



16주차 (2024.02.19)

시나리오:

▼ 펼치기

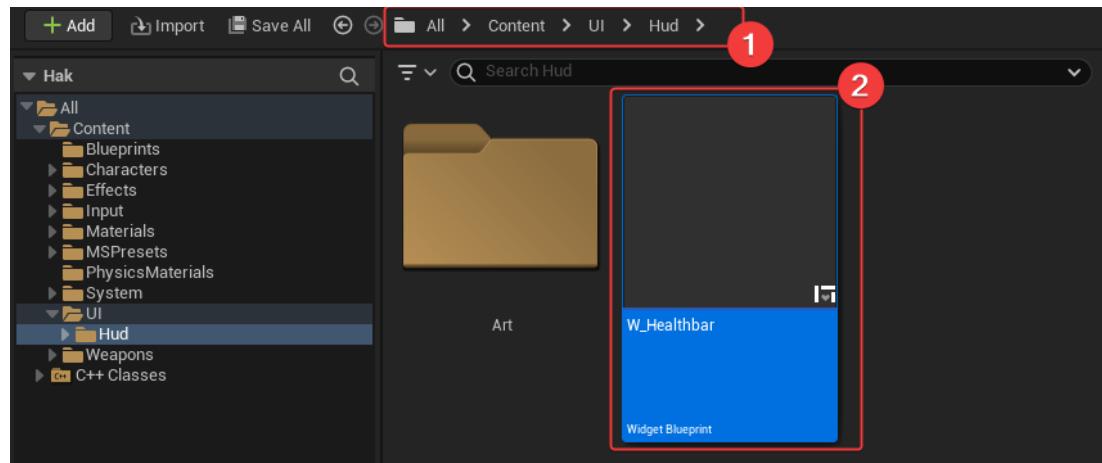
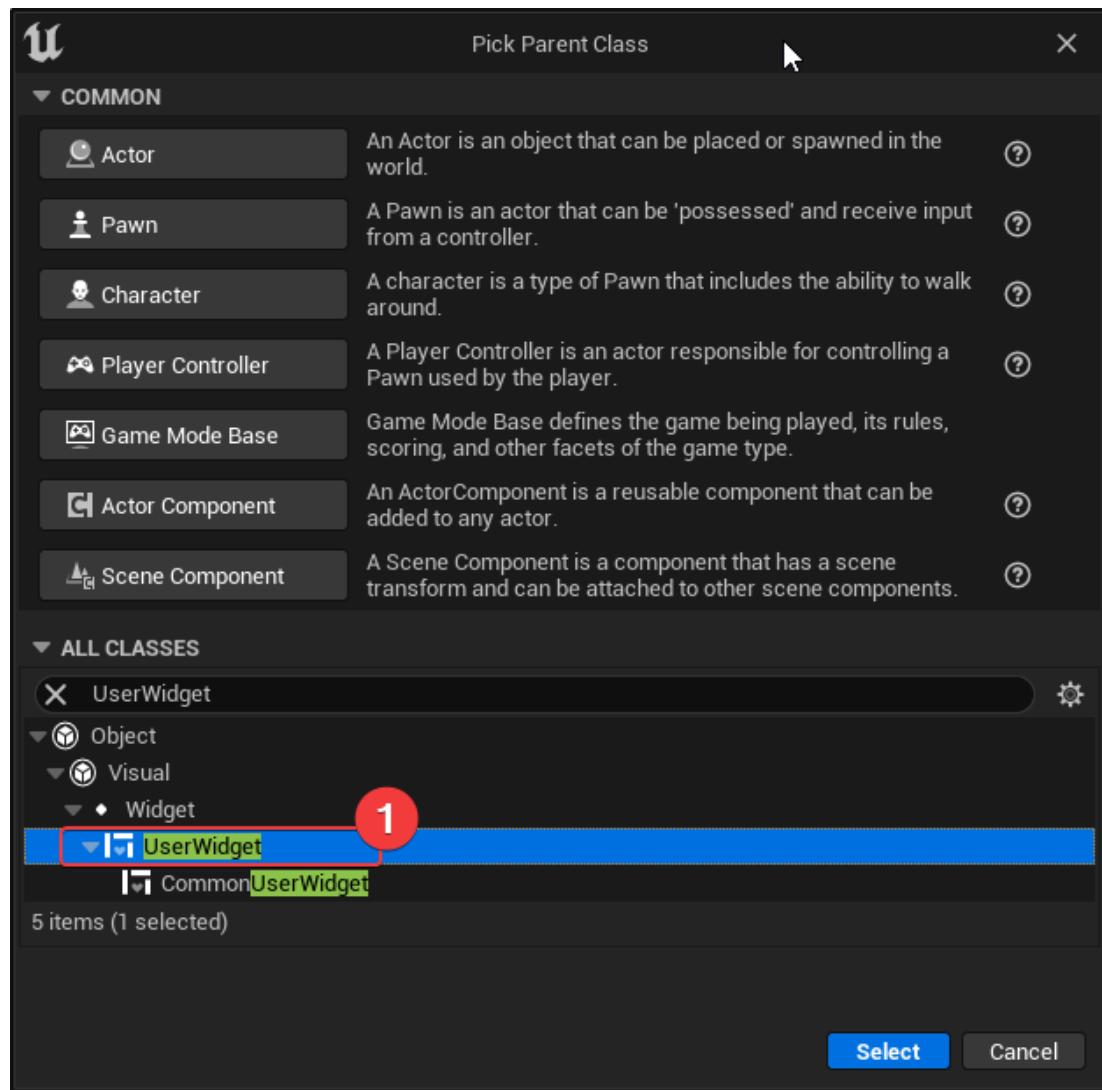
- 아래는 우리가 구현할 부분을 설명하는 영상이다:

https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/e38171a3-b8a6-4bc8-a603-46c3972657fb/UnrealEditor_C3rwp3HnAV.mp4

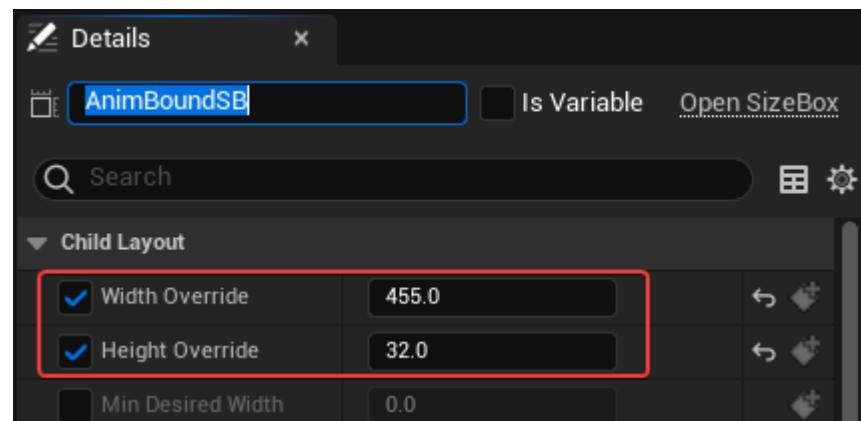
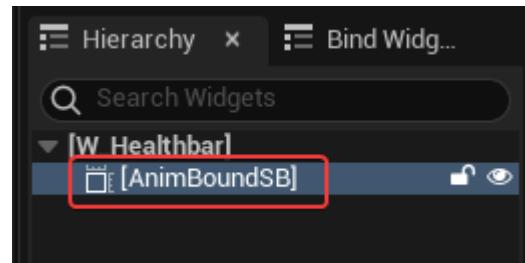
W_Healthbar:

▼ 펼치기

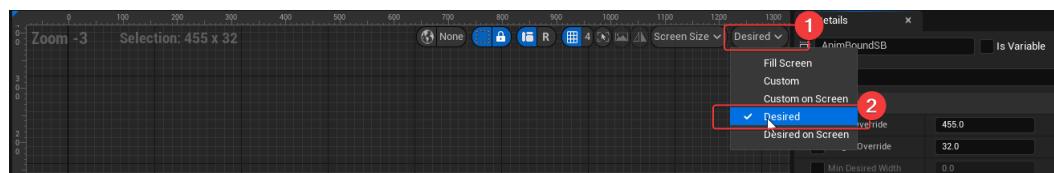
- W_ShooterHUDLayout에 넣을 HealthBar Widget를 만들어주자
- 아래의 경로에 W_HealthBar를 생성하자:



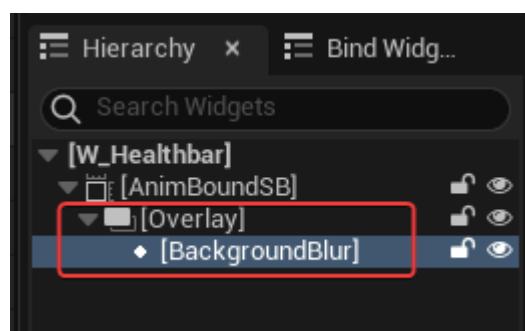
SizeBox의 AnimBoundSB를 아래와 같이 추가하고 속성값을 오버라이드하자:



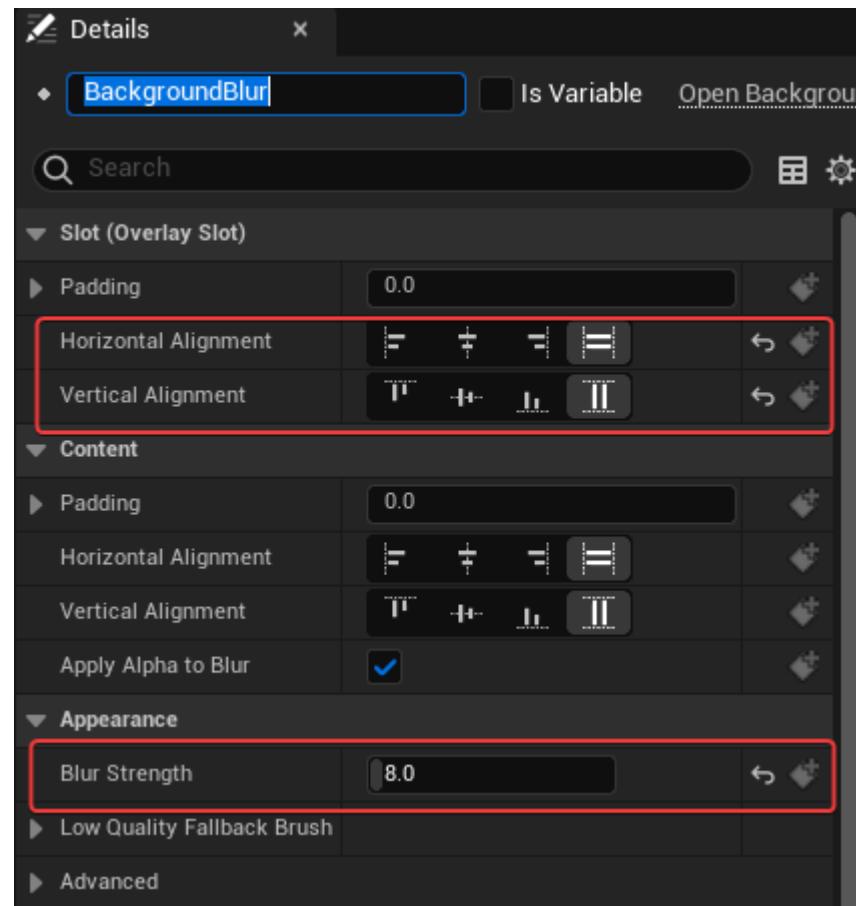
- 위와 같이 SizeBox의 속성값을 오버라이드 했을 경우, 아래와 같이 수정해야 정확한 사이즈를 알 수 있다:



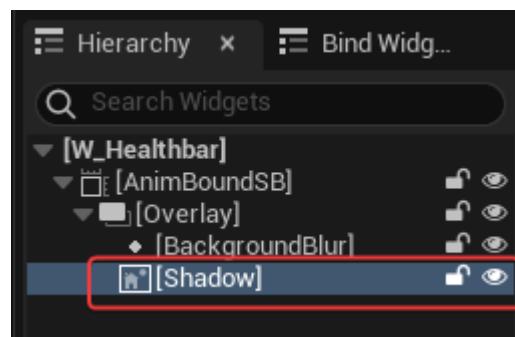
- Overlay를 추가하고, BackgroundBlur를 추가하자:



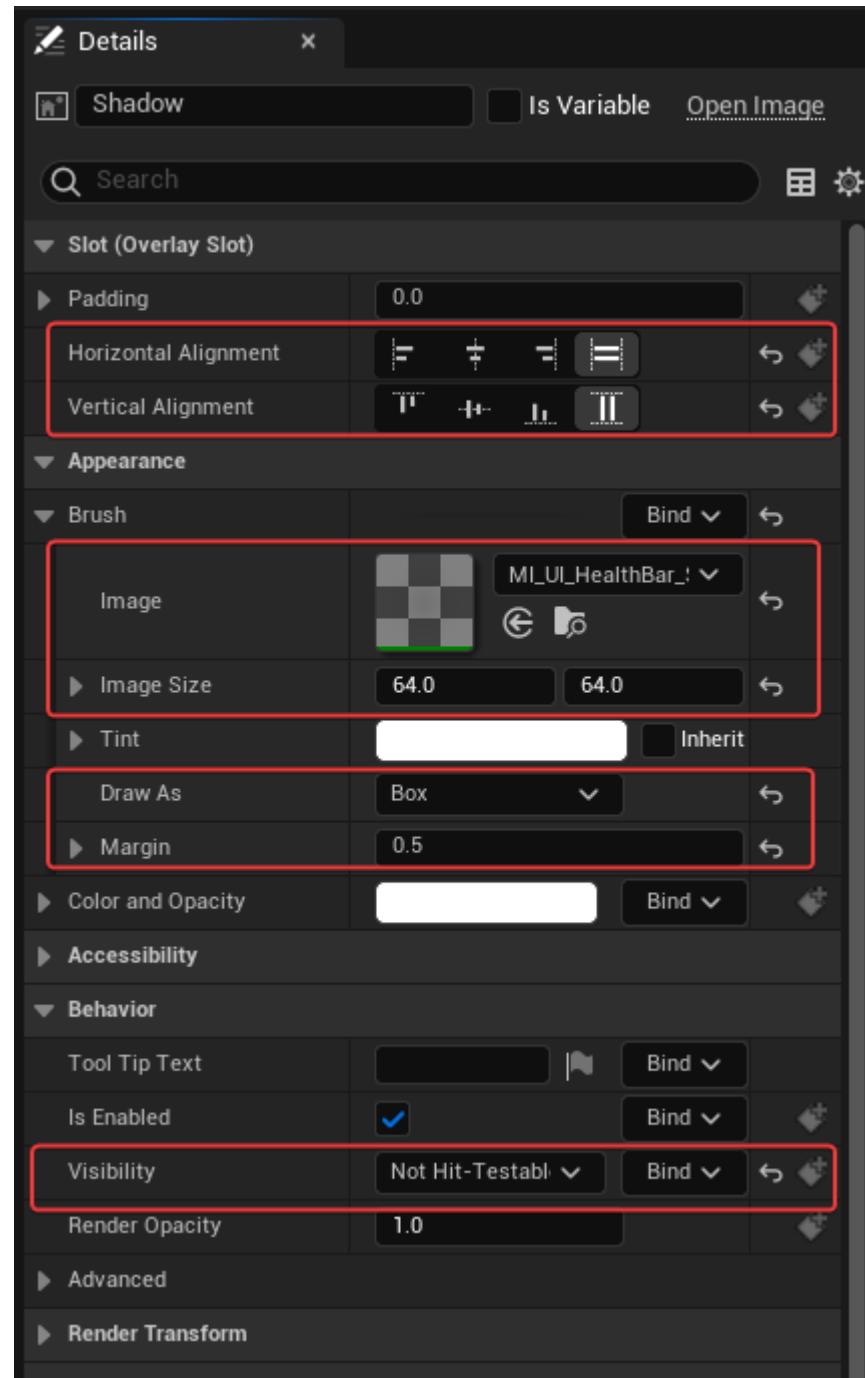
- BackgroundBlur 위젯의 속성값을 업데이트하자:



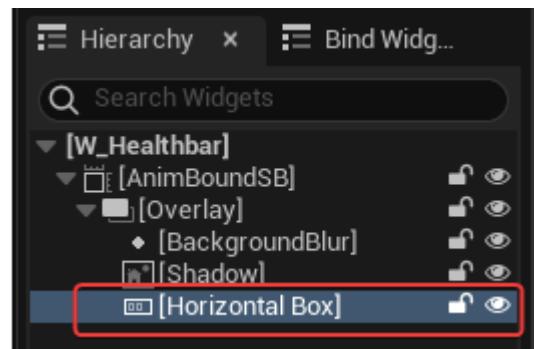
- Image를 추가하고, Shadow라고 명명하자:

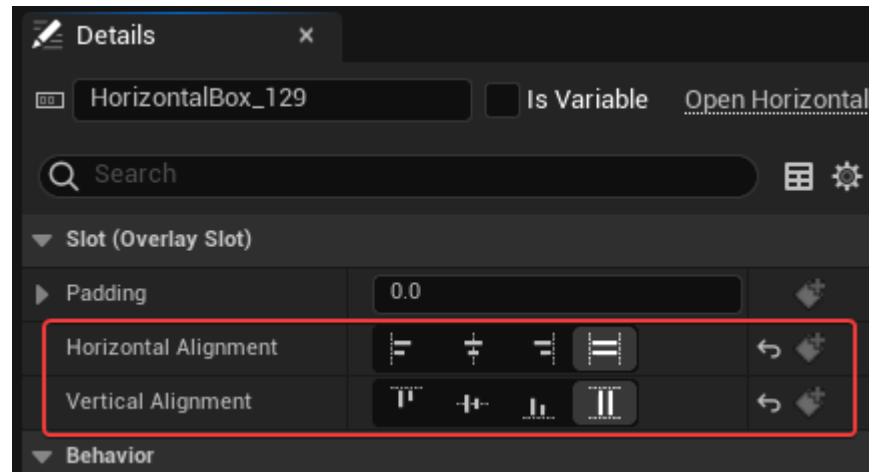


- MI_UI_HealthBar_Shadow를 Migrate하자
- 아래와 같이 속성값을 업데이트하자:

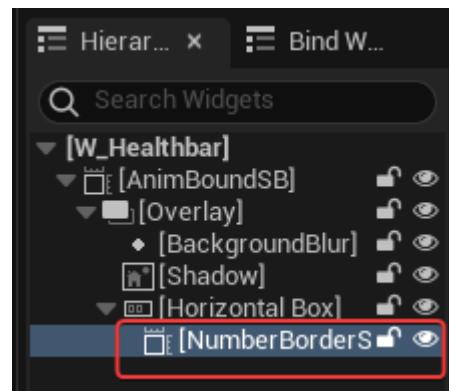


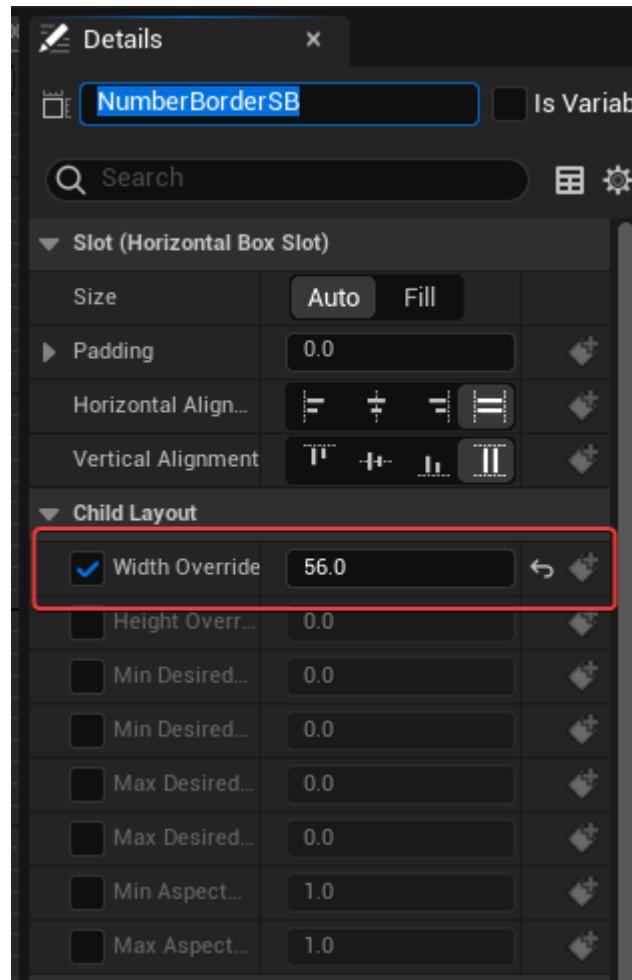
HorizontalBox 추가:



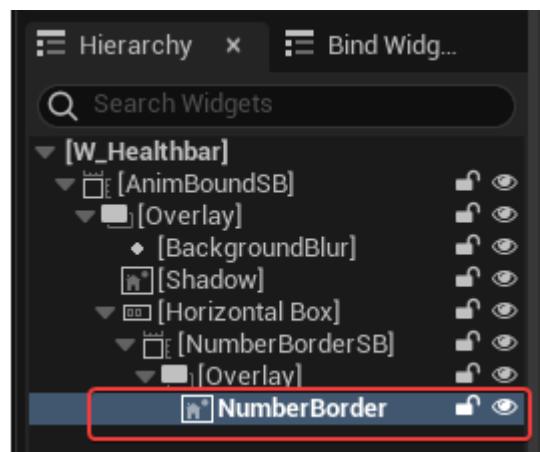


SSizeBox인 NumberBorderSB 추가:



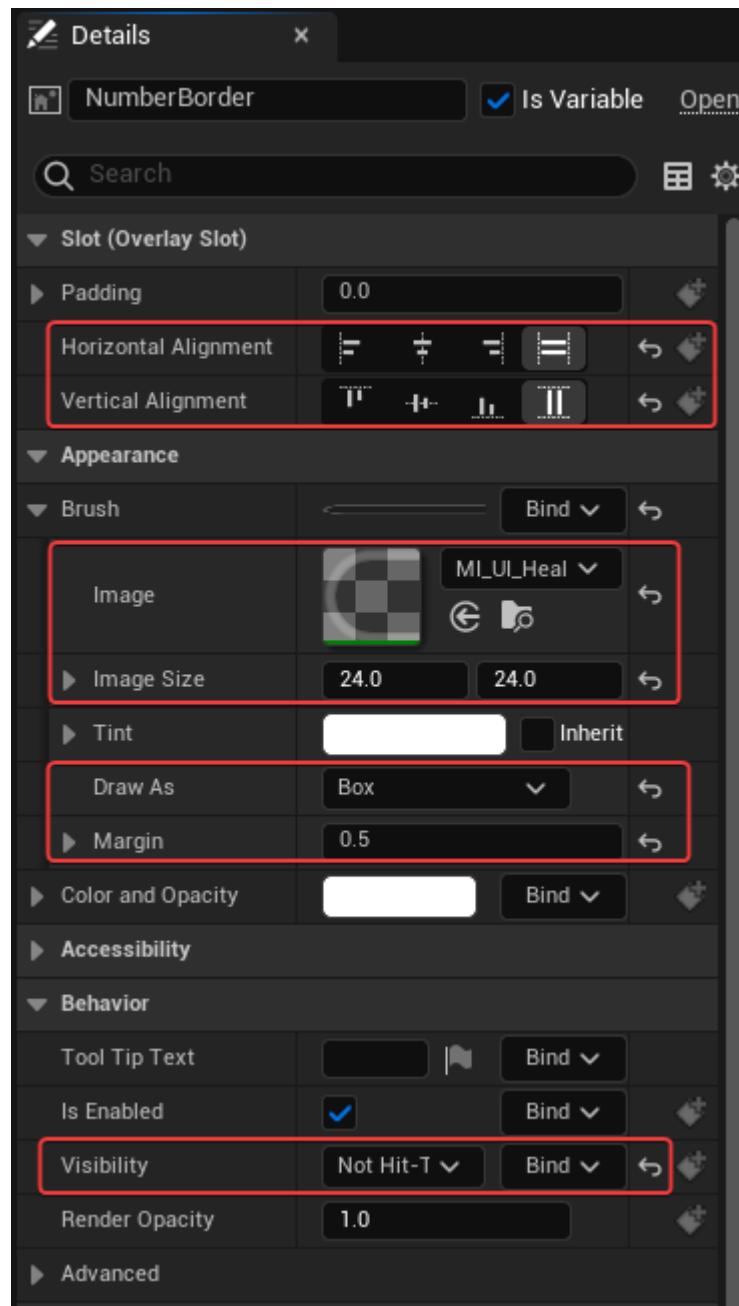


- Overlay 추가하고 NumberBorder인 Image를 추가하자:

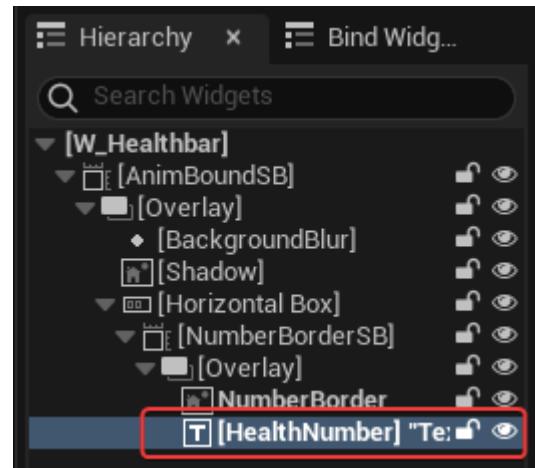


- MI_UI_Healthbar_NumberBorder를 Migrate

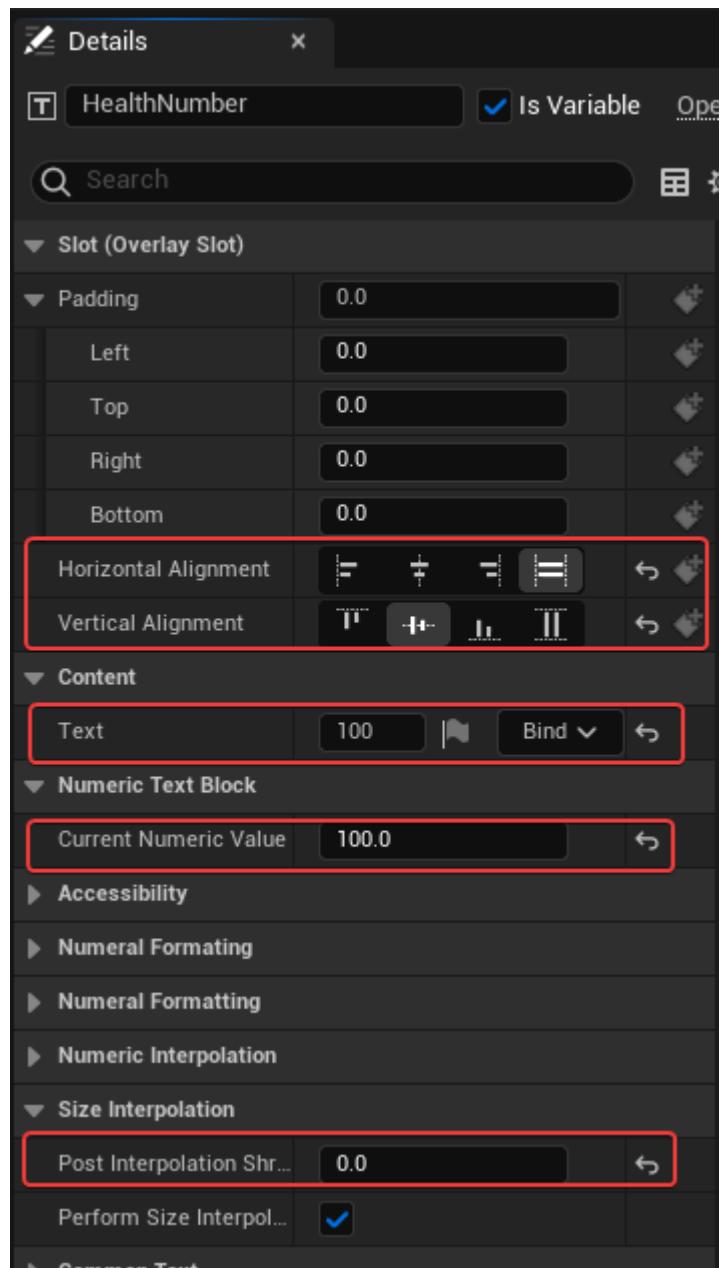
- 아래와 같이 속성값 오버라이드:



HealthNumber인 Common_Numeric_Text을 추가:

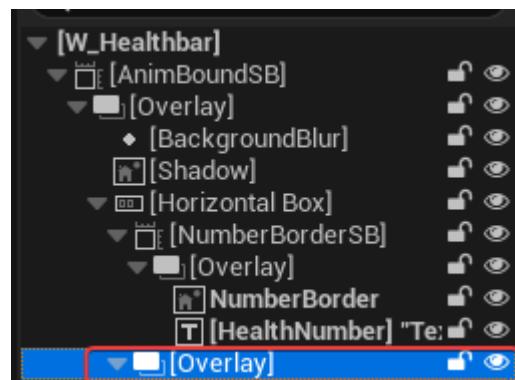


- Orbitron 폰트 추가
- 속성값 오버라이드:

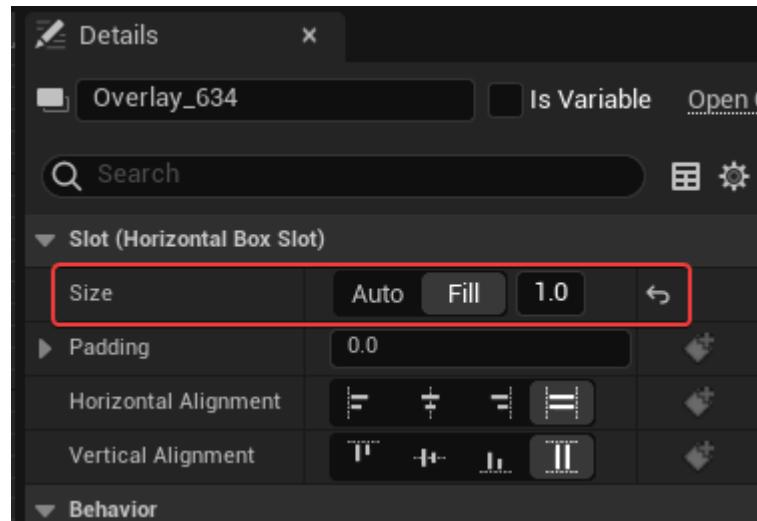




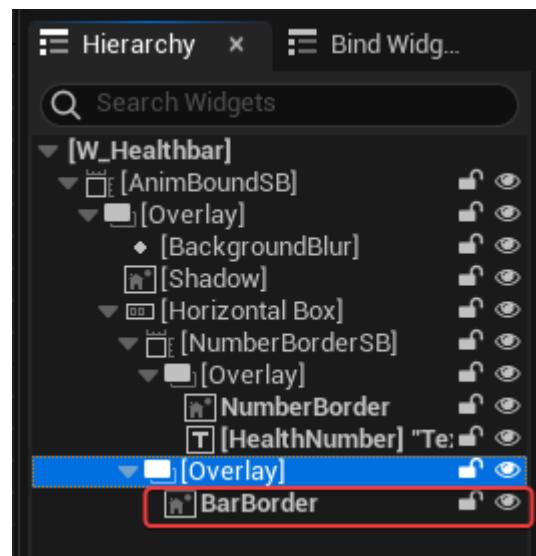
□ Overlay 추가:

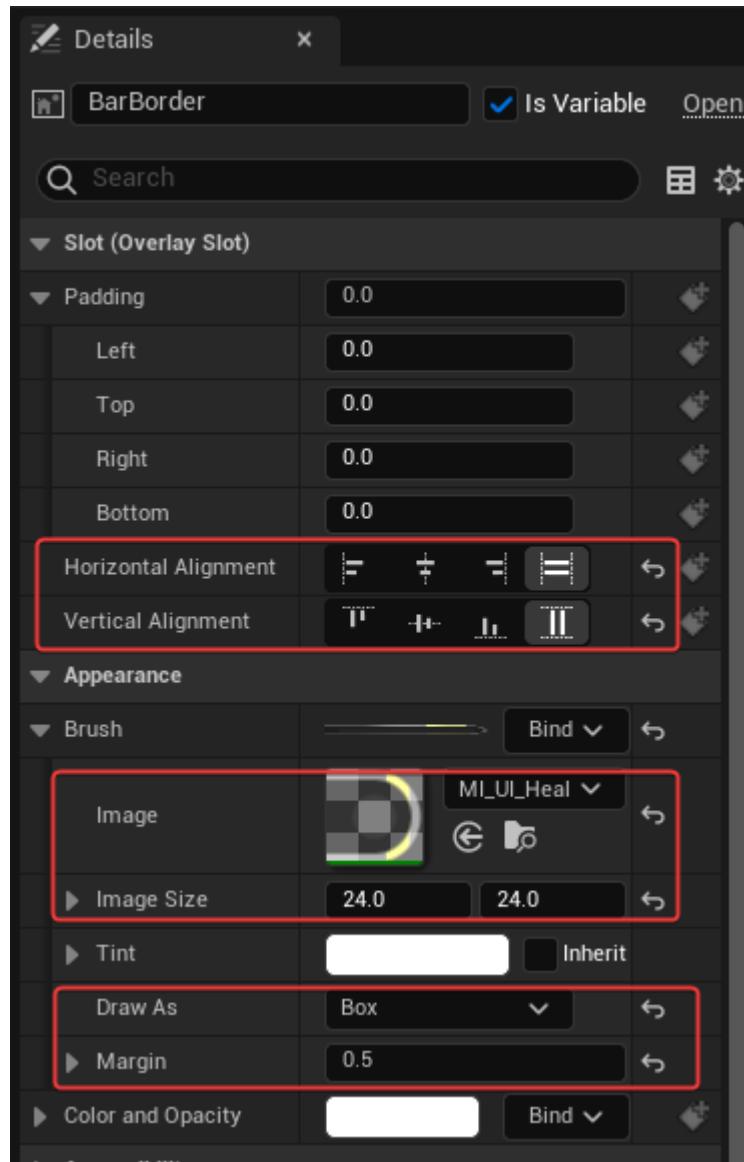


해당 Overlay Widget은 HorizontalBox의 나머지를 채울 것이기 때문에 Fill 설정 하자:



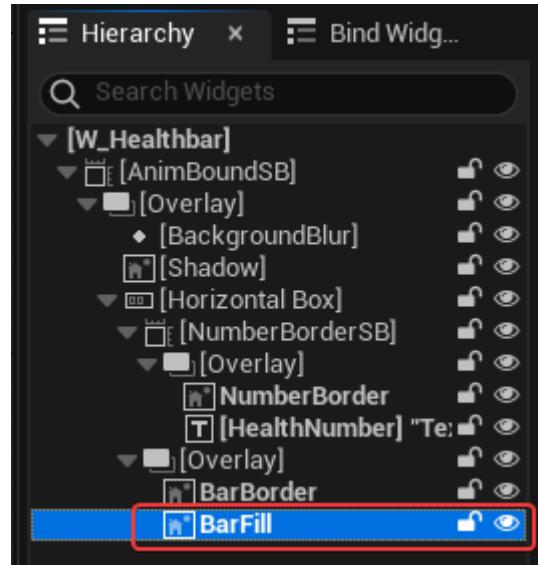
□ Image인 BarBorder를 추가하자:





- MI_UI_HealthBar_BarBorder가 이미 Mirgrate되어 있다

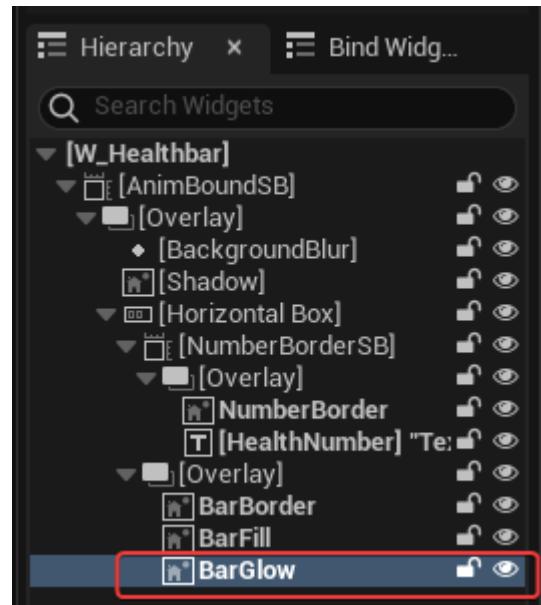
Image인 BarFill을 추가하자:



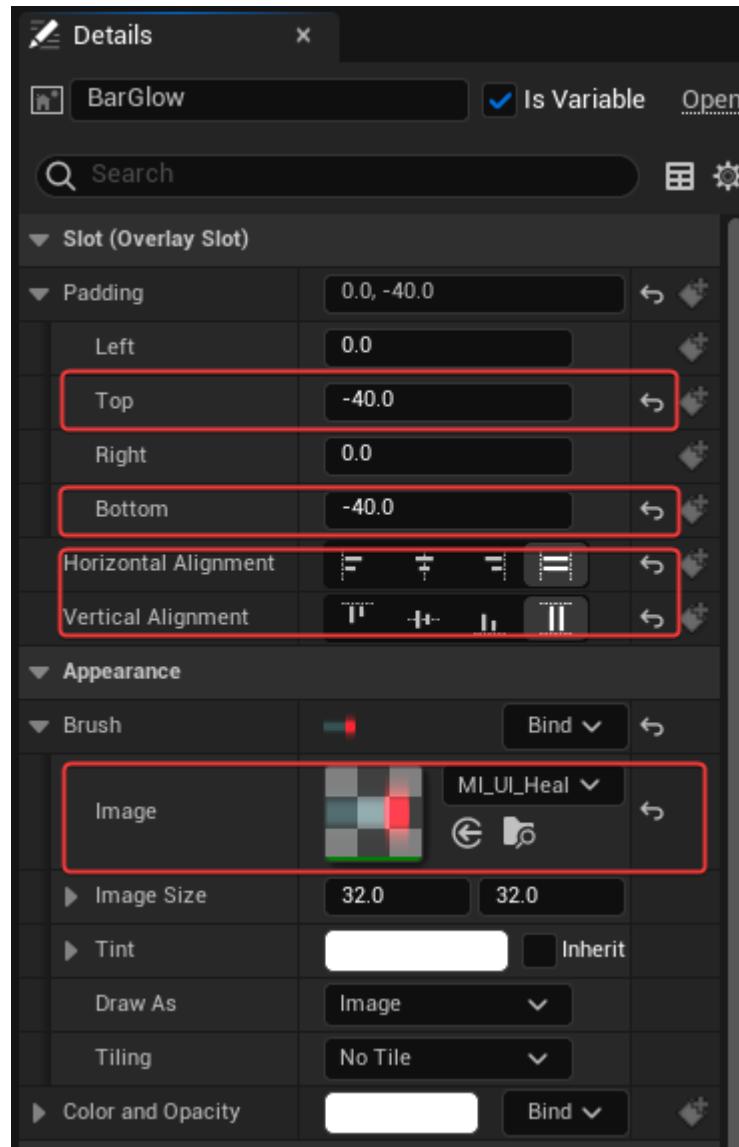


- MI_UI_HealthBar_Fill가 역시 이미 Migrate되어 있다

Image인 BarGlow를 추가하자:

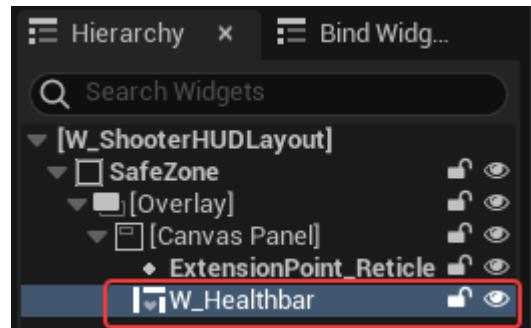


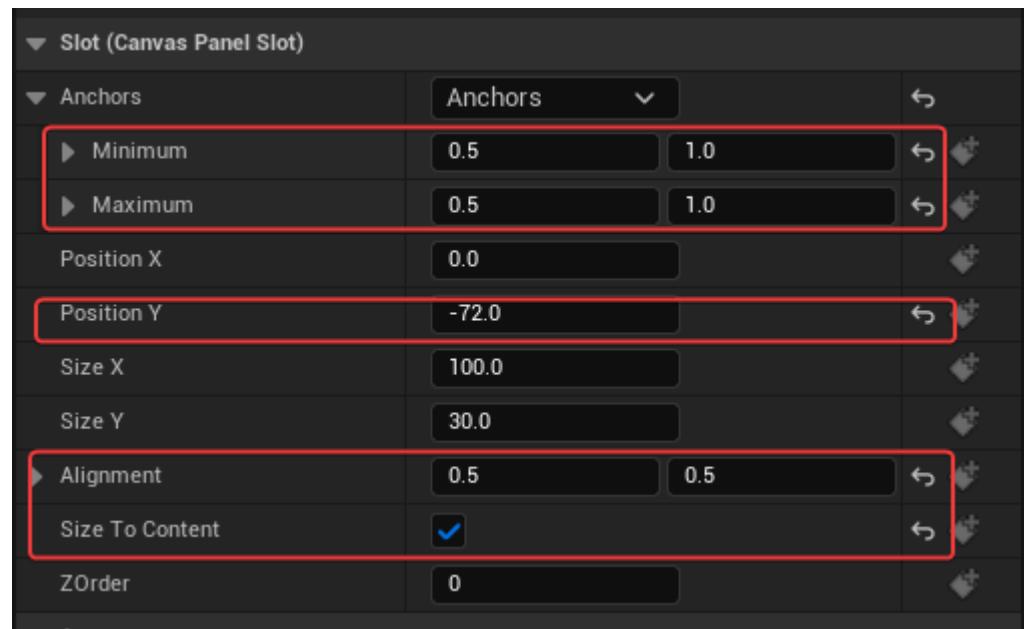
- MI_UI_HealthBar_Glow를 Migrate하자:



참고로, Padding을 음수값으로 주면 영역이 확대된다

- W_HealthBar를 W_ShooterHUDLayout에 추가하자:





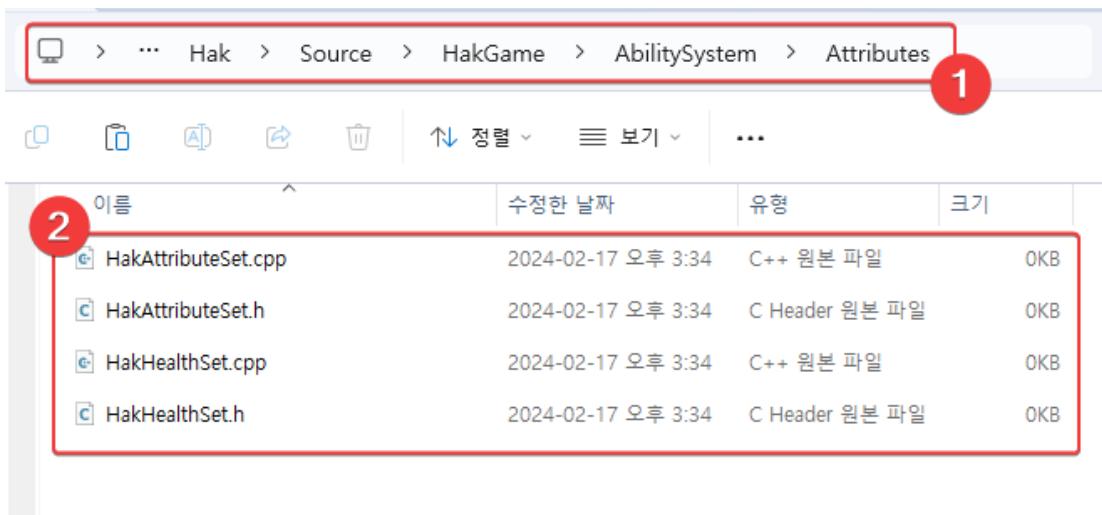
- W_HealthBar의 UI Layout은 완성되었다:



HealthSet:

▼ 펼치기

- Health 정보는 AttributeSet에 저장되어 관리된다:
-
- HakAttributeSet.h/.cpp와 HakHealthSet.h/.cpp를 추가해주자:



HakAttributeSet.h/.cpp를 정의하자:

```

#pragma once

#include "AttributeSet.h"
#include "HakAttributeSet.generated.h"

/**
 * 아래 매크로는 AttributeSet에 Attribute를 추가할 때, 선언 및 정의해야 할 메서드에 대한 간략버전을 제공한다:
 *
 * ATTRIBUTE_ACCESSORS(UHakHealthSet, Health):
 * 이는 아래의 메서드를 선언 및 정의해준다
 *
 * static FGameplayAttribute GetHealthAttribute() {...}
 * float GetHealth() const {...}
 * void SetHealth(float NewVal) {...}
 * void InitHealth(float NewVal) {...}
 */

#define ATTRIBUTE_ACCESSORS(className,PropertyName) \
    GAMEPLAYATTRIBUTE_PROPERTY_GETTER(className,PropertyName) \
    GAMEPLAYATTRIBUTE_VALUE_GETTER(PropertyName) \
    GAMEPLAYATTRIBUTE_VALUE_SETTER(PropertyName) \
    GAMEPLAYATTRIBUTE_VALUE_INITTER(PropertyName)

/**
 * HakAttributeSet
 * - Lyra와 마찬가지로 Hak에서 메인 Attribute Set Class이다
 */
UCLASS()
class UHakAttributeSet : public UAttributeSet
{
    GENERATED_BODY()
public:
    UHakAttributeSet();
}

```

```

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

UHakAttributeSet::UHakAttributeSet()
{
    Super();
}

```

HakHealthSet.h/.cpp를 정의하자:

```

#pragma once

#include "HakAttributeSet.h"
#include "HakHealthSet.generated.h"

/***
 * 아래의 HealthSet은 의미 그대로, 체력에 대한 속성값을 관리한다
 */
UCLASS(BlueprintType)
class UHakHealthSet : public UHakAttributeSet
{
    GENERATED_BODY()
public:
    UHakHealthSet();
}

```

```

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

UHakHealthSet::UHakHealthSet()
: Super()
{ }

