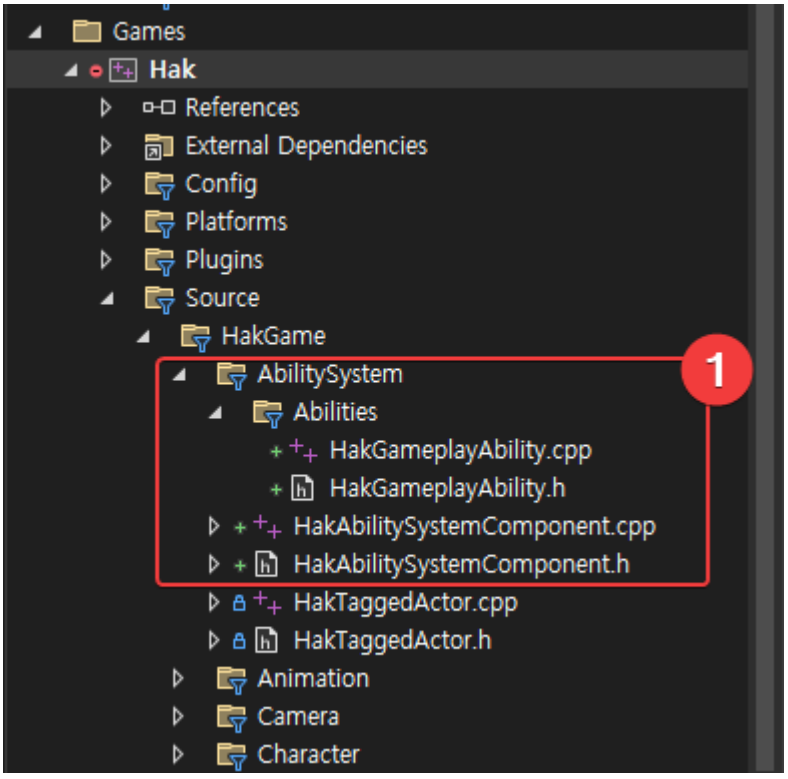
# 12주차

## AbilitySystemComponent:

▼ **펼치기**

- [ ] 소스파일 생성:




- [ ] AbilitySystem/HakAbilitySystemComponent.h/.cpp 생성
- [ ] Abilities/HakGameplayAbility.h/.cpp 생성

- [ ] HakGame.Build.cs에 GameplayAbilities와 GameplayTasks 모듈 추가:

```
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",
        });

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

1

GameplayAbilities 모듈만 추가시, GameplayTasks와의 모듈과의 Dependency가 있어서, 반드시 GameplayTasks를 추가해야 한다.

☐ HakAbilitySystemComponent 기본 구현:

```
#pragma once

#include "AbilitySystemComponent.h"
#include "HakAbilitySystemComponent.generated.h"

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

```
#include "HakAbilitySystemComponent.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakAbilitySystemComponent)

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

}
```

☐ HakGameplayAbility 기본 구현:

```h
#pragma once

#include "Abilities/GameplayAbility.h"
#include "HakGameplayAbility.generated.h"

UCLASS(Abstract)
class UHakGameplayAbility : public UGameplayAbility
{
    GENERATED_BODY()
public:
    UHakGameplayAbility(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
};
```

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

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

}
```

- [ ] HakPlayerState에 HakAbilitySystemComponent를 추가하자:
  - AbilitySystemComponent는 현재 Lyra에서 두 개의 Actor가 소유하고 있다:
    - GameState
    - PlayerState
  - 우리의 GameAbility 적용 대상은 플레이어 관한 것이므로, PlayerState에만 추가하고 후일 필요하다면 GameState에도 추가하자

```h
#pragma once

#include "GameFramework/PlayerState.h"
#include "HakPlayerState.generated.h"

/** forward declaration */
class UHakPawnData;
class UHakExperienceDefinition;
class UHakAbilitySystemComponent;

UCLASS()
class AHakPlayerState : public APlayerState
{
    GENERATED_BODY()
public:
    AHakPlayerState(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * AActor's interface
     */
    virtual void PostInitializeComponents() final;

    /**
     * member methods
     */
    template <class T>
    const T* GetPawnData() const { return Cast<T>(PawnData); }
    void OnExperienceLoaded(const UHakExperienceDefinition* CurrentExperience);
    void SetPawnData(const UHakPawnData* InPawnData);

    UPROPERTY()
    TObjectPtr<const UHakPawnData> PawnData;

    UPROPERTY(VisibleAnywhere, Category="Hak|PlayerState")
    TObjectPtr<UHakAbilitySystemComponent> AbilitySystemComponent;
};
```

1

2

```cpp
#include "HakPlayerState.h"
#include "HakGame/GameModes/HakGameMode.h"
#include "HakGame/GameModes/HakGameState.h"
#include "HakGame/GameModes/HakExperienceManagerComponent.h"
#include "HakGame/GameModes/HakExperienceDefinition.h"
#include "HakGame/Character/HakPawnData.h"
#include "HakGame/AbilitySystem/HakAbilitySystemComponent.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakPlayerState)

PRAGMA_DISABLE_OPTIMIZATION
AHakPlayerState::AHakPlayerState(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
    AbilitySystemComponent = ObjectInitializer.CreateDefaultSubobject<UHakAbilitySystemComponent>(this, TEXT("AbilitySystemComponent"));
}
PRAGMA_ENABLE_OPTIMIZATION
```

1

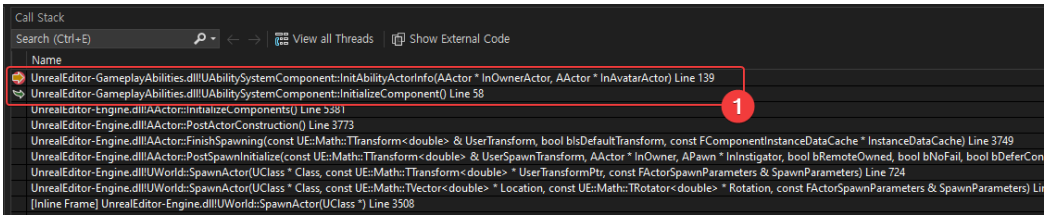
2

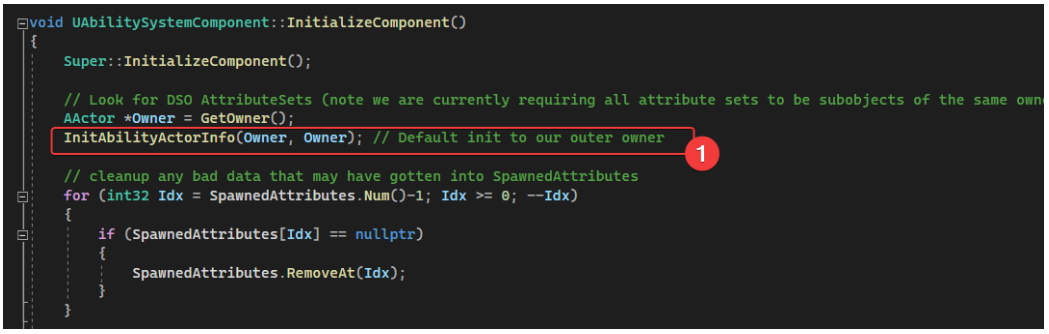
☐ HakPlayerState::PostInitializeComponents() 추가:

```cpp
/**
 * AActor's interface
 */
void AHakPlayerState::PostInitializeComponents()
{
    Super::PostInitializeComponents();

    check(AbilitySystemComponent);
    // 아래의 코드는 우리가 InitAbilityActorInfo를 재호출을 통하는 이유를 설명하는 코드이다:
    {
        // 처음 InitAbilityActorInfo를 호출 당시, OwnerActor와 AvatarActo가 같은 Actor를 가르키고 있으며, 이는 PlayerState이다
        // - OwnerActor는 PlayerState가 의도하는게 맞지만, AvatarActor는 PlayerController가 소유하는 대상인 Pawn이 되어야 한다!
        // - 이를 위해 재-세팅을 해준다
        FGameplayAbilityActorInfo* ActorInfo = AbilitySystemComponent->AbilityActorInfo.Get();
        check(ActorInfo->OwnerActor == this);
        check(ActorInfo->OwnerActor == ActorInfo->AvatarActor);
    }
    AbilitySystemComponent->InitAbilityActorInfo(this, GetPawn());

    AGameStateBase* GameState = GetWorld()->GetGameState();
    check(GameState);

    UHakExperienceManagerComponent* ExperienceManagerComponent = GameState->FindComponentByClass<UHakExperienceManagerComponent>();
    check(ExperienceManagerComponent);

    ExperienceManagerComponent->CallOrRegister_OnExperienceLoaded(FOnHakExperienceLoaded::FDelegate::CreateUObject(this, &ThisClass::OnExperienceLoaded));
}
PRAGMA_ENABLE_OPTIMIZATION
```

1

- 아래와 같이 AbilitySystemComponent::InitializeComponent()에서 AvatarActor를 OwnerActor랑 똑같이 세팅한다:




    - 따라서, AvatarActor도 똑같이 PlayerState로 설정되어 있었다

☐ PawnExtensionComponent::InitializeAbilitySystem()

- 앞서 InitAbilityActorInfo는 GetPawn()에서 nullptr로 아직 Controller가 Pawn을 Possess하기 이전 호출이었다
- Pawn이 Possess된 이후, 제대로 ASC에서 InitAbilityActorInfo()를 호출해야 하는데 이를 PawnExtensionComponent::InitializeAbilitySystem()에서 진행한다

☐ HeroComponent::HandleChangeInitState():

```cpp
void UHakHeroComponent::HandleChangeInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState)
{
    const FHakGameplayTags& InitTags = FHakGameplayTags::Get();

    // DataAvailable -> DataInitialized 단계
    if (CurrentState == InitTags.InitState_DataAvailable && DesiredState == InitTags.InitState_DataInitialized)
    {
        APawn* Pawn = GetPawn<APawn>();
        AHakPlayerState* HakPS = GetPlayerState<AHakPlayerState>();
        if (!ensure(Pawn && HakPS))
        {
            return;
        }

        const bool bIsLocallyControlled = Pawn->IsLocallyControlled();
        const UHakPawnData* PawnData = nullptr;
        if (UHakPawnExtensionComponent* PawnExtComp = UHakPawnExtensionComponent::FindPawnExtensionComponent(Pawn))
        {
            PawnData = PawnExtComp->GetPawnData<UHakPawnData>();

            // DataInitialized 단계까지 오면, Pawn이 Controller에 Possess되어 준비된 상태이다.
            // - InitAbilityActorInfo 호출로 AvatarActor 재설정이 필요하다
            PawnExtComp->InitializeAbilitySystem(HakPS->GetHakAbilitySystemComponent(), HakPS);
        }

        if (bIsLocallyControlled && PawnData)
        {
            // 현재 HakCharacter에 Attach된 CameraComponent를 찾음
            if (UHakCameraComponent* CameraComponent = UHakCameraComponent::FindCameraComponent(Pawn))
            {
                CameraComponent->DetermineCameraModeDelegate.BindUObject(this, &ThisClass::DetermineCameraMode);
            }
        }

        if (AHakPlayerController* HakPC = GetController<AHakPlayerController>())
        {
            if (Pawn->InputComponent != nullptr)
            {
                InitializePlayerInput(Pawn->InputComponent);
            }
        }
    }
}
```

1

☐ GetHakAbilitySystemComponent():

```cpp
UCLASS()
class AHakPlayerState : public APlayerState
{
    GENERATED_BODY()
public:
    AHakPlayerState(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * AActor's interface
     */
    virtual void PostInitializeComponents() final;

    /**
     * member methods
     */
    template <class T>
    const T* GetPawnData() const { return Cast<T>(PawnData); }
    void OnExperienceLoaded(const UHakExperienceDefinition* CurrentExperience);
    void SetPawnData(const UHakPawnData* InPawnData);
    UHakAbilitySystemComponent* GetHakAbilitySystemComponent() const { return AbilitySystemComponent; }

    UPROPERTY()
    TObjectPtr<const UHakPawnData> PawnData;

    UPROPERTY(VisibleAnywhere, Category="Hak|PlayerState")
    TObjectPtr<UHakAbilitySystemComponent> AbilitySystemComponent;
};
```

1

☐ HakPawnExtensionComponent::InitializeAbilitySystem()

```cpp
#pragma once

#include "Components/GameFrameworkInitStateInterface.h"
#include "Components/PawnComponent.h"
#include "GameFramework/Actor.h"
#include "HakPawnExtensionComponent.generated.h"

/** forward declaration */
class UHakPawnData;
class UHakAbilitySystemComponent;

/**
 * 초기화 전반을 조정하는 컴포넌트
 */

UCLASS()
class UHakPawnExtensionComponent : public UPawnComponent, public IGameFrameworkInitStateInterface
{
    GENERATED_BODY()
public:
    UHakPawnExtensionComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** FeatureName 정의 */
    static const FName NAME_ActorFeatureName;

    /**
     * member methods
     */
    static UHakPawnExtensionComponent* FindPawnExtensionComponent(const AActor* Actor) { return (Actor ? Actor->FindComponentByClass<UHakPawnExtensionComponent>() : nullptr); }
    template <class T>
    const T* GetPawnData() const { return Cast<T>(PawnData); }
    void SetPawnData(const UHakPawnData* InPawnData);
    void SetupPlayerInputComponent();
    /** AbilitySystemComponent의 AvatorActor 대상 초기화/해제 호출 */
    void InitializeAbilitySystem(UHakAbilitySystemComponent* InASC, AActor* InOwnerActor);
    void UninitializeAbilitySystem();

    /**
     * UPawnComponent interfaces
     */
    virtual void OnRegister() final;
    virtual void BeginPlay() final;
    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) final;

    /**
     * IGameFrameworkInitStateInterface
     */
    virtual FName GetFeatureName() const final { return NAME_ActorFeatureName; }
    virtual void OnActorInitStateChanged(const FActorInitStateChangedParams& Params) final;
    virtual bool CanChangeInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState) const final;
    virtual void CheckDefaultInitialization() final;

    /**
     * Pawn을 생성한 데이터를 캐싱
     */
    UPROPERTY(EditInstanceOnly, Category = "Hak|Pawn")
    TObjectPtr<const UHakPawnData> PawnData;

    /** AbilitySystemComponent 캐싱 */
    UPROPERTY()
    TObjectPtr<UHakAbilitySystemComponent> AbilitySystemComponent;
};
```

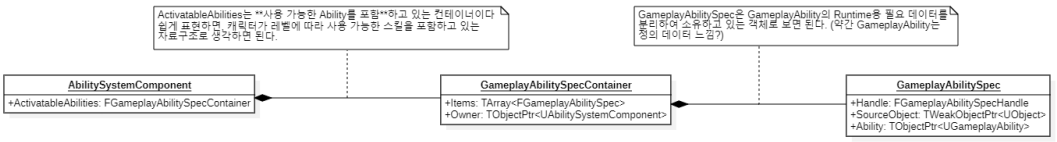
- [ ] HakPawnExtensionComponent::UninitializeAbilitySystem()

```cpp
void UHakPawnExtensionComponent::UninitializeAbilitySystem()
{
    if (!AbilitySystemComponent)
    {
        return;
    }

    AbilitySystemComponent = nullptr;
}
```

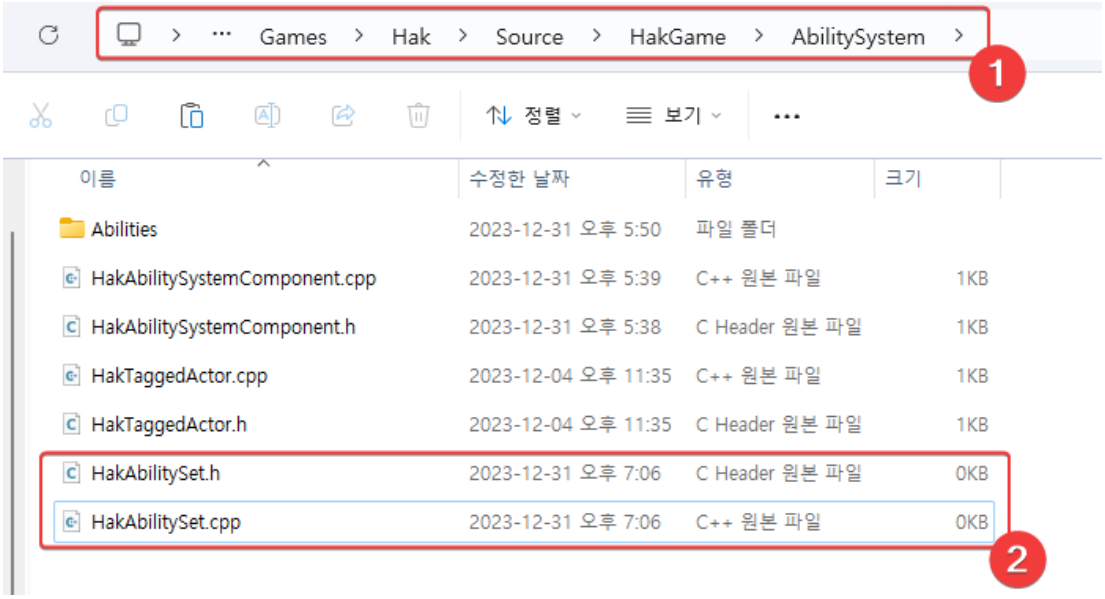
- [ ] InitializeAbilitySystem() 마저 구현:

```cpp
void UHakPawnExtensionComponent::InitializeAbilitySystem(UHakAbilitySystemComponent* InASC, AActor* InOwnerActor)
{
    check(InASC && InOwnerActor);

    if (AbilitySystemComponent == InASC)
    {
        return;
    }

    if (AbilitySystemComponent)
    {
        UninitializeAbilitySystem();
    }

    APawn* Pawn = GetPawnChecked<APawn>();
    AActor* ExistingAvatar = InASC->GetAvatarActor();
    check(!ExistingAvatar);

    // ASC를 업데이트하고, InitAbilityActorInfo를 Pawn과 같이 호출하여, AvatarActor를 Pawn으로 업데이트 해준다
    AbilitySystemComponent = InASC;
    AbilitySystemComponent->InitAbilityActorInfo(InOwnerActor, Pawn);
}
```

☐ ActivatableAbilities 이해:




☐ AbilitySystem/HakAbilitySet.h/.cpp 추가 및 간단한 구조 구현:



```h
#pragma once

#include "Containers/Array.h"
#include "Engine/DataAsset.h"
#include "GameplayTagContainer.h"
#include "GameplayAbilitySpec.h"
#include "Templates/SubclassOf.h"
#include "UObject/ObjectPtr.h"
#include "UObject/UObjectGlobals.h"

#include "HakAbilitySet.generated.h"

/** forward declarations */
class UHakGameplayAbility;
class UHakAbilitySystemComponent;

/**
 * GameplayAbility의 Wrapper 클래스
 * - 추가적인 커스터마이징이 가능함
 */
USTRUCT(BlueprintType)
struct FHakAbilitySet_GameplayAbility
{
    GENERATED_BODY()
public:
    /** 허용된 GameplayAbility */
    UPROPERTY(EditDefaultsOnly)
    TSubclassOf<UHakGameplayAbility> Ability = nullptr;

    /** Input 처리를 위한 GameplayTag */
    UPROPERTY(EditDefaultsOnly)
    FGameplayTag InputTag;

    /** Ability의 허용 조건 (Level) */
    UPROPERTY(EditDefaultsOnly)
    int32 AbilityLevel = 1;
};

/** Gameplay Ability를 좀 더 쉽게 관리하기 위한 Set */
UCLASS(BlueprintType)
class UHakAbilitySet : public UPrimaryDataAsset
{
    GENERATED_BODY()
public:
    UHakAbilitySet(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** 허용된 GameplayAbilities */
    UPROPERTY(EditDefaultsOnly, Category="Gameplay Abilities")
    TArray<FHakAbilitySet_GameplayAbility> GrantedGameplayAbilities;
};
```

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

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

- [ ] PawnData에 AbilitySet을 추가하자:

```cpp
#pragma once

#include "CoreMinimal.h"
#include "Engine/DataAsset.h"
#include "HakGame/Camera/HakCameraMode.h"
#include "HakGame/Input/HakInputConfig.h"
#include "HakPawnData.generated.h"

/** forward declarations */
class UHakAbilitySet;

/**
 * UHakPawnData
 * - non-mutable data asset that contains properties used to define a pawn
 */
UCLASS(BlueprintType)
class UHakPawnData : public UPrimaryDataAsset
{
    GENERATED_BODY()
public:
    UHakPawnData(const FObjectInitializer& ObjectInitializer);

    /** @TODO - 일단 단순히 클래스의 형태만 만들어놓도록 하자 */

    /** Pawn의 Class */
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Hak|Pawn")
    TSubclassOf<APawn> PawnClass;

    /** Camera Mode */
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Hak|Camera")
    TSubclassOf<UHakCameraMode> DefaultCameraMode;

    /** input configuration used by player controlled pawns to create input mappings and bind input actions */
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Hak|InputConfig")
    TObjectPtr<UHakInputConfig> InputConfig;

    /** 해당 Pawn의 Ability System에 허용할 AbilitySet */
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Hak|Abilities")
    TArray<TObjectPtr<UHakAbilitySet>> AbilitySets;
};
```

1

- Pawn에 허용한 AbilitySet이 지정 가능함을 의미한다

☐ PawnData의 AbilitySet을 적용해보자:

- 앞서 보았듯이 ASC(AbilitySystemComponent)에 허용가능한 Ability를 할당하기 위해, ActivatableAbilities에 추가해야 한다

☐ HakPlayerState::SetPawnData()

```cpp
/**
 * member methods
 */
void AHakPlayerState::SetPawnData(const UHakPawnData* InPawnData)
{
    check(InPawnData);

    // PawnData가 두번 설정되는 것은 원하지 않음!
    check(!PawnData);

    PawnData = InPawnData;

    // PawnData의 AbilitySet을 순회하며, ASC에 Ability를 할당(Give)한다
    // - 이 과정에서 ASC의 ActivatableAbilities에 추가된다
    for (UHakAbilitySet* AbilitySet : PawnData->AbilitySets)
    {
        if (AbilitySet)
        {
            AbilitySet->GiveToAbilitySystem(AbilitySystemComponent, nullptr);
        }
    }
}
```

## ☐ HakAbilitySet::GiveToAbilitySystem()

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

protected:
    /** 허용된 GameplayAbilitySpecHandle(int32) */
    UPROPERTY()
    TArray<FGameplayAbilitySpecHandle> AbilitySpecHandles;
};

/** Gameplay Ability를 좀 더 쉽게 관리하기 위한 Set */
UCLASS(BlueprintType)
class UHakAbilitySet : public UPrimaryDataAsset
{
    GENERATED_BODY()
public:
    UHakAbilitySet(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** ASC에 허용가능한 Ability를 추가한다 */
    void GiveToAbilitySystem(UHakAbilitySystemComponent* HakASC, FHakAbilitySet_GrantedHandles* OutGrantedHandles, UObject* SourceObject = nullptr);

    /** 허용된 GameplayAbilities */
    UPROPERTY(EditDefaultsOnly, Category="Gameplay Abilities")
    TArray<FHakAbilitySet_GameplayAbility> GrantedGameplayAbilities;
};
```

```cpp
#include "HakAbilitySet.h"
#include "HakGame/AbilitySystem/HakAbilitySystemComponent.h"
#include "HakGame/AbilitySystem/Abilities/HakGameplayAbility.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakAbilitySet)

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

void UHakAbilitySet::GiveToAbilitySystem(UHakAbilitySystemComponent* HakASC, FHakAbilitySet_GrantedHandles* OutGrantedHandles, UObject* SourceObject)
{
    check(HakASC);

    if (!HakASC->IsOwnerActorAuthoritative())
    {
        return;
    }

    // gameplay abilities를 허용:
    for (int32 AbilityIndex = 0; AbilityIndex < GrantedGameplayAbilities.Num(); ++AbilityIndex)
    {
        const FHakAbilitySet_GameplayAbility& AbilityToGrant = GrantedGameplayAbilities[AbilityIndex];
        if (!IsValid(AbilityToGrant.Ability))
        {
            continue;
        }

        UHakGameplayAbility* AbilityCDO = AbilityToGrant.Ability->GetDefaultObject<UHakGameplayAbility>();

        FGameplayAbilitySpec AbilitySpec(AbilityCDO, AbilityToGrant.AbilityLevel);
        AbilitySpec.SourceObject = SourceObject;
        AbilitySpec.DynamicAbilityTags.AddTag(AbilityToGrant.InputTag);

        const FGameplayAbilitySpecHandle AbilitySpecHandle = HakASC->GiveAbility(AbilitySpec);
        if (OutGrantedHandles)
        {
            OutGrantedHandles->AddAbilitySpecHandle(AbilitySpecHandle);
        }
    }
}
```

