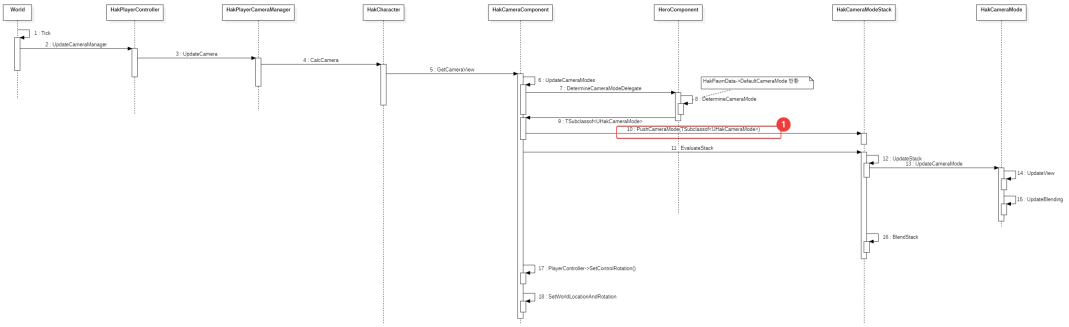
8주차 (2023.11.13)

CameraModeStack

▼ **펼치기**

☐ PushCameraMode를 구현해보자:



- 아래와 같이 이전에 우리는 주석 처리를 하였다:

```cpp
void UHakCameraComponent::UpdateCameraModes()
{
    check(CameraModeStack);

    // DetermineCameraModeDelegate는 CameraMode Class를 반환한다:
    // - 해당 delegate는 HeroComponent의 멤버 함수로 바인딩되어 있다
    if (DetermineCameraModeDelegate.IsBound())
    {
        if (const TSubclassOf<UHakCameraMode> CameraMode = DetermineCameraModeDelegate.Execute())
        {
            // CameraModeStack->PushCameraMode(CameraMode);
        }
    }
}
```

- 아래와 같이 PushCameraMode()를 .h/cpp에 추가해주자:

```cpp
#pragma once

#include "HakCameraMode.generated.h"

/** Camera Blending 대상 유닛 */
UCLASS(Abstract, NotBlueprintable)
class UHakCameraMode : public UObject
{
    GENERATED_BODY()
public:
    UHakCameraMode(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
};

/** Camera Blending을 담당하는 객체 */
UCLASS()
class UHakCameraModeStack : public UObject
{
    GENERATED_BODY()
public:
    UHakCameraModeStack(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * member methods
     */
    UHakCameraMode* GetCameraModeInstance(TSubclassOf<UHakCameraMode>& CameraModeClass);
    void PushCameraMode(TSubclassOf<UHakCameraMode>& CameraModeClass);

    /**
     * member variables
     */

    /** 생성된 CameraMode를 관리 */
    UPROPERTY()
    TArray<TObjectPtr<UHakCameraMode>> CameraModeInstances;

    /** Camera Matrix Blend 업데이트 진행 큐 */
    UPROPERTY()
    TArray<TObjectPtr<UHakCameraMode>> CameraModeStack;
};
```

1

```cpp
void UHakCameraModeStack::PushCameraMode(TSubclassOf<UHakCameraMode>& CameraModeClass)
{
    if (!CameraModeClass)
    {
        return;
    }

    UHakCameraMode* CameraMode = GetCameraModeInstance(CameraModeClass);
}
```

1

☐ GetCameraModeInstance를 구현하자:

```cpp
UHakCameraMode* UHakCameraModeStack::GetCameraModeInstance(TSubclassOf<UHakCameraMode>& CameraModeClass)
{
    check(CameraModeClass);

    // CameraModeInstances에서 먼저 생성되어있는지 확인 후, 반환한다:
    for (UHakCameraMode* CameraMode : CameraModeInstances)
    {
        // CameraMode는 UClass를 비교한다:
        // - 즉, CameraMode는 클래스 타입에 하나만 생기게된다
        if ((CameraMode != nullptr) && (CameraMode->GetClass() == CameraModeClass))
        {
            return CameraMode;
        }
    }

    // CameraModeClass에 알맞는 CameraMode의 인스턴스가 없다면 생성한다:
    UHakCameraMode* NewCameraMode = NewObject<UHakCameraMode>(GetOuter(), CameraModeClass, NAME_None, RF_NoFlags);
    check(NewCameraMode);

    // 여기서 우리는 CameraModeInstances는 CameraModeClass에 맞는 인스턴스를 가지고(관리하는) 컨테이너이라는 것을 알 수 있다
    CameraModeInstances.Add(NewCameraMode);

    return NewCameraMode;
}
```

☐ CameraMode에 멤버 변수들을 추가해주자:

```cpp
/** Camera Blending 대상 유닛 */
UCLASS(Abstract, NotBlueprintable)
class UHakCameraMode : public UObject
{
    GENERATED_BODY()
public:
    UHakCameraMode(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /** 얼마동안 Blend를 진행할지 시간을 의미 */
    UPROPERTY(EditDefaultsOnly, Category="Blending")
    float BlendTime;

    /** 선형적인 Blend 값 [0, 1] */
    float BlendAlpha;

    /**
     * 해당 CameraMode의 최종 Blend 값
     * 앞서 BlendAlpha의 선형 값을 매핑하여 최종 BlendWeight를 계산 (코드를 보며, 이해해보자)
     */
    float BlendWeight;
};
```

1

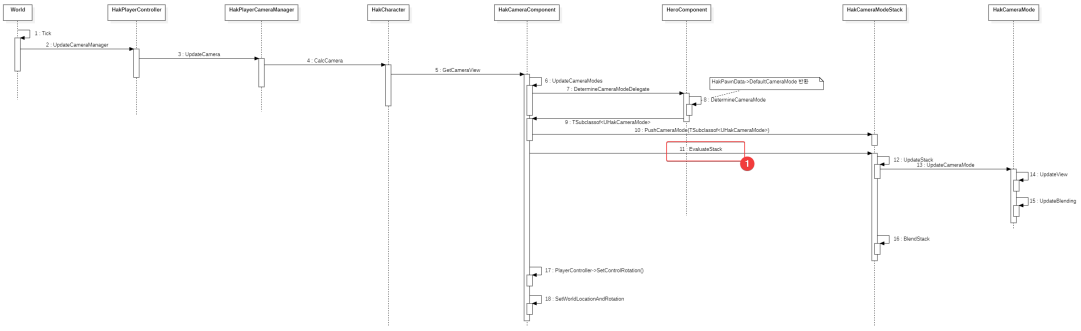
☐ PushCameraMode를 마저 구현해보자:

```cpp
void UHakCameraModeStack::PushCameraMode(TSubclassOf<UHakCameraMode>& CameraModeClass)
{
    if (!CameraModeClass)
    {
        return;
    }

    UHakCameraMode* CameraMode = GetCameraModeInstance(CameraModeClass);
    check(CameraMode);

    int32 StackSize = CameraModeStack.Num();
    if ((StackSize > 0) && (CameraModeStack[0] == CameraMode))
    {
        // CameraModeStack[0] 가장 최근에 이미 CameraMode가 Stacking되었으므로 그냥 리턴!
        return;
    }

    // ExistingStackIndex는 CameraModeStack에서 CameraMode에 맞는 Index를 찾음
    // ExistingStackContribution은 위에서 아래로 최종 BlendWeight 값을 찾기 위해 초기값으로 1.0으로 설정
    int32 ExistingStackIndex = INDEX_NONE;
    float ExistingStackContribution = 1.0f;

    /**
     * BlendWeight    |    ExistingStackCOntribution    |    ExistingStackCOntribution (accumulated)
     * 0.1f           |    (1.0f) * 0.1f = 0.1f         |    (1.0f - 0.1f) = 0.9f
     * 0.3f           |    (0.9f) * 0.3f = 0.27f        |    (1.0f - 0.3f) * 0.9f = 0.63f
     * 0.6f           |    (0.63f) * 0.6f = 0.378f      |    (1.0f - 0.6f) * 0.63f = 0.252f
     * 1.0f           |    (0.252f) * 1.0f = 0.252f     |
     *                |    0.1f + 0.27f + 0.378f + 0.252f = 1.0f!
     */
    for (int32 StackIndex = 0; StackIndex < StackSize; ++StackIndex)
    {
        if (CameraModeStack[StackIndex] == CameraMode)
        {
            ExistingStackIndex = StackIndex;
            // BlendWeight를 CameraMode에 추가해주자:
            // - 여기서 ExistingStackContribution 우리가 찾는 CameraMode를 찾았으니깐, 누적된 BlendWeight와 함께 BlendWeight을 곱해줘야, 루프를 빠져나온다
            ExistingStackContribution *= CameraMode->BlendWeight;
            break;
        }
        else
        {
            // 당연히 우리가 원하는 CamearMode가 아니니깐, InvBlendWeight = (1.0 - BlendWeight)를 곱해줘야, 값이 누적되겠징?
            ExistingStackContribution *= (1.0f - CameraModeStack[StackIndex]->BlendWeight);
        }
    }
    // 해당 루프의 동작 원리는 앞서 적어놓은 표를 확인해보며 이해해보자.

    // 우리는 CameraMode를 Top으로 반영하기 위해, 당연히 중간에 있었다면, 제거하여 다시 Push 해줘야 한다
    if (ExistingStackIndex != INDEX_NONE)
    {
        CameraModeStack.RemoveAt(ExistingStackIndex);
        StackSize--;
    }
    else
    {
        // 없었으니깐 당연히 Contribution은 0으로 세팅해줘야겠징?
        ExistingStackContribution = 0.0f;
    }

    // BlendTime이 0보다 크다는 것은 Blend을 얼마 시간동안 진행함을 의미 따라서, ExistingStackContribution을 적용
    // - 따라서 Blend하지 않는다면, BlendWeight를 1.0을 넣어 새로 넣는 CameraMode만 적용할 것이다
    const bool bShouldBlend = ((CameraMode->BlendTime > 0.f) && (StackSize > 0));
    const float BlendWeight = (bShouldBlend ? ExistingStackContribution : 1.0f);
    CameraMode->BlendWeight = BlendWeight;

    // 흠.. Array를 Stack처럼 사용하는것은 알지만, Index 0에 넣는건 비효율적인데..
    // - Push, Pop 메서드와 같이그냥 last에 넣는게... 어떨까 싶음
    CameraModeStack.Insert(CameraMode, 0);

    // 항상 마지막에는 100%로 해줘야? (아직은 이해가 안됨)
    // - 천천히 로직 보면 이해될거임
    CameraModeStack.Last()->BlendWeight = 1.0f;
}
```

☐ PushCameraMode의 주석을 풀어주자:

```cpp
void UHakCameraComponent::UpdateCameraModes()
{
    check(CameraModeStack);

    // DetermineCameraModeDelegate는 CameraMode Class를 반환한다:
    // - 해당 delegate는 HeroComponent의 멤버 함수로 바인딩되어 있다
    if (DetermineCameraModeDelegate.IsBound())
    {
        if (TSubclassOf<UHakCameraMode> CameraMode = DetermineCameraModeDelegate.Execute())
        {
            CameraModeStack->PushCameraMode(CameraMode);
        }
    }
}
```

- [ ] EvaluateStack을 구현해보자:



- [ ] GetCameraView()에서 EvaluateStack 관련 코드를 추가하자:

```cpp
void UHakCameraComponent::GetCameraView(float DeltaTime, FMinimalViewInfo& DesiredView)
{
    check(CameraModeStack);

    // 해당 함수는 HeroComponent에서 PawnData에서 DefaultCameraMode를 가져와서 CameraModeStack에 추가하여, CameraMode 블랜딩 준비한다
    UpdateCameraModes();

    FHakCameraModeView CameraModeView;
    CameraModeStack->EvaluateStack(DeltaTime, CameraModeView);
}
```

- [ ] UpdateCameraMode() 함수에 대한 간단한 정리 설명
- [ ] FHakCameraModeView를 구현하자:

```cpp
#pragma once

#include "HakCameraMode.generated.h"

/**
 * FHakCameraModeView
 */
struct FHakCameraModeView
{
    FHakCameraModeView();

    FVector Location;
    FRotator Rotation;
    FRotator ControlRotation;
    float FieldOfView;
};
```

```cpp
#include "HakCameraMode.h"
#include "HakPlayerCameraManager.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakCameraMode)

FHakCameraModeView::FHakCameraModeView()
    : Location(ForceInit)
    , Rotation(ForceInit)
    , ControlRotation(ForceInit)
    , FieldOfView(HAK_CAMERA_DEFAULT_FOV)
{
}

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

UHakCameraModeStack::UHakCameraModeStack(const FObjectInitializer& ObjectInitializer)
```

1 HAK_CAMERA_DEFAULT_FOV 정의 포함

☐ EvaluteStack() .h/.cpp에 정의

```cpp
/** Camera Blending을 담당하는 객체 */
UCLASS()
class UHakCameraModeStack : public UObject
{
    GENERATED_BODY()
public:
    UHakCameraModeStack(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * member methods
     */
    UHakCameraMode* GetCameraModeInstance(TSubclassOf<UHakCameraMode>& CameraModeClass);
    void PushCameraMode(TSubclassOf<UHakCameraMode>& CameraModeClass);
    void EvaluateStack(float DeltaTime, FHakCameraModeView& OutCameraModeView);

    /**
     * member variables
     */

    /** 생성된 CameraMode를 관리 */
    UPROPERTY()
    TArray<TObjectPtr<UHakCameraMode>> CameraModeInstances;

    /** Camera Matrix Blend 업데이트 진행 큐 */
    UPROPERTY()
    TArray<TObjectPtr<UHakCameraMode>> CameraModeStack;
};
```

1

```cpp
void UHakCameraModeStack::EvaluateStack(float DeltaTime, FHakCameraModeView& OutCameraModeView)
{

}
```

