(UCommonLocalPlayer* LocalPlayer)
{
    return LayoutClass.LoadSynchronous();
}
```

- [ ] CreateLayoutWidget() 로직 완성하자:

```cpp
void UGameUIPolicy::CreateLayoutWidget(UCommonLocalPlayer* LocalPlayer)
{
    // PlayerController가 할당되었을 경우, LayoutWidget을 생성한다
    if (APlayerController* PlayerController = LocalPlayer->GetPlayerController(GetWorld()))
    {
        // LayoutWidgetClass가 있을 경우 (또! UPrimaryGameLayout은 Abstract이고 이를 상속받는 Class여야 한다)
        TSubclassOf<UPrimaryGameLayout> LayoutWidgetClass = GetLayoutWidgetClass(LocalPlayer);
        if (ensure(LayoutWidgetClass && !LayoutWidgetClass->HasAnyClassFlags(CLASS_Abstract)))
        {
            // PlayerController가 소유한다는 의미에서 Owner를 설정한다
            UPrimaryGameLayout* NewLayoutObject = CreateWidget<UPrimaryGameLayout>(PlayerController, LayoutWidgetClass);

            // FRootViewportLayoutInfo의 Constructor를 정의하자:
            RootViewportLayouts.Emplace(LocalPlayer, NewLayoutObject, true);

            AddLayoutToViewport(LocalPlayer, NewLayoutObject);
        }
    }
}
```

- [ ] FRootViewportLayoutInfo의 == 정의:

```cpp
USTRUCT()
struct FRootViewportLayoutInfo
{
    GENERATED_BODY()
public:
    UPROPERTY(Transient)
    TObjectPtr<ULocalPlayer> LocalPlayer = nullptr;

    UPROPERTY(Transient)
    TObjectPtr<UPrimaryGameLayout> RootLayout = nullptr;

    UPROPERTY(Transient)
    bool bAddedToViewport = false;

    bool operator==(const ULocalPlayer* OtherLocalPlayer) const { return LocalPlayer == OtherLocalPlayer; }

    FRootViewportLayoutInfo() {}
    FRootViewportLayoutInfo(ULocalPlayer* InLocalPlayer, UPrimaryGameLayout* InRootLayout, bool bIsInViewport)
        : LocalPlayer(InLocalPlayer)
        , RootLayout(InRootLayout)
        , bAddedToViewport(bIsInViewport)
    {}
};
```

- [ ] NotifyPlayerRemoved():

```cpp
void UGameUIPolicy::NotifyPlayerRemoved(UCommonLocalPlayer* LocalPlayer)
{
    if (FRootViewportLayoutInfo* LayoutInfo = RootViewportLayouts.FindByKey(LocalPlayer))
    {
        RemoveLayoutFromViewport(LocalPlayer, LayoutInfo->RootLayout);

        // 비활성화 확인할 수 용도로 끈다
        LayoutInfo->bAddedToViewport = false;
    }
}
```

- [ ] NotifyPlayerAdded():

```cpp
void UGameUIPolicy::NotifyPlayerAdded(UCommonLocalPlayer* LocalPlayer)
{
    // PlayerController가 업데이트되면 GameLayout을 업데이트해주기 위해 Delegate를 추가한다
    LocalPlayer->OnPlayerControllerSet.AddWeakLambda(this, [this](UCommonLocalPlayer* LocalPlayer, APlayerController* PlayerController)
    {
        // 우선 추가된 Player가 있으면, 제거하자
        NotifyPlayerRemoved(LocalPlayer);

        // RootViewportLayouts를 순회하며 검색한다:
        // - FRootViewportLayoutInfo의 operator==를 정의해야 한다
        if (FRootViewportLayoutInfo* LayoutInfo = RootViewportLayouts.FindByKey(LocalPlayer))
        {
            // Layout만 업데이트 해준다
            AddLayoutToViewport(LocalPlayer, LayoutInfo->RootLayout);
            LayoutInfo->bAddedToViewport = true;
        }
        else
        {
            // Layout을 생성하고 활성화한다
            CreateLayoutWidget(LocalPlayer);
        }
    });

    if (FRootViewportLayoutInfo* LayoutInfo = RootViewportLayouts.FindByKey(LocalPlayer))
    {
        AddLayoutToViewport(LocalPlayer, LayoutInfo->RootLayout);
        LayoutInfo->bAddedToViewport = true;
    }
    else
    {
        CreateLayoutWidget(LocalPlayer);
    }
}
```

- [ ] UGameUIManagerSubsystem::RemoveLocalPlayer():

```cpp
void UGameUIManagerSubsystem::NotifyPlayerRemoved(UCommonLocalPlayer* LocalPlayer)
{
    if (ensure(LocalPlayer) && CurrentPolicy)
    {
        CurrentPolicy->NotifyPlayerRemoved(LocalPlayer);
    }
}
```

- [ ] UGameUIManagerSubsystem::NotifyPlayerDestroyed():

```cpp
void UGameUIManagerSubsystem::NotifyPlayerDestroyed(UCommonLocalPlayer* LocalPlayer)
{
    if (LocalPlayer && CurrentPolicy)
    {
        CurrentPolicy->NotifyPlayerDestroyed(LocalPlayer);
    }
}
```

- [ ] UGameUIPolicy::NotifyPlayerDestroyed():

```cpp
void UGameUIPolicy::NotifyPlayerDestroyed(UCommonLocalPlayer* LocalPlayer)
{
    NotifyPlayerRemoved(LocalPlayer);

    // Player가 Destroy되므로, OnPlayerControllerSet에서 제거하자
    LocalPlayer->OnPlayerControllerSet.RemoveAll(this);

    // RootViewportLayouts에서 제거하자
    const int32 LayoutInfoIdx = RootViewportLayouts.IndexOfByKey(LocalPlayer);
    if (LayoutInfoIdx != INDEX_NONE)
    {
        // 만약 PrimaryGameLayout가 있으면, Viewport에서도 제거한다
        UPrimaryGameLayout* Layout = RootViewportLayouts[LayoutInfoIdx].RootLayout;
        RootViewportLayouts.RemoveAt(LayoutInfoIdx);
        RemoveLayoutFromViewport(LocalPlayer, Layout);
    }
}
```

- [ ] OnPlayerControllerSet을 Broadcast해야한다:

- PlayerController의 ReceivePlayer()를 통해 가능하다:

```cpp
#include "CommonPlayerController.h"
#include "CommonLocalPlayer.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(CommonPlayerController)