```

- HakHealthSet의 Health, MaxHealth, Healing을 정의하자:

```

#pragma once

#include "HakAttributeSet.h"
#include "AbilitySystemComponent.h"
#include "HakHealthSet.generated.h"

/***
 * 아래의 HealthSet은 의미 그대로, 체력에 대한 속성값을 관리한다
 */
UCLASS(BlueprintType)
class UHakHealthSet : public UHakAttributeSet
{
    GENERATED_BODY()
public:
    UHakHealthSet();

    /**
     * 앞서 HakAttributeSet에서 정의했던 ATTRIBUTE_ACCESSORS를 통해, 아래 정의한 멤버변수와 똑같이 이름을 설정한다
     * - ATTRIBUTE_ACCESSORS의 Macro의 정의부분을 한 번 살펴보자
     */
    ATTRIBUTE_ACCESSORS(UHakHealthSet, Health);
    ATTRIBUTE_ACCESSORS(UHakHealthSet, MaxHealth);
    ATTRIBUTE_ACCESSORS(UHakHealthSet, Healing);

    /** 현재 체력 */
    UPROPERTY(BlueprintReadOnly, Category="Hak|Health")
    FGameplayAttributeData Health;

    /** 체력 최대치 */
    UPROPERTY(BlueprintReadOnly, Category="Hak|Health")
    FGameplayAttributeData MaxHealth;

    /** 체력 회복치 */
    UPROPERTY(BlueprintReadOnly, Category="Hak|Health")
    FGameplayAttributeData Healing;
}

```

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

UHakHealthSet::UHakHealthSet()
: Super()
// 우리는 초기값으로 50.0f의 체력으로 맞추어 놓는다
, Health(50.0f)
// 최대 체력으로 100으로 설정:
// - 참고로 버프나, 장비로 인해 최대 체력은 변경 가능하다
, MaxHealth(100.0f)
```

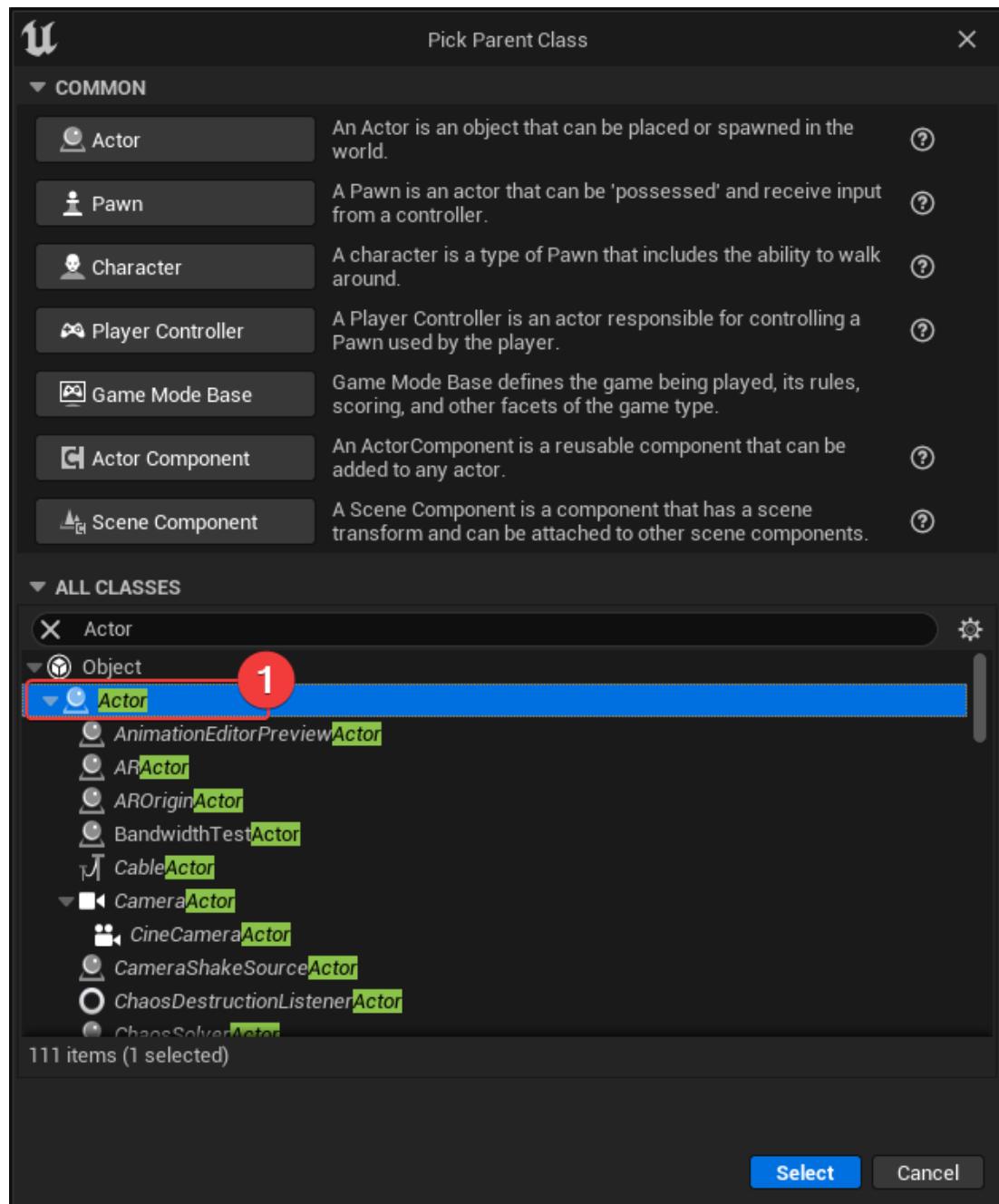
- ATTRIBUTE_ACCESSORS의 Macro들을 한번 살펴보도록 하자.

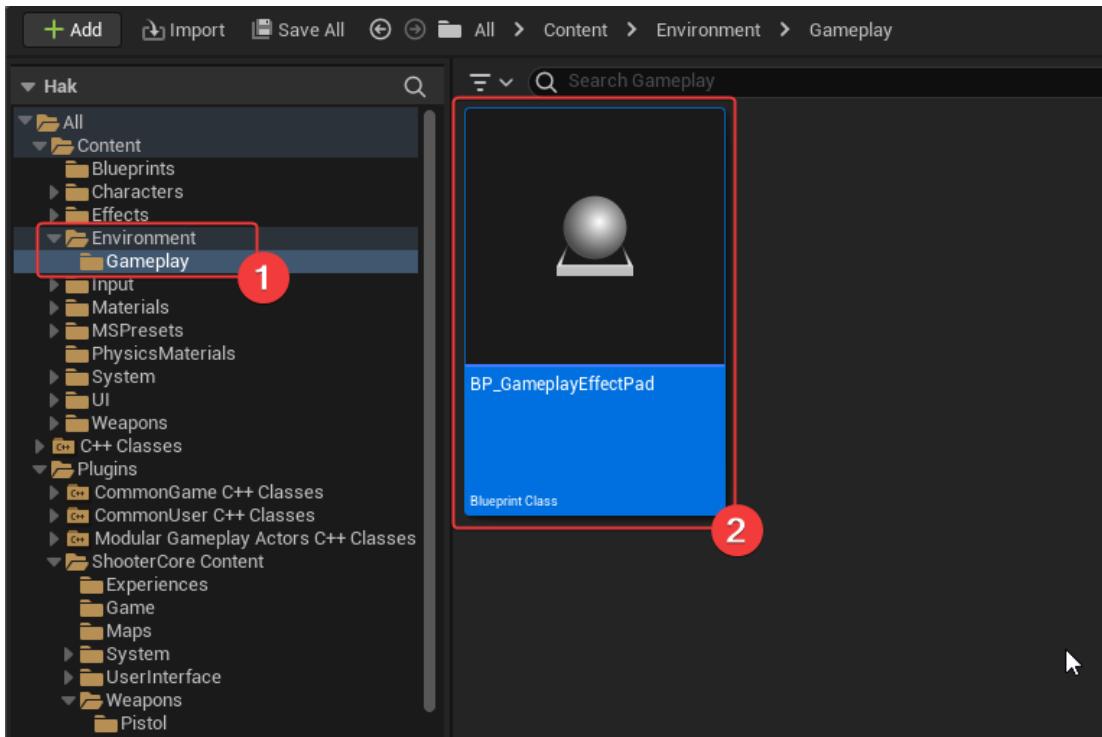
BP_GameplayEffectPad:

▼ 펼치기

- 앞서 본 시나리오에서 우리는 EffectPad를 통해 기본 체력인 50에서 100으로 올릴 것이다.

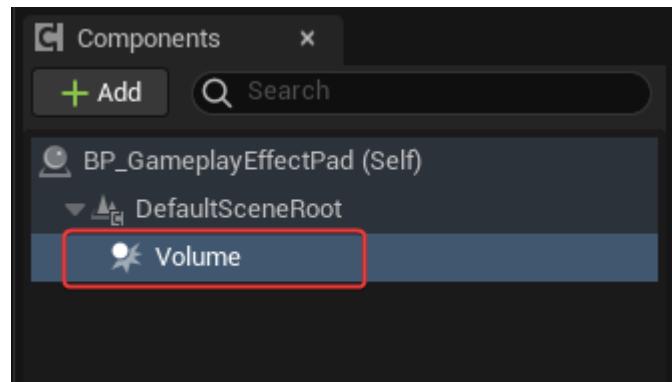
- BP_GameplayEffectPad 생성:



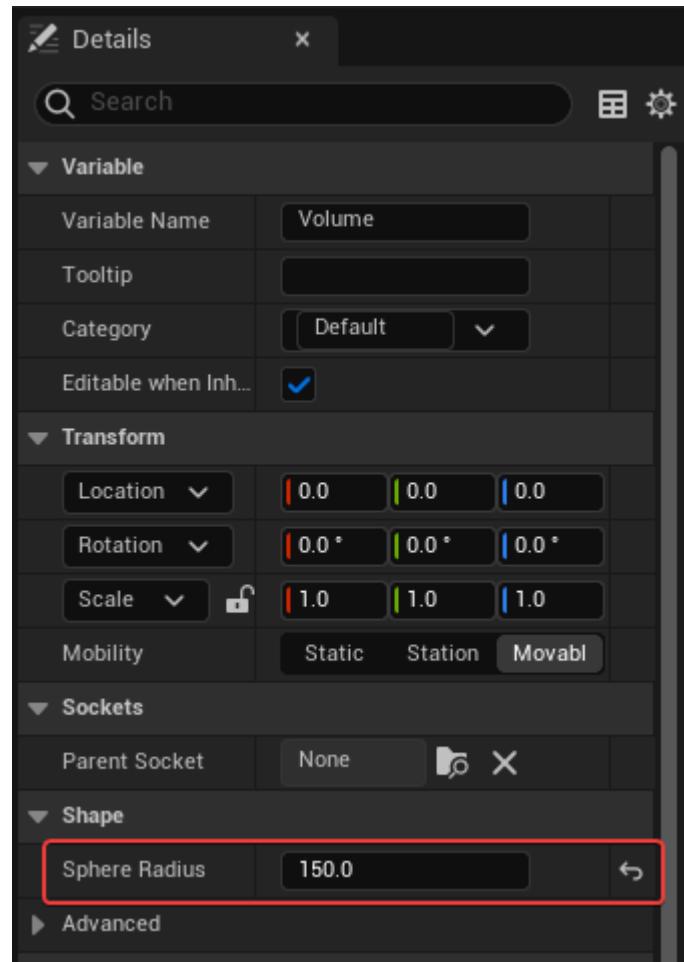


Sphere Collision 추가:

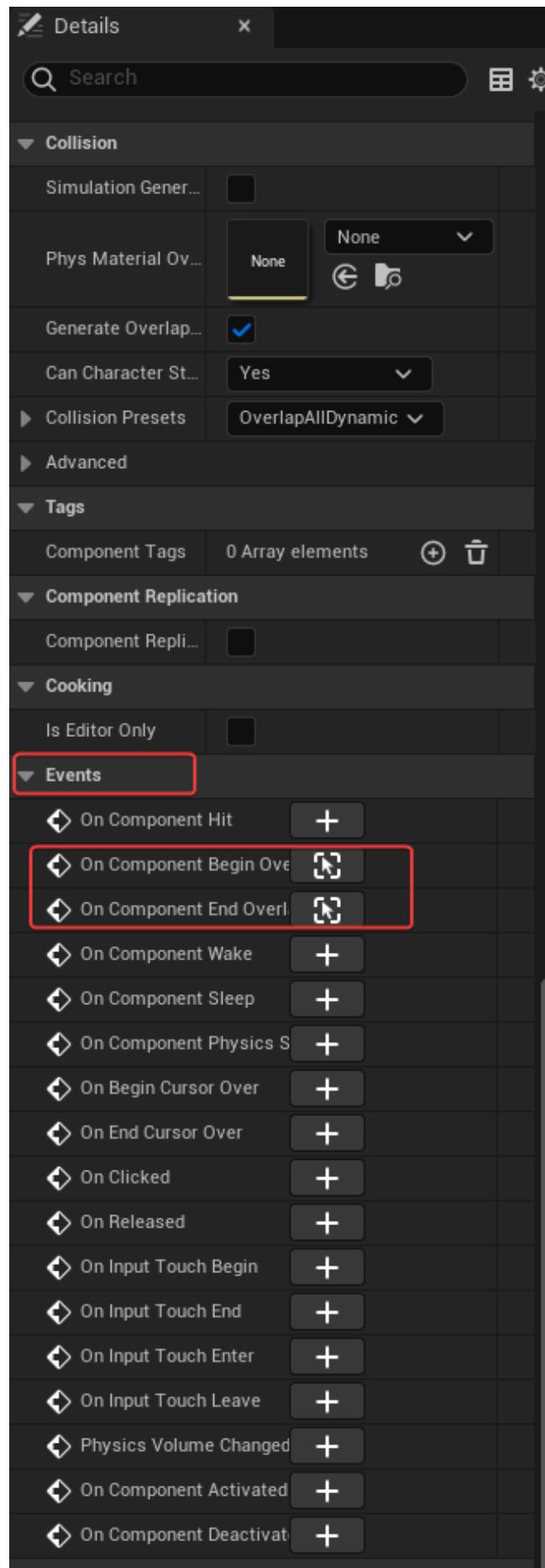
Volume으로 이름 명명:



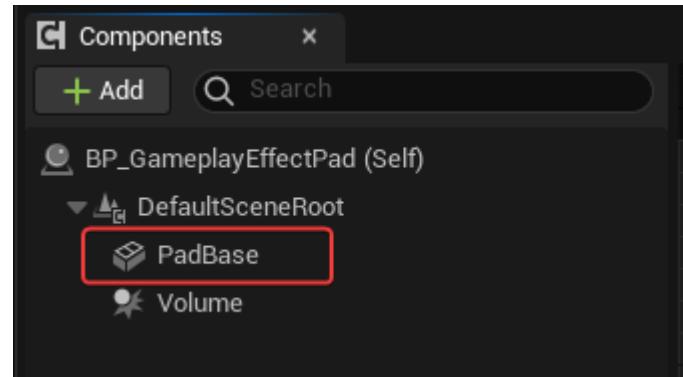
Sphere Collision Radius 수정:



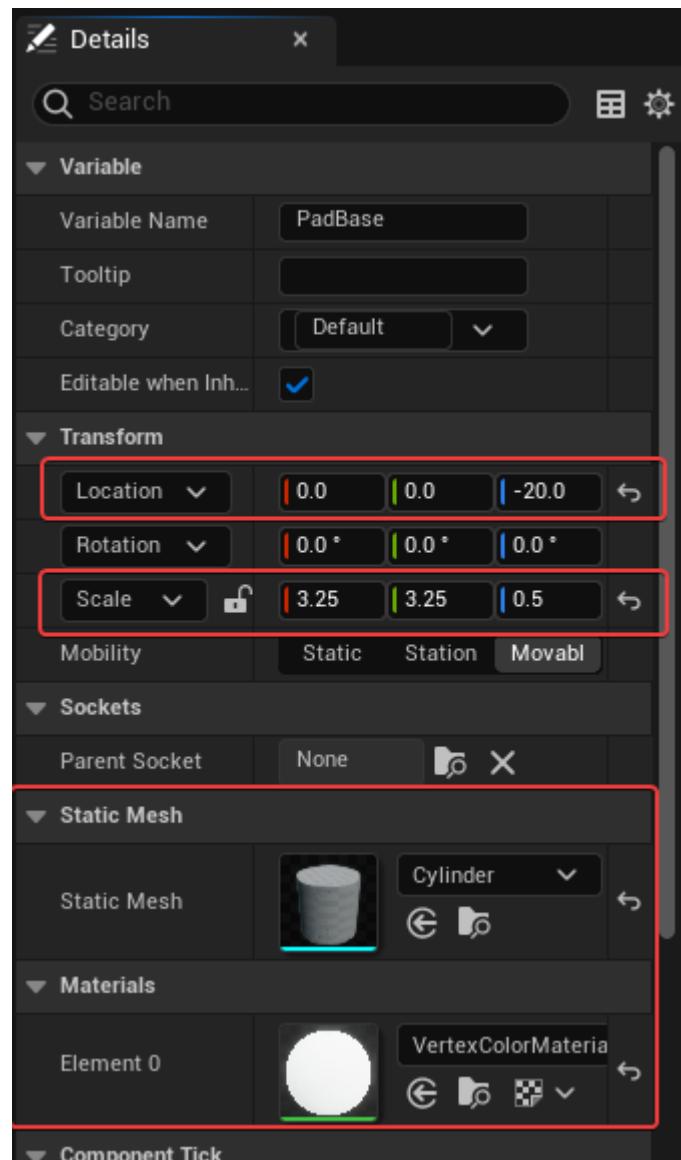
BeginOverlap/EndOverlap 이벤트 추가:



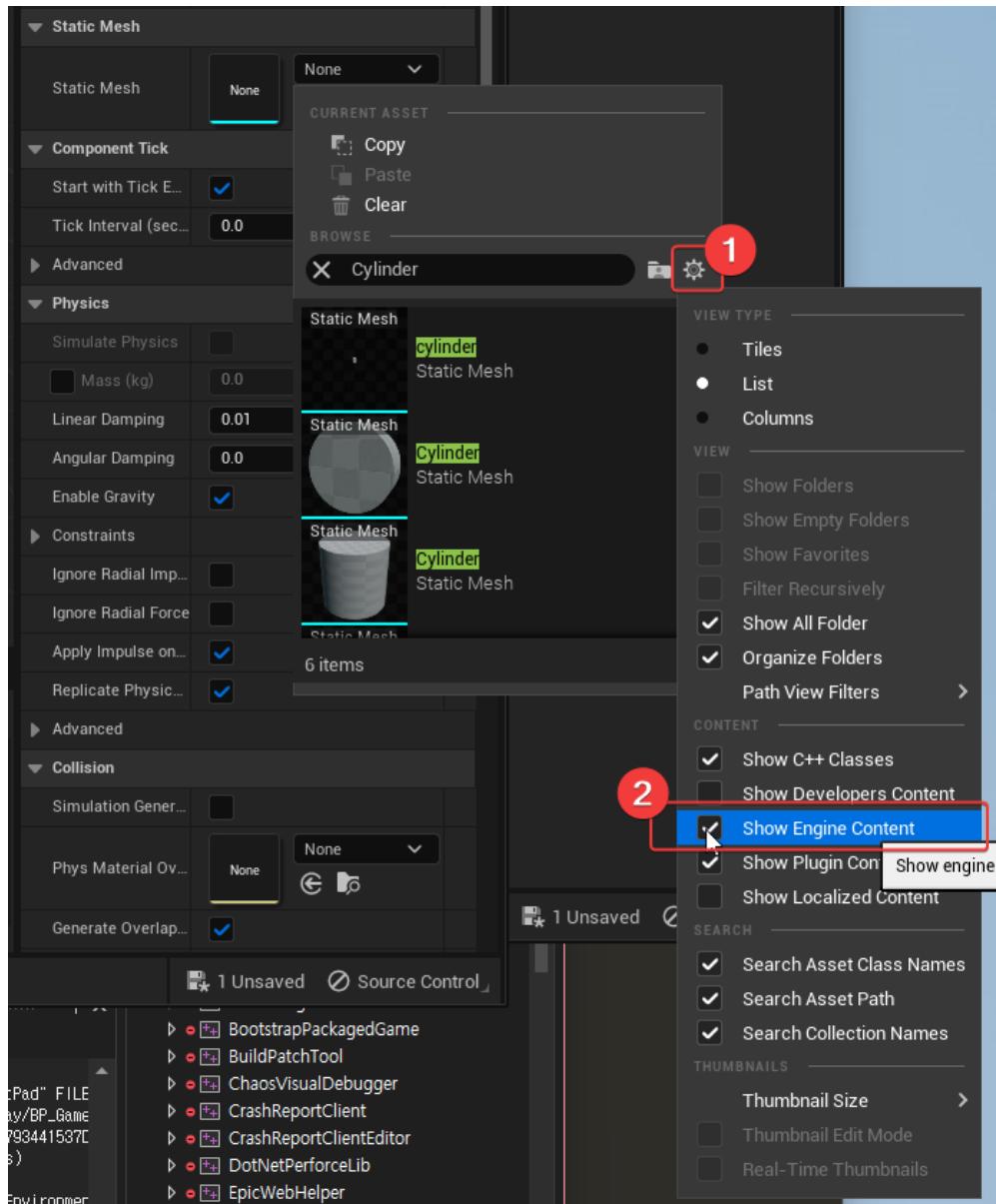
- StaticMesh이 PadBase 추가:



- 속성값 업데이트:

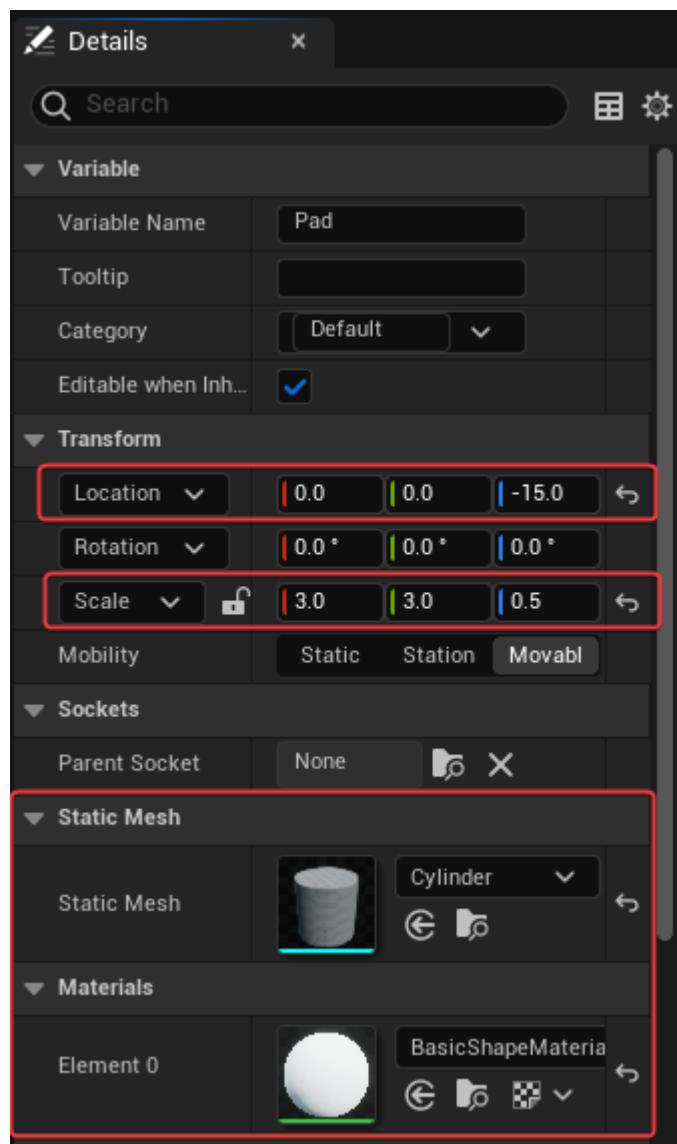
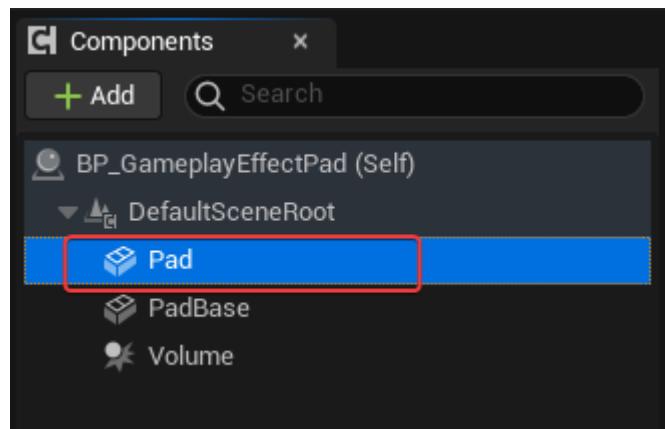


- 아래와 같이 Engine Content를 활성화해야, Cylinder 기본 메시에 접근 가능하다!

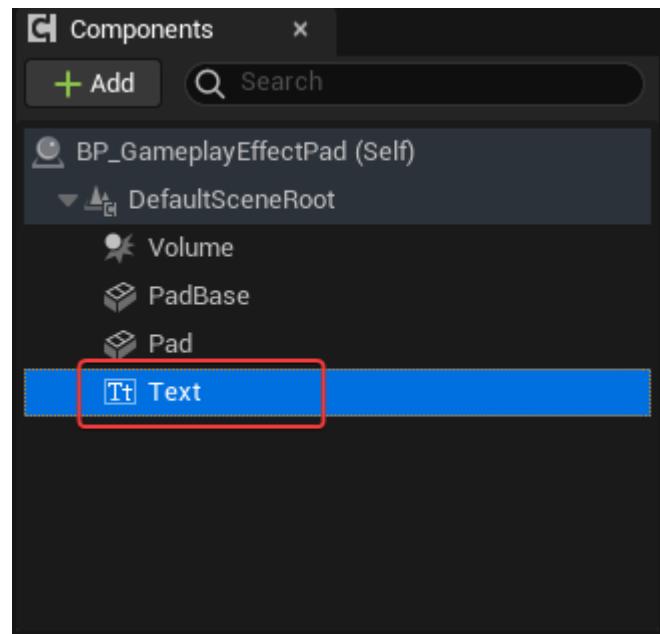


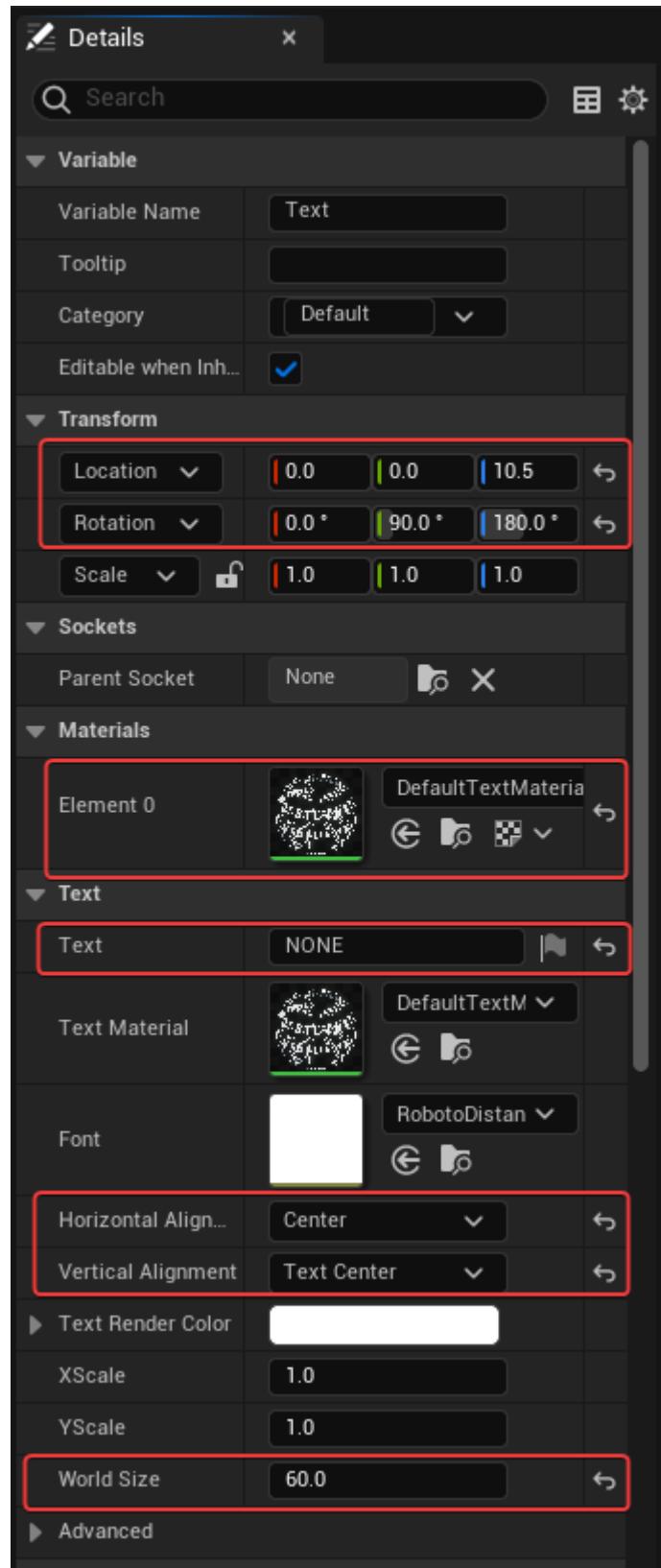
- VertexColorMaterial로 설정하자

StaticMesh인 Pad 추가:



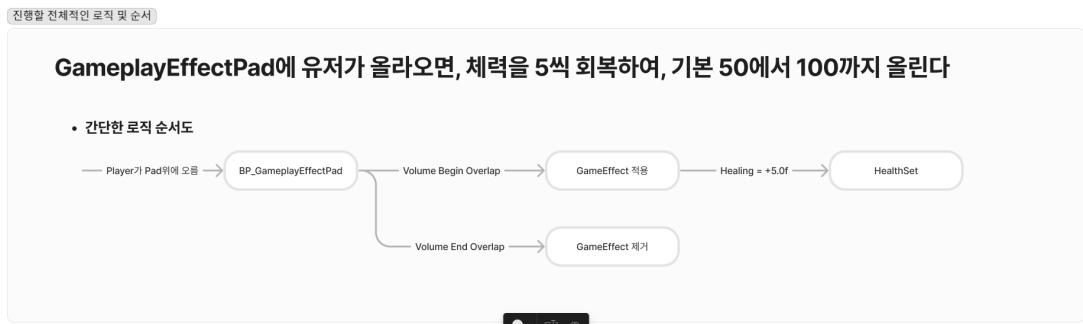
Text 추가:





- 오늘 프로젝트는 GameEffect에 대한 실습을 통한 이해이다.
 - GameEffect를 본격적으로 실습에 앞서, 대략적인 로직 순서도를 이해하는데 필요한 것들이 구현되었다.

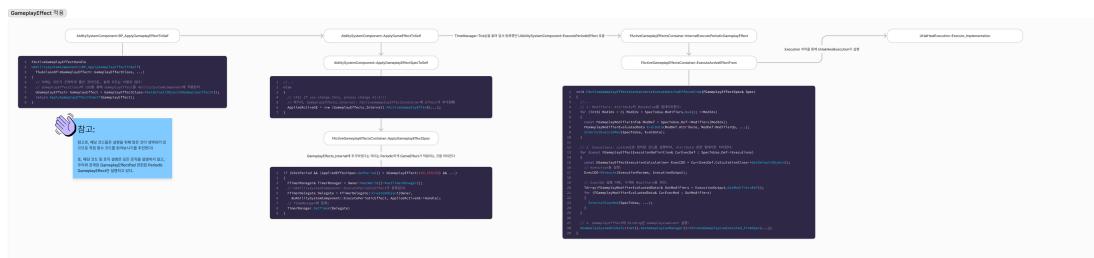
- 아래의 그림을 통해, 우리가 진행할 대략적인 순서와 로직을 이해해보자:



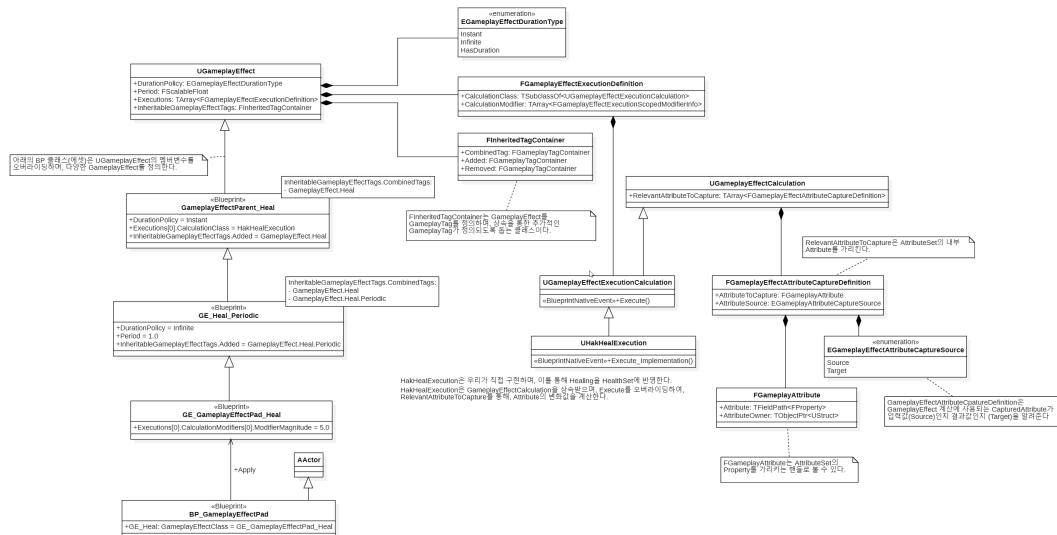
GameplayEffect 적용/해제 이해:

▼ 펼치기

- 아래의 그림을 보며, GameplayEffect의 적용에 대해 이해해보자:



- 앞서 본 로직 이해를 통해 각 클래스-간 관계를 확인하자:





위의 클래스 구조도를 한번에 이해하기 힘들 수 있다. 한번 전체적으로 클론코딩해보고, 어떻게 GameplayEffect가 돌아가는지 디버깅하며 살펴보기 전에 해당 구조도를 읽어보며 분석하는데 도움되었으면 좋겠다.

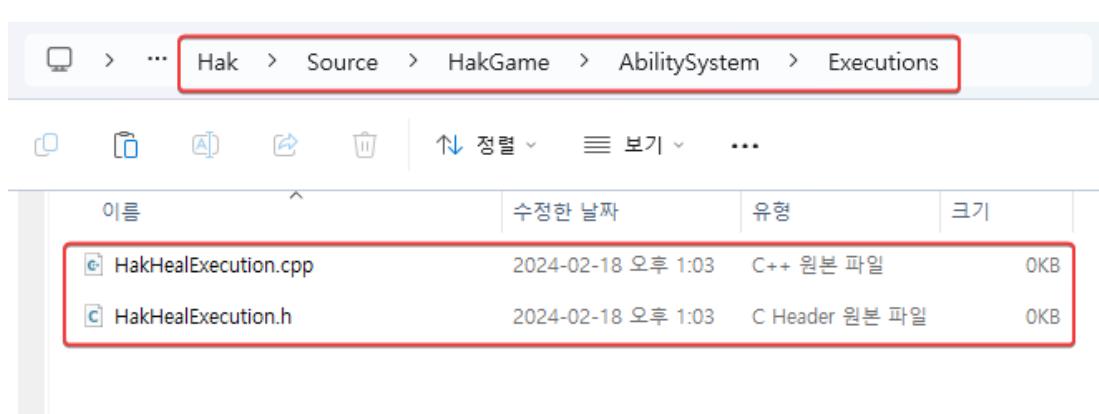
위의 클래스 구조는
필자가 현재 이해하는 GameplayEffect이다.

GE_GameplayEffectPad_Heal:

▼ 펼치기

- 앞서 잠깐 보았던 클래스 다이어그램에서 알 수 있듯이, GE_GameplayEffectPad_Heal은 아래의 클래스 순서로 상속 받고 있다:
 - GameplayEffectParent_Heal
 - GE_Heal_Periodic
 - GE_GameplayEffectPad_Heal
- 위의 순서로 하나씩 구현하도록 하겠다.
- 앞서, 클래스 다이어그램에서 알 수 있듯이, 우리가 구현할 GE_GameplayEffectPad_Heal는 GameplayEffect로 Execution을 기반하고 있다:
 - 그래서 먼저 UHakHealExecution을 구현하자.

HakHealExecution.h/.cpp 파일을 생성하자:



- AbilitySystem에 Executions 폴더를 생성하였다

□ HakHealExecution 정의:

```
#pragma once
#include "GameplayEffectExecutionCalculation.h"
#include "HakHealExecution.generated.h"

// ...
// HakHealExecution은 GameplayEffect의 Execution을 사용자 정의에 따라 GameplayEffect의 처리할 수 있다;
// - HealExecution 이름에서 알 수 있듯이, HealthAttribute의 Healing을 적용한다
// UCLASS()
class UHakHealExecution : public UGameplayEffectExecutionCalculation
{
    GENERATED_BODY()
public:
    UHakHealExecution();
    // 해당 메서드는 GameplayEffectExecutionCalculation의 Execute() BlueprintNativeEvent를 오버라이드 한다
    virtual void Execute_Implementation(const FGameplayEffectCustomExecutionParameters& ExecutionParams, FGameplayEffectCustomExecutionOutput& OutExecutionOutput) const override;
};

// ...
// HakHealExecution.h
// UE_INLINE_GENERATED_CPP_BY_NAME(HakHealExecution)
UHakHealExecution::UHakHealExecution()
: Super()
{
}

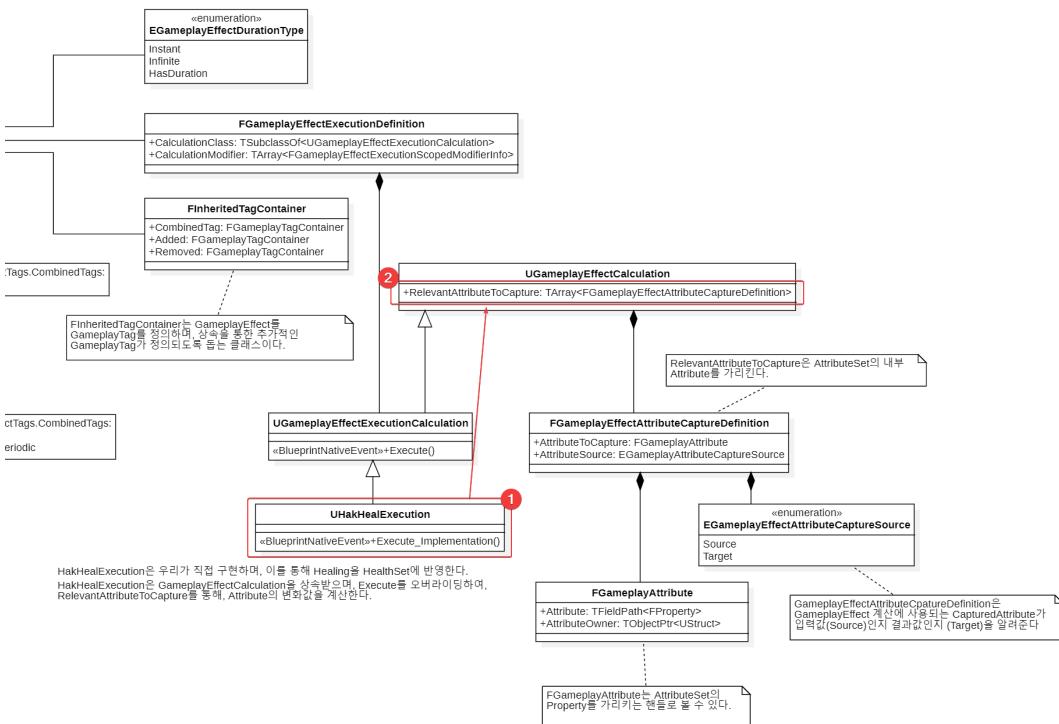
void UHakHealExecution::Execute_Implementation(const FGameplayEffectCustomExecutionParameters& ExecutionParams, FGameplayEffectCustomExecutionOutput& OutExecutionOutput) const
{
}
```

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

UHakHealExecution::UHakHealExecution()
: Super()
{
}

void UHakHealExecution::Execute_Implementation(const FGameplayEffectCustomExecutionParameters& ExecutionParams, FGameplayEffectCustomExecutionOutput& OutExecutionOutput) const
{
}
```

□ HakHealExecution에서 HealthSet의 HealthAttribute에 적용할 값 계산을 위한 입력값을 저장하는 RelevantAttributeToCapture이 설정이 필요하다:



- RelevantAttributesToCapture에 HealthSet이 아닌 Healing할 정보를 담고 있는 CombatSet인 또 다른 AttributeSet이 필요하다.

□ HakCombatSet.h/.cpp 추가:

이름	수정한 날짜	유형	크기
HakAttributeSet.cpp	2024-02-17 오후 3:47	C++ 원본 파일	1KB
HakAttributeSet.h	2024-02-17 오후 3:42	C Header 원본 파일	1KB
HakHealthSet.cpp	2024-02-17 오후 8:14	C++ 원본 파일	1KB
HakHealthSet.h	2024-02-17 오후 8:35	C Header 원본 파일	2KB
HakCombatSet.h	2024-02-18 오후 1:15	C Header 원본 파일	0KB
HakCombatSet.cpp	2024-02-18 오후 1:15	C++ 원본 파일	0KB

□ HakCombatSet 정의:

```
#pragma once

#include "AbilitySystemComponent.h"
#include "HakAttributeSet.h"
#include "HakCombatSet.generated.h"

/**
 * CombatSet은 이름 그대로, 전투와 관련된 Attribute를 담고 있는 Set이다:
 * - 현재는 BaseHeal만 있지만, BaseDamage도 추가하여, 완전한 CombatSet에 필요한 AttributeSet을 정의 가능하다
 */
UCLASS(BlueprintType)
class UHakCombatSet : public UHakAttributeSet
{
    GENERATED_BODY()
public:
    UHakCombatSet();
    ATTRIBUTE_ACCESSORS(UHakCombatSet, BaseHeal);

    /**
     * FGameplayAttribute가 참고하는 실제 AttributeSet에 있는 데이터이다 (float보다 해당 Struct를 사용하는 것을 추천)
     * - Healing의 단위를 의미한다
     * - e.g. 5.0f라면, Period당 5씩 Healing 된다는 의미
     */
    UPROPERTY(BlueprintReadOnly, Category="Hak|Combat")
    FGameplayAttributeData BaseHeal;
};
```

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

UHakCombatSet::UHakCombatSet()
    : Super()
    , BaseHeal(0.0f)
{}
```

- 오늘 우리에게 필요한 AttributeSet을 모두 정의하였다:
 - AttributeSet은 누가 소유할까?

- PlayerState인 HakPlayerState이다
- AbilitySystemComponent가 누가 소유하고 있는가 생각해보자:
 - 그것 또한 HakPlayerState이다!

□ HakPlayerState에 HealthSet과 CombatSet을 추가하자:

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

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

    // 변수로 캐싱하기 보다, Subobject형태로, HealthSet과 CombatSet을 추가하자:
    CreateDefaultSubobject<UHakHealthSet>(TEXT("HealthSet"));
    CreateDefaultSubobject<UHakCombatSet>(TEXT("CombatSet"));
}
```

- HealExecution에 필요한 HealthSet과 CombatSet을 정의 및 생성도 끝났으니, 이어서 HealExecution에 나머지 구현을 진행하자.