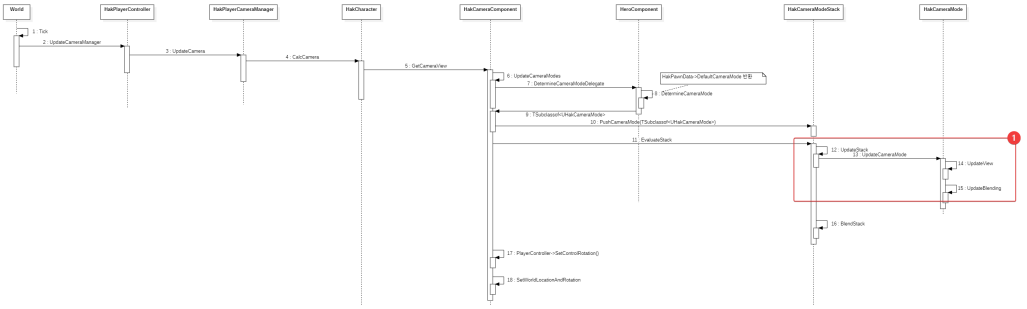
- Evaluate는 **<u>크게 2개로 구분</u>**된다:

```cpp
void UHakCameraModeStack::EvaluateStack(float DeltaTime, FHakCameraModeView& OutCameraModeView)
{
    UpdateStack(DeltaTime);

    BlendStack(OutCameraModeView);
}
```

1

2

☐ UpdateStack()을 구현해보자:



☐ UpdateStack()의 .h/.cpp 정의하자:

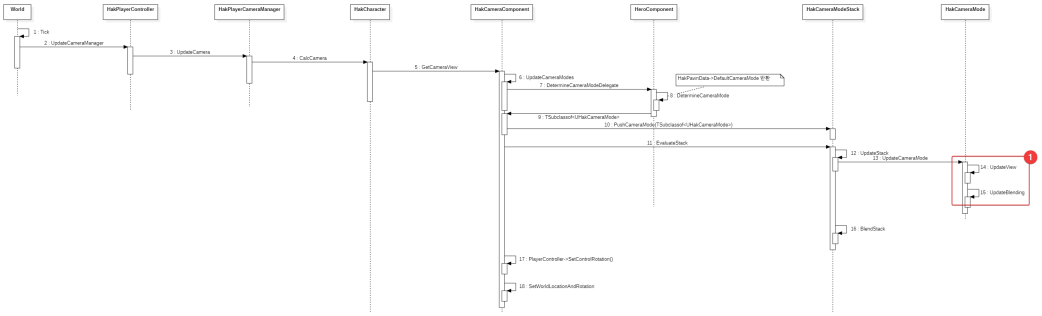
```cpp
/** Camera Blending을 담당하는 객체 */
UCLASS()
class UHakCameraModeStack : public UObject
{
    GENERATED_BODY()
public:
    UHakCameraModeStack(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * member methods
     */
    UHakCameraMode* GetCameraModeInstance(TSubclassOf<UHakCameraMode>& CameraModeClass);
    void PushCameraMode(TSubclassOf<UHakCameraMode>& CameraModeClass);
    void UpdateStack(float DeltaTime);
    void EvaluateStack(float DeltaTime, FHakCameraModeView& OutCameraModeView);

    /**
     * member variables
     */

    /** 생성된 CameraMode를 관리 */
    UPROPERTY()
    TArray<TObjectPtr<UHakCameraMode>> CameraModeInstances;

    /** Camera Matrix Blend 업데이트 진행 큐 */
    UPROPERTY()
    TArray<TObjectPtr<UHakCameraMode>> CameraModeStack;
};
```

☐ UpdateStack의 대략적인 구현:

```cpp
void UHakCameraModeStack::UpdateStack(float DeltaTime)
{
    const int32 StackSize = CameraModeStack.Num();
    if (StackSize <= 0)
    {
        return;
    }

    int32 RemoveCount = 0;
    int32 RemoveIndex = INDEX_NONE;
    for (int32 StackIndex = 0; StackIndex < StackSize; ++StackIndex)
    {
        UHakCameraMode* CameraMode = CameraModeStack[StackIndex];
        check(CameraMode);

        // UpdateCameraMode를 확인해보자:
        CameraMode->UpdateCameraMode(DeltaTime);

        if (CameraMode->BlendWeight >= 1.0f)
        {
            RemoveIndex = (StackIndex + 1);
            RemoveCount = (StackSize - RemoveIndex);
            break;
        }
    }

    if (RemoveCount > 0)
    {
        // 생각해보면 이거 때문에 Pop하고 Push를 안한거일지도?
        CameraModeStack.RemoveAt(RemoveIndex, RemoveCount);
    }
}
```

1

☐ UpdateCameraMode를 구현하자:



☐ 간단한 정의를 .h/.cpp에 구현:

```cpp
/** Camera Blending 대상 유닛 */
UCLASS(Abstract, NotBlueprintable)
class UHakCameraMode : public UObject
{
    GENERATED_BODY()
public:
    UHakCameraMode(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * member methods
     */
    void UpdateCameraMode(float DeltaTime);

    /** 얼마동안 Blend를 진행할지 시간을 의미 */
    UPROPERTY(EditDefaultsOnly, Category="Blending")
    float BlendTime;

    /** 선형적인 Blend 값 [0, 1] */
    float BlendAlpha;

    /**
     * 해당 CameraMode의 최종 Blend 값
     * 앞서 BlendAlpha의 선형 값을 매핑하여 최종 BlendWeight를 계산 (코드를 보며, 이해해보자)
     */
    float BlendWeight;
};
```

1

```cpp
void UHakCameraMode::UpdateCameraMode(float DeltaTime)
{
    UpdateView(DeltaTime);
    UpdateBlending(DeltaTime);
}
```

- [ ] UpdateView()를 구현하자:
  - [ ] 우선 아래 마킹했듯이 멤버 변수를 추가하자:

```cpp
void UHakCameraMode::UpdateView(float DeltaTime)
{
    FVector PivotLocation = GetPivotLocation();
    FRotator PivotRotation = GetPivotRotation();

    PivotRotation.Pitch = FMath::ClampAngle(PivotRotation.Pitch, ViewPitchMin, ViewPitchMax);

    View.Location = PivotLocation;
    View.Rotation = PivotRotation;
    View.ControlRotation = View.Rotation;
    View.FieldOfView = FieldOfView;
}
```

2 1 3

- ViewPitch[Min|Max], View, FieldOfView 변수 추가:

```cpp
/** Camera Blending 대상 유닛 */
UCLASS(Abstract, NotBlueprintable)
class UHakCameraMode : public UObject
{
    GENERATED_BODY()
public:
    UHakCameraMode(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
    * member methods
    */
    void UpdateView(float DeltaTime);
    void UpdateCameraMode(float DeltaTime);

    /** CameraMode에 의해 생성된 CameraModeView */
    FHakCameraModeView View;

    /** Camera Mode의 FOV */
    UPROPERTY(EditDefaultsOnly, Category = "View", Meta = (UIMin = "5.0", UIMax = "170", ClampMin = "5.0", Clampmax = "170.0"))
    float FieldOfView;

    /** View에 대한 Pitch [Min, Max] */
    UPROPERTY(EditDefaultsOnly, Category = "View", Meta = (UIMin = "-89.9", UIMax = "89.9", ClampMin = "-89.9", Clampmax = "89.9"))
    float ViewPitchMin;

    UPROPERTY(EditDefaultsOnly, Category = "View", Meta = (UIMin = "-89.9", UIMax = "89.9", ClampMin = "-89.9", Clampmax = "89.9"))
    float ViewPitchMax;

    /** 얼마동안 Blend를 진행할지 시간을 의미 */
    UPROPERTY(EditDefaultsOnly, Category="Blending")
    float BlendTime;

    /** 선형적인 Blend 값 [0, 1] */
    float BlendAlpha;

    /**
    * 해당 CameraMode의 최종 Blend 값
    * 앞서 BlendAlpha의 선형 값을 매핑하여 최종 BlendWeight를 계산 (코드를 보며, 이해해보자)
    */
    float BlendWeight;
};
```

1

- Constructor에 추가된 변수 초기화:

```cpp
UHakCameraMode::UHakCameraMode(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
    FieldOfView = HAK_CAMERA_DEFAULT_FOV;
    ViewPitchMin = HAK_CAMERA_DEFAULT_PITCH_MIN;
    ViewPitchMax = HAK_CAMERA_DEFAULT_PITCH_MAX;

    BlendTime = 0.0f;
    BlendAlpha = 1.0f;
    BlendWeight = 1.0f;
}
```

1

- [ ] GetPivotLocation()과 GetPivotRotation() 구현:

- GetPivotLocation()
  - GetTargetActor()를 구현하자:

```cpp
AActor* UHakCameraMode::GetTargetActor() const
{
    const UHakCameraComponent* HakCameraComponent = GetHakCameraComponent();
    return HakCameraComponent->GetTargetActor();
}
```

    - GetHakCameraComponent 구현:

```cpp
UHakCameraComponent* UHakCameraMode::GetHakCameraComponent() const
{
    // 우리가 앞서 UHakCameraMode를 생성하는 곳은 UHakCameraModeStack이었다:
    // - 해당 코드를 보면, GetOuter()를 HakCameraMode로 HakCameraComponent로 설정하였다
    // - UHakCameraModeStack::GetCameraModeInstance() 확인
    return CastChecked<UHakCameraComponent>(GetOuter());
}
```

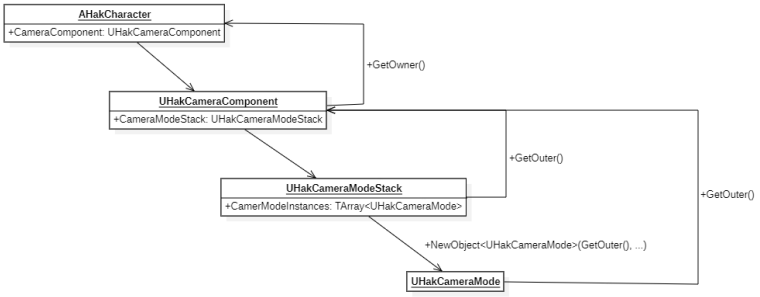
    - UHakCameraComponent::GetTargetActor() 구현:

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

    static UHakCameraComponent* FindCameraComponent(const AActor* Actor) { return (Actor ? Actor->FindComponentByClass<UHakCameraComponent>() : nullptr); }

    /**
     * member methods
     */
    AActor* GetTargetActor() const { return GetOwner(); }
    void UpdateCameraModes();

    /**
     * CameraComponent interface
     */
    virtual void OnRegister() final;
    virtual void GetCameraView(float DeltaTime, FMinimalViewInfo& DesiredView) final;

    /**
     * member variables
     */
    /** 카메라의 blending 기능을 지원하는 stack */
    UPROPERTY()
    TObjectPtr<UHakCameraModeStack> CameraModeStack;

    /** 현재 CameraMode를 가져오는 Delegate */
    FHakCameraModeDelegate DetermineCameraModeDelegate;
};
```

      - 여기서, CameraComponent는 HakCharacter에 생성했었다:

```cpp
AHakCharacter::AHakCharacter(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
    // Tick을 비활성화
    PrimaryActorTick.bCanEverTick = false;
    PrimaryActorTick.bStartWithTickEnabled = false;

    // PawnExtComponent 생성
    PawnExtComponent = CreateDefaultSubobject<UHakPawnExtensionComponent>(TEXT("PawnExtensionComponent"));

    // CameraComponent 생성
    {
        CameraComponent = CreateDefaultSubobject<UHakCameraComponent>(TEXT("CameraComponent"));
        CameraComponent->SetRelativeLocation(FVector(-300.0f, 0.0f, 75.0f));
    }
}
```

  - 여기서 잠깐! CameraMode → CameraModeStack → CameraComponent → Character간의 관계를 UML로 잠깐 정리하고 가자:



- GetPivotRotation()

```cpp
FRotator UHakCameraMode::GetPivotRotation() const
{
    const AActor* TargetActor = GetTargetActor();
    check(TargetActor);

    if (const APawn* TargetPawn = Cast<APawn>(TargetActor))
    {
        // GetViewRoation() 확인, 보통 Pawn의 ControlRotation을 반환
        return TargetPawn->GetViewRotation();
    }

    return TargetActor->GetActorRotation();
}
```

☐ UpdateView 코드 리뷰:

```cpp
void UHakCameraMode::UpdateView(float DeltaTime)
{
    // CameraMode를 가지고 있는 CameraComponent의 Owner인 Character(Pawn)을 활용하여, PivotLocation/Rotation을 반환함
    FVector PivotLocation = GetPivotLocation();
    FRotator PivotRotation = GetPivotRotation();

    // Pitch 값에 대해 Min/Max를 Clamp시킴
    PivotRotation.Pitch = FMath::ClampAngle(PivotRotation.Pitch, ViewPitchMin, ViewPitchMax);

    // FHakCameraModeView에 PivotLocation/Rotation 설정
    View.Location = PivotLocation;
    View.Rotation = PivotRotation;

    // PivotRotation을 똑같이 ControlRotation으로 활용
    View.ControlRotation = View.Rotation;
    View.FieldOfView = FieldOfView;

    // 정리하면, Character의 Location과 ControlRotation을 활용하여, View를 업데이트함
}
```

☐ UpdateBlending 구현:

☐ 우선 EHakCameraModeBlendFunction을 구현하자:

```cpp
/**
 * [0,1]을 BlendFunction에 맞게 재매핑을 위한 타입
 */
UENUM(BlueprintType)
enum class EHakCameraModeBlendFunction : uint8
{
    Linear,
    /**
     * EaseIn/Out은 exponent 값에 의해 조절된다:
     */
    EaseIn,
    EaseOut,
    COUNT
};
```

☐ 해당 Enum을 사용하여, CameraMode에 BlendFunction을 멤버 변수로 넣어주자:

- 그리고 Exponent도!