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

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

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

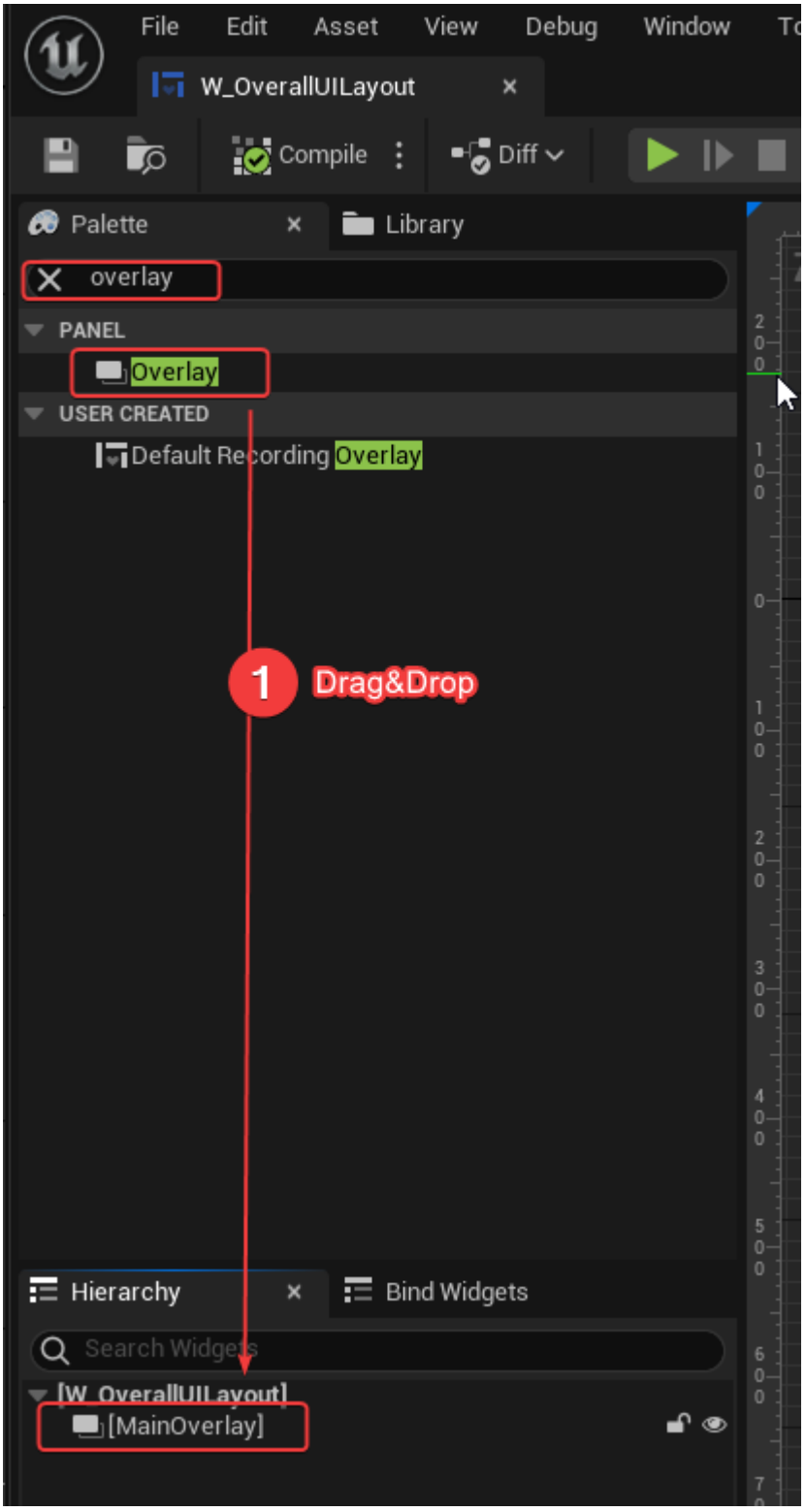
☐ CommonGameInstance::AddLocalPlayer()를 디버깅 하면서 전체적으로 우리가 의도한 대로 작동하는지 확인하자:

```cpp
int32 UCommonGameInstance::AddLocalPlayer(ULocalPlayer* NewPlayer, FPlatformUserId UserId)
{
    // GameInstance에 관리하는 Player 컨테이너에 추가한다 (ReturnVal은 참고로 새로 추가된 Index이다)
    int32 ReturnVal = Super::AddLocalPlayer(NewPlayer, UserId);
    if (ReturnVal != INDEX_NONE)
    {
        // PrimaryPlayer는 처음만 캐싱하는듯하다 (무조건 처음 세팅되면 메인인가 Index==0을 메인으로 잡긴하던데?)
        if (!PrimaryPlayer.IsValid())
        {
            PrimaryPlayer = NewPlayer;
        }

        // GameUIManagerSubsystem를 통해 NotifyPlayerAdded() 호출로 GameLayout을 추가한다
        GetSubsystem<UGameUIManagerSubsystem>()->NotifyPlayerAdded(Cast<UCommonLocalPlayer>(NewPlayer));
    }
    return ReturnVal;
}
```

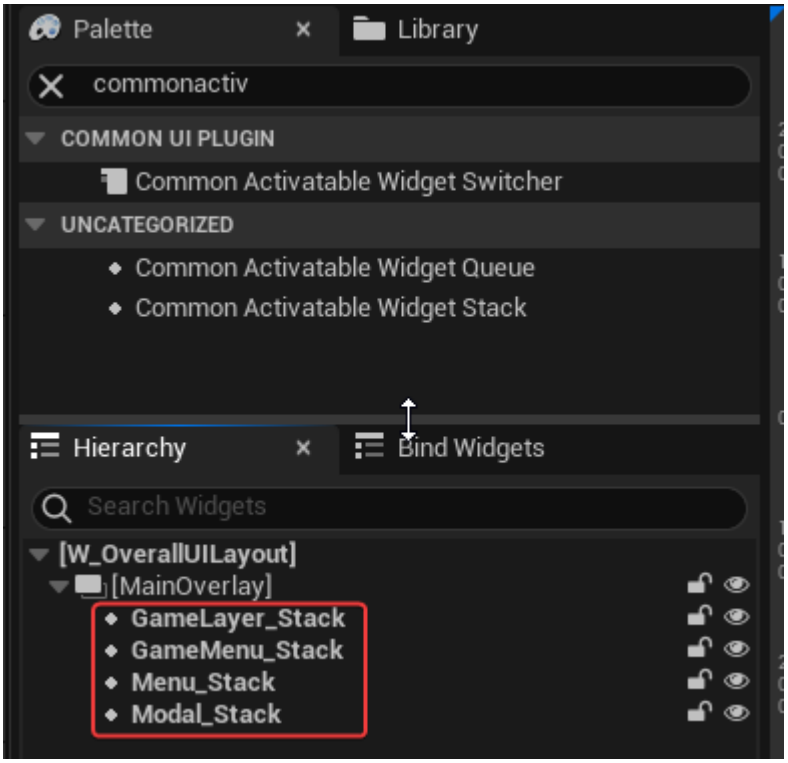
# W_OverallUILayout

▼ **펼치기**

☐ MainOverlay를 추가하자:




☐ MainOverlay 아래 4개의 CommonActivatableWidgetStack을 추가하자:



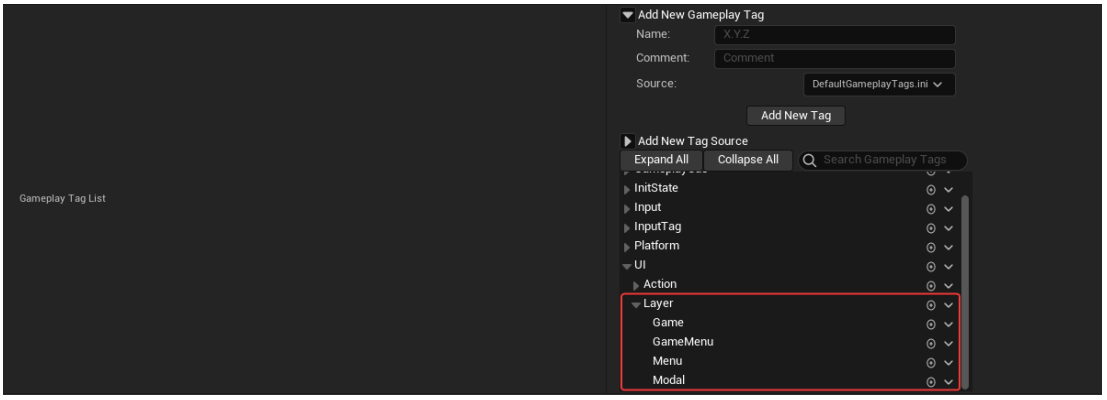

☐ PrimaryGameLayout 멤버 변수 추가 및 RegisterLayer() 선언:

```cpp
#pragma once

#include "CommonUserWidget.h"
#include "GameplayTagContainer.h"
#include "PrimaryGameLayout.generated.h"

/** forward declarations */
class UCommonActivatableWidgetContainerBase;

/**
* 인게임에서 메인 UI의 레이아웃을 담당하는 UMG이다 (Slate vs UMG를 이해하장 - 간단하게 UObject을 기반하는가 아닌가 차이)
* - PrimaryGameLayout은 플레이어당 하나씩 가지는 최상위 UI 레이아웃으로 이해하면 된다
* - PrimaryGameLayout는 레이어 기반으로 UI를 관리한다
*/
UCLASS(Abstract)
class COMMONGAME_API UPrimaryGameLayout : public UCommonUserWidget
{
    GENERATED_BODY()
public:
    /** Layer를 추가하며, GameplayTag를 할당한다 */
    UFUNCTION(BlueprintCallable, Category="Layer")
    void RegisterLayer(FGameplayTag LayerTag, UCommonActivatableWidgetContainerBase* LayerWidget);

    /** 해당 클래스는 Abstract로 설정되었으므로 굳이 FObjectInitializer::Get()할 필요는 없다 */
    UPrimaryGameLayout(const FObjectInitializer& ObjectInitializer);

    /** GameplayTag --- CommonActivatableWidgetContainerBase */
    UPROPERTY(Transient, meta=(Categories="UI.Layer"))
    TMap<FGameplayTag, TObjectPtr<UCommonActivatableWidgetContainerBase>> Layers;
};
```

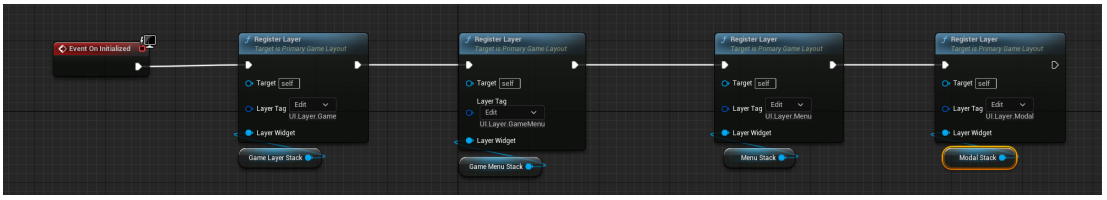
☐ RegisterLayer():

```cpp
#include "PrimaryGameLayout.h"
#include "Widgets/CommonActivatableWidgetContainer.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(PrimaryGameLayout)

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

void UPrimaryGameLayout::RegisterLayer(FGameplayTag LayerTag, UCommonActivatableWidgetContainerBase* LayerWidget)
{
    if (!IsDesignTime())
    {
        LayerWidget->SetTransitionDuration(0.0);
        Layers.Add(LayerTag, LayerWidget);
    }
}
```

☐ W_OverallUILayout의 각 WidgetStack에 연동할 Layer의 GameplayTag를 추가하자:



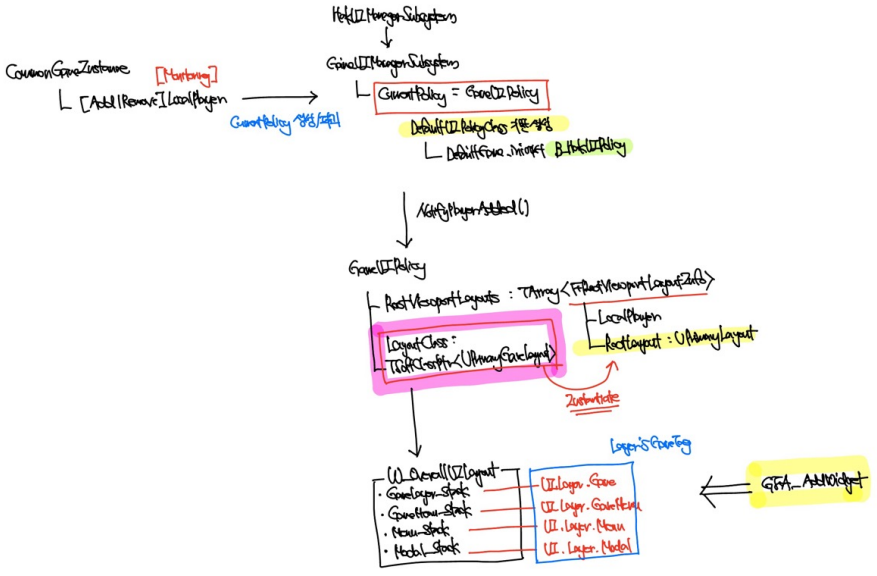

☐ RegisterLayer()를 통해, W_OverallUILayout의 WidgetStack을 Layer로 추가하자:



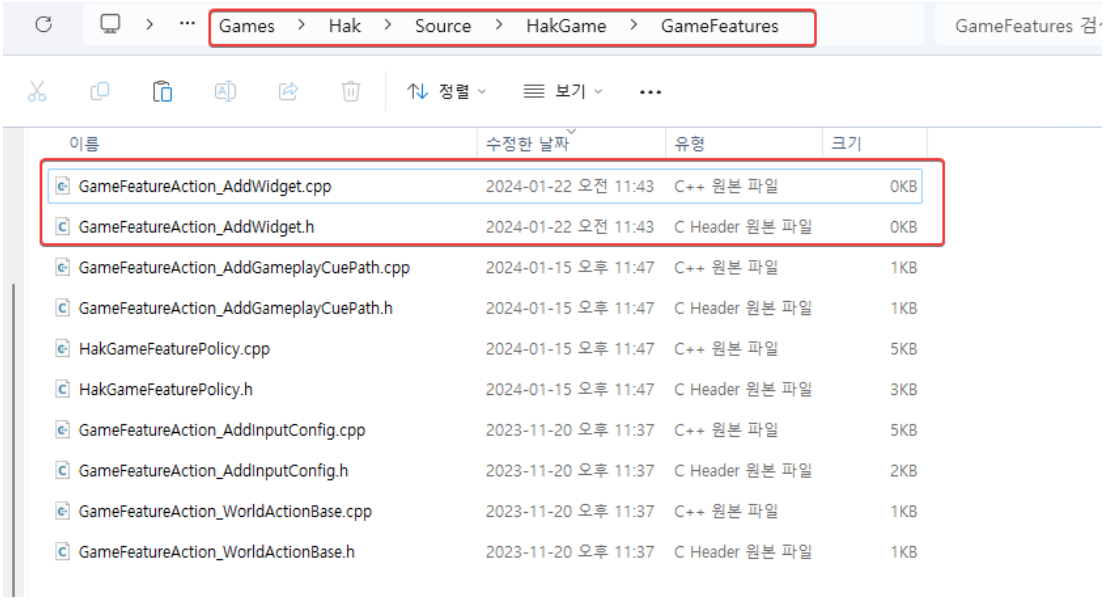

# GFA_AddWidget

▼ **펼치기**

- 지금까지 다룬 내용을 그림으로 정리해보자:



- [ ] GFA_AddWidget.h/.cpp를 추가하자:




- [ ] GFA_AddWidget.h/.cpp 구조 정의:

```cpp
#pragma once

#include "CoreMinimal.h"
#include "GameFeatureAction_WorldActionBase.h"
#include "GameFeatureAction_AddWidget.generated.h"

UCLASS(meta=(DisplayName="Add Widgets"))
class UGameFeatureAction_AddWidgets final : public UGameFeatureAction_WorldActionBase
{
    GENERATED_BODY()
public:

};
```

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

- [ ] GFA_AddWidget 멤버 변수 선언:
  - [ ] Layout과 Widgets:

```cpp
#pragma once

#include "CoreMinimal.h"
#include "CommonActivatableWidget.h"
#include "GameplayTagContainer.h"
#include "GameFeatureAction_WorldActionBase.h"
#include "GameFeatureAction_AddWidget.generated.h"

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

    /** UI의 레이아웃으로 CommonActivatableWidget을 사용 */
    UPROPERTY(EditAnywhere, Category=UI)
    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)
    TSoftClassPtr<UUserWidget> WidgetClass;

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

UCLASS(meta=(DisplayName="Add Widgets"))
class UGameFeatureAction_AddWidgets final : public UGameFeatureAction_WorldActionBase
{
    GENERATED_BODY()
public:
    /**
     * GFA_AddWidget은 형태를 정의하는 Layout과 Layout 위에 올릴 Widget 객체로 Widgets으로 구성된다
     */
    UPROPERTY(EditAnywhere, Category=UI)
    TArray<FHakHUDLayoutRequest> Layout;

    UPROPERTY(EditAnywhere, Category=UI)
    TArray<FHakHUDElementEntry> Widgets;
};
```

☐ GFA 추가/제거의 상태를 가지고 있는 ContextData:

```cpp
UCLASS(meta=(DisplayName="Add Widgets"))
class UGameFeatureAction_AddWidgets final : public UGameFeatureAction_WorldActionBase
{
    GENERATED_BODY()
public:
    struct FPerContextData
    {
        TArray<TSharedPtr<FComponentRequestHandle>> ComponentRequests;
        TArray<TWeakObjectPtr<UCommonActivatableWidget>> LayoutsAdded;

        /** Lyra에서 HUDElement는 UIExtension으로 관리된다. */
        //TArray<FUIExtensionHandle> ExtensionHandles;
    };

    /** GFA Add/Remove 상태 관리 */
    TMap<FGameFeatureStateChangeContext, FPerContextData> ContextData;

    /**
     * GFA_AddWidget은 형태를 정의하는 Layout과 Layout 위에 올릴 Widget 객체로 Widgets으로 구성된다
     */
    UPROPERTY(EditAnywhere, Category=UI)
    TArray<FHakHUDLayoutRequest> Layout;

    UPROPERTY(EditAnywhere, Category=UI)
    TArray<FHakHUDElementEntry> Widgets;
};
```

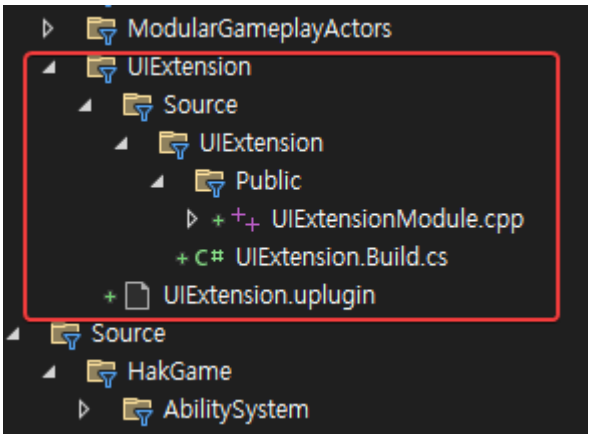
1

☐ 컴파일 Link Error를 해소하기 위해 누락된 모듈을 HakGame.Build.cs에 포함하자:

```cpp
using UnrealBuildTool;

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

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

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

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

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

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

- UIExtension을 관리하는 UIExtensionSystem을 포함하는 UIExtension Plugin이 필요하다

- [ ] UIExtension Plugin을 생성하자:




- [ ] UIExtension.uplugin:

```
{
    "FileVersion": 3,
    "Version": 1,
    "VersionName": "1.0",
    "FriendlyName": "UIExtension",
    "Description": "A subsystem that allows extending UI elements in a modular way.",
    "Category": "UI",
    "CreatedBy": "",
    "CreatedByURL": "",
    "DocsURL": "",
    "MarketplaceURL": "",
    "SupportURL": "",
    "EnabledByDefault": false,
    "CanContainContent": false,
    "IsBetaVersion": false,
    "Installed": false,
    "Modules": [
        {
            "Name": "UIExtension",
            "Type": "Runtime",
            "LoadingPhase": "Default"
        }
    ],
    "Plugins": [
        {
            "Name": "CommonUI",
            "Enabled": true
        },
        {
            "Name": "CommonGame",
            "Enabled": true
        }
    ]
}
```

- [ ] UIExtension.Build.cs

```
using UnrealBuildTool;

1 reference
public class UIExtension : ModuleRules
{
    0 references
    public UIExtension(ReadOnlyTargetRules Target) : base(Target)
    {
        PublicDependencyModuleNames.AddRange(
            new string[]
            {
                "Core",
                "CoreUObject",
                "Engine",
                "SlateCore",
                "Slate",
                "UMG",
                "CommonUI",
                "CommonGame",
                "GameplayTags"
            }
        );

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

        PrivateDependencyModuleNames.AddRange(
            new string[]
            {
                // ... add private dependencies that you statically link with here ...
            }
        );
    }
}
```

- [ ] UIExtensionModule.cpp:

```
#include "Modules/ModuleInterface.h"
#include "Modules/ModuleManager.h"

class FUIExtensionModule : public IModuleInterface
{
public:
    /** IModuleInterface implementation */
    virtual void StartupModule() override;
    virtual void ShutdownModule() override;
};

void FUIExtensionModule::StartupModule()
{
    // This code will execute after your module is loaded into memory; the exact timing is specified in the .uplugin file per-module
}

void FUIExtensionModule::ShutdownModule()
{
    // This function may be called during shutdown to clean up your module.  For modules that support dynamic reloading,
    // we call this function before unloading the module.
}

