13주차 (2024.01.15)

HakAnimInstance::InitializeWithAbilitySystem()

▼ **펼치기**

저번 시간의 문제는 **AnimInstance의 GameplayTagPropertyMap이 제대로 업데이트 되지 않아, 모션의 상태가 제대로 반영되지 않는 문제**였다:

- 해당 문제는 Controller가 Possess 이후, InitAbilityActorInfo가 호출로 AbilityActorInfo가 업데이트 되었고, 이 업데이트 된 정보가 AnimInstance까지 전달되지 못했다
- 이 해결을 위해 명시적으로 Possess 이후, Pawn 변화가 생겼을 경우, HakAnimInstance::InitializeWithAbilitySystem() 호출을 진행해야 한다:

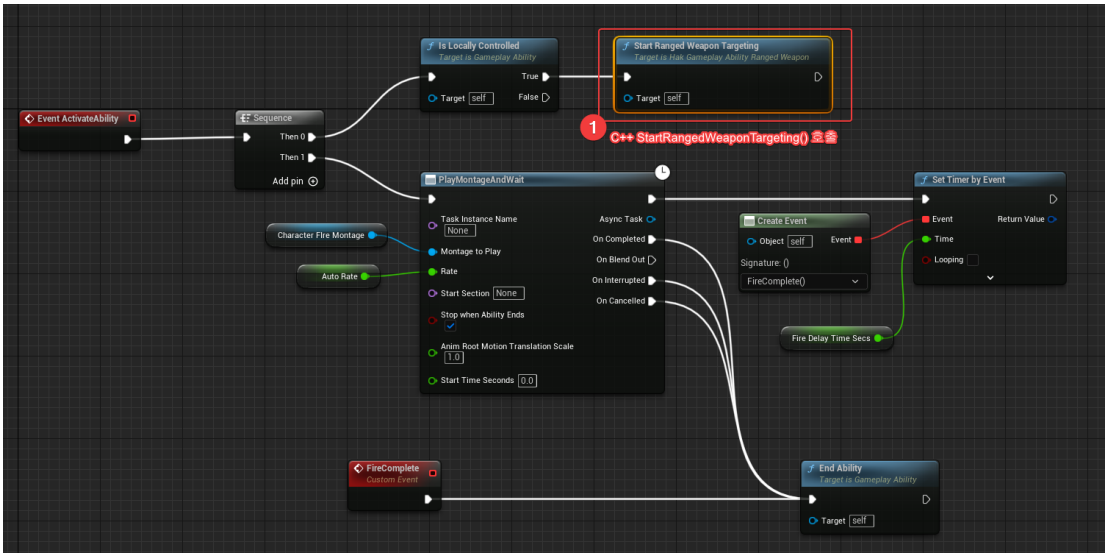
```cpp
void UHakAbilitySystemComponent::InitAbilityActorInfo(AActor* InOwnerActor, AActor* InAvatarActor)
{
    // UHakHeroComponent::HandleChangeInitState 호출을 통해 HakAbilitySystemComponent의 InitializeAbilitySystem()이 호출된다:
    // - 해당 호출에서 InitAbilityActorInfo가 호출된다 -> 이는 Controller가 Possess를 했음을 감지하고 호출하는 과정이다
    FGameplayAbilityActorInfo* ActorInfo = AbilityActorInfo.Get();
    check(ActorInfo);
    check(InOwnerActor);

    const bool bHasNewPawnAvatar = Cast<APawn>(InAvatarActor) && (InAvatarActor != ActorInfo->AvatarActor);

    Super::InitAbilityActorInfo(InOwnerActor, InAvatarActor);

    if (bHasNewPawnAvatar)
    {
        // Controller가 Possess하고 있는 Pawn의 AnimInstance를 명시적으로 재호출하여 초기화한다
        // - 명시적 호출을 진행하지 않으면, Possess 이후, Pawn 변화 이후에 업데이트를 위한 호출이 되지 않는다...
        if (UHakAnimInstance* HakAnimInst = Cast<UHakAnimInstance>(ActorInfo->GetAnimInstance()))
        {
            HakAnimInst->InitializeWithAbilitySystem(this);
        }
    }
}
```

GCN_Weapon_Pistol_Fire

▼ **펼치기**

저번 시간을 회고해보자:

- 우리는 GA_Weapon_Fire에서 StartRangedWeaponTargeting() 호출을 통해, 총탄의 궤적에 따른 충돌 정보(Impact)를 계산하였다:



- 계산한 정보를 TargetData에 넣고, OnTargetDataReadyCallback() 호출:

```cpp
void UHakGameplayAbility_RangedWeapon::StartRangedWeaponTargeting()
{
    // ActorInfo는 AbilitySet에서 GiveAbility() 호출로 설정된다
    // - UGameplayAbility::OnGiveAbility()에서 SetCurrentActorInfo()에서 설정된다
    // - AbilitySystemComponent::GiveAbility()에서 OnGiveAbility() 호출한다
    // - HakAbilitySet::GiveToAbilitySystem()에서 GiveAbility()를 호출한다
    check(CurrentActorInfo);

    AActor* AvatarActor = CurrentActorInfo->AvatarActor.Get();
    check(AvatarActor);

    UAbilitySystemComponent* MyAbilityComponent = CurrentActorInfo->AbilitySystemComponent.Get();
    check(MyAbilityComponent);

    //*** 여기서 Lyra는 샷건 처리와 같은 탄착 처리를 생략하고, 권총으로 진행하였다 (아래의 로직은 간단버전이다)

    // 총알의 궤적의 Hit 정보를 계산
    TArray<FHitResult> FoundHits;
    PerformLocalTargeting(FoundHits);

    // GameplayAbilityTargetData는 Server/Client 간 Ability의 공유 데이터로 이해하면 된다:
    // - 허나, 우리는 싱글플레이이므로 Ability의 데이터로 생각하면 되겠다 (현재 큰 의미가 없다고 볼 수 있다)
    FGameplayAbilityTargetDataHandle TargetData;
    TargetData.UniqueId = 0;

    if (FoundHits.Num() > 0)
    {
        // Cartridge란 일반 권총의 경우, 탄약에 하나의 총알이 들어있지만, 샷건의 경우, 탄약에 여러개의 총알이 있고, **탄약을 카트리지로 생각**하면 될 것 같다
        const int32 CartridgeID = FMath::Rand();
        for (const FHitResult& FoundHit : FoundHits)
        {
            // AbilityTargetData에 SingeTargetHit 정보를 담는다
            // - 참고로 TargetData.Add()의 경우, SharedPtr에 넣기 때문에 여기서 new는 크게 신경 안써도 된다
            FHakGameplayAbilityTargetData_SingleTargetHit* NewTargetData = new FHakGameplayAbilityTargetData_SingleTargetHit();
            NewTargetData->HitResult = FoundHit;
            NewTargetData->CartridgeID = CartridgeID;
            TargetData.Add(NewTargetData);
        }
    }

    // 가공된 AbilityTargetData가 준비되었으므로, OnTargetDataReadyCallback을 호출한다
    OnTargetDataReadyCallback(TargetData, FGameplayTag());
}
```

1 총탄 궤적에 따른 충돌 정보를 TargetData에 저장하였다

2

- 우리는 격발에 대한 이펙트 효과를 표현하기 위해 GCN(GameplayCueNotify)를 호출해야 한다. 이를 OnRangeWeaponTargetDataReady()에 구현해야 한다:

```cpp
void UHakGameplayAbility_RangedWeapon::OnTargetDataReadyCallback(const FGameplayAbilityTargetDataHandle& InData, FGameplayTag ApplicationTag)
{
    UAbilitySystemComponent* MyAbilitySystemComponent = CurrentActorInfo->AbilitySystemComponent.Get();
    check(MyAbilitySystemComponent);

    if (const FGameplayAbilitySpec* AbilitySpec = MyAbilitySystemComponent->FindAbilitySpecFromHandle(CurrentSpecHandle))
    {
        // 현재 Stack에서 InData에서 지금 Local로 Ownership을 가져온다
        FGameplayAbilityTargetDataHandle LocalTargetDataHandle(MoveTemp(const_cast<FGameplayAbilityTargetDataHandle&>(InData)));

        // CommitAbility 호출로 GE(GameplayEffect)를 처리한다
        // - 현재 아직 우리는 GE에 대해 처리를 진행하지 않을 것이다
        if (CommitAbility(CurrentSpecHandle, CurrentActorInfo, CurrentActivationInfo))
        {
            // OnRangeWeaponTargetDataReady BP 노드 호출한다:
            // - 후일 여기서 우리는 GCN(GameplayCueNotify)를 처리할 것이다
            OnRangeWeaponTargetDataReady(LocalTargetDataHandle);
        }
        else
        {
            // CommitAbility가 실패하였으면, EndAbility BP Node 호출한다
            K2_EndAbility();
        }
    }
}
```

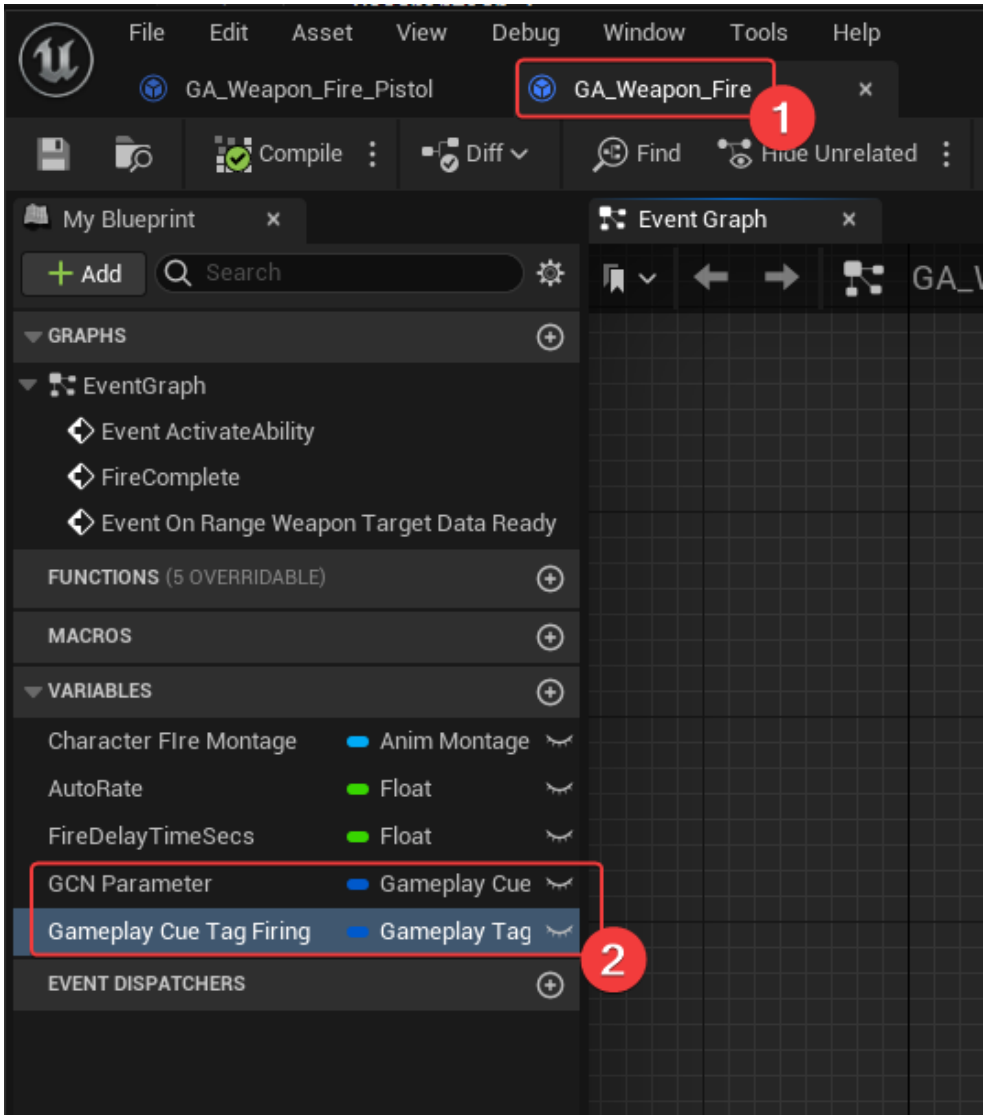
1

- OnRangeWeaponTargetDataReady()는 Blueprint에서 구현해야 할 (정확히는 구현 가능한!) Event 함수이다:

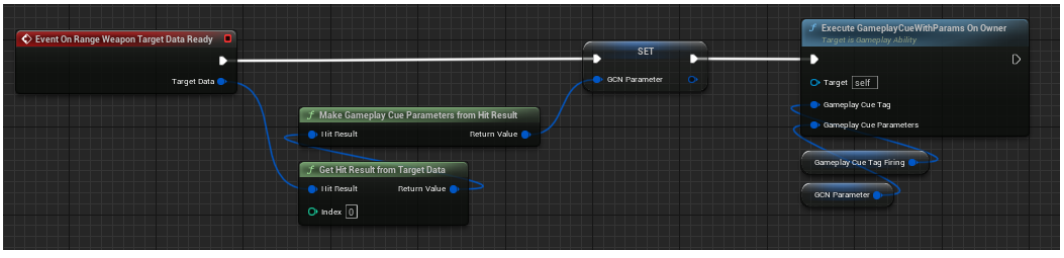
```cpp
UCLASS()
class UHakGameplayAbility_RangedWeapon : public UHakGameplayAbility_FromEquipment
{
    GENERATED_BODY()
public:
    struct FRangedWeaponFiringInput
    {
        FVector StartTrace;
        FVector EndAim;
        FVector AimDir;
        UHakRangedWeaponInstance* WeaponData = nullptr;
        bool bCanPlayBulletFX = false;

        FRangedWeaponFiringInput()
            : StartTrace(ForceInitToZero)
            , EndAim(ForceInitToZero)
            , AimDir(ForceInitToZero)
        {}
    };

    UHakGameplayAbility_RangedWeapon(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    void AddAdditionalTraceIgnoreActors(FCollisionQueryParams& TraceParams) const;
    ECollisionChannel DetermineTraceChannel(FCollisionQueryParams& TraceParams, bool bIsSimulated) const;
    FHitResult WeaponTrace(const FVector& StartTrace, const FVector& EndTrace, float SweepRadius, bool bIsSimulated, TArray<FHitResult>& OutHitResults)
    FHitResult DoSingleBulletTrace(const FVector& StartTrace, const FVector& EndTrace, float SweepRadius, bool bIsSimulated, TArray<FHitResult>& OutHit
    void TraceBulletsInCartridge(const FRangedWeaponFiringInput& InputData, TArray<FHitResult>& OutHits);

    UHakRangedWeaponInstance* GetWeaponInstance();
    FVector GetWeaponTargetingSourceLocation() const;
    FTransform GetTargetingTransform(APawn* SourcePawn, EHakAbilityTargetingSource Source);
    void PerformLocalTargeting(TArray<FHitResult>& OutHits);
    void OnTargetDataReadyCallback(const FGameplayAbilityTargetDataHandle& InData, FGameplayTag ApplicationTag);

    UFUNCTION(BlueprintCallable)
    void StartRangedWeaponTargeting();

    /** called when target data is ready */
    UFUNCTION(BlueprintImplementableEvent)
    void OnRangeWeaponTargetDataReady(const FGameplayAbilityTargetDataHandle& TargetData);
};
```

1

- [ ] GA_Weapon_Fire 변수 추가:



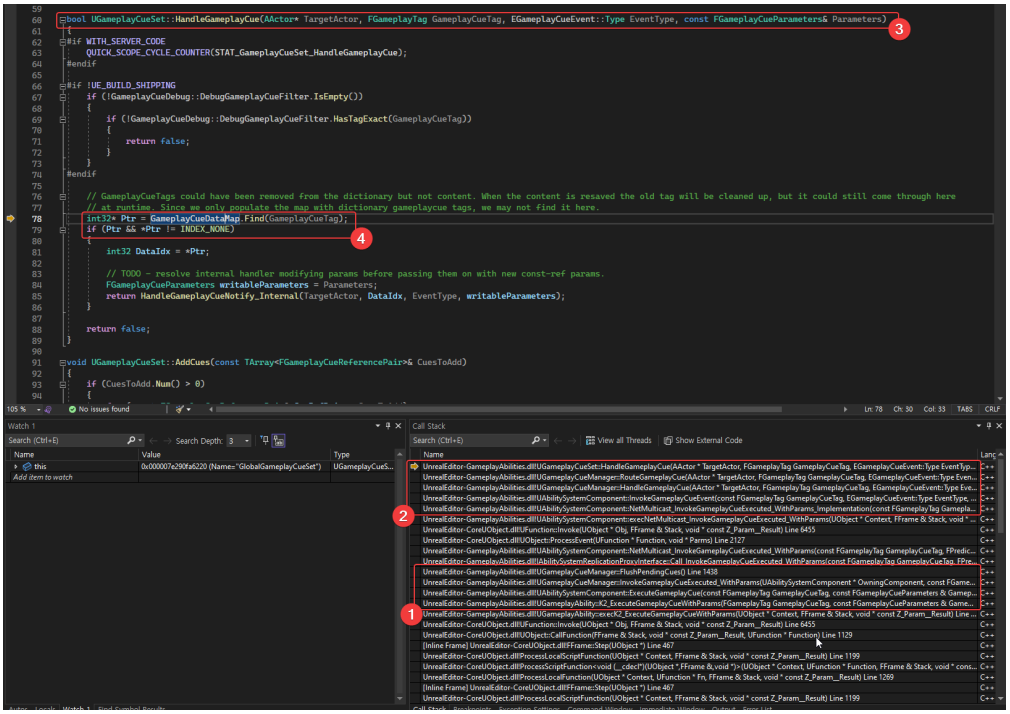
- GCN Parameter (GameplayCueParameters)
- GameplayCueTagFiring (GameplayTag)
- 아래와 같이 OnRangeWeaponTargetDataReady() 이벤트 로직을 완성하자:



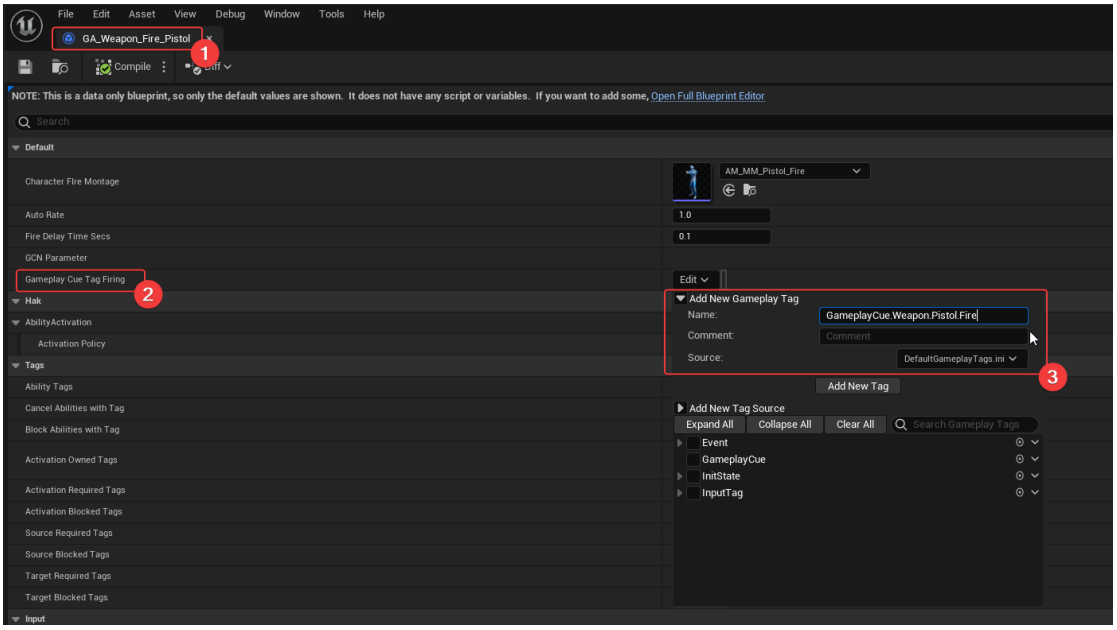
  - 간단히 GetHitResultFromTargetData(), MakeGameplayCueParametersFromHitResult(), ExecuteGameplayCueWithParamsOnOwner()의 C++ 구현을 보자:
    - GCN Parameters에 무슨 내용이 들어가는지?