```cpp
/** Camera Blending 대상 유닛 */
UCLASS(Abstract, NotBlueprintable)
class UHakCameraMode : public UObject
{
    GENERATED_BODY()
public:
    UHakCameraMode(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * member methods
     */
    UHakCameraComponent* GetHakCameraComponent() const;
    AActor* GetTargetActor() const;
    FVector GetPivotLocation() const;
    FRotator GetPivotRotation() const;
    void UpdateView(float DeltaTime);
    void UpdateBlending(float DeltaTime);
    void UpdateCameraMode(float DeltaTime);

    /** CameraMode에 의해 생성된 CameraModeView */
    FHakCameraModeView View;

    /** Camera Mode의 FOV */
    UPROPERTY(EditDefaultsOnly, Category = "View", Meta = (UIMin = "5.0", UIMax = "170", ClampMin = "5.0", Clampmax = "170.0"))
    float FieldOfView;

    /** View에 대한 Pitch [Min, Max] */
    UPROPERTY(EditDefaultsOnly, Category = "View", Meta = (UIMin = "-89.9", UIMax = "89.9", ClampMin = "-89.9", Clampmax = "89.9"))
    float ViewPitchMin;

    UPROPERTY(EditDefaultsOnly, Category = "View", Meta = (UIMin = "-89.9", UIMax = "89.9", ClampMin = "-89.9", Clampmax = "89.9"))
    float ViewPitchMax;

    /** 얼마동안 Blend를 진행할지 시간을 의미 */
    UPROPERTY(EditDefaultsOnly, Category="Blending")
    float BlendTime;

    /** 선형적인 Blend 값 [0, 1] */
    float BlendAlpha;

    /**
     * 해당 CameraMode의 최종 Blend 값
     * 앞서 BlendAlpha의 선형 값을 매핑하여 최종 BlendWeight를 계산 (코드를 보며, 이해해보자)
     */
    float BlendWeight;

    /**
     * EaseIn/Out에 사용한 Exponent
     */
    UPROPERTY(EditDefaultsOnly, Category = "Blending")
    float BlendExponent;

    /** Blend function */
    EHakCameraModeBlendFunction BlendFunction;
};
```

1

- 당연히 Constructor도 업데이트 해주자:

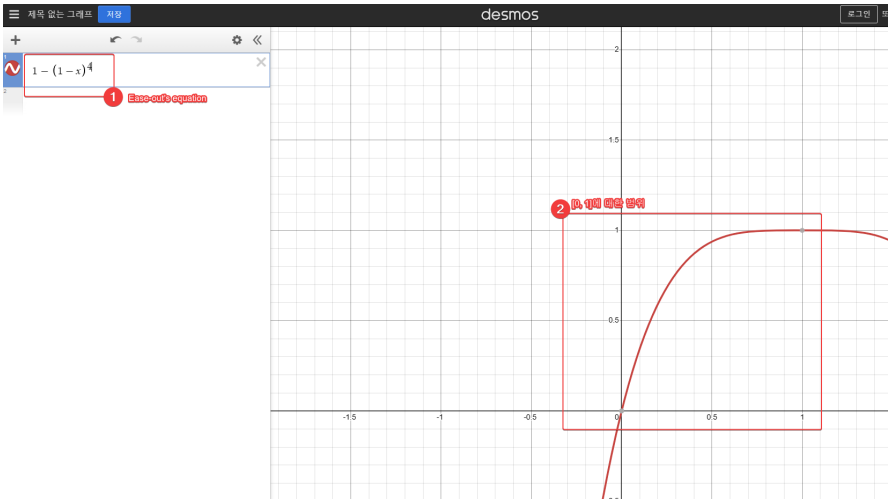
```cpp
UHakCameraMode::UHakCameraMode(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
    FieldOfView = HAK_CAMERA_DEFAULT_FOV;
    ViewPitchMin = HAK_CAMERA_DEFAULT_PITCH_MIN;
    ViewPitchMax = HAK_CAMERA_DEFAULT_PITCH_MAX;

    BlendTime = 0.0f;
    BlendAlpha = 1.0f;
    BlendWeight = 1.0f;

    BlendFunction = EHakCameraModeBlendFunction::EaseOut;
    BlendExponent = 4.0f;
}
```

1

☐ 이어서, UpdateBlending 구현:

```cpp
void UHakCameraMode::UpdateBlending(float DeltaTime)
{
    // BlendAlpha를 DeltaTime을 통해 계산
    if (BlendTime > 0.f)
    {
        // BlendTime은 Blending 과정 총 시간(초)
        // - BlendAlpha는 0 -> 1로 변화하는 과정:
        // - DeltaTime을 활용하여, BlendTime을 1로 볼 경우, 진행 정도를 누적
        BlendAlpha += (DeltaTime / BlendTime);
    }
    else
    {
        BlendAlpha = 1.0f;
    }

    // BlendWeight를 [0,1]를 BlendFunction에 맞게 재매핑
    const float Exponent = (BlendExponent > 0.0f) ? BlendExponent : 1.0f;
    switch (BlendFunction)
    {
    case EHakCameraModeBlendFunction::Linear:
        BlendWeight = BlendAlpha;
        break;
    case EHakCameraModeBlendFunction::EaseIn:
        BlendWeight = FMath::InterpEaseIn(0.0f, 1.0f, BlendAlpha, Exponent);
        break;
    case EHakCameraModeBlendFunction::EaseOut:
        BlendWeight = FMath::InterpEaseOut(0.0f, 1.0f, BlendAlpha, Exponent);
        break;
    case EHakCameraModeBlendFunction::EaseInOut:
        BlendWeight = FMath::InterpEaseInOut(0.0f, 1.0f, BlendAlpha, Exponent);
        break;
    default:
        checkf(false, TEXT("UpdateBlending: Invalid BlendFunction [%d]\n"), (uint8)BlendFunction);
        break;
    }
}
```

- 해당 함수를 그래프를 그려보면 아래와 같다:



- [ ] UpdateCameraMode를 정리하면 아래와 같다:

```cpp
void UHakCameraMode::UpdateCameraMode(float DeltaTime)
{
    // Actor를 활용하여, Pivot[Location|Rotation]을 계산하여, View를 업데이트
    UpdateView(DeltaTime);

    // BlendWeight를 DeltaTime을 활용하여, BlendAlpha 계산 후, BlendFunction에 맞게 재-매핑하여 최종 계산
    UpdateBlending(DeltaTime);
}
```

- [ ] UpdateStack 정리:

```cpp
void UHakCameraModeStack::UpdateStack(float DeltaTime)
{
    const int32 StackSize = CameraModeStack.Num();
    if (StackSize <= 0)
    {
        return;
    }

    // CameraModeStack을 순회하며, CameraMode를 업데이트한다
    int32 RemoveCount = 0;
    int32 RemoveIndex = INDEX_NONE;
    for (int32 StackIndex = 0; StackIndex < StackSize; ++StackIndex)
    {
        UHakCameraMode* CameraMode = CameraModeStack[StackIndex];
        check(CameraMode);

        // UpdateCameraMode를 확인해보자:
        CameraMode->UpdateCameraMode(DeltaTime);

        // 만약 하나라도 CameraMode가 BlendWeight가 1.0에 도달했다면, 그 이후 CameraMode를 제거한다
        if (CameraMode->BlendWeight >= 1.0f)
        {
            RemoveIndex = (StackIndex + 1);
            RemoveCount = (StackSize - RemoveIndex);
            break;
        }
    }

    if (RemoveCount > 0)
    {
        // 생각해보면 이거 때문에 Pop하고 Push를 안한거일지도?
        CameraModeStack.RemoveAt(RemoveIndex, RemoveCount);
    }
}
```

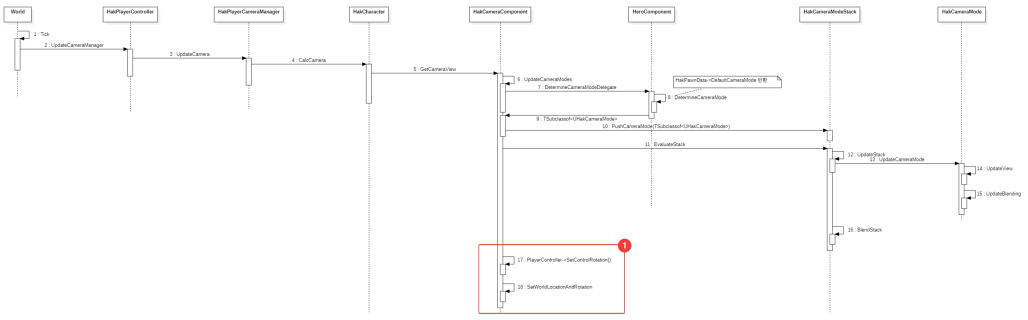
- [ ] BlendStack을 구현하자:

```cpp
void UHakCameraModeStack::BlendStack(FHakCameraModeView& OutCameraModeView) const
{
    const int32 StackSize = CameraModeStack.Num();
    if (StackSize <= 0)
    {
        return;
    }

    // CameraModeStack은 Bottom -> Top 순서로 Blend를 진행한다
    const UHakCameraMode* CameraMode = CameraModeStack[StackSize - 1];
    check(CameraMode);

    // 참고로 가장 아래 (Bottom)은 BlendWeight가 1.0이다!
    OutCameraModeView = CameraMode->View;

    // 이미 Index = [StackSize - 1] 이미 OutCameraModeView로 지정했으므로, StackSize - 2부터 순회하면 된다
    for (int32 StackIndex = (StackSize - 2); StackIndex >= 0; --StackIndex)
    {
        CameraMode = CameraModeStack[StackIndex];
        check(CameraMode);

        // HakCameraModeView Blend 함수를 구현하자:
        OutCameraModeView.Blend(CameraMode->View, CameraMode->BlendWeight);
    }
}
```

- [ ] UHakCameraModeView::Blend 함수를 구현하자:

```cpp
void FHakCameraModeView::Blend(const FHakCameraModeView& Other, float OtherWeight)
{
    if (OtherWeight <= 0.0f)
    {
        return;
    }
    else if (OtherWeight >= 1.0f)
    {
        // Weight가 1.0이면 Other를 덮어쓰면 된다
        *this = Other;
        return;
    }

    // Location + OtherWeight * (Other.Location - Location);
    Location = FMath::Lerp(Location, Other.Location, OtherWeight);

    // Location과 같은 방식 Lerp (ControlRotation과 FieldOfView도 같음)
    const FRotator DeltaRotation = (Other.Rotation - Rotation).GetNormalized();
    Rotation = Rotation + (OtherWeight * DeltaRotation);

    const FRotator DeltaControlRotation = (Other.ControlRotation - ControlRotation).GetNormalized();
    ControlRotation = ControlRotation + (OtherWeight * DeltaControlRotation);

    FieldOfView = FMath::Lerp(FieldOfView, Other.FieldOfView, OtherWeight);
}
```

- [ ] 다시 돌아와서, UHakCameraComponent::GetCameraView() 구현을 진행하자:
  - 우선 우리가 진행해야 할 부분을 UML로 다시보자:



☐ UHakCameraComponent::GetCameraView() 구현:

```cpp
void UHakCameraComponent::GetCameraView(float DeltaTime, FMinimalViewInfo& DesiredView)
{
    check(CameraModeStack);

    // 해당 함수는 HeroComponent에서 PawnData에서 DefaultCameraMode를 가져와서 CameraModeStack에 추가하여, CameraMode 블랜딩 준비한다
    UpdateCameraModes();

    // EvaluateStack은 CameraModeStack에 있는 CameraMode를 업데이트(+블랜딩)하고 CameraModeStack을 Bottom-Top까지 업데이트된 CameraMode들을 Lerp를 진행해준다.
    // - 이에 대한 결과는 CameraModeView에 캐싱된다
    FHakCameraModeView CameraModeView;
    CameraModeStack->EvaluateStack(DeltaTime, CameraModeView);

    if (APawn* TargetPawn = Cast<APawn>(GetTargetActor()))
    {
        if (APlayerController* PC = TargetPawn->GetController<APlayerController>())
        {
            // PlayerController의 ControlRotation을 계산된 CamerModeView의 ControlRotation으로 업데이트해주자
            // - SetControlRotation을 외우기보다 한번 코드를 보자:
            //   - 해당 함수는 PC가 Possess하고 있는 Pawn의 RootComponent에 ControlRotation을 반영한다 (== 조정하고 있는 캐릭터에 회전을 시키겠지?)
            PC->SetControlRotation(CameraModeView.ControlRotation);
        }
    }

    // Camera의 Location과 Rotation을 반영하자:
    // - 시간 있으면, SetWorldLocationAndRotation을 보면서 -> UpdateChildTransform까지 보는 것을 추천한다
    // - SceneGraph 관계 업데이트를 이해할 수 있다
    // 간단하게 정리하면, CameraComponent에 대해 Parent의 SceneGraph 관계를 고려하여 업데이트 진행한다
    SetWorldLocationAndRotation(CameraModeView.Location, CameraModeView.Rotation);

    // FOV 업데이트
    FieldOfView = CameraModeView.FieldOfView;

    /**
    * 여기서 우리는 ControlRotation vs Rotation을 차이점을 이해할 수 있다:
    * - ControlRotation은 PC가 조정하는 Pawn의 Rotation을 적용할 값이다
    * - 그에 반해, Rotation은 Camera에 반영하는 Rotation이라는 것이다
    */

    // FMinimalViewInfo를 업데이트 해준다:
    // - CameraComponent의 변화 사항을 반영해서 최종 렌더링까지 반영하게 됨
    DesiredView.Location = CameraModeView.Location;
    DesiredView.Rotation = CameraModeView.Rotation;
    DesiredView.FOV = CameraModeView.FieldOfView;
    DesiredView.OrthoWidth = OrthoWidth;
    DesiredView.OrthoNearClipPlane = OrthoNearClipPlane;
    DesiredView.OrthoFarClipPlane = OrthoFarClipPlane;
    DesiredView.AspectRatio = AspectRatio;
    DesiredView.bConstrainAspectRatio = bConstrainAspectRatio;
    DesiredView.bUseFieldOfViewForLOD = bUseFieldOfViewForLOD;
    DesiredView.ProjectionMode = ProjectionMode;
    DesiredView.PostProcessBlendWeight = PostProcessBlendWeight;
    if (PostProcessBlendWeight > 0.0f)
    {
        DesiredView.PostProcessSettings = PostProcessSettings;
    }
}
```

☐ 관련 코드 주석 설명 진행

CameraMode_ThirdPerson

▼ **펼치기**

☐ ThirdPerson용으로 CurveData를 활용하여, UpdateView를 오버라이드하자:

```cpp
#pragma once

#include "HakCameraMode.h"
#include "HakCameraMode_ThirdPerson.generated.h"

/** forward declarations */
class UCurveVector;

UCLASS(Abstract, Blueprintable)
class UHakCameraMode_ThirdPerson : public UHakCameraMode
{
    GENERATED_BODY()
public:
    UHakCameraMode_ThirdPerson(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * UHakCameraMode's interface
     */
    virtual void UpdateView(float DeltaTime) override;

    /**
     * member variables
     */
    UPROPERTY(EditDefaultsOnly, Category="Third Person")
    TObjectPtr<const UCurveVector> TargetOffsetCurve;
};
```

1 HakCameraMode의 UpdateView()를 virtual 함수화 하자!

2

☐ UHakCameraMode::UpdateView()의 virtual 함수화

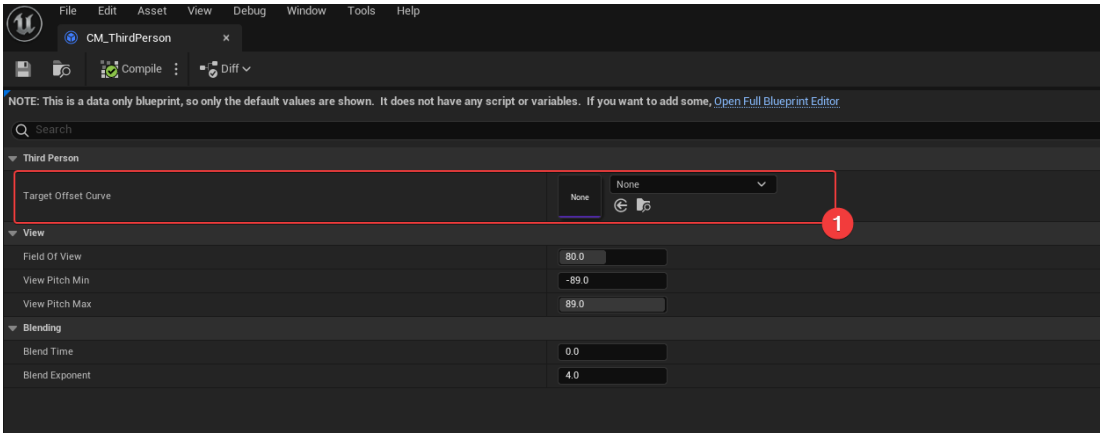
```cpp
/** Camera Blending 대상 유닛 */
UCLASS(Abstract, NotBlueprintable)
class UHakCameraMode : public UObject
{
    GENERATED_BODY()
public:
    UHakCameraMode(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    /**
     * member methods
     */
    UHakCameraComponent* GetHakCameraComponent() const;
    AActor* GetTargetActor() const;
    FVector GetPivotLocation() const;
    FRotator GetPivotRotation() const;
    void UpdateBlending(float DeltaTime);
    void UpdateCameraMode(float DeltaTime);

    /**
     * HakGameMode's interface
     */
    virtual void UpdateView(float DeltaTime);

    /**
     * member variables
     */

    /** CameraMode에 의해 생성된 CameraModeView */
    FHakCameraModeView View;

    /** Camera Mode의 FOV */
    UPROPERTY(EditDefaultsOnly, Category = "View", Meta = (UIMin = "5.0", UIMax = "170", ClampMin = "5.0", Clampmax = "170.0"))
    float FieldOfView;

    /** View에 대한 Pitch [Min, Max] */
    UPROPERTY(EditDefaultsOnly, Category = "View", Meta = (UIMin = "-89.9", UIMax = "89.9", ClampMin = "-89.9", Clampmax = "89.9"))
    float ViewPitchMin;

    UPROPERTY(EditDefaultsOnly, Category = "View", Meta = (UIMin = "-89.9", UIMax = "89.9", ClampMin = "-89.9", Clampmax = "89.9"))
    float ViewPitchMax;

    /** 얼마동안 Blend를 진행할지 시간을 의미 */
    UPROPERTY(EditDefaultsOnly, Category="Blending")
    float BlendTime;

    /** 선형적인 Blend 값 [0, 1] */
    float BlendAlpha;

    /**
     * 해당 CameraMode의 최종 Blend 값
     * 앞서 BlendAlpha의 선형 값을 매핑하여 최종 BlendWeight를 계산 (코드를 보며, 이해해보자)
     */
    float BlendWeight;

    /**
     * EaseIn/Out에 사용한 Exponent
     */
    UPROPERTY(EditDefaultsOnly, Category = "Blending")
    float BlendExponent;

    /** Blend function */
    EHakCameraModeBlendFunction BlendFunction;
};
```

1

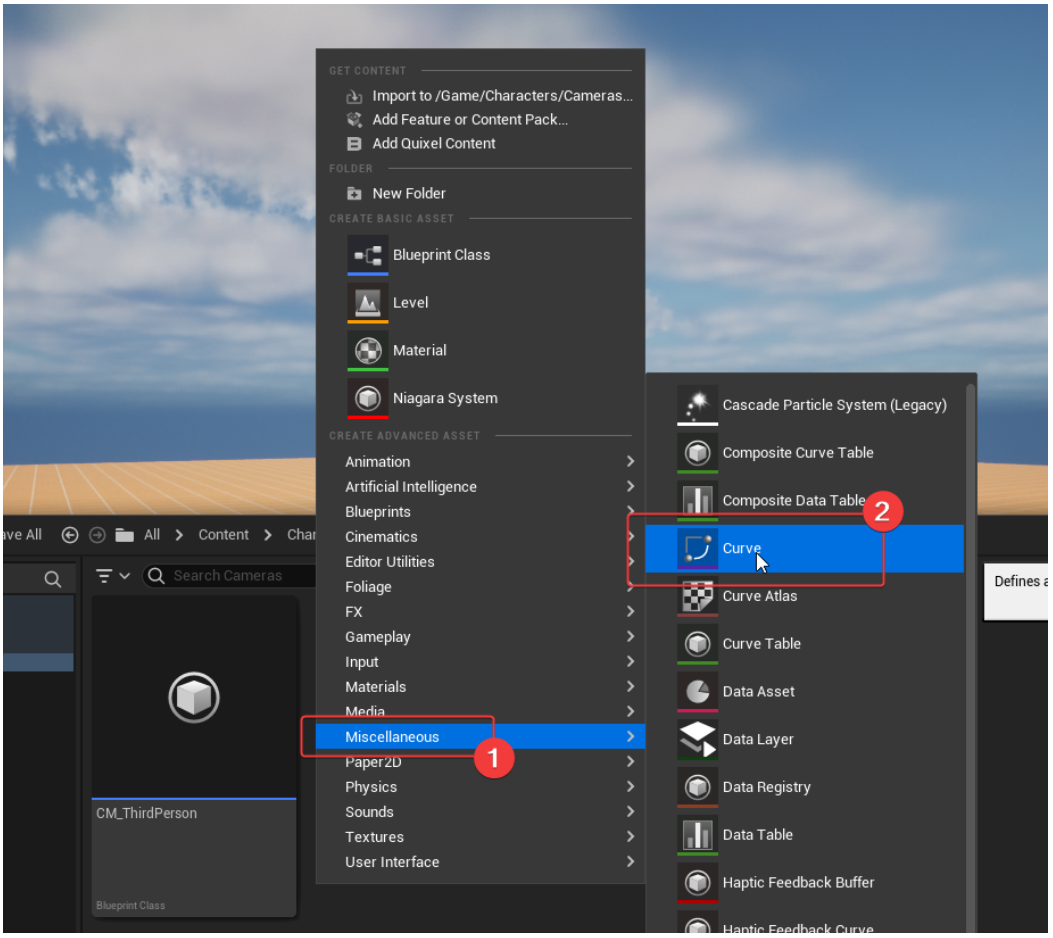
☐ CameraMode_ThirPerson::UpdateView() 구현:

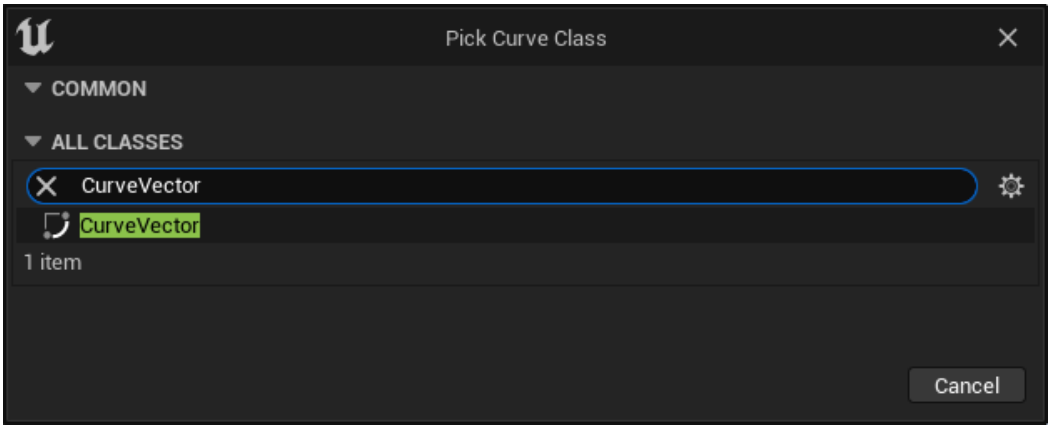
```cpp
#include "HakCameraMode_ThirdPerson.h"
#include "Curves/CurveVector.h"
#include UE_INLINE_GENERATED_CPP_B...E(HakCameraMode_ThirdPerson)

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

void UHakCameraMode_ThirdPerson::UpdateView(float DeltaTime)
{
    FVector PivotLocation = GetPivotLocation();
    FRotator PivotRotation = GetPivotRotation();

    PivotRotation.Pitch = FMath::ClampAngle(PivotRotation.Pitch, ViewPitchMin, ViewPitchMax);

    View.Location = PivotLocation;
    View.Rotation = PivotRotation;
    View.ControlRotation = View.Rotation;
    View.FieldOfView = FieldOfView;

    // TargetOffsetCurve가 오버라이드되어 있다면, Curve에 값을 가져와서 적용 진행
    // - Camera 관점에서 Charater의 어느 부분에 Target으로 할지 결정하는 것으로 이해하면 됨
    if (TargetOffsetCurve)
    {
        const FVector TargetOffset = TargetOffsetCurve->GetVectorValue(PivotRotation.Pitch);
        View.Location = PivotLocation + PivotRotation.RotateVector(TargetOffset);
    }
}
```

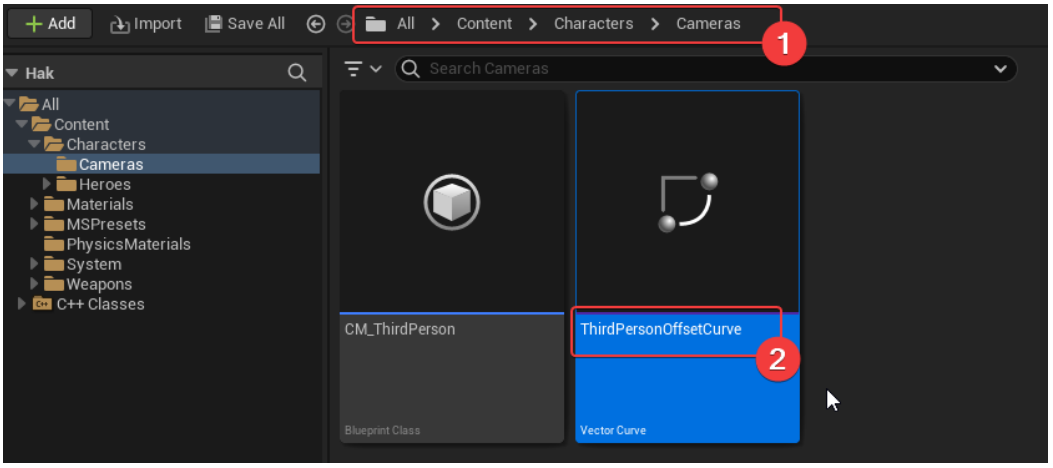
☐ CM_ThirdPerson BP 파일을 열어보자:



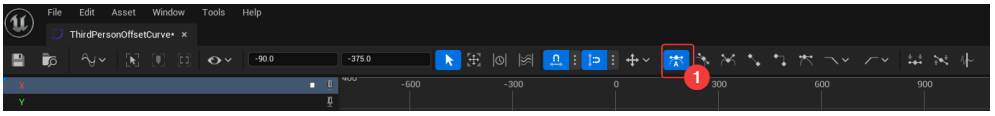
☐ Target Offset Curve에 넣어줄 Curve를 만들어주자:


GET CONTENT
Import to /Game/Characters/Cameras...
Add Feature or Content Pack...
Add Quixel Content
FOLDER
New Folder
CREATE BASIC ASSET
Blueprint Class
Level
Material
Niagara System
CREATE ADVANCED ASSET
Animation
Artificial Intelligence
Blueprints
Cinematics
Editor Utilities
Foliage
FX
Gameplay
Input
Materials
Media
Miscellaneous
Paper2D
Physics
Sounds
Textures
User Interface
Cascade Particle System (Legacy)
Composite Curve Table
Composite Data Table
Curve
Curve Atlas
Curve Table
Data Asset
Data Layer
Data Registry
Data Table
Haptic Feedback Buffer
Haptic Feedback Curve
1
2
All
Content
Search Cameras
CM_ThirdPerson
Blueprint Class

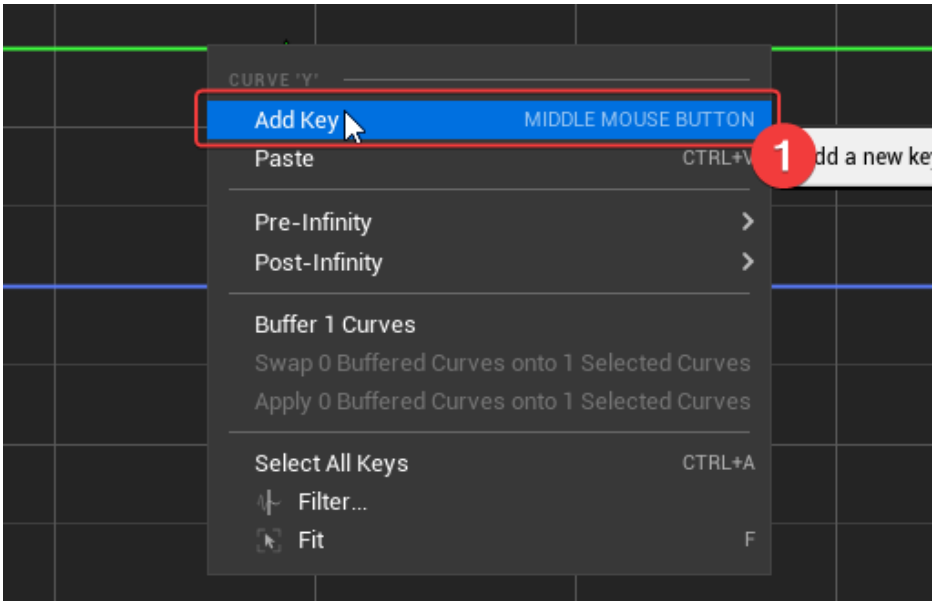

Pick Curve Class
COMMON
ALL CLASSES
CurveVector
CurveVector
1 item
Cancel



- 각 모든 Control Point는 Cubic Interpolation:



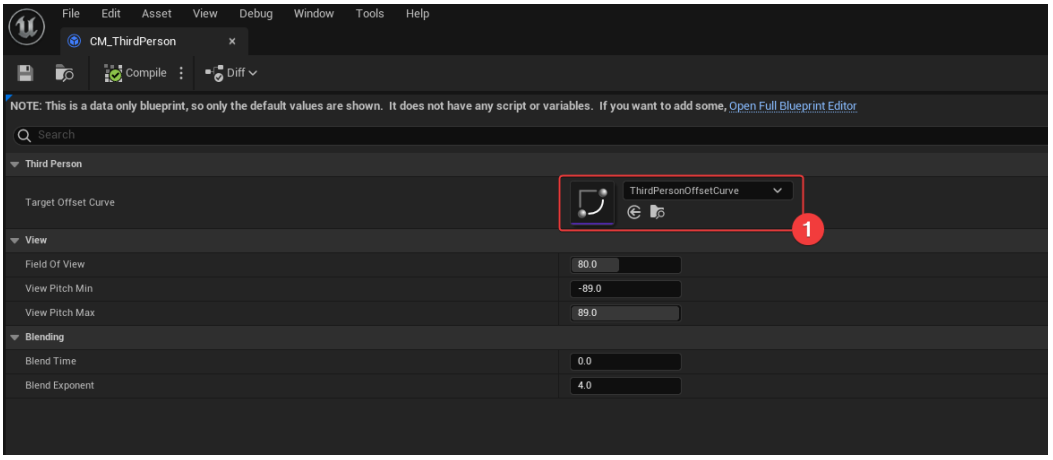
  - Control Point 추가 방법은 아래와 같다:



- X:
  - (-90, -375)
  - (90, -250)
  - (-40, -325) → **<u>자연스러운 효과를 위해</u>** 추가적인 Control Point!
- Y:

    - (-90, 60)
- Z:
    - (-90, 60)
    - (0, -10)

☐ ThirdPersonOffsetCurve를 CM_ThirdPerson에 적용해주자:

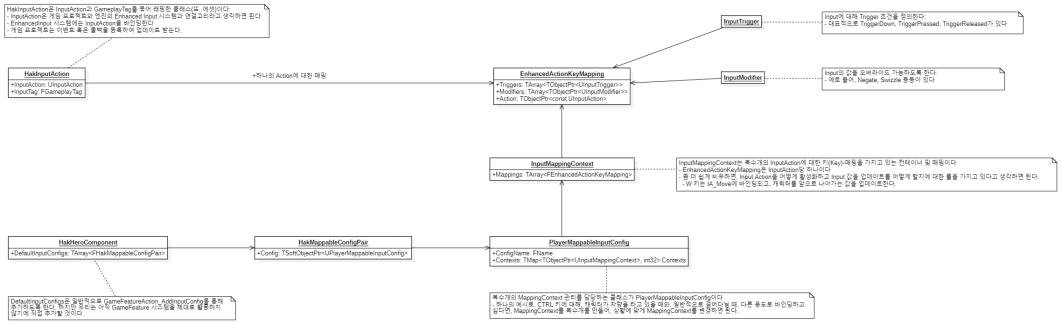


☐ 그럼 아래와 같이 올바른 카메라 세팅으로 잘 렌더링 됨을 확인할 수 있다:

EnhancedInput

▼ 펼치기

☐ EnhancedInput에 대한 간단한 설명:

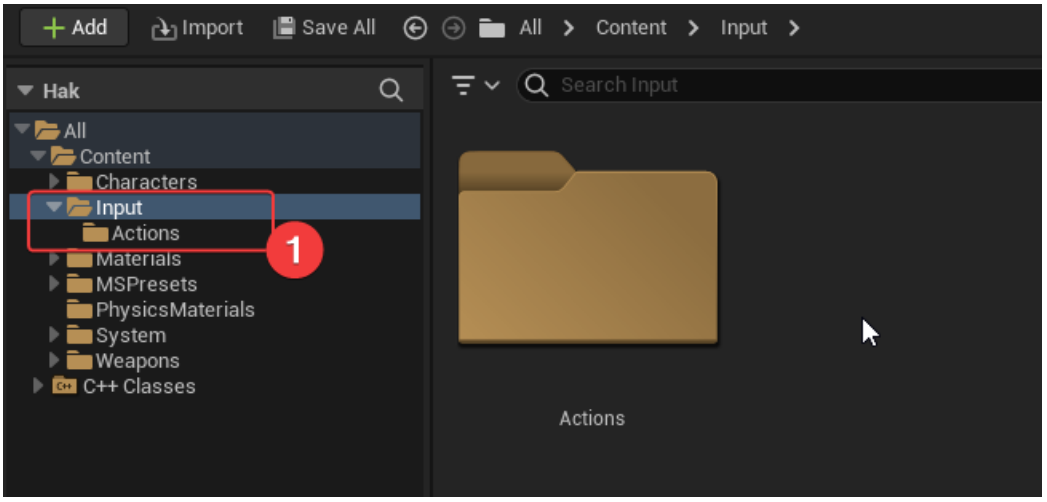


☐ GameplayTag 정의:

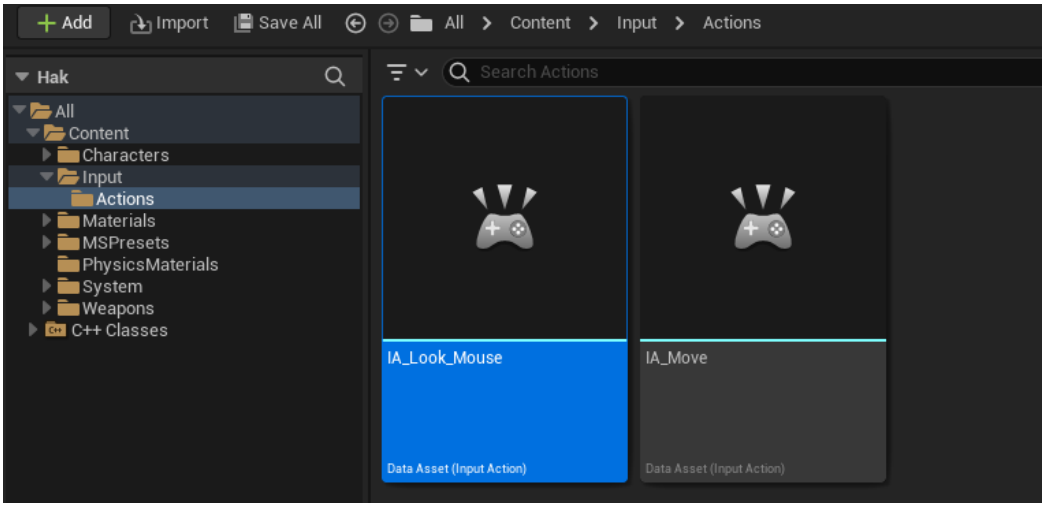
```cpp
#pragma once

#include "Containers/UnrealString.h"
#include "Containers/Map.h"
#include "GameplayTagContainer.h"

/** forward declaration */
class UGameplayTagsManager;

/**
 * HakGameplayTags
 * - singleton containing native gameplay tags
 */
struct FHakGameplayTags
{
    /**
     * static methods
     */
    static const FHakGameplayTags& Get() { return GameplayTags; }
    static void InitializeNativeTags();

    /**
     * member methods
     */
    void AddTag(FGameplayTag& OutTag, const ANSICHAR* TagName, const ANSICHAR* TagComment);
    void AddAllTags(UGameplayTagsManager& Manager);

    /**
     * 아래의 GameplayTag는 초기화 과정 단계를 의미한다:
     * - GameInstance의 초기화 과정에 UGameFrameworkComponentManager의 RegisterInitState로 등록되어 선형적으로(linear)하게 업데이트 된다
     * - 이 초기화 GameplayTag는 게임의 Actor 사이에 공유되며, GameFrameworkInitStateInterface 상속받은 클래스는 초기화 상태(Init State)를 상태머신(State Machine)과 같
     */
    FGameplayTag InitState_Spawned;
    FGameplayTag InitState_DataAvailable;
    FGameplayTag InitState_DataInitialized;
    FGameplayTag InitState_GameplayReady;

    /**
     * Enhanced Input Tags
     */
    FGameplayTag InputTag_Move;
    FGameplayTag InputTag_Look_Mouse;

private:
    // static 변수 초기화는 .cpp에 해주는 것을 잊지 말기!
    static FHakGameplayTags GameplayTags;
};
```

```cpp
void FHakGameplayTags::AddAllTags(UGameplayTagsManager& Manager)
{
    /**
     * GameFrameworkComponentManager init state tags
     */
    AddTag(InitState_Spawned, "InitState.Spawned", "1: Actor/Component has initially spawned and can be extended");
    AddTag(InitState_DataAvailable, "InitState.DataAvailable", "2: All required data has been loaded/replicated and is ready for initialization");
    AddTag(InitState_DataInitialized, "InitState.DataInitialized", "3: The available data has been initialized for this actor/component, but it is
    AddTag(InitState_GameplayReady, "InitState.GameplayReady", "4: The actor/component is fully ready for active gameplay");

    /**
     * Enhanced Input Tags
     */
    AddTag(InputTag_Move, "InputTag.Move", "");
    AddTag(InputTag_Look_Mouse, "InputTag.Look.Mouse", "");
}
```

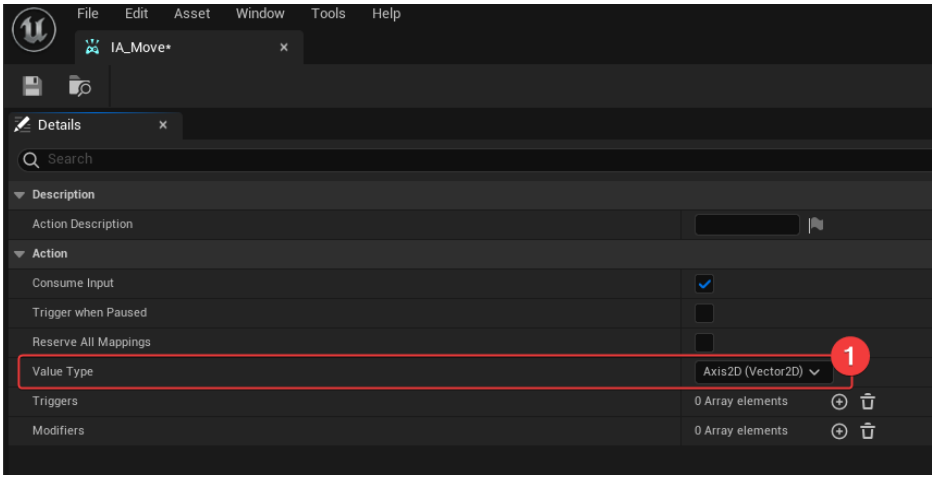
- [ ] InputAction 에셋 생성:
    - 폴더 생성:



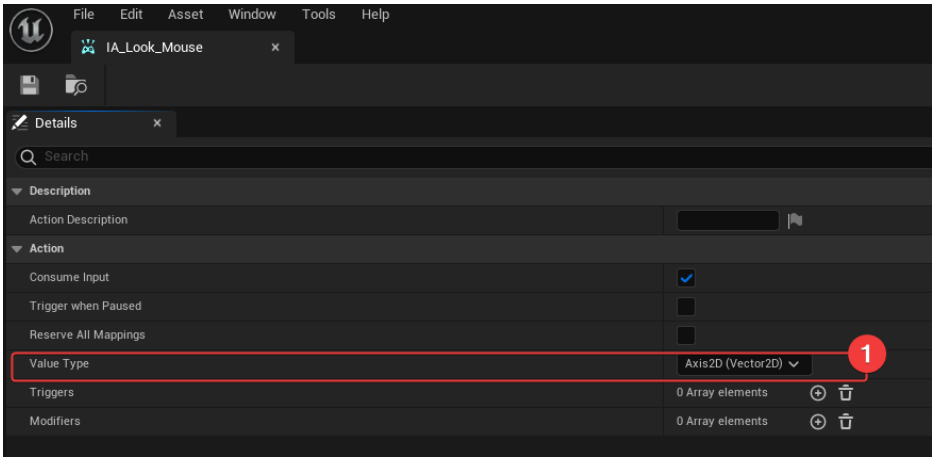
    - IA_Move와 IA_LookMouse. 생성:



    - [ ] 각 InputAction에 대한 속성 값 설정:
        - IA_Move:

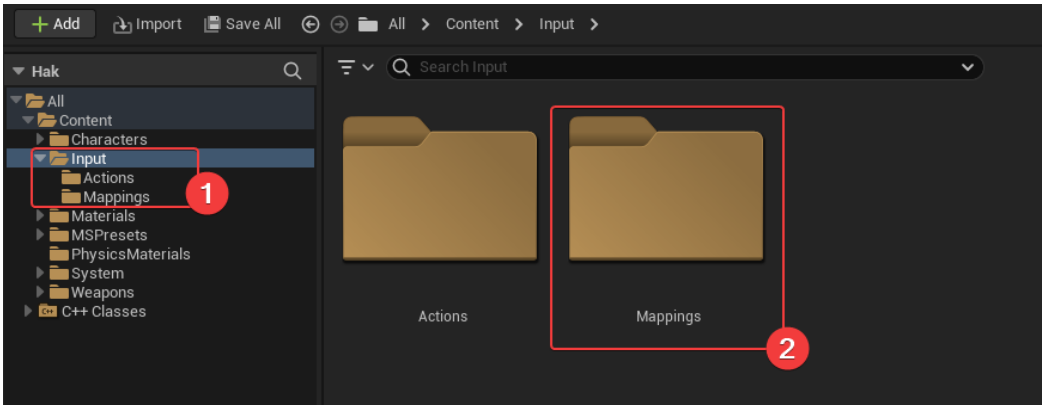


- IA_Look_Mouse:

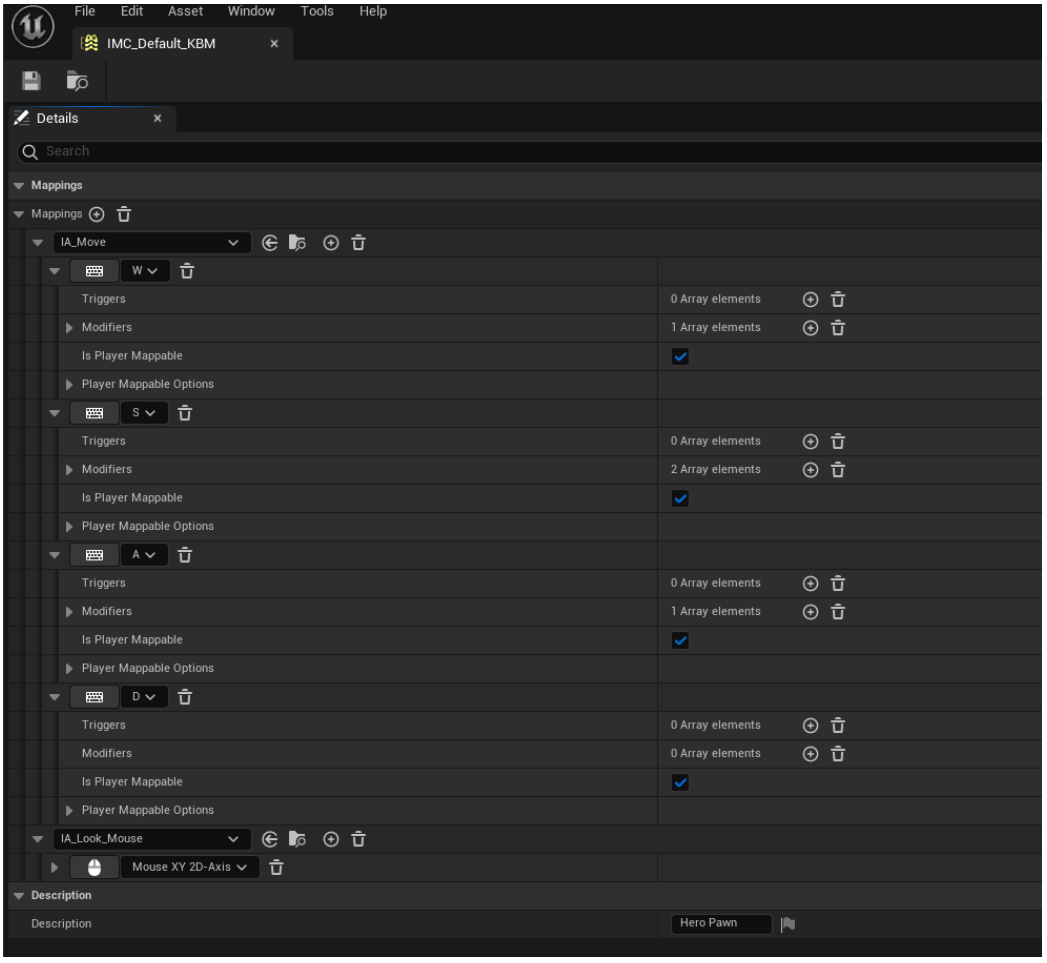


☐ Input Mapping Context 생성:

- 폴더 생성:

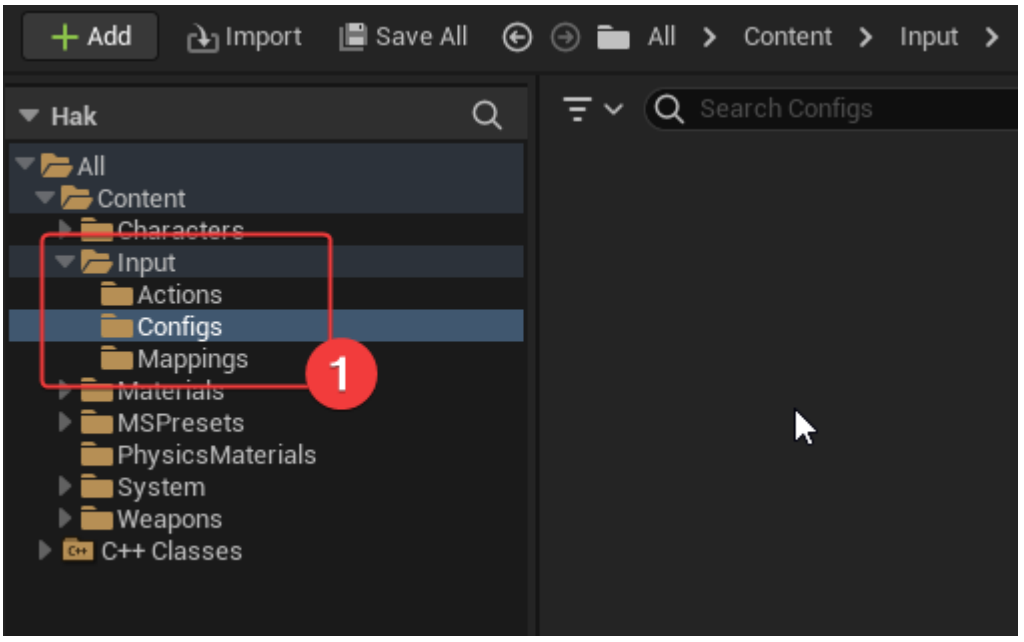


- IMC_Default_KBM 생성:

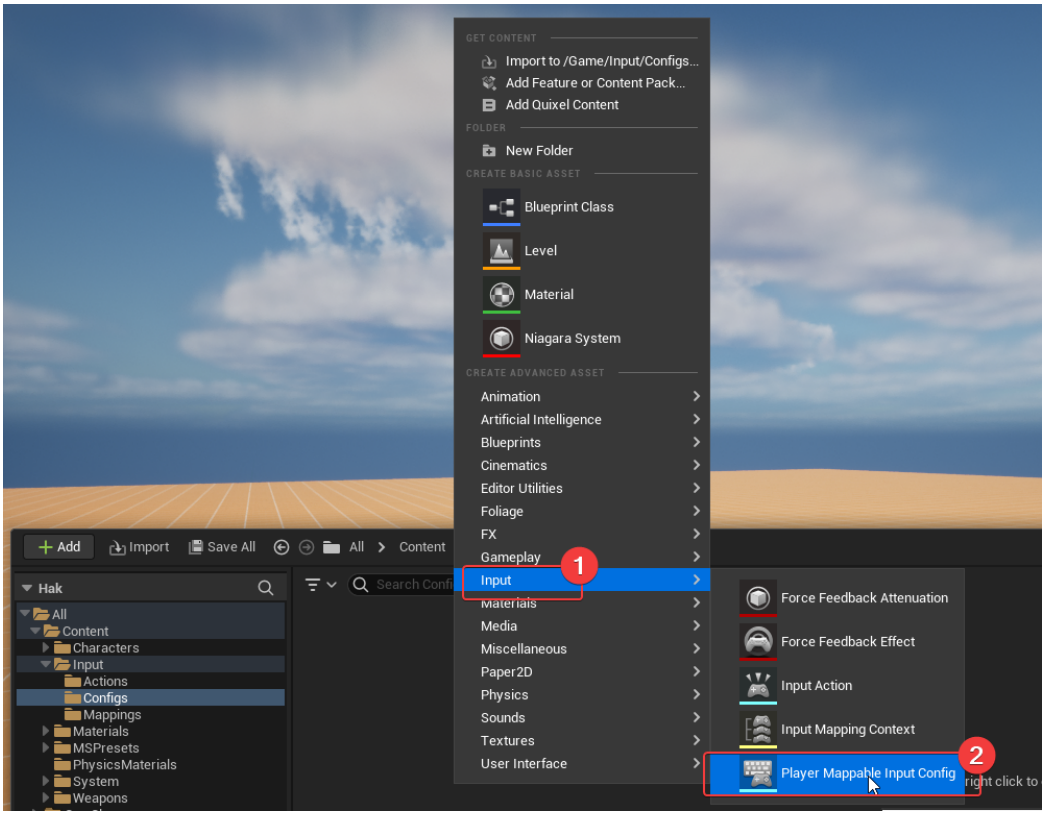


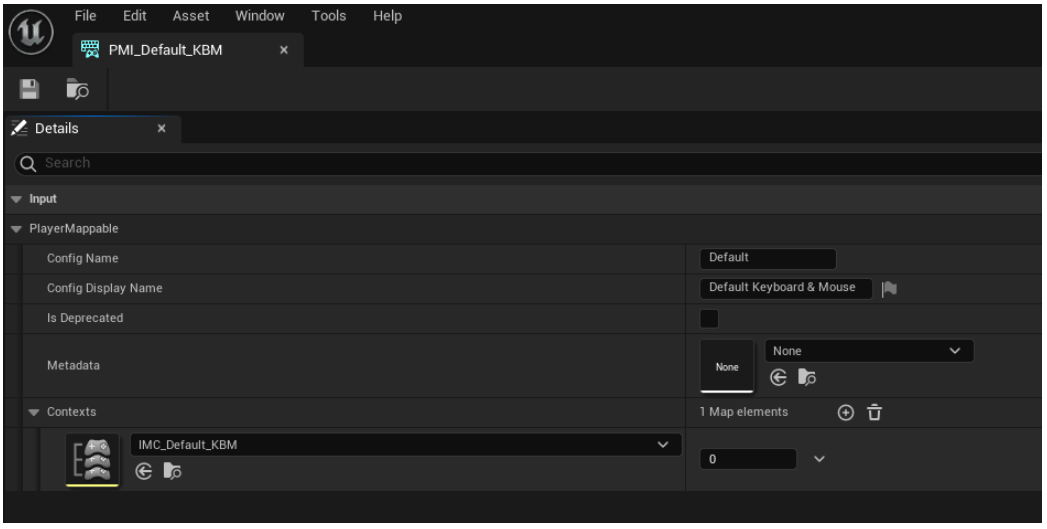
☐ PlayerMappableInputConfig

- 폴더 생성:

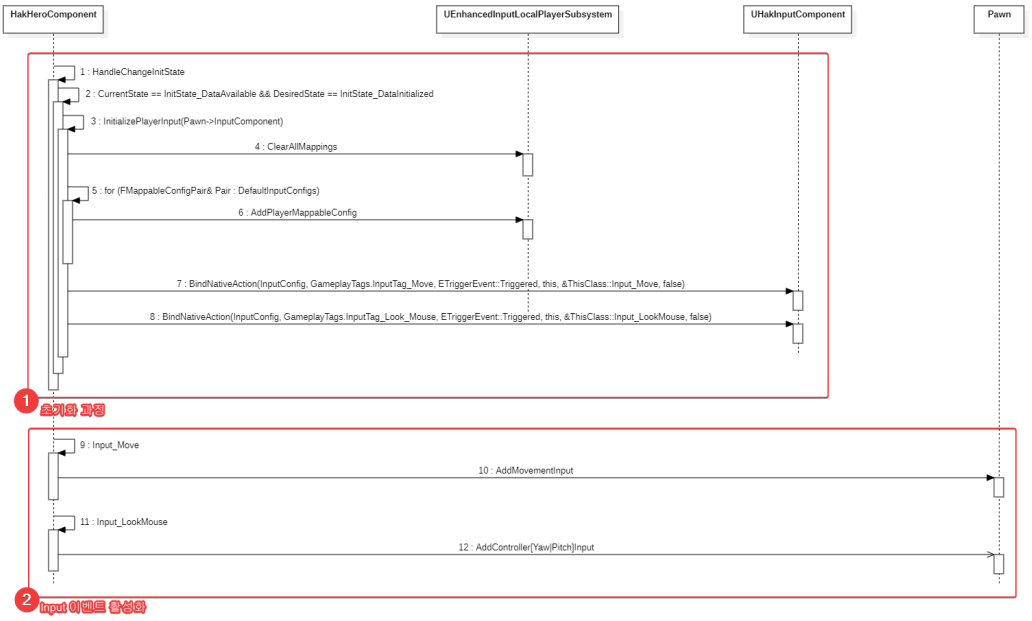


- PMI_Default_KBM 생성:



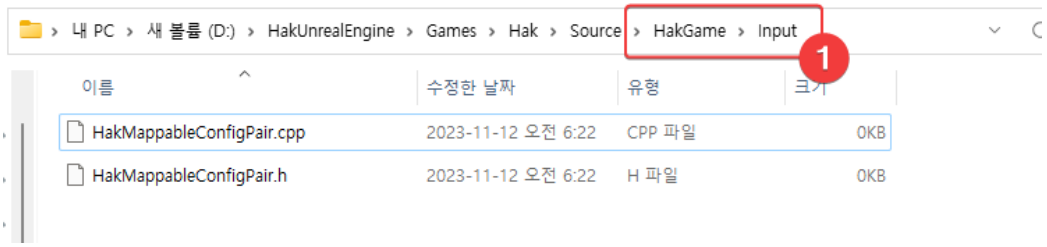


☐ 전체적인 Lyra에서 Input 초기화 및 업데이트 과정에 대한 순서도:



☐ FHakMappableConfigPair 구현:

- .h/.cpp 생성:



  - 꼭, **Input 폴더 생성하기**

- 코드 작성:

```h
#pragma once

#include "HakMappableConfigPair.generated.h"

/** forward declaration */
class UPlayerMappableInputConfig;

USTRUCT()
struct FHakMappableConfigPair
{
    GENERATED_BODY()
public:
    FHakMappableConfigPair() = default;

    UPROPERTY(EditAnywhere)
    TSoftObjectPtr<UPlayerMappableInputConfig> Config;

    UPROPERTY(EditAnywhere)
    bool bShouldActivateAutomatically = true;
};
```

```cpp
#include "HakMappableConfigPair.h"
#include "PlayerMappableInputConfig.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakMappableConfigPair)
```

☐ EnhancedInput 모듈 포함하기:

```cs
public class HakGame : ModuleRules
{
    public HakGame(ReadOnlyTargetRules Target) : base(Target)
    {
        PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;

        PublicDependencyModuleNames.AddRange(new string[] {
            "Core",
            "CoreUObject",
            "Engine",
            "InputCore",
            // GAS
            "GameplayTags",
            // Game Features
            "ModularGameplay",
            "GameFeatures",
            // Input
            "InputCore",
            "EnhancedInput",
        });

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

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

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

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

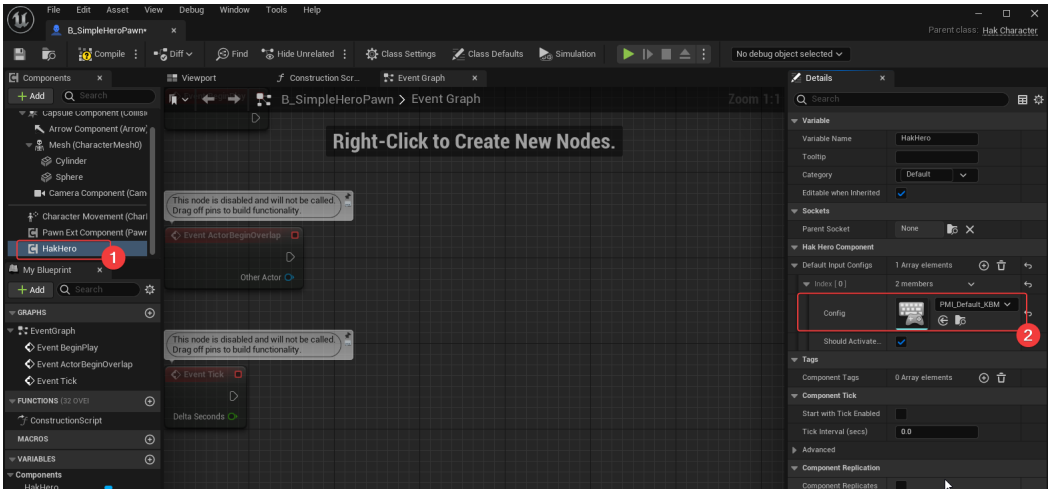
☐ HeroComponent에 MappableConfigPair 멤버 변수 선언하기:

```cpp
#pragma once

#include "Components/PawnComponent.h"
#include "Components/GameFrameworkInitStateInterface.h"
#include "HakHeroComponent.generated.h"

/** forward declaration */
class UHakCameraMode;
template<class TClass> class TSubclassOf;
struct FHakMappableConfigPair; // 1 struct! 주의하기

/**
* 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;

    /**
    * 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;

    /**
    * member variables
    */
    UPROPERTY(EditAnywhere) // 2
    TArray<FHakMappableConfigPair> DefaultInputConfigs;
};
```

☐ 헤더 파일을 추가해주기:

```cpp
#include "HakHeroComponent.h"
#include "HakPawnExtensionComponent.h"
#include "HakGame/HakLogChannels.h"
#include "HakGame/HakGameplayTags.h"
#include "HakGame/Player/HakPlayerState.h"
#include "HakGame/Character/HakPawnData.h"
#include "HakGame/Camera/HakCameraComponent.h"
#include "HakGame/Input/HakMappableConfigPair.h" // 1
#include "Components/GameFrameworkComponentManager.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakHeroComponent)
```

☐ HeroComponent에서 DefaultInputConfigs를 추가해주기:



☐ HakInputConfig.h/.cpp 구현:

```cpp
#pragma once

#include "Containers/Array.h"
#include "GameplayTagContainer.h"
#include "Engine/DataAsset.h"
#include "HakInputConfig.generated.h"

/** forward declaration */
class UInputAction;

/** HakInputAction은 GameplayTag와 InputAction을 연결하는 래퍼 클래스이다 */
USTRUCT(BlueprintType)
struct FHakInputAction
{
    GENERATED_BODY()
public:
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly)
    TObjectPtr<const UInputAction> InputAction = nullptr;

    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Meta=(Categories="InputTag"))
    FGameplayTag InputTag;
};

/**
 * HakInputConfig는 활성화 가능한 InputAction을 가지고 있는 클래스이다
 * - 이는 코드를 보면서, 무슨 설명인지 와닿을 것이다
 */
UCLASS(BlueprintType)
class UHakInputConfig : public UDataAsset
{
    GENERATED_BODY()
public:
    UHakInputConfig(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    const UInputAction* FindNativeInputActionForTag(const FGameplayTag& InputTag, bool bLogNotFound = true) const;
    const UInputAction* FindAbilityInputActionForTag(const FGameplayTag& InputTag, bool bLogNotFound = true) const;

    /**
     * member variables
     */
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Meta=(TitleProperty="InputAction"))
    TArray<FHakInputAction> NativeInputActions;

    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Meta = (TitleProperty = "InputAction"))
    TArray<FHakInputAction> AbilityInputActions;
};
```

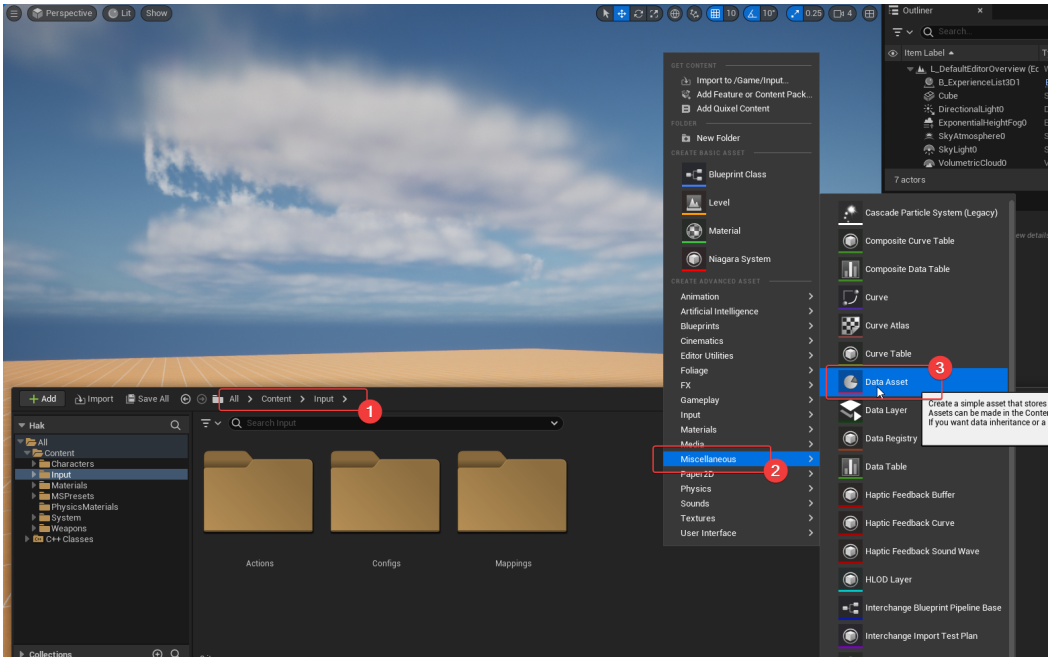
```cpp
#include "HakInputConfig.h"
#include "HakGame/HakLogChannels.h"

#include UE_INLINE_GENERATED_CPP_BY_NAME(HakInputConfig)

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

const UInputAction* UHakInputConfig::FindNativeInputActionForTag(const FGameplayTag& InputTag, bool bLogNotFound) const
{
    // NativeInputActions을 순회하며, Input으로 들어온 InputTag가 있는지 체크한다:
    // - 있으면, 그에 따른 InputAction을 반환하지만, 없다면, 그냥 nullptr을 반환한다.
    for (const FHakInputAction& Action : NativeInputActions)
    {
        if (Action.InputAction && (Action.InputTag == InputTag))
        {
            return Action.InputAction;
        }
    }

    if (bLogNotFound)
    {
        UE_LOG(LogHak, Error, TEXT("can't find NativeInputAction for InputTag [%s] on InputConfig [%s]."), *InputTag.ToString(), *GetNameSafe(this));
    }

    return nullptr;
}

const UInputAction* UHakInputConfig::FindAbilityInputActionForTag(const FGameplayTag& InputTag, bool bLogNotFound) const
{
    for (const FHakInputAction& Action : AbilityInputActions)
    {
        if (Action.InputAction && (Action.InputTag == InputTag))
        {
            return Action.InputAction;
        }
    }

    if (bLogNotFound)
    {
        UE_LOG(LogHak, Error, TEXT("Can't find AbilityInputAction for InputTag [%s] on InputConfig [%s]."), *InputTag.ToString(), *GetNameSafe(this));
    }

    return nullptr;
}
```

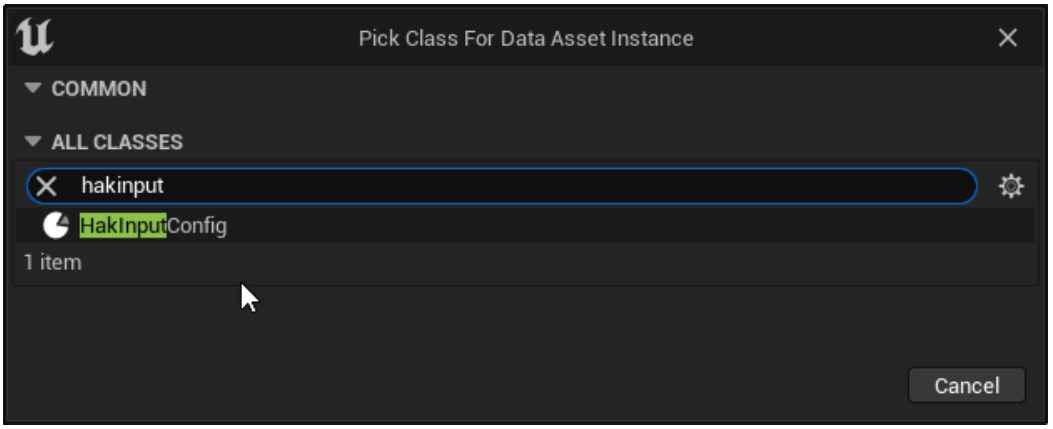
☐ PawnData에 추가시켜주자:

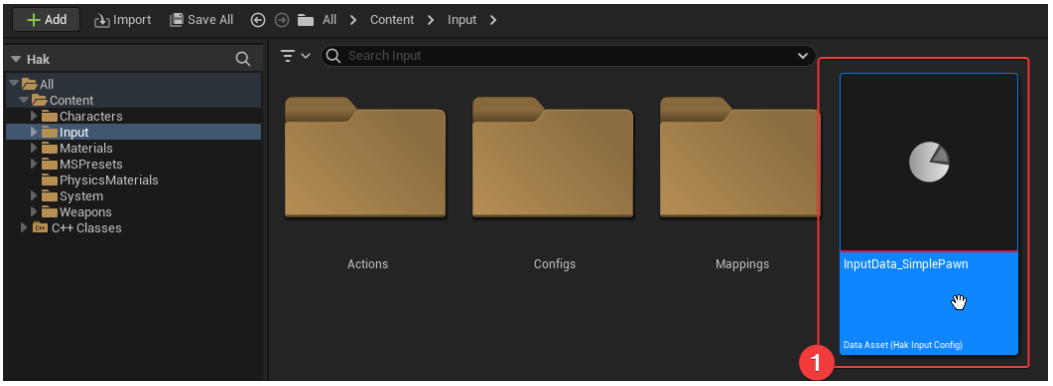
```cpp
/**
 * UHakPawnData
 * - non-mutable data asset that contains properties used to define a pawn
 */
UCLASS(BlueprintType)
class UHakPawnData : public UPrimaryDataAsset
{
    GENERATED_BODY()
public:
    UHakPawnData(const FObjectInitializer& ObjectInitializer);

    /** @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;
};
```

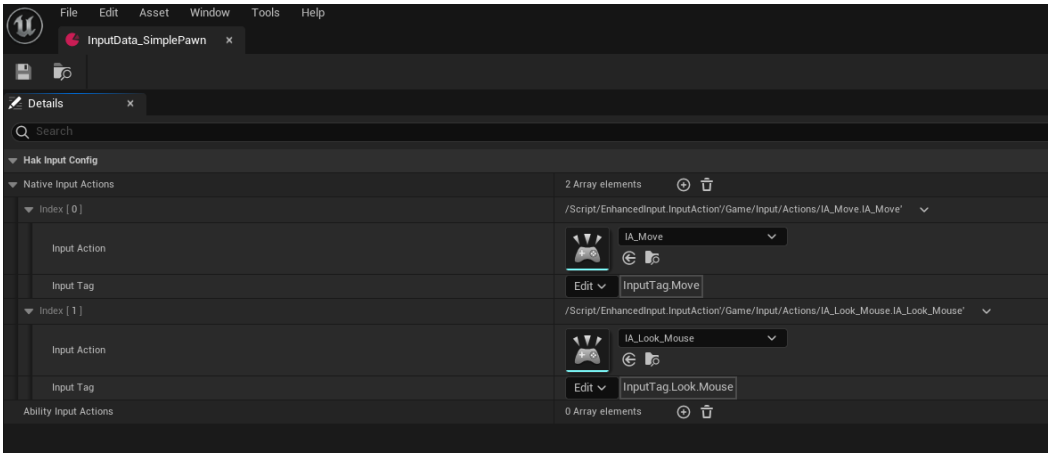
- InputConfig는 허용된 InputAction을 들고 있다.
- 이는 PawnData에 초기화 값으로서 존재한다.

☐ InputConfig 에셋을 만들자:

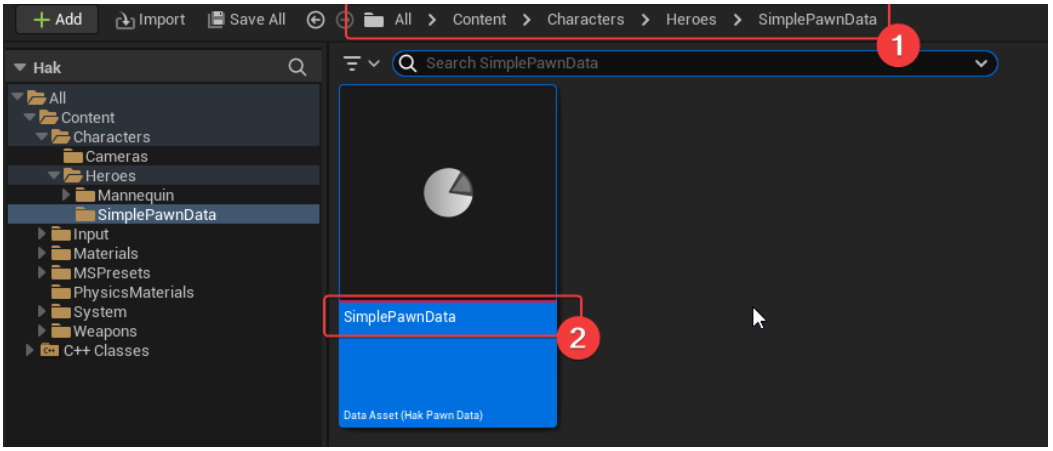

Add
Import
Save All
All
Content
Input
Search Input
Hak
All
Content
Characters
Input
Materials
MSPresets
PhysicsMaterials
System
Weapons
C++ Classes
Actions
Configs
Mappings
Create Basic Asset
Blueprint Class
Level
Material
Niagara System
Create Advanced Asset
Animation
Artificial Intelligence
Blueprints
Cinematics
Editor Utilities
Foliage
FX
Gameplay
Input
Materials
Media
Miscellaneous
Paper 2D
Physics
Sounds
Textures
User Interface
Cascade Particle System (Legacy)
Composite Curve Table
Composite Data Table
Curve
Curve Atlas
Curve Table
Data Asset
Data Layer
Data Registry
Data Table
Haptic Feedback Buffer
Haptic Feedback Curve
Haptic Feedback Sound Wave
HLOD Layer
Interchange Blueprint Pipeline Base
Interchange Import Test Plan
1
2
3

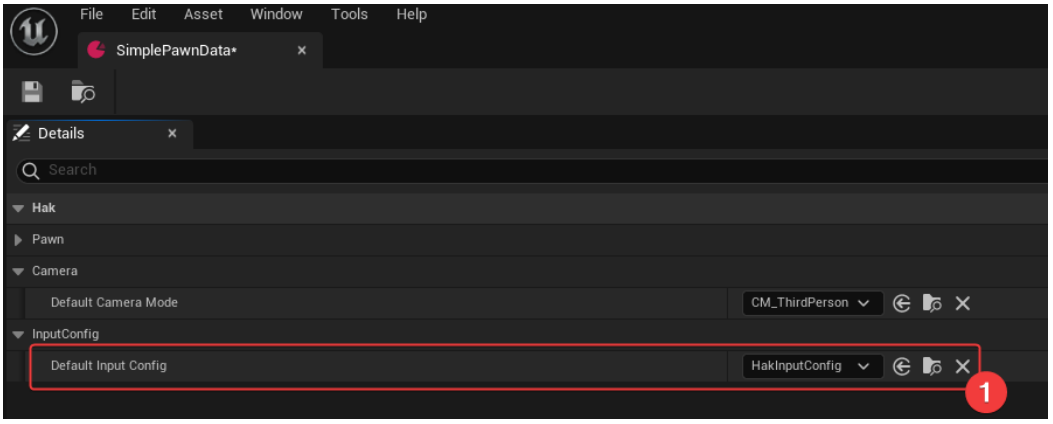

Pick Class For Data Asset Instance
COMMON
ALL CLASSES
hakinput
HakInputConfig
1 item
Cancel


Add
Import
Save All
All
Content
Input
Hak
Search Input
All
Content
Characters
Input
Materials
MSPresets
PhysicsMaterials
System
Weapons
C++ Classes
Actions
Configs
Mappings
InputData_SimplePawn
Data Asset (Hak Input Config)
1



☐ SimplePawnData에 업데이트 해주자:



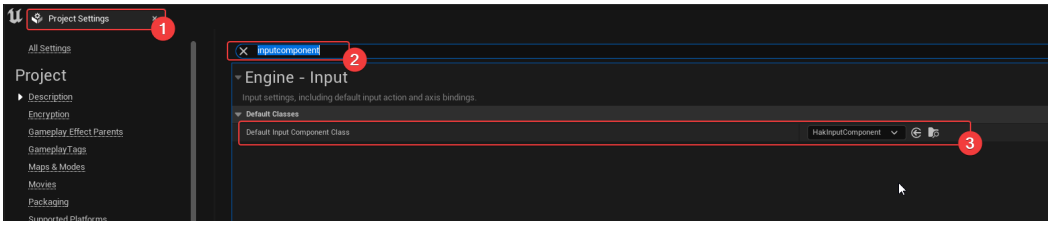


☐ HakInputComponent.h/.cpp 구현:

```h
#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, bool bLogIfNotFound);

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

template <class UserClass, typename FuncType>
void UHakInputComponent::BindNativeAction(const UHakInputConfig* InputConfig, const FGameplayTag& InputTag, ETriggerEvent TriggerEvent, UserClass* Object, FuncType Func, bool bLogIfNotFound)
{
	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, TArray<uint32>& BindHandles)
{
	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());
			}
		}
	}
}
```

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

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

- [ ] InputComponent 오버라이드:

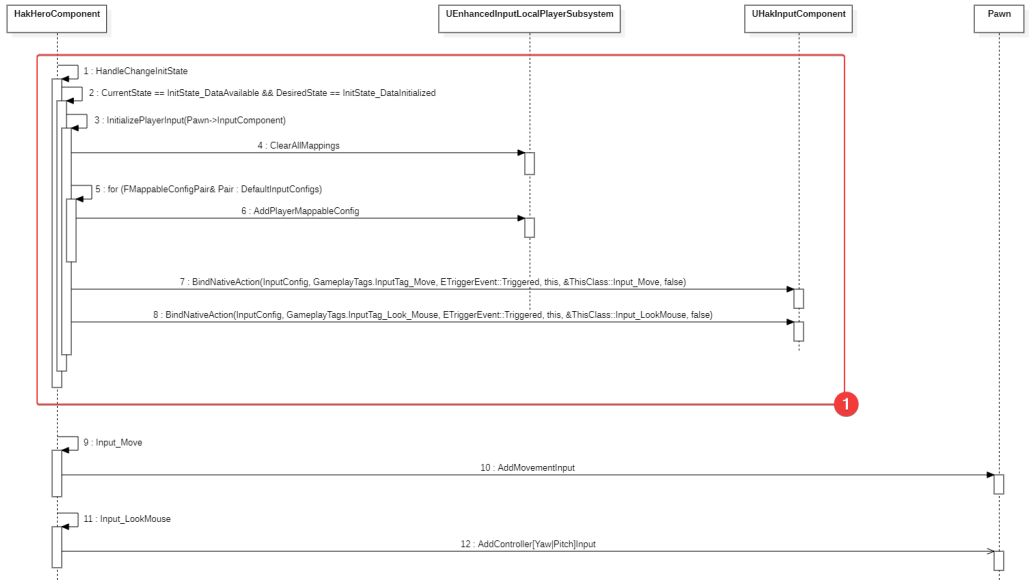


- [ ] 주석에 대한 설명하기

HeroComponent에 Input을 추가하자

▼ **펼치기**

- [ ] Input을 초기화 해주자:



- [ ] HeroComponent::HandleChangeInitState()

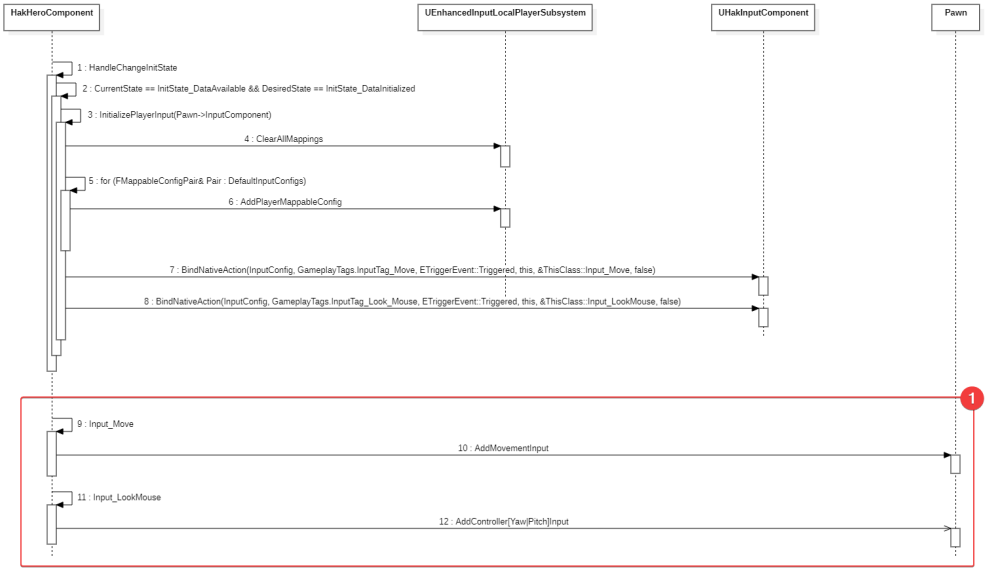
```cpp
void UHakHeroComponent::HandleChangeInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState)
{
    const FHakGameplayTags& InitTags = FHakGameplayTags::Get();

    // 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;
        }

        // Input과 Camera에 대한 핸들링... (TODO)

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

        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 InitializePlayerInput() 구현하자

- [ ] HeroComponent::InitializePlayerInput()
  - [ ] HakInputComponent.h 헤더 파일 포함시키기
  - [ ] #include "EnhancedInputSubsystems.h" 포함시키기
  - [ ] 코드 구현 및 설명:

```cpp
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);
                }
            }
        }
    }
}
```

☐ Input_Move()와 Input_LookMouse() 구현:



- Input_Move():

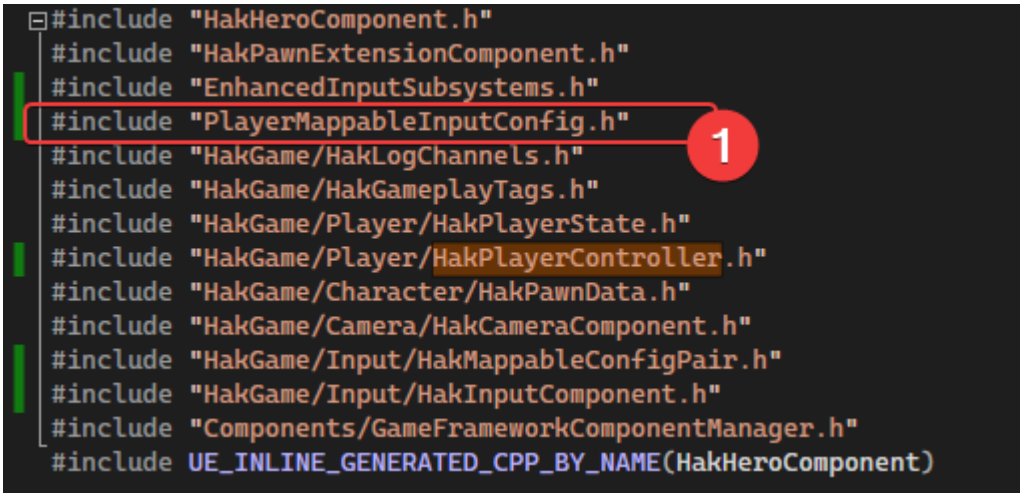
```cpp
void UHakHeroComponent::Input_Move(const FInputActionValue& InputActionValue)
{
    APawn* Pawn = GetPawn<APawn>();
    AController* Controller = Pawn ? Pawn->GetController() : nullptr;

    if (Controller)
    {
        const FVector2D Value = InputActionValue.Get<FVector2D>();
        const FRotator MovementRotation(0.0f, Controller->GetControlRotation().Yaw, 0.0f);

        if (Value.X != 0.0f)
        {
            // Left/Right -> X 값에 들어있음:
            // MovementDirection은 현재 카메라의 RightVector를 의미함 (World-Space)
            const FVector MovementDirection = MovementRotation.RotateVector(FVector::RightVector);

            // AddMovementInput 함수를 한번 보자:
            // - 내부적으로 MovementDirection * Value.X를 MovementComponent에 적용(더하기)해준다
            Pawn->AddMovementInput(MovementDirection, Value.X);
        }

        if (Value.Y != 0.0f)
        {
            // 앞서 Left/Right와 마찬가지로 Forward/Backward를 적용한다
            const FVector MovementDirection = MovementRotation.RotateVector(FVector::ForwardVector);
            Pawn->AddMovementInput(MovementDirection, Value.Y);
        }
    }
}
```

- Input_LookMouse():

```cpp
void UHakHeroComponent::Input_LookMouse(const FInputActionValue& InputActionValue)
{
    APawn* Pawn = GetPawn<APawn>();
    if (!Pawn)
    {
        return;
    }

    const FVector2D Value = InputActionValue.Get<FVector2D>();
    if (Value.X != 0.0f)
    {
        // X에는 Yaw 값이 있음:
        // - Camera에 대해 Yaw 적용
        Pawn->AddControllerYawInput(Value.X);
    }

    if (Value.Y != 0.0f)
    {
        // Y에는 Pitch 값!
        double AimInversionValue = -Value.Y;
        Pawn->AddControllerPitchInput(AimInversionValue);
    }
}
```

- 헤더파일 추가 잊지 말기:

```
#include "HakHeroComponent.h"
#include "HakPawnExtensionComponent.h"
#include "EnhancedInputSubsystems.h"
#include "PlayerMappableInputConfig.h"
#include "HakGame/HakLogChannels.h"
#include "HakGame/HakGameplayTags.h"
#include "HakGame/Player/HakPlayerState.h"
#include "HakGame/Player/HakPlayerController.h"
#include "HakGame/Character/HakPawnData.h"
#include "HakGame/Camera/HakCameraComponent.h"
#include "HakGame/Input/HakMappableConfigPair.h"
#include "HakGame/Input/HakInputComponent.h"
#include "Components/GameFrameworkComponentManager.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakHeroComponent)
```



☐ 잘 적용됨을 확인할 수 있다:

https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/22107c65-8763-415a-b69f-fdcccfb0868c/UnrealEditor_yn8RDeS0pd.mp4

자료

▼ 펼치기

PawnExtension_Hero.mdj