IMPLEMENT_MODULE(FUIExtensionModule, UIExtension)
```

- [ ] 파일이 이렇게 준비되고 내용을 넣으면 GenerateProjectFiles.bat를 실행하면 올바르게 Plugin이 생성되어 Solution 파일에 추가된다

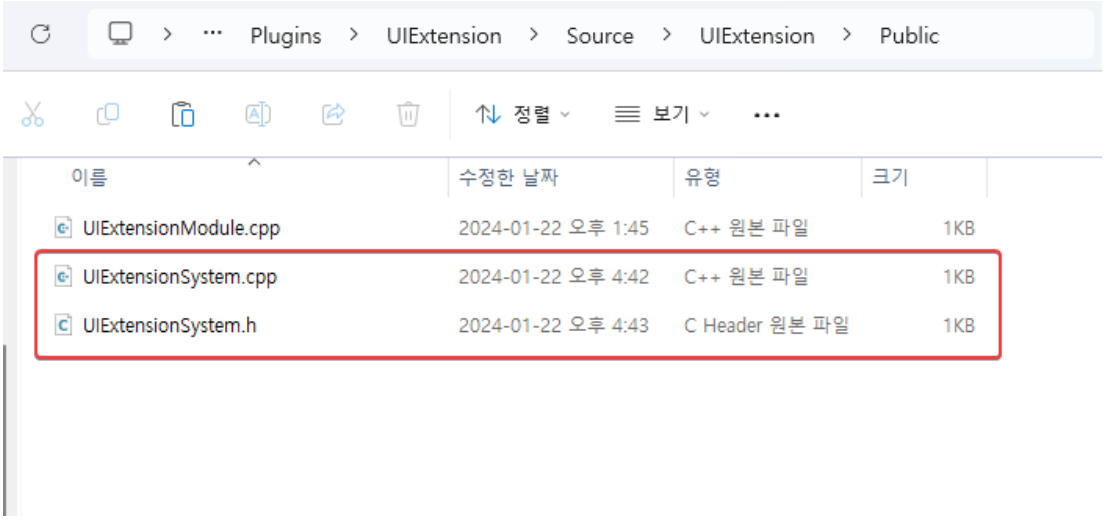
- [ ] UIExtension Plugin과 Module을 HakGame에 추가해주자:
  - [ ] Hak.uproject:


{
"FileVersion": 3,
"EngineAssociation": "",
"Category": "",
"Description": "",
"Modules": [
{
"Name": "HakGame",
"Type": "Runtime",
"LoadingPhase": "Default",
"AdditionalDependencies": [
"Engine"
]
}
],
"Plugins": [
{
"Name": "ModelingToolsEditorMode",
"Enabled": true,
"TargetAllowList": [
"Editor"
]
},
{
"Name": "GameFeatures",
"Enabled": true
},
{
"Name": "ModularGameplay",
"Enabled": true
},
{
"Name": "GameplayAbilities",
"Enabled": true
},
{
"Name": "ShooterCore",
"Enabled": true
},
{
"Name": "AnimationLocomotionLibrary",
"Enabled": true
},
{
"Name": "AnimationWarping",
"Enabled": true
},
{
"Name": "CommonGame",
"Enabled": true
},
{
"Name": "UIExtension",
"Enabled": true
}
]
}

☐ HakGame.Build.cs:

```
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",
        });

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

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

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

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

☐ UIExtensionSystem.h/.cpp 추가:



```cpp
#pragma once

#include "Subsystems/WorldSubsystem.h"
#include "UIExtensionSystem.generated.h"

UCLASS()
class UIEXTENSION_API UUIExtensionSubsystem : public UWorldSubsystem
{
    GENERATED_BODY()
public:

};
```

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

☐ FUIExtensionHandle부터 정의하자:

```cpp
#pragma once

#include "Subsystems/WorldSubsystem.h"
#include "UIExtensionSystem.generated.h"

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

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

UCLASS()
class UIEXTENSION_API UUIExtensionSubsystem : public UWorldSubsystem
{
    GENERATED_BODY()
public:

};
```

☐ FUIExtension:

```cpp
#pragma once

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

#include "UIExtensionSystem.generated.h"

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;
};

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

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

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

template <>
struct TStructOpsTypeTraits<FUIExtensionHandle> : public TStructOpsTypeTraitsBase2<FUIExtensionHandle>
{
    enum
    {
        WithCopy = true,
        WithIdenticalViaEquality = true,
    };
};

UCLASS()
class UIEXTENSION_API UUIExtensionSubsystem : public UWorldSubsystem
{
    GENERATED_BODY()
public:

};
```

☐ GFA_AddWidget에서 주석 처리했던 ExtensionHandles을 해제하자:

```cpp
#pragma once

#include "CoreMinimal.h"
#include "CommonActivatableWidget.h"
#include "GameplayTagContainer.h"
#include "UIExtensionSystem.h"
#include "GameFeatureAction_WorldActionBase.h"
#include "GameFeatureAction_AddWidget.generated.h"

/** forward declarations */
struct FComponentRequestHandle;

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

    /** UI의 레이아웃으로 CommonActivatableWidget을 사용 */
    UPROPERTY(EditAnywhere, Category=UI)
    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)
    TSoftClassPtr<UUserWidget> WidgetClass;

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

UCLASS(meta=(DisplayName="Add Widgets"))
class UGameFeatureAction_AddWidgets final : public UGameFeatureAction_WorldActionBase
{
    GENERATED_BODY()
public:
    struct FPerContextData
    {
        TArray<TSharedPtr<FComponentRequestHandle>> ComponentRequests;
        TArray<TWeakObjectPtr<UCommonActivatableWidget>> LayoutsAdded;

        /** Lyra에서 HUDElement는 UIExtension으로 관리된다. */
        TArray<FUIExtensionHandle> ExtensionHandles;
    };

    /** GFA Add/Remove 상태 관리 */
    TMap<FGameFeatureStateChangeContext, FPerContextData> ContextData;

    /**
     * GFA_AddWidget은 형태를 정의하는 Layout과 Layout 위에 올릴 Widget 객체로 Widgets으로 구성된다
     */
    UPROPERTY(EditAnywhere, Category=UI)
    TArray<FHakHUDLayoutRequest> Layout;

    UPROPERTY(EditAnywhere, Category=UI)
    TArray<FHakHUDElementEntry> Widgets;
};
```

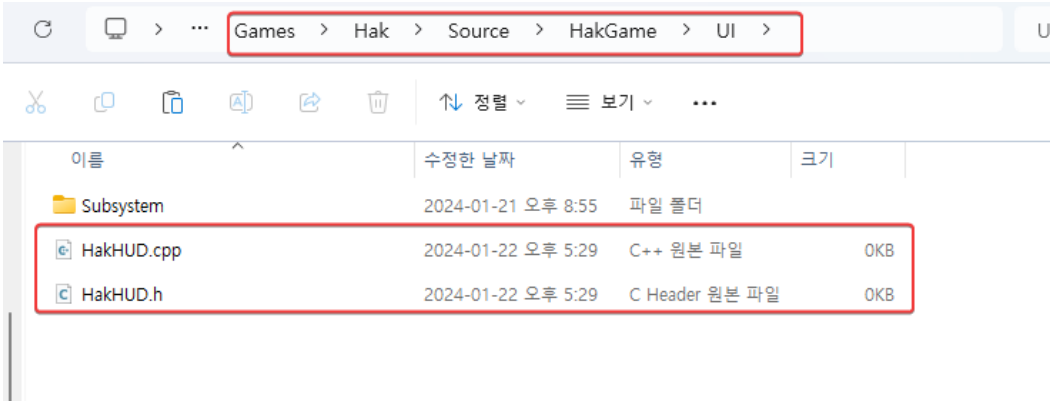
☐ AddToWorld(), OnGameFeatureDeactivating() 인터페이스 구현:

```cpp
UCLASS(meta=(DisplayName="Add Widgets"))
class UGameFeatureAction_AddWidgets final : public UGameFeatureAction_WorldActionBase
{
    GENERATED_BODY()
public:
    /**
    * UGameFeatureAction_WorldActionBase's interface
    */
    virtual void AddToWorld(const FWorldContext& WorldContext, const FGameFeatureStateChangeContext& ChangeContext) override;

    /**
    * UGameFeatureAction's interface
    */
    virtual void OnGameFeatureDeactivating(FGameFeatureDeactivatingContext& Context) override;

    struct FPerContextData
    {
        TArray<TSharedPtr<FComponentRequestHandle>> ComponentRequests;
        TArray<TWeakObjectPtr<UCommonActivatableWidget>> LayoutsAdded;

        /** Lyra에서 HUDElement는 UIExtension으로 관리된다. */
        TArray<FUIExtensionHandle> ExtensionHandles;
    };

    /** GFA Add/Remove 상태 관리 */
    TMap<FGameFeatureStateChangeContext, FPerContextData> ContextData;

    /**
    * GFA_AddWidget은 형태를 정의하는 Layout과 Layout 위에 올릴 Widget 객체로 Widgets으로 구성된다
    */
    UPROPERTY(EditAnywhere, Category=UI)
    TArray<FHakHUDLayoutRequest> Layout;

    UPROPERTY(EditAnywhere, Category=UI)
    TArray<FHakHUDElementEntry> Widgets;
};
```

☐ AddToWorld():

```cpp
void UGameFeatureAction_AddWidgets::AddToWorld(const FWorldContext& WorldContext, const FGameFeatureStateChangeContext& ChangeContext)
{
    UWorld* World = WorldContext.World();
    UGameInstance* GameInstance = WorldContext.OwningGameInstance;
    FPerContextData& ActiveData = ContextData.FindOrAdd(ChangeContext);

    // GameFrameworkComponentManager를 가져올 GameInstance와 World는 GameWorld여야 한다
    if ((GameInstance != nullptr) && (World != nullptr) && World->IsGameWorld())
    {
        // GameFrameworkComponentManager를 가져옴
        if (UGameFrameworkComponentManager* ComponentManager = UGameInstance::GetSubsystem<UGameFrameworkComponentManager>(GameInstance))
        {
            // 기본적으로 Widget을 추가할 대상으로 AHakHUD로 고정한다
            TSoftClassPtr<AActor> HUDActorClass = AHakHUD::StaticClass();

            // GFA_WorldBase와 마찬가지로 HandleActorExtension을 콜백으로 받도록 하자
            TSharedPtr<FComponentRequestHandle> ExtensionRequestHandle = ComponentManager->AddExtensionHandler(
                HUDActorClass,
                UGameFrameworkComponentManager::FExtensionHandlerDelegate::CreateUObject(this, &ThisClass::HandleActorExtension, ChangeContext));

            ActiveData.ComponentRequests.Add(ExtensionRequestHandle);
        }
    }
}
```

☐ HakHUD 생성 및 정의:



```h
#pragma once

#include "GameFramework/HUD.h"
#include "HakHUD.generated.h"

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

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

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

☐ GameFrameworkComponentManager의 Receiver를 대응하기 위한 메서드 추가:

```h
#pragma once

#include "GameFramework/HUD.h"
#include "HakHUD.generated.h"

UCLASS()
class AHakHUD : public AHUD
{
    GENERATED_BODY()
public:
    AHakHUD(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * GameFrameworkComponentManager의 AddReceiver를 위한 메서드들
     */
    virtual void PreInitializeComponents() override;
    virtual void BeginPlay() override;
    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) override;
};
```

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

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

void AHakHUD::PreInitializeComponents()
{
    Super::PreInitializeComponents();

    // HakHUD를 Receiver로 Actor를 추가하자
    UGameFrameworkComponentManager::AddGameFrameworkComponentReceiver(this);
}

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

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

- [ ] HakHUD를 GFA_AddWidget을 추가하자:

```cpp
#include "GameFeatureAction_AddWidget.h"
#include "HakGame/UI/HakHUD.h"
#include "Components/GameFrameworkComponentManager.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(GameFeatureAction_AddWidget)

void UGameFeatureAction_AddWidgets::AddToWorld(const FWorldContext& WorldContext, const FGameFeatureStateChangeContext& ChangeContext)
{
    UWorld* World = WorldContext.World();
    UGameInstance* GameInstance = WorldContext.OwningGameInstance;
    FPerContextData& ActiveData = ContextData.FindOrAdd(ChangeContext);

    // GameFrameworkComponentManager를 가져올 GameInstance와 World는 GameWorld여야 한다
    if ((GameInstance != nullptr) && (World != nullptr) && World->IsGameWorld())
    {
        // GameFrameworkComponentManager를 가져옴
        if (UGameFrameworkComponentManager* ComponentManager = UGameInstance::GetSubsystem<UGameFrameworkComponentManager>(GameInstance))
        {
            // 기본적으로 Widget을 추가할 대상으로 AHakHUD로 고정한다
            TSoftClassPtr<AActor> HUDActorClass = AHakHUD::StaticClass();

            // GFA_WorldBase와 마찬가지로 HandleActorExtension을 콜백으로 받도록 하자
            TSharedPtr<FComponentRequestHandle> ExtensionRequestHandle = ComponentManager->AddExtensionHandler(
                HUDActorClass,
                UGameFrameworkComponentManager::FExtensionHandlerDelegate::CreateUObject(this, &ThisClass::HandleActorExtension, ChangeContext));

