



7주차 (2023.10.30)

GameFrameworkComponentManager InitState

▼ 펼치기

- HeroComponent와 PawnExtensionComponent의 .h/.cpp 추가
- HeroComponent와 PawnExtensionComponent 초기적인 구현 진행:

- HakHeroComponent:

```
HakHeroComponent.cpp      HakHeroComponent.h      HakPawnExt...omponent.cpp      HakPawnExt...nComponent.h
Hak
1  #pragma once
2
3  #include "Components/PawnComponent.h"
4  #include "Components/GameFrameworkInitStateInterface.h"
5  #include "HakHeroComponent.generated.h"
6
7  /**
8  * component that sets up input and camera handling for player controlled pawns (or bots that simulate players)
9  * - this depends on a PawnExtensionComponent to coordinate initialization
10 */
11 /**
12 * 카메라, 입력 등 플레이어가 제어하는 시스템의 초기화를 처리하는 컴포넌트
13 */
14 UCLASS(Blueprintable)
15 class UHakHeroComponent : public UPawnComponent, public IGameFrameworkInitStateInterface
16 {
17     GENERATED_BODY()
18 public:
19     UHakHeroComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
};
```

```
HakHeroComponent.cpp      HakHeroComponent.h      HakPawnExt...omponent.cpp      HakPawnExt...nComponent.h
Hak
1  #include "HakHeroComponent.h"
2  #include UE_INLINE_GENERATED_CPP_BY_NAME(HakHeroComponent)
3
4  UHakHeroComponent::UHakHeroComponent(const FObjectInitializer& ObjectInitializer)
5      : Super(ObjectInitializer)
6  {
7      PrimaryComponentTick.bStartWithTickEnabled = false;
8      PrimaryComponentTick.bCanEverTick = false;
9 }
```

- HakPawnExtensionComponent:

```

1   #pragma once
2
3   #include "Components/GameFrameworkInitStateInterface.h"
4   #include "Components/PawnComponent.h"
5   #include "GameFramework/Actor.h"
6   #include "HakPawnExtensionComponent.generated.h"
7
8   /**
9    * 초기화 전반을 조정하는 컴포넌트
10   */
11
12   UCLASS()
13   class UHakPawnExtensionComponent : public UPawnComponent, public IGameFrameworkInitStateInterface
14   {
15       GENERATED_BODY()
16   public:
17       UHakPawnExtensionComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
18   };

```

```

1   #include "HakPawnExtensionComponent.h"
2   #include UE_INLINE_GENERATED_CPP_BY_NAME(HakPawnExtensionComponent)
3
4   UHakPawnExtensionComponent::UHakPawnExtensionComponent(const FObjectInitializer& ObjectInitializer)
5   : Super(ObjectInitializer)
6   {
7       // 기본적으로 Tick을 꺼버리자:
8       PrimaryComponentTick.bStartWithTickEnabled = false;
9       PrimaryComponentTick.bCanEverTick = false;
10  };

```

□ 간단한 HakPawnExtensionComponent와 HakHeroComponent에 대한 설명:

- `PawnExtensionComponent` :
 - 초기화 전반을 담당하는 Component
- `HeroComponent` :
 - 카메라, 입력 등 플레이어가 제어하는 시스템의 초기화를 처리하는 Component

□ HakCharacter에 대한 추가 구현:

- `HakCharacter.h`:

```

#pragma once
#include "GameFramework/Character.h"
#include "HakCharacter.generated.h"

/** forward declaration */
class UHakPawnExtensionComponent; 1

UCLASS()
class AHakCharacter : public ACharacter
{
    GENERATED_BODY()
public:
    AHakCharacter(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category="HakCharacter")
    TObjectPtr<UHakPawnExtensionComponent> PawnExtComponent; 2
};

```

- `HakCharacter.cpp`:

```

1  #include "HakCharacter.h"
2  #include "HakPawnExtensionComponent.h" ① forward declaration에 대한 헤더 포함
3  #include UE_INLINE_GENERATED_CPP_BY_NAME(HakCharacter)
4
5  AHakCharacter::AHakCharacter(const FObjectInitializer& ObjectInitializer)
6  {
7      : Super(ObjectInitializer)
8
9      // Tick을 비활성화
10     PrimaryActorTick.bCanEverTick = false;
11     PrimaryActorTick.bStartWithTickEnabled = false;
12
13     // PawnExtComponent 생성
14     PawnExtComponent = CreateDefaultSubobject<UHakPawnExtensionComponent>(TEXT("PawnExtensionComponent"));

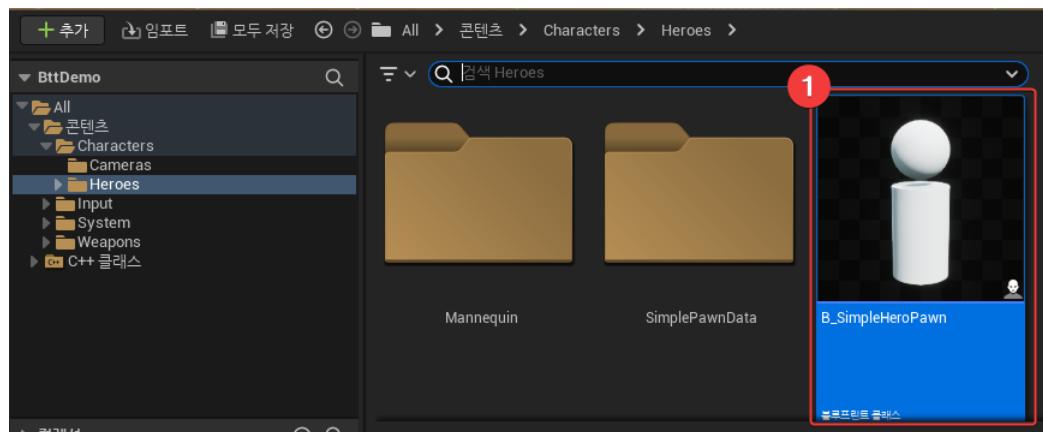
```

□ HeroComponent 추가하기:

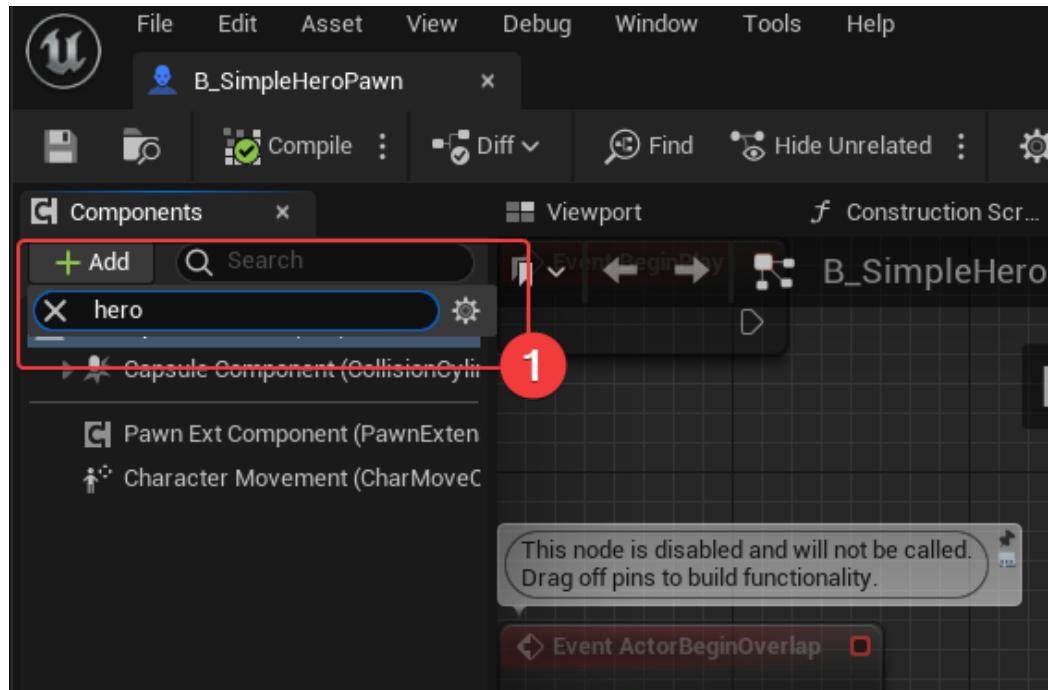


Lyra 프로젝트에서 HeroComponent는 BP에 직접 추가해주고 있다.

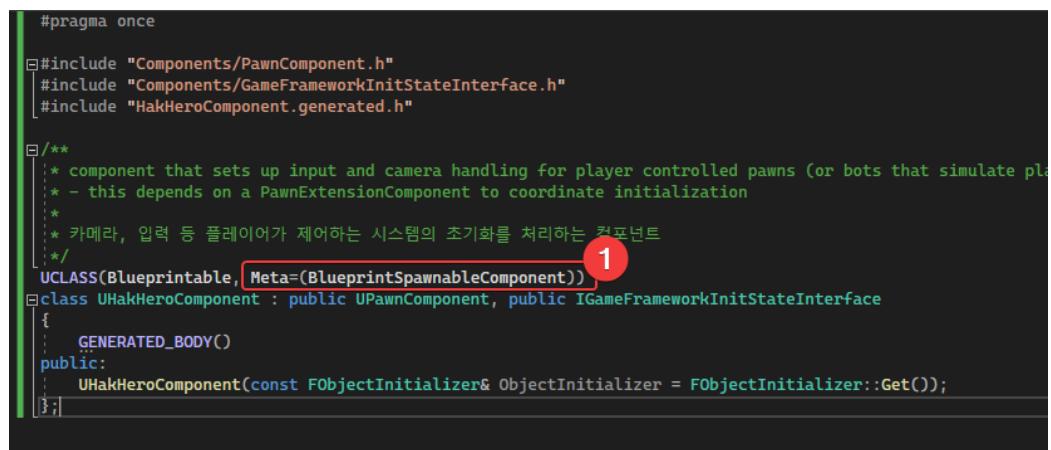
- B_SimpleHeroPawn에서 HeroComponent를 추가해주자:-



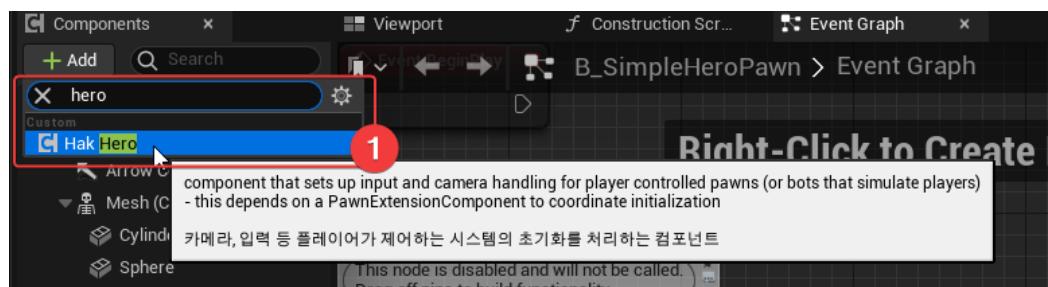
- HeroComponent를 소환할 수 없다:



- 아래와 같이 Meta에 BlueprintSpawnableComponent를 추가해야 한다:



- 아래와 같이 이제 추가 가능하다:



[PawnExtensionComponent]

OnRegister()