## ☐ UAbilitySystemComponent::GiveAbility()

```cpp
FGameplayAbilitySpecHandle UAbilitySystemComponent::GiveAbility(const FGameplayAbilitySpec& Spec)
{
    if (!IsValid(Spec.Ability))
    {
        ABILITY_LOG(Error, TEXT("GiveAbility called with an invalid Ability Class."));

        return FGameplayAbilitySpecHandle();
    }

    if (!IsOwnerActorAuthoritative())
    {
        ABILITY_LOG(Error, TEXT("GiveAbility called on ability %s on the client, not allowed!"), *Spec.Ability->GetName());

        return FGameplayAbilitySpecHandle();
    }

    // If locked, add to pending list. The Spec.Handle is not regenerated when we receive, so returning this is ok.
    if (AbilityScopeLockCount > 0)
    {
        AbilityPendingAdds.Add(Spec);
        return Spec.Handle;
    }

    ABILITYLIST_SCOPE_LOCK();

    // 여기에서 ActivatableAbilities.Items에 새로 추가되어, 허용가능한 Ability로 등재됨을 확인할 수 있다
    FGameplayAbilitySpec& OwnedSpec = ActivatableAbilities.Items[ActivatableAbilities.Items.Add(Spec)];

    if (OwnedSpec.Ability->GetInstancingPolicy() == EGameplayAbilityInstancingPolicy::InstancedPerActor)
    {
        // Create the instance at creation time
        CreateNewInstanceOfAbility(OwnedSpec, Spec.Ability);
    }

    OnGiveAbility(OwnedSpec);
    MarkAbilitySpecDirty(OwnedSpec, true);

    return OwnedSpec.Handle;
}
```

- [ ] HakAbilitySet_GrantedHandles::AddAbilitySpecHandle()

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

    void AddAbilitySpecHandle(const FGameplayAbilitySpecHandle& Handle);

protected:
    /** 허용된 GameplayAbilitySpecHandle(int32) */
    UPROPERTY()
    TArray<FGameplayAbilitySpecHandle> AbilitySpecHandles;
};
```

```cpp
void FHakAbilitySet_GrantedHandles::AddAbilitySpecHandle(const FGameplayAbilitySpecHandle& Handle)
{
    if (Handle.IsValid())
    {
        AbilitySpecHandles.Add(Handle);
    }
}
```

---

💡 아직 우린 PawnData에 GameplayAbility를 넣어 사용하지는 않지만, 전체적인 이해 측면에서 진행하였다. 빠른 시간 내에 해당 기능을 활용하여, GameplayAbility를 추가할 것이라고 생각한다.

# Our Goal:

▼ **펼치기**

- 우리는 이와 같이 총을 쏘는 모션을 GameplayAbility로 Lyra를 Clone해볼 것이다:

  https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/c35911cc-cce7-4251-bd60-5e4fb1545fa4/Untitled.mp4

- 다음 강의에서 총의 머즐(Muzzle) 효과와 데칼(Decal) 효과는 이어서 진행할 예정이다

# GiveToAbilitySystem:

▼ **펼치기**

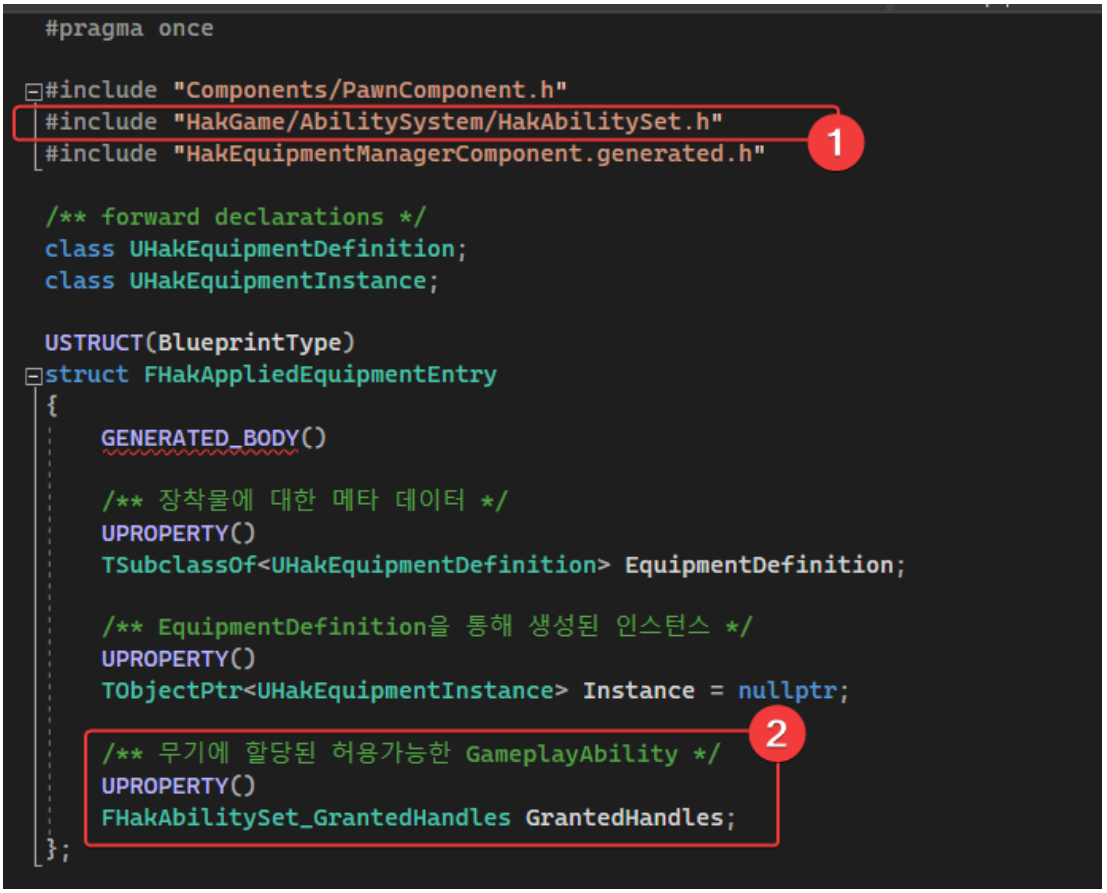
- 우선 Lyra의 경우, 기존 무기에 애니메이션을 가지고 있는 형태와 일관되게 GameplayAbility도 총의 격발의 경우, 무기에서 시작된다:
  - 이는 좀 나아가면, 장착물(Equipment)에서 시작된다
  - 정리하면, **장착물(Equipement)에 AbilitySet이 할당됨을 의미한다**

---

☐ HakAbilitySet을 HakEquipementDefinition에 추가하자:

```
/**
 * HakEquipementDefinition은 장착 아이템에 대한 정의 클래스(메타 데이터)이다
 */
UCLASS(BlueprintType, Blueprintable)
class UHakEquipmentDefinition : public UObject
{
    GENERATED_BODY()
public:
    UHakEquipmentDefinition(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** 해당 메타 데이터를 사용하면, 어떤 인스턴스를 Spawn할지 결정하는 클래스 */
    UPROPERTY(EditDefaultsOnly, Category=Equipment)
    TSubclassOf<UHakEquipmentInstance> InstanceType;

    /** 해당 장착 아이템을 사용하면, 어떤 Actor가 Spawn이 되는지 정보를 담고 있다 */
    UPROPERTY(EditDefaultsOnly, Category=Equipment)
    TArray<FHakEquipmentActorToSpawn> ActorsToSpawn;

    /** 장착을 통해 부여 가능한 Ability Set */
    UPROPERTY(EditDefaultsOnly, Category=Equipment)
    TArray<TObjectPtr<UHakAbilitySet>> AbilitySetsToGrant;
};
```

☐ HakAbilitySet_GrantedHandles을 HakAppliedEquipementEntry에 추가하자:




- 여기서 우린 HakAbilitySet_GrantedHandles을 정의한 이유를 찾을 수 있다:
  - 해당 객체는 GameAbilitySpec의 Handle(ID) 값을 가지고 있으며, 장착물(Equipement)와 같은 오브젝트에 **GameplayAbility의 간접 링크를 제공**한다.

☐ AddEntry() 추가 구현을 통해, ASC에 Ability를 부여하자:

```cpp
UHakEquipmentInstance* FHakEquipmentList::AddEntry(TSubclassOf<UHakEquipmentDefinition> EquipmentDefinition)
{
    UHakEquipmentInstance* Result = nullptr;
    check(EquipmentDefinition != nullptr);
    check(OwnerComponent);
    check(OwnerComponent->GetOwner()->HasAuthority());

    // EquipmentDefinition의 멤버 변수들은 EditDefaultsOnly로 정의되어 있어 GetDefault로 들고 와도 우리에게 필요한 것들이 모두 들어있다
    const UHakEquipmentDefinition* EquipmentCDO = GetDefault<UHakEquipmentDefinition>(EquipmentDefinition);

    TSubclassOf<UHakEquipmentInstance> InstanceType = EquipmentCDO->InstanceType;
    if (!InstanceType)
    {
        InstanceType = UHakEquipmentInstance::StaticClass();
    }

    // Entries에 추가해주자
    FHakAppliedEquipmentEntry& NewEntry = Entries.AddDefaulted_GetRef();
    NewEntry.EquipmentDefinition = EquipmentDefinition;
    NewEntry.Instance = NewObject<UHakEquipmentInstance>(OwnerComponent->GetOwner(), InstanceType);
    Result = NewEntry.Instance;

    UHakAbilitySystemComponent* ASC = GetAbilitySystemComponent();
    check(ASC);
    {
        for (TObjectPtr<UHakAbilitySet> AbilitySet : EquipmentCDO->AbilitySetsToGrant)
        {
            AbilitySet->GiveToAbilitySystem(ASC, &NewEntry.GrantedHandles, Result);
        }
    }

    // ActorsToSpawn을 통해, Actor들을 인스턴스화 해주자
    // - 어디에? EquipmentInstance에!
    Result->SpawnEquipmentActors(EquipmentCDO->ActorsToSpawn);

    return Result;
}
```

☐ HakEquipmentList::GetAbilitySystemComponent() 구현:

```cpp
/**
 * 참고로 EquipmentInstance의 인스턴스를 Entry에서 관리하고 있다:
 * - HakEquipmentList는 생성된 객체를 관리한다고 보면 된다
 */
USTRUCT(BlueprintType)
struct FHakEquipmentList
{
    GENERATED_BODY()

    FHakEquipmentList(UActorComponent* InOwnerComponent = nullptr)
        : OwnerComponent(InOwnerComponent)
    {}

    UHakEquipmentInstance* AddEntry(TSubclassOf<UHakEquipmentDefinition> EquipmentDefinition);
    void RemoveEntry(UHakEquipmentInstance* Instance);

    UHakAbilitySystemComponent* GetAbilitySystemComponent() const;

    /** 장착물에 대한 관리 리스트 */
    UPROPERTY()
    TArray<FHakAppliedEquipmentEntry> Entries;

    UPROPERTY()
    TObjectPtr<UActorComponent> OwnerComponent;
};
```

```cpp
UHakAbilitySystemComponent* FHakEquipmentList::GetAbilitySystemComponent() const
{
    check(OwnerComponent);
    AActor* OwningActor = OwnerComponent->GetOwner();

    // GetAbilitySystemComponentFromActor를 잠시 확인해보자:
    // - EquipmentManagerComponent는 AHakCharacter를 Owner로 가지고 있다
    // - 해당 함수는 IAbilitySystemInterface를 통해 AbilitySystemComponent를 반환한다
    // - 우리는 HakCharacter에 IAbilitySystemInterface를 상속받을 필요가 있다
    return Cast<UHakAbilitySystemComponent>(UAbilitySystemGlobals::GetAbilitySystemComponentFromActor(OwningActor));
}
```

☐ HakCharacter에 IAbilitySystemInterface를 상속받아 필요한 메서드 정의하자:

```
UCLASS()
class AHakCharacter : public AModularCharacter, public IAbilitySystemInterface
{
    GENERATED_BODY()
public:
    AHakCharacter(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * ACharacter interfaces
     */
    virtual void SetupPlayerInputComponent(UInputComponent* PlayerInputComponent) final;

    /**
     * IAbilitySystemInterface
     */
    virtual UAbilitySystemComponent* GetAbilitySystemComponent() const override;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category="Hak|Character")
    TObjectPtr<UHakPawnExtensionComponent> PawnExtComponent;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category="Hak|Character")
    TObjectPtr<UHakCameraComponent> CameraComponent;
};
```

```
UAbilitySystemComponent* AHakCharacter::GetAbilitySystemComponent() const
{
    // 앞서, 우리는 PawnExtensionComponent에 AbilitySystemComponent를 캐싱하였다
    return PawnExtComponent->GetHakAbilitySystemComponent();
}
```

☐ PawnExtensionComponent::GetHakAbilitySystemComponent()

```
UCLASS()
class UHakPawnExtensionComponent : public UPawnComponent, public IGameFrameworkInitStateInterface
{
    GENERATED_BODY()
public:
    UHakPawnExtensionComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** FeatureName 정의 */
    static const FName NAME_ActorFeatureName;

    /**
     * member methods
     */
    static UHakPawnExtensionComponent* FindPawnExtensionComponent(const AActor* Actor) { return (Actor ? Actor->FindComponentByClass<UHakPawnExtensionComponent>() : nullptr); }
    template <class T>
    const T* GetPawnData() const { return Cast<T>(PawnData); }
    void SetPawnData(const UHakPawnData* InPawnData);
    void SetupPlayerInputComponent();
    UHakAbilitySystemComponent* GetHakAbilitySystemComponent() const { return AbilitySystemComponent; }
    /** AbilitySystemComponent의 AvatorActor 내장 초기화/해제 호출 */
    void InitializeAbilitySystem(UHakAbilitySystemComponent* InASC, AActor* InOwnerActor);
    void UninitializeAbilitySystem();
```

☐ RemoveEntry()도 구현해주자:

```cpp
void FHakEquipmentList::RemoveEntry(UHakEquipmentInstance* Instance)
{
    // 단순히 그냥 Entries를 순회하며, Instance를 찾아서
    for (auto EntryIt = Entries.CreateIterator(); EntryIt; ++EntryIt)
    {
        FHakAppliedEquipmentEntry& Entry = *EntryIt;
        if (Entry.Instance == Instance)
        {
            UHakAbilitySystemComponent* ASC = GetAbilitySystemComponent();
            check(ASC);
            {
                // TakeFromAbilitySystem은 GiveToAbilitySystem 반대 역활로, ActivatableAbilities에서 제거한다
                Entry.GrantedHandles.TakeFromAbilitySystem(ASC);
            }

            // Actor 제거 작업 및 iterator를 통한 안전하게 Array에서 제거 진행
            Instance->DestroyEquipmentActors();
            EntryIt.RemoveCurrent();
        }
    }
}
```

- [ ] HakAbilitySet_GrantedHandles::TakeFromAbilitySystem()

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

    void AddAbilitySpecHandle(const FGameplayAbilitySpecHandle& Handle);
    void TakeFromAbilitySystem(UHakAbilitySystemComponent* HakASC);

protected:
    /** 허용된 GameplayAbilitySpecHandle(int32) */
    UPROPERTY()
    TArray<FGameplayAbilitySpecHandle> AbilitySpecHandles;
};
```

1


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

    void AddAbilitySpecHandle(const FGameplayAbilitySpecHandle& Handle);
    void TakeFromAbilitySystem(UHakAbilitySystemComponent* HakASC);

protected:
    /** 허용된 GameplayAbilitySpecHandle(int32) */
    UPROPERTY()
    TArray<FGameplayAbilitySpecHandle> AbilitySpecHandles;
};
```

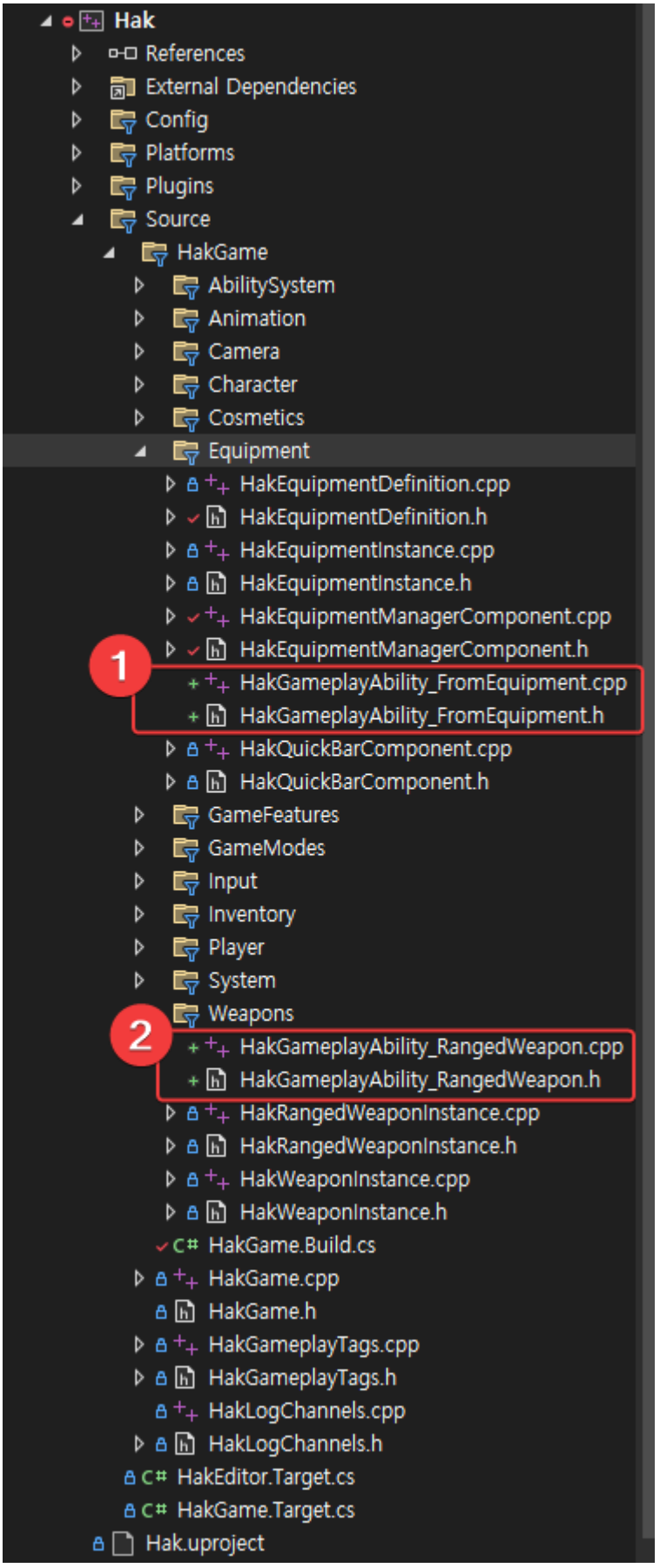
1


# GameplayAbility_RangedWeapon:

▼ **펼치기**

- 앞서 정의한 EquipmentDefinition에 할당할 GameplayAbility를 정의해야 한다:
  - Pistol에 할당할 GameplayAbility가 모태가 되는 GameplayAbility_RangedWeapon을 정의하자

- [ ] 파일을 추가해주자:


Hak
References
External Dependencies
Config
Platforms
Plugins
Source
HakGame
AbilitySystem
Animation
Camera
Character
Cosmetics
Equipment
HakEquipmentDefinition.cpp
HakEquipmentDefinition.h
HakEquipmentInstance.cpp
HakEquipmentInstance.h
HakEquipmentManagerComponent.cpp
HakEquipmentManagerComponent.h
1
HakGameplayAbility_FromEquipment.cpp
HakGameplayAbility_FromEquipment.h
HakQuickBarComponent.cpp
HakQuickBarComponent.h
GameFeatures
GameModes
Input
Inventory
Player
System
Weapons
2
HakGameplayAbility_RangedWeapon.cpp
HakGameplayAbility_RangedWeapon.h
HakRangedWeaponInstance.cpp
HakRangedWeaponInstance.h
HakWeaponInstance.cpp
HakWeaponInstance.h
HakGame.Build.cs
HakGame.cpp
HakGame.h
HakGameplayTags.cpp
HakGameplayTags.h
HakLogChannels.cpp
HakLogChannels.h
HakEditor.Target.cs
HakGame.Target.cs
Hak.uproject

- [ ] Equipment/HakGameplayAbility_FromEquipment.h/.cpp 생성하자:
- [ ] Weapon/HakGameplayAbility_RangedWeapon.h/.cpp 생성하자:

- [ ] HakGameplayAbility_FromEquipment의 기본 형태 구현하자:

```cpp
#pragma once

#include "CoreMinimal.h"
#include "HakGame/AbilitySystem/Abilities/HakGameplayAbility.h"
#include "HakGameplayAbility_FromEquipment.generated.h"

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

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

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

- [ ] HakGameplayAbility_RangedWeapon의 기본 형태 구현하자:

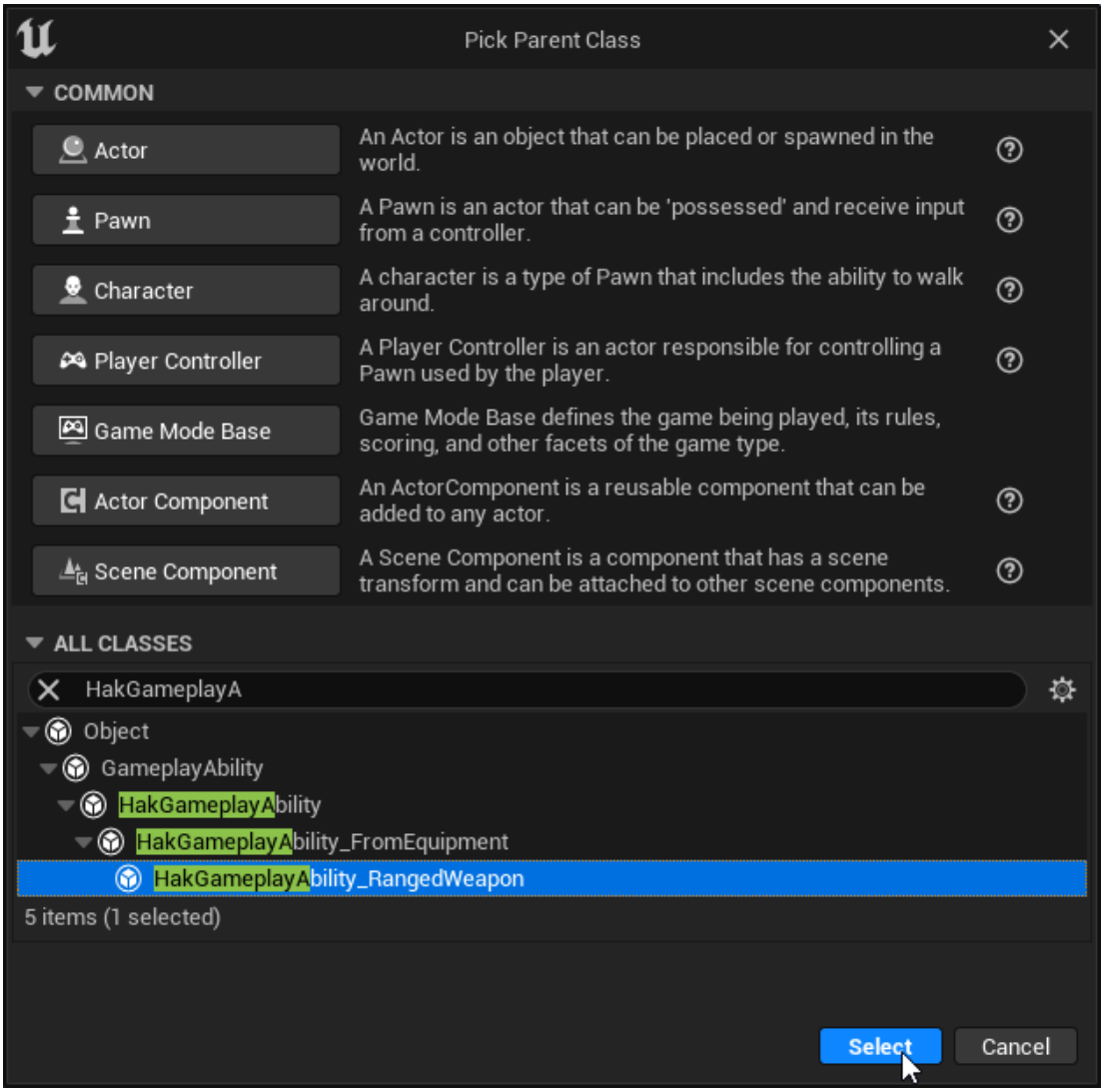
```cpp
#pragma once

#include "CoreMinimal.h"
#include "HakGame/Equipment/HakGameplayAbility_FromEquipment.h"
#include "HakGameplayAbility_RangedWeapon.generated.h"

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

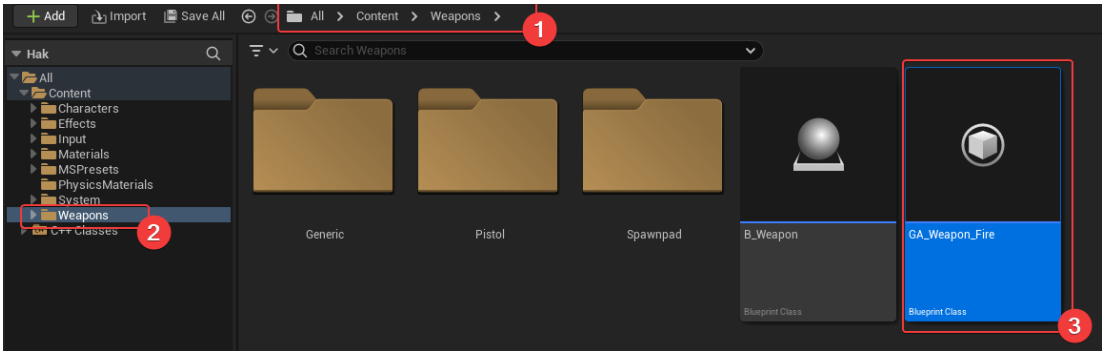
```cpp
#include "HakGameplayAbility_RangedWeapon.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakGameplayAbility_RangedWeapon)

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

# GA_Weapon_Fire_Pistol:

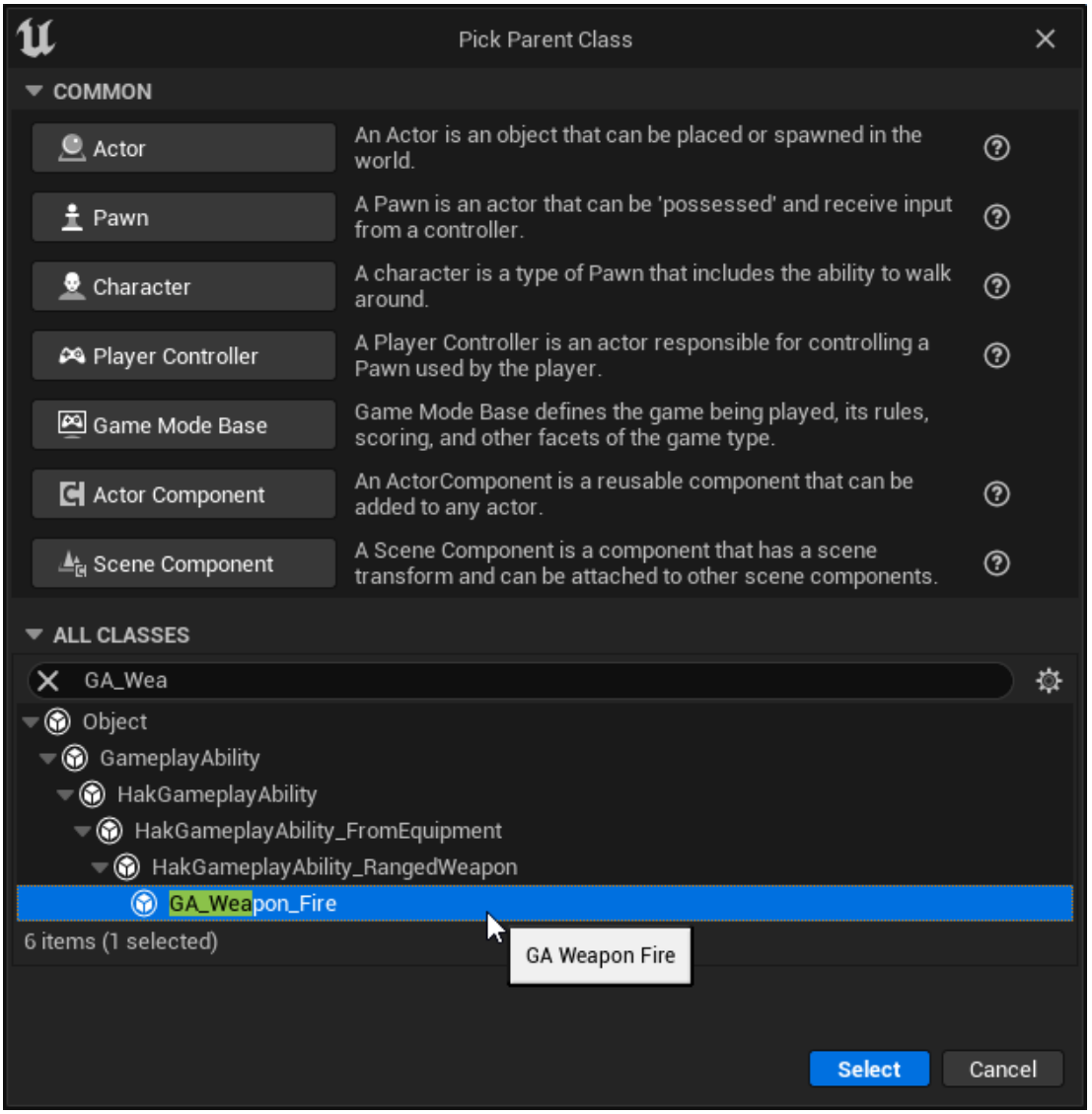
▼ **펼치기**

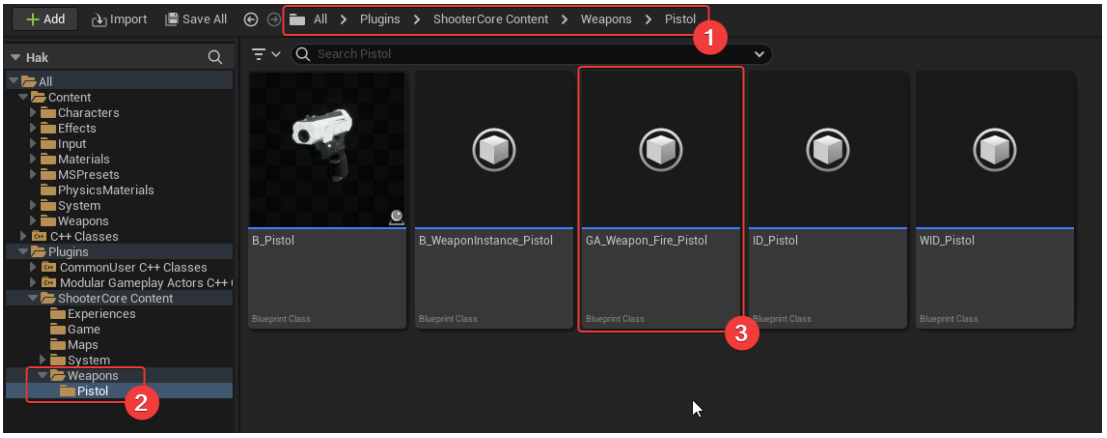
- [ ] HakGameplayAbility_RangedWeapon을 상속받는 GA_Weapon_Fire을 만들어 주자:



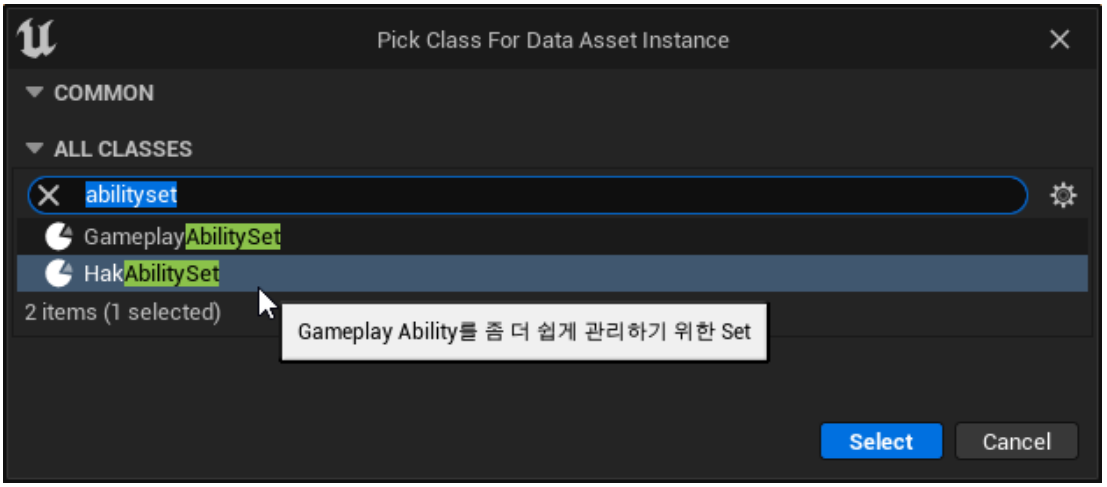



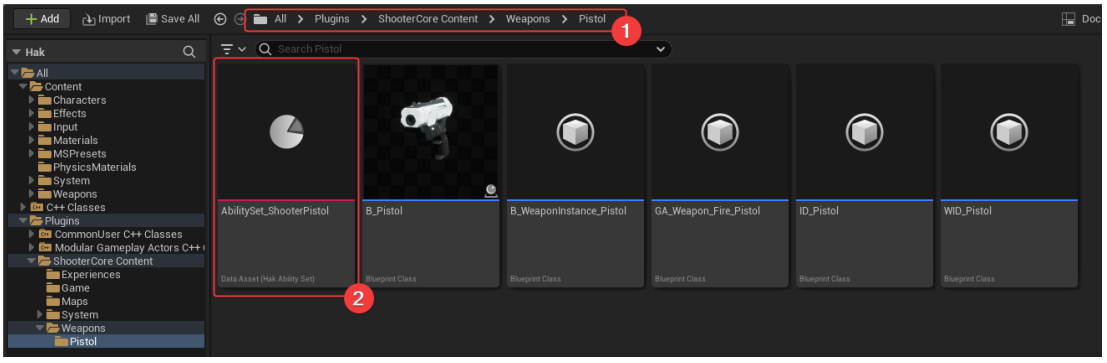

☐ GA_Weapon_Fire을 상속받는 GA_Weapon_Fire_Pistol을 생성하자:



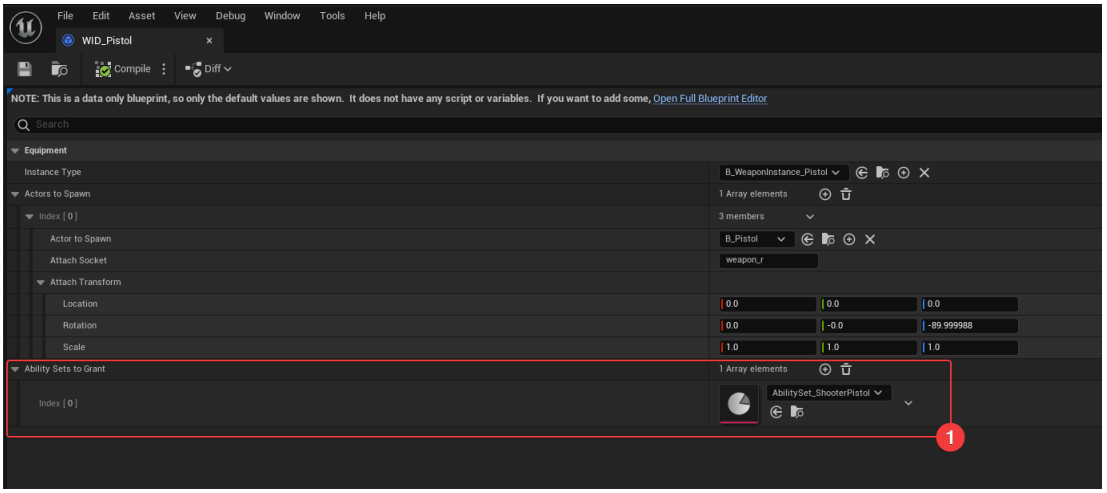




☐ Pistol의 EquipmentDefinition인 WID_Pistol에 넣을 AbilitySet을 생성하자:

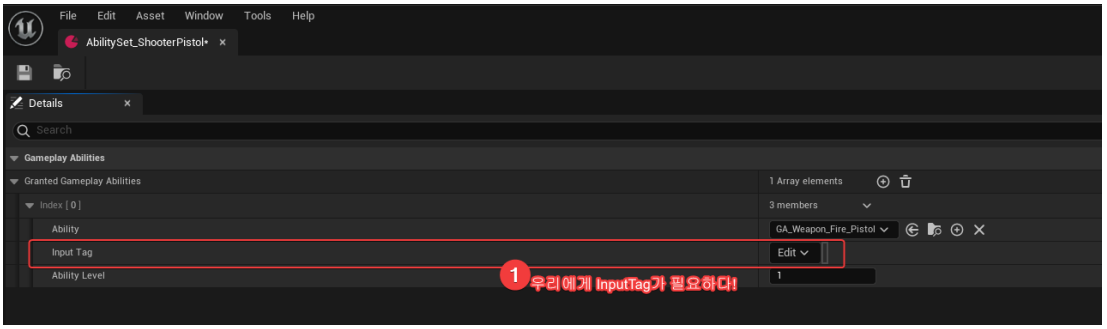


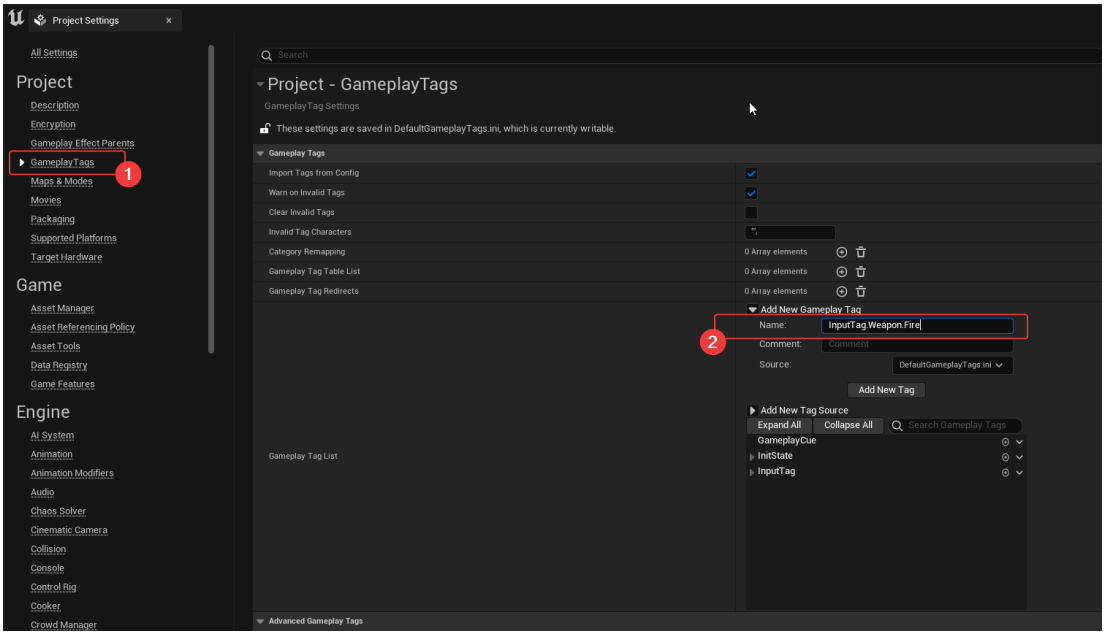

☐ 생성한 AbilitySet_ShooterPistol을 Equipement Definition인 WID_Pistol에 AbilitySetsToGrant에 추가해주자:




☐ AbilitySet_ShooterPistol에 허용할(Granted) Ability를 넣어주자:




☐ GameplayTag인 InputTag를 추가하자:




☐ AbilitySet_ShooterPistol에 InputTag에 추가하자:

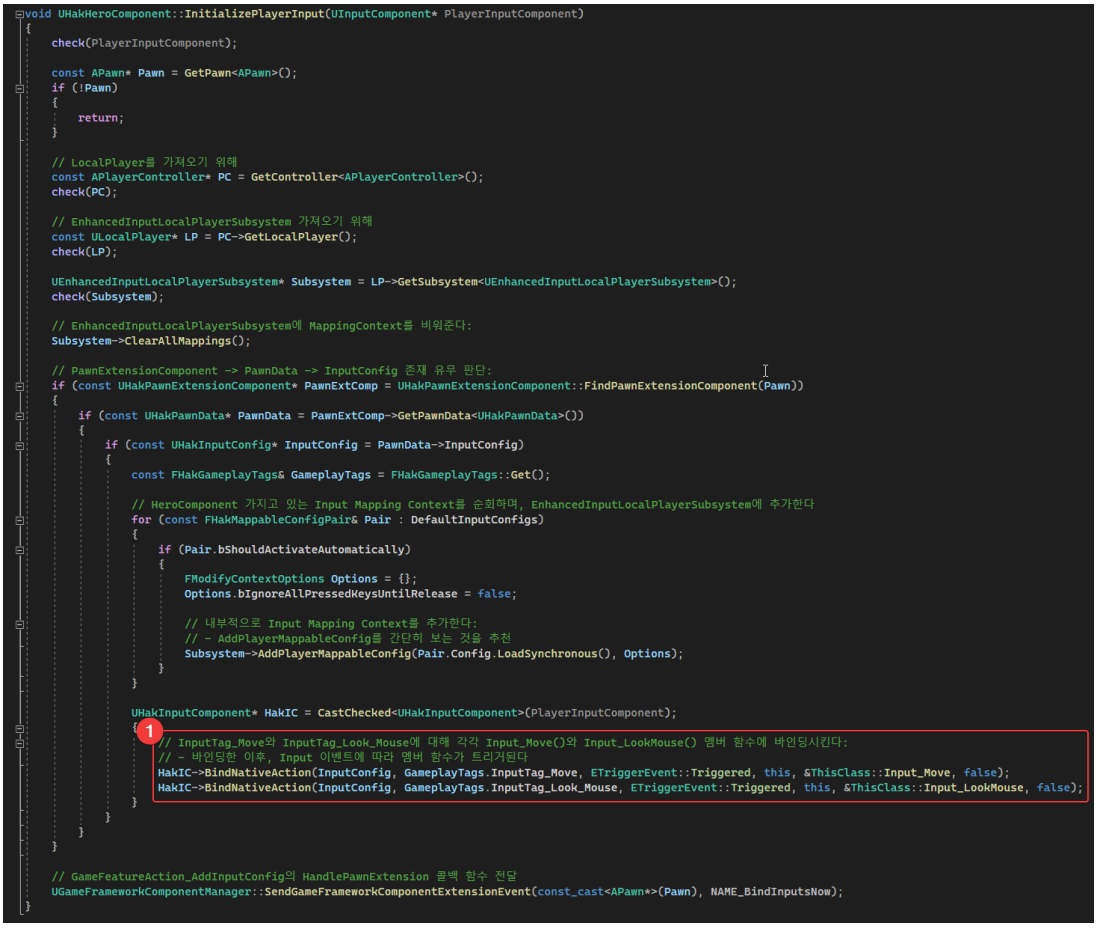

---

- **<u>InputTag를 활성화하여, GA_Weapon_Fire_Pistol의 GameplayAbility를 활성화하는 로직이 필요</u>**하다:
  - 아마 Input을 통해 진행하지 않을까 싶은데 계속 진행해보자

# AbilitySystemComponent의 Input 처리:

▼ **펼치기**

- Input 관련 GameplayTag는 HeroComponent에 처리했음을 우리는 기억하고 있다:



```
void UHakHeroComponent::InitializePlayerInput(UInputComponent* PlayerInputComponent)
{
    check(PlayerInputComponent);

    const APawn* Pawn = GetPawn<APawn>();
    if (!Pawn)
    {
        return;
    }

    // LocalPlayer를 가져오기 위해
    const APlayerController* PC = GetController<APlayerController>();
    check(PC);

    // EnhancedInputLocalPlayerSubsystem 가져오기 위해
    const ULocalPlayer* LP = PC->GetLocalPlayer();
    check(LP);

    UEnhancedInputLocalPlayerSubsystem* Subsystem = LP->GetSubsystem<UEnhancedInputLocalPlayerSubsystem>();
    check(Subsystem);

    // EnhancedInputLocalPlayerSubsystem에 MappingContext를 비워준다:
    Subsystem->ClearAllMappings();

    // PawnExtensionComponent -> PawnData -> InputConfig 존재 유무 판단:
    if (const UHakPawnExtensionComponent* PawnExtComp = UHakPawnExtensionComponent::FindPawnExtensionComponent(Pawn))
    {
        if (const UHakPawnData* PawnData = PawnExtComp->GetPawnData<UHakPawnData>())
        {
            if (const UHakInputConfig* InputConfig = PawnData->InputConfig)
            {
                const FHakGameplayTags& GameplayTags = FHakGameplayTags::Get();

                // HeroComponent 가지고 있는 Input Mapping Context를 순회하며, EnhancedInputLocalPlayerSubsystem에 추가한다
                for (const FHakMappableConfigPair& Pair : DefaultInputConfigs)
                {
                    if (Pair.bShouldActivateAutomatically)
                    {
                        FModifyContextOptions Options = {};
                        Options.bIgnoreAllPressedKeysUntilRelease = false;

                        // 내부적으로 Input Mapping Context를 추가한다:
                        // - AddPlayerMappableConfig를 간단히 보는 것을 추천
                        Subsystem->AddPlayerMappableConfig(Pair.Config.LoadSynchronous(), Options);
                    }
                }

                UHakInputComponent* HakIC = CastChecked<UHakInputComponent>(PlayerInputComponent);
                {
                    // InputTag_Move와 InputTag_Look_Mouse에 대해 각각 Input_Move()와 Input_LookMouse() 멤버 함수에 바인딩시킨다:
                    // - 바인딩한 이후, Input 이벤트에 따라 멤버 함수가 트리거된다
                    HakIC->BindNativeAction(InputConfig, GameplayTags.InputTag_Move, ETriggerEvent::Triggered, this, &ThisClass::Input_Move, false);
                    HakIC->BindNativeAction(InputConfig, GameplayTags.InputTag_Look_Mouse, ETriggerEvent::Triggered, this, &ThisClass::Input_LookMouse, false);
                }
            }
        }
    }

    // GameFeatureAction_AddInputConfig의 HandlePawnExtension 콜백 함수 전달
    UGameFrameworkComponentManager::SendGameFrameworkComponentExtensionEvent(const_cast<APawn*>(Pawn), NAME_BindInputsNow);
}
```

  - 우리는 **Ability 관련 InputTag를 처리할 로직이 필요**하다.