            ActiveData.ComponentRequests.Add(ExtensionRequestHandle);
        }
    }
}
```

- [ ] HandleActorExtension():

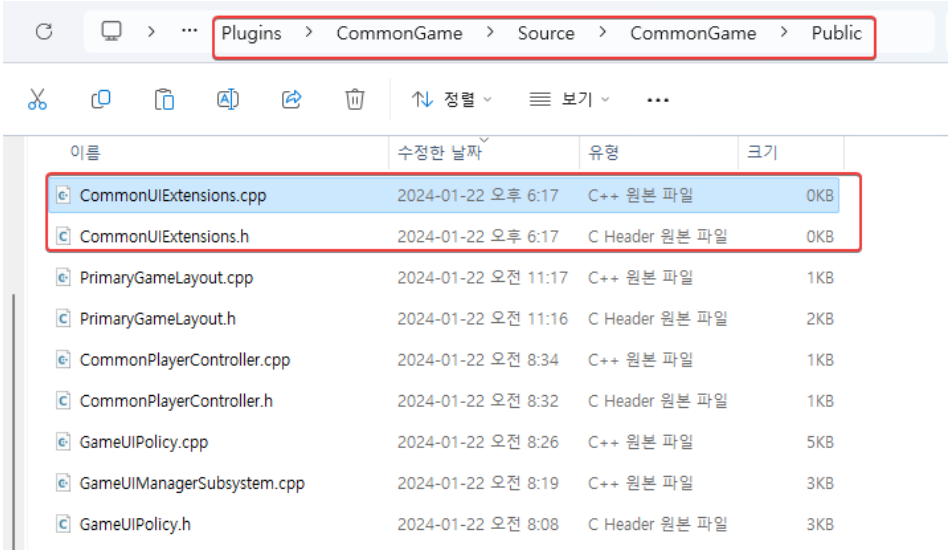
```cpp
void UGameFeatureAction_AddWidgets::HandleActorExtension(AActor* Actor, FName EventName, FGameFeatureStateChangeContext ChangeContext)
{
    FPerContextData& ActiveData = ContextData.FindOrAdd(ChangeContext);

    // 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);
    }
}
```

- [ ] AddWidgets()

```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>();
        for (const FHakHUDElementEntry& Entry : Widgets)
        {
            ActiveData.ExtensionHandles.Add(ExtensionSubsystem->RegisterExtensionAsWidgetForContext(Entry.SlotID, LocalPlayer, Entry.WidgetClass.Get(), -1));
        }
    }
}
```

1
2

- [ ] UCommonUIExtensions
  - [ ] CommonGame에 UCommonUIExtensions을 추가하자:




```h
#pragma once

#include "GameplayTagContainer.h"
#include "Kismet/BlueprintFunctionLibrary.h"

#include "CommonUIExtensions.generated.h"

/** forward declarations */
class UCommonActivatableWidget;
class ULocalPlayer;

UCLASS()
class COMMONGAME_API UCommonUIExtensions : public UBlueprintFunctionLibrary
{
    GENERATED_BODY()
public:
    UCommonUIExtensions() {}

    static UCommonActivatableWidget* PushContentToLayer_ForPlayer(const ULocalPlayer* LocalPlayer, FGameplayTag LayerName, TSubclassOf<UCommonActivatableWidget> WidgetClass);
};
```

```cpp
#include "CommonUIExtensions.h"
#include "CommonActivatableWidget.h"
#include "GameUIManagerSubsystem.h"
#include "GameUIPolicy.h"
#include "PrimaryGameLayout.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(CommonUIExtensions)

UCommonActivatableWidget* UCommonUIExtensions::PushContentToLayer_ForPlayer(const ULocalPlayer* LocalPlayer, FGameplayTag LayerName, TSubclassOf<UCommonActivatableWidget> WidgetClass)
{
    // LocalPlayer를 통해, GameUIManagerSubsystem을 가져옴
    if (UGameUIManagerSubsystem* UIManager = LocalPlayer->GetGameInstance()->GetSubsystem<UGameUIManagerSubsystem>())
    {
        // UIManager에서 현재 활성화된 UIPolicy를 가져오고
        if (UGameUIPolicy* Policy = UIManager->GetCurrentPolicy())
        {
            // Policy에서 LocalPlayer에 맞는 PrimaryGameLayout을 가져온다
            if (UPrimaryGameLayout* RootLayout = Policy->GetRootLayout(CastChecked<UCommonLocalPlayer>(LocalPlayer)))
            {
                // 그리고 PrimaryGameLayout, W_OverallUILayout의 LayerName에 Stack으로 WidgetClass(UCommonActivatableWidget)를 넣어준다
                return RootLayout->PushWidgetToLayerStack(LayerName, WidgetClass);
            }
        }
    }
    return nullptr;
}
```

1
2
3

UIManager→GetCurrentUIPolicy() 오타가 있으니 유의하자!

- [ ] GetCurrentPolicy():

```cpp
/**
 * GameUIManagerSubsystem은 GameInstanceSubsystem의 기반한다
 * - 여기서 주목해야 할 점은 UGameUIManagerSubsystem은 UCLASS 속성으로 Abstract라고 정의되어 있다
 * - 해당 클래스는 단독으로 사용 불가하며, 누군가 상속받은 Concrete Class로서 활용되야 한다
 */
UCLASS(Abstract, config=Game)
class COMMONGAME_API UGameUIManagerSubsystem : public UGameInstanceSubsystem
{
    GENERATED_BODY()
public:
    UGameUIManagerSubsystem();

    void SwitchToPolicy(UGameUIPolicy* InPolicy);

    /**
     * UGameInstanceSubsystem's interfaces
     */
    virtual void Initialize(FSubsystemCollectionBase& Collection) override;
    virtual void Deinitialize() override;
    virtual bool ShouldCreateSubsystem(UObject* Outer) const override;

    /**
     * UGameUIManagerSubsystem's interfaces
     */
    virtual void NotifyPlayerAdded(UCommonLocalPlayer* LocalPlayer);
    virtual void NotifyPlayerRemoved(UCommonLocalPlayer* LocalPlayer);
    virtual void NotifyPlayerDestroyed(UCommonLocalPlayer* LocalPlayer);

    const UGameUIPolicy* GetCurrentUIPolicy() const { return CurrentPolicy; }
    UGameUIPolicy* GetCurrentUIPolicy() { return CurrentPolicy; }

    UPROPERTY(Transient)
    TObjectPtr<UGameUIPolicy> CurrentPolicy = nullptr;

    /**
     * default UI policy를 생성할 class
     * - 우리는 해당 클래스는 B_BttGameUIPolicy로 지정할 것이다
     */
    UPROPERTY(Config, EditAnywhere)
    TSoftClassPtr<UGameUIPolicy> DefaultUIPolicyClass;
};
```

☐ UGameUIPolicy::GetRootLayout()

```cpp
/**
 * UGameUIPolicy가 Abstract라는 것을 기억하자
 */
UCLASS(Abstract, Blueprintable)
class COMMONGAME_API UGameUIPolicy : public UObject
{
    GENERATED_BODY()
public:
    UPrimaryGameLayout* GetRootLayout(const UCommonLocalPlayer* LocalPlayer) const;

    TSubclassOf<UPrimaryGameLayout> GetLayoutWidgetClass(UCommonLocalPlayer* LocalPlayer);
    void CreateLayoutWidget(UCommonLocalPlayer* LocalPlayer);

    void AddLayoutToViewport(UCommonLocalPlayer* LocalPlayer, UPrimaryGameLayout* Layout);
    void RemoveLayoutFromViewport(UCommonLocalPlayer* LocalPlayer, UPrimaryGameLayout* Layout);

    void NotifyPlayerAdded(UCommonLocalPlayer* LocalPlayer);
    void NotifyPlayerRemoved(UCommonLocalPlayer* LocalPlayer);
    void NotifyPlayerDestroyed(UCommonLocalPlayer* LocalPlayer);

    /** LocalPlayer에 바인딩된 UI의 Layout으로 생각하면 된다 (아직 의미가 모호할 수 있는데, 하나씩 구현해보면서, 어떤 느낌인가 더 와닫을 것이다) */
    UPROPERTY(EditAnywhere)
    TSoftClassPtr<UPrimaryGameLayout> LayoutClass;

    /** 보통 싱글 게임에서는 LocalPlayer-PrimaryGameLayout 하나만 있겠지만, 멀티플레이의 경우, 복수개 가능하다 (리플레이를 생각해보자) */
    UPROPERTY(Transient)
    TArray<FRootViewportLayoutInfo> RootViewportLayouts;
};
```

```cpp
UPrimaryGameLayout* UGameUIPolicy::GetRootLayout(const UCommonLocalPlayer* LocalPlayer) const
{
    const FRootViewportLayoutInfo* LayoutInfo = RootViewportLayouts.FindByKey(LocalPlayer);
    return LayoutInfo ? LayoutInfo->RootLayout : nullptr;
}
```

☐ PushWidgetToLayerStack():

```cpp
/**
 * 인게임에서 메인 UI의 레이아웃을 담당하는 UMG이다 (Slate vs UMG를 이해하장 - 간단하게 UObject을 기반하는가 아닌가 차이)
 * - PrimaryGameLayout은 플레이어당 하나씩 가지는 최상위 UI 레이아웃으로 이해하면 된다
 * - PrimaryGameLayout는 레이어 기반으로 UI를 관리한다
 */
UCLASS(Abstract)
class COMMONGAME_API UPrimaryGameLayout : public UCommonUserWidget
{
    GENERATED_BODY()
public:
    /** LayerName에 따른 ActivatableWidgetContainerBase를 가져옴 */
    UCommonActivatableWidgetContainerBase* GetLayerWidget(FGameplayTag LayerName);

    template <typename ActivatableWidgetT = UCommonActivatableWidget>
    ActivatableWidgetT* PushWidgetToLayerStack(FGameplayTag LayerName, UClass* ActivatableWidgetClass)
    {
        return PushWidgetToLayerStack<ActivatableWidgetT>(LayerName, ActivatableWidgetClass, [](ActivatableWidgetT&) {});
    }

    template <typename ActivatableWidgetT = UCommonActivatableWidget>
    ActivatableWidgetT* PushWidgetToLayerStack(FGameplayTag LayerName, UClass* ActivatableWidgetClass, TFunctionRef<void(ActivatableWidgetT&)> InitInstanceFunc)
    {
        static_assert(TIsDerivedFrom<ActivatableWidgetT, UCommonActivatableWidget>::IsDerived, "only CommonActivatableWidgets can be used here");

        if (UCommonActivatableWidgetContainerBase* Layer = GetLayerWidget(LayerName))
        {
            return Layer->AddWidget<ActivatableWidgetT>(ActivatableWidgetClass, InitInstanceFunc);
        }

        return nullptr;
    }

    /** Layer를 추가하며, GameplayTag를 할당한다 */
    UFUNCTION(BlueprintCallable, Category="Layer")
    void RegisterLayer(FGameplayTag LayerTag, UCommonActivatableWidgetContainerBase* LayerWidget);

    /** 해당 클래스는 Abstract로 설정되었으므로 굳이 FObjectInitializer::Get()할 필요는 없다 */
    UPrimaryGameLayout(const FObjectInitializer& ObjectInitializer);

    /** GameplayTag --- CommonActivatableWidgetContainerBase */
    UPROPERTY(Transient, meta=(Categories="UI.Layer"))
    TMap<FGameplayTag, TObjectPtr<UCommonActivatableWidgetContainerBase>> Layers;
};
```

```cpp
UCommonActivatableWidgetContainerBase* UPrimaryGameLayout::GetLayerWidget(FGameplayTag LayerName)
{
    return Layers.FindRef(LayerName);
}
```

☐ AddWidget()에 CommonUIExtensions.h를 추가하자:

```cpp
#include "GameFeatureAction_AddWidget.h"
#include "HakGame/UI/HakHUD.h"
#include "CommonUIExtensions.h"
#include "Components/GameFrameworkComponentManager.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(GameFeatureAction_AddWidget)

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>();
        for (const FHakHUDElementEntry& Entry : Widgets)
        {
            ActiveData.ExtensionHandles.Add(ExtensionSubsystem->RegisterExtensionAsWidgetForContext(Entry.SlotID, LocalPlayer, Entry.WidgetClass.Get(), -1));
        }
    }
}
```

☐ UUIExtensionSubsystem의 멤버 변수 정의:

```cpp
UCLASS()
class UIEXTENSION_API UUIExtensionSubsystem : public UWorldSubsystem
{
    GENERATED_BODY()
public:
    /** GameplayTag(Slot) --- FUIExtension(WidgetClass) */
    typedef TArray<TSharedPtr<FUIExtension>> FExtensionList;
    TMap<FGameplayTag, FExtensionList> ExtensionMap;
};
```

☐ RegisterExtensionAsWidgetForContext()

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

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

```cpp
#include "UIExtensionSystem.h"
#include "Blueprint/UserWidget.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(UIExtensionSystem)

FUIExtensionHandle UUIExtensionSubsystem::RegisterExtensionAsWidgetForContext(const FGameplayTag& ExtensionPointTag, UObject* ContextObject, TSubclassOf<UUserWidget> WidgetClass, int32 Priority)
{
    return RegisterExtensionAsData(ExtensionPointTag, ContextObject, WidgetClass, Priority);
}

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;

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

- [ ] 여기까지 AddWidget 완성!

- [ ] RemoveWidgets():

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

    // HakHUD에 추가된 CommonActivatableWidget을 순회하며, Deativate 시켜준다
    for (TWeakObjectPtr<UCommonActivatableWidget>& AddedLayout : ActiveData.LayoutsAdded)
    {
        if (AddedLayout.IsValid())
        {
            AddedLayout->DeactivateWidget();
        }
    }
    ActiveData.LayoutsAdded.Reset();

    // UIExtension에 대해 순회하며, Unregister() 한다
    for (FUIExtensionHandle& Handle : ActiveData.ExtensionHandles)
    {
        // Unregister()는 UUIExtensionSystem에서 제거가 올바르게 되야 한다
        Handle.Unregister();
    }
    ActiveData.ExtensionHandles.Reset();
}
```

- [ ] FUIExtensionHandle::Unregister():

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

- [ ] UUIExtensionSubsystem::UnregisterExtension():

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

- [ ] FUIExtensionHandle::IsValid():

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

- [ ] OnGameFeatureDeactivating():

```cpp
void UGameFeatureAction_AddWidgets::OnGameFeatureDeactivating(FGameFeatureDeactivatingContext& Context)
{
    Super::OnGameFeatureDeactivating(Context);

    FPerContextData* ActiveData = ContextData.Find(Context);
    if (ensure(ActiveData))
    {
        Reset(*ActiveData);
    }
}
```

- [ ] Reset():

```cpp
void UGameFeatureAction_AddWidgets::Reset(FPerContextData& ActiveData)
{
    ActiveData.ComponentRequests.Empty();
    ActiveData.LayoutsAdded.Empty();

    for (FUIExtensionHandle& Handle : ActiveData.ExtensionHandles)
    {
        Handle.Unregister();
    }
    ActiveData.ExtensionHandles.Reset();
}
```