□ CombatSet의 BaseHealAttribute를 GameplayEffectCalculation의 RelevantAttributesToCapture에 등록하자:

```
#include "HakHealExecution.h"
#include "HakGame/AbilitySystem/Attributes/HakCombatSet.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakHealExecution)

/**
 * 해당 Struct를 사용하여, FGameplayEffectAttributeCaptureDefinition 인스턴스화하여 전달한다
 */
struct FHealStatics
{
    /** AttributeSet의 어떤 Attribute를 Capture할 것인지와 어떻게 Capture할지 정의를 담고 있다 (한번 보고 오자) */
    FGameplayEffectAttributeCaptureDefinition BaseHealDef;

    FHealStatics()
    {
        BaseHealDef = FGameplayEffectAttributeCaptureDefinition(UHakCombatSet::GetBaseHealAttribute(), EGameplayEffectAttributeCaptureSource::Source, true);
    }

    static FHealStatics* HealStatics()
    {
        // 계속 FHealStatics를 생성하는 것은 부하이니, 한번만 생성하고 재사용한다
        static FHealStatics Statics;
        return Statics;
    }
};

UHakHealExecution::UHakHealExecution()
: Super()
{
    // Source로 (입력값) Attribute를 캡처를 정의하자
    // - CombatSet::BaseHeal을 통해 Healing 값을 정의하고 최종 Execute할 때, 해당 값을 가져와서 Health에 Healing을 적용한다
    RelevantAttributesToCapture.Add(HealStatics().BaseHealDef);
}
```

□ HakHealExecution의 Execute_Implementation (BlueprintNativeEvent)

```

void UHakHealExecution::Execute_Implementation(const FGameplayEffectCustomExecutionParameters& ExecutionParams, FGameplayEffectCustomExecutionOutput& OutExecutionOutput) const
{
    // GameplayEffectSpec은 GameplayEffect의 현들로 생각하면 된다
    const FGameplayEffectSpecC Spec = ExecutionParams.GetOwningSpec();

    float BaseHeal = 0.0f;
    {
        FAggregatorEvaluateParameters EvaluateParameters;

        // 해당 함수 호출을 통해 HakCombatSet의 BaseHeal 값을 가져온다 (혹은 원가 Modifier에 누적되어 있다면, 최종 계산 결과가 나온다)
        ExecutionParams.AttemptCalculateCapturedAttributeMagnitude(HealStatics().BaseHealDef, EvaluateParameters, BaseHeal);
    }

    // RelevantAttributesCapture를 통해 최종 계산된 BaseHeal을 0.0이하가 되지 않도록 한다 (Healing이아니면!)
    const float HealingDone = FMath::Max(0.0f, BaseHeal);
    if (HealingDone > 0.0f)
    {
        // GameplayEffectCalculation 이후, Modifier로서, 추가된다.
        // - 해당 Modifier는 CombatSet에서 가져온 BaseHeal을 활용하여, HealthSet의 Healing에 추가해준다
        OutExecutionOutput.AddOutputModifier(FGameplayModifierEvaluatedData(UHakHealthSet::GetHealingAttribute(), EGameplayModOp::Additive, HealingDone));
    }
}

```

- 아직 우리는 HealthSet에 Healing에 값을 업데이트했지, Health에는 업데이트하지 않았다:
 - 어디서 업데이트 하는 걸까?
 - HakHealthSet에서 진행된다
 - **Healing의 값 변화를 감지하고 Health를 Healing 값을 활용하여 업데이트한다**

□ HakHealthSet::PreAttributeBaseChange와
HakHealthSet::PreAttributeChange:

□ 해당 메서드들을 오버라이드:

```


/** 
 * 아래의 HealthSet은 의미 그대로, 체력에 대한 속성값을 관리한다
 */
UCLASS(BlueprintType)
class UHakHealthSet : public UHakAttributeSet
{
    GENERATED_BODY()
public:
    UHakHealthSet();

    /**
     * 앞서 HakAttributeSet에서 정의했던, ATTRIBUTE_ACCESSORS를 통해, 아래 정의한 멤버변수와 똑같이 이름을 설정한다
     * - ATTRIBUTE_ACCESSORS의 Macro의 정의부분을 한번 살펴보자
     */
    ATTRIBUTE_ACCESSORS(UHakHealthSet, Health);
    ATTRIBUTE_ACCESSORS(UHakHealthSet, MaxHealth);
    ATTRIBUTE_ACCESSORS(UHakHealthSet, Healing);

    /**
     * Attribute의 값을 ClampAttribute()를 활용하여, 값의 범위를 유지시켜주기 위해
     * PreAttributeBaseChange와 PreAttributeChange 오버라이드
     */
    void ClampAttribute(const FGameplayAttribute& Attribute, float& NewValue) const;
    virtual void PreAttributeBaseChange(const FGameplayAttribute& Attribute, float& NewValue) const override;
    virtual void PreAttributeChange(const FGameplayAttribute& Attribute, float& NewValue) override;

    /** 현재 체력 */
    UPROPERTY(BlueprintReadOnly, Category="Hak|Health")
    FGameplayAttributeData Health;

    /** 체력 최대치 */
    UPROPERTY(BlueprintReadOnly, Category="Hak|Health")
    FGameplayAttributeData MaxHealth;

    /** 체력 회복치 */
    UPROPERTY(BlueprintReadOnly, Category="Hak|Health")
    FGameplayAttributeData Healing;
};


```

□ ClampAttribute():

```

void UHakHealthSet::ClampAttribute(const FGameplayAttribute& Attribute, float& NewValue) const
{
    // HealthAttribute는 [0, GetMaxHealth]로 설정
    if (Attribute == GetHealthAttribute())
    {
        NewValue = FMath::Clamp(NewValue, 0.0f, GetMaxHealth());
    }
    // MaxHealthAttribute는 [1.0, inf]로 설정:
    // 즉, MaxHealth는 1미만이 될 수 없음!
    else if (Attribute == GetMaxHealthAttribute())
    {
        NewValue = FMath::Max(NewValue, 1.0f);
    }
}

```

□ PreAttributeBaseChange()와 PreAttributeChanged():

```

void UHakHealthSet::PreAttributeBaseChange(const FGameplayAttribute& Attribute, float& NewValue) const
{
    Super::PreAttributeBaseChange(Attribute, NewValue);
    ClampAttribute(Attribute, NewValue);
}

void UHakHealthSet::PreAttributeChange(const FGameplayAttribute& Attribute, float& NewValue)
{
    Super::PreAttributeChange(Attribute, NewValue);
    ClampAttribute(Attribute, NewValue);
}

```

□ UHealthSet의 PreGameplayEffectExecute와 PostGameplayEffectExecute:

```

/***
 * 아래의 HealthSet은 의미 그대로, 체력에 대한 속성값을 관리한다
 */
UCLASS(BlueprintType)
class UHakHealthSet : public UHakAttributeSet
{
    GENERATED_BODY()
public:
    UHakHealthSet();

    /**
     * 앞서 HakAttributeSet에서 정의했던, ATTRIBUTE_ACCESSORS를 통해, 아래 정의한 멤버변수와 똑같이 이름을 설정한다
     * - ATTRIBUTE_ACCESSORS의 Macro의 정의부분을 한번 살펴보자
     */
    ATTRIBUTE_ACCESSORS(UHakHealthSet, Health);
    ATTRIBUTE_ACCESSORS(UHakHealthSet, MaxHealth);
    ATTRIBUTE_ACCESSORS(UHakHealthSet, Healing);

    /**
     * Attribute의 값을 ClampAttribute()를 활용하여, 값의 범위를 유지시켜주기 위해
     * PreAttributeBaseChange와 PreAttributeChange 오버라이드
     */
    void ClampAttribute(const FGameplayAttribute& Attribute, float& NewValue) const;
    virtual void PreAttributeBaseChange(const FGameplayAttribute& Attribute, float& NewValue) const override;
    virtual void PreAttributeChange(const FGameplayAttribute& Attribute, float& NewValue) override;

    /**
     * GameplayEffect가 HealthSet의 Attribute를 수정하기 전에 불리는 콜백함수이다:
     * - 이는 AttributeSet의 주석에도 잘 나와있듯이, Healing이 업데이트되면, Health를 Healing을 적용하여 업데이트 가능하다
     */
    virtual bool PreGameplayEffectExecute(const FGameplayEffectModCallbackData& Data) override;
    virtual void PostGameplayEffectExecute(const FGameplayEffectModCallbackData& Data) override;

    /** 현재 체력 */
    UPROPERTY(BlueprintReadOnly, Category="Hak|Health")
    FGameplayAttributeData Health;

    /** 체력 최대치 */
    UPROPERTY(BlueprintReadOnly, Category="Hak|Health")
    FGameplayAttributeData MaxHealth;

    /** 체력 회복치 */
    UPROPERTY(BlueprintReadOnly, Category="Hak|Health")
    FGameplayAttributeData Healing;
}

```

```

bool UHakHealthSet::PreGameplayEffectExecute(FGameplayEffectModCallbackData& Data)
{
    return Super::PreGameplayEffectExecute(Data);
}

void UHakHealthSet::PostGameplayEffectExecute(const FGameplayEffectModCallbackData& Data)
{
    Super::PostGameplayEffectExecute(Data);

    float MinimumHealth = 0.0f;

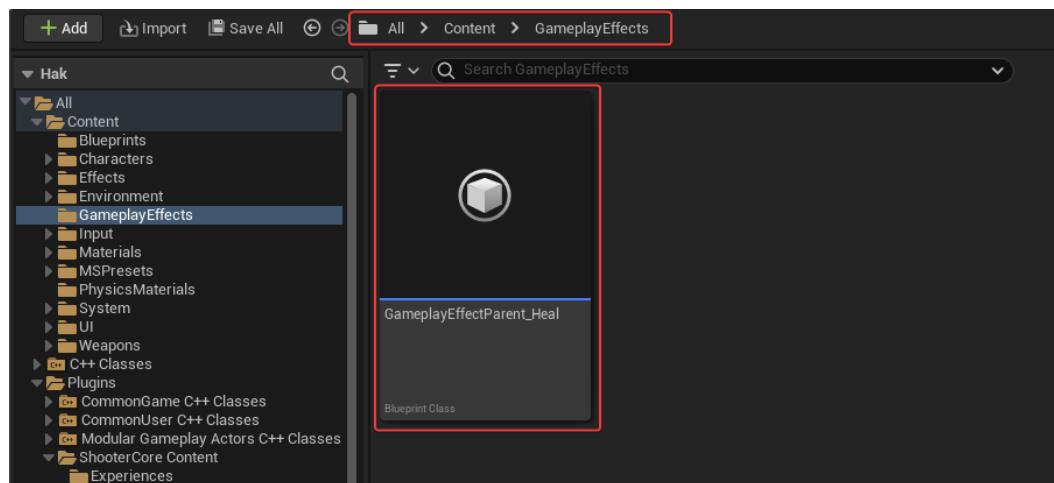
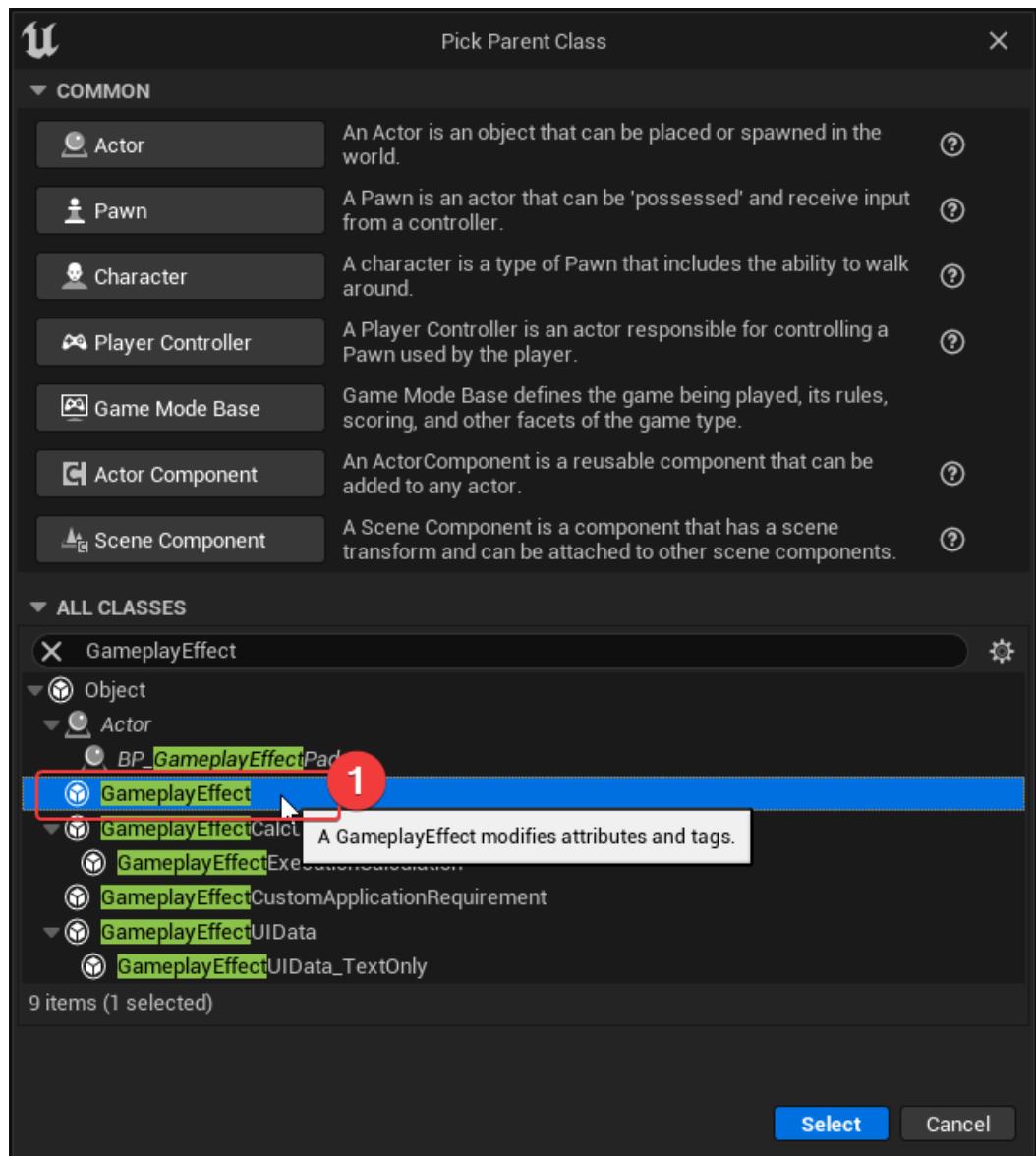
    // *** Healing이 업데이트 될 경우, Healing을 Health에 적용하고, Healing을 초기화해준다
    if (Data.EvaluatedData.Attribute == GetHealingAttribute())
    {
        SetHealth(FMath::Clamp(GetHealth() + GetHealing(), MinimumHealth, GetMaxHealth()));
        SetHealing(0.0f);
    }
    // Health 업데이트의 경우, [0,MaxHealth]로 맞추어주자
    else if (Data.EvaluatedData.Attribute == GetHealthAttribute())
    {
        SetHealth(FMath::Clamp(GetHealth(), MinimumHealth, GetMaxHealth()));
    }
}

```

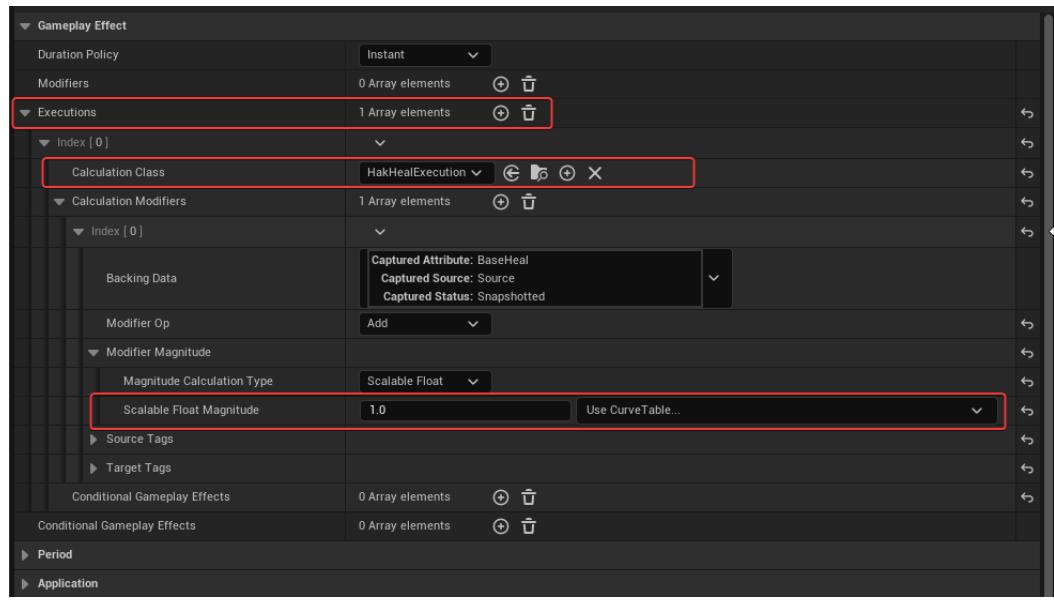
- 이제 HakHealExecution 구현이 끝났다:
 - 이를 활용하여, GameplayEffect에 적용 가능하다!
-

□ GameplayEffectParent_Heal:

□ GameplayEffectParent_Heal 생성:



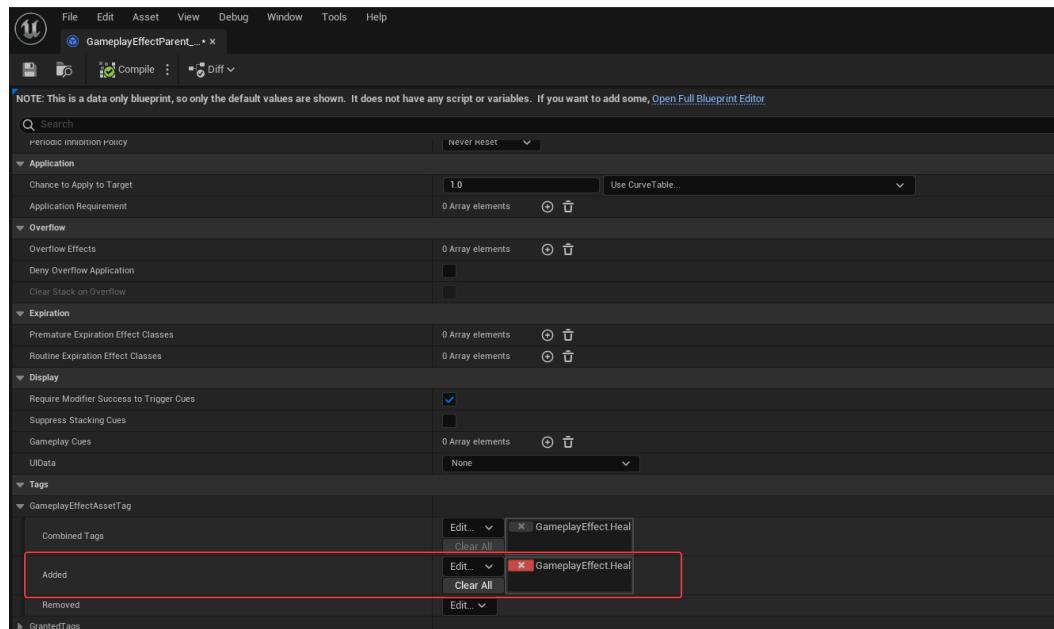
아래와 같이 GameplayEffect 관련 속성값을 오버라이드 하자:



□ BackingData에 살펴보기:

- HealExecution은 HakCombatSet의 BaseHeal을 입력 Attribute로 받는다

□ GameplayEffectParent_Heal의 GameplayTag를 할당하자:



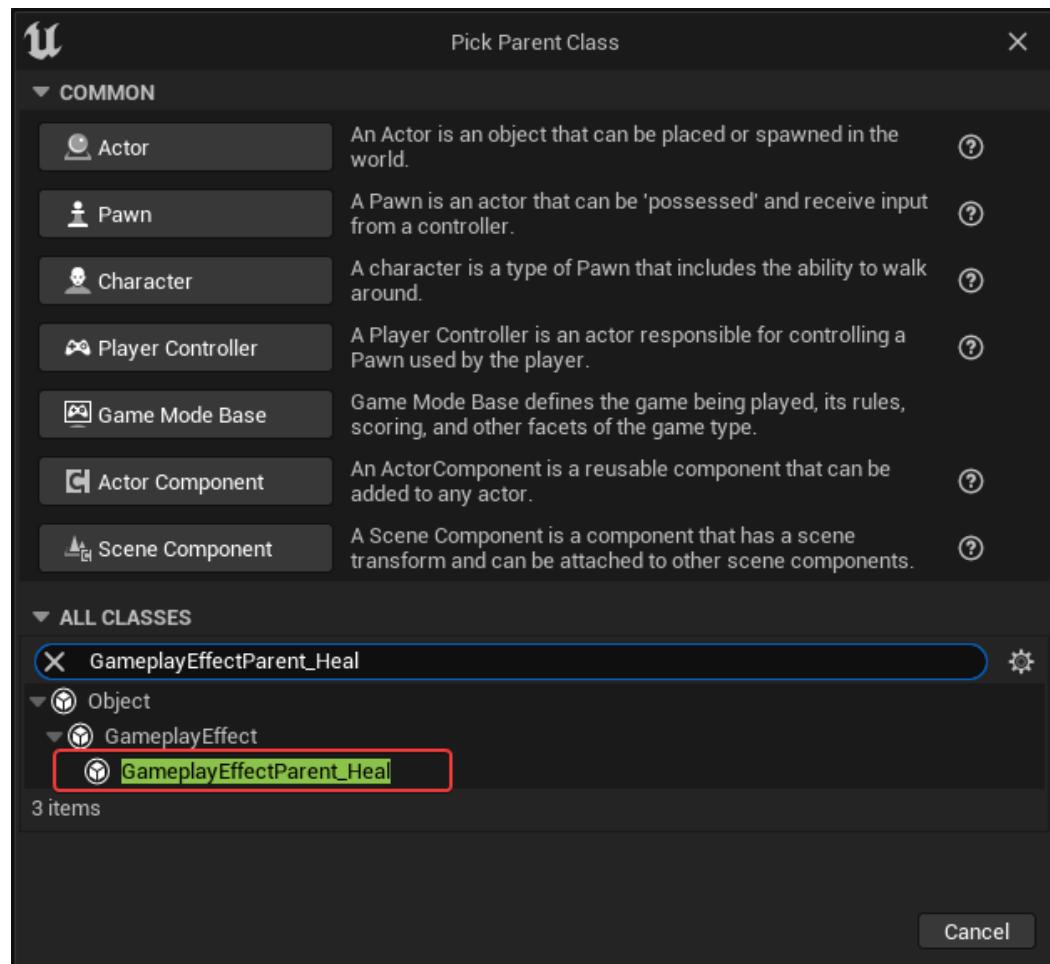
- InheritedTagContainer이므로 해당 클래스에 대해 Added에 추가하면 된다

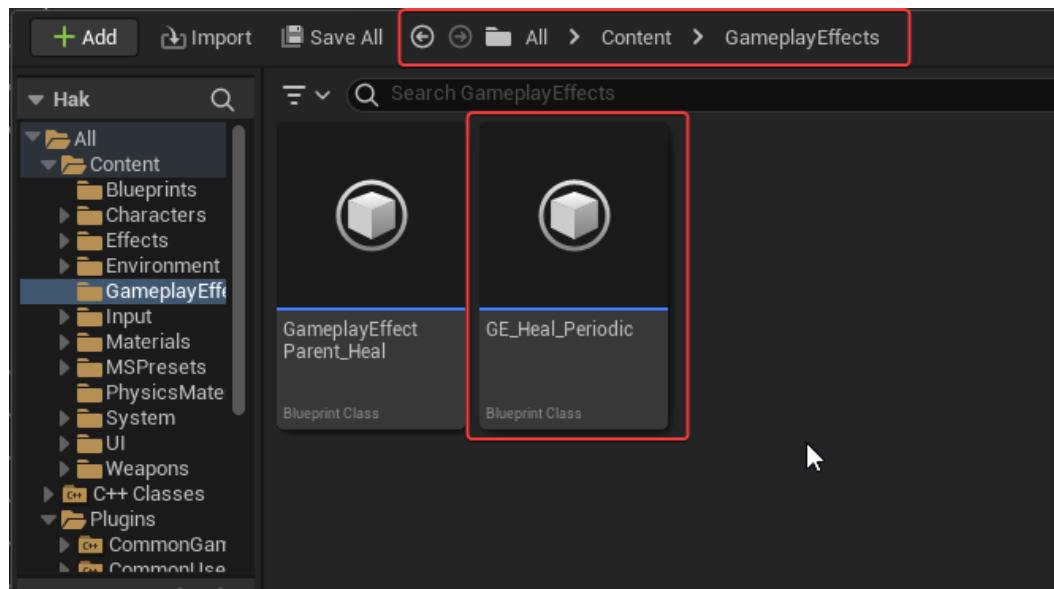


GAS에서 Gameplay 관련 클래스는 GameplayTag는 Identifier라고 생각하자

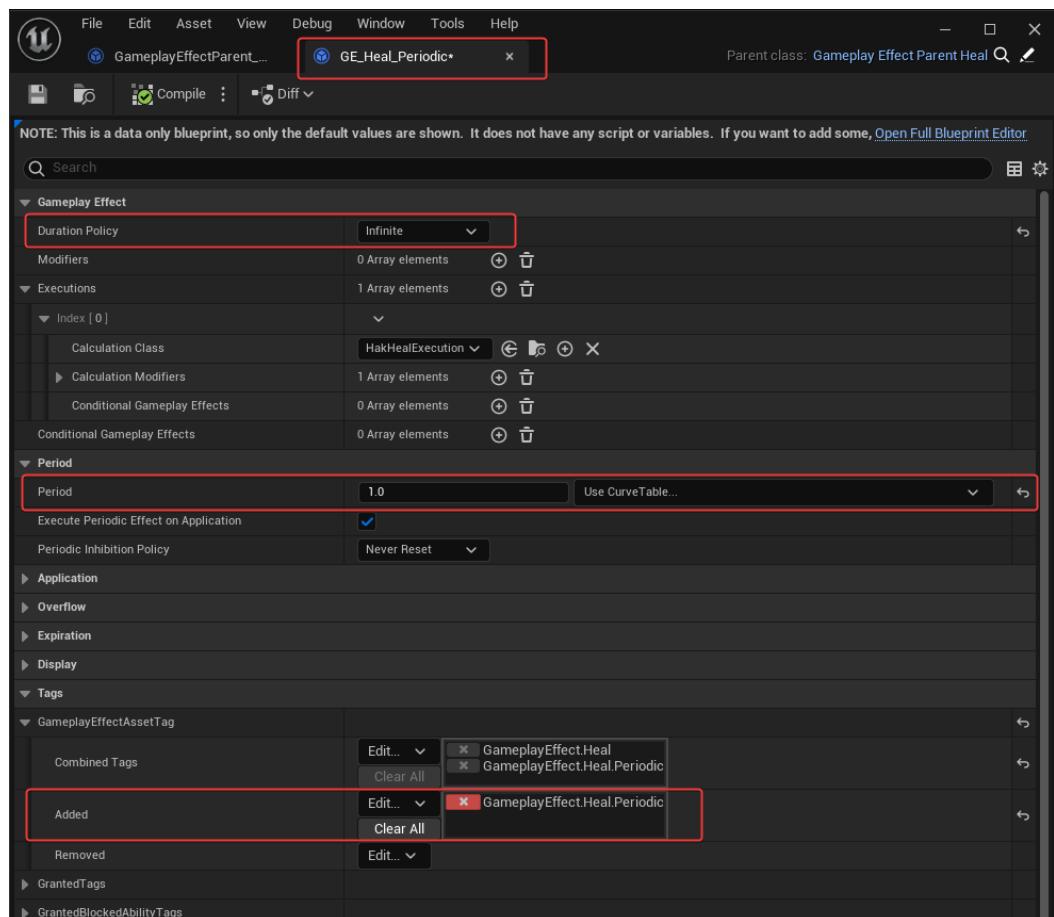
GE_Heal_Periodic:

GE_Heal_Periodic 생성:



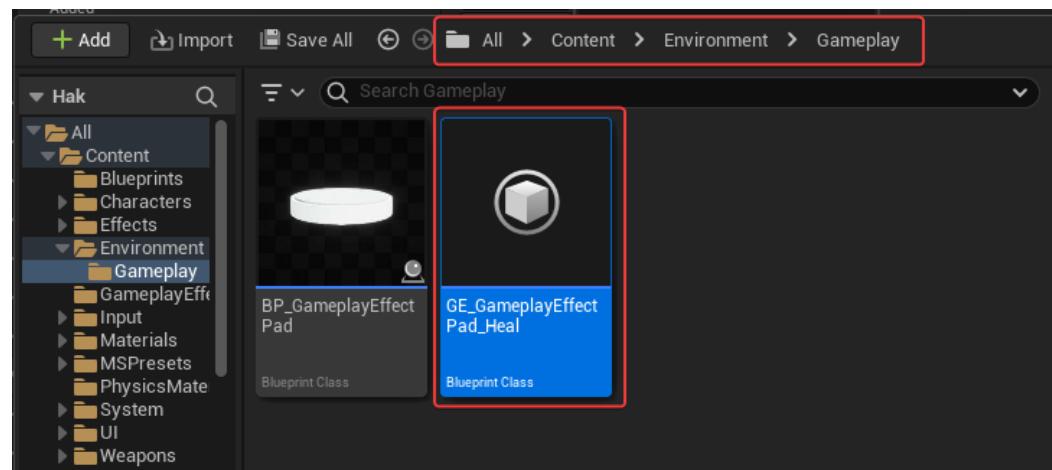
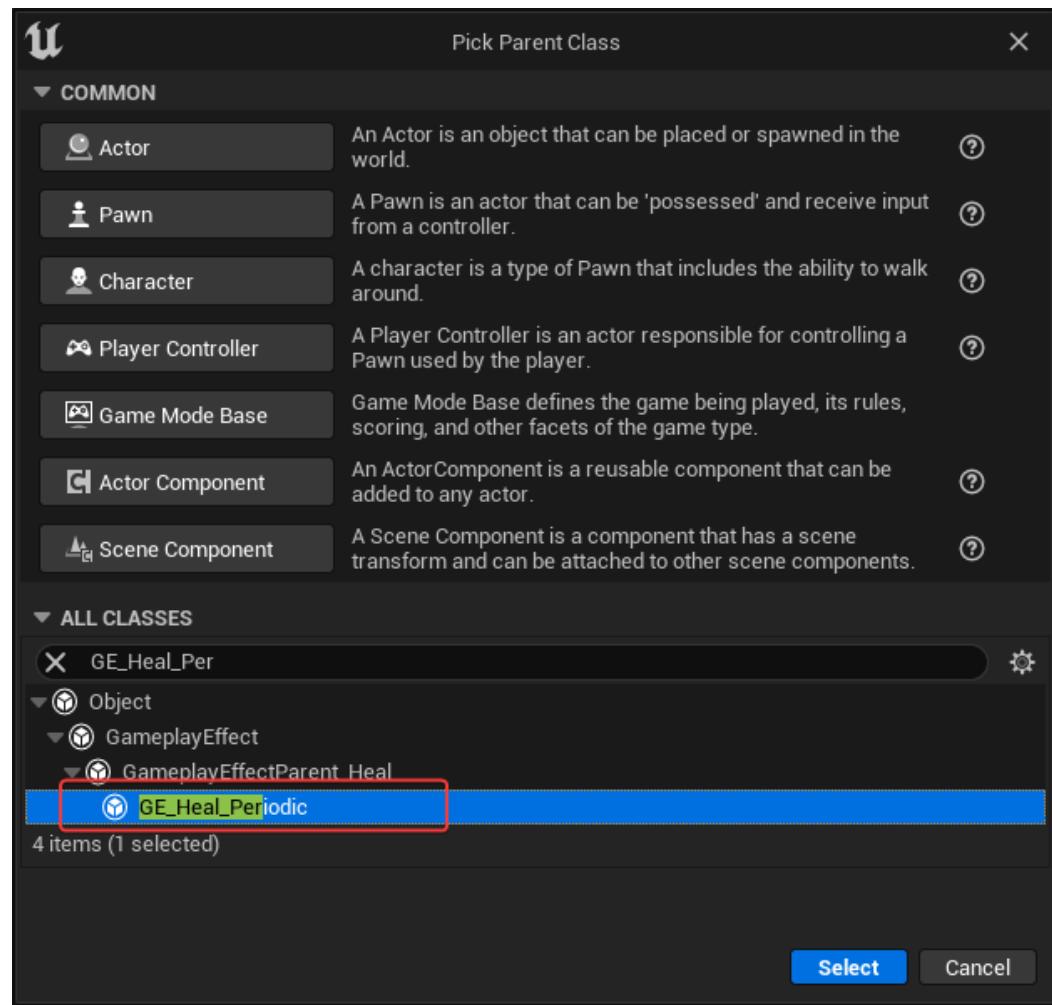


GE_Heal_Periodic 속성값 오버라이드:

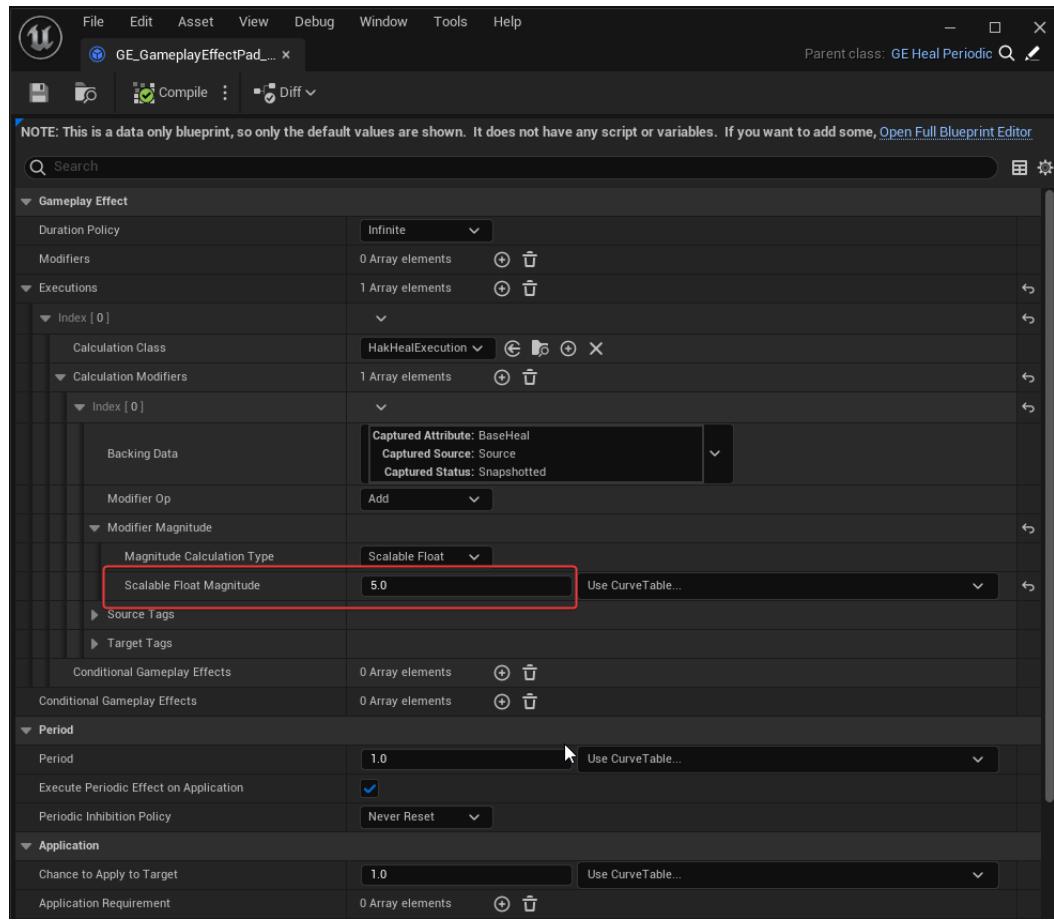


GE_GameplayEffectPad_Heal:

GE_GameplayEffectPad_Heal 생성:



GE_GameplayEffectPad_Heal 속성값 업데이트:

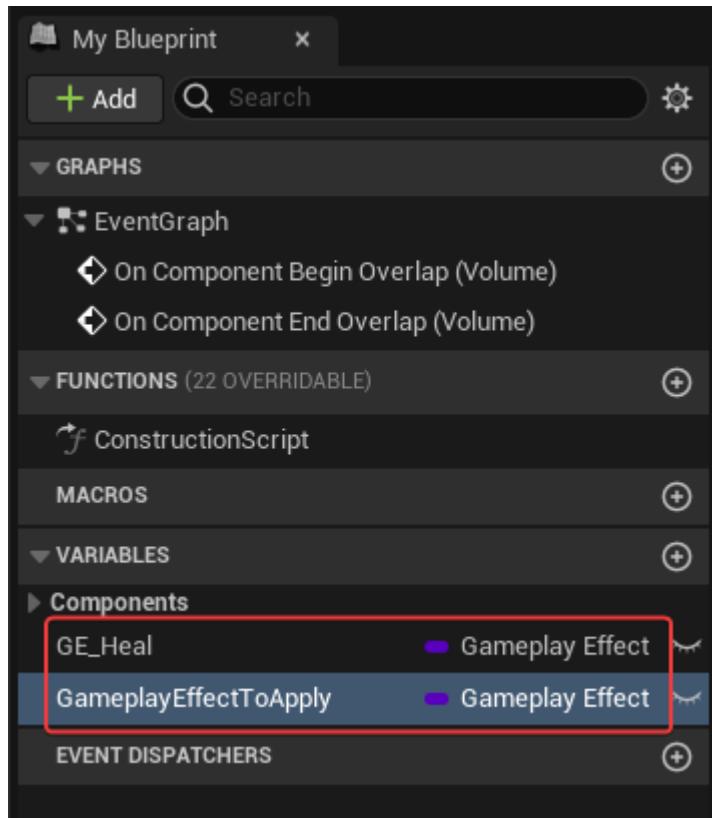


- 이제 GE_GameplayEffectPad_Heal이 준비되었다:
 - BP_GameplayEffectPad에 GameplayEffect를 할당하자

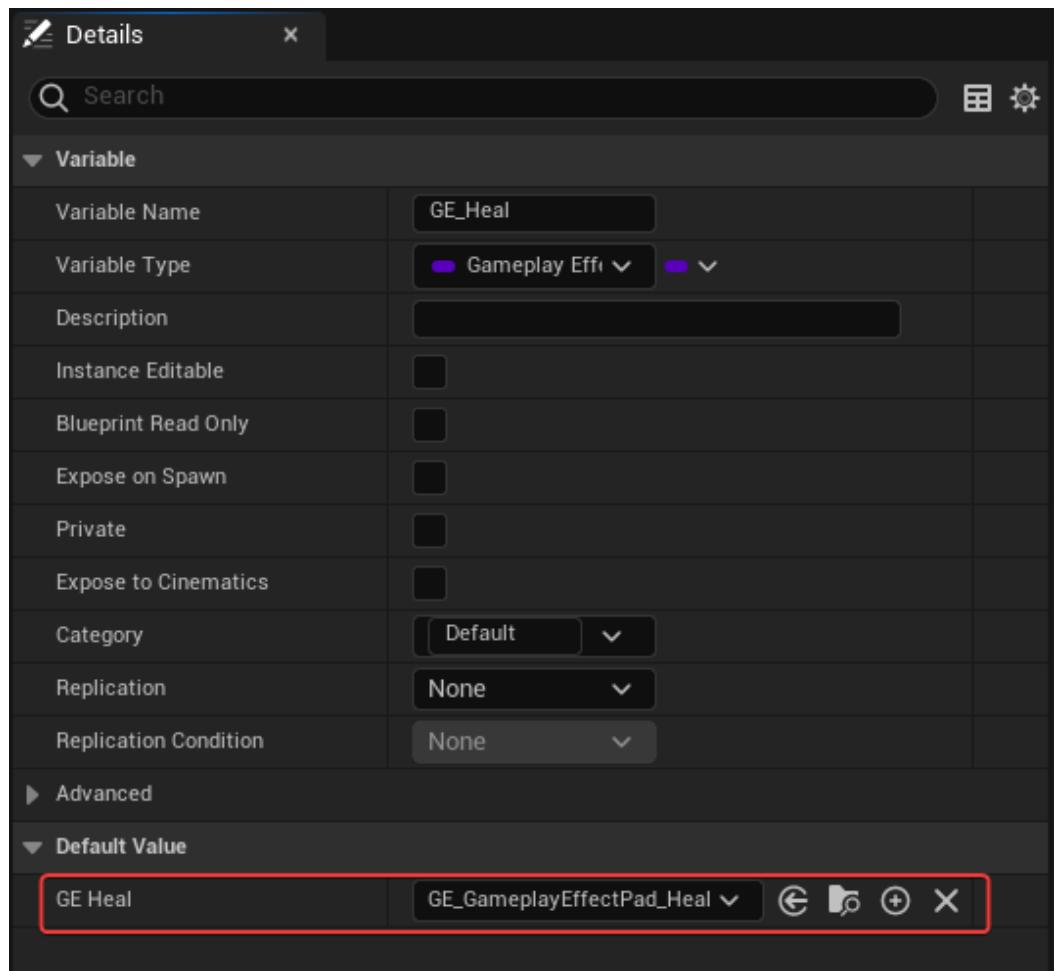
BP_GameplayEffectPad - 2:

▼ 펼치기

- GE_GameplayEffectPad_Heal이 준비되었으니, BP_GameplayEffectPad에 적용하기 위해 아래와 같이 멤버변수를 추가하자:



- 참고로, 위의 GE_Heal과 GameplayEffectToAppplay 둘 다
GameplayEffect의 Class이다!
- GE_Heal 디플트값으로 GE_GameplayEffectPad_Heal로 설정하자:



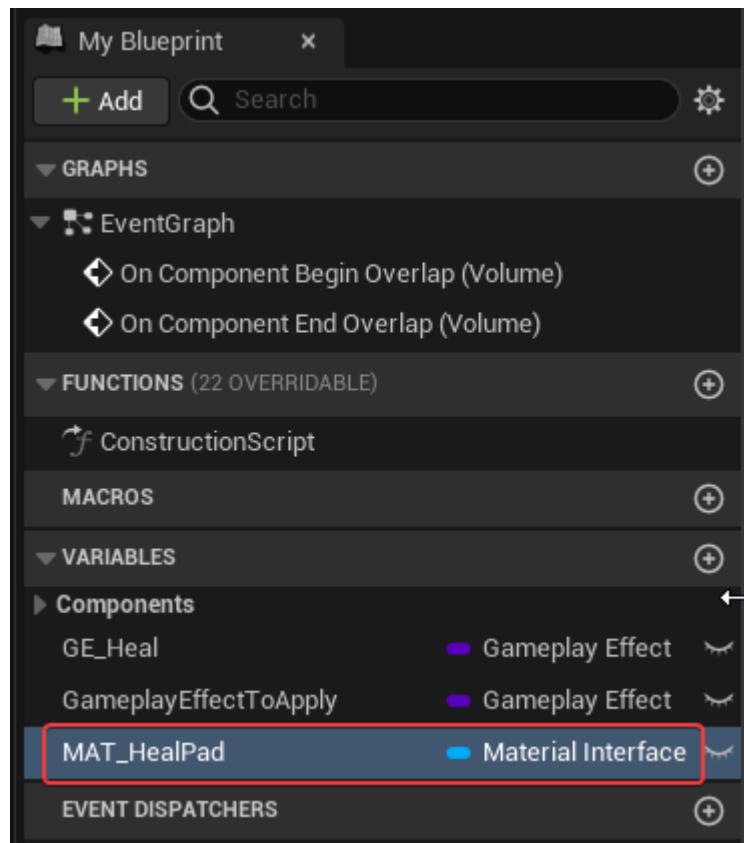
- GameplayEffectToApply에는 세팅 안한다!



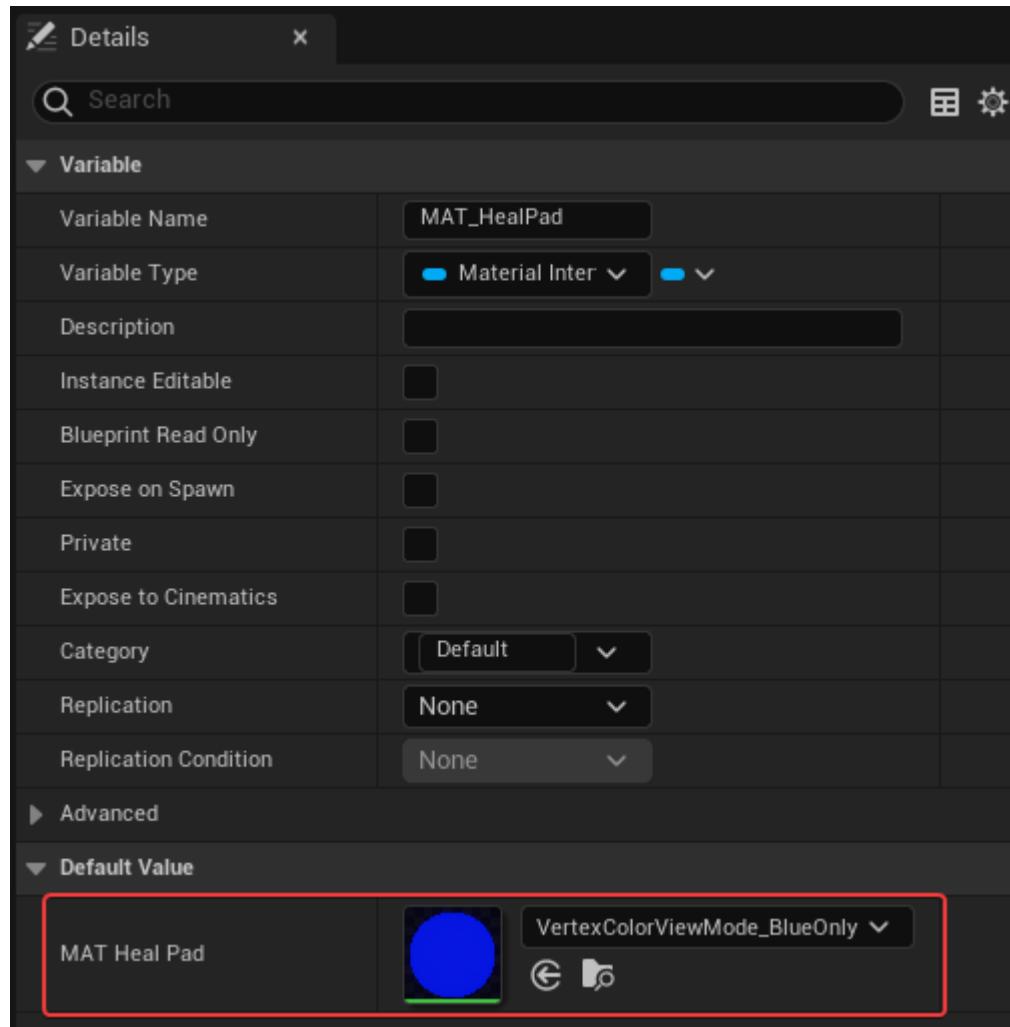
왜 하나만 있어도 되는데 2개를 넣을까?

- 일단 Lyra에서 BP_GameplayEffectPad는 GameplayEffectToApply에 Heal이 아닌 Damage용 GE로 설정할 수 있게 되어있다.
- 이를 통해, 하나의 BP로 복수개의 효과를 갖는 트랩을 만들 수 있는 것이다.