---

☐ HakInputComponent에 BindAbilityActions 추가하자:

- 이미 우리가 미리 구현해 놓았다:

```cpp
#pragma once

#include "EnhancedInputComponent.h"
#include "InputTriggers.h"
#include "InputActionValue.h"
#include "HakInputConfig.h"
#include "HakInputComponent.generated.h"

UCLASS()
class UHakInputComponent : public UEnhancedInputComponent
{
    GENERATED_BODY()
public:
    UHakInputComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
    * member methods
    */
    template <class UserClass, typename FuncType>
    void BindNativeAction(const UHakInputConfig* InputConfig, const FGameplayTag& InputTag, ETriggerEvent TriggerEvent, UserClass* Object, FuncType Func, boo

    template <class UserClass, typename PressedFuncType, typename ReleasedFuncType>
    void BindAbilityActions(const UHakInputConfig* InputConfig, UserClass* Object, PressedFuncType PressedFunc, ReleasedFuncType ReleasedFunc, TArray<uint32>
};

template <class UserClass, typename FuncType>
void UHakInputComponent::BindNativeAction(const UHakInputConfig* InputConfig, const FGameplayTag& InputTag, ETriggerEvent TriggerEvent, UserClass* Object, Fu
{
    check(InputConfig);

    // 여기서 알 수 있듯이, InputConfig는 활성화 가능한 InputAction을 담고 있다.
    // - 만약 InputConfig에 없는 InputAction을 Binding시키면, nullptr을 반환하여, 바인딩하는데 실패한다!
    if (const UInputAction* IA = InputConfig->FindNativeInputActionForTag(InputTag, bLogIfNotFound))
    {
        BindAction(IA, TriggerEvent, Object, Func);
    }
}

template <class UserClass, typename PressedFuncType, typename ReleasedFuncType>
void UHakInputComponent::BindAbilityActions(const UHakInputConfig* InputConfig, UserClass* Object, PressedFuncType PressedFunc, ReleasedFuncType ReleasedFunc
{
    check(InputConfig);

    // AbilityAction에 대해서는 그냥 모든 InputAction에 다 바인딩 시킨다!
    for (const FHakInputAction& Action : InputConfig->AbilityInputActions)
    {
        if (Action.InputAction && Action.InputTag.IsValid())
        {
            if (PressedFunc)
            {
                BindHandles.Add(BindAction(Action.InputAction, ETriggerEvent::Triggered, Object, PressedFunc, Action.InputTag).GetHandle());
            }

            if (ReleasedFunc)
            {
                BindHandles.Add(BindAction(Action.InputAction, ETriggerEvent::Completed, Object, ReleasedFunc, Action.InputTag).GetHandle());
            }
        }
    }
}
```

- 이제 BIndAbilityActions가 어떤 용도로 쓰이는지 알 수 있다

☐ HeroComponent::InitializePlayerInput()에 BindAbilityActions을 추가하자:

```cpp
// PawnExtensionComponent -> PawnData -> InputConfig 존재 유무 판단:
if (const UHakPawnExtensionComponent* PawnExtComp = UHakPawnExtensionComponent::FindPawnExtensionComponent(Pawn))
{
    if (const UHakPawnData* PawnData = PawnExtComp->GetPawnData<UHakPawnData>())
    {
        if (const UHakInputConfig* InputConfig = PawnData->InputConfig)
        {
            const FHakGameplayTags& GameplayTags = FHakGameplayTags::Get();

            // HeroComponent 가지고 있는 Input Mapping Context를 순회하며, EnhancedInputLocalPlayerSubsystem에 추가한다
            for (const FHakMappableConfigPair& Pair : DefaultInputConfigs)
            {
                if (Pair.bShouldActivateAutomatically)
                {
                    FModifyContextOptions Options = {};
                    Options.bIgnoreAllPressedKeysUntilRelease = false;

                    // 내부적으로 Input Mapping Context를 추가한다:
                    // - AddPlayerMappableConfig를 간단히 보는 것을 추천
                    Subsystem->AddPlayerMappableConfig(Pair.Config.LoadSynchronous(), Options);
                }
            }

            UHakInputComponent* HakIC = CastChecked<UHakInputComponent>(PlayerInputComponent);
            {
                // InputTag_Move와 InputTag_Look_Mouse에 대해 각각 Input_Move()와 Input_LookMouse() 멤버 함수에 바인딩시킨다:
                // - 바인딩한 이후, Input 이벤트에 따라 멤버 함수가 트리거된다
                {
                    TArray<uint32> BindHandles;
                    HakIC->BindAbilityActions(InputConfig, this, &ThisClass::Input_AbilityInputTagPressed, &ThisClass::Input_AbilityInputTagReleased, BindHandles);
                }
                HakIC->BindNativeAction(InputConfig, GameplayTags.InputTag_Move, ETriggerEvent::Triggered, this, &ThisClass::Input_Move, false);
                HakIC->BindNativeAction(InputConfig, GameplayTags.InputTag_Look_Mouse, ETriggerEvent::Triggered, this, &ThisClass::Input_LookMouse, false);
            }
        }
    }
}
```

☐ Input_AbilityInputTagPressed()와 Input_AbilityInputTagReleased()를 추가하자:

```cpp
/**
 * component that sets up input and camera handling for player controlled pawns (or bots that simulate players)
 * - this depends on a PawnExtensionComponent to coordinate initialization
 *
 * 카메라, 입력 등 플레이어가 제어하는 시스템의 초기화를 처리하는 컴포넌트
 */
UCLASS(Blueprintable, Meta=(BlueprintSpawnableComponent))
class UHakHeroComponent : public UPawnComponent, public IGameFrameworkInitStateInterface
{
    GENERATED_BODY()
public:
    UHakHeroComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** FeatureName 정의 */
    static const FName NAME_ActorFeatureName;

    /** Extension Event 이름 정의 */
    static const FName NAME_BindInputsNow;

    /**
     * UPawnComponent interface
     */
    virtual void OnRegister() final;
    virtual void BeginPlay() final;
    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) final;

    /**
     * IGameFrameworkInitStateInterface
     */
    virtual FName GetFeatureName() const final { return NAME_ActorFeatureName; }
    virtual void OnActorInitStateChanged(const FActorInitStateChangedParams& Params) final;
    virtual bool CanChangeInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState) const final;
    virtual void HandleChangeInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState) final;
    virtual void CheckDefaultInitialization() final;

    /**
     * member methods
     */
    TSubclassOf<UHakCameraMode> DetermineCameraMode() const;
    void InitializePlayerInput(UInputComponent* PlayerInputComponent);

    void Input_Move(const FInputActionValue& InputActionValue);
    void Input_LookMouse(const FInputActionValue& InputActionValue);
    void Input_AbilityInputTagPressed(FGameplayTag InputTag);
    void Input_AbilityInputTagReleased(FGameplayTag InputTag);

    /**
     * member variables
     */
    UPROPERTY(EditAnywhere)
    TArray<FHakMappableConfigPair> DefaultInputConfigs;
};
```

1

```cpp
void UHakHeroComponent::Input_AbilityInputTagPressed(FGameplayTag InputTag)
{

}

void UHakHeroComponent::Input_AbilityInputTagReleased(FGameplayTag InputTag)
{

}
```

☐ Input_AbilityInputTagPressed/Released() 구현하자:

```cpp
void UHakHeroComponent::Input_AbilityInputTagPressed(FGameplayTag InputTag)
{
    if (const APawn* Pawn = GetPawn<APawn>())
    {
        if (const UHakPawnExtensionComponent* PawnExtComp = UHakPawnExtensionComponent::FindPawnExtensionComponent(Pawn))
        {
            if (UHakAbilitySystemComponent* HakASC = PawnExtComp->GetHakAbilitySystemComponent())
            {
                HakASC->AbilityInputTagPressed(InputTag);
            }
        }
    }
}

void UHakHeroComponent::Input_AbilityInputTagReleased(FGameplayTag InputTag)
{
    if (const APawn* Pawn = GetPawn<APawn>())
    {
        if (const UHakPawnExtensionComponent* PawnExtComp = UHakPawnExtensionComponent::FindPawnExtensionComponent(Pawn))
        {
            if (UHakAbilitySystemComponent* HakASC = PawnExtComp->GetHakAbilitySystemComponent())
            {
                HakASC->AbilityInputTagReleased(InputTag);
            }
        }
    }
}
```

☐ HakAbilitySystemComponent::AbilityInputTagPressed/Released 구현:

```cpp
void UHakAbilitySystemComponent::AbilityInputTagPressed(const FGameplayTag& InputTag)
{
    if (InputTag.IsValid())
    {
        // 허용된 GameplayAbilitySpec을 순회
        for (const FGameplayAbilitySpec& AbilitySpec : ActivatableAbilities.Items)
        {
            // Ability가 존재하고, DynamicAbilityTags에 InputTag에 있을 경우, InputPressed/Held에 넣어 Ability 처리를 대기한다
            if (AbilitySpec.Ability && (AbilitySpec.DynamicAbilityTags.HasTagExact(InputTag)))
            {
                InputPressedSpecHandles.AddUnique(AbilitySpec.Handle);
                InputHeldSpecHandles.AddUnique(AbilitySpec.Handle);
            }
        }
    }
}

void UHakAbilitySystemComponent::AbilityInputTagReleased(const FGameplayTag& InputTag)
{
    if (InputTag.IsValid())
    {
        for (const FGameplayAbilitySpec& AbilitySpec : ActivatableAbilities.Items)
        {
            if (AbilitySpec.Ability && (AbilitySpec.DynamicAbilityTags.HasTagExact(InputTag)))
            {
                // Released에 추가하고, Held에서는 제거해준다
                InputReleasedSpecHandles.AddUnique(AbilitySpec.Handle);
                InputHeldSpecHandles.Remove(AbilitySpec.Handle);
            }
        }
    }
}
```

☐ Input[Pressed|Released]SpecHandles와 InputHeldSpecHandles를 추가하자:

```cpp
#pragma once

#include "AbilitySystemComponent.h"
#include "GameplayAbilitySpec.h"
#include "HakAbilitySystemComponent.generated.h"

UCLASS()
class UHakAbilitySystemComponent : public UAbilitySystemComponent
{
    GENERATED_BODY()
public:
    UHakAbilitySystemComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * member methods
     */
    void AbilityInputTagPressed(const FGameplayTag& InputTag);
    void AbilityInputTagReleased(const FGameplayTag& InputTag);

    /** Ability Input 처리할 Pending Queue */
    TArray<FGameplayAbilitySpecHandle> InputPressedSpecHandles;
    TArray<FGameplayAbilitySpecHandle> InputReleasedSpecHandles;
    TArray<FGameplayAbilitySpecHandle> InputHeldSpecHandles;
};
```

1

---

- AbilityInput 관련하여 이벤트를 생성했다:
  - Pending Queue에 들어간 AbilitySpec을 처리해야 하는 로직이 필요하다

☐ HakPlayerController의 PostProcessInput()를 오버라이드하여, ASC의 Pending된 Input 관련 GameplayAbility를 처리하도록 하자:

☐ HakPlayerController::PostProcessInput()

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

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

1

```cpp
#include "HakPlayerController.h"
#include "HakPlayerState.h"
#include "HakGame/Camera/HakPlayerCameraManager.h"
#include "HakGame/AbilitySystem/HakAbilitySystemComponent.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakPlayerController)

AHakPlayerController::AHakPlayerController(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
    PlayerCameraManagerClass = AHakPlayerCameraManager::StaticClass();
}

void AHakPlayerController::PostProcessInput(const float DeltaTime, const bool bGamePaused)
{
    // 우선 PostProcessInput()가 언제 호출되는지 확인해보자:
    // - UPlayerInput::ProcessInputStack()에서 호출된다

    if (UHakAbilitySystemComponent* HakASC = GetHakAbilitySystemComponent())
    {
        HakASC->ProcessAbilityInput(DeltaTime, bGamePaused);
    }

    Super::PostProcessInput(DeltaTime, bGamePaused);
}
```

- [ ] HakPlayerController::GetHakAbilitySystemComponent()

```cpp
#pragma once

#include "ModularPlayerController.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 AModularPlayerController
{
    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;
};
```

```cpp
AHakPlayerState* AHakPlayerController::GetHakPlayerState() const
{
    // ECastCheckedType의 NullAllowed는 Null 반환을 의도할 경우 유용하다
    return CastChecked<AHakPlayerState>(PlayerState, ECastCheckedType::NullAllowed);
}

UHakAbilitySystemComponent* AHakPlayerController::GetHakAbilitySystemComponent() const
{
    const AHakPlayerState* HakPS = GetHakPlayerState();
    return (HakPS ? HakPS->GetHakAbilitySystemComponent() : nullptr);
}
```

- [ ] HakAbilitySystemComponent::ProcessAbilityInput()

```cpp
UCLASS()
class UHakAbilitySystemComponent : public UAbilitySystemComponent
{
    GENERATED_BODY()
public:
    UHakAbilitySystemComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * member methods
     */
    void AbilityInputTagPressed(const FGameplayTag& InputTag);
    void AbilityInputTagReleased(const FGameplayTag& InputTag);
    void ProcessAbilityInput(float DeltaTime, bool bGamePaused);

    /** Ability Input 처리할 Pending Queue */
    TArray<FGameplayAbilitySpecHandle> InputPressedSpecHandles;
    TArray<FGameplayAbilitySpecHandle> InputReleasedSpecHandles;
    TArray<FGameplayAbilitySpecHandle> InputHeldSpecHandles;
};
```

```cpp
void UHakAbilitySystemComponent::ProcessAbilityInput(float DeltaTime, bool bGamePaused)
{
    TArray<FGameplayAbilitySpecHandle> AbilitiesToActivate;

    // InputHeldSpecHandles에 대해 Ability 처리를 위해 AbilitiesToActivate에 추가한다
    for (const FGameplayAbilitySpecHandle& SpecHandle : InputHeldSpecHandles)
    {
        // FindAbilitySpecFromHandle 확인:
        // - ActivatableAbilities의 Handle 값 비교를 통해 GameplayAbilitySpec을 반환한다
        if (const FGameplayAbilitySpec* AbilitySpec = FindAbilitySpecFromHandle(SpecHandle))
        {
            if (AbilitySpec->Ability && !AbilitySpec->IsActive())
            {
                const UHakGameplayAbility* HakAbilityCDO = CastChecked<UHakGameplayAbility>(AbilitySpec->Ability);

                // ActivationPolicy가 WhileInputActive 속성이면 활성화로 등록
                if (HakAbilityCDO->ActivationPolicy == EHakAbilityActivationPolicy::WhileInputActive)
                {
                    AbilitiesToActivate.AddUnique(AbilitySpec->Handle);
                }
            }
        }
    }

    for (const FGameplayAbilitySpecHandle& SpecHandle : InputPressedSpecHandles)
    {
        if (FGameplayAbilitySpec* AbilitySpec = FindAbilitySpecFromHandle(SpecHandle))
        {
            if (AbilitySpec->Ability)
            {
                AbilitySpec->InputPressed = true;

                if (AbilitySpec->IsActive())
                {
                    // 이미 Ability가 활성화되어 있을 경우, Input Event(InputPressed)만 호출
                    // - AbilitySpecInputPressed 확인
                    AbilitySpecInputPressed(*AbilitySpec);
                }
                else
                {
                    const UHakGameplayAbility* HakAbilityCDO = CastChecked<UHakGameplayAbility>(AbilitySpec->Ability);

                    // ActivationPolicy가 OnInputTriggered 속성이면 활성화로 등록
                    if (HakAbilityCDO->ActivationPolicy == EHakAbilityActivationPolicy::OnInputTriggered)
                    {
                        AbilitiesToActivate.AddUnique(AbilitySpec->Handle);
                    }
                }
            }
        }
    }

    // 등록된 AbilitiesToActivate를 한꺼번에 등록 시작:
    for (const FGameplayAbilitySpecHandle& AbilitySpecHandle : AbilitiesToActivate)
    {
        // 모든 것이 잘 진행되었다면, CallActivate 호출로 BP의 Activate 노드가 실행될 것임
        TryActivateAbility(AbilitySpecHandle);
    }

    // 이번 프레임에 Release되었다면, 관련 GameplayAbility 처리:
    for (const FGameplayAbilitySpecHandle& SpecHandle : InputReleasedSpecHandles)
    {
        if (FGameplayAbilitySpec* AbilitySpec = FindAbilitySpecFromHandle(SpecHandle))
        {
            if (AbilitySpec->Ability)
            {
                AbilitySpec->InputPressed = false;
                if (AbilitySpec->IsActive())
                {
                    AbilitySpecInputReleased(*AbilitySpec);
                }
            }
        }
    }

    // InputHeldSpecHandles은 InputReleasedSpecHandles 추가될때 제거된다!
    InputPressedSpecHandles.Reset();
    InputReleasedSpecHandles.Reset();
}
```

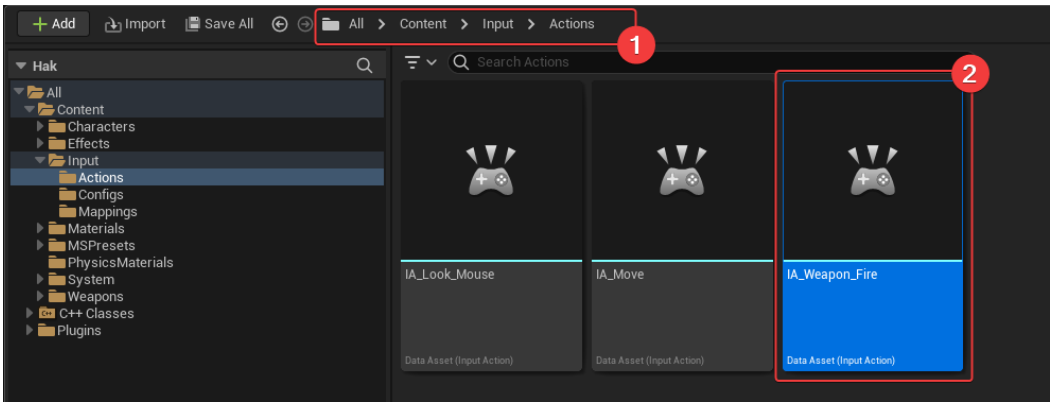
☐ HakAbilityActivationPolicy 추가:

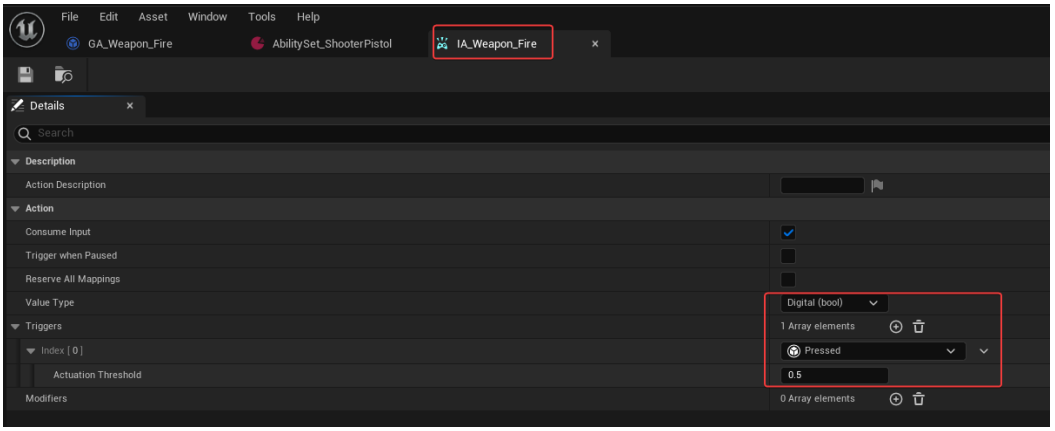
```cpp
#pragma once

#include "Abilities/GameplayAbility.h"
#include "HakGameplayAbility.generated.h"

UENUM(BlueprintType)
enum class EHakAbilityActivationPolicy : uint8
{
    /** Input이 Trigger 되었을 경우 (Presssed/Released) */
    OnInputTriggered,
    /** Input이 Held되어 있을 경우 */
    WhileInputActive,
    /** avatar가 생성되었을 경우, 바로 할당 */
    OnSpawn,
};

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

    /** 언제 GA가 활성화될지 정책 */
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Hak|AbilityActivation")
    EHakAbilityActivationPolicy ActivationPolicy;
};
```

```cpp
UHakGameplayAbility::UHakGameplayAbility(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
    ActivationPolicy = EHakAbilityActivationPolicy::OnInputTriggered;
}
```

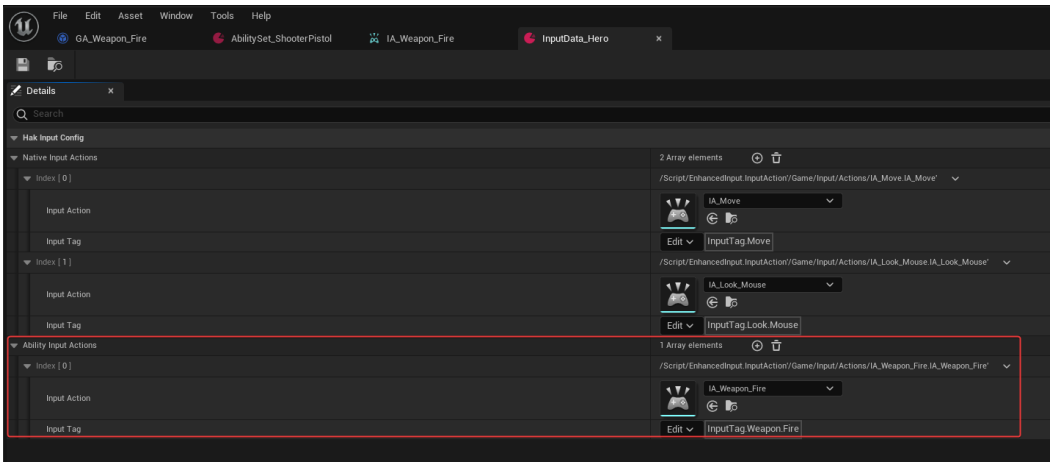
☐ InputTag.Weapon.Fire의 GameplayTag가 활성화되기 위한 Input Binding을 진행해야 한다:

- ☐ InputAction, IA_Weapon_Fire을 추가하자:

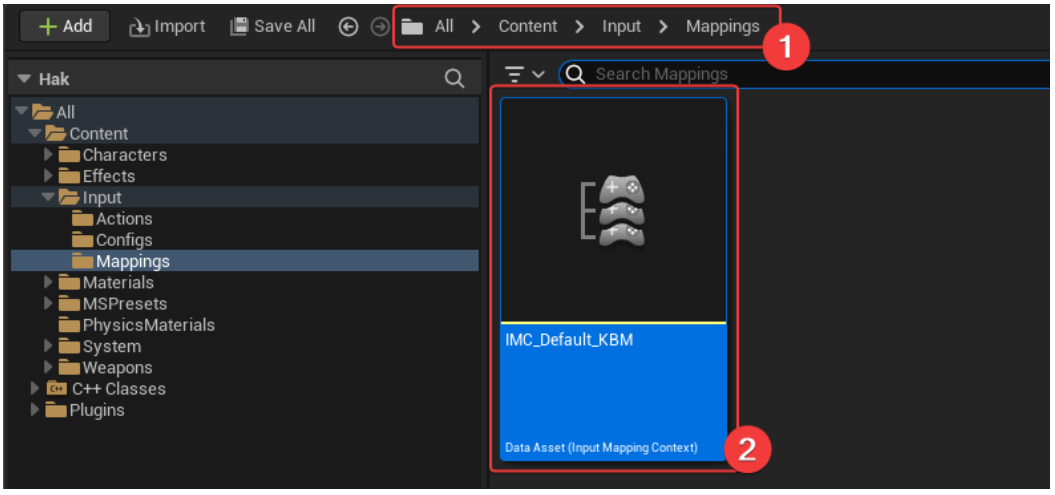


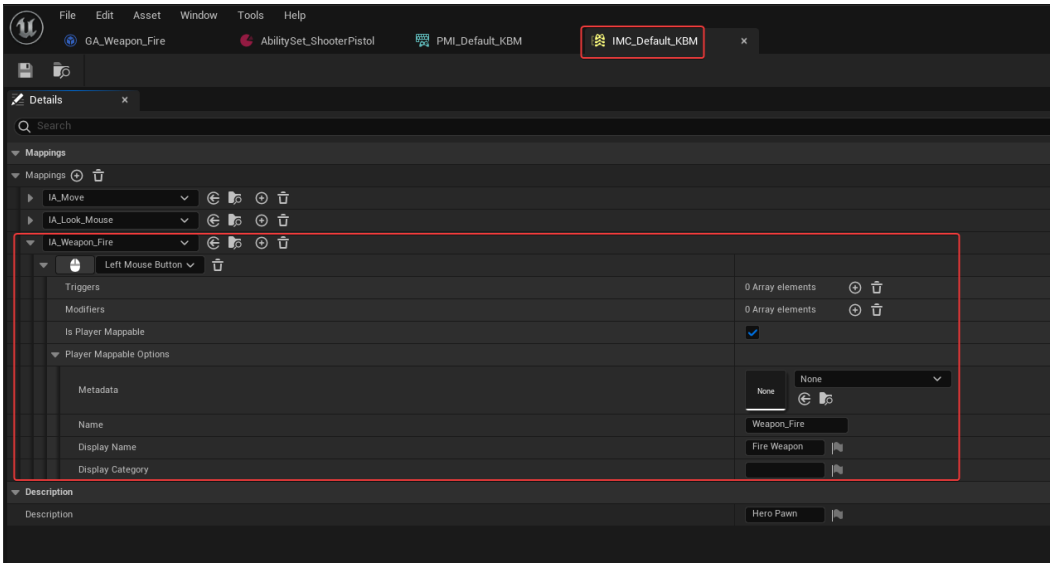



☐ InputData_Hero에 InputAction Binding인 Ability Input Actions을 추가하자:




☐ IA_Weapon_Fire의 InputMapping을 추가하자:

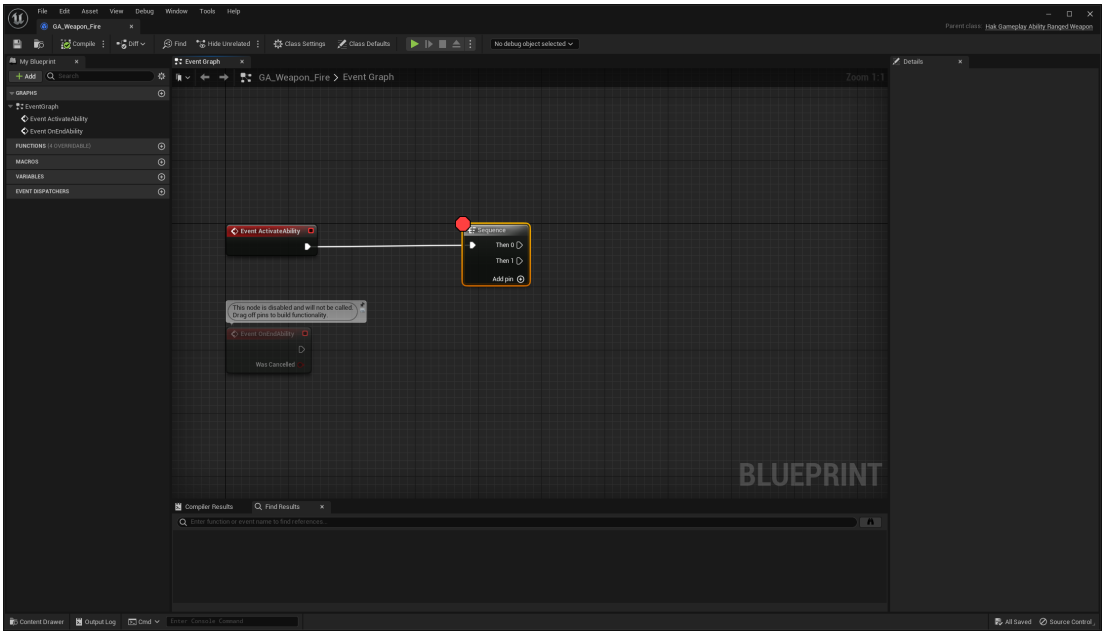




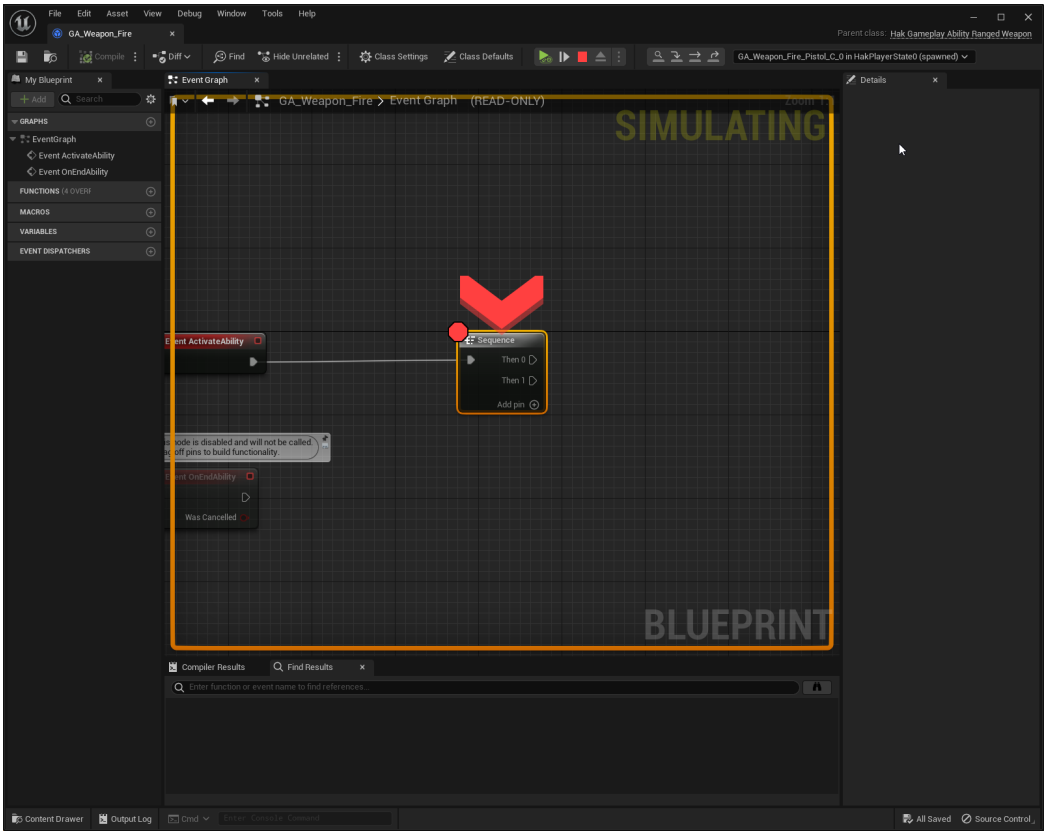

# GA_Weapon_Fire:

▼ **펼치기**

☐ Event ActivateAbility에 호출되는지 확인하자:




- 아래와 같이 잘 작동함을 확인할 수 있다:

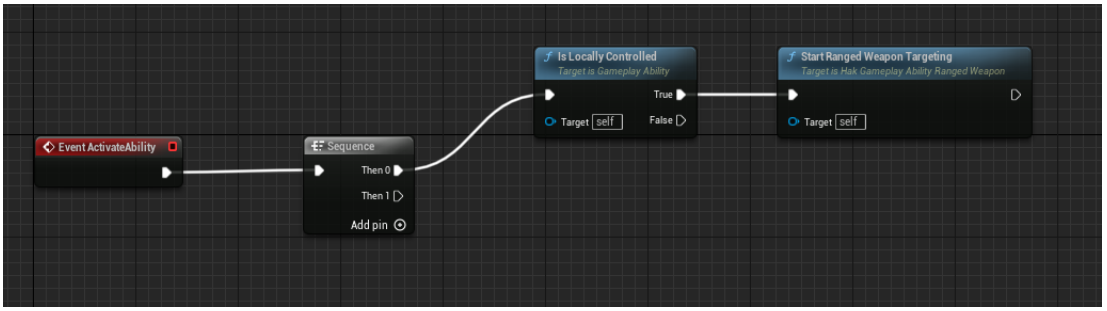



☐ HakGameplayAbility_RangedWeapon::StartRangedWeaponTargeting() 추가:

```
UCLASS()
class UHakGameplayAbility_RangedWeapon : public UHakGameplayAbility_FromEquipment
{
    GENERATED_BODY()
public:
    UHakGameplayAbility_RangedWeapon(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UFUNCTION(BlueprintCallable)
    void StartRangedWeaponTargeting();
};
```

```
void UHakGameplayAbility_RangedWeapon::StartRangedWeaponTargeting()
{

}
```

☐ GA_Weapon_Fire의 ActivateAbility Event에 추가:




☐ StartRangedWeaponTargeting() 구현:

```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
            FHakGameplayAbilityTargetData_SingleTargetHit* NewTargetData = new FHakGameplayAbilityTargetData_SingleTargetHit();
            NewTargetData->HitResult = FoundHit;
            NewTargetData->CartridgeID = CartridgeID;
            TargetData.Add(NewTargetData);
        }
    }

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

☐ PerformLocalTargeting() 구현:

```cpp
UCLASS()
class UHakGameplayAbility_RangedWeapon : public UHakGameplayAbility_FromEquipment
{
    GENERATED_BODY()
public:
    UHakGameplayAbility_RangedWeapon(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    void PerformLocalTargeting(TArray<FHitResult>& OutHits);

    UFUNCTION(BlueprintCallable)
    void StartRangedWeaponTargeting();
};
```

```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);
    }
}
```

- **EAxis::X로 하는 이유는 ForwardVector가 X축이다**:

UnrealMath.cpp | Vector.h | HakGameplay...mEquipment.h | HakGameplayA...quipment.cpp | HakGameplay...edWeapon.cpp | HakGameplay...gedWeapon.h* | AbilitySystemC..._Abilities.c

UE5 | FMath

```cpp
#include "UObject/PropertyPortFlags.h"
#include "Math/IntRect.h"
#include "Math/Matrix.h"
#include "Math/Quat.h"
#include "Math/Vector.h"
#include "Math/Vector4.h"

DEFINE_LOG_CATEGORY(LogUnrealMath);

/*-----------------------------------------------------------------------------
    Globals
-----------------------------------------------------------------------------*/

template<> const FMatrix44f FMatrix44f::Identity(FPlane4f(1, 0, 0, 0), FPlane4f(0, 1, 0, 0), FPlane4f(0, 0, 1, 0), FPlane4f(0,
template<> const FMatrix44d FMatrix44d::Identity(FPlane4d(1, 0, 0, 0), FPlane4d(0, 1, 0, 0), FPlane4d(0, 0, 1, 0), FPlane4d(0,

template<> const FQuat4f FQuat4f::Identity(0.f, 0.f, 0.f, 1.f);
template<> const FQuat4d FQuat4d::Identity(0.0, 0.0, 0.0, 1.0);

template<> const FRotator3f FRotator3f::ZeroRotator(0, 0, 0);
template<> const FRotator3d FRotator3d::ZeroRotator(0, 0, 0);

template<> const FVector3f FVector3f::ZeroVector(0, 0, 0);
template<> const FVector3f FVector3f::OneVector(1, 1, 1);
template<> const FVector3f FVector3f::UpVector(0, 0, 1);
template<> const FVector3f FVector3f::DownVector(0, 0, -1);
template<> const FVector3f FVector3f::ForwardVector(1, 0, 0);
template<> const FVector3f FVector3f::BackwardVector(-1, 0, 0);
template<> const FVector3f FVector3f::RightVector(0, 1, 0);
template<> const FVector3f FVector3f::LeftVector(0, -1, 0);
template<> const FVector3f FVector3f::XAxisVector(1, 0, 0);
template<> const FVector3f FVector3f::YAxisVector(0, 1, 0);
template<> const FVector3f FVector3f::ZAxisVector(0, 0, 1);
template<> const FVector3d FVector3d::ZeroVector(0, 0, 0);
template<> const FVector3d FVector3d::OneVector(1, 1, 1);
template<> const FVector3d FVector3d::UpVector(0, 0, 1);
template<> const FVector3d FVector3d::DownVector(0, 0, -1);
template<> const FVector3d FVector3d::ForwardVector(1, 0, 0);
template<> const FVector3d FVector3d::BackwardVector(-1, 0, 0);
template<> const FVector3d FVector3d::RightVector(0, 1, 0);
template<> const FVector3d FVector3d::LeftVector(0, -1, 0);
template<> const FVector3d FVector3d::XAxisVector(1, 0, 0);
template<> const FVector3d FVector3d::YAxisVector(0, 1, 0);
template<> const FVector3d FVector3d::ZAxisVector(0, 0, 1);

template<> const FVector2f FVector2f::ZeroVector(0, 0);
template<> const FVector2f FVector2f::UnitVector(1, 1);
template<> const FVector2f FVector2f::Unit45Deg(UE_INV_SQRT_2, UE_INV_SQRT_2);
template<> const FVector2d FVector2d::ZeroVector(0, 0);
template<> const FVector2d FVector2d::UnitVector(1, 1);
template<> const FVector2d FVector2d::Unit45Deg(UE_INV_SQRT_2, UE_INV_SQRT_2);
```

- [ ] GetWeaponInstance()

```cpp
UCLASS()
class UHakGameplayAbility_RangedWeapon : public UHakGameplayAbility_FromEquipment
{
    GENERATED_BODY()
public:
    UHakGameplayAbility_RangedWeapon(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UHakRangedWeaponInstance* GetWeaponInstance();
    void PerformLocalTargeting(TArray<FHitResult>& OutHits);

    UFUNCTION(BlueprintCallable)
    void StartRangedWeaponTargeting();
};
```

```cpp
UHakRangedWeaponInstance* UHakGameplayAbility_RangedWeapon::GetWeaponInstance()
{
    return Cast<UHakRangedWeaponInstance>(GetAssociatedEquipment());
}
```

☐ HakGameAbility_FromEquipment::GetAssociatedEquipment()

```cpp
UCLASS()
class UHakGameplayAbility_FromEquipment : public UHakGameplayAbility
{
    GENERATED_BODY()
public:
    UHakGameplayAbility_FromEquipment(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UHakEquipmentInstance* GetAssociatedEquipment() const;
};
```

```cpp
UHakEquipmentInstance* UHakGameplayAbility_FromEquipment::GetAssociatedEquipment() const
{
    // CurrentActorInfo의 AbilitySystemComponent와 CurrentSpecHandle을 활용하여, GameplayAbilitySpec을 가져옴:
    // - CurrentSpecHandle은 SetCurrentActorInfo() 호출할 때, Handle 값을 받아서 저장됨:
    // - CurrentSpecHandle와 CurrentActorInfo은 같이 함
    // - FindAbilitySpecFromHandle은 GiveAbility로 허용된 ActivatableAbilities를 순회하여 GameplayAbilitySpec을 찾아냄
    if (FGameplayAbilitySpec* Spec = UGameplayAbility::GetCurrentAbilitySpec())
    {
        // GameplayAbility_FromEquipment는 EquipmentInstance로부터 GiveAbility를 진행했으므로, SourceObject에 EquipmentInstance가 저장되어 있음
        return Cast<UHakEquipmentInstance>(Spec->SourceObject.Get());
    }
    return nullptr;
}
```

☐ RangedWeaponFiringInput 정의:

```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());

    UHakRangedWeaponInstance* GetWeaponInstance();
    void PerformLocalTargeting(TArray<FHitResult>& OutHits);

    UFUNCTION(BlueprintCallable)
    void StartRangedWeaponTargeting();
};
```

- [ ] GetTargetingTransform()

```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());

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

    UFUNCTION(BlueprintCallable)
    void StartRangedWeaponTargeting();
};
```

- [ ] GetWeaponTargetingSourceLocation()

```cpp
FVector UHakGameplayAbility_RangedWeapon::GetWeaponTargetingSourceLocation() const
{
    // 미구현인거 같다... Weapon 위치가 아닌 그냥 Pawn의 위치를 가져온다...
    APawn* const AvatarPawn = Cast<APawn>(GetAvatarActorFromActorInfo());
    check(AvatarPawn);

    const FVector SourceLoc = AvatarPawn->GetActorLocation();
    return SourceLoc;
}
```

- [ ] GetTargetingTransform()

```cpp
FTransform UHakGameplayAbility_RangedWeapon::GetTargetingTransform(APawn* SourcePawn, EHakAbilityTargetingSource Source)
{
    check(SourcePawn);
    check(Source == EHakAbilityTargetingSource::CameraTowardsFocus);

    // 참고로 아래 로직은 CameraTowardsFocus만 추출한 로직이다:
    // - 완전한 로직은 Lyra를 참고

    AController* Controller = SourcePawn->Controller;
    if (Controller == nullptr)
    {
        return FTransform();
    }

    // 매직넘버이다...
    double FocalDistance = 1024.0f;
    FVector FocalLoc;
    FVector CamLoc;
    FRotator CamRot;

    // PlayerController로부터, Location과 Rotation 정보를 가져옴
    APlayerController* PC = Cast<APlayerController>(Controller);
    check(PC);
    PC->GetPlayerViewPoint(CamLoc, CamRot);

    FVector AimDir = CamRot.Vector().GetSafeNormal();
    FocalLoc = CamLoc + (AimDir * FocalDistance);

    // WeaponLoc이 아닌 Pawn의 Loc이다
    const FVector WeaponLoc = GetWeaponTargetingSourceLocation();
    FVector FinalCamLoc = FocalLoc + (((WeaponLoc - FocalLoc) | AimDir) * AimDir);

#if 0
    {
        // WeaponLoc (사실상 ActorLoc)
        DrawDebugPoint(GetWorld(), WeaponLoc, 10.0f, FColor::Red, false, 60.0f);
        // CamLoc
        DrawDebugPoint(GetWorld(), CamLoc, 10.0f, FColor::Yellow, false, 60.0f);
        // FinalCamLoc
        DrawDebugPoint(GetWorld(), FinalCamLoc, 10.0f, FColor::Magenta, false, 60.0f);

        // (WeaponLoc - FocalLoc)
        DrawDebugLine(GetWorld(), FocalLoc, WeaponLoc, FColor::Yellow, false, 60.0f, 0, 2.0f);
        // (AimDir)
        DrawDebugLine(GetWorld(), CamLoc, FocalLoc, FColor::Blue, false, 60.0f, 0, 2.0f);

        // Project Direction Line
        DrawDebugLine(GetWorld(), WeaponLoc, FinalCamLoc, FColor::Red, false, 60.0f, 0, 2.0f);
    }
#endif

    // Camera -> Focus 계산 완료
    return FTransform(CamRot, FinalCamLoc);
}
```

☐ MaxDamageRange 추가:

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

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category="WeaponConfig", meta=(ForceUnits=cm))
    float MaxDamageRange = 25000.0f;
};
```

☐ TraceBulletsInCartridge()

```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 TraceBulletsInCartridge(const FRangedWeaponFiringInput& InputData, TArray<FHitResult>& OutHits);

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

    UFUNCTION(BlueprintCallable)
    void StartRangedWeaponTargeting();
};
```

```cpp
void UHakGameplayAbility_RangedWeapon::TraceBulletsInCartridge(const FRangedWeaponFiringInput& InputData, TArray<FHitResult>& OutHits)
{
    UHakRangedWeaponInstance* WeaponData = InputData.WeaponData;
    check(WeaponData);

    // MaxDamageRange를 고려하여, EndTrace를 정의하자
    const FVector BulletDir = InputData.AimDir;
    const FVector EndTrace = InputData.StartTrace + (BulletDir * WeaponData->MaxDamageRange);

    // HitLocation의 초기화 값으로 EndTrace로 설정
    FVector HitLocation = EndTrace;

    // 총알을 하나 Trace 진행한다:
    // - 참고로 Lyra의 경우, 샷건과 같은 Cartridge에 여러개의 총알이 있을 경우를 처리하기 위해 for-loop을 활용하여, 복수개 Bullet을 Trace한다
    TArray<FHitResult> AllImpacts;
    FHitResult Impact = DoSingleBulletTrace(InputData.StartTrace, EndTrace, WeaponData->BulletTraceWeaponRadius, /**bIsSimulated=*/ false, /**out*/ AllImpacts);

    const AActor* HitActor = Impact.GetActor();
    if (HitActor)
    {
        if (AllImpacts.Num() > 0)
        {
            OutHits.Append(AllImpacts);
        }

        HitLocation = Impact.ImpactPoint;
    }

    // OutHits가 적어도 하나가 존재하도록, EndTrace를 활용하여, OutHits에 추가해준다
    if (OutHits.Num() == 0)
    {
        if (!Impact.bBlockingHit)
        {
            Impact.Location = EndTrace;
            Impact.ImpactPoint = EndTrace;
        }

        OutHits.Add(Impact);
    }
}
```

☐ BulletTraceWeaponRadius 추가:

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

    /** 유효 사거리 */
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category="WeaponConfig", meta=(ForceUnits=cm))
    float MaxDamageRange = 25000.0f;

    /** 총탄 범위 (Sphere Trace Sweep) */
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category="WeaponConfig", meta = (ForceUnits = cm))
    float BulletTraceWeaponRadius = 0.0f;
};
```

## ☐ DoSingleBulletTrace()

```cpp
void UHakGameplayAbility_RangedWeapon::DoSingleBulletTrace(const FVector& StartTrace, const FVector& EndTrace, float SweepRadius, bool bIsSimulated, TArray<FHitResult>& OutHits) const
{
    FHitResult Impact;

    // 우선 SweepRadius 없이 한번 Trace 진행한다 (SweepTrace는 무겁기 때문)
    // - FindFirstPawnHitResult()를 여러번 Trace 진행을 막기 위해, OutHits를 확인해서 APawn 충돌 정보있으면 더이상 Trace하지 않는다
    if (FindFirstPawnHitResult(OutHits) == INDEX_NONE)
    {
        Impact = WeaponTrace(StartTrace, EndTrace, /*SweepRadius=*/0.0f, bIsSimulated, /*out*/ OutHits);
    }

    if (FindFirstPawnHitResult(OutHits) == INDEX_NONE)
    {
        // 만약 SweepRadius가 0보다 크면, 0.0일때 대비 충돌 가능성이 커지므로 한번 더 Trace 진행
        if (SweepRadius > 0.0f)
        {
            // SweepHits에 Trace의 OutHits 정보를 저장
            TArray<FHitResult> SweepHits;
            Impact = WeaponTrace(StartTrace, EndTrace, SweepRadius, bIsSimulated, SweepHits);

            // Sphere Trace로 진행한 결과인 SweepHits를 검색하여, Pawn이 있는가 검색
            const int32 FirstPawnIdx = FindFirstPawnHitResult(SweepHits);
            if (SweepHits.IsValidIndex(FirstPawnIdx))
            {
                // 만약 있다면, SweepHits를 FirstPawnIdx까지 순회하며, bBlockingHit와 기존 OutHits에 없을 경우 체크한다
                bool bUseSweepHits = true;
                for (int32 Idx = 0; Idx < FirstPawnIdx; ++Idx)
                {
                    const FHitResult& CurHitResult = SweepHits[Idx];

                    auto Pred = [&CurHitResult](const FHitResult& Other)
                    {
                        return Other.HitObjectHandle == CurHitResult.HitObjectHandle;
                    };

                    // OutHits에 있다면... SweepHits를 OutHits로 업데이트 하지 않는다 (이미 충돌했던 정보가 있으니깐?) (early-out용)
                    // - OutHits에 bBlockingHits가 SweepHits로 있음을 알게되었음
                    if (CurHitResult.bBlockingHit && OutHits.ContainsByPredicate(Pred))
                    {
                        bUseSweepHits = false;
                        break;
                    }
                }

                // SweepHits
                if (bUseSweepHits)
                {
                    OutHits = SweepHits;
                }
            }
        }
    }

    return Impact;
}
```

## ☐ FindFirstPawnHitResult()

```cpp
int32 FindFirstPawnHitResult(const TArray<FHitResult>& HitResults)
{
    for (int32 Idx = 0; Idx < HitResults.Num(); ++Idx)
    {
        const FHitResult& CurHitResult = HitResults[Idx];
        if (CurHitResult.HitObjectHandle.DoesRepresentClass(APawn::StaticClass()))
        {
            return Idx;
        }
        else
        {
            AActor* HitActor = CurHitResult.HitObjectHandle.FetchActor();

            // 한단계 AttachParent에 Actor가 Pawn이라면?
            // - 보통 복수개 단계로 AttachParent를 하지 않으므로, AttachParent 대상이 APawn이라고 생각할 수도 있겠다
            if ((HitActor != nullptr) && (HitActor->GetAttachParentActor() != nullptr) && (Cast<APawn>(HitActor->GetAttachParentActor()) != nullptr))
            {
                return Idx;
            }
        }
    }
    return INDEX_NONE;
}
```

## ☐ WeaponTrace()

```cpp
FHitResult UHakGameplayAbility_RangedWeapon::WeaponTrace(const FVector& StartTrace, const FVector& EndTrace, float SweepRadius, bool bIsSimulated, TArray<FHitResult>& OutHitResults) const
{
    TArray<FHitResult> HitResults;

    // Complex Geometry로 Trace를 진행하며, AvatarActor를 AttachParent를 가지는 오브젝트와의 충돌은 무시한다
    FCollisionQueryParams TraceParams(SCENE_QUERY_STAT(WeaponTrace), /*bTraceComplex=*/true, /*IgnoreActor=*/GetAvatarActorFromActorInfo());
    TraceParams.bReturnPhysicalMaterial = true;

    // AvatarActor에 부착된 Actors를 찾아 IgnoredActors에 추가한다
    AddAdditionalTraceIgnoreActors(TraceParams);

    // Weapon 관련 Collision Channel로 Trace 진행
    const ECollisionChannel TraceChannel = DetermineTraceChannel(TraceParams, bIsSimulated);
    if (SweepRadius > 0.0f)
    {
        GetWorld()->SweepMultiByChannel(HitResults, StartTrace, EndTrace, FQuat::Identity, TraceChannel, FCollisionShape::MakeSphere(SweepRadius), TraceParams);
    }
    else
    {
        GetWorld()->LineTraceMultiByChannel(HitResults, StartTrace, EndTrace, TraceChannel, TraceParams);
    }

    FHitResult Hit(ForceInit);
    if (HitResults.Num() > 0)
    {
        // HitResults 중에 중복(같은) Object의 HitResult 정보를 제거
        for (FHitResult& CurHitResult : HitResults)
        {
            auto Pred = [&CurHitResult](const FHitResult& Other)
            {
                return Other.HitObjectHandle == CurHitResult.HitObjectHandle;
            };

            if (!OutHitResults.ContainsByPredicate(Pred))
            {
                OutHitResults.Add(CurHitResult);
            }
        }

        // Hit의 가장 마지막 값을 Impact로 저장
        Hit = OutHitResults.Last();
    }
    else
    {
        // Hit의 결과 값을 기본 값으로 캐싱
        Hit.TraceStart = StartTrace;
        Hit.TraceEnd = EndTrace;
    }

    return Hit;
}
```

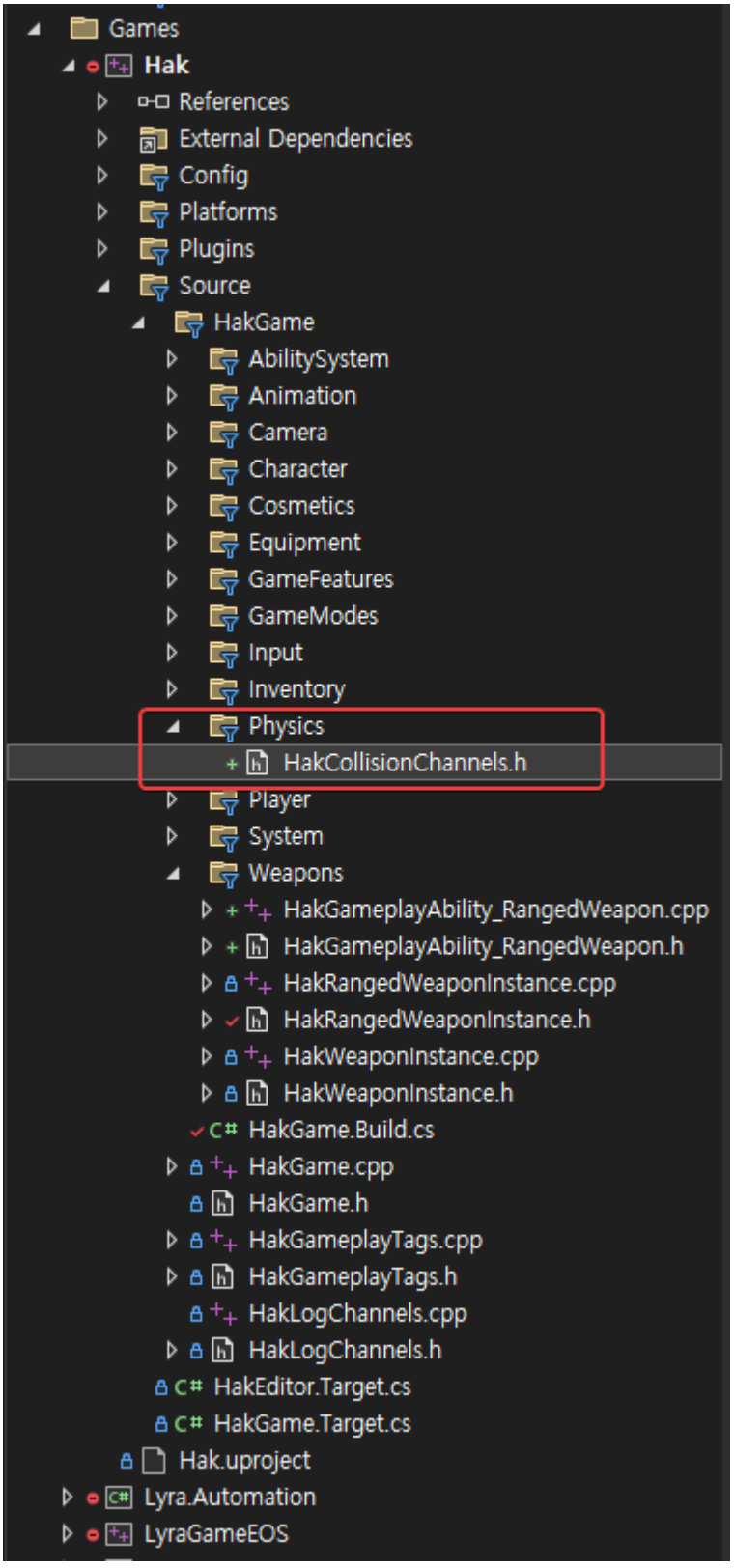
- [ ] AddAdditionalTraceIgnoreActors()

```cpp
void UHakGameplayAbility_RangedWeapon::AddAdditionalTraceIgnoreActors(FCollisionQueryParams& TraceParams) const
{
    if (AActor* Avatar = GetAvatarActorFromActorInfo())
    {
        TArray<AActor*> AttachedActors;

        // GetAttachedActors를 한번 보자:
        // - 해당 함수는 Recursively하게 모든 Actors를 추출한다
        // - 근데 왜 앞서 FindFirstPawnHitResult 이건 왜 한단계만 할까? ---
        Avatar->GetAttachedActors(AttachedActors);

        TraceParams.AddIgnoredActors(AttachedActors);
    }
}
```

- [ ] DetermineTraceChannel()

```cpp
ECollisionChannel UHakGameplayAbility_RangedWeapon::DetermineTraceChannel(FCollisionQueryParams& TraceParams, bool bIsSimulated) const
{
    return Hak_TraceChannel_Weapon;
}
```

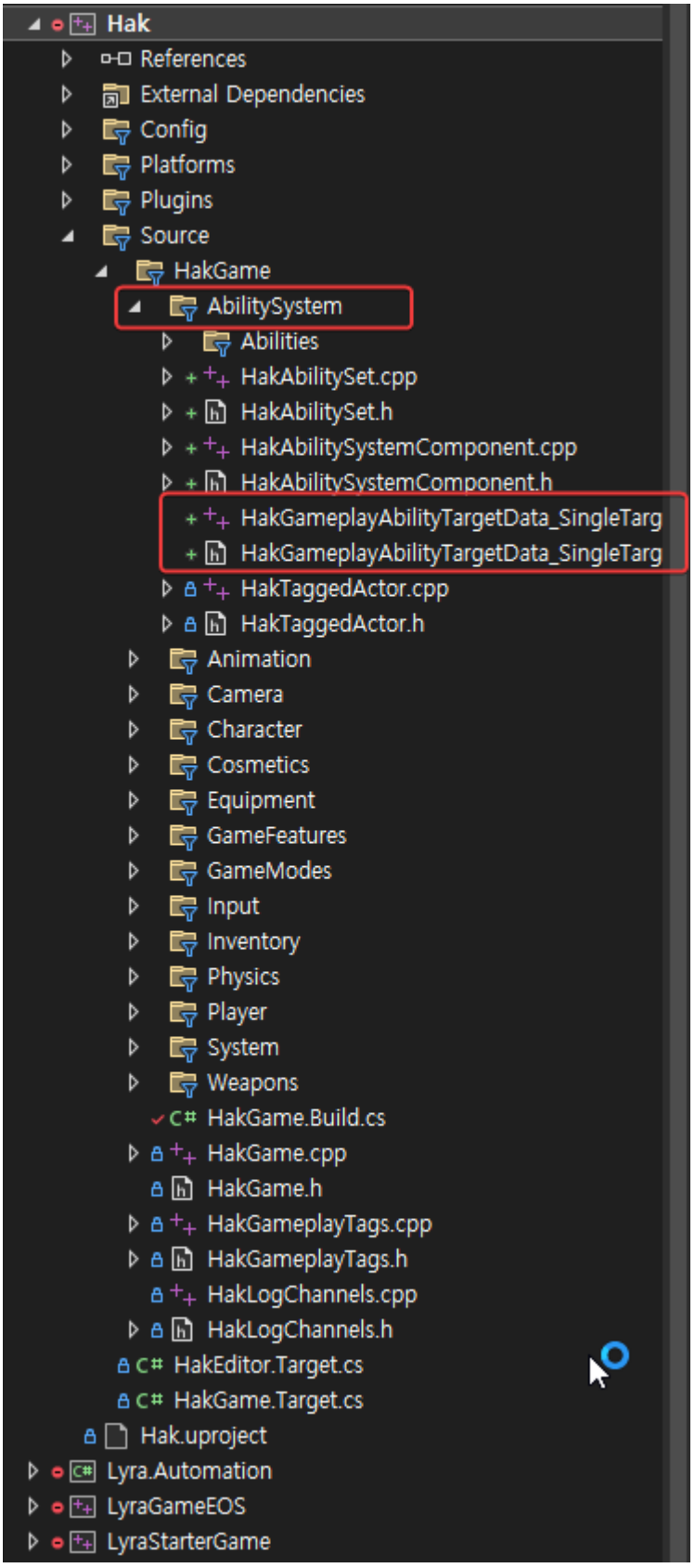
- [ ] HakCollisionChannels.h
  - [ ] 파일 추가:




☐ HakCollisionChannels 구현:

```cpp
#pragma once

#include "CoreMinimal.h"
#include "Engine/EngineTypes.h"

/** Weapon용 Collision Trace Channel */
#define Hak_TraceChannel_Weapon                ECC_GameTraceChannel2
```

- [ ] HakGameplayAbilityTargetData_SingleTargetHit:
  - [ ] 파일 생성:




☐ HakGameplayAbilityTargetData_SingleTargetHit 구현:

```h
#pragma once

#include "Abilities/GameplayAbilityTargetTypes.h"
#include "HakGameplayAbilityTargetData_SingleTarget.generated.h"

USTRUCT()
struct FHakGameplayAbilityTargetData_SingleTargetHit : public FGameplayAbilityTargetData_SingleTargetHit
{
    GENERATED_BODY()
public:
    FHakGameplayAbilityTargetData_SingleTargetHit()
        : CartridgeID(-1)
    {}

    virtual UScriptStruct* GetScriptStruct() const override
    {
        return FHakGameplayAbilityTargetData_SingleTargetHit::StaticStruct();
    }

    /** 탄약 ID (카트리지) */
    UPROPERTY()
    int32 CartridgeID;
};
```

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

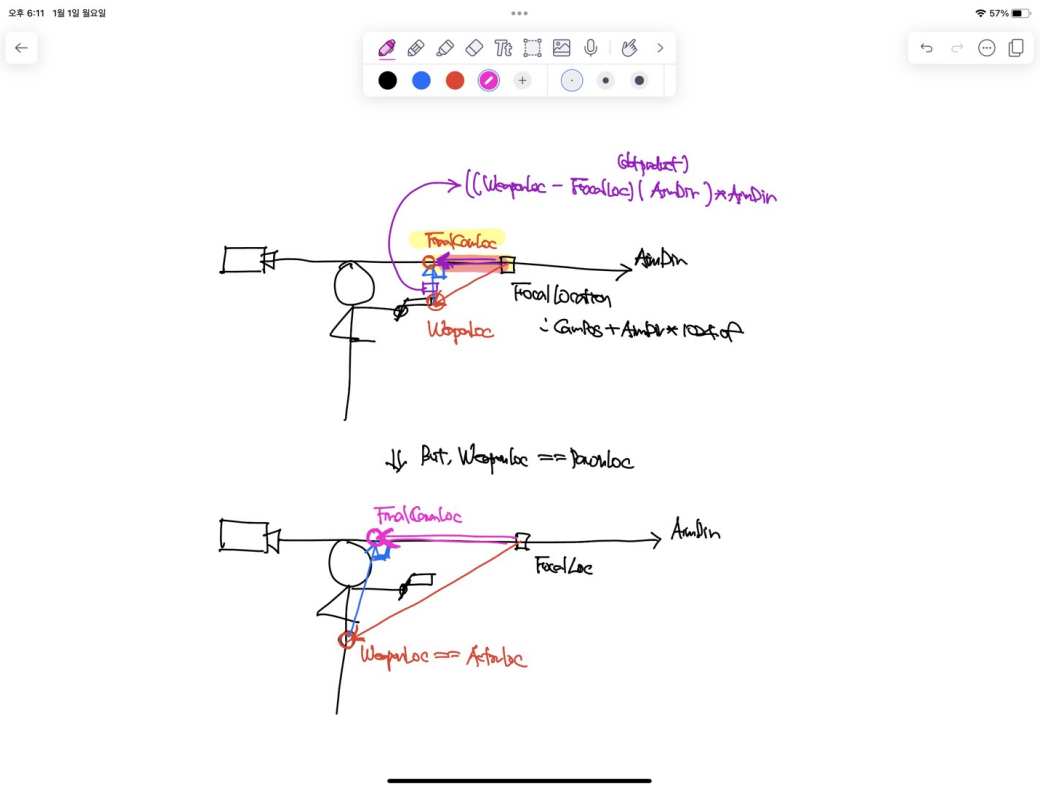
- [ ] OnTargetDataReadyCallback()

```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();
        }
    }
}
```

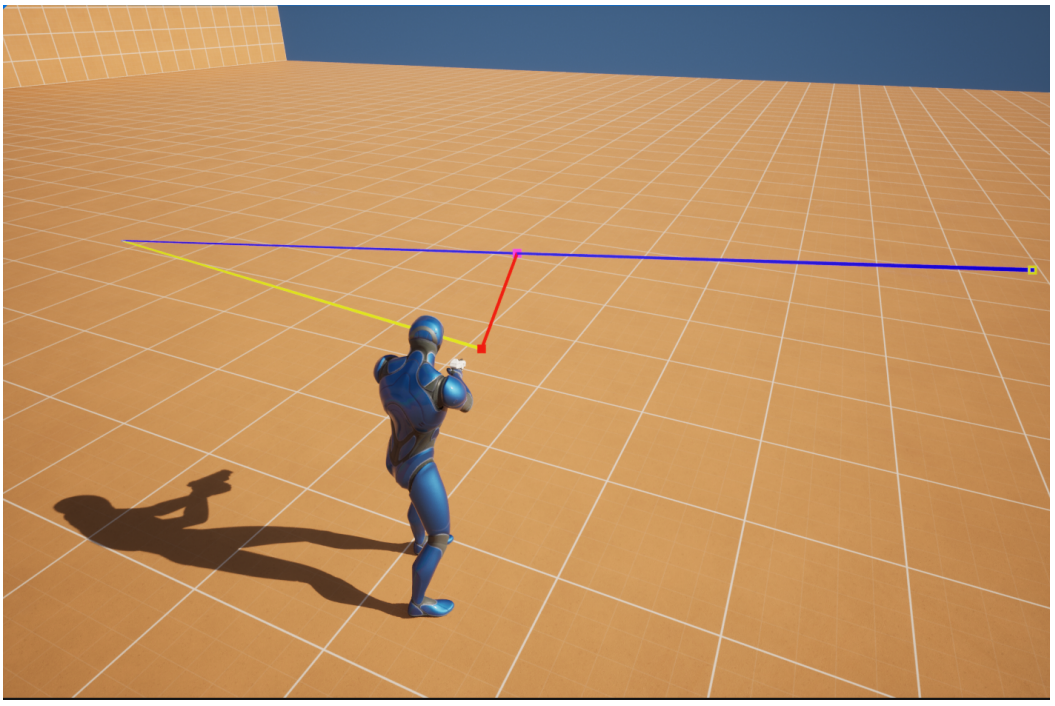
- [ ] GetTargetingTransform을 이해해보자:

```cpp
FTransform UHakGameplayAbility_RangedWeapon::GetTargetingTransform(APawn* SourcePawn, EHakAbilityTargetingSource Source)
{
    check(SourcePawn);
    check(Source == EHakAbilityTargetingSource::CameraTowardsFocus);

    // 참고로 아래 로직은 CameraTowardsFocus만 추출한 로직이다:
    // - 완전한 로직은 Lyra를 참고

    AController* Controller = SourcePawn->Controller;
    if (Controller == nullptr)
    {
        return FTransform();
    }

    // 매직넘버이다...
    double FocalDistance = 1024.0f;
    FVector FocalLoc;
    FVector CamLoc;
    FRotator CamRot;

    // PlayerController로부터, Location과 Rotation 정보를 가져옴
    APlayerController* PC = Cast<APlayerController>(Controller);
    check(PC);
    PC->GetPlayerViewPoint(CamLoc, CamRot);

    FVector AimDir = CamRot.Vector().GetSafeNormal();
    FocalLoc = CamLoc + (AimDir * FocalDistance);

    // WeaponLoc이 아닌 Pawn의 Loc이다
    const FVector WeaponLoc = GetWeaponTargetingSourceLocation();
    FVector FinalCamLoc = FocalLoc + (((WeaponLoc - FocalLoc) | AimDir) * AimDir);

#if 1
    {
        // WeaponLoc (사실상 ActorLoc)
        DrawDebugPoint(GetWorld(), WeaponLoc, 10.0f, FColor::Red, false, 60.0f);
        // CamLoc
        DrawDebugPoint(GetWorld(), CamLoc, 10.0f, FColor::Yellow, false, 60.0f);
        // FinalCamLoc
        DrawDebugPoint(GetWorld(), FinalCamLoc, 10.0f, FColor::Magenta, false, 60.0f);

        // (WeaponLoc - FocalLoc)
        DrawDebugLine(GetWorld(), FocalLoc, WeaponLoc, FColor::Yellow, false, 60.0f, 0, 2.0f);
        // (AimDir)
        DrawDebugLine(GetWorld(), CamLoc, FocalLoc, FColor::Blue, false, 60.0f, 0, 2.0f);

        // Project Direction Line
        DrawDebugLine(GetWorld(), WeaponLoc, FinalCamLoc, FColor::Red, false, 60.0f, 0, 2.0f);
    }
#endif

    // Camera -> Focus 계산 완료
    return FTransform(CamRot, FinalCamLoc);
}
```

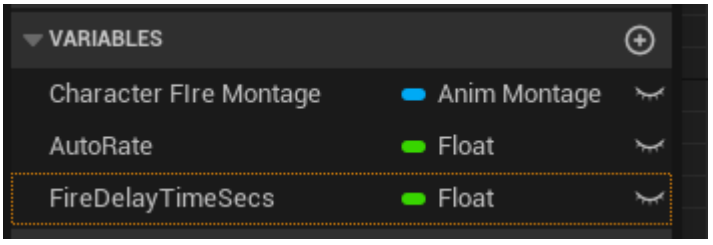
☐ 그림으로 이해해보자:



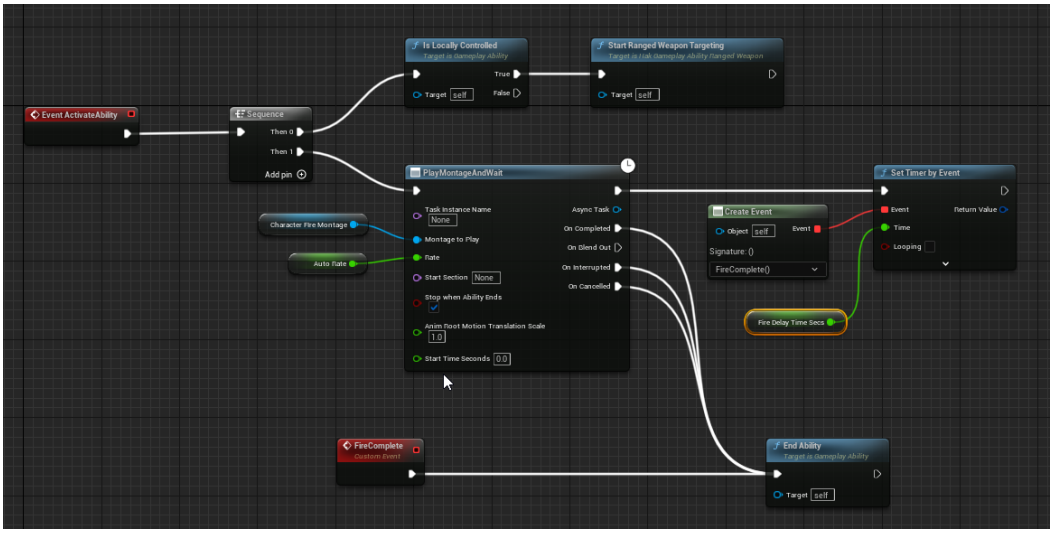
- [ ] 비주얼 디버거로 이해해보자:



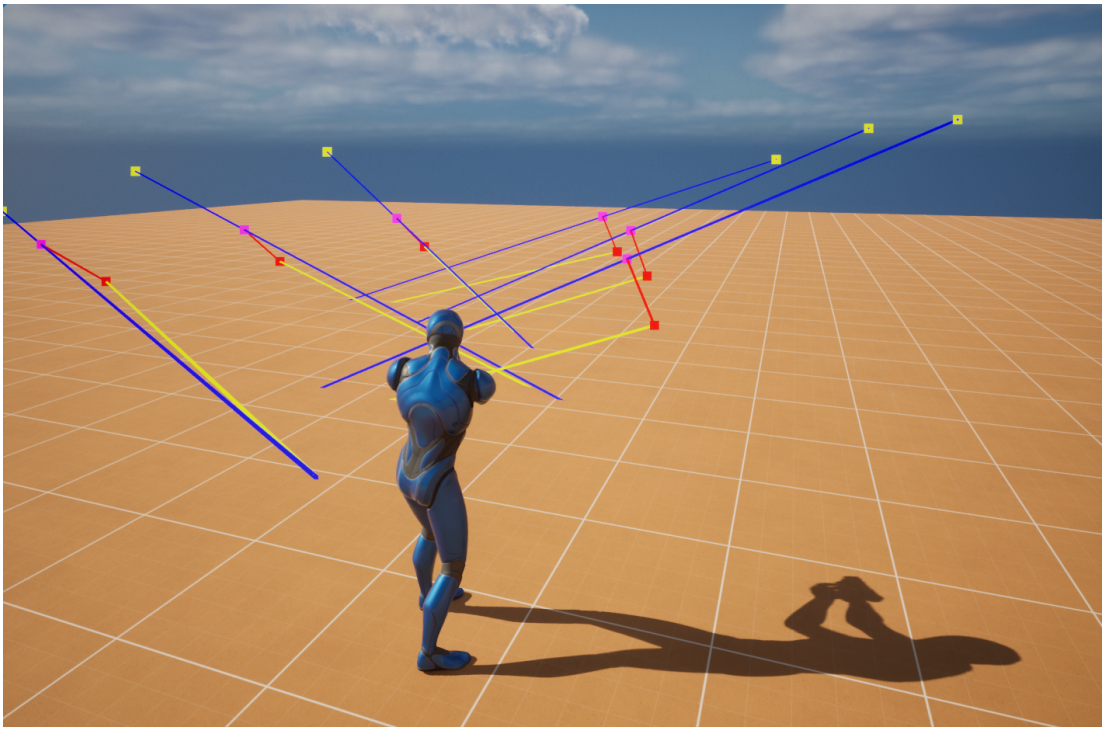
- [ ] GA_Weapon_Fire의 ActivateAbility를 완성시키자:
  - [ ] GA_Weapon_Fire 변수를 추가하자:




    - AutoRate == `1.0`
    - FireDelayTimeSecs = `0.1`
  - [ ] BP 로직 완성:




- [ ] 연속적으로 앞서 비주얼 디버깅 정보가 남는 것을 확인할 수 있다:

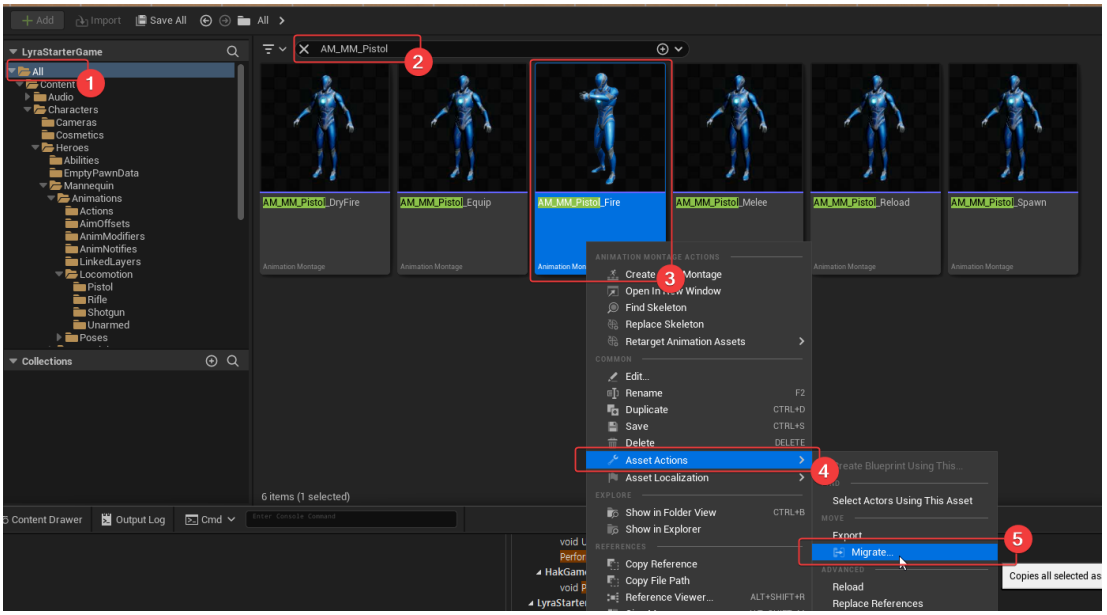


# GA_Weapon_Fire_Pistol:

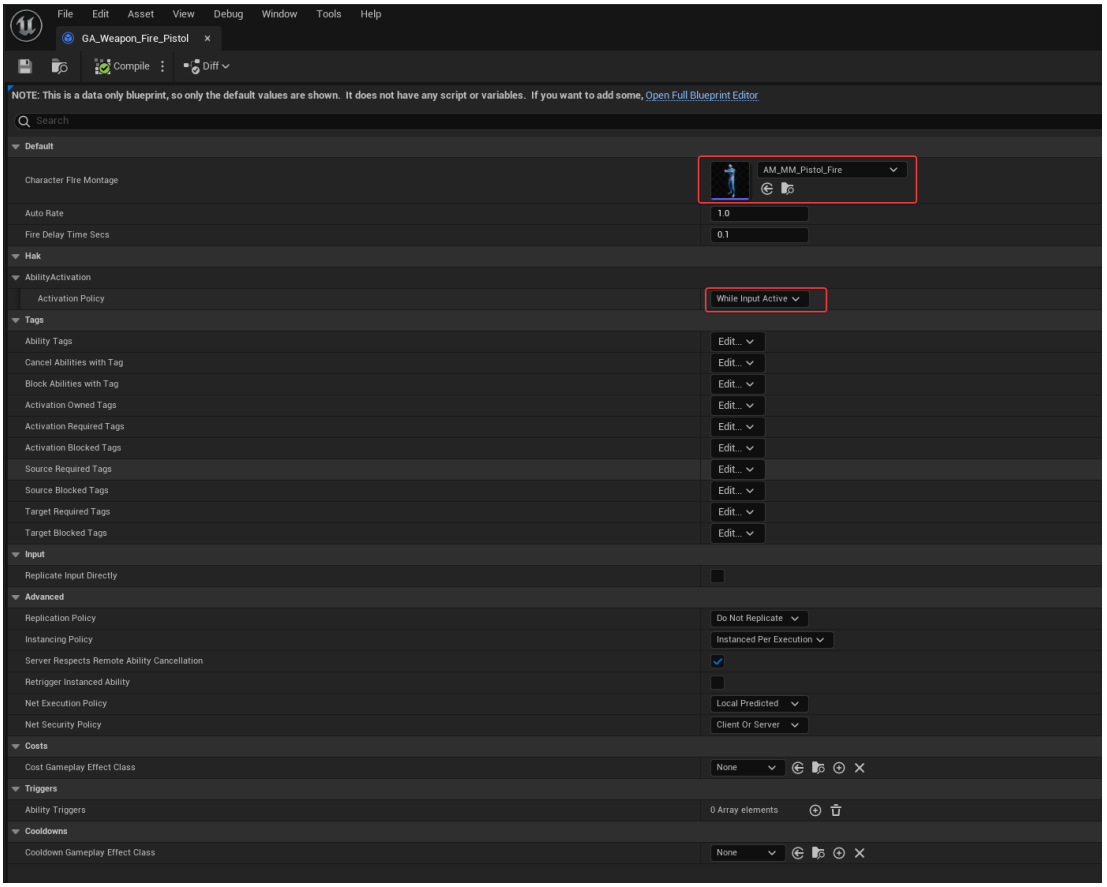
▼ **펼치기**

- 이제 Fire Animation을 추가할 차례이다:
  - 앞서, CharacterFireMontage의 변수를 GA_Weapon_Fire에 추가하였다
  - GA_Weapon_Fire_Pistol에 Montage를 추가해보자

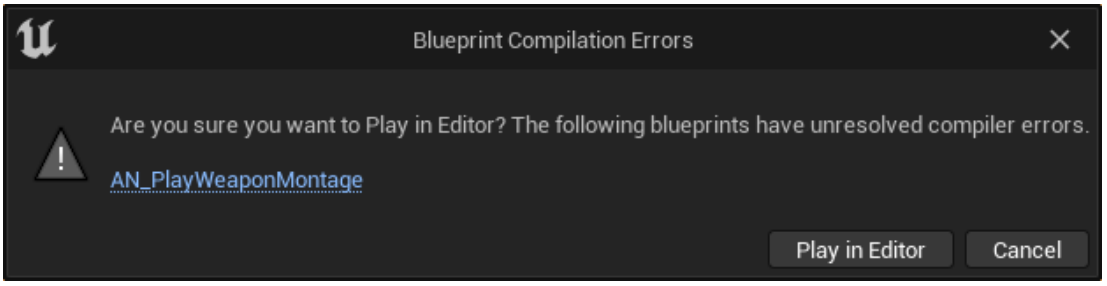
☐ AM_MM_Pistol_Fire을 Import하자:



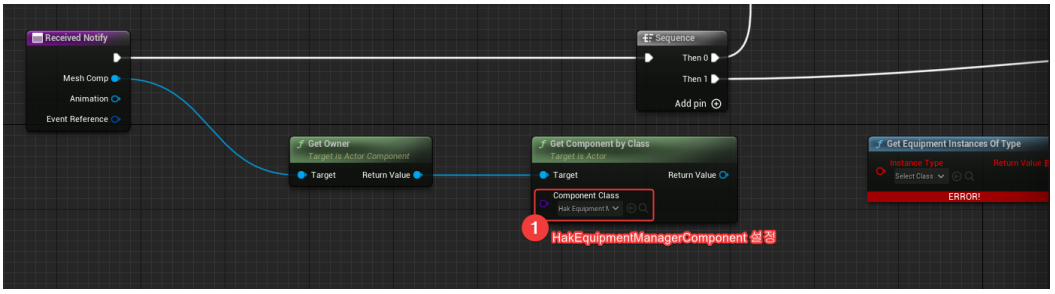

☐ GA_Weapon_Fire_Pistol을 채워 넣어주자:




☐ PIE를 누르면, 아래와 같이 AN_PlayWeaponMontage 에러가 있다고 뜬다:




- [ ] AN_PlayWeaponMontage을 제대로 컴파일 시키자:
  - [ ] GetComponentByClass 제대로 설정:




  - [ ] EquipmentManagerComponent::GetEquipmentInstancesOfType 구현:

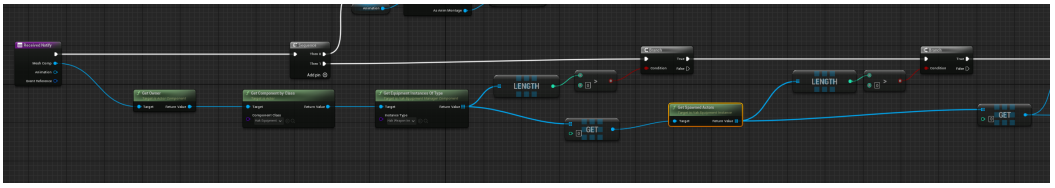
```cpp
TArray<UHakEquipmentInstance*> UHakEquipmentManagerComponent::GetEquipmentInstancesOfType(TSubclassOf<UHakEquipmentInstance> InstanceType) const
{
    TArray<UHakEquipmentInstance*> Results;

    // EquipmentList를 순회하며
    for (const FHakAppliedEquipmentEntry& Entry : EquipmentList.Entries)
    {
        if (UHakEquipmentInstance* Instance = Entry.Instance)
        {
            // InstanceType에 맞는 Class이면 Results에 추가하여 반환
            // - 우리의 경우, HakRangedWeaponInstance가 될거임
            if (Instance->IsA(InstanceType))
            {
                Results.Add(Instance);
            }
        }
    }
    return Results;
}
```

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

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

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

    UPROPERTY()
    FHakEquipmentList EquipmentList;
};
```

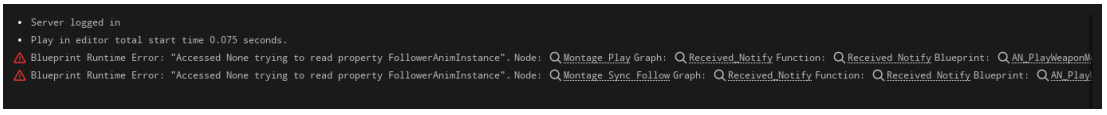
  - [ ] HakEquipmentInstance::GetSpawnedActors()

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

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

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

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

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

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

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

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

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

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

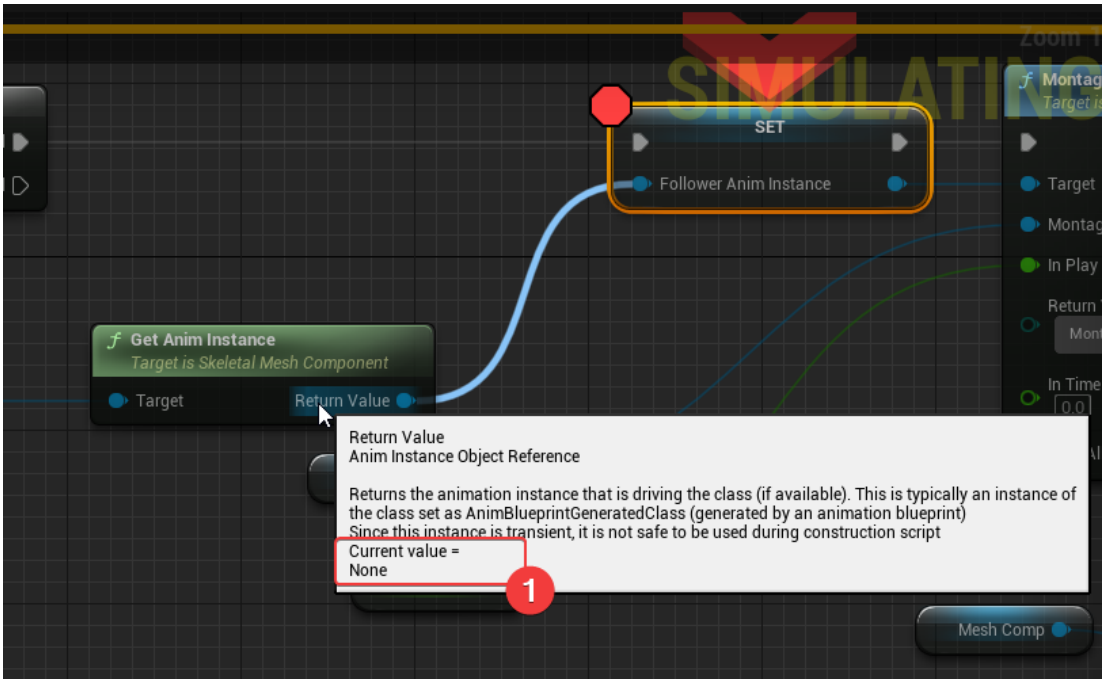
- [ ] BP 수정 완료:



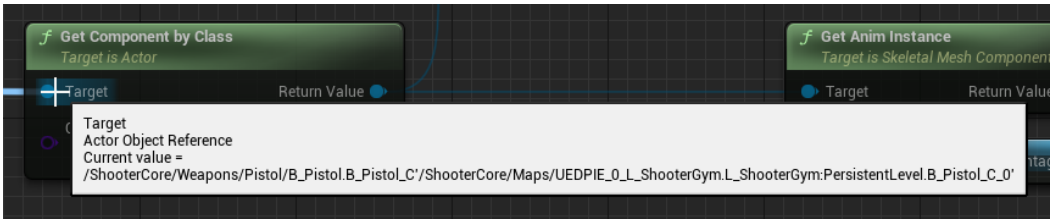

- [ ] 아래와 같이 MessageLog에 BP Runtime Error라고 뜬다:



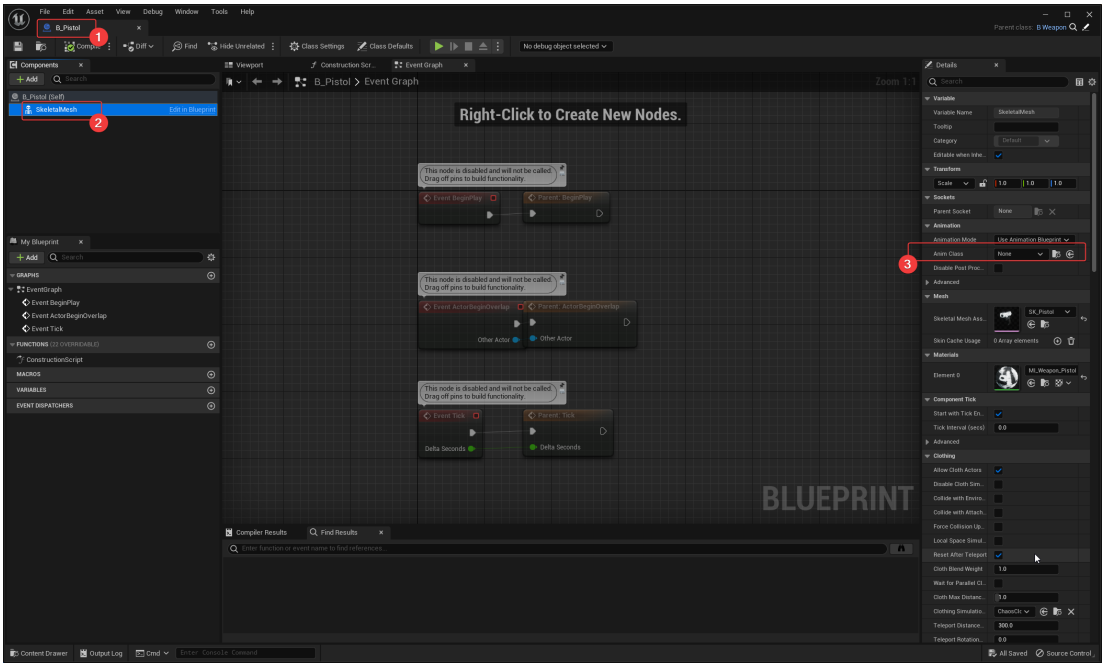

- [ ] AN_PlayWeaponMontage를 디버깅하면 아래와 같이 AnimInstance를 가져오는데 실패한다:



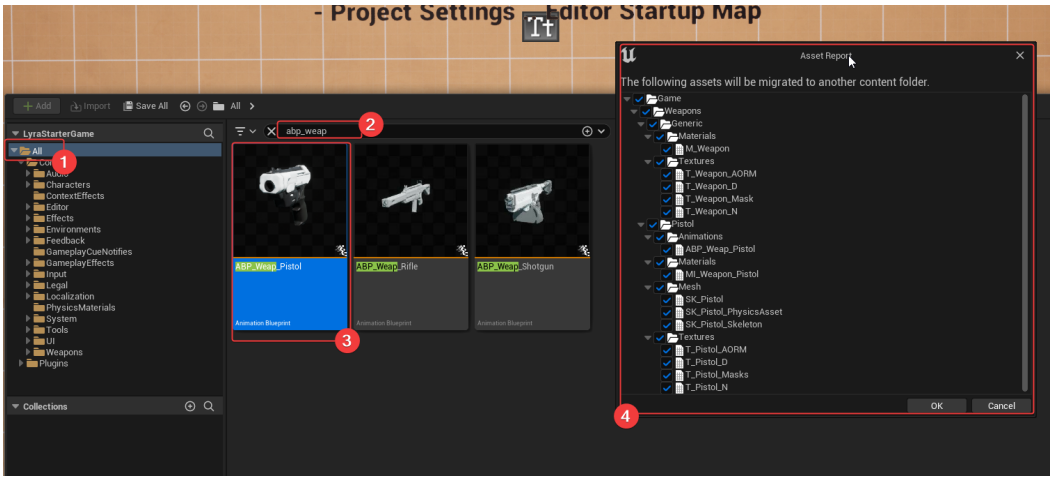

☐ SkeletalMesh가 누구인지 보니 아래와 같다:



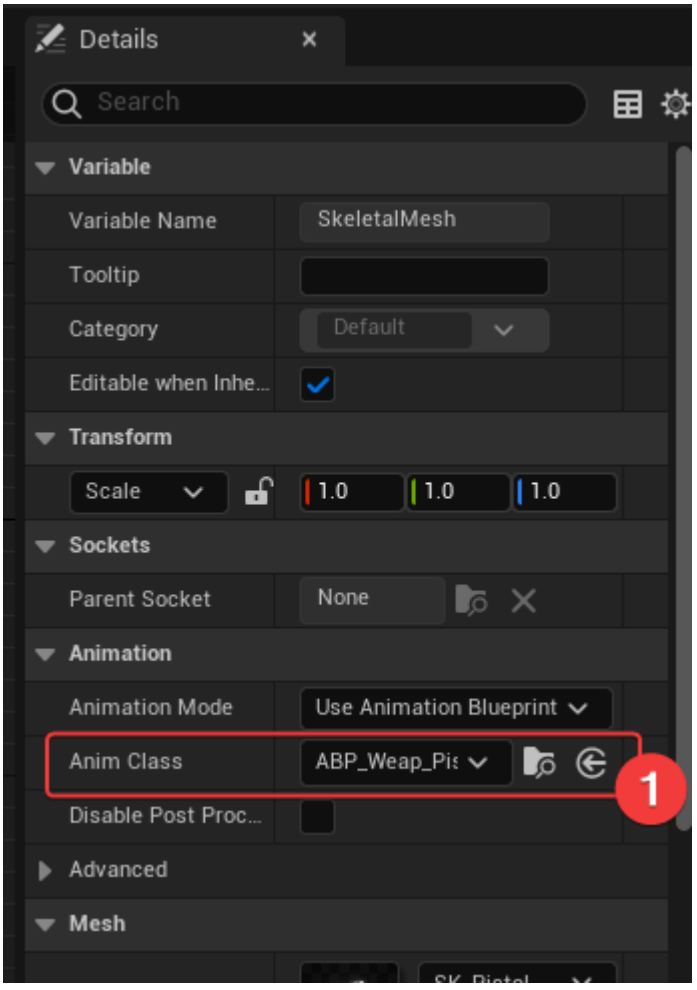

☐ B_Pistol을 보면, AnimInstance가 설정되어있지 않다:



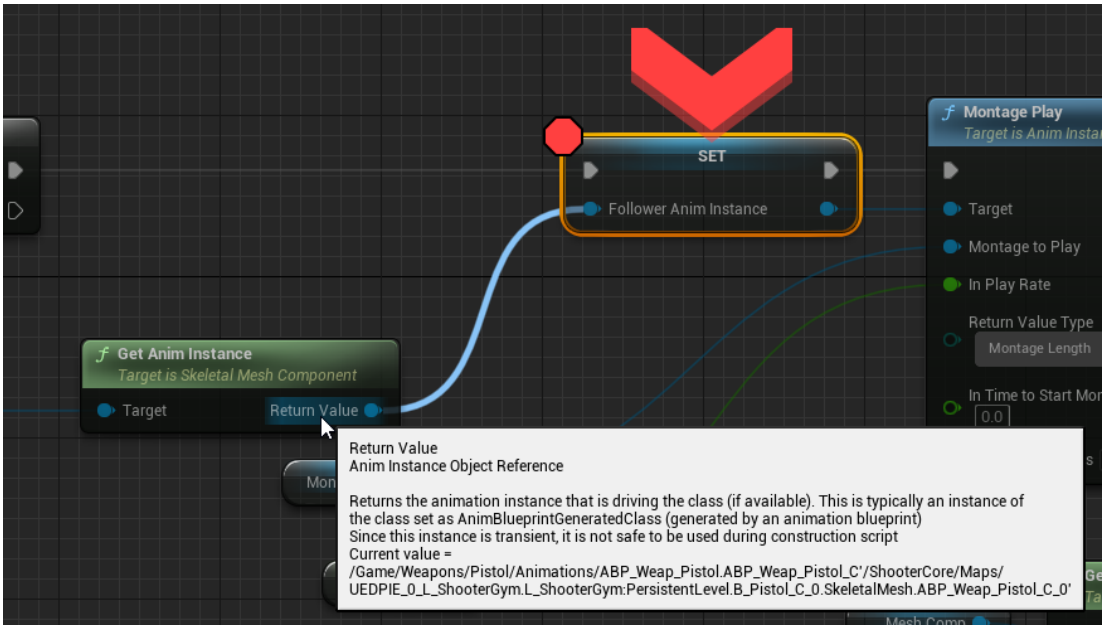
☐ ABP_Weap_Pistol을 Migrate하자:




☐ 아래와 같이 설정해주자:




☐ 잘 반환됨을 확인할 수 있다:

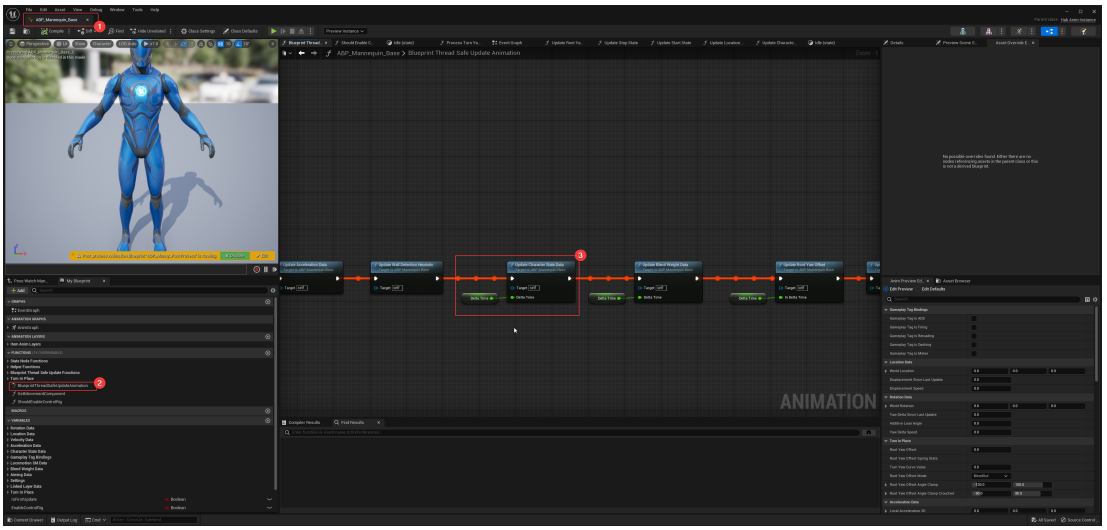


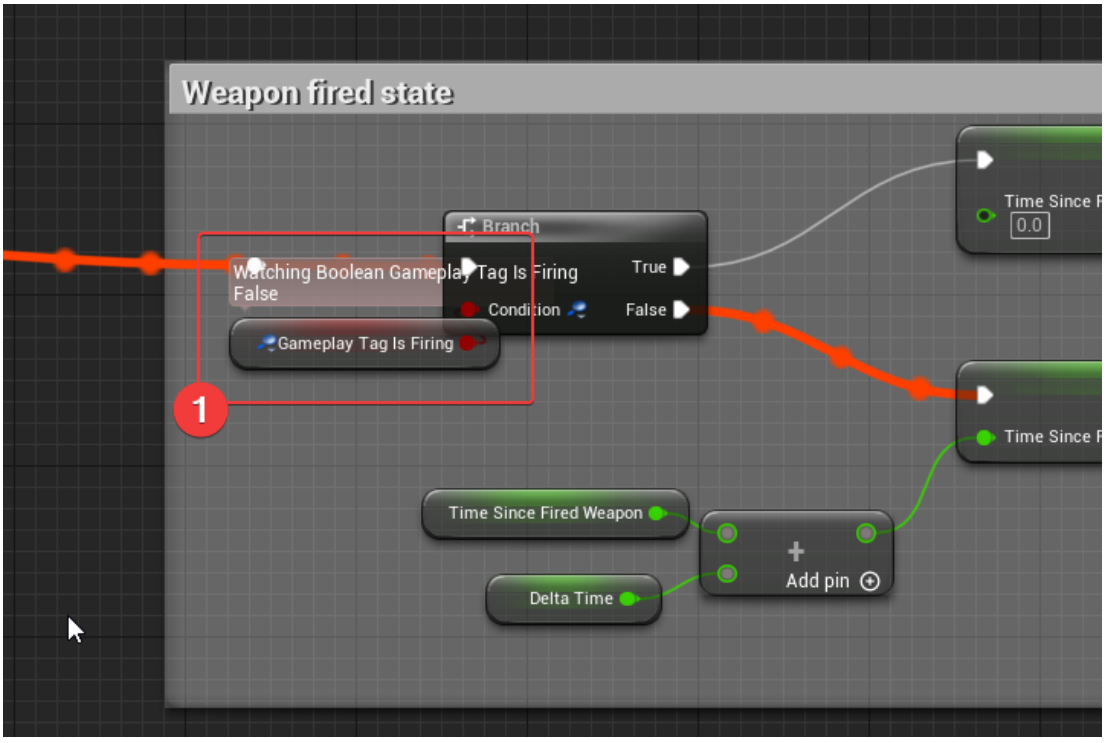

☐ 근데 약간 모션이 이상하다…

https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/055f2f18-fbac-46bd-9e6a-caf86dbad8ac/UnrealEditor_nh05tSu2XZ.mp4

- 총을 쏠 때, 바로 Aim 모션이 들어가야 할거 같은데 총만 반동하고 있다…

☐ 이는 **GameTag를 통해, AnimBP에 상태 변화에 영향을 주지 못하고 있기 때문**이다:






☐ 제대로 반영하기 위해, HakAnimInstance에 GameplayTagPropertyMap을 추가해야 한다:

☐ GameplayTagBlueprintPropertyMap을 HakAnimInstance에 추가하자:

```
#pragma once

#include "CoreMinimal.h"
#include "GameplayEffectTypes.h"
#include "Animation/AnimInstance.h"
#include "HakAnimInstance.generated.h"

UCLASS()
class UHakAnimInstance : public UAnimInstance
{
    GENERATED_BODY()
public:
    UHakAnimInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** 해당 속성값은 Lyra의 AnimBP에서 사용되는 값이므로 정의해주자 */
    UPROPERTY(BlueprintReadOnly, Category="Character State Data")
    float GroundDistance = -1.0f;

    /** GameplayTag와 AnimInstance의 속성값을 매핑해준다 */
    UPROPERTY(EditDefaultsOnly, Category = "GameplayTags")
    FGameplayTagBlueprintPropertyMap GameplayTagPropertyMap;
};
```

☐ 이를 제대로 반영하기 위해, NativeInitializeAnimation를 오버라이드 해야 한다:

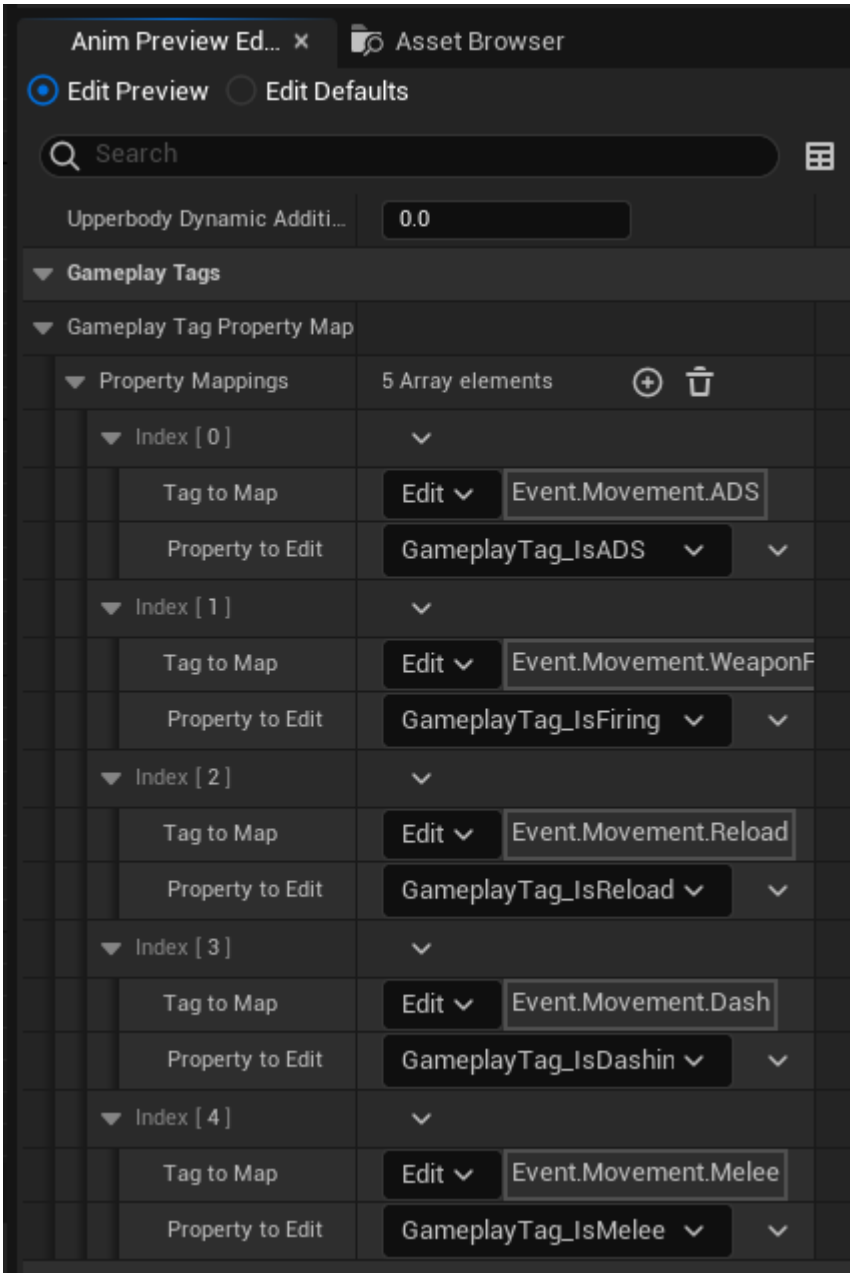
```h
#pragma once

#include "CoreMinimal.h"
#include "GameplayEffectTypes.h"
#include "Animation/AnimInstance.h"
#include "HakAnimInstance.generated.h"

UCLASS()
class UHakAnimInstance : public UAnimInstance
{
    GENERATED_BODY()
public:
    UHakAnimInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * UAnimInstance's interface
     */
    virtual void NativeInitializeAnimation() override;

    /** 해당 속성값은 Lyra의 AnimBP에서 사용되는 값이므로 정의해주자 */
    UPROPERTY(BlueprintReadOnly, Category="Character State Data")
    float GroundDistance = -1.0f;

    /** GameplayTag와 AnimInstance의 속성값을 매핑해준다 */
    UPROPERTY(EditDefaultsOnly, Category = "GameplayTags")
    FGameplayTagBlueprintPropertyMap GameplayTagPropertyMap;
};
```

```cpp
void UHakAnimInstance::NativeInitializeAnimation()
{
    Super::NativeInitializeAnimation();

    if (AActor* OwningActor = GetOwningActor())
    {
        if (UAbilitySystemComponent* ASC = UAbilitySystemGlobals::GetAbilitySystemComponentFromActor(OwningActor))
        {
            InitializeWithAbilitySystem(ASC);
        }
    }
}
```

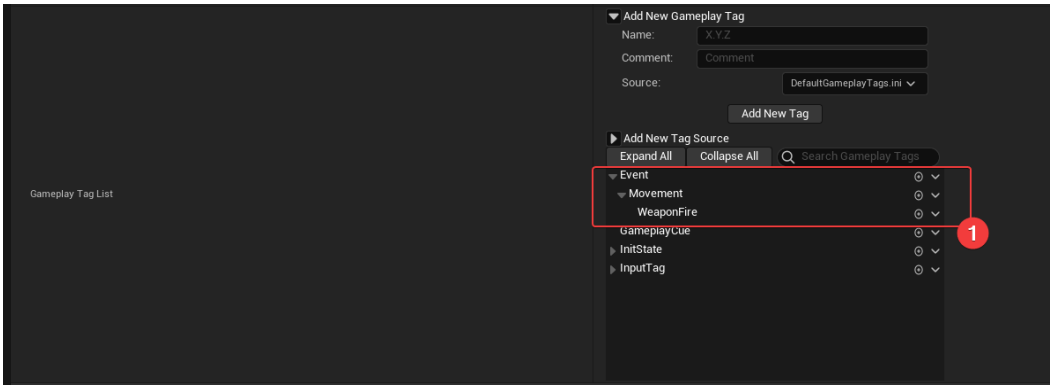
☐ UHakAnimInstance::InitializeWithAbilitySystem() 구현:

```cpp
void UHakAnimInstance::InitializeWithAbilitySystem(UAbilitySystemComponent* ASC)
{
    // ASC 내부 관리하는 GameplayTag와 AnimInstance의 멤버 Property와 Delegate를 연결하여, 값 변화에 대한 반영을 진행한다
    GameplayTagPropertyMap.Initialize(this, ASC);
}
```

☐ 그럼 기존 ABP_Mannequin_Base에 설정되어 있던 GameplayTagPropertyMap이 드러난다:

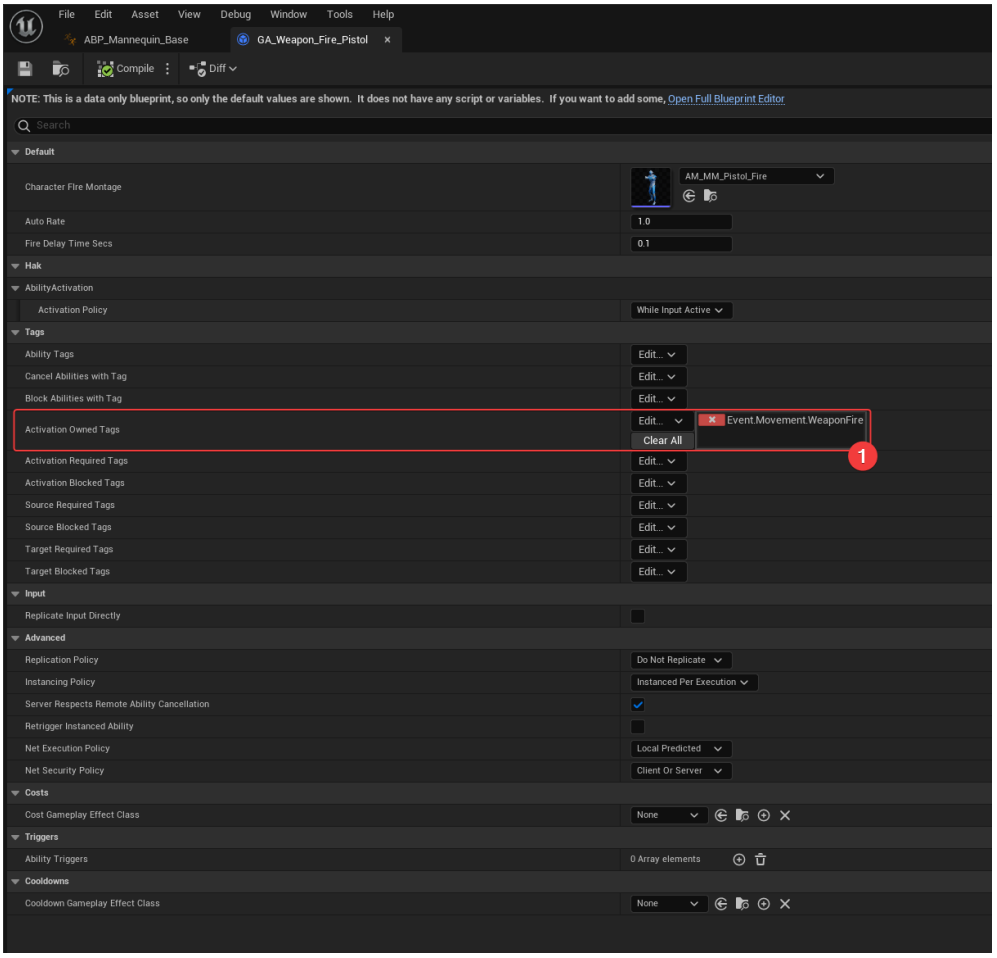



- [ ] 우리는 여기서 Event.Movement.WeaponFire GameplayTag를 적용해야 한다:
  - [ ] 우선 ProjectSettings에서 GameplayTag를 추가하자:



☐ 그럼 해당 GameplayTag는 ASC에서 어떻게 반영되어야 할까?

- 우리는 **GameplayAbility_RangedWeapon을 상속받는 GA_Weapon_Fire_Pistol에서 Activate될 때, Gameplay Tag를 적용하면 되겠다!**
- 아래와 같이, 해당 Ability가 활성화될 경우, GameplayTag도 활성화해주도록 하자:

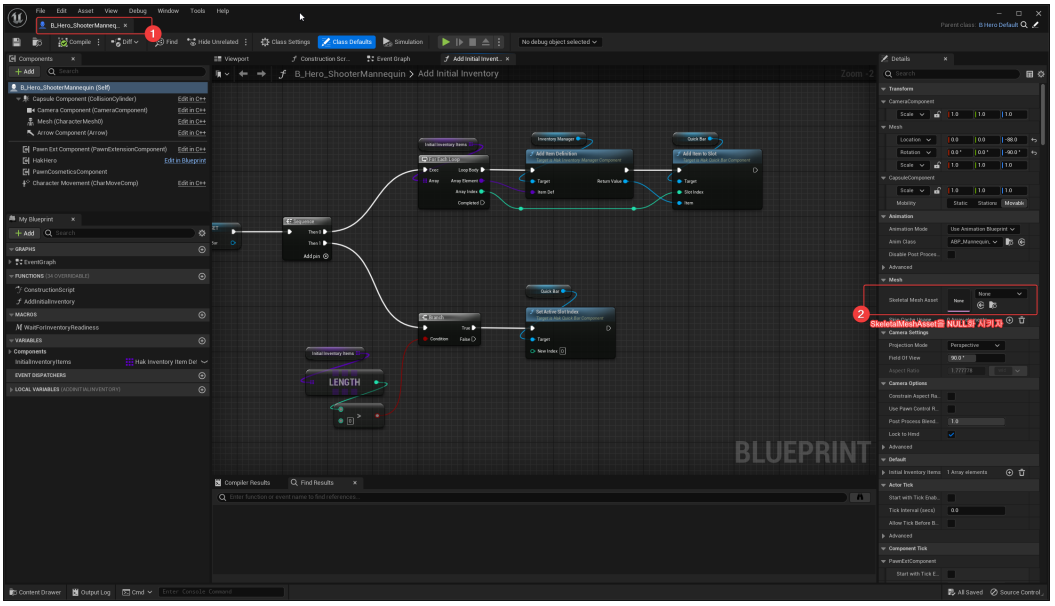



---

💣💣💣💣💣

☐ 여기까지 했을 경우, 안되는 경우, 해결 방법 두 가지:

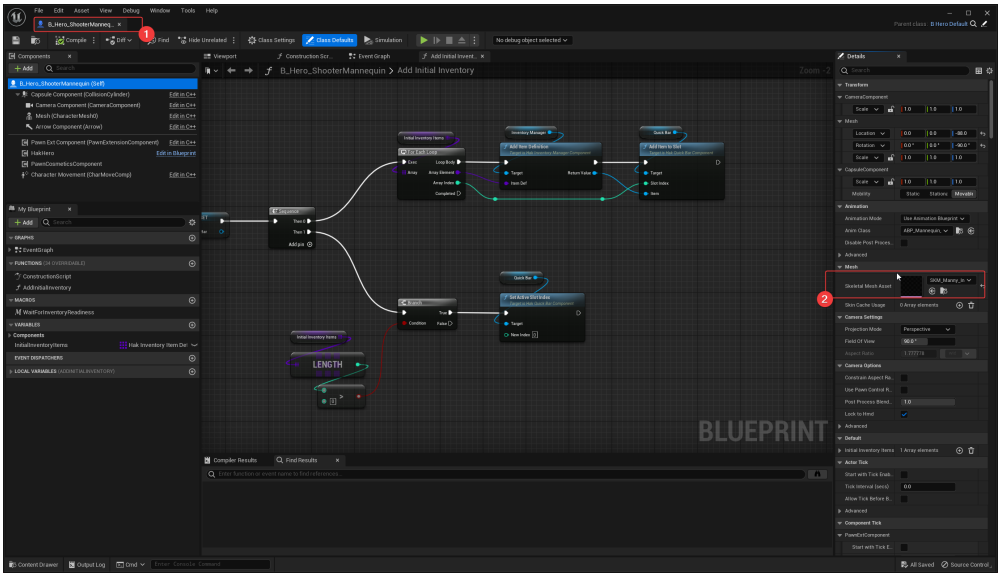
☐ **야매...** 방법:




- 해당 방법은 필자가 앞서 프로토 타이핑에서 작동했던 이유로 파악!

## ☐ 올바른 방법(? Lyra Way~):

☐ 다시 B_HeroShooterMannequinn의 SkeletalMeshAsset을 SKM_Manny_Inv로 설정:




☐ HakAbilitySystemComponent::InitAbilityActorInfo 오버라이드:

```cpp
#pragma once

#include "AbilitySystemComponent.h"
#include "GameplayAbilitySpec.h"
#include "HakAbilitySystemComponent.generated.h"

UCLASS()
class UHakAbilitySystemComponent : public UAbilitySystemComponent
{
    GENERATED_BODY()
public:
    UHakAbilitySystemComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
    * AbilitySystemComponent's interface
    */
    virtual void InitAbilityActorInfo(AActor* InOwnerActor, AActor* InAvatarActor) override;

    /**
    * member methods
    */
    void AbilityInputTagPressed(const FGameplayTag& InputTag);
    void AbilityInputTagReleased(const FGameplayTag& InputTag);
    void ProcessAbilityInput(float DeltaTime, bool bGamePaused);

    /** Ability Input 처리할 Pending Queue */
    TArray<FGameplayAbilitySpecHandle> InputPressedSpecHandles;
    TArray<FGameplayAbilitySpecHandle> InputReleasedSpecHandles;
    TArray<FGameplayAbilitySpecHandle> InputHeldSpecHandles;
};
```

1

```cpp
void UHakAbilitySystemComponent::InitAbilityActorInfo(AActor* InOwnerActor, AActor* InAvatarActor)
{
    FGameplayAbilityActorInfo* ActorInfo = AbilityActorInfo.Get();
    check(ActorInfo);
    check(InOwnerActor);

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

    Super::InitAbilityActorInfo(InOwnerActor, InAvatarActor);

    if (bHasNewPawnAvatar)
    {
        if (UHakAnimInstance* LyraAnimInst = Cast<UHakAnimInstance>(ActorInfo->GetAnimInstance()))
        {
            LyraAnimInst->InitializeWithAbilitySystem(this);
        }
    }
}
```

1 InitAbilityActorInfo() 호출할 때마다 AnimInstance를 InitializeWithAbilitySystem() 강제 호출해준다

💣💣💣💣💣

- [ ] 아래의 코드를 활성화해서 총탄 궤적을 예상해보자:

https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/c35911cc-cce7-4251-bd60-5e4fb1545fa4/Untitled.mp4