- ExecuteGameplayCueWithParamsOnOwner() 어떻게 작동하는지?
  - GameplayCueManager에 FGameplayCuePendingExecute로 PendingExecuteCues에 저장
  - FlushPendingCues() 호출할 때, AbilitySystemComponent를 통한 NetMulticast_InvokeGameplayCueExecuted_WithParams() 호출
  - AbilitySystemComponent의 InvokeGameplayCueEvent()와 RouteGameplayCue() 그리고 HandleGameplayCue()까지 호출
  - GameplayCueDataMap에 **GameplayTag와 연동된 GameplayCue를 찾아서 호출:**

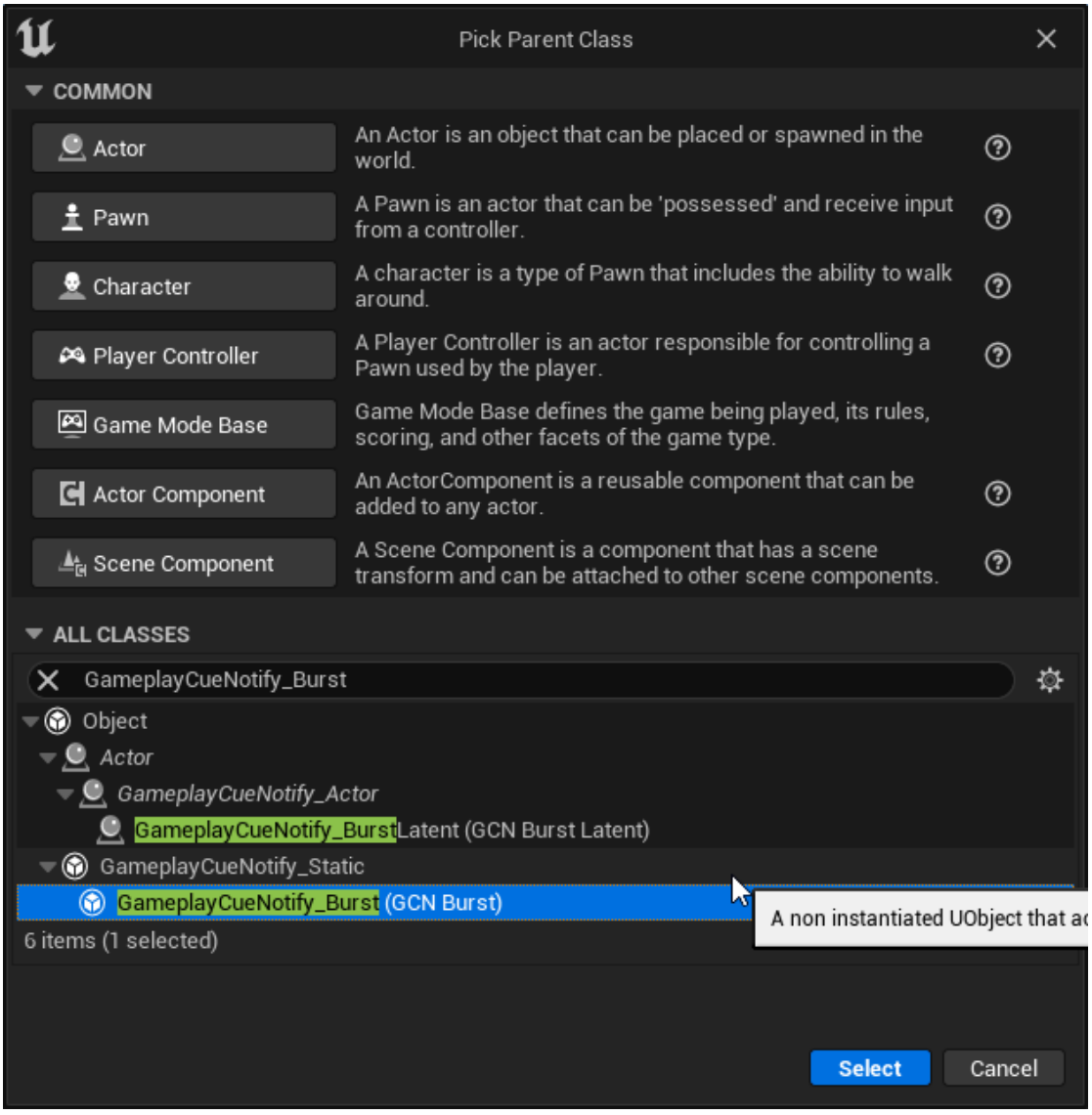
◦ 여기서 알아둬야 할 점은 `GameplayCue 실행은 GameplayTag 연동으로 일어난다` :

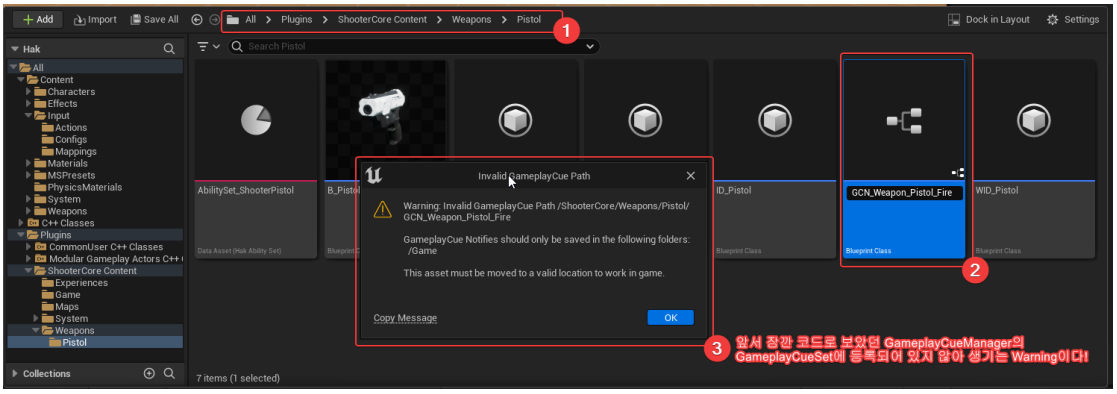


☐ GA_Weapon_Fire_Pistol에 GameplayCue 실행과 연동할 GameplayTag를 추가시키자:



☐ GameplayTag와 매칭되는 GameplayCue는 GCN(GameplayCueNotify)라는 에셋으로 만들어진다:





- 지금은 OK를 누르고 넘어가자

GameFeature_AddGameplayCuePaths

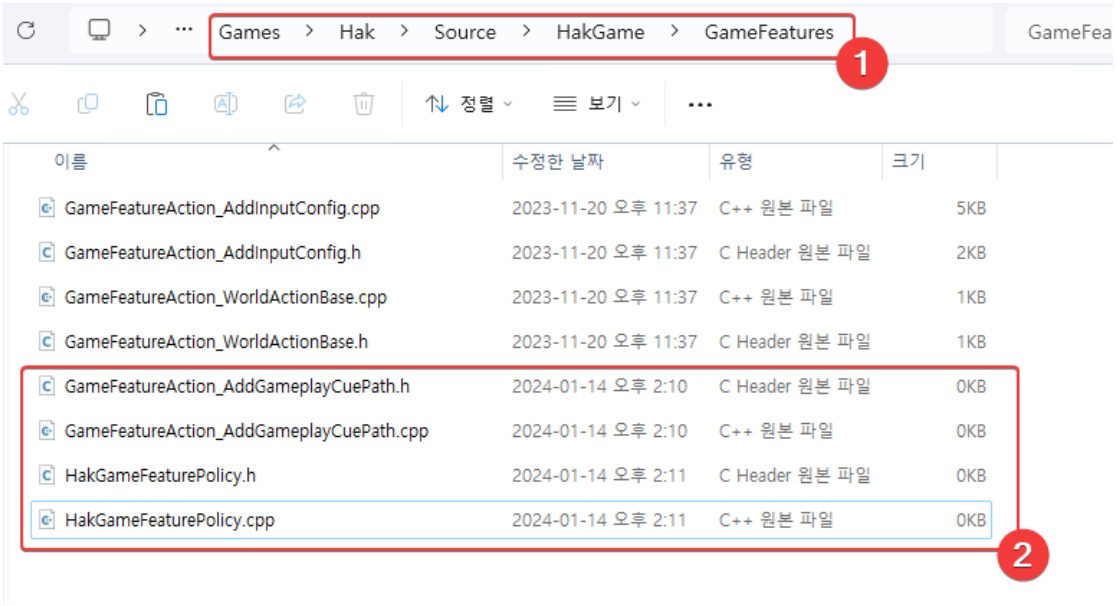
▼ **펼치기**

GameplayCueNotify는 GameplayCueNotify가 있는 경로를 추가해 줌으로써 GameplayCueManager가 게임 시작 부분에서 GameplayTag와 GameplayCueNotify를 연동하여, GameplayCueSet으로 관리될 수 있도록 해주어야 한다!

이를 GameFeatureAction으로 GameFeaturePlugin 별로 관리할 수 있게 진행하면, 기존 GameFeatureAction의 Activate/Deactivate와 달리, 우리가 추가할 **GameFeatureAction_AddGameplayCuePath는 Registering/Unregistering 과정에 등록되어야 한다:**

- 이는 게임 시작할 때, 끝날 때 초기화 가능하고, 런타임에 변경은 불가능한 GameFeature라는 의미이기도 하다

☐ GameFeatureAction_AddGameplayCuePath와 HakGameFeaturePolicy 파일 추가:



☐ GameFeatureAction_AddGameplayCuePath 구현:

```cpp
#pragma once

#include "CoreMinimal.h"
#include "GameFeatureAction.h"
#include "GameFeatureAction_AddGameplayCuePath.generated.h"

UCLASS()
class UGameFeatureAction_AddGameplayCuePath : public UGameFeatureAction
{
    GENERATED_BODY()
public:
    UGameFeatureAction_AddGameplayCuePath();

    /** GameplayCueNotify 에셋 경로 추가 */
    UPROPERTY(EditAnywhere, Category="GameFeature|GameplayCues")
    TArray<FDirectoryPath> DirectoryPathsToAdd;
};
```

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

UGameFeatureAction_AddGameplayCuePath::UGameFeatureAction_AddGameplayCuePath()
    : Super()
{
    DirectoryPathsToAdd.Add(FDirectoryPath{ TEXT("/GameplayCues") });
}
```

☐ HakGameFeaturePolicy 구조:

```cpp
#pragma once

#include "Containers/Array.h"
#include "GameFeatureStateChangeObserver.h"
#include "GameFeaturesProjectPolicies.h"
#include "UObject/Object.h"
#include "UObject/ObjectPtr.h"
#include "UObject/UObjectGlobals.h"
#include "HakGameFeaturePolicy.generated.h"

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

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

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

☐ GameFeature_AddGameplayCuePaths 구조:

```cpp
#pragma once

#include "Containers/Array.h"
#include "GameFeatureStateChangeObserver.h"
#include "GameFeaturesProjectPolicies.h"
#include "UObject/Object.h"
#include "UObject/ObjectPtr.h"
#include "UObject/UObjectGlobals.h"
#include "HakGameFeaturePolicy.generated.h"

UCLASS()
class UHakGameplayFeaturePolicy : public UDefaultGameFeaturesProjectPolicies
{
    GENERATED_BODY()
public:
    UHakGameplayFeaturePolicy(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
};

/**
 * GameFeature Plugin이 Register/Unregister 단계에서 GameplayCuePath가 등록되어 GameplayCueManager가 관련 GameplayCue 에셋을 스캔할 수 있도록 해야 한다:
 * 이를 위해 GameFeatureStateChangeObserver를 활용하여, 옵저버 패턴으로 이를 가능하게 하도록 한다:
 * - 아래에 우리가 Override할 메서드에서 유추할 수 있다시피 GameFeatureStateChangeObserver를 활용하면 가능하다!
 * - 이를 위해 추가적 구현이 필요한데 이는 UHakGameplayFeaturePolicy를 참고하자.
 *
 * 참고로, GameFeatureAction에도 Registering/Unregistering이 있긴하다:
 * - 허나, 우리가 앞서 구현했다시피, 직접 Registering/Unregistering 호출해줘야 한다
 *   - Registering/Unregistering 구현해봐야 호출하지 않으면 아무 의미 없다...
 * - Lyra에서는 GameplayFeaturePolicy를 오버라이딩하는 방식으로 이를 진행하였다
 */
UCLASS()
class UHakGameFeature_AddGameplayCuePaths : public UObject, public IGameFeatureStateChangeObserver
{
    GENERATED_BODY()
public:
    /**
     * IGameFeatureStateChangeObserver interface
     */
    virtual void OnGameFeatureRegistering(const UGameFeatureData* GameFeatureData, const FString& PluginName, const FString& PluginURL) override;
    virtual void OnGameFeatureUnregistering(const UGameFeatureData* GameFeatureData, const FString& PluginName, const FString& PluginURL) override;
};
```

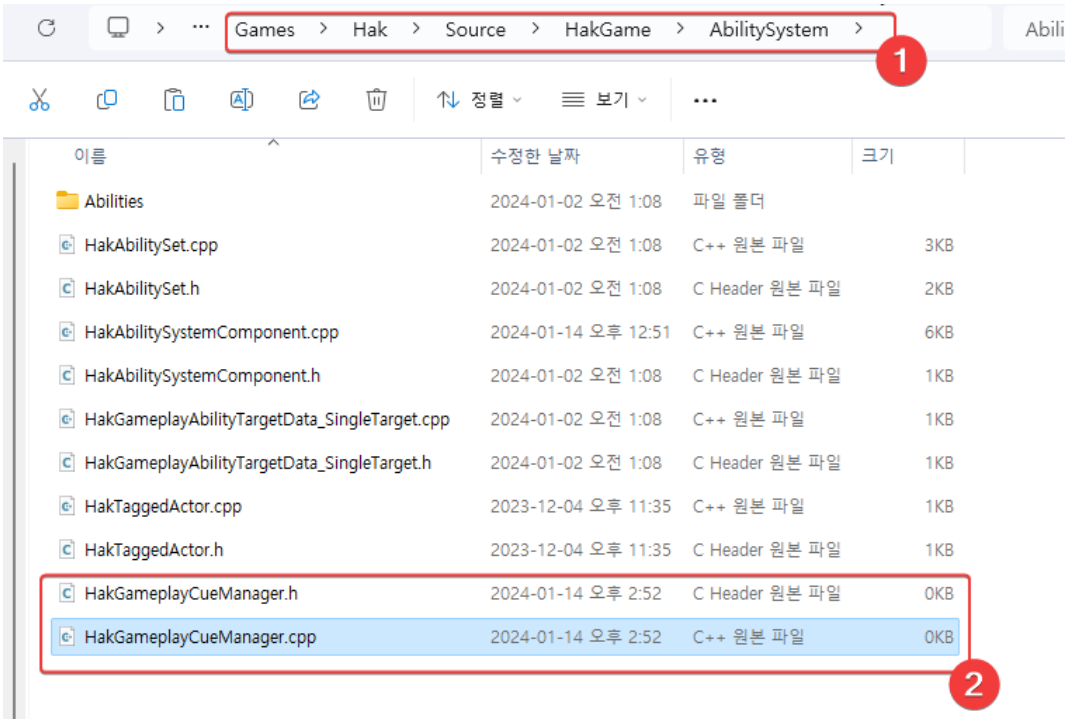
```cpp
#include "HakGameFeaturePolicy.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakGameFeaturePolicy)

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

void UHakGameFeature_AddGameplayCuePaths::OnGameFeatureRegistering(const UGameFeatureData* GameFeatureData, const FString& PluginName, const FString& PluginURL)
{

}

void UHakGameFeature_AddGameplayCuePaths::OnGameFeatureUnregistering(const UGameFeatureData* GameFeatureData, const FString& PluginName, const FString& PluginURL)
{

}
```

☐ HakGameFeaturePolicy 구현:

```cpp
#pragma once

#include "Containers/Array.h"
#include "GameFeatureStateChangeObserver.h"
#include "GameFeaturesProjectPolicies.h"
#include "UObject/Object.h"
#include "UObject/ObjectPtr.h"
#include "UObject/UObjectGlobals.h"
#include "HakGameFeaturePolicy.generated.h"

/**
 * UHakGameplayFeaturePolicy는 GameFeature Plugin이 Memory에 로딩되거나 Active(활성화)를 관리하는 StateMachine을 모니터링 가능하다
 */
UCLASS()
class UHakGameplayFeaturePolicy : public UDefaultGameFeaturesProjectPolicies
{
    GENERATED_BODY()
public:
    UHakGameplayFeaturePolicy(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * GameFeaturesProjectPolicies interfaces
     */
    virtual void InitGameFeatureManager() override;
    virtual void ShutdownGameFeatureManager() override;

    /** Observer로서 등록한 객체를 관리 (아래 HakGameFeature_AddGameplayCuePaths가 하나의 예시) */
    UPROPERTY(Transient)
    TArray<TObjectPtr<UObject>> Observers;
};

/**
 * GameFeature Plugin이 Register/Unregister 단계에서 GameplayCuePath가 등록되어 GameplayCueManager가 관련 GameplayCue 에셋을 스캔할 수 있도록 해야 한다:
 * 이를 위해 GameFeatureStateChangeObserver를 활용하여, 옵저버 패턴으로 이를 가능하게 하도록 한다:
 * - 아래에 우리가 Override할 메서드에서 유추할 수 있다시피 GameFeatureStateChangeObserver를 활용하면 가능하다!
 * - 이를 위해 추가적 구현이 필요한데 이는 UHakGameplayFeaturePolicy를 참고하자.
 *
 * 참고로, GameFeatureAction에도 Registering/Unregistering이 있긴하다:
 * - 허나, 우리가 앞서 구현했다시피, 직접 Registering/Unregistering 호출해줘야 한다
 *   - Registering/Unregistering 구현해봐야 호출하지 않으면 아무 의미 없다...
 * - Lyra에서는 GameplayFeaturePolicy를 오버라이딩하는 방식으로 이를 진행하였다
 */
UCLASS()
class UHakGameFeature_AddGameplayCuePaths : public UObject, public IGameFeatureStateChangeObserver
{
    GENERATED_BODY()
public:
    /**
     * IGameFeatureStateChangeObserver interface
     */
    virtual void OnGameFeatureRegistering(const UGameFeatureData* GameFeatureData, const FString& PluginName, const FString& PluginURL) override;
    virtual void OnGameFeatureUnregistering(const UGameFeatureData* GameFeatureData, const FString& PluginName, const FString& PluginURL) override;
};
```

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

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

void UHakGameplayFeaturePolicy::InitGameFeatureManager()
{
    // GameFeature_AddGameplayCuePaths를 등록
    Observers.Add(NewObject<UHakGameFeature_AddGameplayCuePaths>());

    // Observers를 순회하며, GameFeaturesSubsystem에 Observers를 순회하며 등록
    UGameFeaturesSubsystem& Subsystem = UGameFeaturesSubsystem::Get();
    for (UObject* Observer : Observers)
    {
        Subsystem.AddObserver(Observer);
    }

    // 앞서, Subsystem에 이미 Observer를 등록했으므로, GameFeatureManager가 초기화되는 과정에 반영될 것임
    Super::InitGameFeatureManager();
}

void UHakGameplayFeaturePolicy::ShutdownGameFeatureManager()
{
    Super::ShutdownGameFeatureManager();

    UGameFeaturesSubsystem& Subsystem = UGameFeaturesSubsystem::Get();
    for (UObject* Observer : Observers)
    {
        Subsystem.RemoveObserver(Observer);
    }
    Observers.Empty();
}

void UHakGameFeature_AddGameplayCuePaths::OnGameFeatureRegistering(const UGameFeatureData* GameFeatureData, const FString& PluginName, const FString& PluginURL)
{
}

void UHakGameFeature_AddGameplayCuePaths::OnGameFeatureUnregistering(const UGameFeatureData* GameFeatureData, const FString& PluginName, const FString& PluginURL)
{
}
```

- [ ] HakGameplayCueManager 구조:
  - [ ] HakGameplayCueManager 파일 추가:



  - [ ] HakGameplayCueManager 구조 구현:

```h
#pragma once

#include "GameplayCueManager.h"
#include "HakGameplayCueManager.generated.h"

UCLASS()
class UHakGameplayCueManager : public UGameplayCueManager
{
    GENERATED_BODY()
public:
    static UHakGameplayCueManager* Get();

    UHakGameplayCueManager(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
};
```

```cpp
#include "HakGameplayCueManager.h"
#include "AbilitySystemGlobals.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakGameplayCueManager)

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

UHakGameplayCueManager* UHakGameplayCueManager::Get()
{
    return Cast<UHakGameplayCueManager>(UAbilitySystemGlobals::Get().GetGameplayCueManager());
}
```

☐ GameplayCueManager 오버라이드:

DefaultGame.ini | HakGameplayCueManager.cpp | HakGameplayCueManager.h | HakGameFeaturePolicy.cpp | HakGameFeaturePolicy.h

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

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

[/Script/Engine.AssetManagerSettings]
-PrimaryAssetTypesToScan=(PrimaryAssetType="Map",AssetBaseClass=/Script/Engine.World,bHasBlueprintClas
-PrimaryAssetTypesToScan=(PrimaryAssetType="PrimaryAssetLabel",AssetBaseClass=/Script/Engine.PrimaryAs
+PrimaryAssetTypesToScan=(PrimaryAssetType="Map",AssetBaseClass="/Script/Engine.World",bHasBlueprintCl
+PrimaryAssetTypesToScan=(PrimaryAssetType="PrimaryAssetLabel",AssetBaseClass="/Script/Engine.PrimaryA
+PrimaryAssetTypesToScan=(PrimaryAssetType="HakUserFacingExperienceDefinition",AssetBaseClass="/Script
+PrimaryAssetTypesToScan=(PrimaryAssetType="HakExperienceDefinition",AssetBaseClass="/Script/HakGame.H
bOnlyCookProductionAssets=False
bShouldManagerDetermineTypeAndName=False
bShouldGuessTypeAndNameInEditor=True
bShouldAcquireMissingChunksOnLoad=False
bShouldWarnAboutInvalidAssets=True
MetaDataTagsForAssetRegistry=()
```

☐ HakGameFeature_AddGameplayCuePaths::OnGameFeatureRegistering()

```cpp
void UHakGameFeature_AddGameplayCuePaths::OnGameFeatureRegistering(const UGameFeatureData* GameFeatureData, const FString& PluginName, const FString& PluginURL)
{
    // PluginName을 활용하여, PluginRootPath를 계산
    const FString PluginRootPath = TEXT("/") + PluginName;

    // GameFeatureData의 Action을 순회:
    // - 예로 들어, ShooterCore의 GameFeatureData에 Action을 추가했다면, 그 추가된 Action이 대상이 됨
    for (const UGameFeatureAction* Action : GameFeatureData->GetActions())
    {
        // 그중에 우리는 _AddGameplayCuepath를 찾아서 처리
        if (const UGameFeatureAction_AddGameplayCuePath* AddGameplayCueGFA = Cast<UGameFeatureAction_AddGameplayCuePath>(Action))
        {
            const TArray<FDirectoryPath>& DirsToAdd = AddGameplayCueGFA->DirectoryPathsToAdd;

            // GameplayCueManager를 가져와서, GFA에 등록된 DirsToAdd를 추가하면서 GCM의 데이터가 업데이트 되도록 진행
            if (UHakGameplayCueManager* GCM = UHakGameplayCueManager::Get())
            {
                // RuntimeCueSet을 가져옴
                UGameplayCueSet* RuntimeGameplayCueSet = GCM->GetRuntimeCueSet();
                const int32 PreInitializeNumCues = RuntimeGameplayCueSet ? RuntimeGameplayCueSet->GameplayCueData.Num() : 0;

                for (const FDirectoryPath& Directory : DirsToAdd)
                {
                    FString MutablePath = Directory.Path;

                    // PluginPackagePath를 한번 보정해 줌 -> 보정된 결과는 MutablePath로 들어옴
                    UGameFeaturesSubsystem::FixPluginPackagePath(MutablePath, PluginRootPath, false);

                    // GCM의 RuntimeGameplayCueObjectLibrary의 Path에 추가
                    GCM->AddGameplayCueNotifyPath(MutablePath, /*bShouldRescanCueAssets=*/false);
                }

                // 초기화 시켜주고! (새로 경로가 추가되었으니 호출해줘야 함)
                if (!DirsToAdd.IsEmpty())
                {
                    GCM->InitializeRuntimeObjectLibrary();
                }

                // Cue의 에셋이 Pre와 Post랑 비교하여 갯수가 달라져있다면, 명시적으로 RefreshGameplayCuePrimaryAsset 호출
                const int32 PostInitializeNumCues = RuntimeGameplayCueSet ? RuntimeGameplayCueSet->GameplayCueData.Num() : 0;
                if (PreInitializeNumCues != PostInitializeNumCues)
                {
                    // RefreshGameplayCuePrimaryAsset 함수는 추가된 GCN을 AssetManager에 등록해주는 역할을 해준다
                    GCM->RefreshGameplayCuePrimaryAsset();
                }
            }
        }
    }
}
```

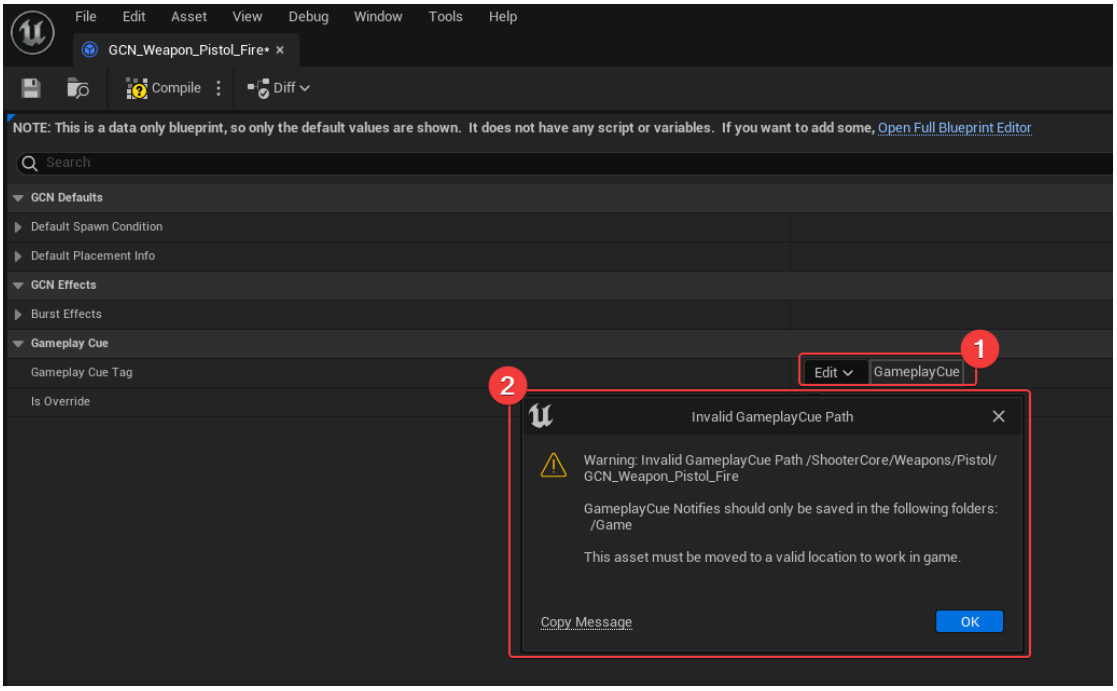
- [ ] HakGameCueManager::RefreshGameplayCuePrimaryAsset()

```cpp
#include "HakGameplayCueManager.h"
#include "AbilitySystemGlobals.h"
#include "GameplayCueSet.h"
#include "HakGame/System/HakAssetManager.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakGameplayCueManager)

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

UHakGameplayCueManager* UHakGameplayCueManager::Get()
{
    return Cast<UHakGameplayCueManager>(UAbilitySystemGlobals::Get().GetGameplayCueManager());
}

const FPrimaryAssetType UFortAssetManager_GameplayCueRefsType = TEXT("GameplayCueRefs");
const FName UFortAssetManager_GameplayCueRefsName = TEXT("GameplayCueReferences");
const FName UFortAssetManager_LoadStateClient = FName(TEXT("Client"));

void UHakGameplayCueManager::RefreshGameplayCuePrimaryAsset()
{
    TArray<FSoftObjectPath> CuePaths;
    UGameplayCueSet* RuntimeGameplayCueSet = GetRuntimeCueSet();
    if (RuntimeGameplayCueSet)
    {
        RuntimeGameplayCueSet->GetSoftObjectPaths(CuePaths);
    }

    // 새로 추가된 에셋의 경로를 CuePaths에 추가하여!
    FAssetBundleData BundleData;
    BundleData.AddBundleAssetsTruncated(UFortAssetManager_LoadStateClient, CuePaths);

    // PrimaryAssetId를 고정된 이름으로 설정하여 (const로 설정되어 있음)
    FPrimaryAssetId PrimaryAssetId = FPrimaryAssetId(UFortAssetManager_GameplayCueRefsType, UFortAssetManager_GameplayCueRefsName);

    // 에셋 매니저에 추가 진행
    UAssetManager::Get().AddDynamicAsset(PrimaryAssetId, FSoftObjectPath(), BundleData);
}
```

- [ ] HakGameFeature_AddGameplayCuePaths::OnGameFeatureUnregistering()

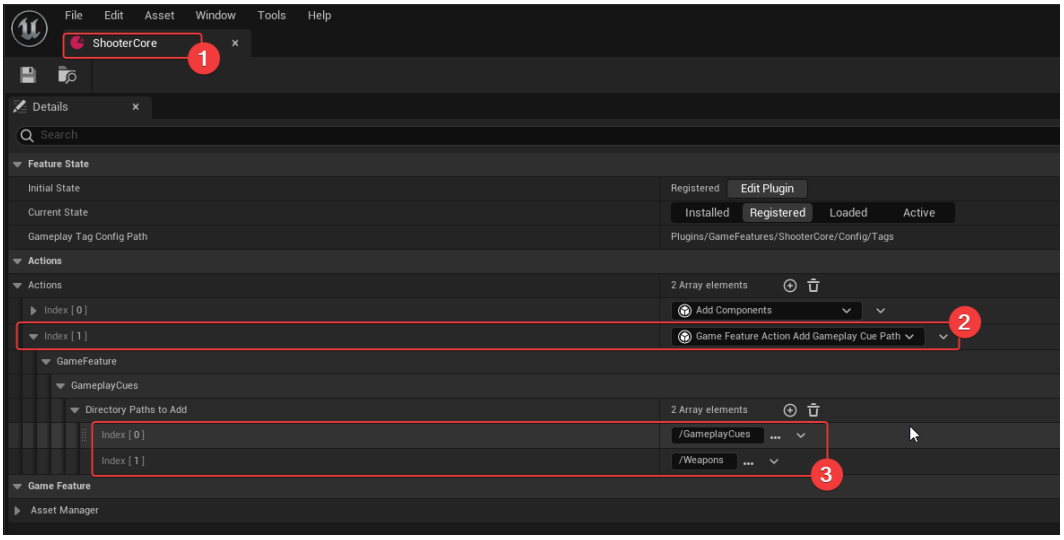
```cpp
void UHakGameFeature_AddGameplayCuePaths::OnGameFeatureUnregistering(const UGameFeatureData* GameFeatureData, const FString& PluginName, const FString& PluginURL)
{
    const FString PluginRootPath = TEXT("/") + PluginName;
    for (const UGameFeatureAction* Action : GameFeatureData->GetActions())
    {
        if (const UGameFeatureAction_AddGameplayCuePath* AddGameplayCueGFA = Cast<UGameFeatureAction_AddGameplayCuePath>(GameFeatureData))
        {
            const TArray<FDirectoryPath>& DirsToAdd = AddGameplayCueGFA->DirectoryPathsToAdd;
            if (UHakGameplayCueManager* GCM = UHakGameplayCueManager::Get())
            {
                int32 NumRemoved = 0;
                for (const FDirectoryPath& Directory : DirsToAdd)
                {
                    FString MutablePath = Directory.Path;
                    UGameFeaturesSubsystem::FixPluginPackagePath(MutablePath, PluginRootPath, false);
                    NumRemoved += GCM->RemoveGameplayCueNotifyPath(MutablePath, /** bShouldRescanCueAssets=*/ false);
                }

                ensure(NumRemoved == DirsToAdd.Num());

                if (NumRemoved > 0)
                {
                    GCM->InitializeRuntimeObjectLibrary();
                }
            }
        }
    }
}
```

☐ GCN_Weapon_Pistol_Fire를 수정을 가하면, 아직까지 아까와 똑같은 에러가 뜬다:



☐ 앞서 로직을 작성하면서, GameFeatureData의 GFA를 참조하여 경로를 추가했다:

- GameFeatureData에 AddGameplayCuePath GFA를 추가하자:

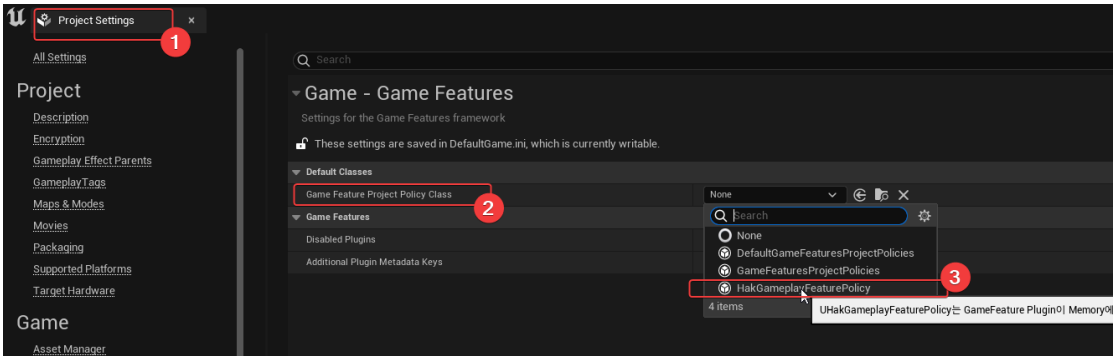


☐ HakGameFeaturePolicy에 해당 GFA가 잘 들어오는지 확인해보자:

```cpp
void UHakGameplayFeaturePolicy::InitGameFeatureManager()
{
    // GameFeature_AddGameplayCuePaths를 등록
    Observers.Add(NewObject<UHakGameFeature_AddGameplayCuePaths>());

    // Observers를 순회하며, GameFeaturesSubsystem에 Observers를 순회하며 등록
    UGameFeaturesSubsystem& Subsystem = UGameFeaturesSubsystem::Get();
    for (UObject* Observer : Observers)
    {
        Subsystem.AddObserver(Observer);
    }

    // 앞서, Subsystem에 이미 Observer를 등록했으므로, GameFeatureManager가 초기화되는 과정에 반영될 것임
    Super::InitGameFeatureManager();
}
```

- 들어오지 않는다! 이유는? HakGameplayFeaturePolicy를 오버라이드하지 않았다!

☐ GameFeatureProjectPolicyClass를 설정해주자:



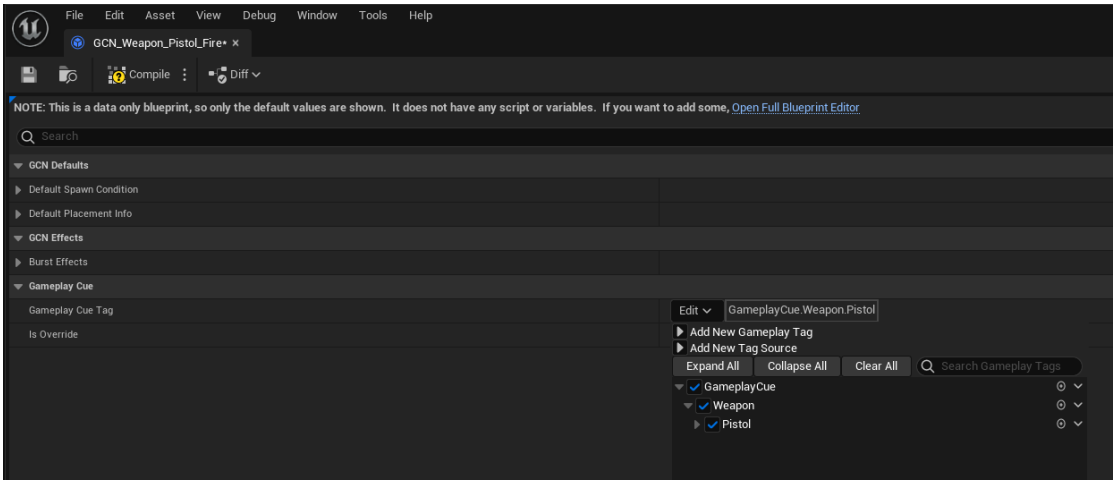
☐ 이제 아래와 같이 잘 들어온다:

```cpp
void UHakGameplayFeaturePolicy::InitGameFeatureManager()
{
    // GameFeature_AddGameplayCuePaths를 등록
    Observers.Add(NewObject<UHakGameFeature_AddGameplayCuePaths>());

    // Observers를 순회하며, GameFeaturesSubsystem에 Observers를 순회하며 등록
    UGameFeaturesSubsystem& Subsystem = UGameFeaturesSubsystem::Get();
    for (UObject* Observer : Observers)
    {
        Subsystem.AddObserver(Observer);
    }

    // 앞서, Subsystem에 이미 Observer를 등록했으므로, GameFeatureManager가 초기화되는 과정에 반영될 것임
    Super::InitGameFeatureManager();
}
```

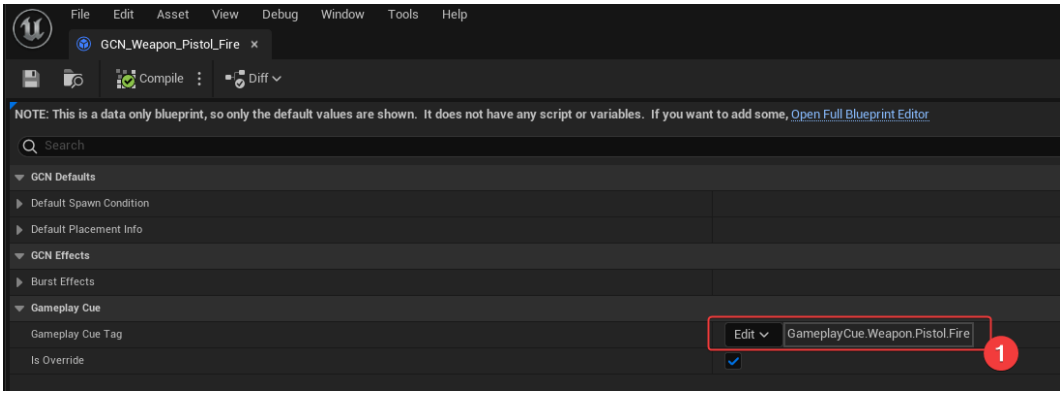
☐ 더 내려가서, GameplayCuePath가 잘 추가되는지도 보자:

```cpp
void UHakGameFeature_AddGameplayCuePaths::OnGameFeatureRegistering(const UGameFeatureData* GameFeatureData, const FString& PluginName, const FString& PluginURL)
{
    // PluginName을 활용하여, PluginRootPath를 계산
    const FString PluginRootPath = TEXT("/") + PluginName;

    // GameFeatureData의 Action을 순회:
    // - 예로 들어, ShooterCore의 GameFeatureData에 Action을 추가했다면, 그 추가된 Action이 대상이 됨
    for (const UGameFeatureAction* Action : GameFeatureData->GetActions())
    {
        // 그중에 우리는 _AddGameplayCuepath를 찾아서 처리
        if (const UGameFeatureAction_AddGameplayCuePath* AddGameplayCueGFA = Cast<UGameFeatureAction_AddGameplayCuePath>(Action))
        {
            const TArray<FDirectoryPath>& DirsToAdd = AddGameplayCueGFA->DirectoryPathsToAdd;

            // GameplayCueManager를 가져와서, GFA에 등록된 DirsToAdd를 추가하면서 GCM의 데이터가 업데이트 되도록 진행
            if (UHakGameplayCueManager* GCM = UHakGameplayCueManager::Get())
            {
                // RuntimeCueSet을 가져옴
                UGameplayCueSet* RuntimeGameplayCueSet = GCM->GetRuntimeCueSet();
                const int32 PreInitializeNumCues = RuntimeGameplayCueSet ? RuntimeGameplayCueSet->GameplayCueData.Num() : 0;

                for (const FDirectoryPath& Directory : DirsToAdd)
                {
                    FString MutablePath = Directory.Path;

                    // PluginPackagePath를 한번 보정해 줌 -> 보정된 결과는 MutablePath로 들어옴
                    UGameFeaturesSubsystem::FixPluginPackagePath(MutablePath, PluginRootPath, false); ≤ 5ms elapsed

                    // GCM의 RuntimeGameplayCueObjectLibrary의 Path에 추가
                    GCM->AddGameplayCueNotifyPath(MutablePath, /*bShouldRescanCueAssets=*/false);
                }
```

☐ 앞서 Warning이 나왔던 경우에도 아래와 같이 아무 문제 없이 잘 설정됨을 알 수 있다:



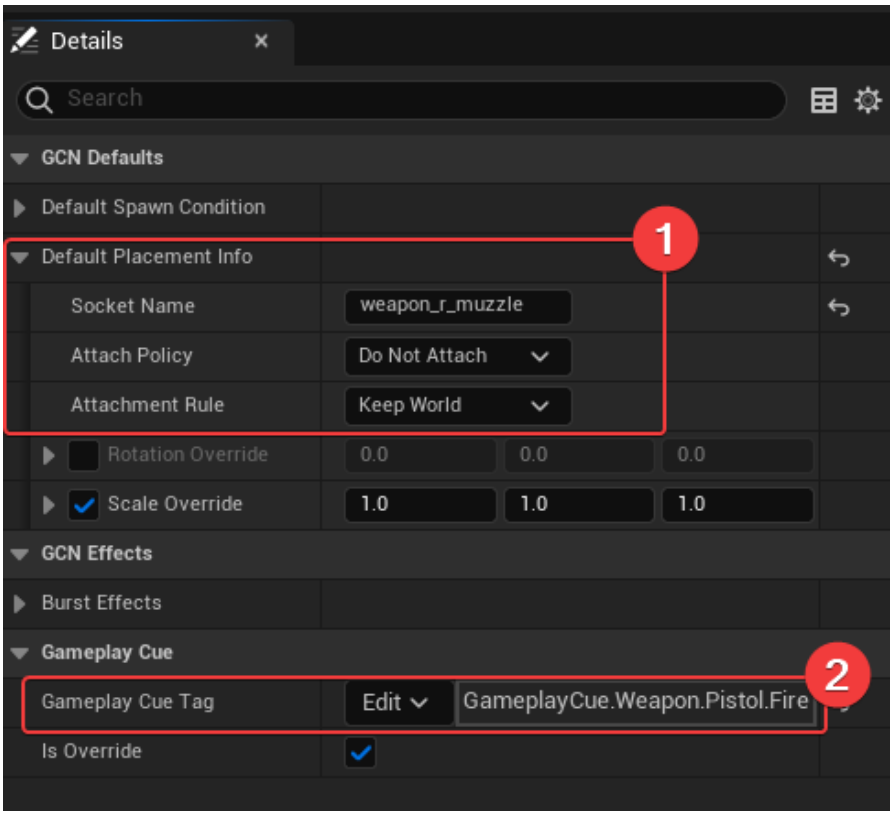
- GCN_Weapon_Pistol_Fire에 매칭되는 GameplayTag인 GameplayCue.Weapon.Pistol.Fire를 잘 설정해주자:



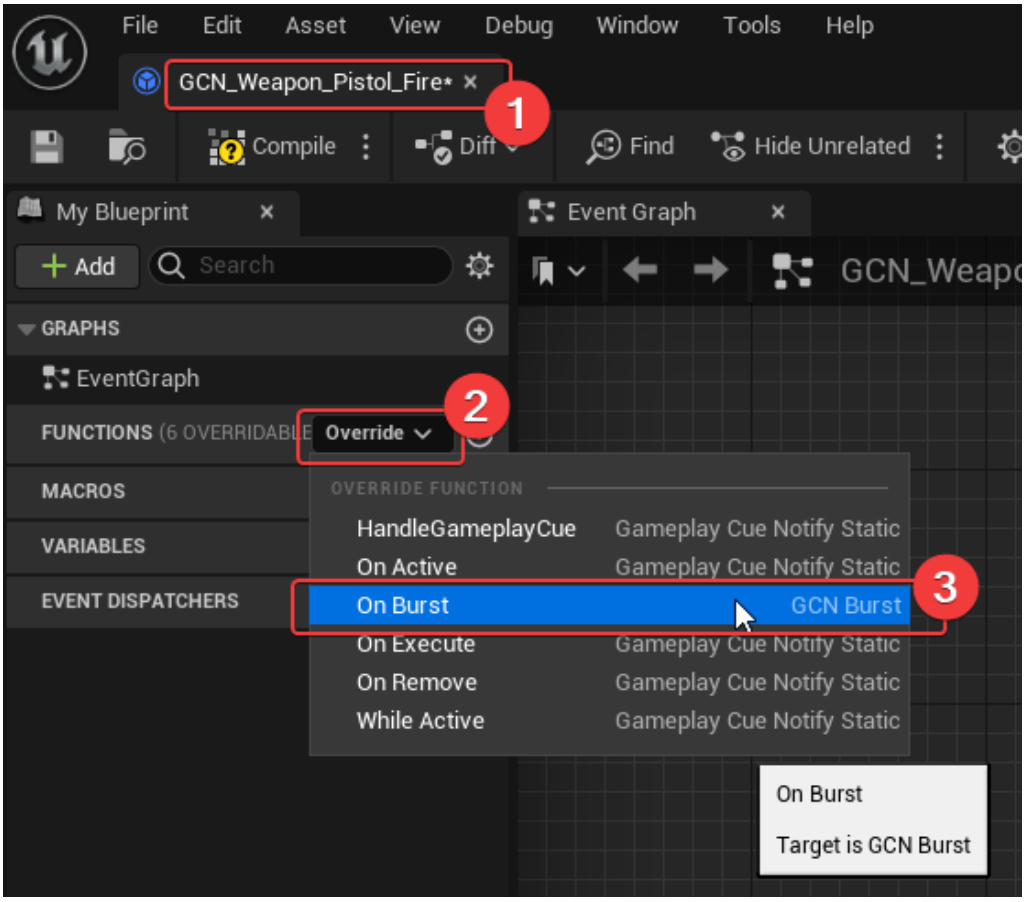
GCN_Weapon_Pistol_Fire

▼ **펼치기**

☐ GCN_Weapon_Pistol_Fire Default값 설정:

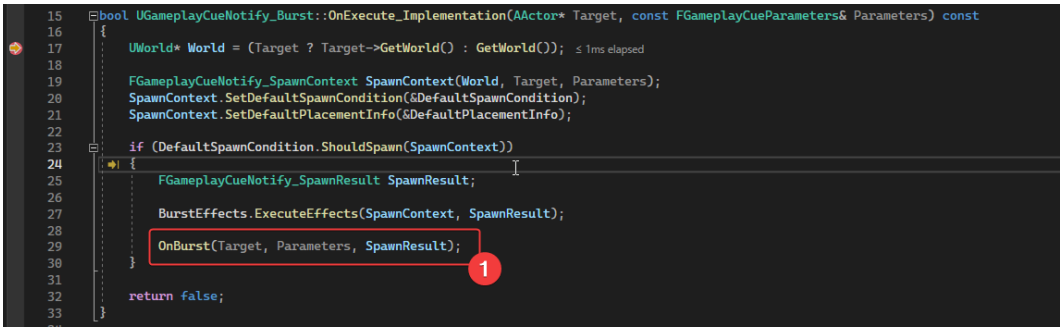


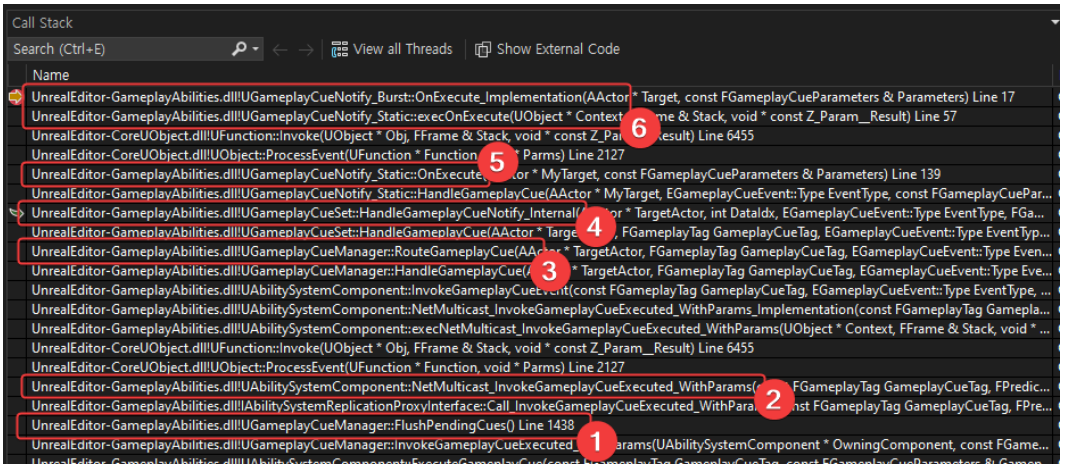
☐ OnBurst 함수 오버로딩:



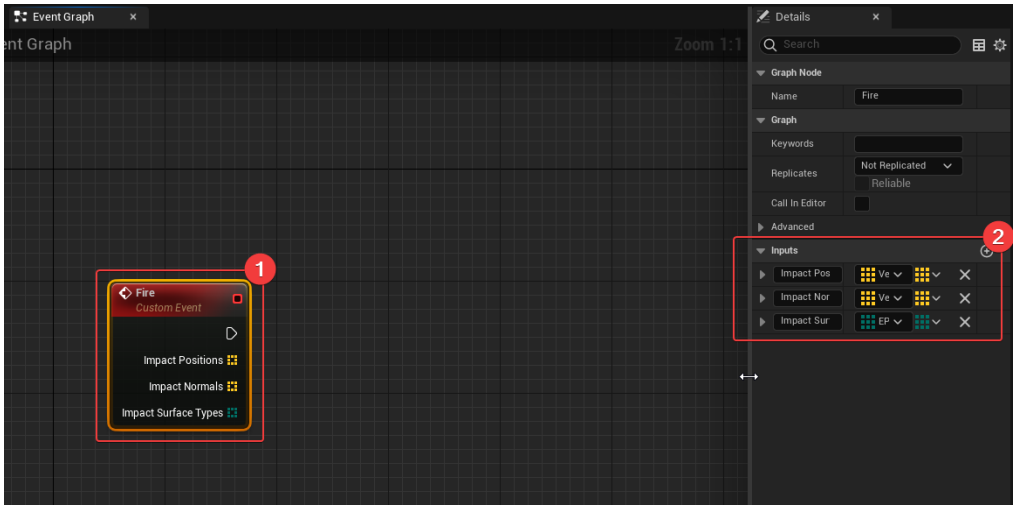
☐ OnBurst는 언제 불리는지 잠깐 확인해보자:

- 우리는 GCN_Weapon_Pistol_Fire에 대해 매칭되는 GameplayTag인 GameplayCue.Weapon.Pistol.Fire이 설정되어 있다.
- GameplayCueNotify_Burst::OnExecute_Implementation():

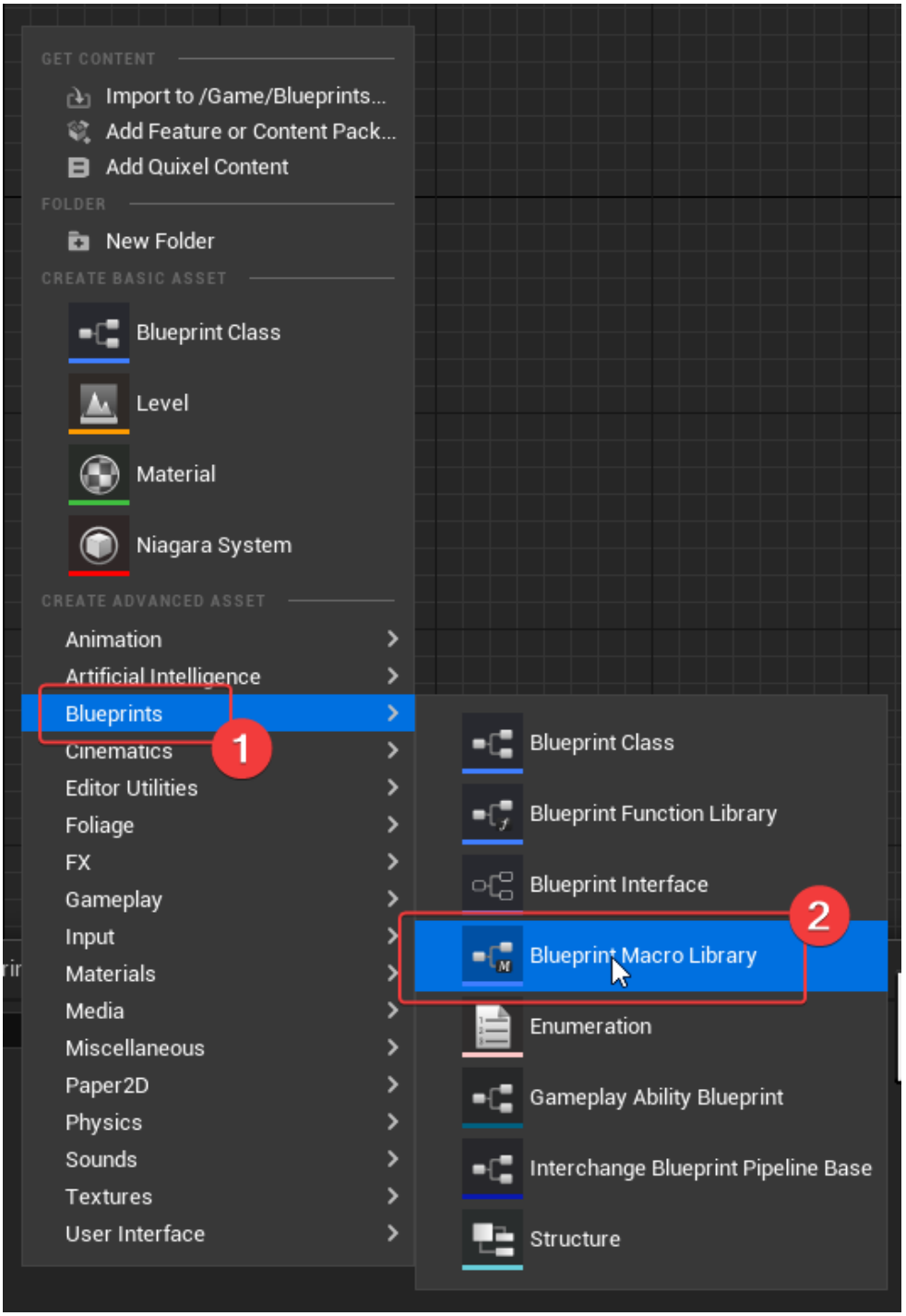


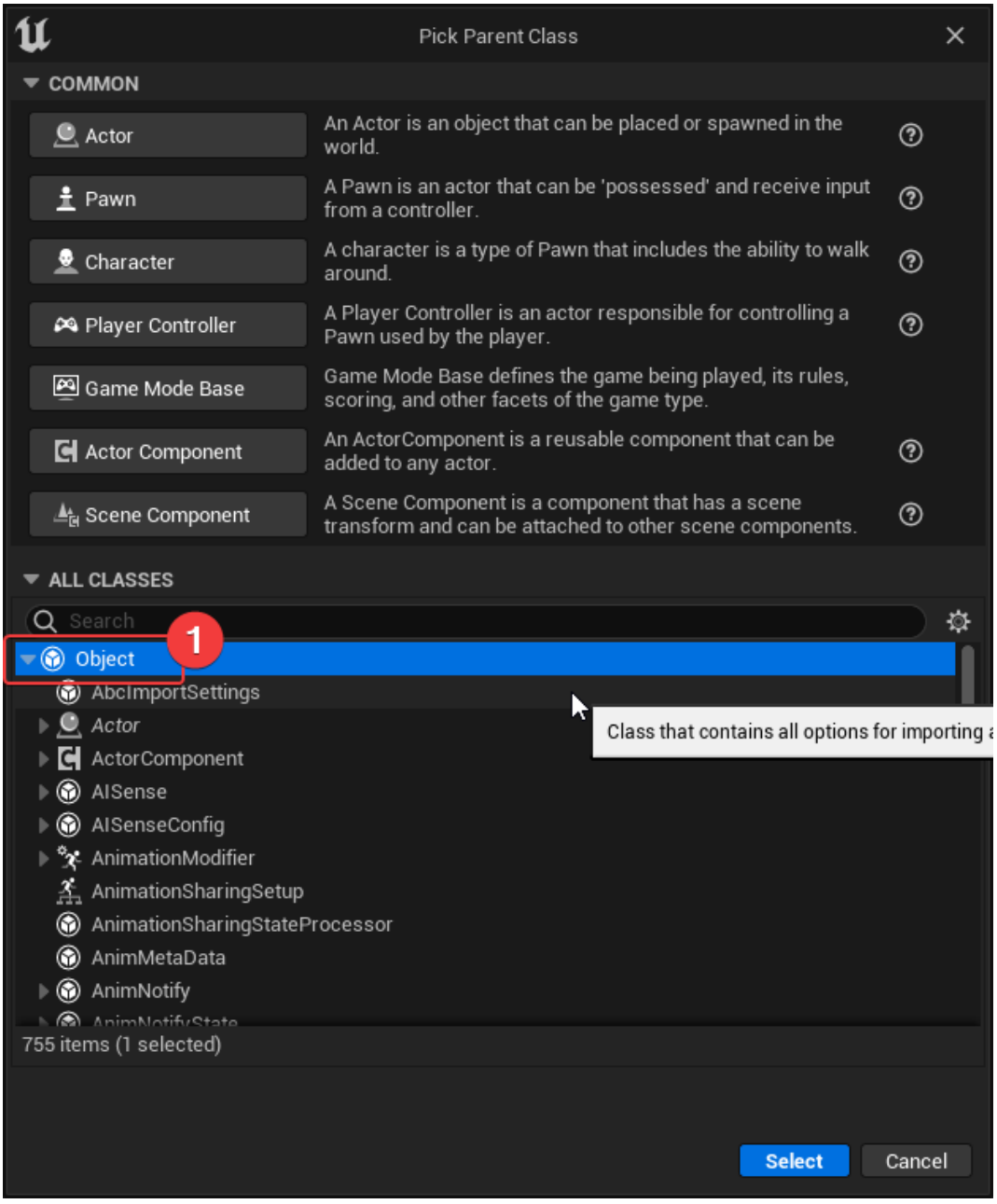


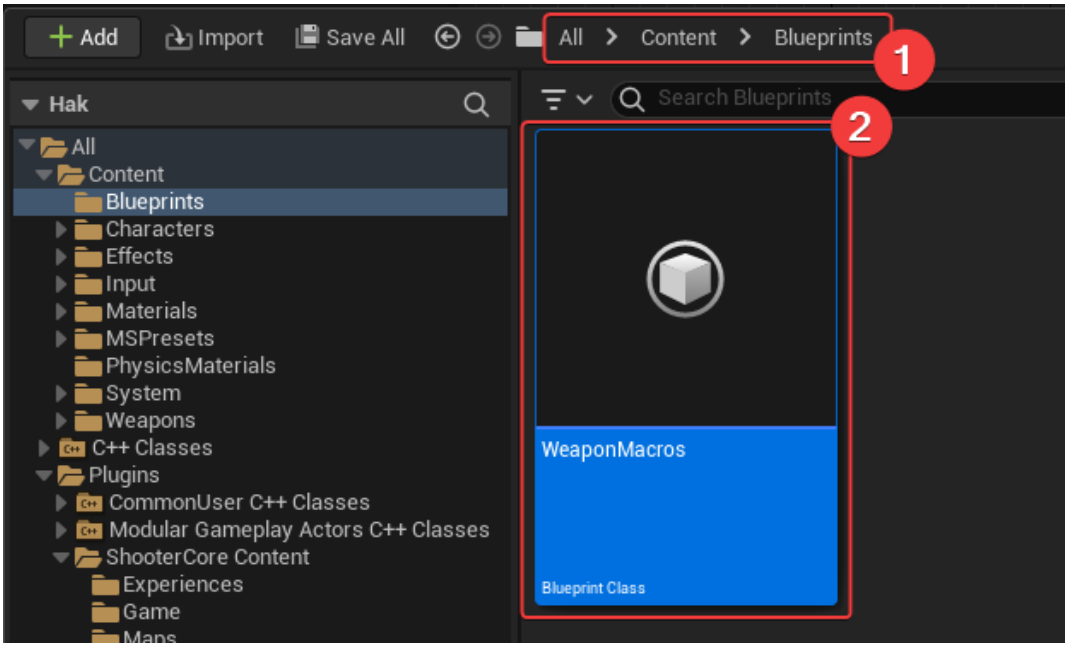
- [ ] OnBurst() BP 로직을 완성하자:
  - [ ] B_Weapon의 Custom Event로서 Fire를 생성해주자:
    - 3개의 Input도 같이 설정하자:



      - Impact Positions (Array<Vector>)
      - Impact Normals (Array<Vector>)
      - Impact Surface Types (Array<EPhysicalSurface>)
    - B_Pistol이 아닌 B_Weapon에 넣은 이유는 무기는 공통된 GameplayCue 실행 로직을 가지고 있기 때문이다.
  - [ ] Blueprint Macro Library 에셋을 생성하여, HakGetWeapon Macro를 추가하자:

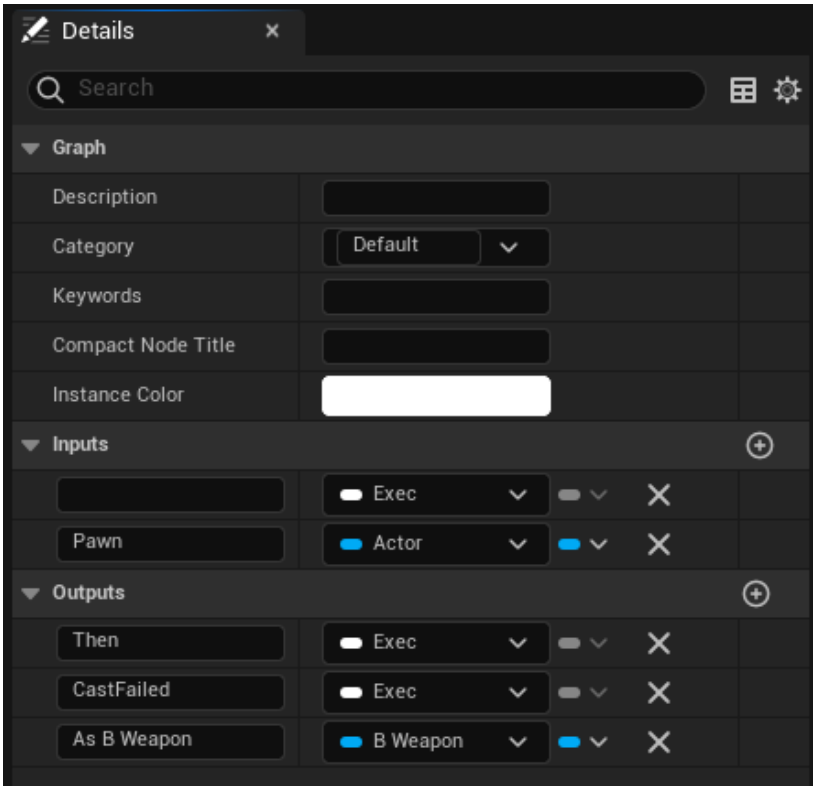

GET CONTENT
Import to /Game/Blueprints...
Add Feature or Content Pack...
Add Quixel Content
FOLDER
New Folder
CREATE BASIC ASSET
Blueprint Class
Level
Material
Niagara System
CREATE ADVANCED ASSET
Animation
Artificial Intelligence
Blueprints
Cinematics
Editor Utilities
Foliage
FX
Gameplay
Input
Materials
Media
Miscellaneous
Paper2D
Physics
Sounds
Textures
User Interface
1
Blueprint Class
Blueprint Function Library
Blueprint Interface
Blueprint Macro Library
2
Enumeration
Gameplay Ability Blueprint
Interchange Blueprint Pipeline Base
Structure


Pick Parent Class
COMMON
Actor
An Actor is an object that can be placed or spawned in the world.
Pawn
A Pawn is an actor that can be 'possessed' and receive input from a controller.
Character
A character is a type of Pawn that includes the ability to walk around.
Player Controller
A Player Controller is an actor responsible for controlling a Pawn used by the player.
Game Mode Base
Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.
Actor Component
An ActorComponent is a reusable component that can be added to any actor.
Scene Component
A Scene Component is a component that has a scene transform and can be attached to other scene components.
ALL CLASSES
Search
1
Object
AbcImportSettings
Actor
ActorComponent
AISense
AISenseConfig
AnimationModifier
AnimationSharingSetup
AnimationSharingStateProcessor
AnimMetaData
AnimNotify
Class that contains all options for importing
755 items (1 selected)
Select
Cancel

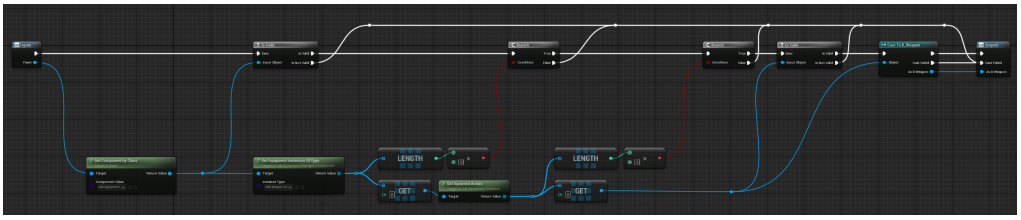


☐ 이름 변경하여, 새로운 Macro 추가하자:

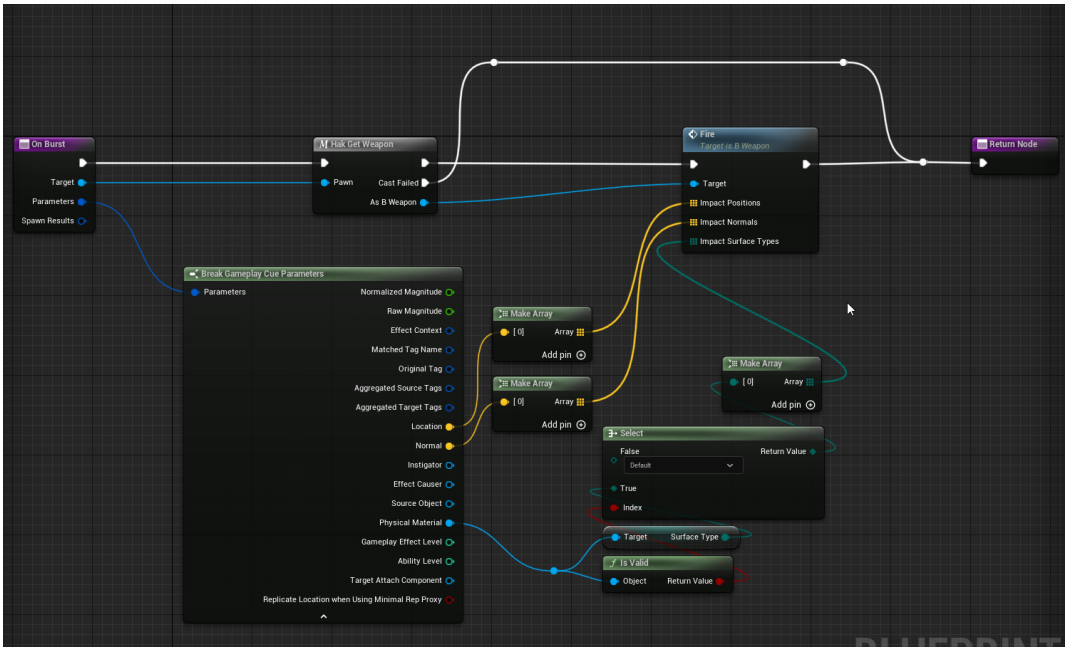
☐ Inputs와 Outputs를 설정해주자:



☐ HakGetWeapon 로직 완성하자:



☐ BP 로직을 완성하자:



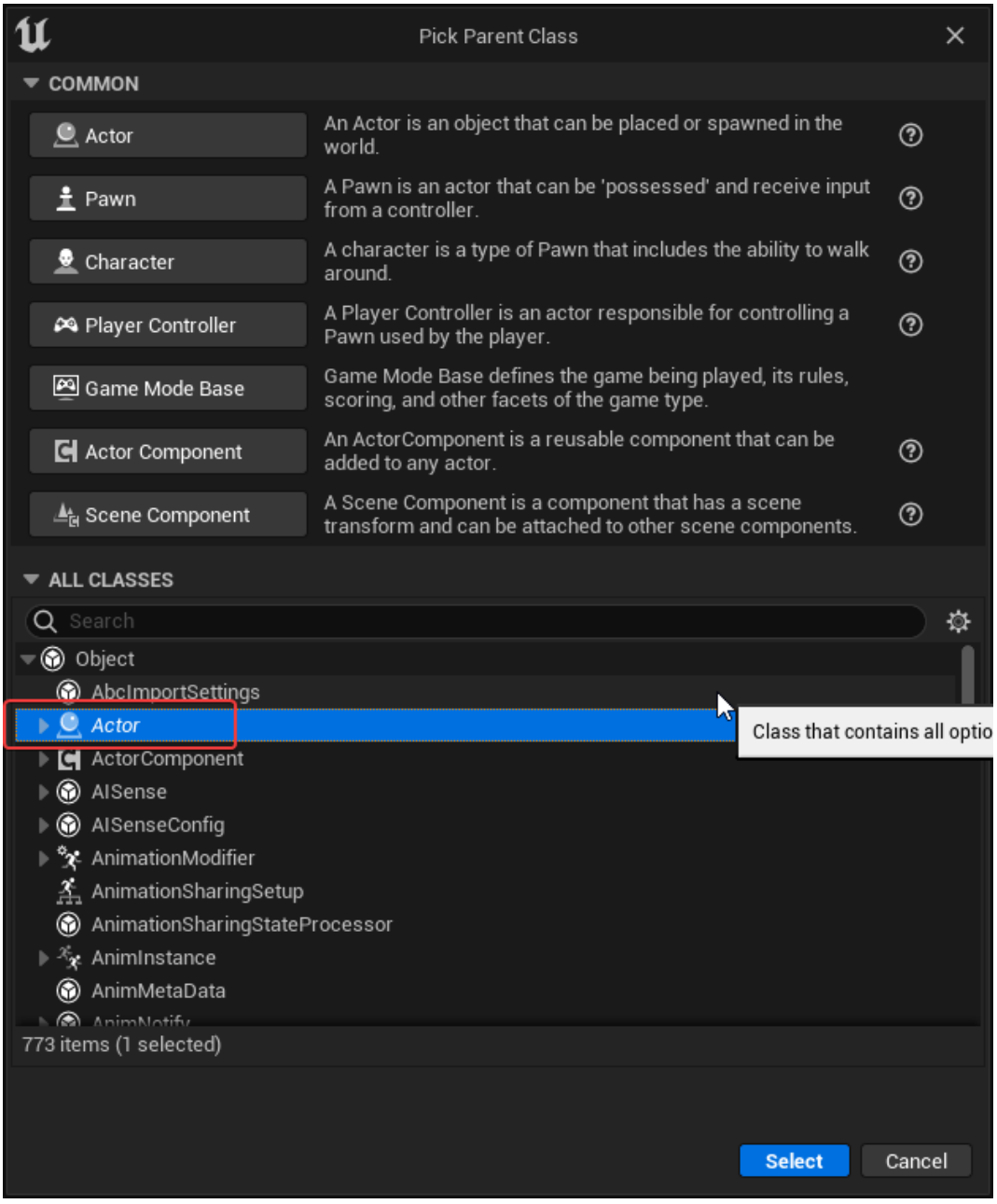
B_Weapon: Fire

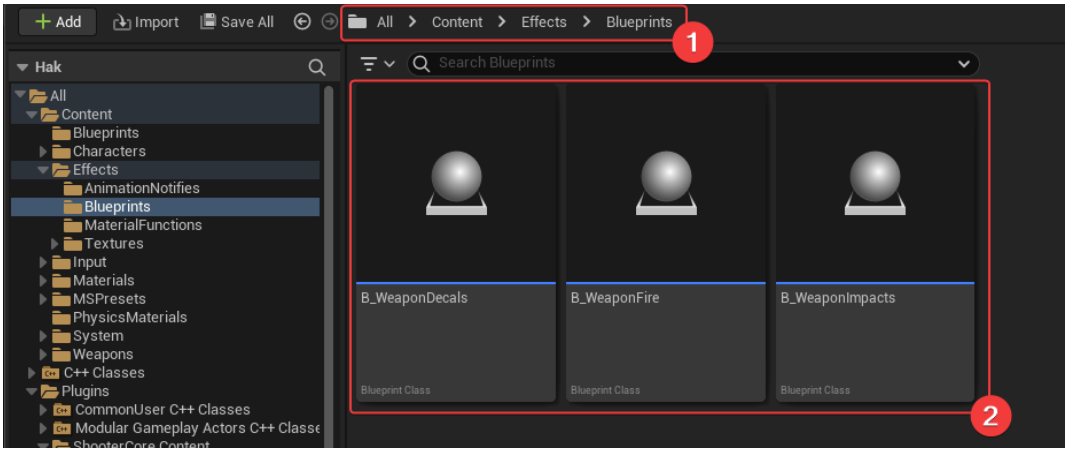
▼ **펼치기**

여기까지 오면, B_Weapon에서 Fire 로직을 처리하게 된다:

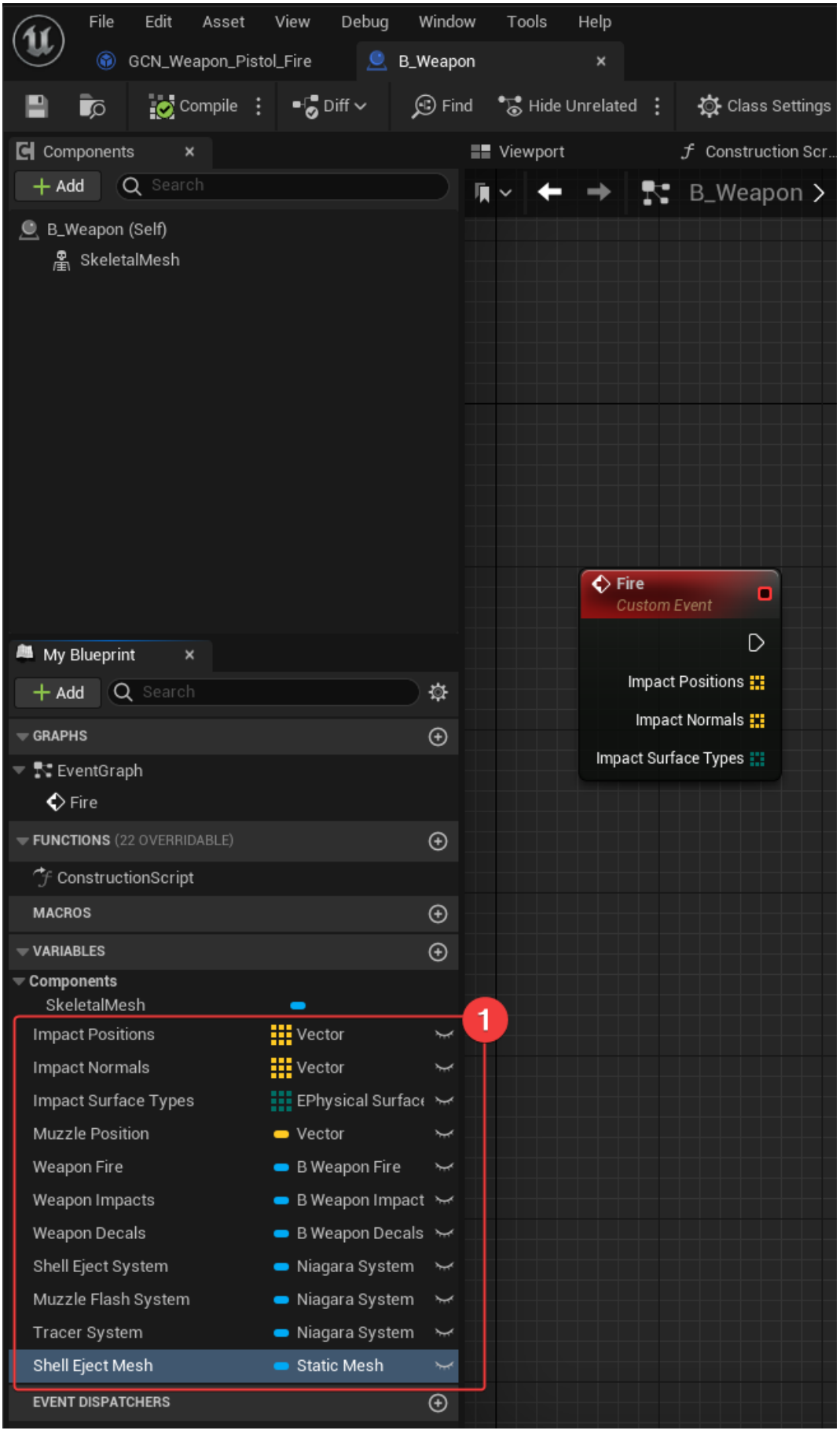
- [ ] B_Weapon: Fire에 필요한 모든 변수들을 미리 정의하자:
  - [ ] 변수에 필요한 BP 객체가 있다:
    - B_Weapon_Fire
      - 격발을 위한 이펙트 객체
    - B_Weapon_Impacts
      - 피격 이펙트 객체
    - B_Weapon_Decals
      - 피격 데칼 이펙트 객체

    해당 객체를 통해 필요한 Effect의 생성/파괴를 담당한다
  - [ ] 아래의 경로에 3개의 BP 객체를 정의하자:

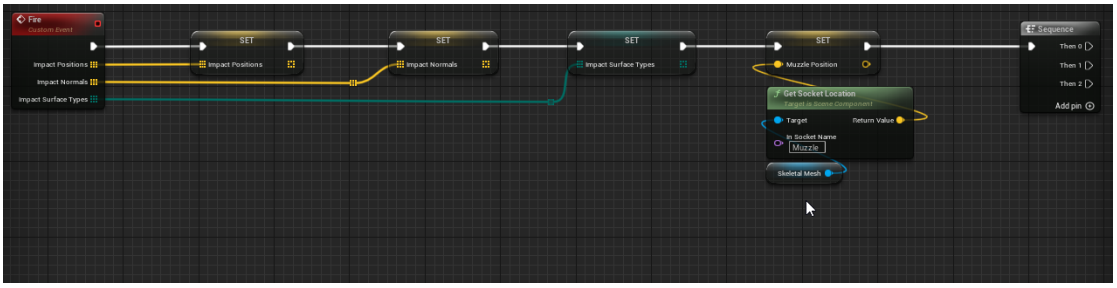




☐ BP 변수를 마저 정의하자:


File
Edit
Asset
View
Debug
Window
Tools
Help
GCN_Weapon_Pistol_Fire
B_Weapon
Compile
Diff
Find
Hide Unrelated
Class Settings
Components
Add
Search
B_Weapon (Self)
SkeletalMesh
Viewport
Construction Scr...
B_Weapon
Fire
Custom Event
Impact Positions
Impact Normals
Impact Surface Types
My Blueprint
Add
Search
GRAPHS
EventGraph
Fire
FUNCTIONS (22 OVERRIDABLE)
ConstructionScript
MACROS
VARIABLES
Components
SkeletalMesh
Impact Positions
Vector
Impact Normals
Vector
Impact Surface Types
EPhysical Surface
Muzzle Position
Vector
Weapon Fire
B Weapon Fire
Weapon Impacts
B Weapon Impact
Weapon Decals
B Weapon Decals
Shell Eject System
Niagara System
Muzzle Flash System
Niagara System
Tracer System
Niagara System
Shell Eject Mesh
Static Mesh
EVENT DISPATCHERS
1

☐ 우선 GCN에서 넘겨온 Inputs를 B_Weapon 멤버 변수에 저장하고 필요한 변수들도 준비하자:



- 앞서 우리는 Effect 종류로 Weapon_Fire, Weapon_Impacts 그리고 Weapon_Decals을 정의하였다. 이들을 하나씩 처리해보도록 하자.

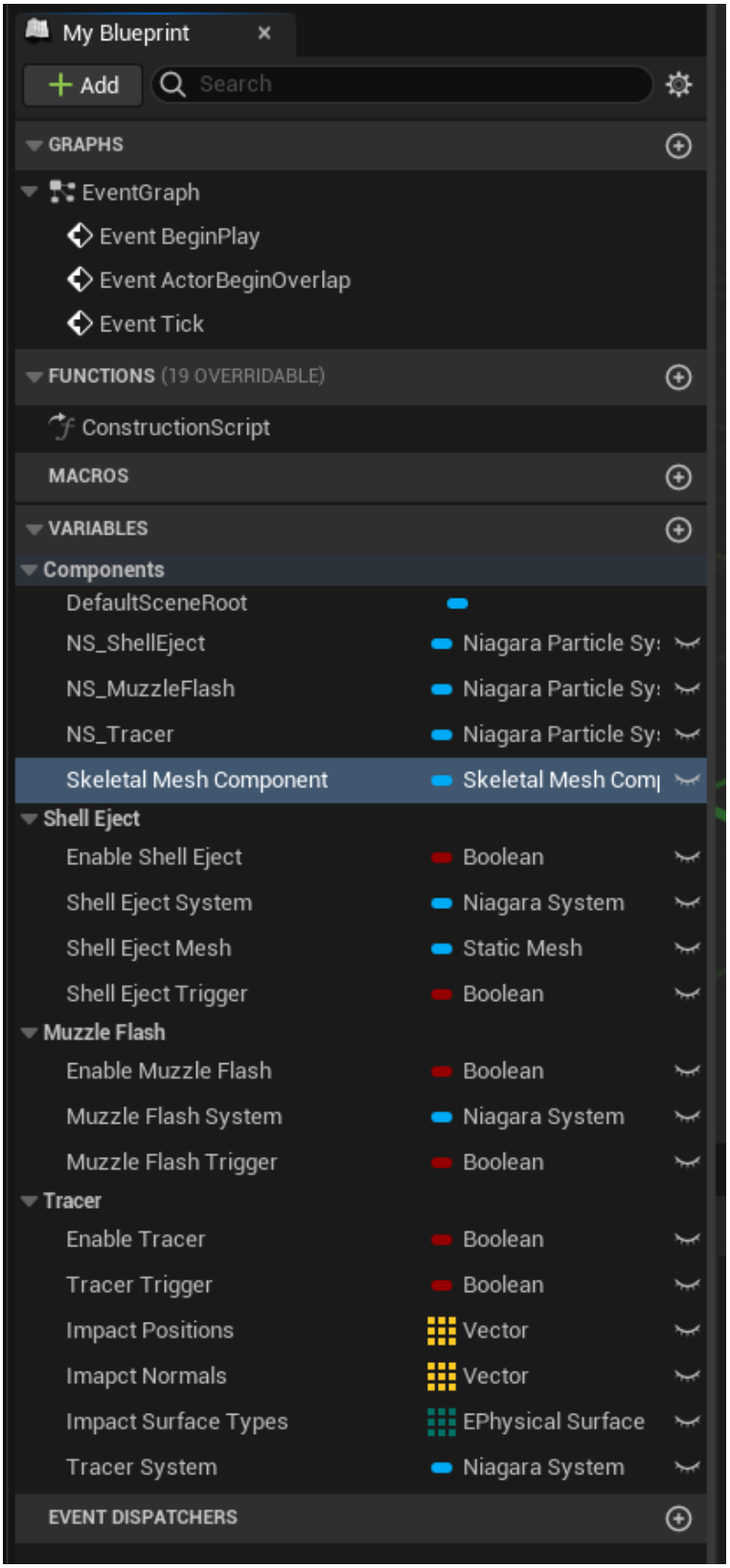
B_WeaponFire

▼ **펼치기**

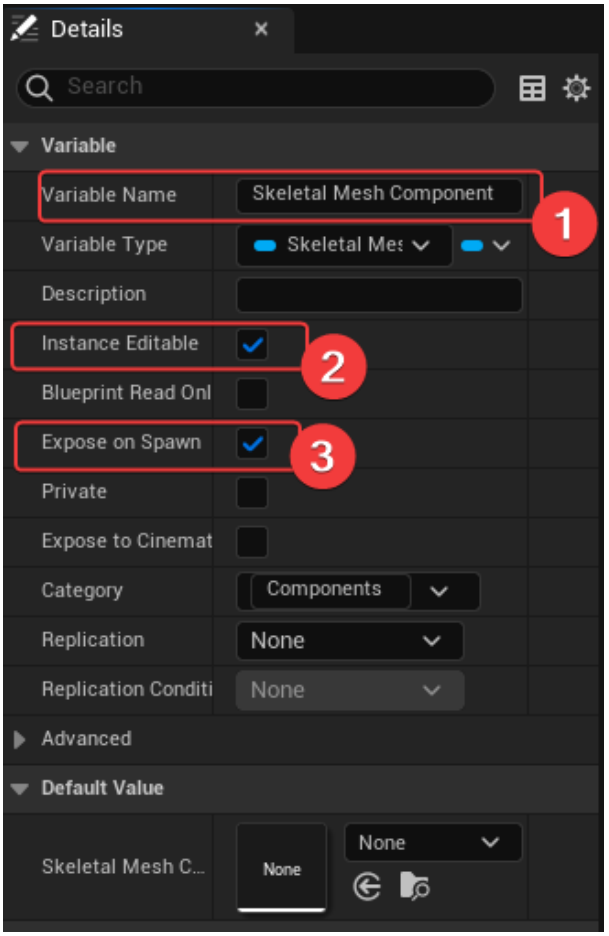
☐ B_Weapon에서 B_WeaponFire Actor를 생성하기에 앞서, B_WeaponFire에 필요한 변수를 정의하자:

- WeaponFire에는 세가지 이펙트 종류가 있다:
  - Shell Eject: 탄피 효과
  - Muzzle Flash: 머즐 효과
  - Tracer : 총탄 궤적 효과
- 위 세가지 이펙트를 Niagara Particle System Component 3개를 Actor에 생성하고, 일정 시간 지난 후에 Actor를 파괴하며, 생성된 3개의 Component 또한 제거된다

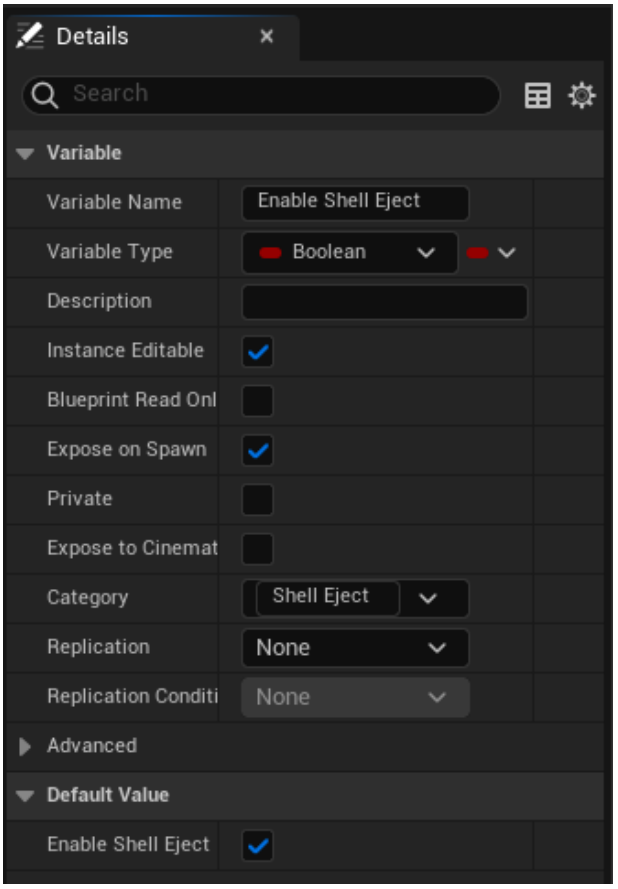
☐ 전체적인 멤버 변수들을 정의하자:



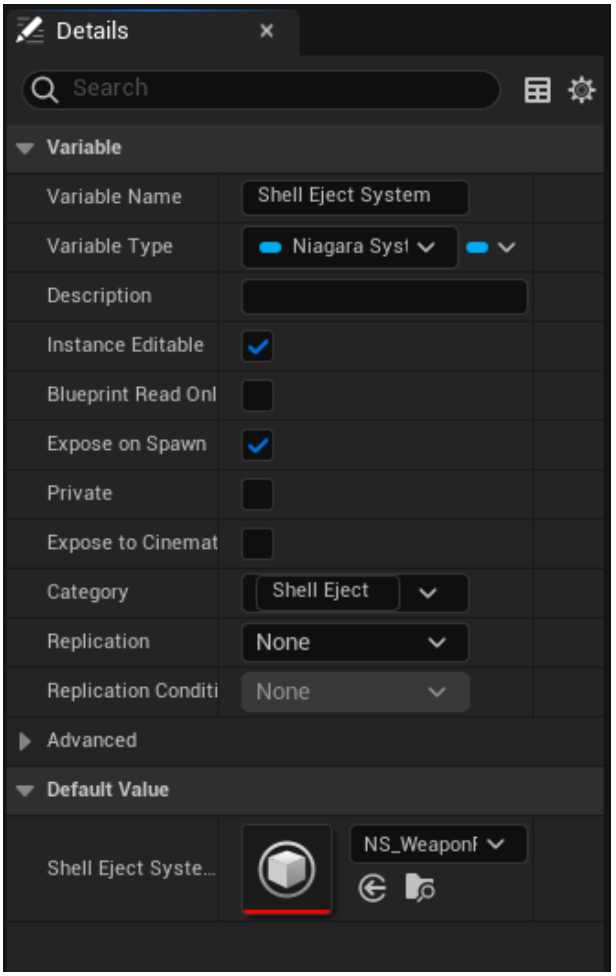
☐ Components의 Spawn 변수 정의하자:



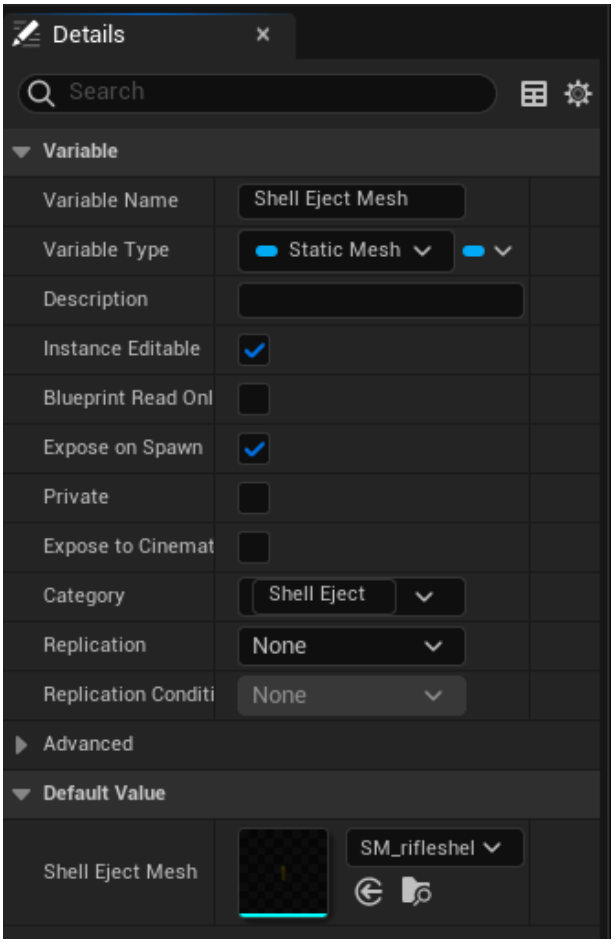
☐ Shell Eject의 기본값 및 Spawn 변수를 정의하자:



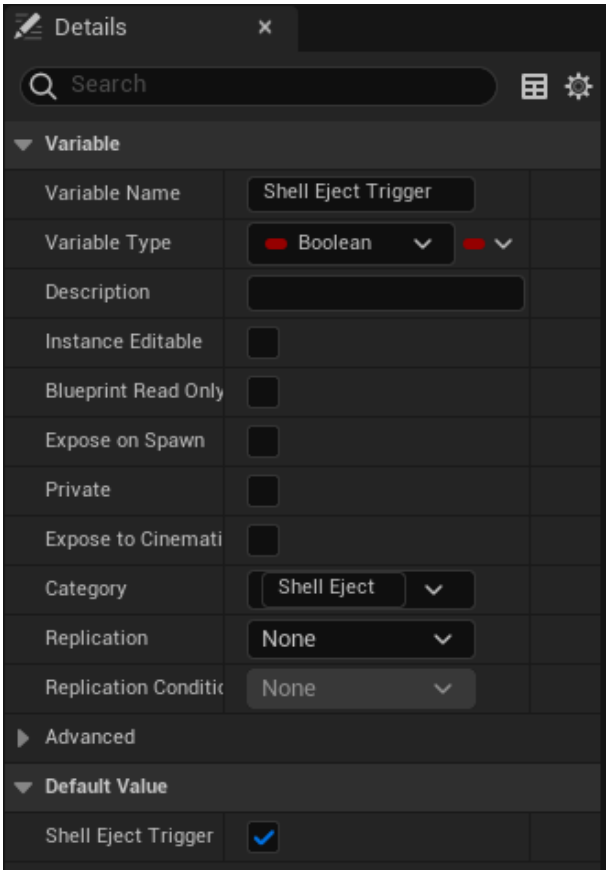
☐ NS_WeaponFire_ShellEject 임포트하고, Shell Eject System 기본값 설정하자:



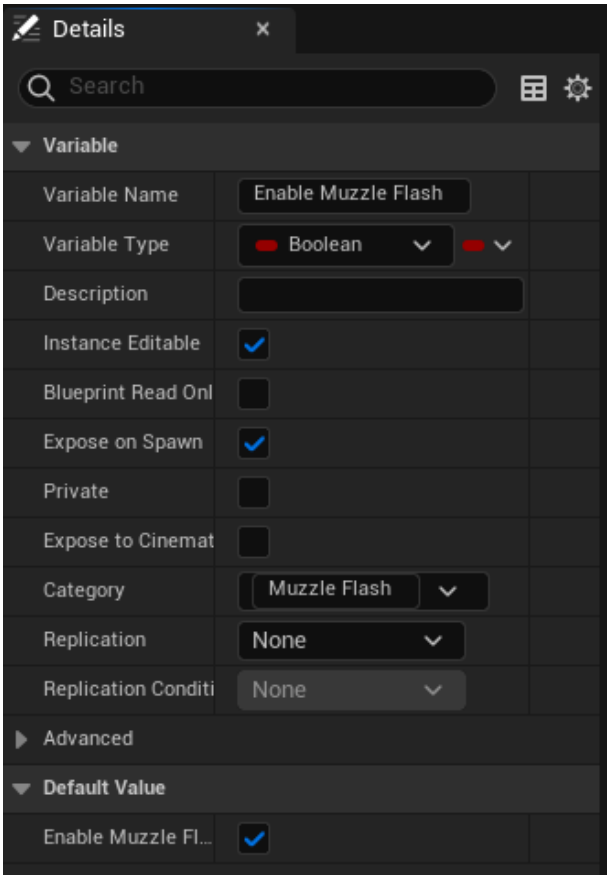
☐ SM_rifleshell 임포트하고, Shell Eject Mesh 기본값 설정하자:



☐ Shell Eject Trigger 기본값 설정:

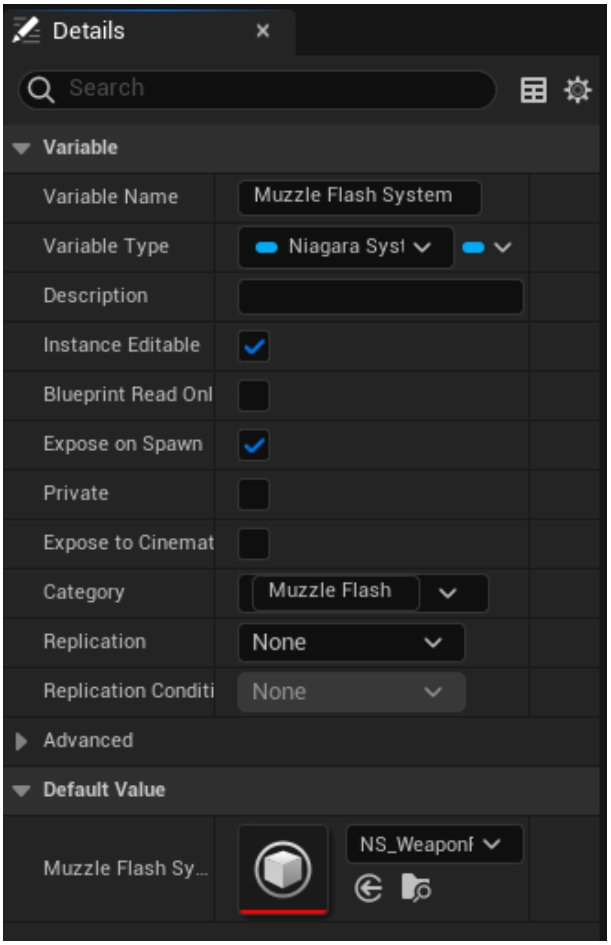


- [ ] Muzzle Flash의 기본값 Spawn 변수를 설정하자:
  - [ ] Enable Muzzle Flash

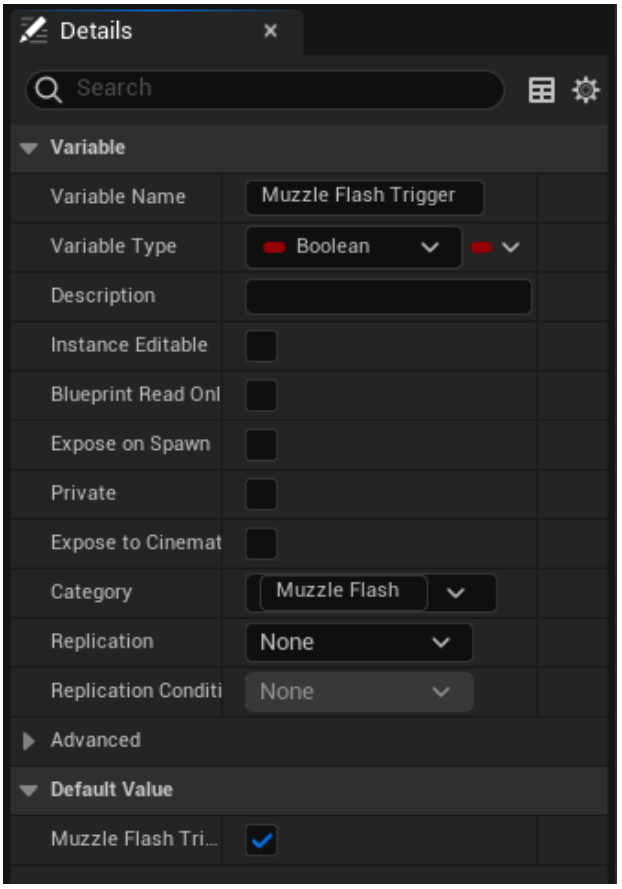


☐ Muzzle Flash System

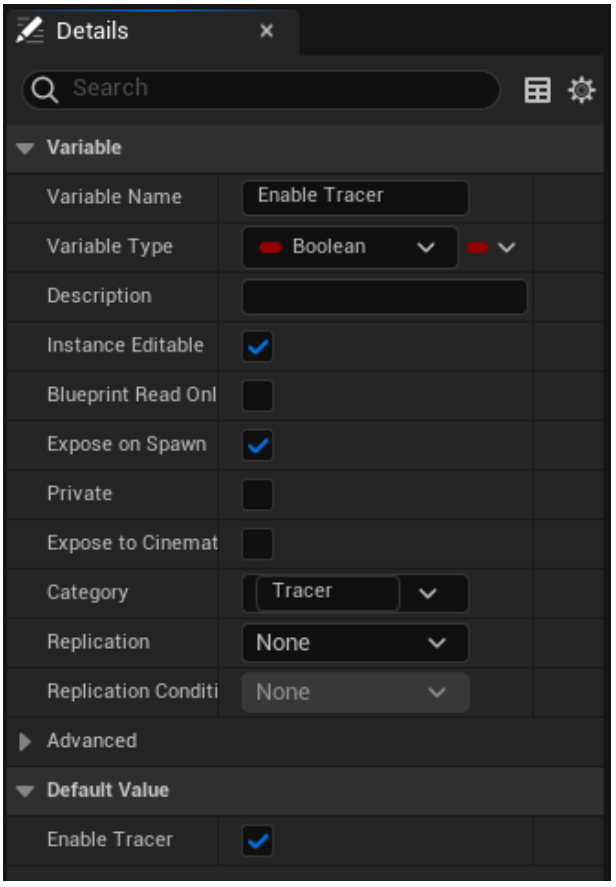
NS_WeaponFire 임포트 및 설정:/



- [ ] Muzzle Flash Trigger

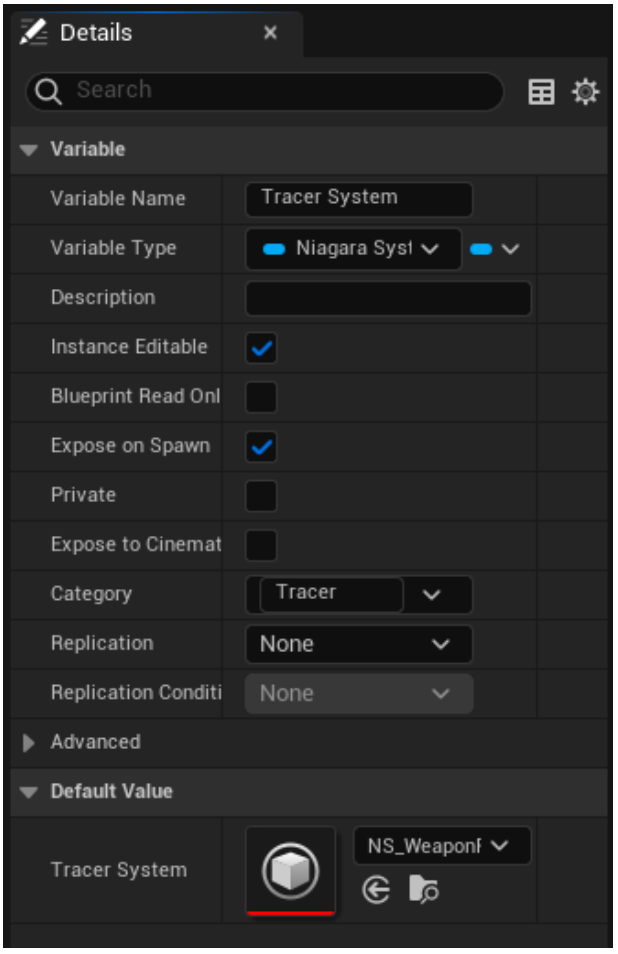


- [ ] Tracer의 기본값 Spawn 변수를 설정하자:
  - [ ] Enable Tracer

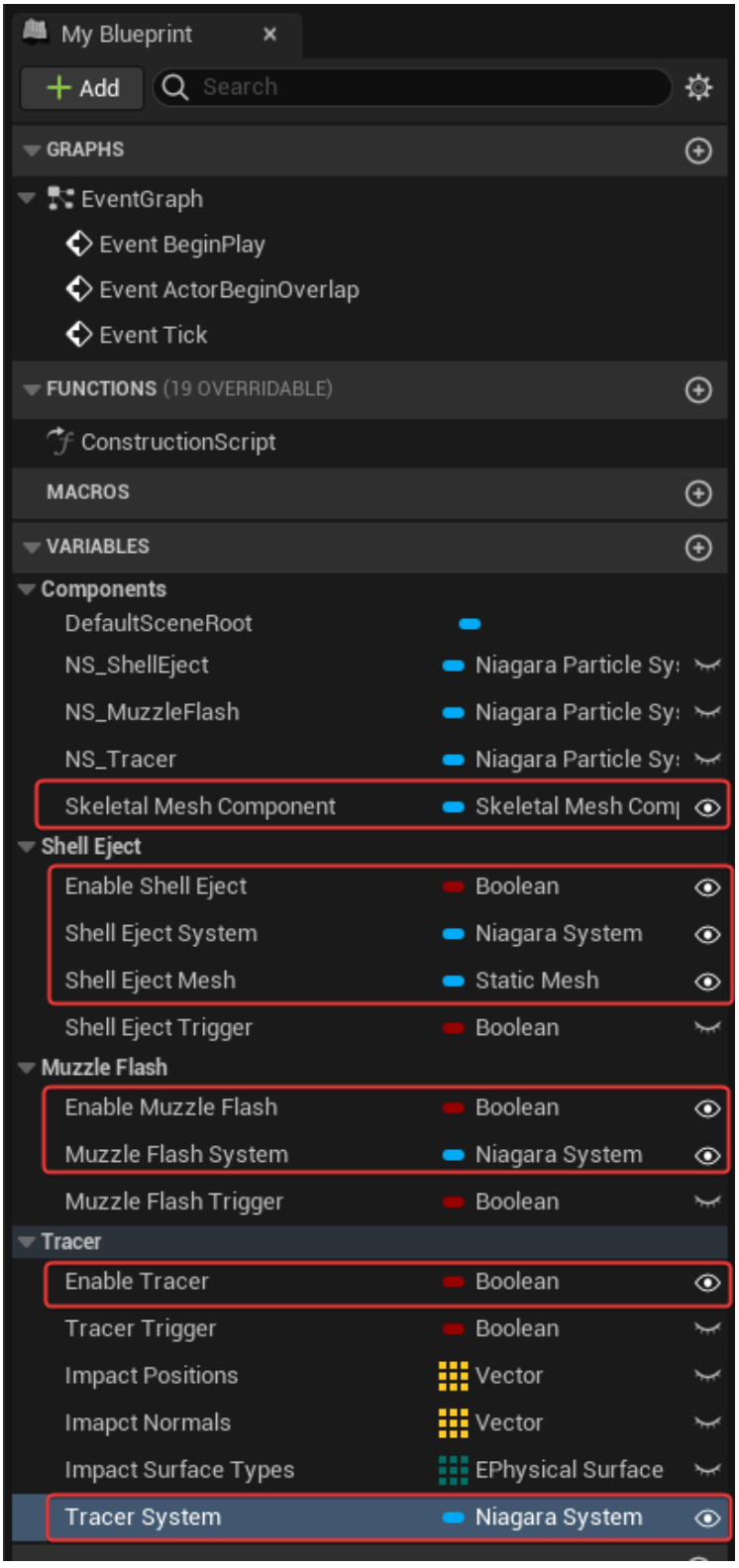


☐ Tracer System

NS_WeaponFire_Tracer 임포트 및 기본값 설정



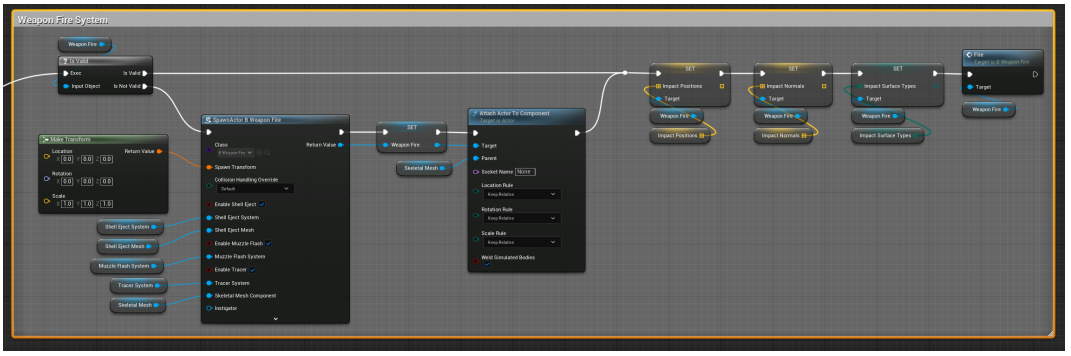
☐ B_WeaponFire를 생성할 경우 드러나는 변수들을 정리하면 아래와 같다:



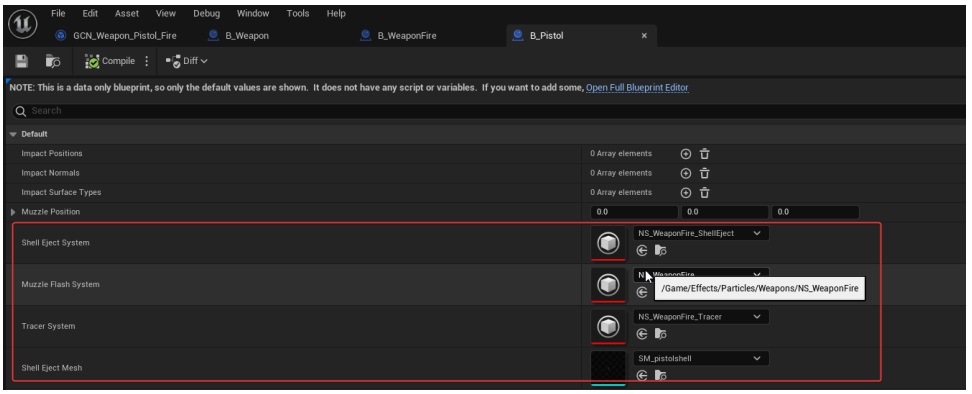
☐ 이제 B_WeaponFire를 Spawn할 변수들이 준비되었으니 B_Weapon의 Weapon Fire 로직을 완성하자:

- ☐ 우선 B_WeaponFire의 CustomEvent를 생성하자:

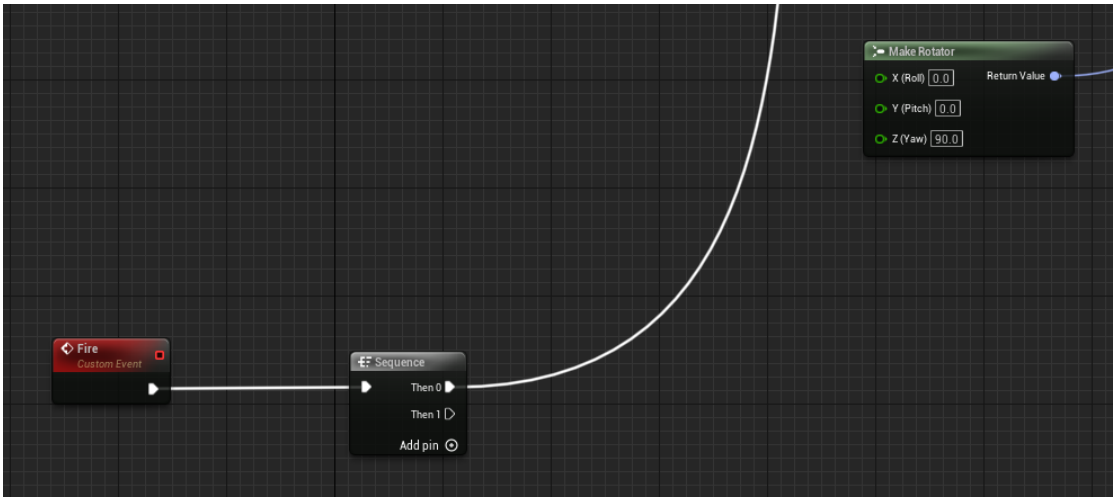
- [ ] 아래와 같이 로직을 완성하자:



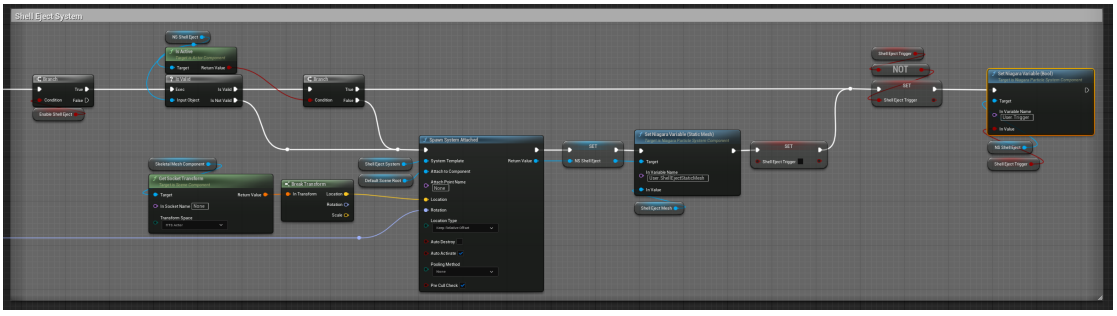
- SpawnActor하는 경우, Shell Eject System, Shell Eject Mesh, Muzzle Flash System, Tracer System은 B_Pistol에서 받기 때문에 이를 설정해주어야 한다:
  - [ ] SM_pistolshell 임포트
  - [ ] 아래와 같이 B_Pistol을 설정하자:



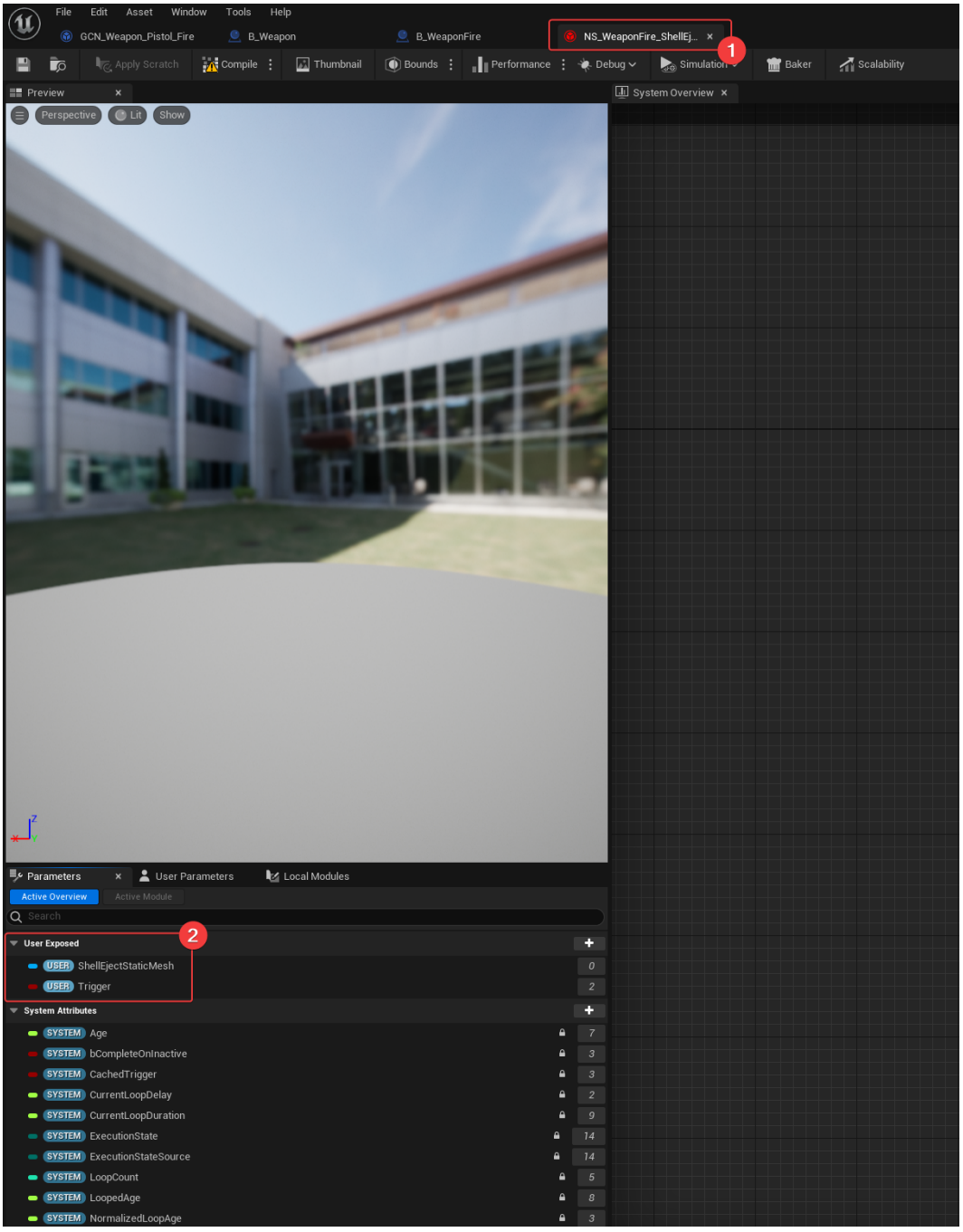
- [ ] 기본 Sequence를 만들자:



☐ B_WeaponFire의 Eject 이펙트:



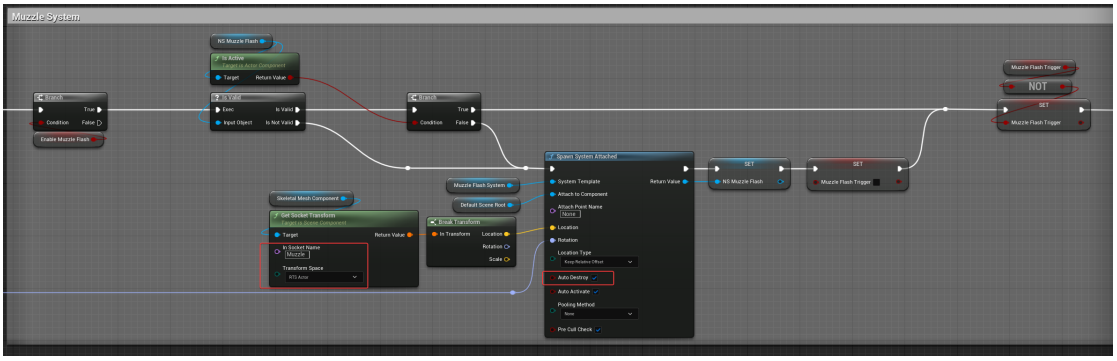
- 참고로 우리가 세팅한 Niagara Variable은 어디서 설정된걸까?

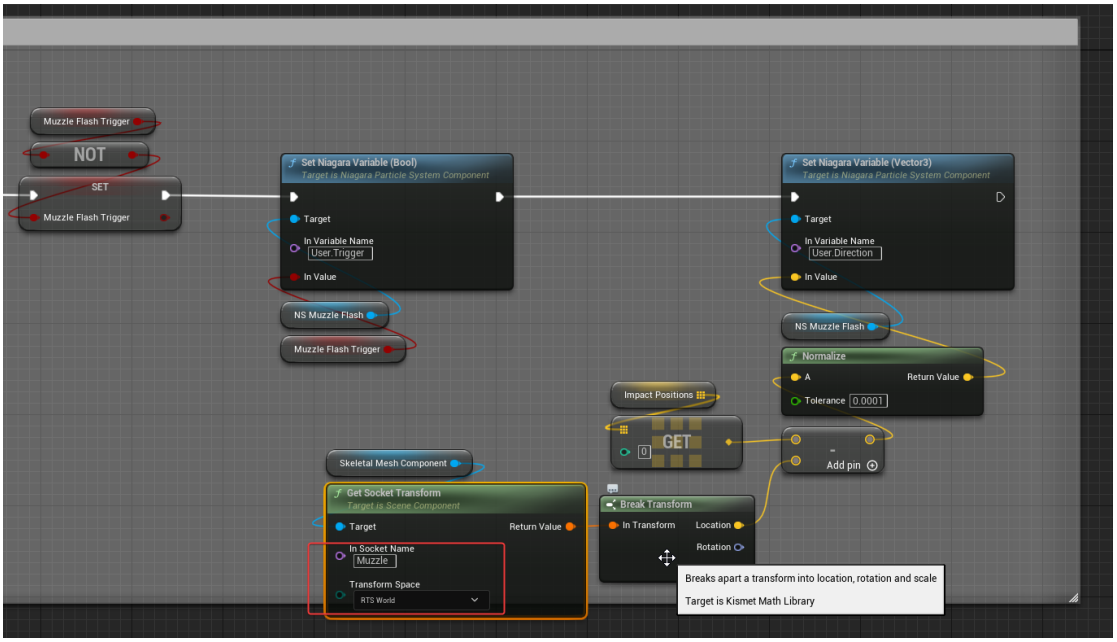


- [ ] 그럼 아래와 같이 탄피가 나오는 효과가 표현된다:

https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/af23ec60-6b7b-400f-9cf4-8197fa3f4aab/UnrealEditor_f50OZrY6yd.mp4

- [ ] 다음은 Muzzle System을 구현하자:



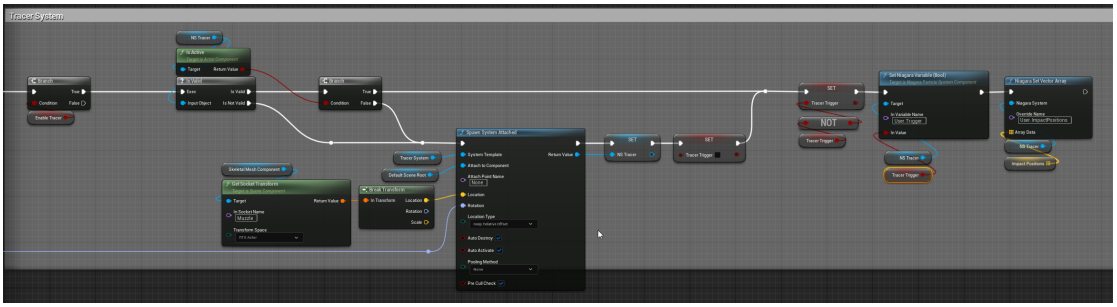


- Impact Position은 World 좌표계이므로, Muzzle Socket 좌표를 가져올 때, Actor가 아닌 World 기반으로 가져와서 Direction을 구한다.

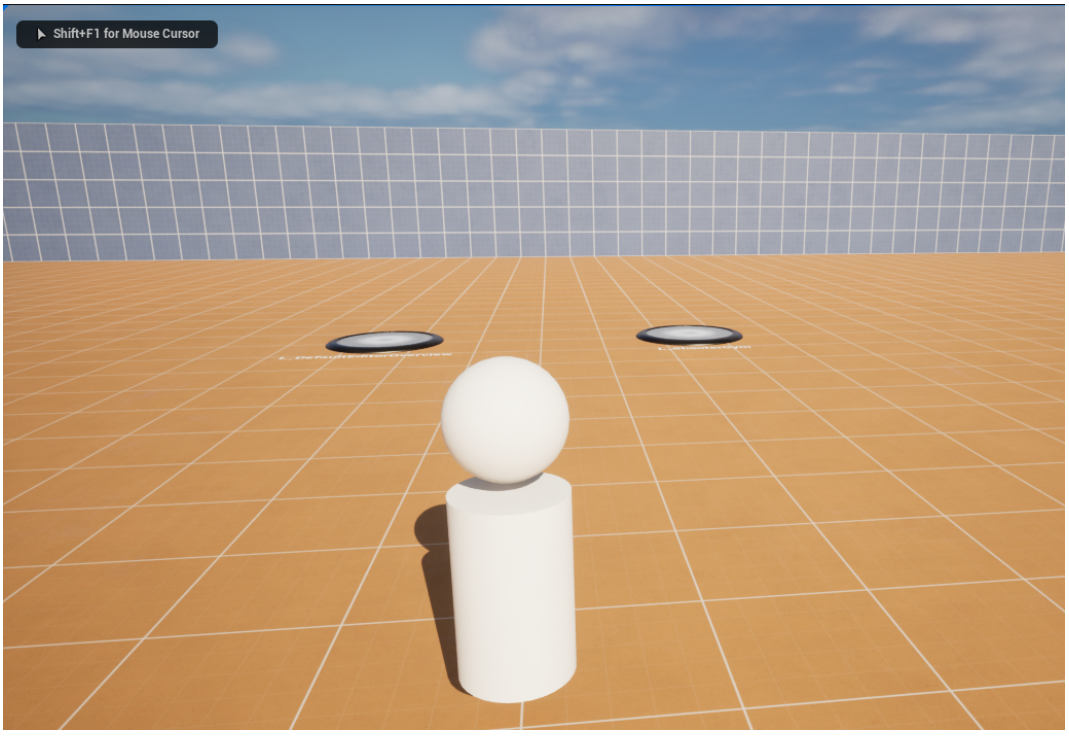
☐ 아래와 같이 Muzzle 효과도 잘 나온다:

https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/21c05c07-6529-4d61-9f5a-46ea66c51c5c/UnrealEditor_NSjXGHMWoC.mp4

☐ Tracer System 구현하자:



- [ ] Tracer가 잘 보이지 않으니, 배경 Platform을 새로운 Material로 바꾸어주자:
  - [ ] MI_MS_Blue_1을 임포트
  - [ ] EditorDefaultOverview 맵과 ShooterGym 맵에 대해 변경해주자:



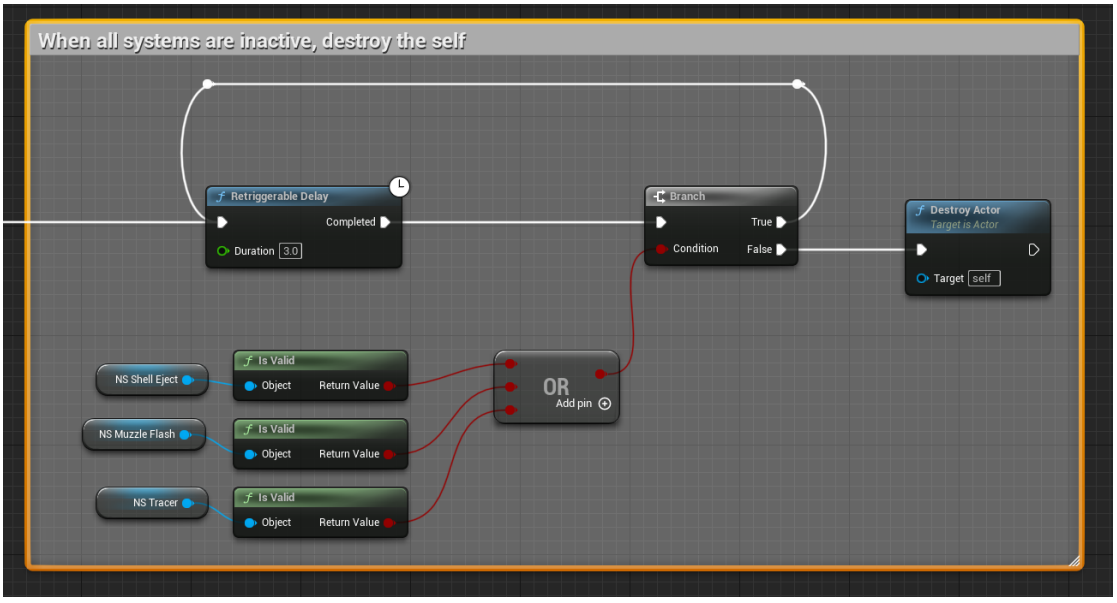
- [ ] 구현된 Tracer System은 아래와 같이 잘 보인다:

https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/0ba3d364-99e3-4e2a-ac34-eae03524e6f5/UnrealEditor_8dzdS7W5p8.mp4

- [ ] 이제 Tracer도 구현되었으니, DebugDraw를 꺼주자:

```cpp
void UHakGameplayAbility_RangedWeapon::PerformLocalTargeting(TArray<FHitResult>& OutHits)
{
    APawn* const AvatarPawn = Cast<APawn>(GetAvatarActorFromActorInfo());

    UHakRangedWeaponInstance* WeaponData = GetWeaponInstance();
    if (AvatarPawn && AvatarPawn->IsLocallyControlled() && WeaponData)
    {
        FRangedWeaponFiringInput InputData;
        InputData.WeaponData = WeaponData;
        InputData.bCanPlayBulletFX = true;

        const FTransform TargetTransform = GetTargetingTransform(AvatarPawn, EHakAbilityTargetingSource::CameraTowardsFocus);
        // 언리얼은 ForwardVector가 (1, 0, 0) 즉 EAxis::X이다
        // - GetUnitAxis()를 살펴보자
        InputData.AimDir = TargetTransform.GetUnitAxis(EAxis::X);
        InputData.StartTrace = TargetTransform.GetTranslation();
        InputData.EndAim = InputData.StartTrace + InputData.AimDir * WeaponData->MaxDamageRange;

#if 0
        {
            static float DebugThickness = 2.0f;
            DrawDebugLine(GetWorld(), InputData.StartTrace, InputData.StartTrace + (InputData.AimDir * 100.0f), FColor::Yellow, false, 10.0f, 0, DebugThickness);
        }
#endif

        TraceBulletsInCartridge(InputData, OutHits);
    }
}
```

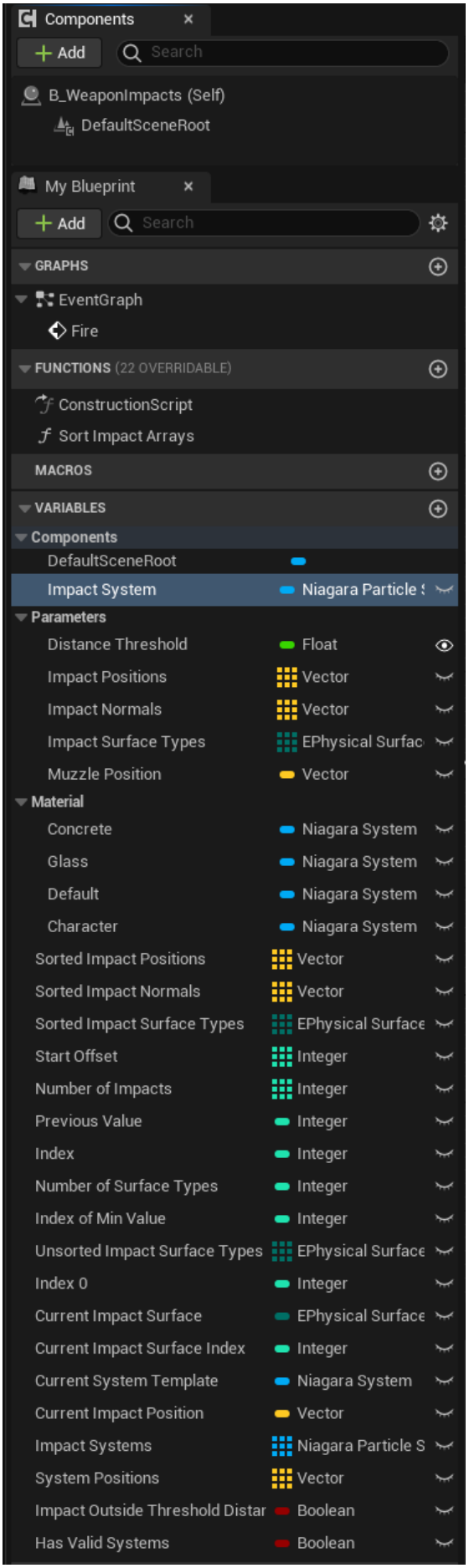
☐ 이제 WeaponFire가 Actor로서 생성한 Niagara Particle System Component가 전부 파괴되었으면, 그때 사라져야 한다:



B_WeaponImpacts - 1

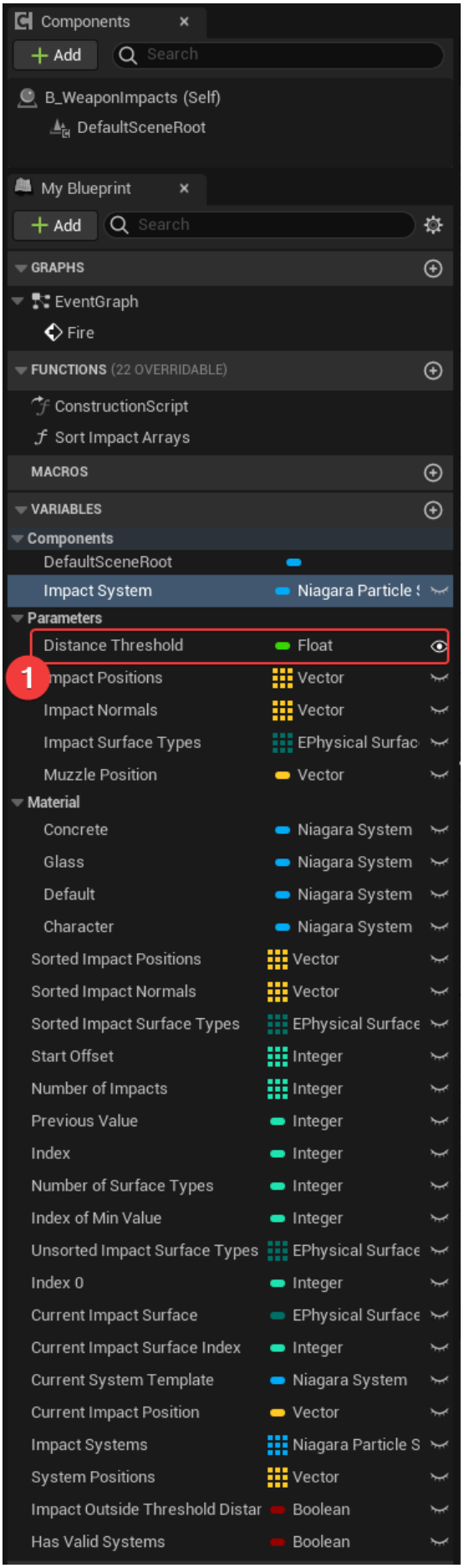
▼ **펼치기**

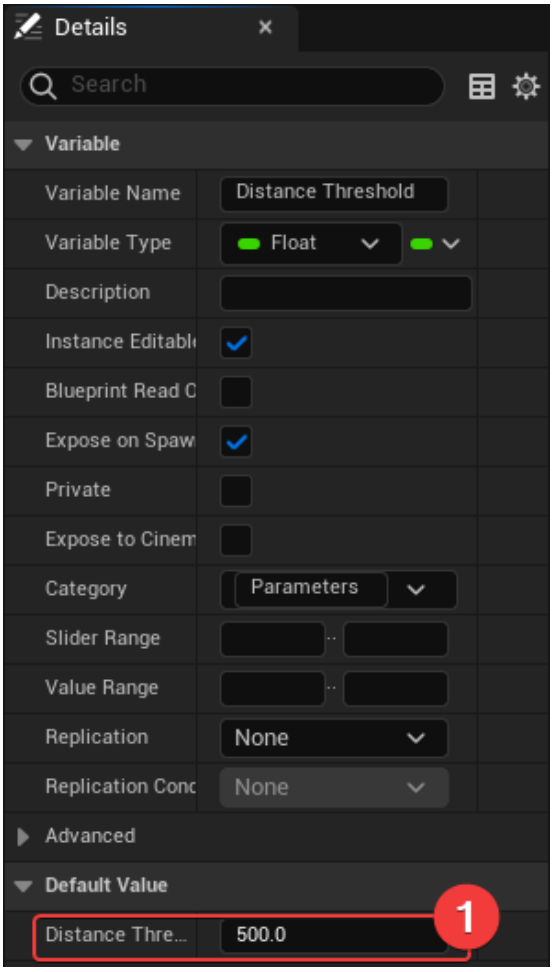
☐ B_Weapon에 WeaponImpacts를 생성하기에 앞서, B_WeaponImpacts 멤버 변수를 정의하자:


Components
Add
Search
B_WeaponImpacts (Self)
DefaultSceneRoot
My Blueprint
Add
Search
GRAPHS
EventGraph
Fire
FUNCTIONS (22 OVERRIDABLE)
ConstructionScript
Sort Impact Arrays
MACROS
VARIABLES
Components
DefaultSceneRoot
Impact System
Niagara Particle S
Parameters
Distance Threshold
Float
Impact Positions
Vector
Impact Normals
Vector
Impact Surface Types
EPhysical Surfac
Muzzle Position
Vector
Material
Concrete
Niagara System
Glass
Niagara System
Default
Niagara System
Character
Niagara System
Sorted Impact Positions
Vector
Sorted Impact Normals
Vector
Sorted Impact Surface Types
EPhysical Surface
Start Offset
Integer
Number of Impacts
Integer
Previous Value
Integer
Index
Integer
Number of Surface Types
Integer
Index of Min Value
Integer
Unsorted Impact Surface Types
EPhysical Surface
Index 0
Integer
Current Impact Surface
EPhysical Surface
Current Impact Surface Index
Integer
Current System Template
Niagara System
Current Impact Position
Vector
Impact Systems
Niagara Particle S
System Positions
Vector
Impact Outside Threshold Distar
Boolean
Has Valid Systems
Boolean

- 현재 로컬 변수가 많지만, 여러분들은 BP 로직을 직접 적어가며 해보길 바랍니다.

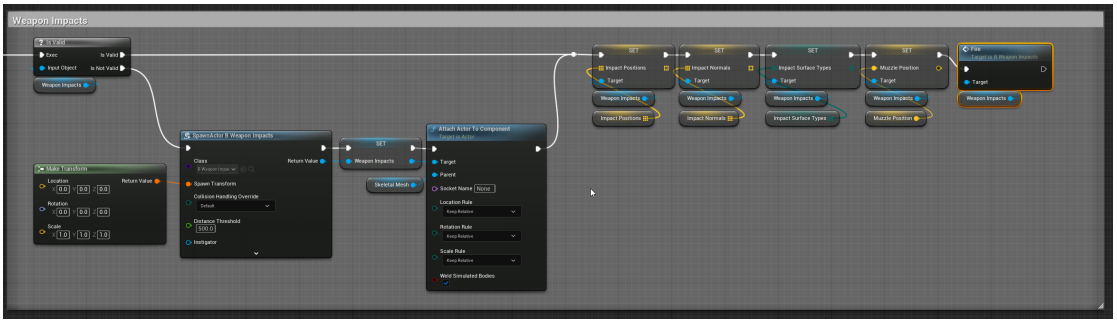
☐ Spawn 변수는 하나만 있다:


Components
Add
Search
B_WeaponImpacts (Self)
DefaultSceneRoot
My Blueprint
Add
Search
GRAPHS
EventGraph
Fire
FUNCTIONS (22 OVERRIDABLE)
ConstructionScript
Sort Impact Arrays
MACROS
VARIABLES
Components
DefaultSceneRoot
Impact System
Niagara Particle S
Parameters
Distance Threshold
Float
1
mpact Positions
Vector
Impact Normals
Vector
Impact Surface Types
EPhysical Surfac
Muzzle Position
Vector
Material
Concrete
Niagara System
Glass
Niagara System
Default
Niagara System
Character
Niagara System
Sorted Impact Positions
Vector
Sorted Impact Normals
Vector
Sorted Impact Surface Types
EPhysical Surface
Start Offset
Integer
Number of Impacts
Integer
Previous Value
Integer
Index
Integer
Number of Surface Types
Integer
Index of Min Value
Integer
Unsorted Impact Surface Types
EPhysical Surface
Index 0
Integer
Current Impact Surface
EPhysical Surface
Current Impact Surface Index
Integer
Current System Template
Niagara System
Current Impact Position
Vector
Impact Systems
Niagara Particle S
System Positions
Vector
Impact Outside Threshold Distar
Boolean
Has Valid Systems
Boolean

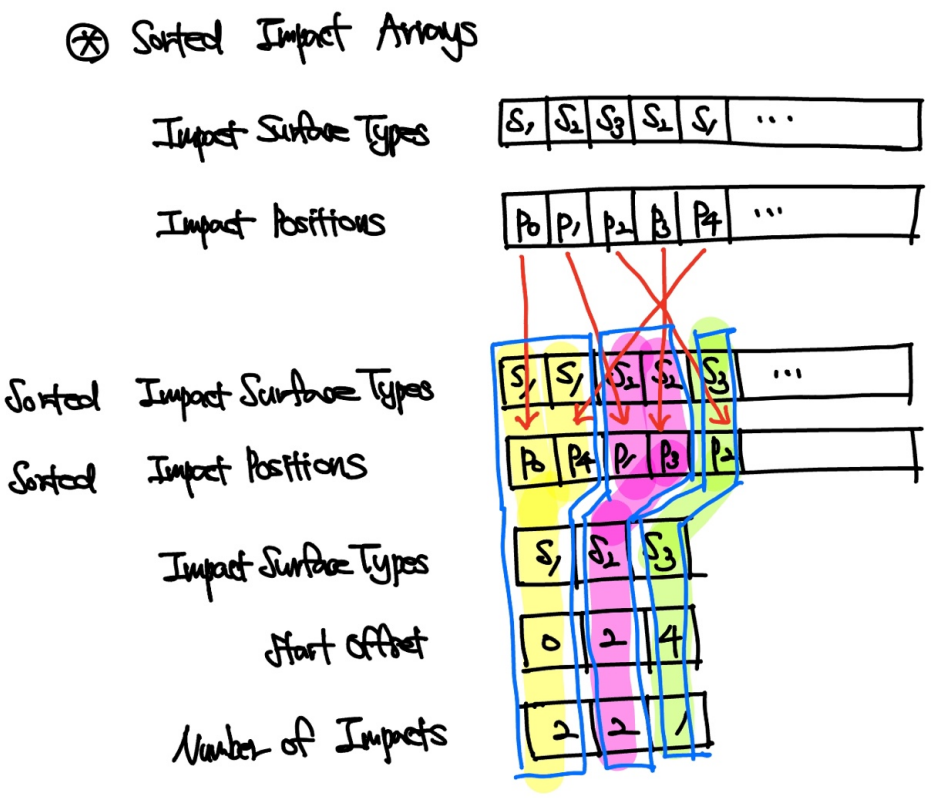


☐ Custom Event Fire() 생성하자:

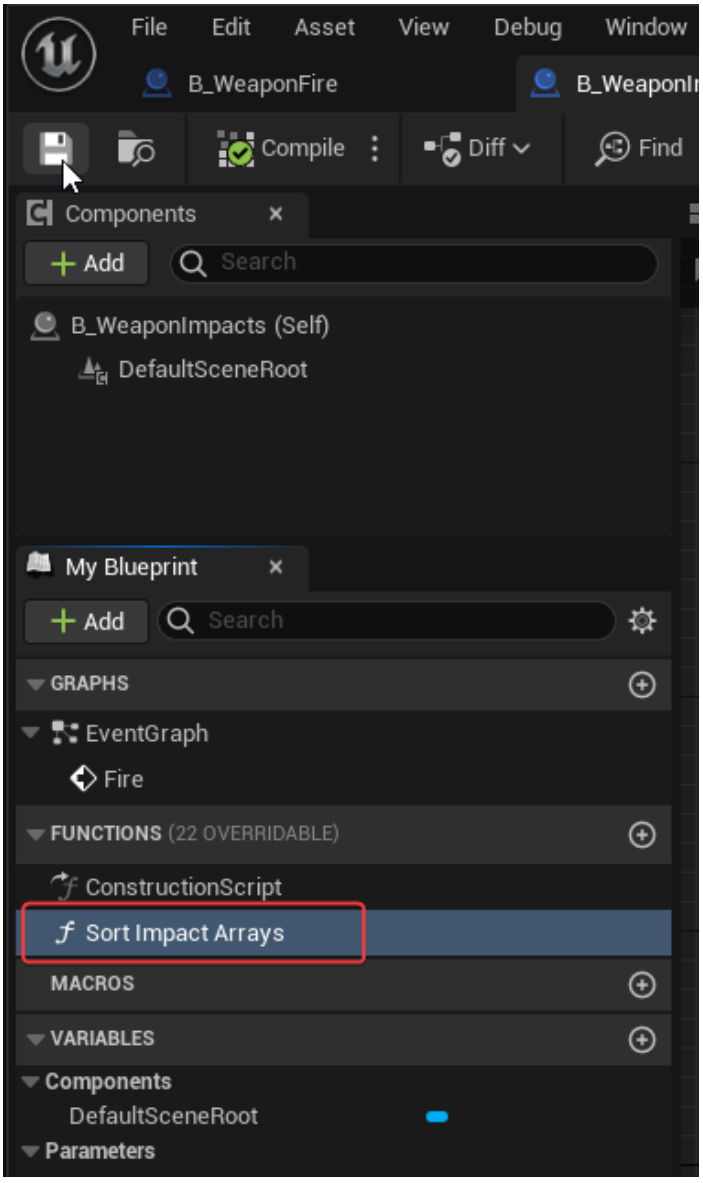
☐ B_Weapon의 Weapon Impact Actor를 생성하자:



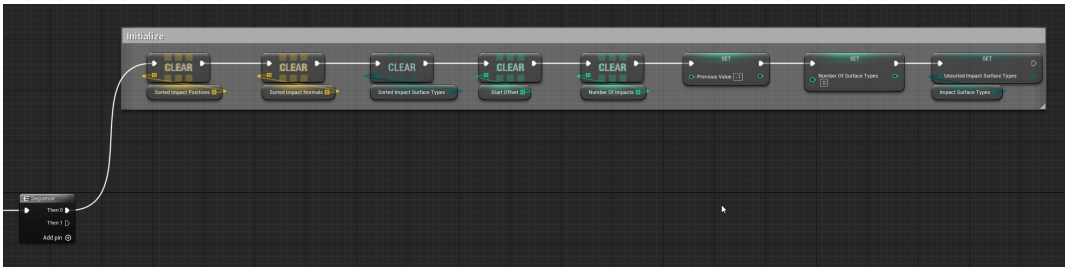
☐ Sort Impact Arrays:



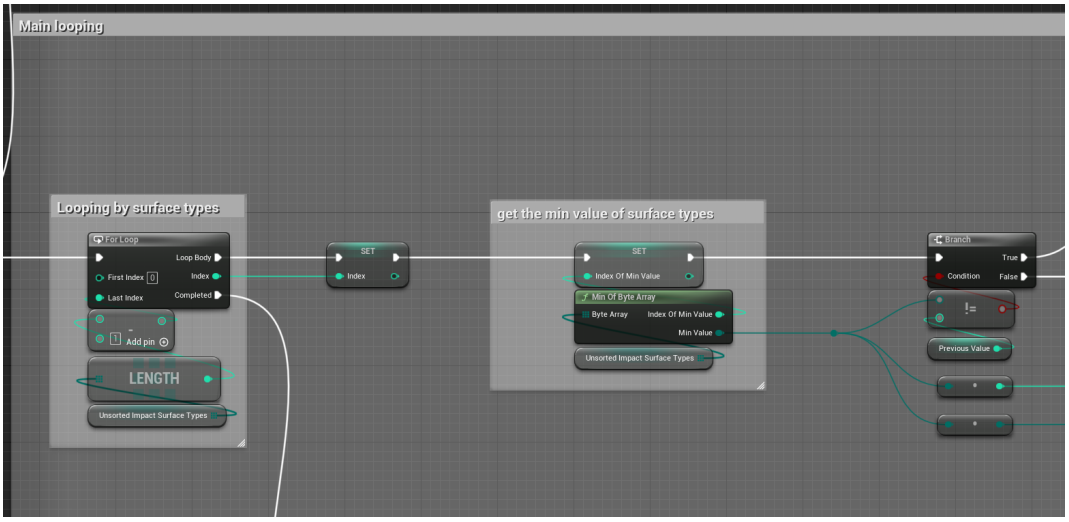
☐ Sort Impact Arrays 함수 추가하자:

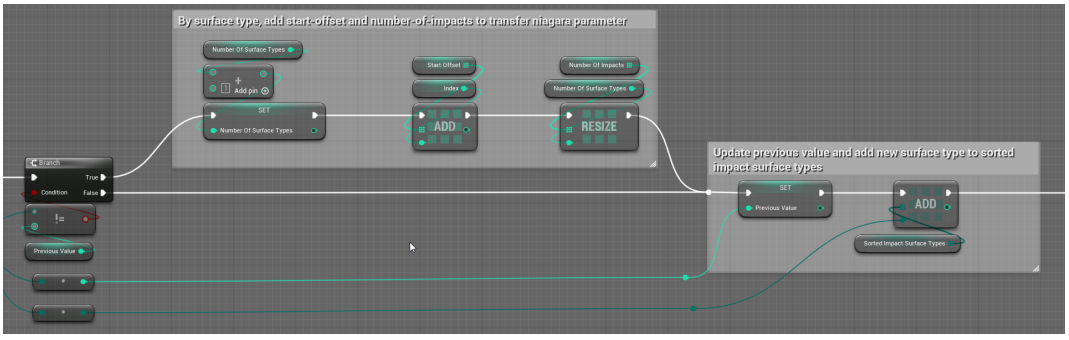


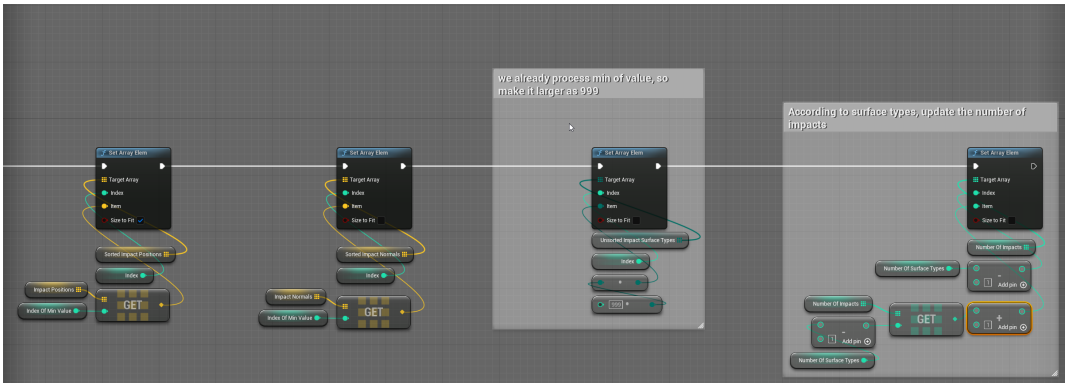
☐ 해당 함수의 초기화 부분을 구현하자:



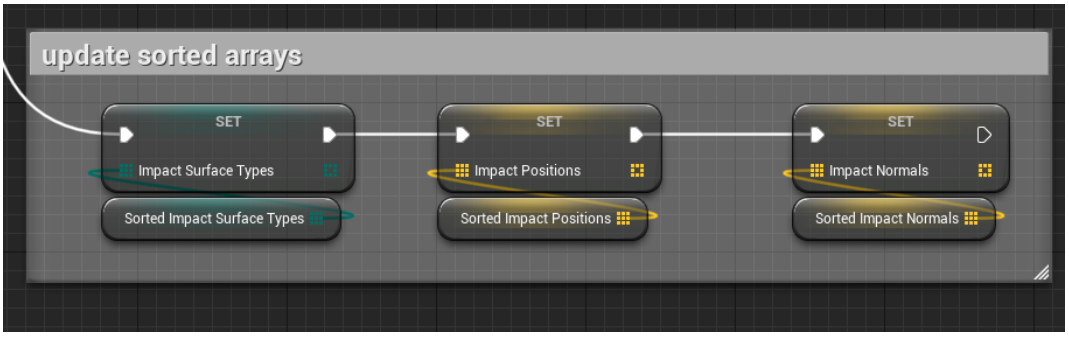
☐ 메인 루프 함수:


Main looping
Looping by surface types
For Loop
Loop Body
First Index
Index
Last Index
Completed
Add pin
LENGTH
Unsorted Impact Surface Types
SET
Index
get the min value of surface types
SET
Index Of Min Value
Min Of Byte Array
Byte Array
Index Of Min Value
Min Value
Unsorted Impact Surface Types
Branch
True
Condition
False
Previous Value


By surface type, add start-offset and number-of-impacts to transfer niagara parameter
Number Of Surface Types
Add pin
SET
Number Of Surface Types
Start Offset
Index
ADD
Number Of Impacts
Number Of Surface Types
RESIZE
Branch
True
Condition
False
Previous Value
Update previous value and add new surface type to sorted impact surface types
SET
Previous Value
ADD
Sorted Impact Surface Types


we already process min of value, so make it larger as 999
According to surface types, update the number of impacts
Set Array Elem
Target Array
Index
Item
Size to Fit
Sorted Impact Positions
Sorted Impact Normals
Unsorted Impact Surface Types
Number Of Impacts
Number Of Surface Types
GET
Add pin

☐ 루프 끝나고:


update sorted arrays
SET
Impact Surface Types
Sorted Impact Surface Types
SET
Impact Positions
Sorted Impact Positions
SET
Impact Normals
Sorted Impact Normals