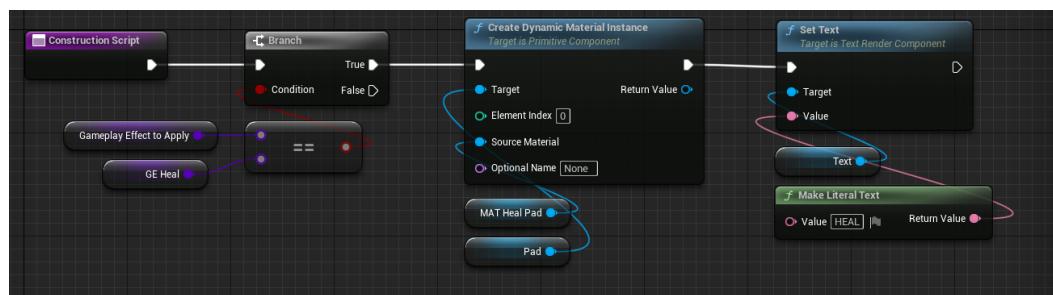
- MaterialInterface(Object)로 MAT_HealPad를 추가하자:



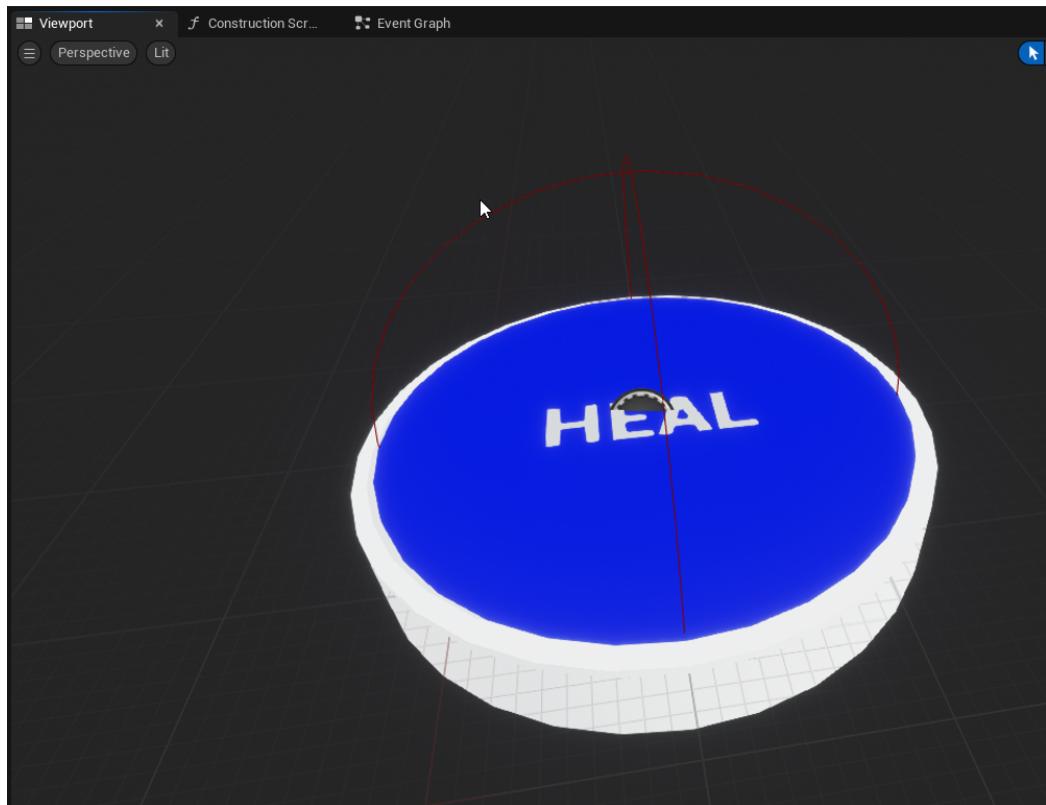
□ 아래와 같이 기본값으로 VertexColorViewMode_BlueOnly로 설정하자:



- 이제 준비된 멤버변수를 활용하여, HealPad용 Material인 MAT_HealPad를 적용하여, 의도한 블루 트랩을 만들어보자:
- 아래와 같이 ConstructionScript에 BP 로직을 추가하자:

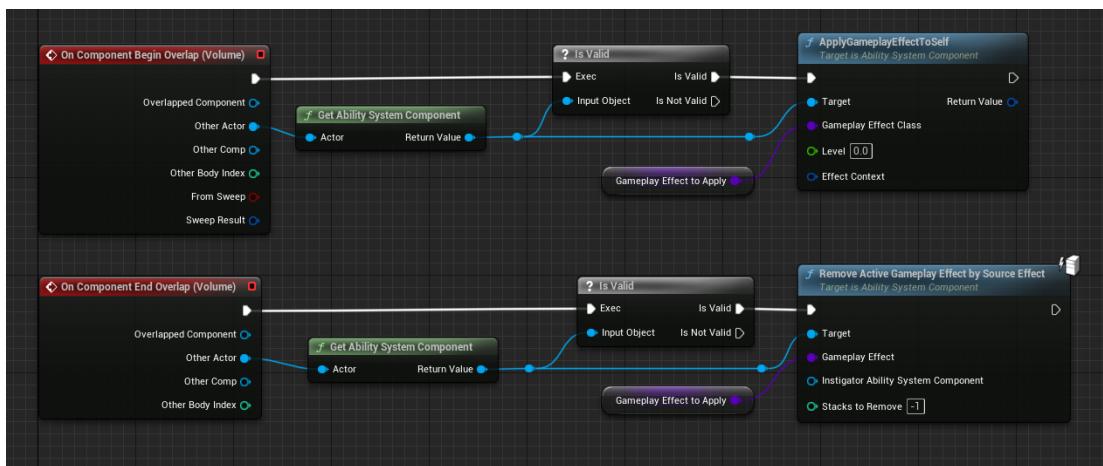


- 그럼 아래와 같이 BP_GameplayEffectPad가 형태가 바뀐다:

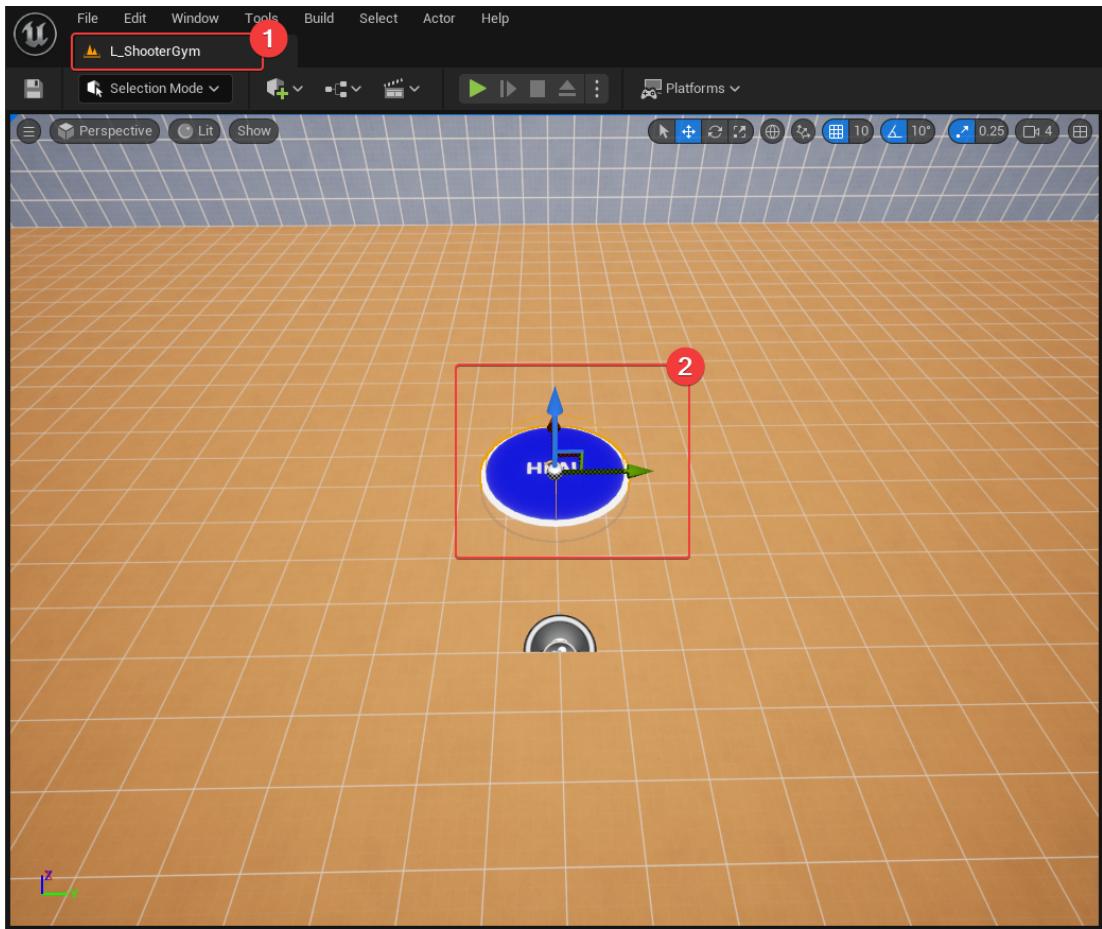


- 다시 처음에 의도했던 BP_GameplayEffectPad로 돌아가 우리는 해당 트랩에 캐릭터가 상호작용이 일어나면, Healing을 하고 싶었다

□ 아래와 같이 Component Begin/End Overlap을 BP로직 완성해주자:



□ BP_GameplayEffectPad를 L_ShooterGym 레벨에 배치하자:



- 이제 캐릭터를 해당 HEAL 트랩에 상호작용하여, Healing이 되는지 확인해보자:
- 앞서 간단히 보았듯이, HealthSet의 Healing이 작동하는지 혹은 HealExecution을 확인해봐도 된다:

- HakHealExecution::Execute_Implementation():

```

35     void UHakHealExecution::Execute_Implementation(const FGameplayEffectCustomExecutionParameters& ExecutionParams, FGameplayEffectCustomExecutionParameters OutExecutionOutput)
36     {
37         // GameplayEffectSpec은 GameplayEffect의 헨들로 생각하면 된다
38         const FGameplayEffectSpec& Spec = ExecutionParams.GetOwningSpec();
39
40         float BaseHeal = 0.0f;
41
42         {
43             FAggregatorEvaluateParameters EvaluateParameters;
44
45             // 해당 함수 호출을 통해 HakCombatSet의 BaseHeal 값을 가져온다 (혹은 원가 Modifier에 누적되어 있다면, 최종 계산 결과가 나온다)
46             ExecutionParams.AttemptCalculateCapturedAttributeMagnitude(HealStatics().BaseHealDef, EvaluateParameters, BaseHeal);
47
48             // RelevantAttributesCapture를 통해 최종 계산된 BaseHeal을 0.0이하가 되지 않도록 한다 (Healing이니깐!)
49             const float HealingDone = FMath::Max(0.0f, BaseHeal);
50
51             if (HealingDone > 0.0f) {
52                 // GameplayEffectCalculation 이후, Modifier로서, 추가한다:
53                 // - 해당 Modifier는 CombatSet에서 가져온 BaseHeal을 활용하여, HealthSet의 Healing에 추가해준다
54                 OutExecutionOutput.AddOutputModifier(FGameplayModifierEvaluatedData(UHakHealthSet::GetHealingAttribute(), EGameplayModOp::Additive,
55             }
56         }

```

- HealthSet::PostExecuteGameplayEffect():

```

46
47     void UHakHealthSet::PostGameplayEffectExecute(const FGameplayEffectModCallbackData& Data)
48     {
49         Super::PostGameplayEffectExecute(Data);
50
51         float MinimumHealth = 0.0f;
52
53         // *** Healing이 업데이트 될 경우, Healing을 Health에 적용하고, Healing을 초기화해준다
54         if (Data.EvaluatedData.Attribute == GetHealingAttribute())
55         {
56             SetHealth(FMath::Clamp(GetHealth() + GetHealing(), MinimumHealth, GetMaxHealth()));
57             SetHealing(0.0f); < 27ms elapsed
58         }
59         // Health 업데이트의 경우, [0,MaxHealth]로 맞추어주자
60         else if (Data.EvaluatedData.Attribute == GetHealthAttribute())
61         {
62             SetHealth(FMath::Clamp(GetHealth(), MinimumHealth, GetMaxHealth()));
63         }
64     }

```

Name	Type
↳ this	UHakHealthSet *
↳ UHakAttributeSet	UHakAttribute...
↳ Health	FGameplayAttr...
↳ MaxHealth	FGameplayAttr...
↳ Healing	FGameplayAttr...
Add item to watch	

- 이제 W_Healthbar UI에 출력할 HealthSet이 준비되었다! UI로 가보자!

HakHealthComponent:

▼ 펼치기

- W_Healthbar의 UI 로직을 완성하기에 앞서, UI에서 HealthSet을 가져오기 위해 HakHealthComponent를 활용한다:
 - HealthComponent는 PlayerState에 있는 HealthSet을 AbilitySystemComponent을 거쳐 Pawn(Character)에서 사용하도록 도와주는 Component이다.

□ HakHealthComponent 생성:

이름	수정한 날짜	유형	크기
HakCharacter.cpp	2024-01-02 오전 1:08	C++ 원본 파일	2KB
HakCharacter.h	2024-01-02 오전 1:08	C Header 원본 파일	1KB
HakHeroComponent.cpp	2024-01-02 오전 1:08	C++ 원본 파일	12KB
HakHeroComponent.h	2024-01-02 오전 1:08	C Header 원본 파일	3KB
HakPawnData.cpp	2023-10-28 오후 8:04	C++ 원본 파일	1KB
HakPawnData.h	2024-01-02 오전 1:08	C Header 원본 파일	2KB
HakPawnExtensionComponent.cpp	2024-01-02 오전 1:08	C++ 원본 파일	8KB
HakPawnExtensionComponent.h	2024-01-02 오전 1:08	C Header 원본 파일	3KB
HakHealthComponent.h	2024-02-18 오후 4:46	C Header 원본 파일	0KB
HakHealthComponent.cpp	2024-02-18 오후 4:46	C++ 원본 파일	0KB

□ HakHealthComponent 정의:

```

#pragma once
#include "Components/GameFrameworkComponent.h"
#include "Delegates/Delegate.h"
#include "HakHealthComponent.generated.h"

/** forward declarations */
class UHakAbilitySystemComponent;
class UHakHealthSet;
class UHakHealthComponent;
class AActor;
struct FOnAttributeChangeData;

/** Health 변화 블록을 위한 멀리게이트 */
DECLARE_DYNAMIC_MULTICAST_DELEGATE_FourParams(FHakHealth_AttributeChanged, UHakHealthComponent*, HealthComponent, float, OldValue, float, NewValue, AActor*, Instigator);

/**
 * - Character(Pawn)에 대해 체력관련 처리를 담당하는 Component이다
 * - 참고로 해당 클래스는 Blueprintable이다;
 * - 이는 멤버변수인 Delegate를 UI에서 바인딩하기 위함이다 (자세한건 물론하면서 알아보자)
 */
UCLASS(Blueprintable)
class UHakHealthComponent : public UGameFrameworkComponent
{
    GENERATED_BODY()
public:
    UHakHealthComponent(const FObjectInitializer& ObjectInitializer);

    /** HealthSet을 접근하기 위한 AbilitySystemComponent */
    UPROPERTY()
    TObjectPtr<UHakAbilitySystemComponent> AbilitySystemComponent;

    /** 개성된 HealthSet 레퍼런스 */
    UPROPERTY()
    TObjectPtr<const UHakHealthSet> HealthSet;

    /** health 변화에 따른 Delegate(Multicast) */
    FHakHealth_AttributeChanged OnhealthChanged;
};

```

```

#include "HakHealthComponent.h"
#include "HakGame/HakLogChannels.h"
#include "GameplayEffect.h"
#include "GameplayEffectExtension.h"
#include "HakGame/AbilitySystem/HakAbilitySystemComponent.h"
#include "HakGame/AbilitySystem/Attributes/HakHealthSet.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakHealthComponent)

UHakHealthComponent::UHakHealthComponent(const FObjectInitializer& ObjectInitializer)
: Super(ObjectInitializer)
{
    // HealthComponent는 PlayerState의 HealthSet과 Character(Pawn)간 Bridge 역할이라고 생각하면 된다:
    // - 따로 로직이 업데이트 될 필요가 없는 이벤트 기반으로 동작하는 컴포넌트로 이해하면 된다
    PrimaryComponentTick.bStartWithTickEnabled = false;
    PrimaryComponentTick.bCanEverTick = false;

    // InitializeWithAbilitySystem으로 ASC가 초기화되기 전까지 HealthSet과 ASC는 null이다:
    AbilitySystemComponent = nullptr;
    HealthSet = nullptr;
}

```

- HealthSet과 ASC 초기화를 위한 InitializeWithAbilitySystem와 UninitializeWithAbilitySystem:

```


/**
 * Character(Pawn)에 대해 체력관련 처리를 담당하는 Component이다
 * - 참고로 해당 클래스는 Blueprintable이다:
 * - 이는 멤버변수인 Delegate를 UI에서 바인딩하기 위함이다 (자세한건 클론하면서 알아보자)
 */
UCLASS(Blueprintable)
class UHakHealthComponent : public UGameFrameworkComponent
{
    GENERATED_BODY()

public:
    UHakHealthComponent(const FObjectInitializer& ObjectInitializer);

    /** ASC와 HealthSet 초기화 */
    void InitializeWithAbilitySystem(UHakAbilitySystemComponent* InASC);
    void UninitializeWithAbilitySystem();

    /** HealthSet을 접근하기 위한 AbilitySystemComponent */
    UPROPERTY()
    TObjectPtr<UHakAbilitySystemComponent> AbilitySystemComponent;

    /** 캐싱된 HealthSet 레퍼런스 */
    UPROPERTY()
    TObjectPtr<const UHakHealthSet> HealthSet;

    /** health 변화에 따른 Delegate(Multicast) */
    FHakHealth_AttributeChanged OnHealthChanged;
};


```

```

void UHakHealthComponent::InitializeWithAbilitySystem(UHakAbilitySystemComponent* InASC)
{
    // AActor::HakCharacter를 상속받고 있는 클래스일 것이다
    AActor* Owner = GetOwner();
    check(Owner);

    if (AbilitySystemComponent)
    {
        UE_LOG(LogHak, Error, TEXT("HakHealthComponent: Health component for owner [%s] has already been initialized with an ability system."), *GetNameSafe(Owner));
        return;
    }

    // ASC 캐싱
    AbilitySystemComponent = InASC;
    if (!AbilitySystemComponent)
    {
        UE_LOG(LogHak, Error, TEXT("HakHealthComponent: Cannot initialize health component for owner [%s] with NULL ability system."), *GetNameSafe(Owner));
        return;
    }

    // AbilitySystemComponent::GetSet은 SpawndAttributes에서 가져온다:
    // - ASC PlayerState에서 Subobject는 생성하고 따로 ASC에 등록하지 않는데 어떻게 등록되어있을까?
    // - AbilitySystemComponent::InitializeComponent()에서 GetObjectsWithOuter로 SpawndAttributes에 추가된다:
    //   - 잘 생각해보자 HealthSet은 PlayerState의 Subobject로 있고, ASC 또한 PlayerState에 있다:
    //     -> 이는 ASC에서 GetObjectsWithOuter로 HealthSet이 접근된다!!!
    // - 한번 AbilitySystemComponent::InitializeComponent()을 보자
    HealthSet = AbilitySystemComponent->GetSet<UHakHealthSet>();
    if (!HealthSet)
    {
        UE_LOG(LogHak, Error, TEXT("HakHealthComponent: Cannot initialize health component for owner [%s] with NULL health set on the ability system."), *GetNameSafe(Owner));
        return;
    }

    // 초기화 한번 해줬으니깐 Broadcast 해주자
    OnHealthChanged.Broadcast(this, HealthSet->GetHealth(), HealthSet->GetHealth(), nullptr);
}

void UHakHealthComponent::UninitializeWithAbilitySystem()
{
    AbilitySystemComponent = nullptr;
    HealthSet = nullptr;
}

```

- OnHealthChanged는 InitializeWithAbilitySystem()에만 호출되는게 아닌!
HealthSet의 Health가 변경이 있을 때, 호출되야 한다

AbilitySystemComponent를 통해, HealthSet의 HealthAttribute가 변경이 있을 때, 콜백으로 OnHealthChanged를 호출해주자:

```

/**
 * Character(Pawn)에 대해 체력관련 처리를 담당하는 Component이다
 * - 참고로 해당 클래스는 Blueprintable이다:
 * - 이는 멤버변수인 Delegate를 UI에서 바인딩하기 위함이다 (자세한건 물론하면서 알아보자)
 */
UCLASS(Blueprintable)
class UHakHealthComponent : public UGameFrameworkComponent
{
    GENERATED_BODY()

public:
    UHakHealthComponent(const FObjectInitializer& ObjectInitializer);

    /** ASC와 HealthSet 초기화 */
    void InitializeWithAbilitySystem(UHakAbilitySystemComponent* InASC);
    void UninitializeWithAbilitySystem();

    /** ASC를 통해, HealthSet의 HealthAttribute 변경이 있을 때 호출하는 메서드 (내부적으로 OnHealthChanged 호출) */
    void HandleHealthChanged(const FOnAttributeChangeData& ChangeData);

    /** HealthSet을 접근하기 위한 AbilitySystemComponent */
    UPROPERTY()
    TObjectPtr<UHakAbilitySystemComponent> AbilitySystemComponent;

    /** 캐싱된 HealthSet 레퍼런스 */
    UPROPERTY()
    TObjectPtr<const UHakHealthSet> HealthSet;

    /** health 변화에 따른 Delegate(Multicast) */
    FHakHealth_AttributeChanged OnHealthChanged;
};

```

```

static AActor* GetInstigatorFromAttrChangeData(const FOnAttributeChangeData& ChangeData)
{
    // GameEffectModifier에 Data가 있을 경우만 호출되는가보다 (사실 우리는 크게 관심없음)
    if (ChangeData.GEModData != nullptr)
    {
        const FGameplayEffectContextHandle& EffectContext = ChangeData.GEModData->EffectSpec.GetEffectContext();
        return EffectContext.GetOriginalInstigator();
    }
    return nullptr;
}

void UHakHealthComponent::HandleHealthChanged(const FOnAttributeChangeData& ChangeData)
{
    OnHealthChanged.Broadcast(this, ChangeData.OldValue, ChangeData.NewValue, GetInstigatorFromAttrChangeData(ChangeData));
}

```

- HandleHealthChanged를 ASC의
GetGameplayAttributeValueChangeDelegate를 통해 HealthSet의
HealthAttribute 변화가 있을때마다 이벤트를 받자:

```

void UHakHealthComponent::InitializeWithAbilitySystem(UHakAbilitySystemComponent* InASC)
{
    // Actor는 HakCharacter를 상속받고 있는 클래스일 것이다
    AActor* Owner = GetOwner();
    check(Owner);

    if (AbilitySystemComponent)
    {
        UE_LOG(LogHak, Error, TEXT("HakHealthComponent: Health component for owner [%s] has already been initialized with an ability system."), *GetNameSafe(Owner));
        return;
    }

    // ASC 초기화
    AbilitySystemComponent = InASC;
    if (!AbilitySystemComponent)
    {
        UE_LOG(LogHak, Error, TEXT("HakHealthComponent: Cannot initialize health component for owner [%s] with NULL ability system."), *GetNameSafe(Owner));
        return;
    }

    // AbilitySystemComponent::GetSet은 SpannedAttributes에서 가져온다;
    // - 그냥 PlayerState의 Subobject로 생성하고 따른 ASC에 등록하적이 있는데 이렇게 등록되어있을까?
    // - AbilitySystemComponent::InitializeComponent()에서 GetObjectsWithOuter로 SpannedAttributes에 추가된다:
    // - 잘 생각해보자 HealthSet은 PlayerState의 Subobject로 있고, ASC 또한 PlayerState에 있다;
    // - > 이는 ASC에서 GetObjectsWithOuter로 HealthSet이 접근된다!!!
    // - 한번 AbilitySystemComponent::InitializeComponent()를 보자
    HealthSet = AbilitySystemComponent->GetSet<UHakHealthSet>();
    if (!HealthSet)
    {
        UE_LOG(LogHak, Error, TEXT("HakHealthComponent: Cannot initialize health component for owner [%s] with NULL health set on the ability system."), *GetNameSafe(Owner));
        return;
    }

    // HealthSet의 HealthAttribute의 업데이트가 일어날때마다 호출할 블록으로 멤버메서드 HandleHealthChanged를 등록하자:
    AbilitySystemComponent->GetGameplayAttributeValueChangeDelegate(UHakHealthSet::GetHealthAttribute()).AddUObject(this, &ThisClass::HandleHealthChanged);

    // 초기화 한번 해줬으니까 Broadcast 해주자
    OnHealthChanged.Broadcast(this, HealthSet->GetHealth(), HealthSet->GetHealth(), nullptr);
}