□ 구현하면 아래와 같다:

```
1  #pragma once
2
3  #include "Components/GameFrameworkInitStateInterface.h"
4  #include "Components/PawnComponent.h"
5  #include "GameFramework/Actor.h"
6  #include "HakPawnExtensionComponent.generated.h"
7
8  /**
9  * 초기화 전반을 조정하는 컴포넌트
10 */
11
12 UCLASS()
13 class UHakPawnExtensionComponent : public UPawnComponent, public IGameFrameworkInitStateInterface
14 {
15     GENERATED_BODY()
16 public:
17     UHakPawnExtensionComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
18
19     /**
20     * UPawnComponent interfaces
21     */
22     virtual void OnRegister() final;
23 }
```

```
1  #include "HakPawnExtensionComponent.h" 1
2  #include "HakGame/HakLogChannels.h"
3  #include UE_INLINE_GENERATED_CPP_BY_NAME(HakPawnExtensionComponent)
4
5  UHakPawnExtensionComponent::UHakPawnExtensionComponent(const FObjectInitializer& ObjectInitializer)
6      : Super(ObjectInitializer)
7  {
8      // 기본적으로 Tick을 꺼버리자:
9      PrimaryComponentTick.bStartWithTickEnabled = false;
10     PrimaryComponentTick.bCanEverTick = false;
11 }
12
13 PRAGMA_DISABLE_OPTIMIZATION
14 #include "Components/GameFrameworkComponentManager.h"
15 void UHakPawnExtensionComponent::OnRegister()
16 {
17     Super::OnRegister();
18
19     // 올바른 Actor에 등록되었는지 확인:
20     if (!GetPawn<APawn>())
21     {
22         UE_LOG(LogHak, Error, TEXT("this component has been added to a BP whose base class is not a Pawn!"));
23         return;
24     }
25
26
27     // GameFrameworkComponentManager에 InitState 사용을 위해 등록 진행:
28     // - 등록은 상속받았던 IGameFrameworkInitStateInterface 메서드 RegisterInitStateFeature()를 활용
29     // - 해당 함수를 간단히 보기
30     RegisterInitStateFeature();
31
32     // 디버깅을 위한 함수
33     if (UGameFrameworkComponentManager* Manager = UGameFrameworkComponentManager::GetForActor(GetOwningActor()))
34     {
35         Manager->...
36     }
37
38
39 PRAGMA_ENABLE_OPTIMIZATION
```

□ 간단히 RegisterInitStateFeature()를 살펴보기

- 그러나 내부 코드는 1단계 Depth 정도만 들어가고, 나머지는 코멘트를 참고해주시기를 바란다고 알려드리기 (심화 단계로서 수준이 있는 분들을 위한 숙제)



코멘트로 종종 강의 이후에 DM으로 질문 주시는 분들이 있는데
그분들의 질문이 심화적이라, 넘어가기 보다 제가 보는 관점에서
주석으로 정리하여 추가적인 숙제를 드립니다~ 😊

**조교님들이 이를 공유해 드릴 것이며, 혹시 누락되었다면, 요청
하시면 되겠습니까~!**

(여러다보니.. 제가 강의 준비에는 배의 시간이 듣다는...)

- 간단히 디버깅하며, ActorFeatureMap의 상태를 보자:

The screenshot shows the Unreal Engine Editor interface. In the top half, a code editor displays C++ code for a component registration function. A tooltip is visible over a comment about registering in the editor. In the bottom half, a Watch window shows the state of variables. The variable 'Manager' is set to a specific memory address and is of type UGameFrameworkComponentManager*. The variable 'ActorFeatureMap' is highlighted with a red circle and labeled '2 아무것도 등록되어있지 않다!' (2 Nothing is registered!). A red arrow points from this label to the 'ActorFeatureMap' entry in the Watch window, which shows it is empty with a Num=4 value.

```
13 // PRAGMA_DISABLE_OPTIMIZATION
14 #include "Components/GameFrameworkComponentManager.h"
15 void UHakPawnExtensionComponent::OnRegister()
16 {
17     Super::OnRegister();
18
19     // 올바른 Actor에 등록되었는지 확인:
20     if (!GetPawn<APawn>())
21     {
22         UE_LOG(LogHak, Error, TEXT("this component has been added to a BP whose base class is not a Pawn!"));
23         return;
24     }
25
26
27     // GameFrameworkComponentManager에 InitState 사용을 위해 등록 진행:
28     // - 등록은 상속받았던 IGameFrameworkInitStateInterface 메서드 RegisterInitStateFeature()를 활용
29     // - 해당
30     RegisterInitStateFeature();
31
32     // 디버깅을 위한 함수
33     UGameFrameworkComponentManager* Manager = UGameFrameworkComponentManager::GetForActor(GetOwningActor());
34
35     // 8ms elapsed
36
37     PRAGMA_ENABLE_OPTIMIZATION
38 }
```

Watch 1

| Name | Type | Value |
|-----------------------------|------------------------------------|--|
| Manager | UGameFrameworkComponentManager* | 0x00009239d50e00 (Name= "GameFrameworkComponentManager_0") |
| ActorFeatureMap | TMap<UObjectKey, UGameFramework... | Empty |
| ClassFeatureChangeDelegates | TMap<UObjectKey, UGameFramework... | empty |
| StateChangeQueue | TArray<TTuple<AActor ...> | -1 |
| CurrentStateChange | int | |

Output

```
[2023-10-29-10, onecore\efnet\thnet
[2023-10-29-10, [2023-10-29-10, [2023-10-29-10, [2023-10-29-10,
```

- 이유는 FeatureName이 정의되어 있지 않아서 그렇다:

```

1 #pragma once
2
3 #include "Components/GameFrameworkInitStateInterface.h"
4 #include "Components/PawnComponent.h"
5 #include "GameFramework/Actor.h"
6 #include "HakPawnExtensionComponent.generated.h"
7
8 /**
9 * 초기화 전반을 조정하는 커스텀
10 */
11
12 UCLASS()
13 class UHakPawnExtensionComponent : public UPawnComponent, public IGameFrameworkInitStateInterface
14 {
15     GENERATED_BODY()
16 public:
17     UHakPawnExtensionComponent(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
18
19     /** FeatureName 정의 */
20     static const FName NAME_ActorFeatureName;
21
22     /**
23     * UPawnComponent interfaces
24     */
25     virtual void OnRegister() final;
26
27     /**
28     * IGameFrameworkInitStateInterface
29     */
30     virtual FName GetFeatureName() const final { return NAME_ActorFeatureName; }
31

```

1
2

```

1 #include "HakPawnExtensionComponent.h"
2 #include "HakGame/HakLogChannels.h"
3 #include UE_INLINE_GENERATED_CPP_BY_NAME(HakPawnExtensionComponent)
4
5 /**
6 * feature name을 component 단위나 component를 빼고 pawn extension만 넣은 것을 확인할 수 있다
7 */
8 const FName UHakPawnExtensionComponent::NAME_ActorFeatureName("PawnExtension");
9
10 UHakPawnExtensionComponent::UHakPawnExtensionComponent(const FObjectInitializer& ObjectInitializer)
11     : Super(ObjectInitializer)
12 {
13     // 기본적으로 Tick을 꺼버리자:
14     PrimaryComponentTick.bStartWithTickEnabled = false;
15     PrimaryComponentTick.bCanEverTick = false;
16

```

1
2
3

- 위를 추가해주면, ActorFeatureMap에 올바르게 추가된 것을 확인할 수 있다:

| | | |
|-----------------------------------|--|------------------------------|
| Manager | 0x00000bb9b64f1800 (Name="GameFrameworkComponentManager_0") | UGameFrameworkCom... |
| ↳ UGameFrame... [1] | ↳ ComponentManager [1] (Name="GameFrameworkComponentManager_0") | ↳ UnrealEditor-ModularG... |
| ↳ UGameInstanc... [1] | ↳ UGameInstanc... [1] (Name="GameFrameworkComponentManager_0") | ↳ UGameInstancSubsys... |
| ↳ RequestTrac... [1] | ↳ Empty [1] | ↳ TMap<UGameFrame... |
| ↳ ComponentCl... [1] | ↳ Empty [1] | ↳ TMap<UClass *,TSet<F... |
| ↳ ReceiverClas... [1] | ↳ Empty [1] | ↳ TMap<UGameFramewo... |
| ↳ ReceiverClas... [1] | ↳ Empty [1] | ↳ TMap<UGameFramewo... |
| ↳ AllReceivers [1] | ↳ Empty [1] | ↳ TSet<FObjec... |
| ↳ InitStateOrd... [1] | ↳ Num=4 [1] | ↳ TArray<FGameplayTag,T... |
| ↳ ActorFeatu... [2] | ↳ Num=1 [2] | ↳ TMap<FObjec... |
| ↳ [0] [2] | ↳ {{ObjectIn... [2]} [2]} [2] | ↳ UGameFrameworkCom... |
| ↳ [0Key] [2] | ↳ [ObjectIndex=38384 ObjectSerialNumber=26376] [2] | ↳ FObjectKey |
| ↳ [1:Value] [2] | ↳ (ActorClass=0x00000bb9b378f500 (Name="B_SimpleHeroPawn_C") Registr... | ↳ UGameFrameworkCom... |
| ↳ ActorClass [2] | ↳ 0x00000bb9b378f500 (Name="B_SimpleHeroPawn_C") [2] | ↳ TWeakObjectPr...[UClass... |
| ↳ RegisteredStates [2] | ↳ Num=1 [2] | ↳ TArray<UGameFrame... |
| ↳ [0] [3] | ↳ (FeatureName="PawnExtension" [3]) [3] | ↳ TMap<FGam... |
| ↳ FeatureName [3] | ↳ "PawnExtension" [3] | ↳ FName |
| ↳ CurrentState [3] | ↳ {TagName="None"} [3] | ↳ FGameplayTag |
| ↳ Implementer [3] | ↳ 0x00000bb9b522f180 (Name = "PawnExtensionComponent", Owner = 0x0000... | ↳ TWeakObjectPr...[UObj... |
| ↳ [Raw View] [3] | ↳ {AllocatorInstance=...} [3] | ↳ TArray<UGameFrame... |
| ↳ Raw View [3] | ↳ ArrayNum=1 ArrayMax=4 [3] | ↳ TArray<UGameFrame... |
| ↳ RegisteredDelegates [3] | ↳ Empty [3] | ↳ TMap<FGam... |
| ↳ [Raw View] [3] | ↳ {...} [3] | ↳ TMap<FObjec... |
| ↳ ClassFeatureChangeDelegates [3] | ↳ {...} [3] | ↳ TMap<UGameFrame... |
| ↳ StateChangeQueue [3] | ↳ Empty [3] | ↳ TArray<TTuple<AActor ... |

□ BeginPlay/EndPlay

□ EndPlay부터 간단히 설명하자:

```

void UHakPawnExtensionComponent::EndPlay(const EEndPlayReason::Type EndPlayReason)
{
    // 앞서, OnRegister의 RegisterInitStateFeature()의 쌍을 맞추어주자
    UnregisterInitStateFeature();

    Super::EndPlay(EndPlayReason);
}

```

- 별거 없다 → 앞서 OnRegister()에서 호출했던 RegisterInitStateFeature()에 대한 쌍을 맞추기 위해 UnregisterInitStateFeature() 호출해주자:
 - 당연히 UGameFrameworkComponentManger에서 제거해주겠징?

□ BeginPlay()를 구현해보자:

```

void UHakPawnExtensionComponent::BeginPlay()
{
    Super::BeginPlay();

    // FeatureName에 NAME_None을 넣으면, Actor에 등록된 Feature Component의 InitState 상태를 관찰하겠다는 의미:
    BindOnActorInitStateChanged(NAME_None, FGameplayTag(), false);

    // InitState_Spawned로 상태 변환:
    // - TryToChangeInitState는 아래와 같이 진행된다:
    //   1. CanChangeInitState로 상태 변화 가능성 유무 판단
    //   2. HandleChangeInitState로 내부 상태 변화 (Feature Component)
    //   3. BindOnActorInitStateChanged로 Bind된 Delegate를 조건에 맞게 호출해 줌
    // - HakPawnExtensionComponent의 경우, 모든 Actor의 Feature 상태 변화에 대해 OnActorInitStateChanged()가 호출됨
    ensure(TryToChangeInitState(FHakGameplayTags::Get().InitState_Spawned));

    // 해당 함수는 우리가 오버라이딩 한다:
    // - 이 함수를 ForceUpdateInitState와 같은 느낌으로 이해해주면 된다
    // - 현재 강제 업데이트 진행 (물론 CanChangeInitState와 HandleChangeInitState를 진행해준다)
    CheckDefaultInitialization();
}

```

□ 간단한 코드 설명

□ IGameFrameworkInitStateInterface 오버라이딩:

```

/** 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;

    /**
     * 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;
};

```

```

void UHakPawnExtensionComponent::OnActorInitStateChanged(const FActorInitStateChangedParams& Params)
{
    ...
}

Ebool UHakPawnExtensionComponent::CanChangeInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState) const
{
    ...
    return false;
}

void UHakPawnExtensionComponent::CheckDefaultInitialization()
{
    ...
}

```

- BindOnActorInitStateChanged()의 대상이 되는 OnActorInitStateChanged:
- CanChangeInitState():
- 강제 업데이트 메서드로 CheckDefaultInitialization():
 - 참고로 해당 메서드는 CheckDefaultInitializationForImplementers()를 통해 불리기도 한다:
 - CheckDefaultInitializationForImplementers은 나를 제외한 Actor에 바인딩된 Feature Component에 대해 CheckDefaultInitialization을 호출한다:
 - 참고로, CheckDefaultInitialization()은 비어있다:

```
/** Override to try and progress the default initialization path, likely using ContinueInitStateChain */
virtual void CheckDefaultInitialization() {}
```

□ CheckDefaultInitialization() 구현:

```
void UHakPawnExtensionComponent::CheckDefaultInitialization()
{
    // PawnExtensionComponent는 Feature Component들의 초기화를 관리하는 Component이다:
    // - 따라서, Actor와 바인딩된 Feature Component들에 대해 CheckDefaultInitialization을 호출해주도록 한다 (ForceUpdate 노름?):
    // - 이 메서드를 IDefaultInitializationInterface가 제공하는데, CheckDefaultInitializationForImplementers이다:
    // - 간단히 CheckDefaultInitializationForImplementers 보자:
    CheckDefaultInitializationForImplementers();

    const FHakGameplayTags& InitTags = FHakGameplayTags::Get();
    // 사용자 정의 InitState를 직접 넘겨줘야 한다... (사실 이런 종 일에서 할 수 있을거 같은데 굳이...)
    static const TArray<FGameplayTag> StateChain = { InitTags.InitState_Spawned, InitTags.InitState_DataAvailable, InitTags.InitState_DataInitialized, InitTags.InitState_ConditionAvailable };
    // CanChangeInitState()와 HandleChangeInitState() 그리고 ChangeFeatureInitState 호출을 통한 OnActorInitStateChanged Delegate 호출까지 진행해준다:
    ContinueInitStateChain(StateChain);
}
```

□ 코드 설명



일단 여기까지 하고, HeroComponent도 PawnExtComponent와 비슷한 상태까지 OnActorInitStateChanged와 CanChangeInitState 그리고 HandleChangeInitState까지 인터페이스 구현까지 해보자.

[HeroComponent]

- 앞서, PawnExtensionComponent를 구현했으니, 빠르게 필요한 부분을 Clone 해보자:

```


    /**
     * 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());
        1    /** FeatureName 정의 */
        static const FName NAME_ActorFeatureName;

        2    /**
         * 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;
    };


```

```


1    /** FeatureName 정의: static member variable 초기화 */
    static const FName UHakHeroComponent::NAME_ActorFeatureName("Hero");

2    /**
     * UHakHeroComponent::OnRegister()
     */
    void UHakHeroComponent::OnRegister()
    {
        Super::OnRegister();

        // 올바른 Actor에 등록되었는지 확인:
        if (!GetPawn<APawn>())
        {
            UE_LOG(LogHak, Error, TEXT("this component has been added to a BP whose base class is not a Pawn!"));
            return;
        }

        RegisterInitStateFeature();
    }

3    /**
     * UHakHeroComponent::BeginPlay()
     */
    void UHakHeroComponent::BeginPlay()
    {
        Super::BeginPlay();

        // PawnExtensionComponent에 대해서 (PawnExtension Feature) OnActorInitStateChanged() 관찰하도록 (Observing)
        BindOnActorInitStateChanged(UHakPawnExtensionComponent::NAME_ActorFeatureName, FGameplayTag(), false);

        // InitState_Changed로 초기화
        ensure(TryToChangeInitState(FHakGameplayTags::Get().InitState_Spawned));

        // ForceUpdate 진행
        CheckDefaultInitialization();
    }

3    /**
     * UHakHeroComponent::EndPlay(const EEndPlayReason::Type EndPlayReason)
     */
    void UHakHeroComponent::EndPlay(const EEndPlayReason::Type EndPlayReason)
    {
        UnregisterInitStateFeature();
        Super::EndPlay(EndPlayReason);
    }

3    /**
     * IGameFrameworkInitStateInterface
     */
    void UHakHeroComponent::OnActorInitStateChanged(const FActorInitStateChangedParams& Params)

3    /**
     * UHakHeroComponent::CanChangeInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState) const
     */
    bool UHakHeroComponent::CanChangeInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState) const
    {
        return false;
    }

3    /**
     * UHakHeroComponent::HandleChangeInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState)
     */
    void UHakHeroComponent::HandleChangeInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState)
    {
    }

3    /**
     * UHakHeroComponent::CheckDefaultInitialization()
     */
    void UHakHeroComponent::CheckDefaultInitialization()
    {
        // 앞서 BindOnActorInitStateChanged에서 보았듯이 Hero Feature는 Pawn Extension Feature에 종속되어 있으므로, CheckDefaultInitializationForImplementers 호출하지 않음:

        // ContinueInitStateChain은 앞서 PawnExtensionComponent와 같음
        const FHakGameplayTags& InitTags = FHakGameplayTags::Get();
        static const TArray<FGameplayTag> StateChain = { InitTags.InitState_Spawned, InitTags.InitState_DataAvailable, InitTags.InitState_DataInitialized, InitTags.InitState_Condition };
        ContinueInitStateChain(StateChain);
    }


```

□ BeginPlay()에 대한 코드 설명

- CheckDefaultInitialization() 코드 설명 (Pawn Extension과 차이점 중심으로)
- HandleChangeInitState()
 - HeroComponent에서만 오버라이드하는 메서드
- 코드 설명

```
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)
    }
}
```

[CanChangeInitState]

PawnExtensionComponent와 HeroComponent의 CanChangeInitState()를 통해 어떻게 State Machine 변화가 일어나는지 확인해보자:

- PawnExtensionComponent

- PawnData 정의:

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

/** forward declaration */
class UHakPawnData; 1

/**
 * 초기화 전반을 조정하는 커스텀
 */

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

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

    /**
     * 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; 2
};
```

- 코드 설명:

```

bool UHakPawnExtensionComponent::CanChangeInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState) const
{
    check(Manager);

    APawn* Pawn = GetPawn<APawn>();
    const FHakGameplayTags& InitTags = FHakGameplayTags::Get();

    // InitState_Spawned 초기화
    if (!CurrentState.IsValid() && DesiredState == InitTags.InitState_Spawned)
    {
        // Pawn이 잘 세팅 안 되어있으면 바로 Spawned로 넘어감!
        if (Pawn)
        {
            return true;
        }
    }

    // Spawner -> DataAvailable
    if (CurrentState == InitTags.InitState_Spawned && DesiredState == InitTags.InitState_DataAvailable)
    {
        // 아마 PawnData를 누군가 설정하는 모양이다
        if (!IPawnData)
        {
            return false;
        }

        // LocallyControlled인 Pawn의 Controller가 없으면 예외!
        const bool bIsLocallyControlled = Pawn->IsLocallyControlled();
        if (!bIsLocallyControlled)
        {
            if (!GetController<AController>())
            {
                return false;
            }
        }
    }

    return true;
}

// DataAvailable -> DataInitialized
if (CurrentState == InitTags.InitState_DataAvailable && DesiredState == InitTags.InitState_DataInitialized)
{
    // Actor에 바인드된 모든 Feature들이 DataAvailable 상태일 때, DataInitialized로 넘어감:
    // - HaveAllFeaturesReachedInitState 확인
    return Manager->HaveAllFeaturesReachedInitState(Pawn, InitTags.InitState_DataAvailable);
}

// DataInitialized -> GameplayReady
if (CurrentState == InitTags.InitState_DataInitialized && DesiredState == InitTags.InitState_GameplayReady)
{
    return true;
}

// 위의 선형적인(linear) transition이 아니면 false!
return false;
}

```

- 모든 각각의 InitState에 따른 변화를 구현 해놓음 (선형적!)
- Spawned → DataAvailable일 경우, 누군가 PawnExtComponent의 PawnData를 업데이트 해놓아야 한다!

□ HeroComponent

□ 코드 설명:

```

bool UHakHeroComponent::CanChangeInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState) const
{
    check(Manager);

    const FHakGameplayTags& InitTags = FHakGameplayTags::Get();
    APawn* Pawn = GetPawn<APawn>();
    AHakPlayerState* HakPS = GetPlayerState<AHakPlayerState>();

    // Spawmed 초기화
    if (!CurrentState.IsValid() && DesiredState == InitTags.InitState_Spawned)
    {
        if (Pawn)
        {
            return true;
        }
    }

    // Spawmed -> DataAvailable
    if (CurrentState == InitTags.InitState_Spawned && DesiredState == InitTags.InitState_DataAvailable)
    {
        if (!HakPS)
        {
            return false;
        }

        return true;
    }

    // DataAvailable -> DataInitialized
    if (CurrentState == InitTags.InitState_DataAvailable && DesiredState == InitTags.InitState_DataInitialized)
    {
        // PawnExtensionComponent가 DataInitialized될 때까지 기다림 (== 모든 Feature Component가 DataAvailable인 상태)
        return HakPS && Manager->HasFeatureReachedInitState(Pawn, UHakPawnExtensionComponent::NAME_ActorFeatureName, InitTags.InitState_DataInitialized);
    }

    // DataInitialized -> GameplayReady
    if (CurrentState == InitTags.InitState_DataInitialized && DesiredState == InitTags.InitState_GameplayReady)
    {
        return true;
    }

    return false;
}

```

- HasFeatureReachdInitState() 관련 설명:
 - PawnExtensionComponent가 DataInitialized로 도달하는 의미:
 - 모든 Feature Component들이 DataAvailable 상태로 준비됨

[OnActorInitStateChanged]

PawnExtensionComponent

구현 및 코드 설명:

```

void UHakPawnExtensionComponent::OnActorInitStateChanged(const FActorInitStateChangedParams& Params)
{
    if (Params.FeatureName != NAME_ActorFeatureName)
    {
        // HakPawnExtensionComponent는 다른 Feature Component들의 상태가 DataAvailable를 관찰하여, Sync를 맞추는 구간이 있었다 (CanChangeInitState)
        // – 이를 가능케 하기 위해, OnActorInitStateChanged에서는 DataAvailable에 대해 지속적으로 CheckDefaultInitialization을 호출하여, 상태를 확인한다
        const FHakGameplayTags& InitTags = FHakGameplayTags::Get();
        if (Params.FeatureState == InitTags.InitState_DataAvailable)
        {
            CheckDefaultInitialization();
        }
    }
}

```

HeroComponent:

구현 및 코드 설명:

```

    /**
     * IGameFrameworkInitStateInterface
     */
    void UHakHeroComponent::OnActorInitStateChanged(const FActorInitStateChangedParams& Params)
    {
        const FHakGameplayTags& InitTags = FHakGameplayTags::Get();
        if (Params.FeatureName == UHakPawnExtensionComponent::NAME_ActorFeatureName)
        {
            // HakPawnExtensionComponent의 DataInitialized 상태 변화 관찰 후, HakHeroComponent도 DataInitialized 상태로 변경
            // - CanChangeInitState 확인
            if (Params.FeatureState == InitTags.InitState_DataInitialized)
            {
                CheckDefaultInitialization();
            }
        }
    }

```

[PawnExtensionComponent의 PawnData 설정]

- 아직 InitState 변경에 있어 PawnExtensionComponent에서 Spawnd → DataAvailable로 넘어갈 경우, PawnData 설정을 누군가 해줘야 선형적(Linear)하게 InitState 처리를 이어나갈 수가 있다:

```

bool UHakPawnExtensionComponent::CanChangeInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState) const
{
    check(Manager);

    APawn* Pawn = GetPawn<APawn>();
    const FHakGameplayTags& InitTags = FHakGameplayTags::Get();

    // InitState_Spawned 초기화
    if (!CurrentState.IsValid() && DesiredState == InitTags.InitState_Spawned)
    {
        // Pawn이 잘 세팅만 되어있으면 바로 Spawnd로 넘어감!
        if (Pawn)
        {
            return true;
        }
    }

    // Spawnd -> DataAvailable
    if (CurrentState == InitTags.InitState_Spawned && DesiredState != InitTags.InitState_DataAvailable)
    {
        // 아마 PawnData를 누군가 설정하는 모양이다
        if (!PawnData)
        {
            return false;
        }
    }

    // LocallyControlled의 Pawn이 Controller가 없으면 예외!
    const bool bIsLocallyControlled = Pawn->IsLocallyControlled();
    if (!bIsLocallyControlled)
    {
        if (!GetController<AController>())
        {
            return false;
        }
    }

    return true;
}

// DataAvailable -> DataInitialized
if (CurrentState == InitTags.InitState_DataAvailable && DesiredState == InitTags.InitState_DataInitialized)
{
    // Actor에 바인드된 모든 Feature들이 DataAvailable 상태일 때, DataInitialized로 넘어감:
    // - HaveAllFeaturesReachedInitState 확인
    return Manager->HaveAllFeaturesReachedInitState(Pawn, InitTags.InitState_DataAvailable);
}

// DataInitialized -> GameplayReady
if (CurrentState == InitTags.InitState_DataInitialized && DesiredState == InitTags.InitState_GameplayReady)
{
    return true;
}

// 위의 선형적인(Linear) transition이 아니면 false!
return false;
}

```

1

- 잠시 고민해보면, PawnData는 Character를 Spawn할 경우, 활용되는 데이터였다:
 - Experience 로딩 이후, Character가 Spawn되고 나서 PawnData를 설정하면 되겠다

그러기 위해, 앞서 오버라이드했던 SpawnDefaultPawnAtTransform_Implementation을 재구현해주면 되겠다:

- 기존 Super::SpawnDefaultPawnAtTransform_Implementation() 활용하여, 재구현:

```
APawn* AHakGameMode::SpawnDefaultPawnAtTransform_Implementation(AController* NewPlayer, const FTransform& SpawnTransform)
{
    FActorSpawnParameters SpawnInfo;
    SpawnInfo.Instigator = GetInstigator();
    SpawnInfo.ObjectFlags |= RF_Transient;
    SpawnInfo.bDeferConstruction = true;

    if (UClass* PawnClass = GetDefaultPawnClassForController(NewPlayer))
    {
        if (APawn* SpawnedPawn = GetWorld()->SpawnActor<APawn>(PawnClass, SpawnTransform, SpawnInfo))
        {
            // FindPawnExtensionComponent 구현
            if (UHakPawnExtensionComponent* PawnExtComp = UHakPawnExtensionComponent::FindPawnExtensionComponent(SpawnedPawn))
            {
                if (const UHakPawnData* PawnData = GetPawnDataForController(NewPlayer))
                {
                    PawnExtComp->SetPawnData(PawnData); 1
                }
            }
            SpawnedPawn->FinishSpawning(SpawnTransform);
            return SpawnedPawn;
        }
    }

    return nullptr;
}
```

SetPawn 구현:

```
void UHakPawnExtensionComponent::SetPawnData(const UHakPawnData* InPawnData)
{
    // Pawn에 대해 Authority가 없을 경우, SetPawnData는 진행하지 않음
    APawn* Pawn = GetPawnChecked<APawn>();
    if (Pawn->GetLocalRole() != ROLE_Authority)
    {
        return;
    }

    if (InPawnData)
    {
        return;
    }

    // PawnData 업데이트
    PawnData = InPawnData;

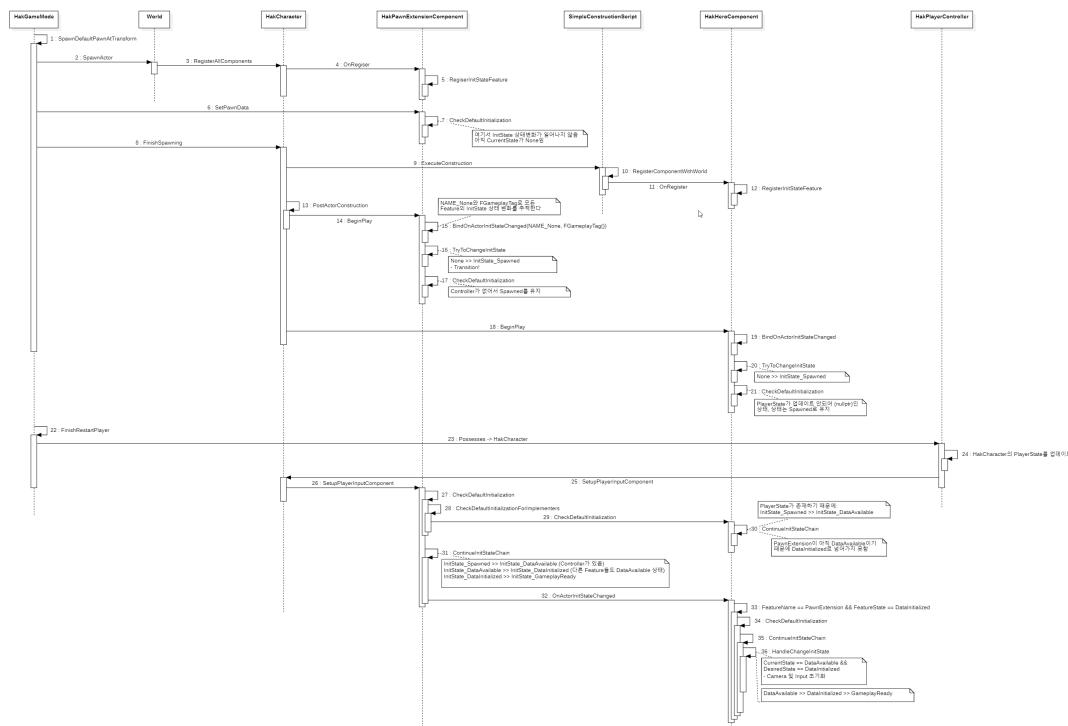
    // PawnData가 업데이트 되었으므로, InitState에서 Spawed -> DataAvailbe Transition을 호출
    CheckDefaultInitialization();
}
```

- CheckDefaultInitialization() 호출을 주목하자!

InitState 순서도 및 디버깅

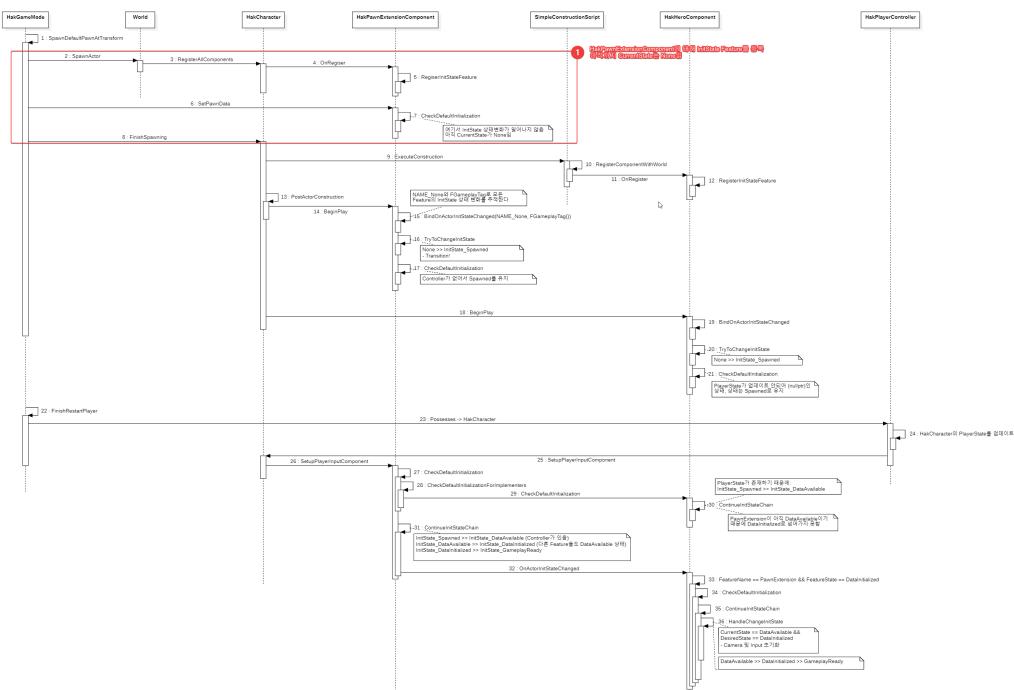
▼ 펼치기

순서도를 잠깐 보며, 앞서 구현한 내용을 다시 확인 해보자:



하나씩 디버깅해보며, 해당 과정을 살펴보자:

□ PawnExtension의 OnRegister:

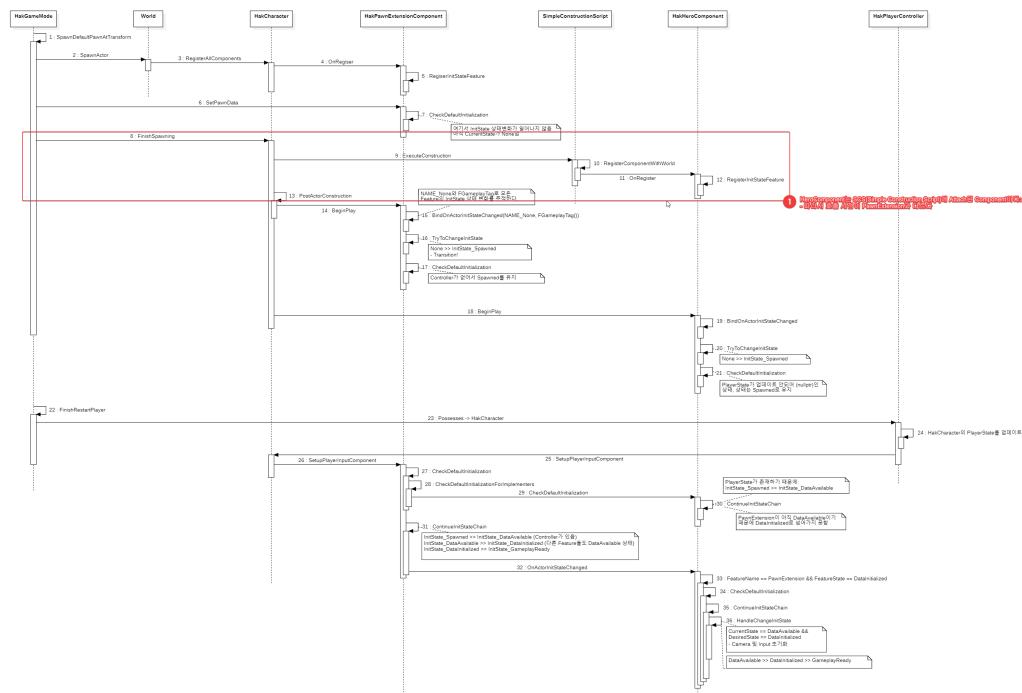


□ 코드를 디버깅 해보며, CurrentState를 변화를 확인해보자:

- CheckDefaultInitialization에 ContinueInitStateChain은 **None**에 대한 처리가 없다 → 따라서, 상태 변화는 없다!

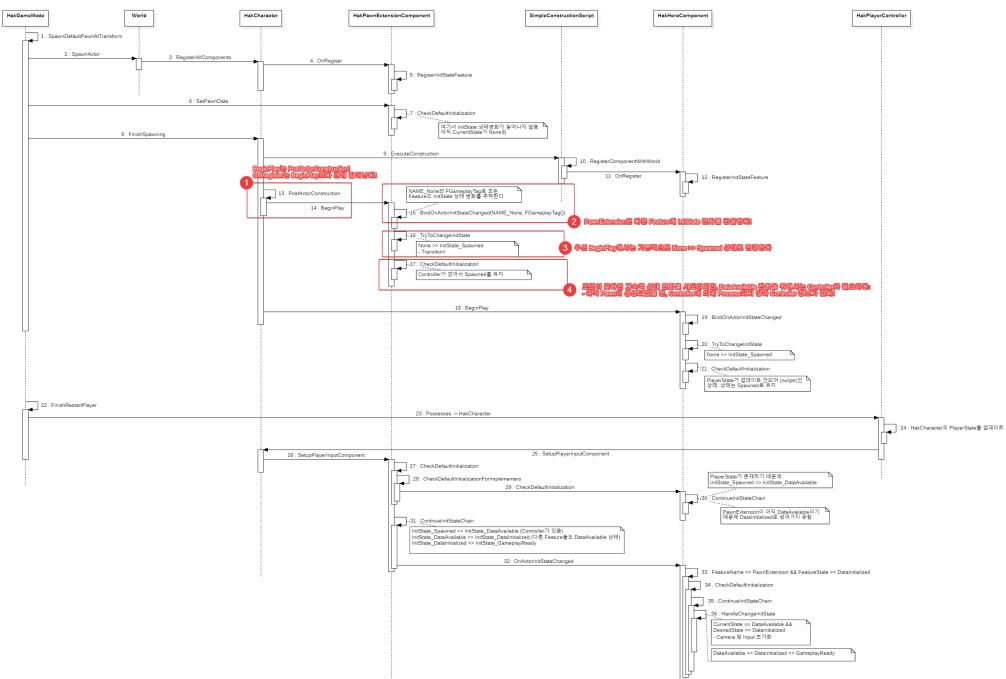
- SetPawn()이 HeroComponent의 OnRegister보다 먼저 불린다!

HeroComponent의 OnRegister:



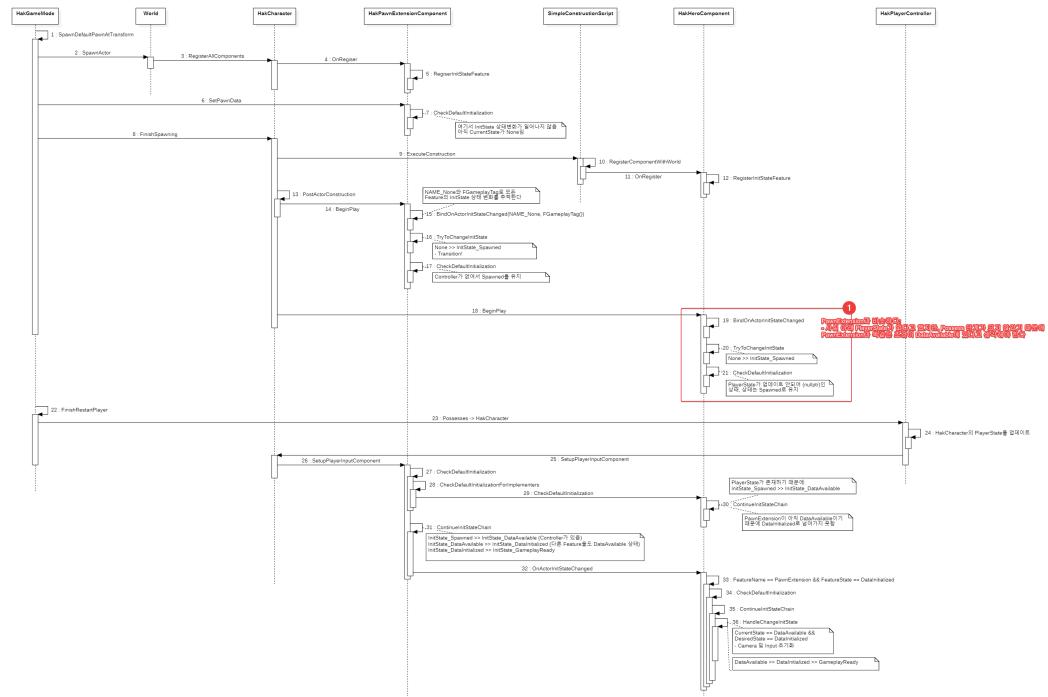
간단히 콜스택을 보면, HeroComponent의 SCS 초기화 시점을 확인해보자:

PawnExtension의 BeginPlay:



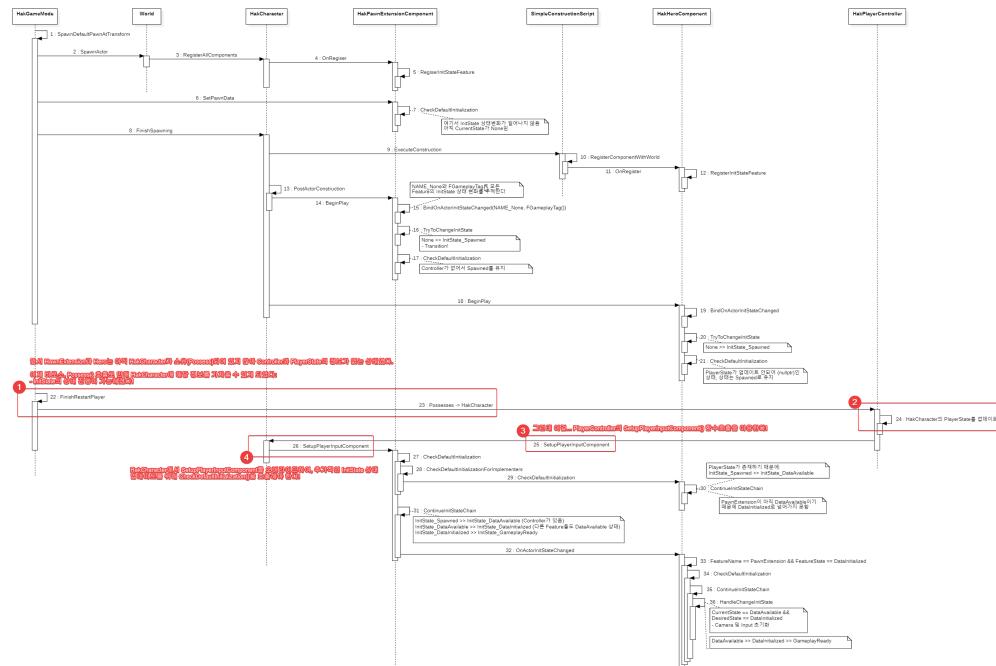
□ 코드 디버깅

□ HeroComponent의 BeginPlay:



□ 코드 디버깅

□ SetupPlayerInputComponent:



□ SetupPlayerInputComponent 구현:

```
/** forward declaration */
class UHakPawnExtensionComponent;

UCLASS()
class AHakCharacter : public ACharacter
{
    GENERATED_BODY()
public:
    AHakCharacter(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

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

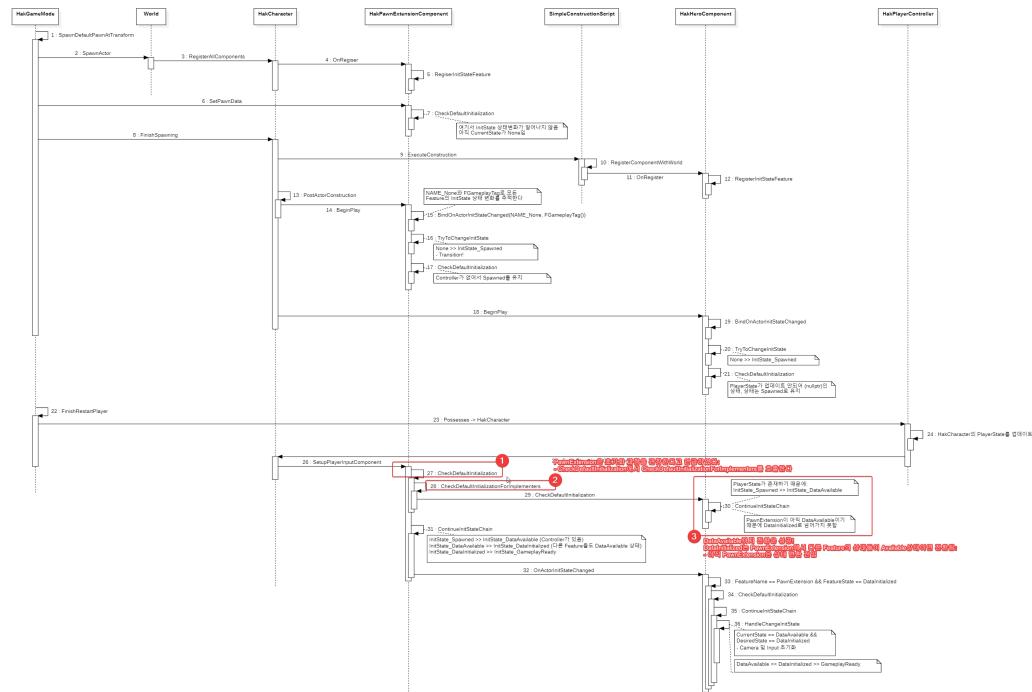
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category="Hak|Character")
    TObjectPtr<UHakPawnExtensionComponent> PawnExtComponent;
};
```

```
void AHakCharacter::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);

    // Pawn이 Possess로서, Controller와 PlayerState 정보 접근이 가능한 상태가 되었음:
    // - SetupPlayerInputComponent 확인
    PawnExtComponent->SetupPlayerInputComponent();
}
```

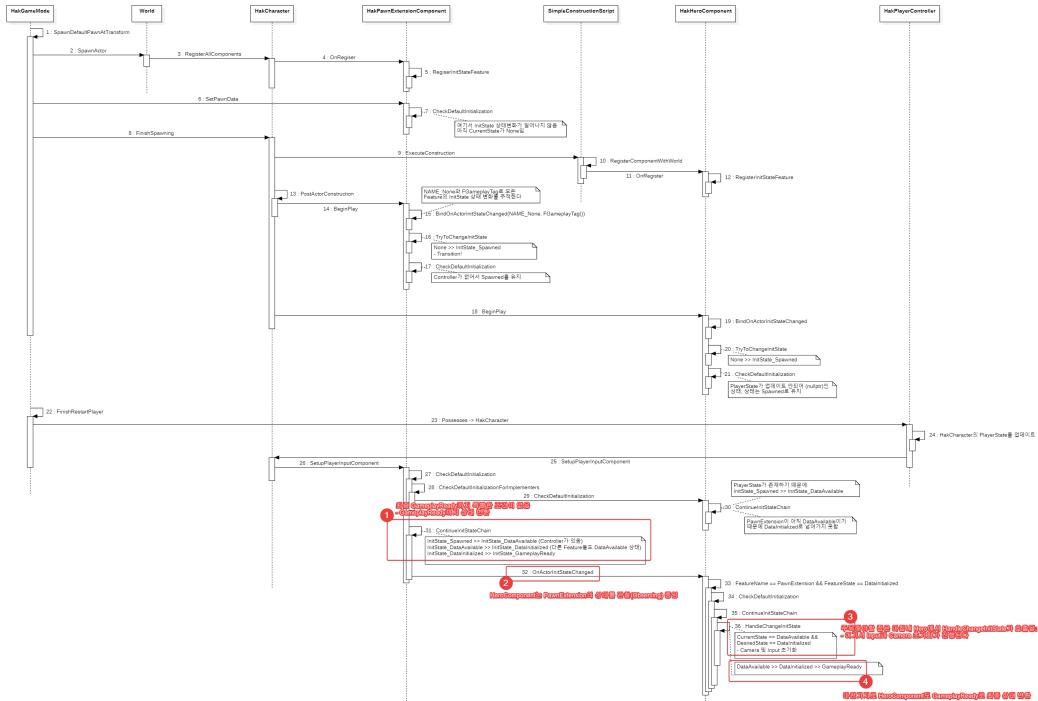
- 코드 디버깅을 통해, Spawned → DataAvailable로 바뀌는지 확인해보기

- CheckDefaultInitializationForImplementers:



- ## □ 코드 디버깅

- ### InitState 상태 변화:



□ 코드 디버깅

□ InitState에 대한 정리:

- InitState는 우리가 Lyra를 프레임워크로써 활용하는데 초기화라는 첫걸음이다
 - 중요한 것은 Lyra가 단순히 어떻게 돌고 있다가 아닌, 어떻게 우리가 이걸 활용하여 초기화 시킬 수 있는가에 대한 것이다!

- 물론 문서에는 이렇게 구구절절하게 나와있다:

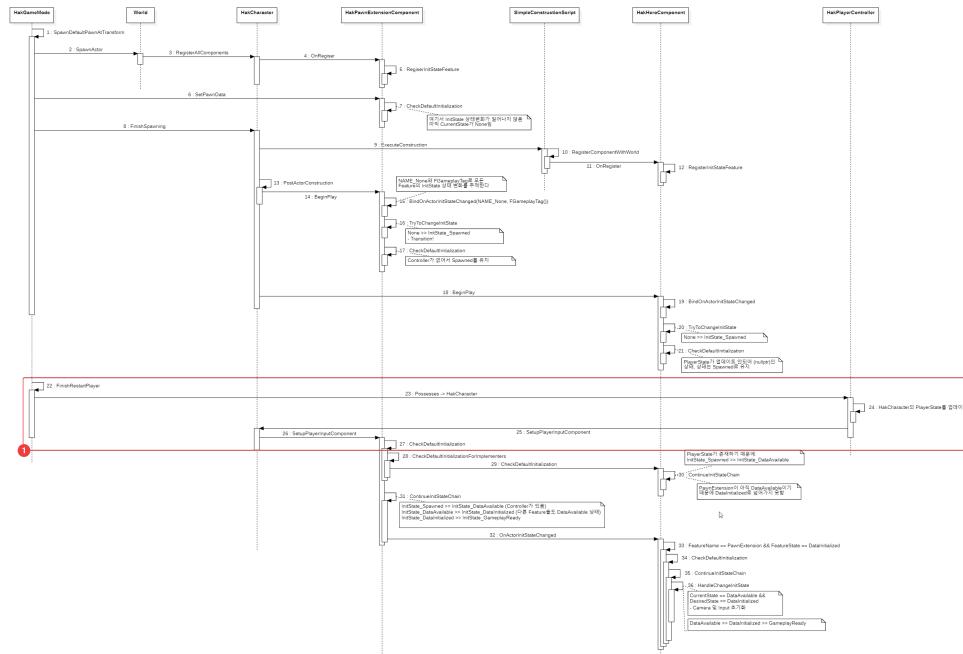
1. 캐릭터가 초기에 클라이언트와 서버에 호출되면 모든 컴포넌트를 어태치하고 등록합니다. 여기에는 두 개의 초기화 스테이트 컴포넌트와 `LyraAbilitySystemComponent` 등의 기타 요소가 포함됩니다.
2. 캐릭터에서 `BeginPlay` 가 호출되면 모든 컴포넌트에서 `BeginPlay` 를 호출하려고 시도합니다. 서버에서는 즉시 호출하려 하지만 클라이언트에서는 리플리케이트된 프로퍼티 전부가 초기 데이터를 보내기 전까지 `BeginPlay` 를 호출하지 않습니다. 호출 시도는 리플리케이트할 데이터가 얼마나 많은지에 따라 각 컴포넌트에서 여러 차례 발생합니다.
3. `BeginPlay` 가 히어로(Hero) 컴포넌트나 라이라 폰(Lyra Pawn) 컴포넌트에서 호출되면, 이 컴포넌트들은 `BindOnActorInitStateChanged` 를 호출하여 초기화 스테이트 변경 사항을 리슨한 다음 `CheckDefaultInitialization` 을 호출하여 4 스테이트 초기화 체인을 따르고자 시도합니다. 이 시점에서 두 컴포넌트 모두 `InitState.Spawned` 에 도달하며 계속해서 초기화를 시도합니다.
4. 히어로 컴포넌트는 `InitState.DataAvailable` 로 트랜지션을 시도할 때 플레이어 스테이트 및 입력 컴포넌트가 준비됐는지 확인합니다. 해당 데이터를 이용할 수 없다면 스테이트 머신은 `CheckDefaultInitialization` 이 호출될 때까지 정지됩니다. 필수 데이터가 이용 가능하다면 `DataAvailable` 로 트랜지션하지만, 아직 `DataInitialized` 로 트랜지션할 수는 없습니다.
5. 폰 익스텐션(Pawn Extension) 컴포넌트는 `CheckDefaultInitialization` 을 호출할 때 가능한 경우 먼저 히어로 컴포넌트 등의 다른 컴포넌트에 초기화 스테이트 머신을 진행하도록 지시합니다. 그런 다음 자신의 스테이트를 `InitState.DataAvailable` 로 진행하려 할 때 `PawnData` 와 컨트롤러가 완전히 사용 가능한지 확인합니다. 폰 익스텐션 컴포넌트는 다양한 `OnRep` 함수로부터 `CheckDefaultInitialization` 을 호출하여 중요한 크로스 액터 레퍼런스가 리플리케이트를 마친 뒤 스테이트 머신을 진행시키려고 시도합니다. 또 한 가지 옵션은 네이티브 틱 함수로부터 초기화 함수를 호출하는 것입니다.
6. 폰 익스텐션 컴포넌트는 히어로 컴포넌트와 같은 나머지 피처가 전부 `DataAvailable` 에 도달하기 전까지 `InitState.DataInitialized` 로 진행하지 않습니다. 트랜지션이 일어나면 히어로 컴포넌트와 기타 리슨 중인 모든 것에 대해 `OnActorInitStateChanged` 함수를 활성화합니다.
7. 그렇게 되면 익스텐션 컴포넌트는 `InitState.DataInitialized` 로 넘어가며, 이에 따라 히어로 컴포넌트 또한 `DataInitialized` 로 넘어갑니다. 이 트랜지션 동안 게임플레이 어빌리티가 생성되어 플레이어 입력에 바인딩됩니다.
8. 그런 다음 히어로 컴포넌트와 폰 익스텐션 컴포넌트는 `InitState.GameplayReady` 로 트랜지션합니다. `InitState.GameplayReady` 는 이 스테이트가 도달하기까지 대기하도록 등록된 `W_Nameplate` 등의 클래스에서 블루프린트 콜백을 활성화합니다.

■ 이것이 무슨 의미가 있겠는가... 😭

아래의 두 가지를 기억해두자:

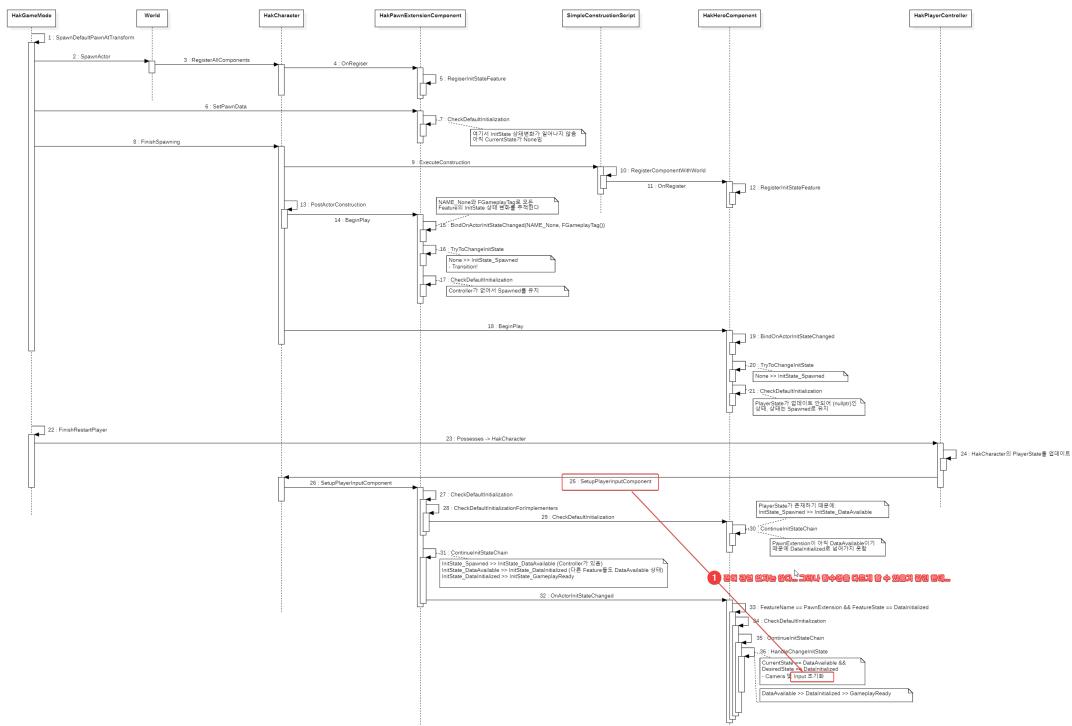
1. InitState를 활용한 초기화에서 우리가 기억해야 할 것은 선형적 구현이다!

- 아래와 같이 연결이 끊어지거나 다음 틱(Tick)의 경우, 연결시켜 줘야 한다:



2. PawnExtension을 활용하거나 이와 비슷한 룰을 담당하는 Component를 통해 초기화의 흐름을 제어해주자.

- 물론 CheckDefaultInitialization, BindOnActorInitStateChanged.
CanChangeInitState와 같은 앞서 활용한 메서드의 용도를 잘 기억해두자!
 - 자꾸 SetupInputComponent의 이름이 걸리긴 하지만...



PlayerCameraManager

▼ 펼치기

- Camera 폴더 생성
- 아래와 같이 .h/.cpp를 생성해주기:

| 이름 | 수정한 날짜 | 유형 | 크기 |
|-------------------------------|--------------------|--------|-----|
| HakCameraComponent.cpp | 2023-10-30 오후 5:19 | CPP 파일 | OKB |
| HakCameraComponent.h | 2023-10-30 오후 5:19 | H 파일 | OKB |
| HakCameraMode.cpp | 2023-10-30 오후 5:19 | CPP 파일 | OKB |
| HakCameraMode.h | 2023-10-30 오후 5:19 | H 파일 | OKB |
| HakCameraMode_ThirdPerson.cpp | 2023-10-30 오후 5:20 | CPP 파일 | OKB |
| HakCameraMode_ThirdPerson.h | 2023-10-30 오후 5:20 | H 파일 | OKB |
| HakPlayerCameraManager.cpp | 2023-10-30 오후 5:18 | CPP 파일 | OKB |
| HakPlayerCameraManager.h | 2023-10-30 오후 5:18 | H 파일 | OKB |

- PlayerCameraManager.h/.cpp 구현:

```

#pragma once

#include "Camera/PlayerCameraManager.h"
#include "HakPlayerCameraManager.generated.h"

/**
 * Controller에 바인딩된 CameraManager
 */
#define HAK_CAMERA_DEFAULT_FOV (80.0f)
#define HAK_CAMERA_DEFAULT_PITCH_MIN (-89.0f)
#define HAK_CAMERA_DEFAULT_PITCH_MAX (89.0f)

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

```

```

#include "HakPlayerCameraManager.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakPlayerCameraManager)

UAHakPlayerCameraManager::UAHakPlayerCameraManager(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
    DefaultFOV = HAK_CAMERA_DEFAULT_FOV;
    ViewPitchMin = HAK_CAMERA_DEFAULT_PITCH_MIN;
    ViewPitchMax = HAK_CAMERA_DEFAULT_PITCH_MAX;
}

```

- PlayerCameraManager를 HakPlayerController에 오버라이드:

```

#include "HakPlayerController.h"
#include "HakGame/Camera/HakPlayerCameraManager.h" ①
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakPlayerController)

UAHakPlayerController::UAHakPlayerController(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
    PlayerCameraManagerClass = AHakPlayerCameraManager::StaticClass(); ②
}

```

- HakCameraComponent.h/.cpp 구현:

```

#pragma once

#include "Camera/CameraComponent.h"
#include "HakCameraComponent.generated.h"

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

```

```

#include "HakCameraComponent.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakCameraComponent)

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

```

- HakCameraComponent를 HakCharacter에 생성:

```

#pragma once

#include "GameFramework/Character.h"
#include "HakCharacter.generated.h"

/** forward declaration */
class UHakPawnExtensionComponent;
class UHakCameraComponent; 1

UCLASS()
class AHakCharacter : public ACharacter
{
    GENERATED_BODY()
public:
    AHakCharacter(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

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

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

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category="Hak|Character")
    TObjectPtr<UHakCameraComponent> CameraComponent; 2
};

```

```

#include "HakCharacter.h"
#include "HakPawnExtensionComponent.h"
#include "HakGame/Camera/HakCameraComponent.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakCharacter)

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));
} 1

void AHakCharacter::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);

    // Pawn이 Possess로서, Controller와 PlayerState 정보 접근이 가능한 상태가 되었음:
    // - SetupPlayerInputComponent 확인
    PawnExtComponent->SetupPlayerInputComponent();
}

```

[CameraMode]

- 우선 CameraComponent의 멤버 변수부터 완성시키자:

```

#pragma once

#include "Camera/CameraComponent.h"
#include "HakCameraComponent.generated.h"

/** forward declaration */
class UHakCameraModeStack;
class UHakCameraMode;
/** template forward declaration */
template <class TClass> class TSubclassOf;

/** (return type, delegate_name) */
DECLARE_DELEGATE_RetVal(TSubclassOf<UHakCameraMode>, FHakCameraModeDelegate);

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

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

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

```

HakCameraMode와 HakCameraModeStack 레이아웃 구현:

```

#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 variables
     */

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

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

```

```

#include "HakCameraMode.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakCameraMode)

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

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

```

□ Camera Blending 기능을 CameraComponent에 추가하자:

```

#pragma once

#include "Camera/CameraComponent.h"
#include "HakCameraComponent.generated.h"

/** forward declaration */
class UHakCameraModeStack;
class UHakCameraMode;
/** template forward declaration */
template <class TClass> class TSubclassOf;

/** (return type, delegate_name) */
DECLARE_DELEGATE_RetVal(TSubclassOf<UHakCameraMode>, FHakCameraModeDelegate);

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

    /**
     * CameraComponent interface
     */
    virtual void OnRegister() final; // 1

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

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

```

```

#include "HakCameraComponent.h"
#include "HakCameraMode.h" // 1
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakCameraComponent)

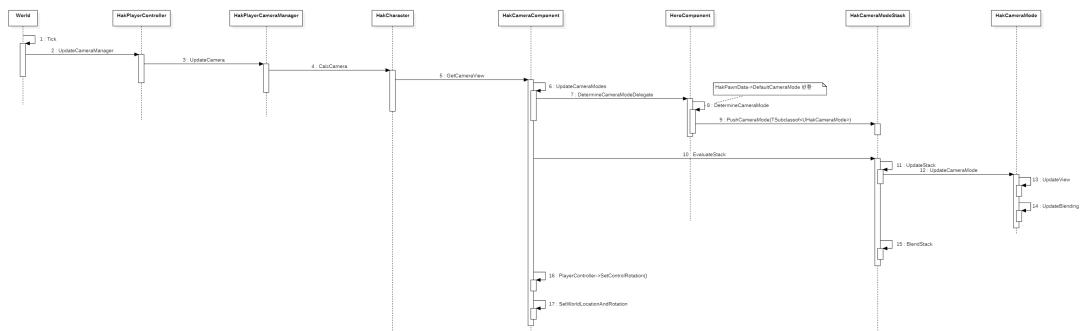
UHakCameraComponent::UHakCameraComponent(const FObjectInitializer& ObjectInitializer)
{
    : Super(ObjectInitializer),
    , CameraModeStack(nullptr)
}

void UHakCameraComponent::OnRegister()
{
    Super::OnRegister();
    if (!CameraModeStack)
    {
        // 초기화 (BeginPlay와 같은)가 딱히 필요없는 객체로 NewObject로 할당
        CameraModeStack = NewObject<UHakCameraModeStack>(this);
    }
}

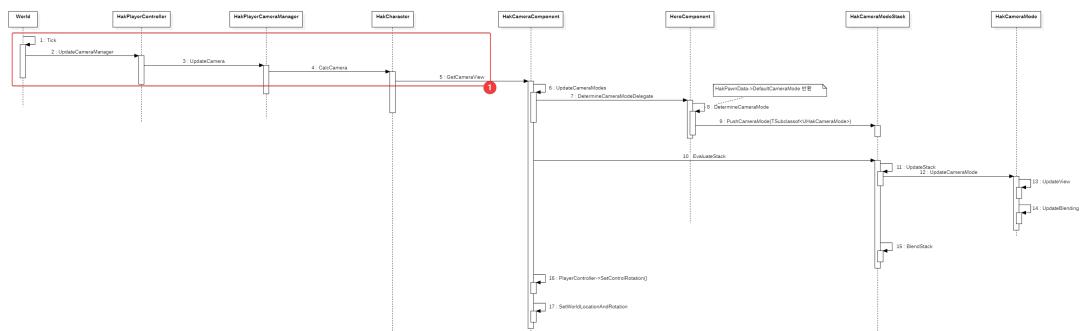
```

[Camera Update 흐름]

- 전체적인 흐름을 아래의 순서도를 통해 보자:



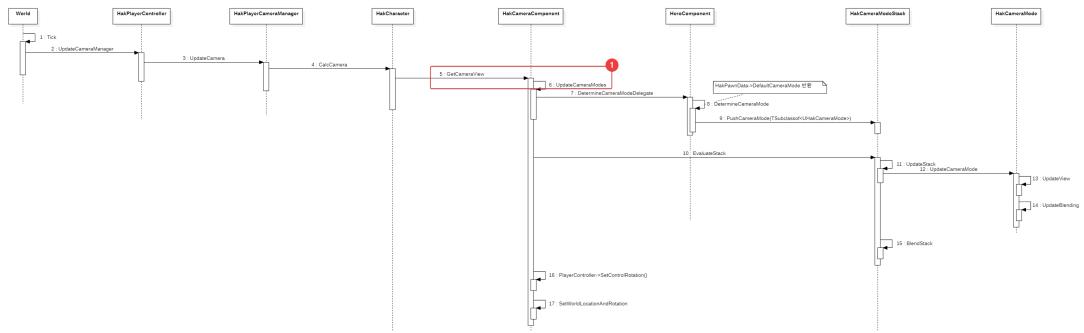
- CameraComponent는 다른 SceneComponent와 달리 월드 내 레벨이 아닌, PlayerController에서 시작해서, CameraManager에서 CameraComponent를 순회하여 계산한다:



- 아래의 코드를 통해 보기:

```
1610 // Update cameras and streaming volumes
1611 {
1612     SCOPE_CYCLE_COUNTER(STAT_UpdateCameraTime);
1613     CSV_SCOPE_TIMING_STAT_EXCLUSIVE(Camera);
1614 
1615     // Update cameras last. This needs to be done before NetUpdates, and after all actors have been ticked.
1616     for( FConstPlayerControllerIterator Iterator = GetPlayerControllerIterator(); Iterator; ++Iterator )
1617     {
1618         if (APlayerController* PlayerController = Iterator->Get())
1619         {
1620             if (!bIsPaused || PlayerController->ShouldPerformFullTickWhenPaused())
1621             {
1622                 PlayerController->UpdateCameraManager(DeltaSeconds); 1
1623             }
1624             else if (PlayerController->PlayerCameraManager && FCameraPhotographyManager::IsSupported(this))
1625             {
1626                 PlayerController->PlayerCameraManager->UpdateCameraPhotographyOnly();
1627             }
1628         }
1629         if( !bIsPaused && IsGameWorld())
1630         {
1631             // Update world's required streaming levels
1632             InternalUpdateStreamingState();
1633         }
1634     }
1635 }
```

- CameraComponent의 매틱(Tick) 상태 업데이트는 GetCameraView를 통해 진행한다:



□ GetCameraView를 오버라이드하자:

```

#pragma once

#include "Camera/CameraComponent.h"
#include "HakCameraComponent.generated.h"

/** forward declaration */
class UHakCameraModeStack;
class UHakCameraMode;
/** template forward declaration */
template <class TClass> class TSubclassOf;

/** (return type, delegate_name) */
DECLARE_DELEGATE_RetVal(TSubclassOf<UHakCameraMode>, FHakCameraModeDelegate);

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

    /**
     * CameraComponent interface
     */
    virtual void OnRegister() final;
    virtual void GetCameraView(float DeltaTime, FMinimalViewInfo& DesiredView) final; 1

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

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

```

```

#include "HakCameraComponent.h"
#include "HakCameraMode.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakCameraComponent)

UHakCameraComponent::UHakCameraComponent(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
    , CameraModeStack(nullptr)
{ }

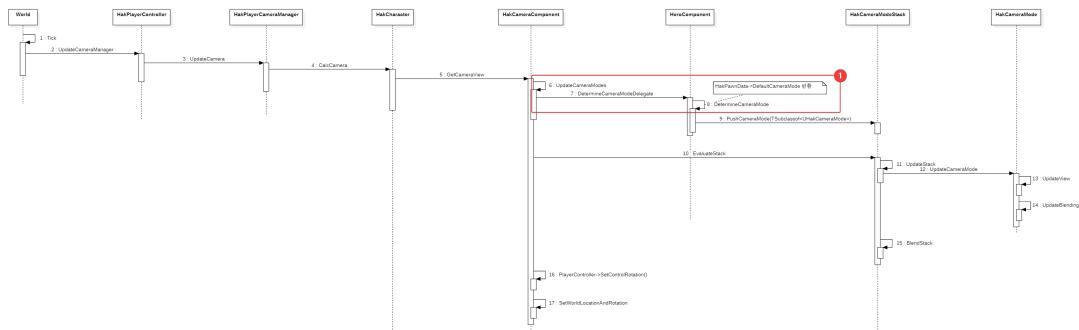
void UHakCameraComponent::OnRegister()
{
    Super::OnRegister();
    if (!CameraModeStack)
    {
        // 초기화 (BeginPlay와 같은)가 딱히 필요없는 객체로 NewObject로 할당
        CameraModeStack = NewObject<UHakCameraModeStack>(this);
    }
}

void UHakCameraComponent::GetCameraView(float DeltaTime, FMinimalViewInfo& DesiredView)
{
}

```

1

□ UpdateCameraModes:



□ UpdateCameraModes를 구현하자:

```

#pragma once

#include "Camera/CameraComponent.h"
#include "HakCameraComponent.generated.h"

/** forward declaration */
class UHakCameraModeStack;
class UHakCameraMode;
/** template forward declaration */
template <class TClass> class TSubclassOf;

/** (return type, delegate_name) */
DECLARE_DELEGATE_RetVal(TSubclassOf<UHakCameraMode>, FHakCameraModeDelegate);

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

    /**
     * member methods
     */
    void UpdateCameraModes(); 1

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

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

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

```

```

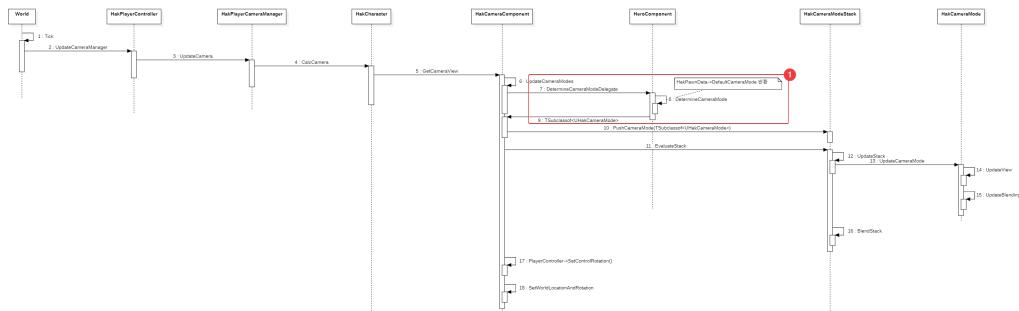
void UHakCameraComponent::UpdateCameraModes()
{
    check(CameraModeStack);

    // DetermineCameraModeDelegate는 CameraMode Class를 반환한다:
    // - 해당 delegate는 HeroComponent의 멤버 함수로 바인딩되어 있다
    if (DetermineCameraModeDelegate.IsBound())
    {
        if (const TSubclassOf<UHakCameraMode> CameraMode = DetermineCameraModeDelegate.Execute())
        {
            // CameraModeStack->PushCameraMode(CameraMode); 2
        }
    }
}

void UHakCameraComponent::GetCameraView(float DeltaTime, FMinimalViewInfo& DesiredView)
{
    check(CameraModeStack);
    UpdateCameraModes(); 1
}

```

- DetermineCameraModeDelegate를 우리는 HeroComponent에서 바인딩 해줘야 한다:



□ 우선 바인딩할 멤버 함수로 DetermineCameraMode를 정의하자:

```
#pragma once
#include "Components/PawnComponent.h"
#include "Components/GameFrameworkInitStateInterface.h"
#include "HakHeroComponent.generated.h"

/** forward declaration */
class UHakCameraMode;
template<class TClass> class TSubclassOf;

/**
 * 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;

    /**
     * IPawnComponent 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; // 1
};


```

```
TSubclassOf<UHakCameraMode> UHakHeroComponent::DetermineCameraMode() const
{
    const APawn* Pawn = GetPawn<APawn>();
    if (!Pawn)
    {
        return nullptr;
    }

    if (UHakPawnExtensionComponent* PawnExtComp = UHakPawnExtensionComponent::FindPawnExtensionComponent(Pawn))
    {
        if (const UHakPawnData* PawnData = PawnExtComp->GetPawnData<UHakPawnData>()) // 1
        {
            return PawnData->DefaultCameraMode; // 2
        }
    }

    return nullptr;
}
```

□ GetPawnData와 DefaultCameraMode를 정의해주자:

```


/***
 * 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* PawnData);
    void SetupPlayerInputComponent();

    /**
     * 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 AActorInitStateChangedParams& Params) final;
    virtual bool CanChangeInInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState) const final;
    virtual void CheckDefaultInitialization() final;

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


```

```


<#include "CoreMinimal.h"
<#include "Engine/DataAsset.h"
<#include "HakGame/Camera/HakCameraMode.h"
<#include "HakPawnData.generated.h" 1

/**
 * 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; 2
};


```

□ HeroComponent에서 DetermineCameraMode를 바인딩해주자:

- 우리가 앞서 HeroComponent에서 DataAvailable → DataInitialized 과정에 정의했던 HandleInitStateChanged()에 넣어주면 된다:

```


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);
            }
        }
    }
}


```

- 간단히 CameraMode_ThirdPerson을 정의해서 PawnData에 넣어주자:

```
#pragma once

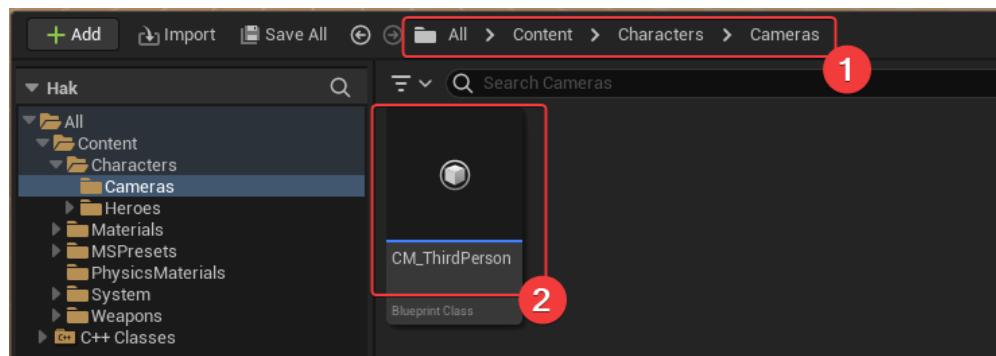
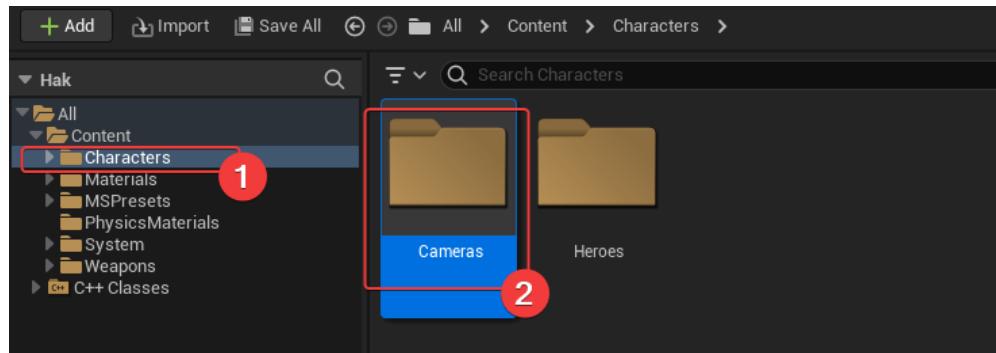
#include "HakCameraMode.h"
#include "HakCameraMode_ThirdPerson.generated.h"

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

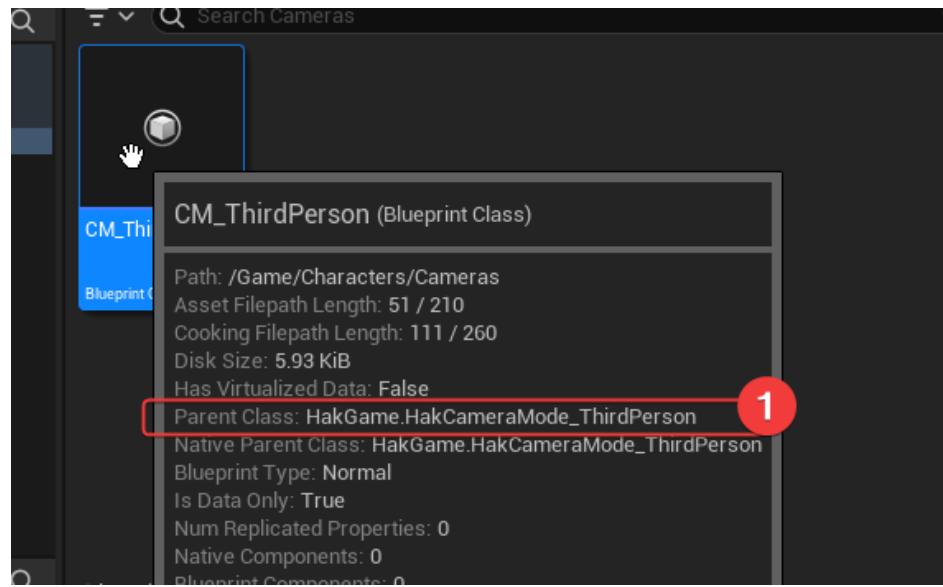
```
#include "HakCameraMode_ThirdPerson.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakCameraMode_ThirdPerson)

UHakCameraMode_ThirdPerson::UHakCameraMode_ThirdPerson(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
```

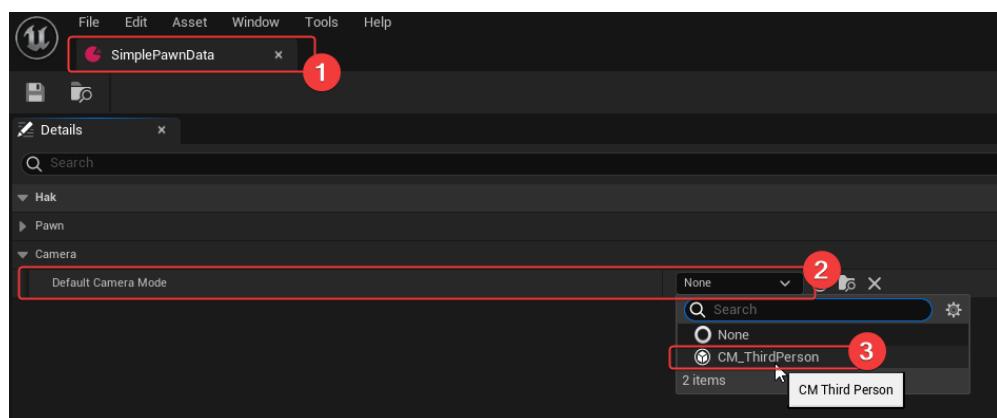
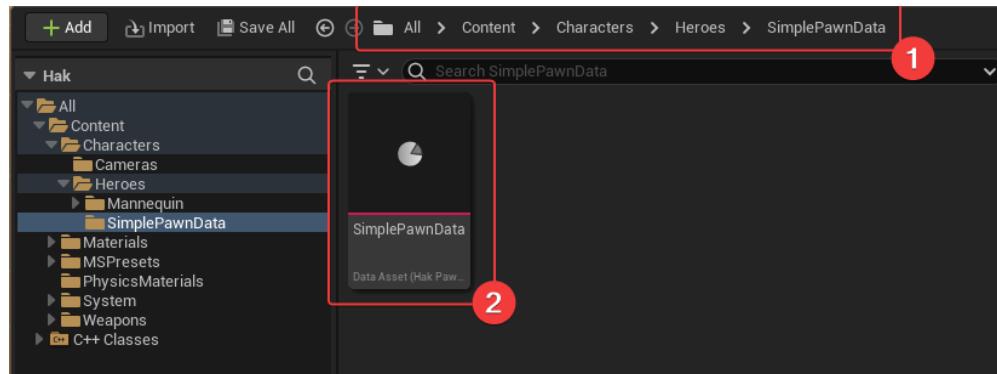
- CM_ThirdPerson Blueprint를 만들어주자:



- HakCameraMode_ThirdPerson을 상속받아 만들어주자:



□ PawnData에 설정해주자:



□ 디버깅하여 DetermineCameraMode가 잘 반환되는지 확인해보자:

1. HandleChangeInitState

```

142     void UHakHeroComponent::HandleChangeInitState(UGameFrameworkComponentManager* Manager, FGameplayTag CurrentState, FGameplayTag DesiredState)
143     {
144         const FHakGameplayTags& InitTags = FHakGameplayTags::Get();
145
146         // CurrentState == InitTags.InitState_DataAvailable && DesiredState == InitTags.InitState_DataInitialized
147         if (CurrentState == InitTags.InitState_DataAvailable && DesiredState == InitTags.InitState_DataInitialized)
148         {
149             APawn* Pawn = GetPawn<APawn>();
150             AHakPlayerstate* HakPS = GetPlayerState<AHakPlayerState>();
151
152             if (!ensure(Pawn && HakPS))
153             {
154                 return;
155             }
156
157             // Input과 Camera에 대한 편집... (TODO)
158
159             const bool bIsLocallyControlled = Pawn->IsLocallyControlled();
160             const UHakPawnData* PawnData = nullptr;
161             if (UHakPawnExtensionComponent* PawnExtComp = FindPawnExtensionComponent(Pawn))
162             {
163                 PawnData = PawnExtComp->GetPawnData<UHakPawnData>();
164             }
165
166             if (bIsLocallyControlled && PawnData)
167             {
168                 // 현재 HakCharacter에 Attach된 CameraComponent 찾음
169                 if (UHakCameraComponent* CameraComponent = UHakCameraComponent::FindCameraComponent(Pawn))
170                 {
171                     CameraComponent->DetermineCameraModeDelegate.BindUOB(this, &ThisClass::DetermineCameraMode);
172                 }
173             }
174         }
175     }

```

2. DetermineCameraMode

```

21 #PRAGMA_DISABLE_OPTIMIZATION
22 TSubclassOf<UHakCameraMode> UHakHeroComponent::DetermineCameraMode() const
23 {
24     const APawn* Pawn = GetPawn<APawn>(); 1
25     if (!Pawn)
26     {
27         return nullptr;
28     }
29
30     if (UHakPawnExtensionComponent* PawnExtComp = UHakPawnExtensionComponent::FindPawnExtensionComponent(Pawn))
31     {
32         if (const UHakPawnData* PawnData = PawnExtComp->GetPawnData<UHakPawnData>())
33         {
34             return PawnData->DefaultCameraMode; 2
35         }
36     }
37
38     return nullptr;
39 }
#PRAGMA_ENABLE_OPTIMIZATION

```