```

- 그럼 이제 HakHealthComponent를 생성하고, InitializeWithAbilitySystem()
를 호출하자:

- HakHealthComponent를 HakCharacter에 생성하자:

```

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

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

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

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

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

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Hak|Character")
    TObjectPtr<UHakHealthComponent> HealthComponent;
};

```

```

#include "HakCharacter.h"
#include "HakPawnExtensionComponent.h"
#include "HakHealthComponent.h"
#include "HakGame/Camera/HakCameraComponent.h"
#include "HakGame/AbilitySystem/HakAbilitySystemComponent.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));
    }

    // HealthComponent 생성
    {
        HealthComponent = CreateDefaultSubobject<UHakHealthComponent>(TEXT("HealthComponent"));
    }
}

```

- InitializeWithAbilitySystem와 UninitializeWithAbilitySystem 호출:
 - HakPawnExtensionComponent의 OnAbilitySystemInitialized/OnAbilitySystemUninitialized 추가:

```


    /**
     * 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;

        /**
         * member methods
         */
        static UHakPawnExtensionComponent* FindPawnExtensionComponent(const AActor* Actor) { return (Actor ? Actor->FindComponentByClass<UHakPawnExtensionComponent>() : nullptr); }

        template <class T>
        const T* GetPawnData() const { return Cast<T>(PawnData); }
        void SetPawnData(const UHakPawnData* InPawnData);
        void SetupPlayerInputComponent();
        UHakAbilitySystemComponent* GetHakAbilitySystemComponent() const { return AbilitySystemComponent; }

        /** AbilitySystemComponent의 AvatarActor 대상 초기화/해제 호출 */
        void InitializeAbilitySystem(UHakAbilitySystemComponent* InASC, AActor* InOwnerActor);
        void UninitializeAbilitySystem();

        /** OnAbilitySystem[Initialized|Uninitialized] Delegate에 추가 */
        void OnAbilitySystemInitialized_RegisterAndCall(FSimpleMulticastDelegate::FDelegate Delegate);
        void OnAbilitySystemUninitialized_Register(FSimpleMulticastDelegate::FDelegate Delegate);

        /**
         * 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 생성할 데이터를 생성
         */
        UPROPERTY(EditInstanceOnly, Category = "Hak|Pawn")
        TObjectPtr<UHakPawnData> PawnData;

        /** AbilitySystem Component 캐싱 */
        UPROPERTY()
        TObjectPtr<UHakAbilitySystemComponent> AbilitySystemComponent;

        /** ASC Init||Uninit의 Delegate 추가 */
        FSimpleMulticastDelegate OnAbilitySystemInitialized;
        FSimpleMulticastDelegate OnAbilitySystemUninitialized;
    };


```

```


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

    if (AbilitySystemComponent == InASC)
    {
        return;
    }

    if (AbilitySystemComponent)
    {
        UninitializeAbilitySystem();
    }

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

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

    // OnAbilitySystemInitialized에 바인딩된 Delegate 호출
    OnAbilitySystemInitialized.Broadcast();
}

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

    if (AbilitySystemComponent->GetAvatarActor() == GetOwner())
    {
        // OnAbilitySystemUninitialized에 바인딩된 Delegate 호출
        OnAbilitySystemUninitialized.Broadcast();
    }

    AbilitySystemComponent = nullptr;
}

PRAGMA_DISABLE_OPTIMIZATION

void UHakPawnExtensionComponent::OnAbilitySystemInitialized_RegisterAndCall(FSimpleMulticastDelegate::FDelegate Delegate)
{
    // OnAbilitySystemInitialized의 UObject가 바인딩되어 있지 않으면 추가 (Uniqueness)
    if (!OnAbilitySystemInitialized.IsBoundToObject(Delegate.GetObject()))
    {
        OnAbilitySystemInitialized.Add(Delegate);
    }

    // 이미 ASC가 설정되었으면, Delegate에 추가하는게 아닌 바로 호출 (이미 초기화되어 있으니까!)
    if (AbilitySystemComponent)
    {
        Delegate.Execute();
    }
}

void UHakPawnExtensionComponent::OnAbilitySystemUninitialized_Register(FSimpleMulticastDelegate::FDelegate Delegate)
{
    if (!OnAbilitySystemUninitialized.IsBoundToObject(Delegate.GetObject()))
    {
        OnAbilitySystemUninitialized.Add(Delegate);
    }
}


```

- HakCharacter에 OnAbilitySystem[Un]Initialized() 메서드 추가 후, PawnExtComponent를 통해 OnAbilitySystem[Un]Initialized 등록:

```

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

    void OnAbilitySystemInitialized();
    void OnAbilitySystemUninitialized();

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

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

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

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

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Hak|Character")
    TObjectPtr<UHakHealthComponent> HealthComponent;
}

```

```

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));

// HealthComponent 생성
HealthComponent = CreateDefaultSubobject<UHakHealthComponent>(TEXT("HealthComponent"));

void AHakCharacter::OnAbilitySystemInitialized()
{
    UHakAbilitySystemComponent* HakASC = Cast<UHakAbilitySystemComponent>(GetAbilitySystemComponent());
    Check(HakASC);

    // HealthComponent의 ASC를 통한 초기화
    HealthComponent->InitializeWithAbilitySystem(HakASC);
}

void AHakCharacter::OnAbilitySystemUninitialized()
{
    HealthComponent->UninitializeWithAbilitySystem();
}

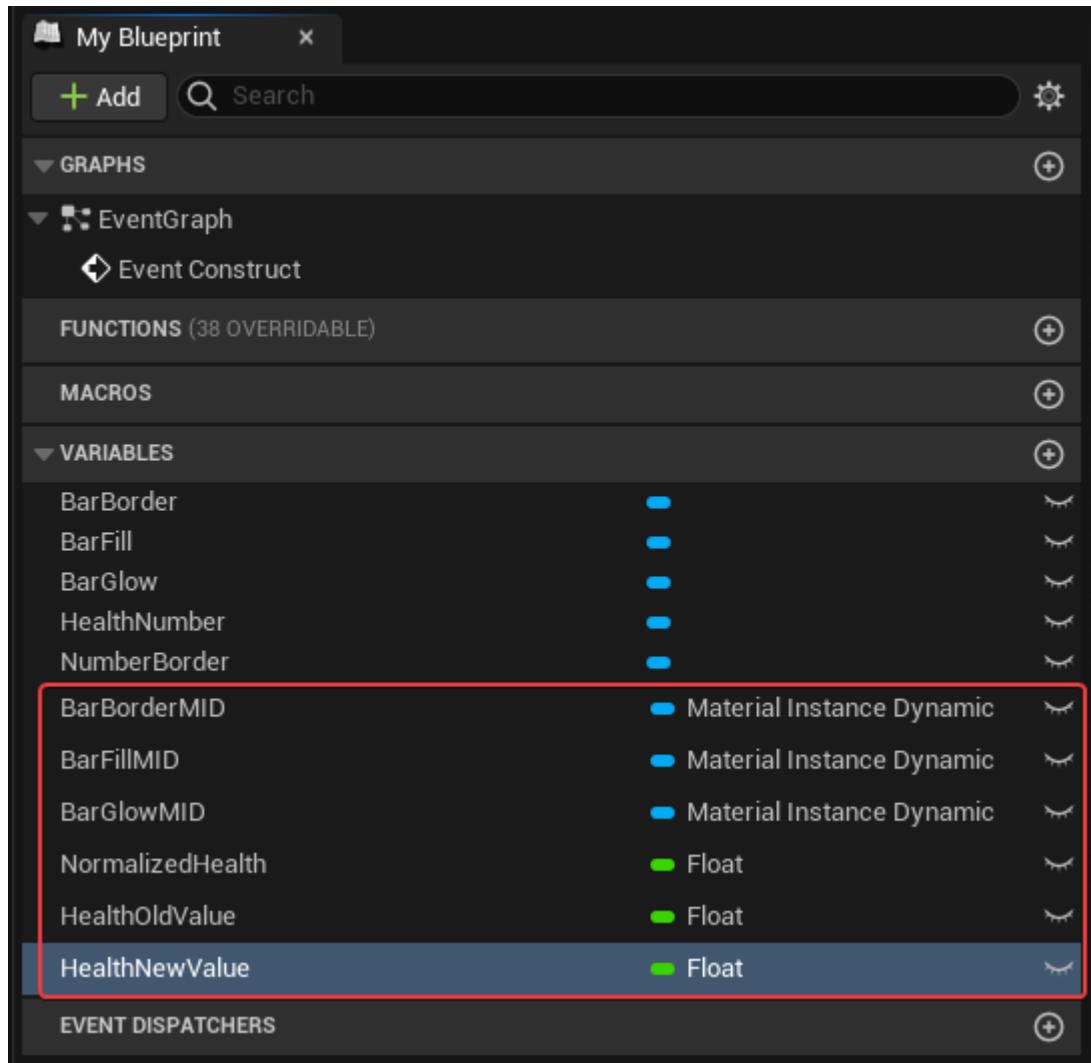
```

- 이제 HakCharacter를 통해, ASC 초기화 호출이 가능해졌다:
 - 현재 HealthComponent가 이를 활용하고 있다.

W_Healthbar - 2:

▼ 펼치기

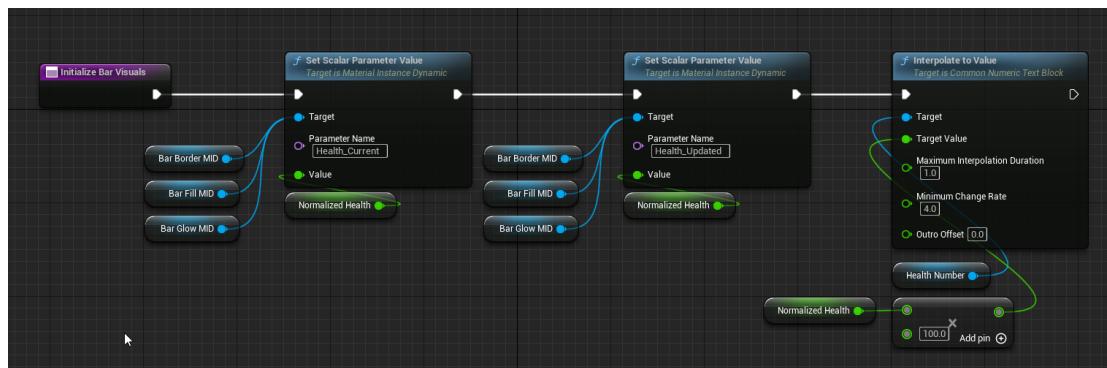
- 필요한 멤버 변수 추가하기:



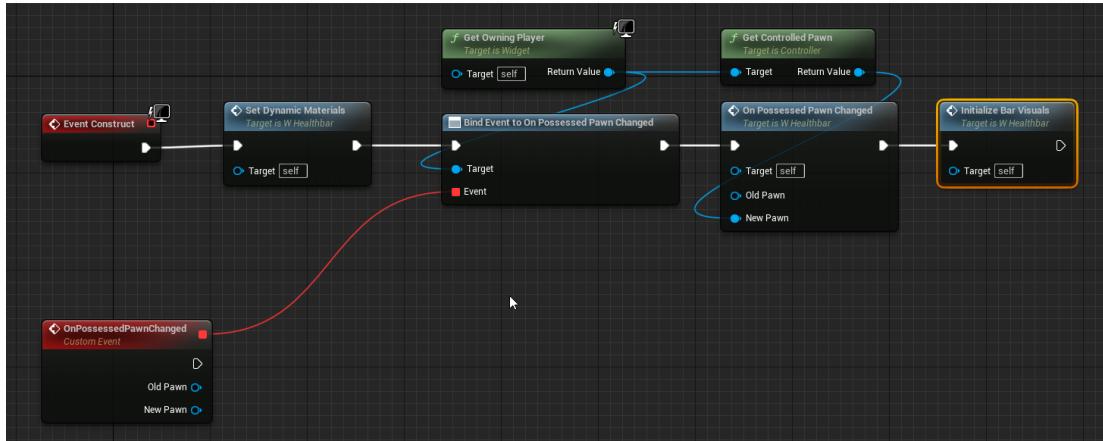
SetDynamicMaterials BP 함수 정의하자:



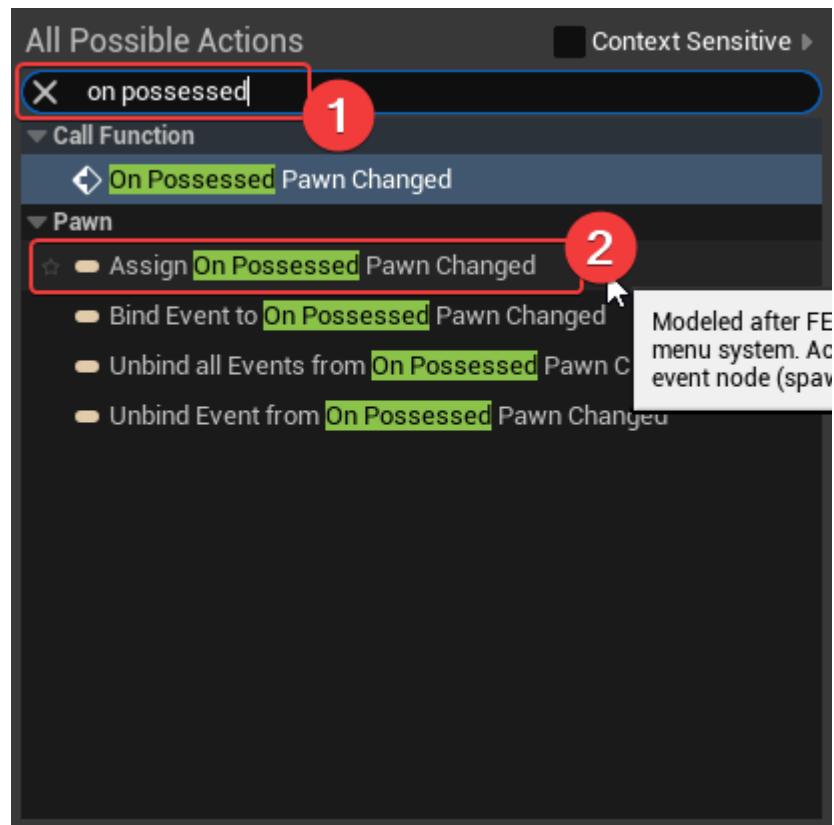
InitializeBarVisuals BP 함수 정의하자:



□ EventConstruct BP 로직 완성:



- Bind Event to OnPossessedPawnChanged와 CustomEvent인 OnPossessedPawnChanged 생성하기 위해, Assign OnPossessedPawnChanged를 활용해야 한다:



□ OnPossessedPawnChanged BP 로직을 완성하기에 앞서, HakHealthComponent에 BP 로직 완성에 필요한 BP 함수를 정의하자:

```

/**
 * Character(Pawn)에 대해 체력관련 처리를 담당하는 Component이다
 * - 참고로 해당 클래스는 Blueprintable이다:
 * - 이는 멤버변수인 Delegate를 UI에서 바인딩하기 위함이다 (자세한건 클론하면서 알아보자)
 */
UCLASS(Blueprintable)
class UHakHealthComponent : public UGameFrameworkComponent
{
    GENERATED_BODY()
public:
    UHakHealthComponent(const FObjectInitializer& ObjectInitializer);

    /**
     * BP 지원 메서드:
     */

    /** Actor(보통 ACharacter/APawn)의 HealthComponent를 반환 */
    UFUNCTION(BlueprintPure, Category="Hak|Health")
    static UHakHealthComponent* FindHealthComponent(const AActor* Actor);

    /** 아래의 UFUNCTION은 HealthSet의 Attribute에 접근하기 위한 BP Accessor 함수들 */
    UFUNCTION(BlueprintCallable, Category="Hak|Health")
    float GetHealth() const;

    UFUNCTION(BlueprintCallable, Category="Hak|Health")
    float GetMaxHealth() const;

    UFUNCTION(BlueprintCallable, Category="Hak|Health")
    float GetHealthNormalized() const;

    /** ASC와 HealthSet 초기화 */
    void InitializeWithAbilitySystem(UHakAbilitySystemComponent* InASC);
    void UninitializeWithAbilitySystem();

    /** ASC를 통해, HealthSet의 HealthAttribute 변경이 있을 때 호출하는 메서드 (내부적으로 OnHealthChanged 호출) */
    void HandleHealthChanged(const FOnAttributeChangeData& ChangeData);

    /** HealthSet을 접근하기 위한 AbilitySystemComponent */
    UPROPERTY()
    TObjectPtr<UHakAbilitySystemComponent> AbilitySystemComponent;

    /** 캐싱된 HealthSet 레퍼런스 */
    UPROPERTY()
    TObjectPtr<const UHakHealthSet> HealthSet;

    /** health 변화에 따른 Delegate(Multicast) */
    FHakHealth_AttributeChanged OnHealthChanged;
};

```

```

UHakHealthComponent* UHakHealthComponent::FindHealthComponent(const AActor* Actor)
{
    if (!Actor)
    {
        return nullptr;
    }

    UHakHealthComponent* HealthComponent = Actor->FindComponentByClass<UHakHealthComponent>();
    return HealthComponent;
}

float UHakHealthComponent::GetHealth() const
{
    return (HealthSet ? HealthSet->GetHealth() : 0.0f);
}

float UHakHealthComponent::GetMaxHealth() const
{
    return (HealthSet ? HealthSet->GetMaxHealth() : 0.0f);
}

float UHakHealthComponent::GetHealthNormalized() const
{
    if (HealthSet)
    {
        const float Health = HealthSet->GetHealth();
        const float MaxHealth = HealthSet->GetMaxHealth();
        return ((MaxHealth > 0.0f) ? (Health / MaxHealth) : 0.0f);
    }
    return 0.0f;
}

```

□ 앞서, 우리가 OnHealthChanged에 대해 UPROPERTY 선언을 깜박했다:

```


    /**
     * Character(Pawn)에 대해 체력관련 처리를 담당하는 Component이다
     * - 참고로 해당 클래스는 Blueprintable이다:
     * - 이는 멤버변수인 Delegate를 UI에서 바인딩하기 위함이다 (자세한건 클론하면서 알아보자)
     */
    UCLASS(Blueprintable)
    class UHakHealthComponent : public UGameFrameworkComponent
    {
        GENERATED_BODY()
    public:
        UHakHealthComponent(const FObjectInitializer& ObjectInitializer);

        /**
         * BP 지원 메서드:
         */

        /** Actor(보통 ACharacter/APawn)의 HealthComponent를 반환 */
        UFUNCTION(BlueprintPure, Category="Hak|Health")
        static UHakHealthComponent* FindHealthComponent(const AActor* Actor);

        /** 아래의 UFUNCTION은 HealthSet의 Attribute에 접근하기 위한 BP Accessor 함수들 */
        UFUNCTION(BlueprintCallable, Category="Hak|Health")
        float GetHealth() const;

        UFUNCTION(BlueprintCallable, Category="Hak|Health")
        float GetMaxHealth() const;

        UFUNCTION(BlueprintCallable, Category="Hak|Health")
        float GetHealthNormalized() const;

        /** ASC와 HealthSet 초기화 */
        void InitializeWithAbilitySystem(UHakAbilitySystemComponent* InASC);
        void UninitializeWithAbilitySystem();

        /** ASC를 통해, HealthSet의 HealthAttribute 변경이 있을때 호출하는 메서드 (내부적으로 OnHealthChanged 호출) */
        void HandleHealthChanged(const FOnAttributeChangeData& ChangeData);

        /** HealthSet을 접근하기 위한 AbilitySystemComponent */
        UPROPERTY()
        TObjectPtr<UHakAbilitySystemComponent> AbilitySystemComponent;

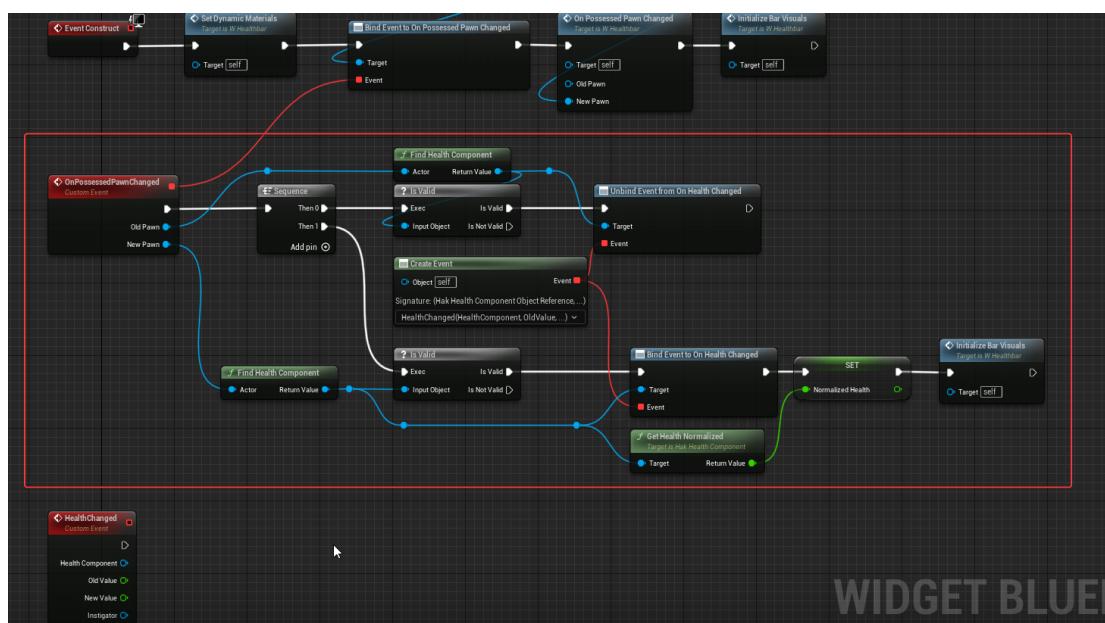
        /** 캐싱된 HealthSet 레퍼런스 */
        UPROPERTY()
        TObjectPtr<const UHakHealthSet> HealthSet;

        /** health 변화에 따른 Delegate(Multicast) */
        UPROPERTY(BlueprintAssignable)
        FHakHealth_AttributeChanged OnHealthChanged;
    };

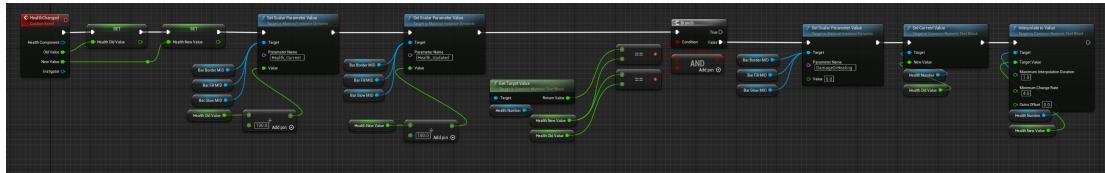

```

1 BlueprintAssignable은 BP에서도 Delegate 할당 (등록) 가능한다는 의미!

□ OnPossessedPawnChanged:



□ HealthChanged:



- 아래와 같이 잘 작동함을 볼 수 있다:

https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/e38171a3-b8a6-4bc8-a603-46c3972657fb/UnrealEditor_C3rwp3HnAV.mp4



참고로, Lyra와 같이 약간 UI가 Animating되는 효과는 없다.
이유는 UI Animation을 넣지 않았기 때문이다:

