17주차 (2024.02.25)

The Goal:

▼ **Details**

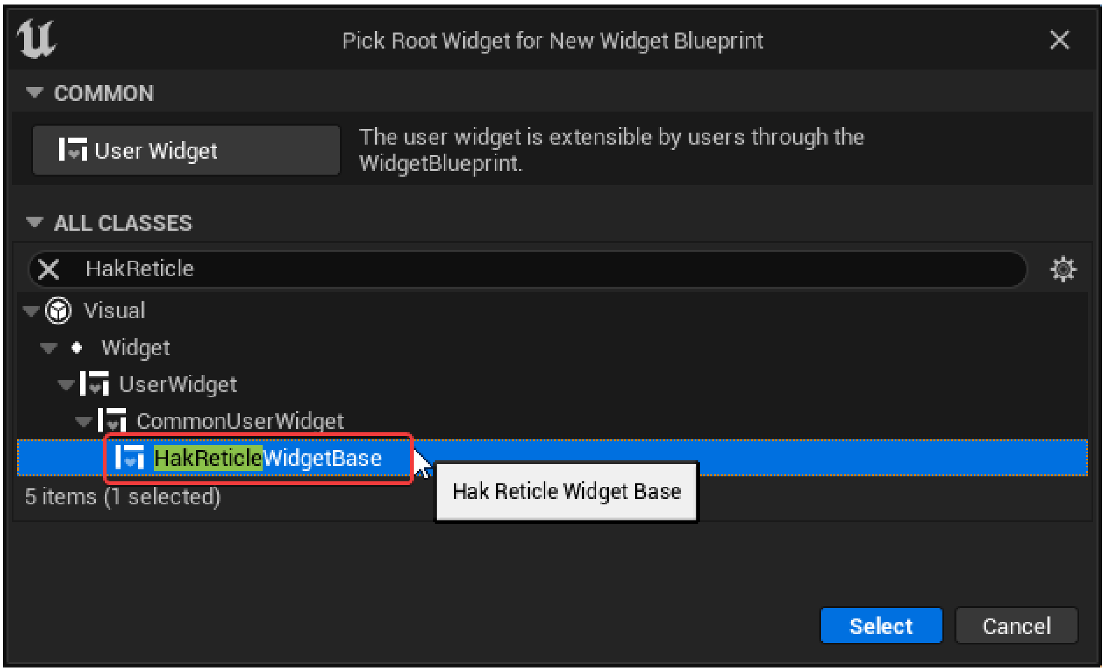
- The video for today's goal:

https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/b405c3e5-cd25-43fe-a4fc-27ace917440e/UnrealEditor_vV6mNn0arB.mp4

W_AmmoCounter_Pistol:

▼ **Details**

□ Create W_AmmoCounter_Pistol BP:



Pick Root Widget for New Widget Blueprint

COMMON

User Widget — The user widget is extensible by users through the WidgetBlueprint.

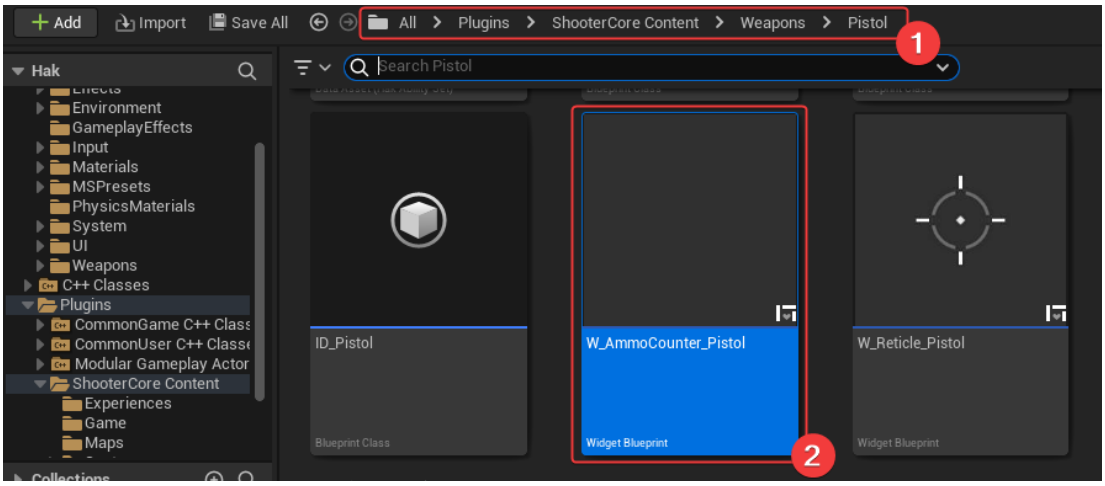
ALL CLASSES

HakReticle

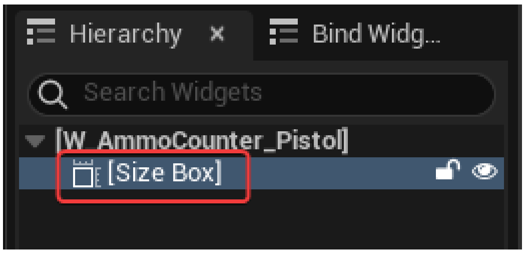
```
Visual
  Widget
    UserWidget
      CommonUserWidget
        HakReticleWidgetBase
```

5 items (1 selected)

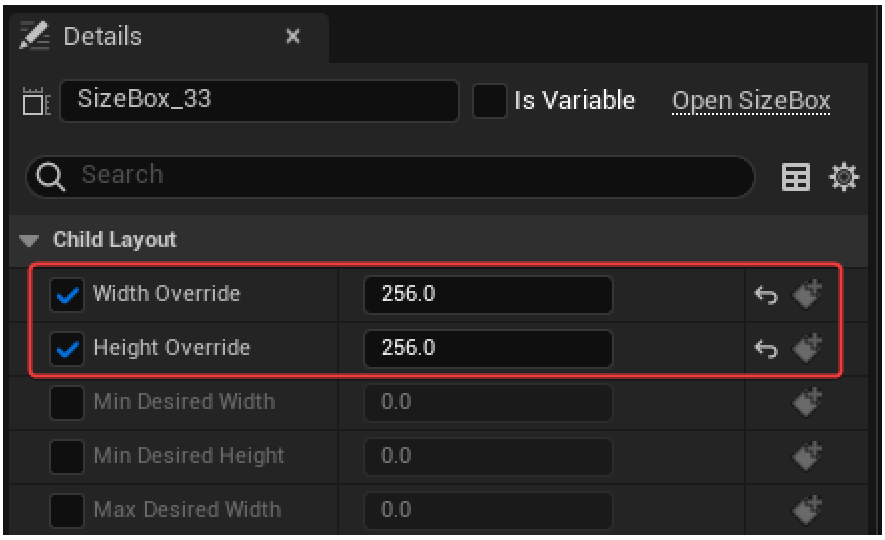
Hak Reticle Widget Base



- □ Make the layout for W_AmmoCounter_Pistol:
  - □ Add SizeBox:

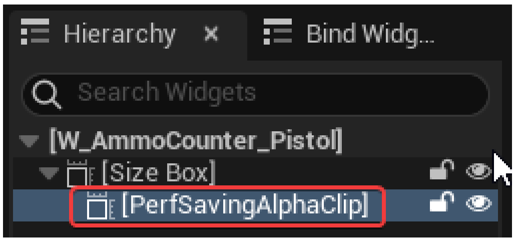


- Override SizeBox's properties:

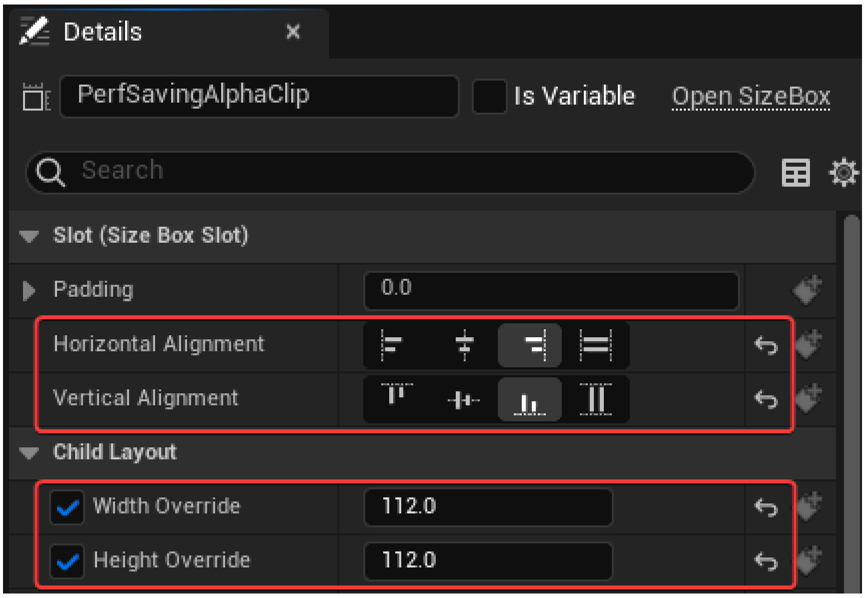


☑ Width Override	256.0
☑ Height Override	256.0
□ Min Desired Width	0.0
□ Min Desired Height	0.0
□ Max Desired Width	0.0

  - □ Add PerfSavingAlpha with SizeBox:



- Override properties:



PerfSavingAlphaClip

☐ Is Variable

Open SizeBox

Slot (Size Box Slot)

Padding 0.0

Horizontal Alignment

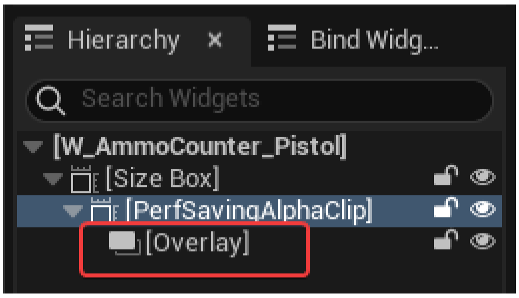
Vertical Alignment

Child Layout

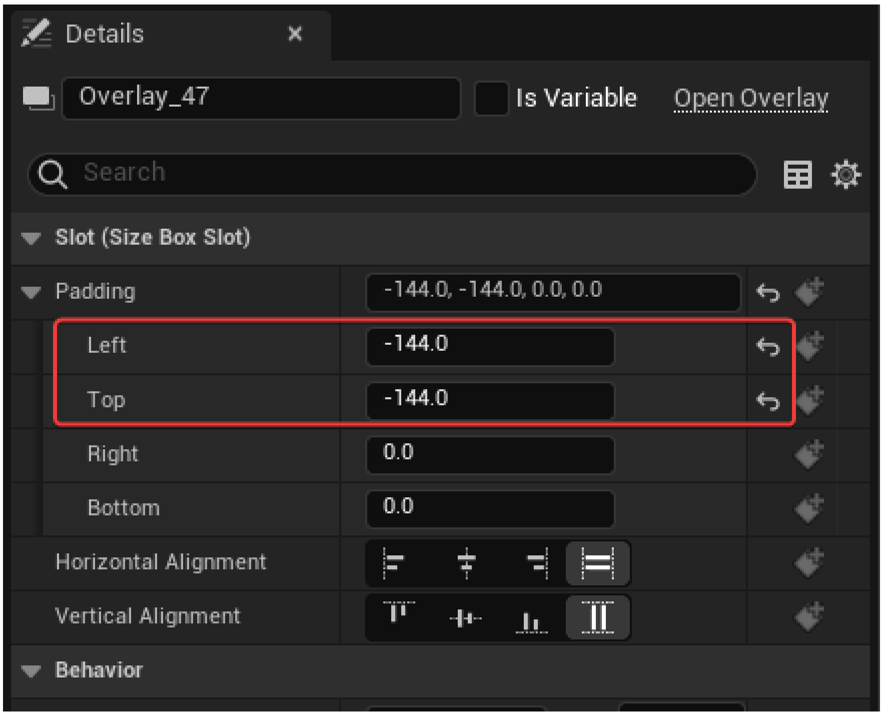
☑ Width Override 112.0

☑ Height Override 112.0

☐ Add Overlay:



- Override properties:



Details

Overlay_47 □ Is Variable Open Overlay

Search

Slot (Size Box Slot)

Padding -144.0, -144.0, 0.0, 0.0

Left -144.0

Top -144.0

Right 0.0

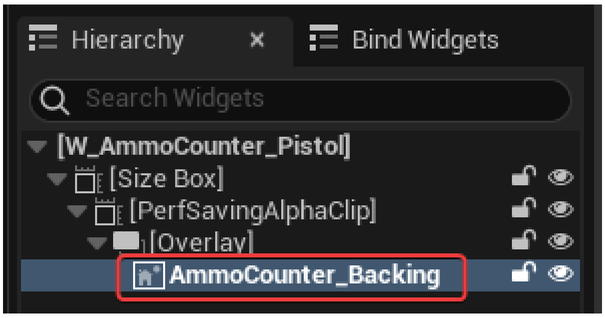
Bottom 0.0

Horizontal Alignment

Vertical Alignment

Behavior

□ Add AmmoCounter_Backing:



Hierarchy

Bind Widgets

Search Widgets

[W_AmmoCounter_Pistol]

[Size Box]

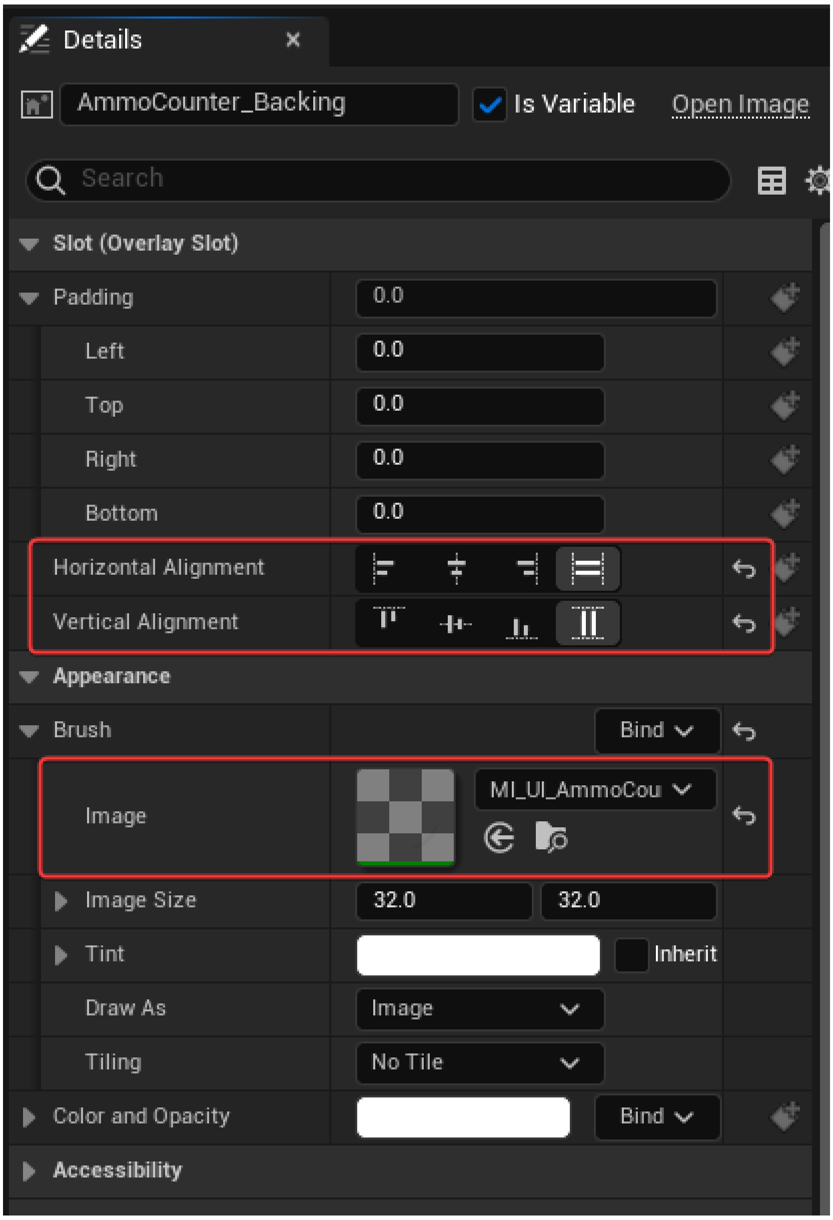
[PerfSavingAlphaClip]

[Overlay]

AmmoCounter_Backing

□ Migrate `MI_UI_AmmoCounter_Pistol_Packing`

□ Override properties:



Details

AmmoCounter_Backing ☑ Is Variable Open Image

Search

Slot (Overlay Slot)

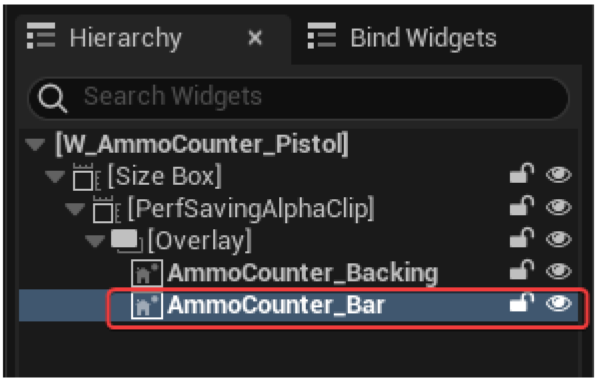
Padding	0.0
Left	0.0
Top	0.0
Right	0.0
Bottom	0.0
Horizontal Alignment	
Vertical Alignment	

Appearance

Brush	Bind
Image	MI_UI_AmmoCou
Image Size	32.0 32.0
Tint	Inherit
Draw As	Image
Tiling	No Tile
Color and Opacity	Bind

Accessibility

□ Add AmmoCounter_Bar with Image:



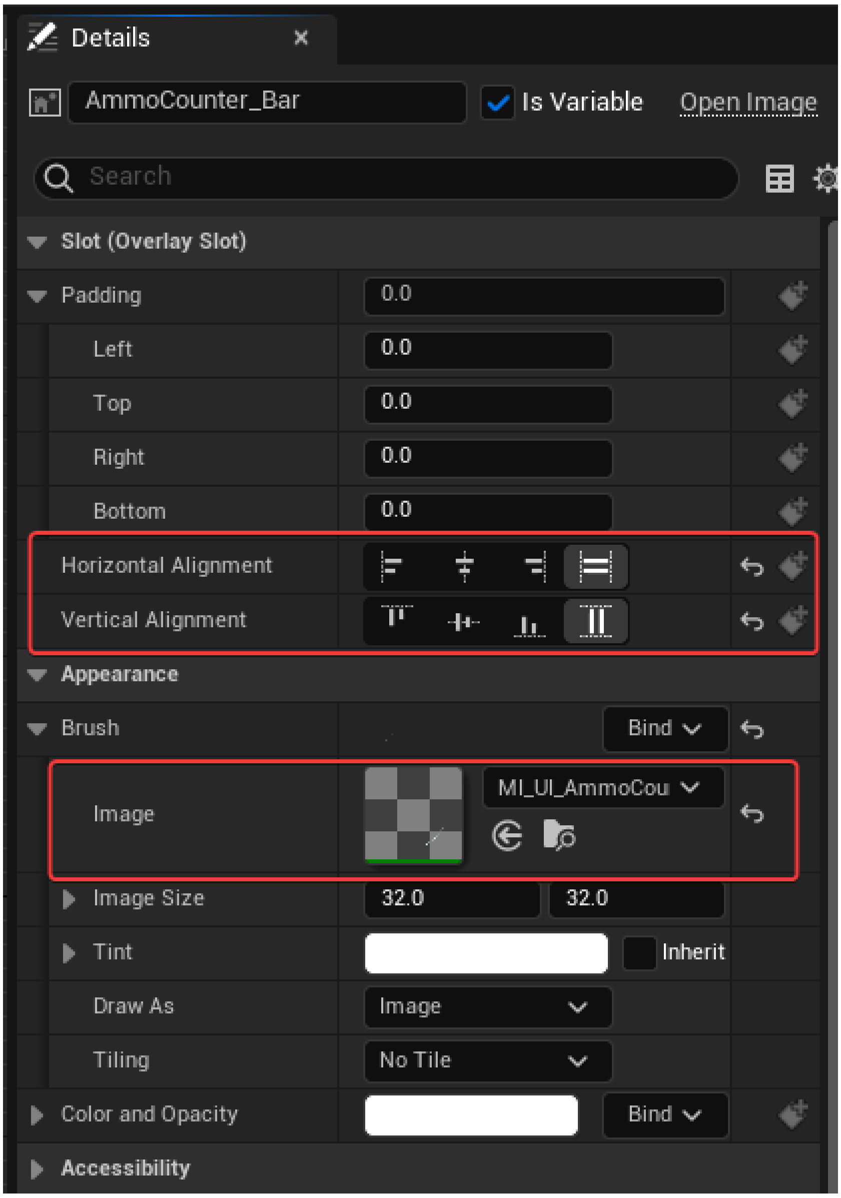
Hierarchy | Bind Widgets

Search Widgets

- [W_AmmoCounter_Pistol]
  - [Size Box]
    - [PerfSavingAlphaClip]
      - [Overlay]
        - AmmoCounter_Backing
        - AmmoCounter_Bar

□ Migrate `MI_UI_AmmoCounter_Pistol`

□ Override properties:



Details

AmmoCounter_Bar ☑ Is Variable Open Image

Search

Slot (Overlay Slot)

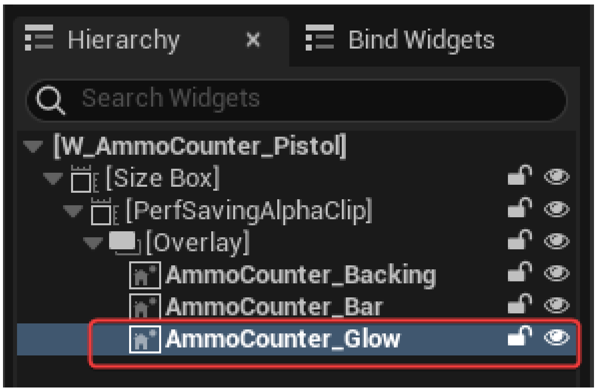
Padding	0.0
Left	0.0
Top	0.0
Right	0.0
Bottom	0.0
Horizontal Alignment	
Vertical Alignment	

Appearance

Brush		Bind
Image	MI_UI_AmmoCou	
Image Size	32.0	32.0
Tint		Inherit
Draw As	Image	
Tiling	No Tile	
Color and Opacity		Bind

Accessibility

□ Add AmmoCounter_Glow with Image:



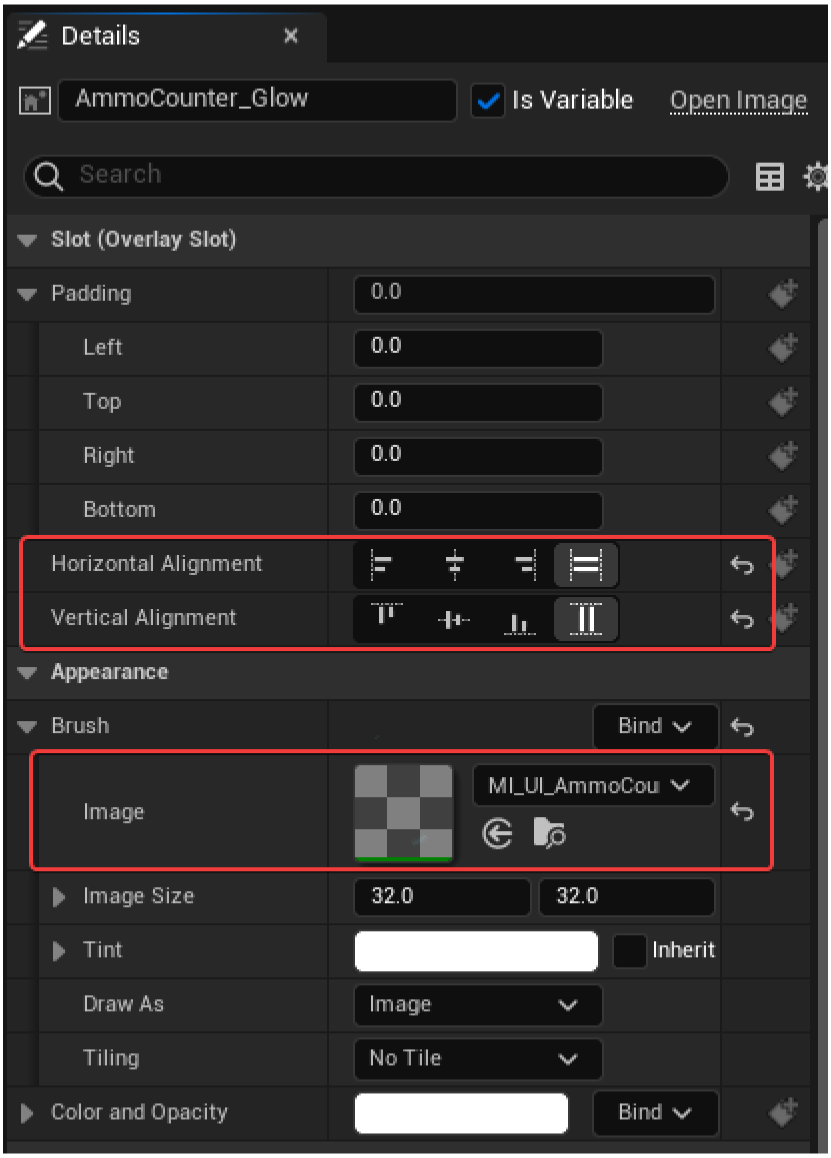
Hierarchy Bind Widgets

Search Widgets

- [W_AmmoCounter_Pistol]
  - [Size Box]
    - [PerfSavingAlphaClip]
      - [Overlay]
        - AmmoCounter_Backing
        - AmmoCounter_Bar
        - AmmoCounter_Glow

□ Migrate `MI_UI_AmmoCounter_Pistol_Glow`

□ Override properties:



Details

AmmoCounter_Glow ☑ Is Variable Open Image

Search

Slot (Overlay Slot)

Padding 0.0

Left 0.0

Top 0.0

Right 0.0

Bottom 0.0

Horizontal Alignment

Vertical Alignment

Appearance

Brush Bind

Image MI_UI_AmmoCou

Image Size 32.0 32.0

Tint ☐ Inherit

Draw As Image

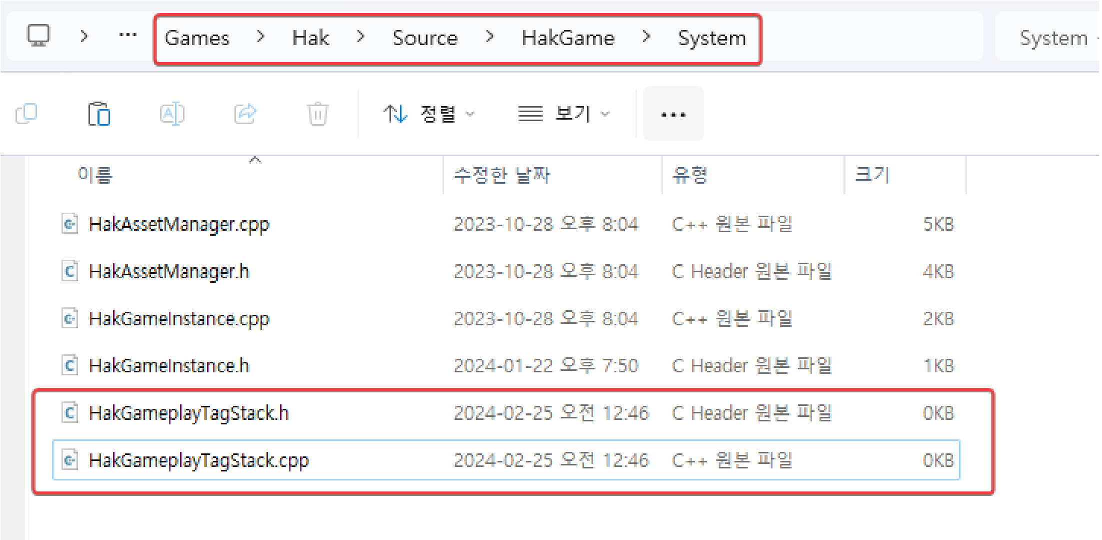
Tiling No Tile

Color and Opacity Bind

HakGameplayTagStack:

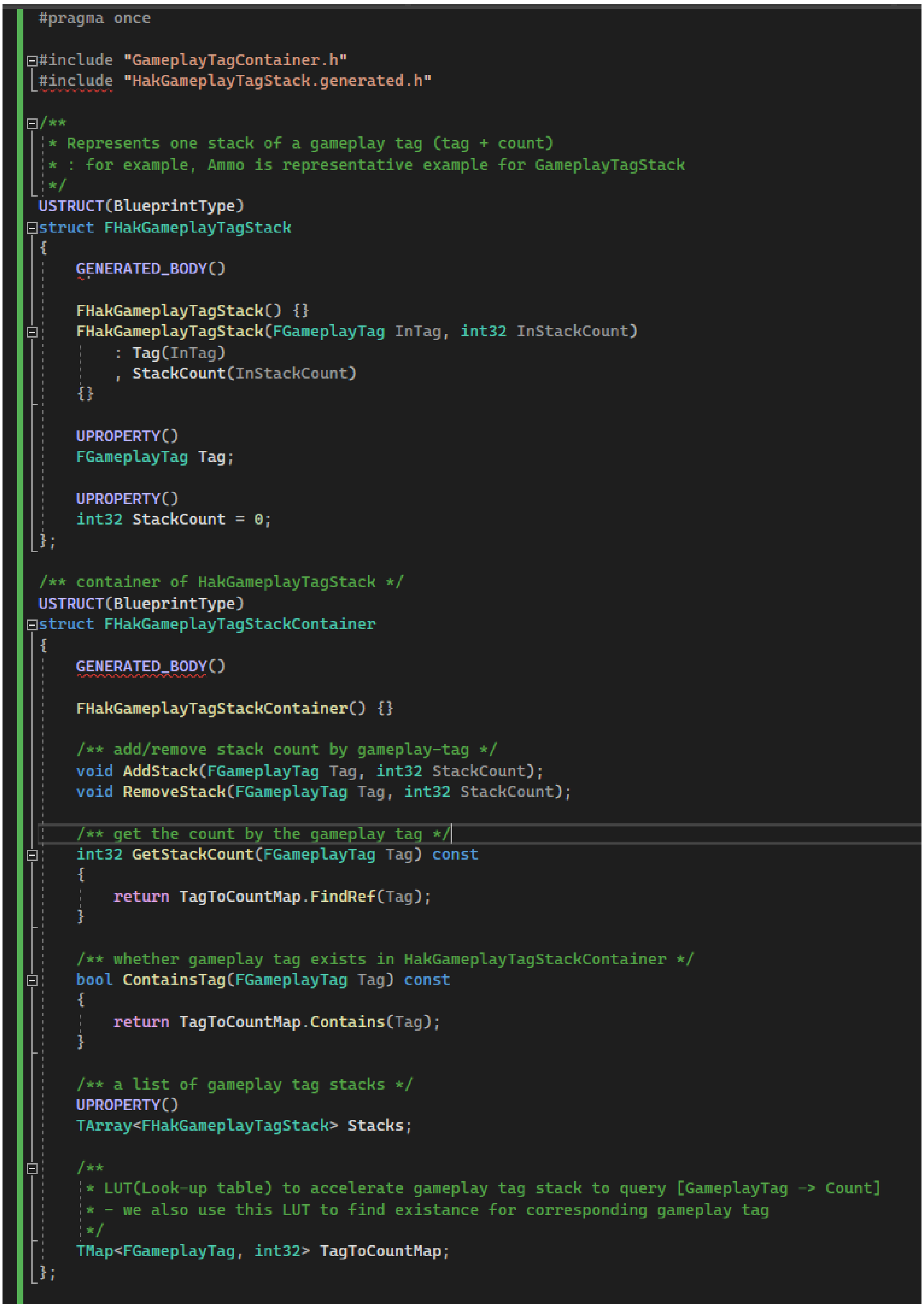
▼ **Details**

☐ Add HakGameplayTagStack.h/.cpp files like below:

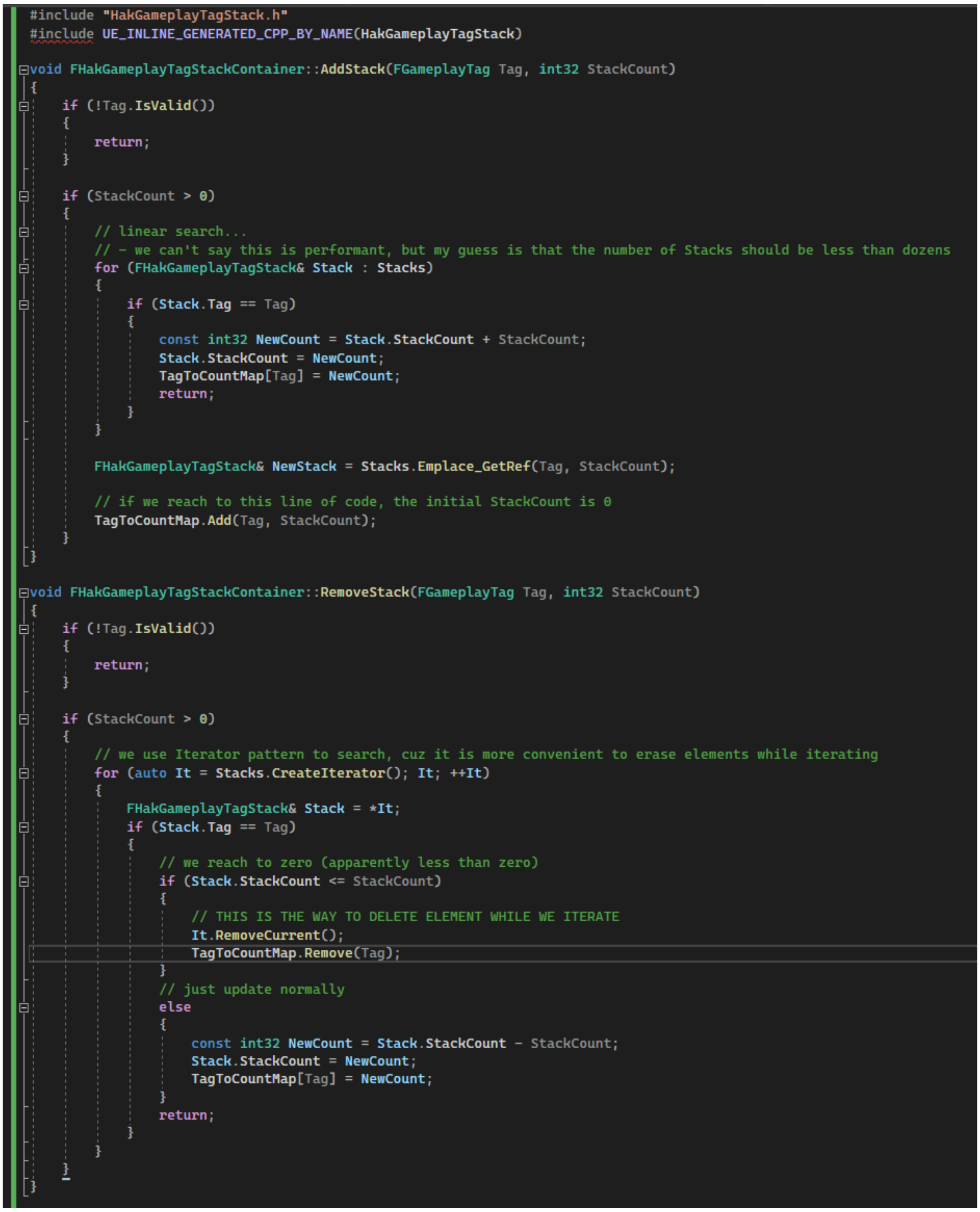


이름	수정한 날짜	유형	크기
HakAssetManager.cpp	2023-10-28 오후 8:04	C++ 원본 파일	5KB
HakAssetManager.h	2023-10-28 오후 8:04	C Header 원본 파일	4KB
HakGameInstance.cpp	2023-10-28 오후 8:04	C++ 원본 파일	2KB
HakGameInstance.h	2024-01-22 오후 7:50	C Header 원본 파일	1KB
HakGameplayTagStack.h	2024-02-25 오전 12:46	C Header 원본 파일	0KB
HakGameplayTagStack.cpp	2024-02-25 오전 12:46	C++ 원본 파일	0KB

□ implements basic layouts for FHakGameplayTagStack and FHakGameplayTagStackContainer:

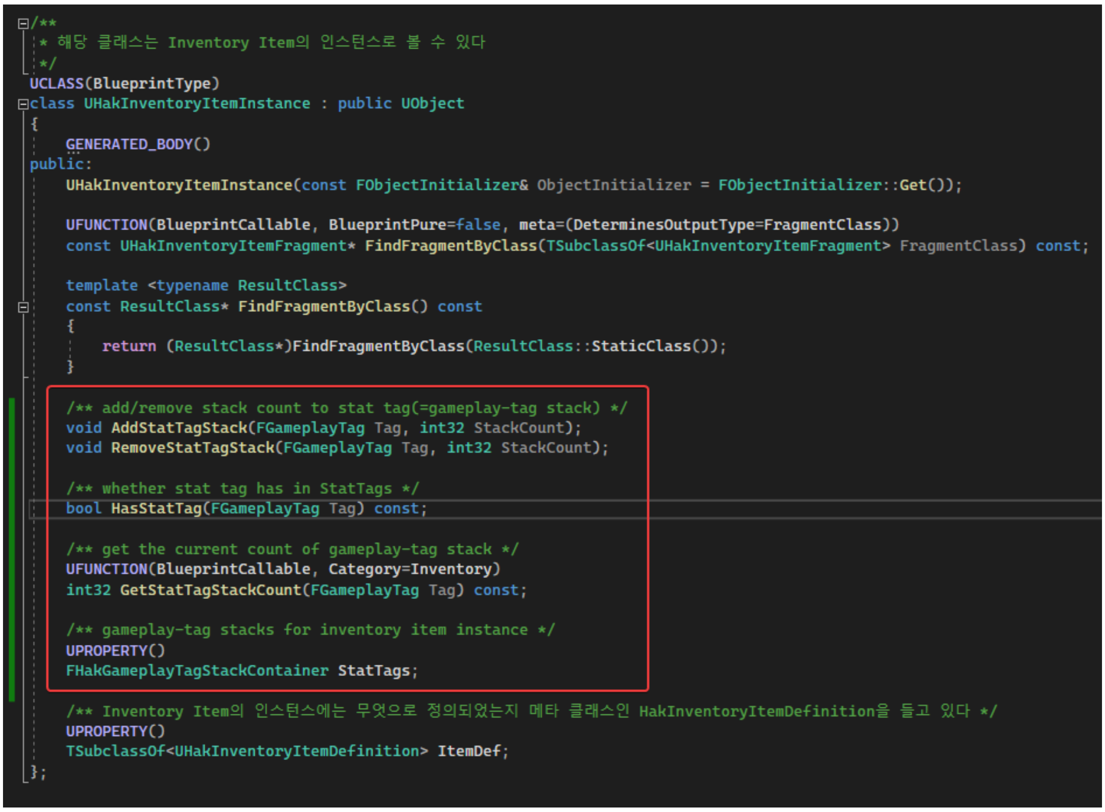


```cpp
#pragma once

#include "GameplayTagContainer.h"
#include "HakGameplayTagStack.generated.h"

/**
 * Represents one stack of a gameplay tag (tag + count)
 * : for example, Ammo is representative example for GameplayTagStack
 */
USTRUCT(BlueprintType)
struct FHakGameplayTagStack
{
    GENERATED_BODY()

    FHakGameplayTagStack() {}
    FHakGameplayTagStack(FGameplayTag InTag, int32 InStackCount)
        : Tag(InTag)
        , StackCount(InStackCount)
    {}

    UPROPERTY()
    FGameplayTag Tag;

    UPROPERTY()
    int32 StackCount = 0;
};

/** container of HakGameplayTagStack */
USTRUCT(BlueprintType)
struct FHakGameplayTagStackContainer
{
    GENERATED_BODY()

    FHakGameplayTagStackContainer() {}

    /** add/remove stack count by gameplay-tag */
    void AddStack(FGameplayTag Tag, int32 StackCount);
    void RemoveStack(FGameplayTag Tag, int32 StackCount);

    /** get the count by the gameplay tag */
    int32 GetStackCount(FGameplayTag Tag) const
    {
        return TagToCountMap.FindRef(Tag);
    }

    /** whether gameplay tag exists in HakGameplayTagStackContainer */
    bool ContainsTag(FGameplayTag Tag) const
    {
        return TagToCountMap.Contains(Tag);
    }

    /** a list of gameplay tag stacks */
    UPROPERTY()
    TArray<FHakGameplayTagStack> Stacks;

    /**
     * LUT(Look-up table) to accelerate gameplay tag stack to query [GameplayTag -> Count]
     * - we also use this LUT to find existance for corresponding gameplay tag
     */
    TMap<FGameplayTag, int32> TagToCountMap;
};
```

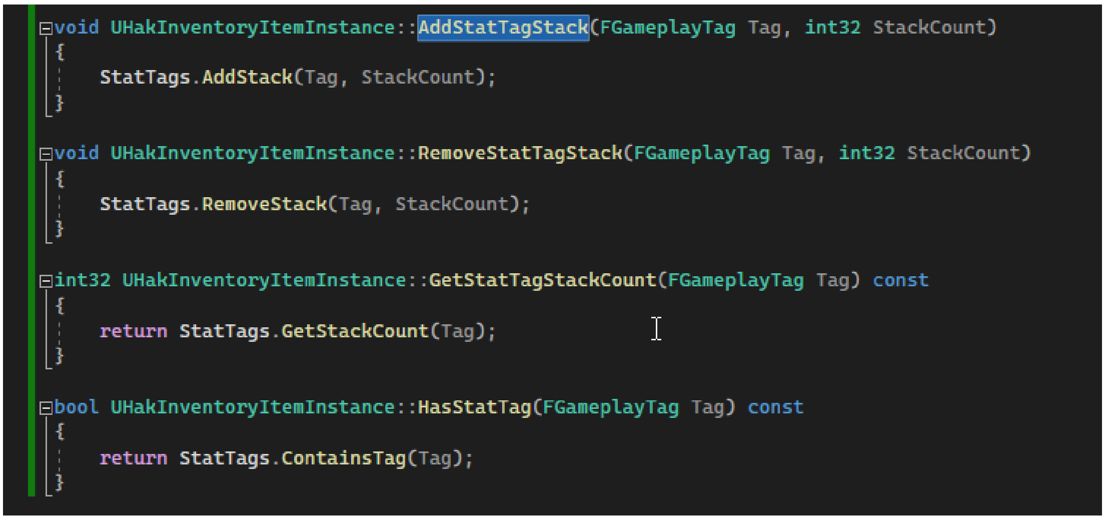


```
#include "HakGameplayTagStack.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakGameplayTagStack)

void FHakGameplayTagStackContainer::AddStack(FGameplayTag Tag, int32 StackCount)
{
    if (!Tag.IsValid())
    {
        return;
    }

    if (StackCount > 0)
    {
        // linear search...
        // - we can't say this is performant, but my guess is that the number of Stacks should be less than dozens
        for (FHakGameplayTagStack& Stack : Stacks)
        {
            if (Stack.Tag == Tag)
            {
                const int32 NewCount = Stack.StackCount + StackCount;
                Stack.StackCount = NewCount;
                TagToCountMap[Tag] = NewCount;
                return;
            }
        }

        FHakGameplayTagStack& NewStack = Stacks.Emplace_GetRef(Tag, StackCount);

        // if we reach to this line of code, the initial StackCount is 0
        TagToCountMap.Add(Tag, StackCount);
    }
}

void FHakGameplayTagStackContainer::RemoveStack(FGameplayTag Tag, int32 StackCount)
{
    if (!Tag.IsValid())
    {
        return;
    }

    if (StackCount > 0)
    {
        // we use Iterator pattern to search, cuz it is more convenient to erase elements while iterating
        for (auto It = Stacks.CreateIterator(); It; ++It)
        {
            FHakGameplayTagStack& Stack = *It;
            if (Stack.Tag == Tag)
            {
                // we reach to zero (apparently less than zero)
                if (Stack.StackCount <= StackCount)
                {
                    // THIS IS THE WAY TO DELETE ELEMENT WHILE WE ITERATE
                    It.RemoveCurrent();
                    TagToCountMap.Remove(Tag);
                }
                // just update normally
                else
                {
                    const int32 NewCount = Stack.StackCount - StackCount;
                    Stack.StackCount = NewCount;
                    TagToCountMap[Tag] = NewCount;
                }
                return;
            }
        }
    }
}
```

□ Add FHakGameplayTagStackContainer to HakInventoryItemInstance:



```
/**
 * 해당 클래스는 Inventory Item의 인스턴스로 볼 수 있다
 */
UCLASS(BlueprintType)
class UHakInventoryItemInstance : public UObject
{
	GENERATED_BODY()
public:
	UHakInventoryItemInstance(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

	UFUNCTION(BlueprintCallable, BlueprintPure=false, meta=(DeterminesOutputType=FragmentClass))
	const UHakInventoryItemFragment* FindFragmentByClass(TSubclassOf<UHakInventoryItemFragment> FragmentClass) const;

	template <typename ResultClass>
	const ResultClass* FindFragmentByClass() const
	{
		return (ResultClass*)FindFragmentByClass(ResultClass::StaticClass());
	}

	/** add/remove stack count to stat tag(=gameplay-tag stack) */
	void AddStatTagStack(FGameplayTag Tag, int32 StackCount);
	void RemoveStatTagStack(FGameplayTag Tag, int32 StackCount);

	/** whether stat tag has in StatTags */
	bool HasStatTag(FGameplayTag Tag) const;

	/** get the current count of gameplay-tag stack */
	UFUNCTION(BlueprintCallable, Category=Inventory)
	int32 GetStatTagStackCount(FGameplayTag Tag) const;

	/** gameplay-tag stacks for inventory item instance */
	UPROPERTY()
	FHakGameplayTagStackContainer StatTags;

	/** Inventory Item의 인스턴스에는 무엇으로 정의되었는지 메타 클래스인 HakInventoryItemDefinition을 들고 있다 */
	UPROPERTY()
	TSubclassOf<UHakInventoryItemDefinition> ItemDef;
};
```



```
void UHakInventoryItemInstance::AddStatTagStack(FGameplayTag Tag, int32 StackCount)
{
	StatTags.AddStack(Tag, StackCount);
}

void UHakInventoryItemInstance::RemoveStatTagStack(FGameplayTag Tag, int32 StackCount)
{
	StatTags.RemoveStack(Tag, StackCount);
}

int32 UHakInventoryItemInstance::GetStatTagStackCount(FGameplayTag Tag) const
{
	return StatTags.GetStackCount(Tag);
}

bool UHakInventoryItemInstance::HasStatTag(FGameplayTag Tag) const
{
	return StatTags.ContainsTag(Tag);
}
```

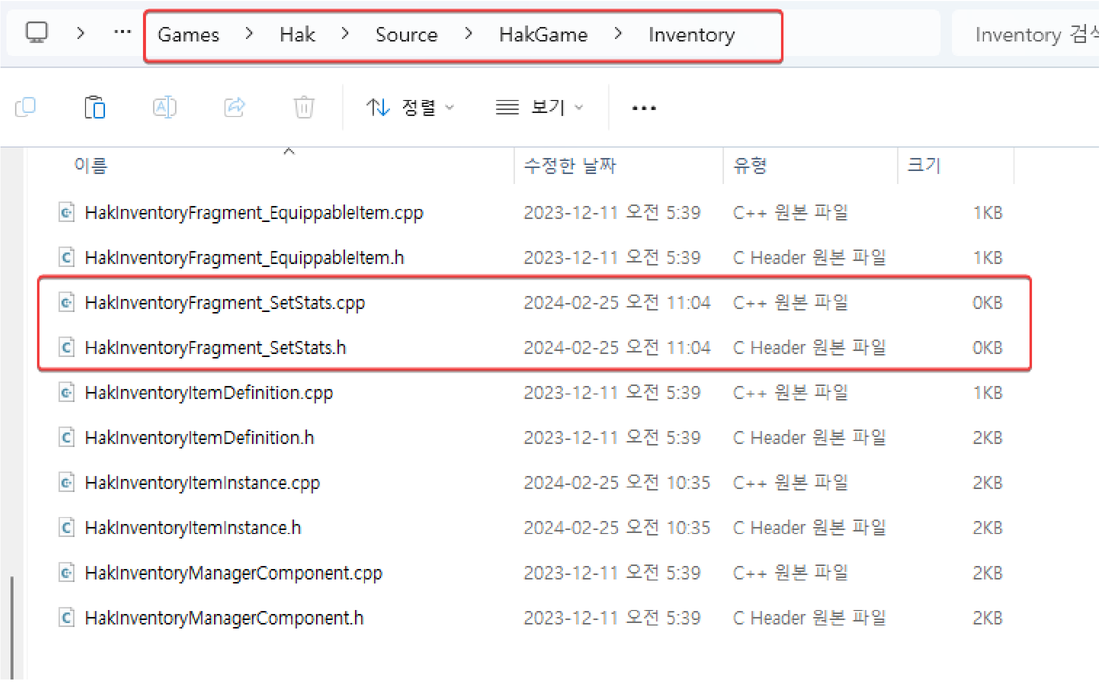
HakInventoryFragment_SetStats:

▼ **Details**

- Previously, we implement HakGameplayTagStack and we assign it to InventoryItemInstance as StatTags which describes what kind of stats are given by inventory item instance.
    - We need **to define the way of assign stat tag to inventory item instance:**

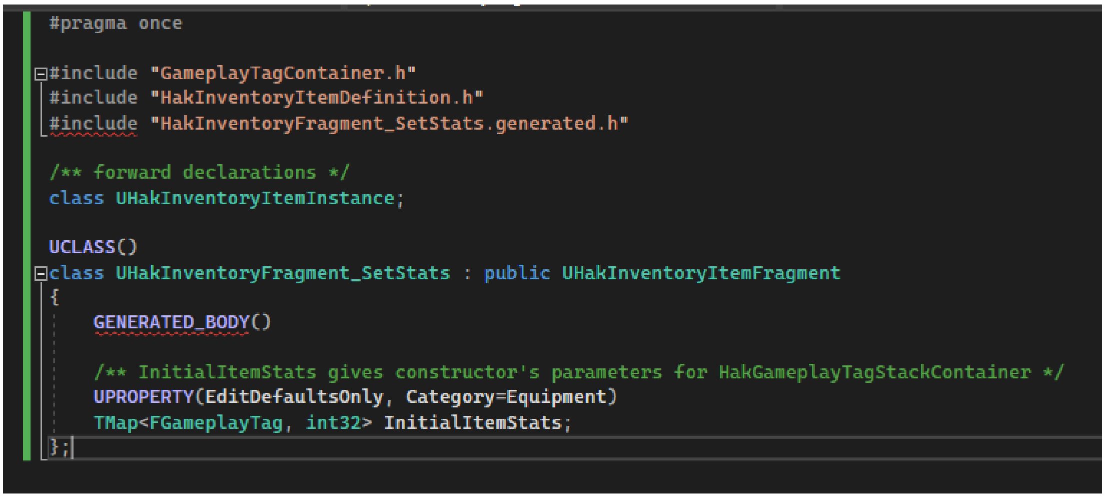
- We are going to use `HakInventoryItemFragment`

□ Add HakInventoryFragment_SetStats.h/.cpp files:

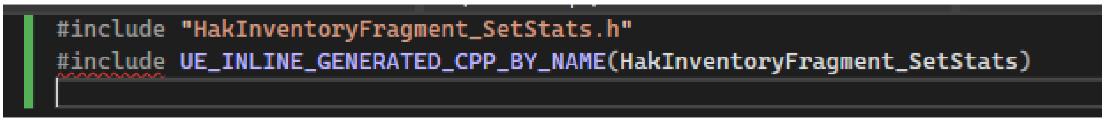


이름	수정한 날짜	유형	크기
HakInventoryFragment_EquippableItem.cpp	2023-12-11 오전 5:39	C++ 원본 파일	1KB
HakInventoryFragment_EquippableItem.h	2023-12-11 오전 5:39	C Header 원본 파일	1KB
HakInventoryFragment_SetStats.cpp	2024-02-25 오전 11:04	C++ 원본 파일	0KB
HakInventoryFragment_SetStats.h	2024-02-25 오전 11:04	C Header 원본 파일	0KB
HakInventoryItemDefinition.cpp	2023-12-11 오전 5:39	C++ 원본 파일	1KB
HakInventoryItemDefinition.h	2023-12-11 오전 5:39	C Header 원본 파일	2KB
HakInventoryItemInstance.cpp	2024-02-25 오전 10:35	C++ 원본 파일	2KB
HakInventoryItemInstance.h	2024-02-25 오전 10:35	C Header 원본 파일	2KB
HakInventoryManagerComponent.cpp	2023-12-11 오전 5:39	C++ 원본 파일	2KB
HakInventoryManagerComponent.h	2023-12-11 오전 5:39	C Header 원본 파일	2KB

□ Implement HakInventoryFragment_SetStats.h/.cpp:



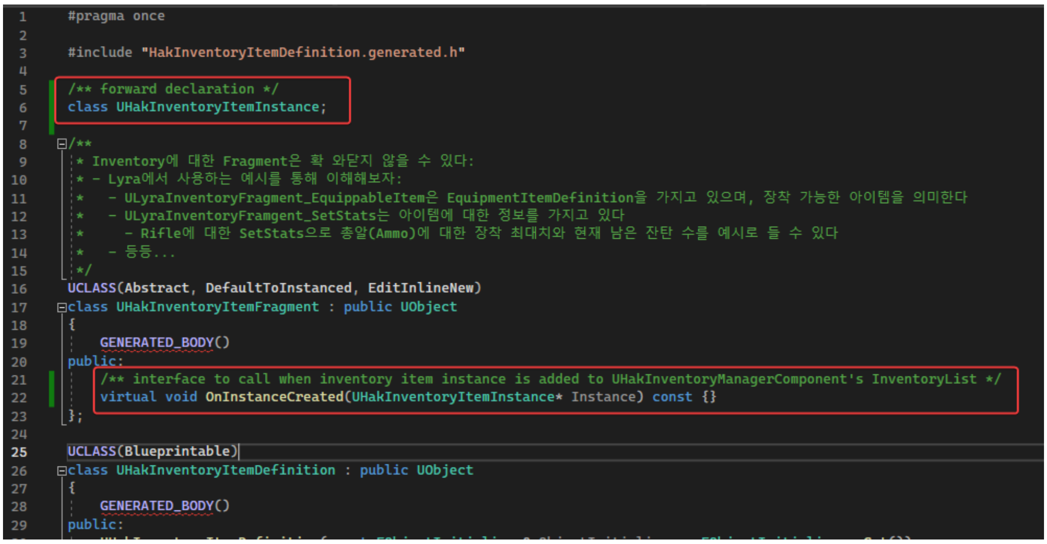
```
#pragma once

#include "GameplayTagContainer.h"
#include "HakInventoryItemDefinition.h"
#include "HakInventoryFragment_SetStats.generated.h"

/** forward declarations */
class UHakInventoryItemInstance;

UCLASS()
class UHakInventoryFragment_SetStats : public UHakInventoryItemFragment
{
    GENERATED_BODY()

    /** InitialItemStats gives constructor's parameters for HakGameplayTagStackContainer */
    UPROPERTY(EditDefaultsOnly, Category=Equipment)
    TMap<FGameplayTag, int32> InitialItemStats;
};
```



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

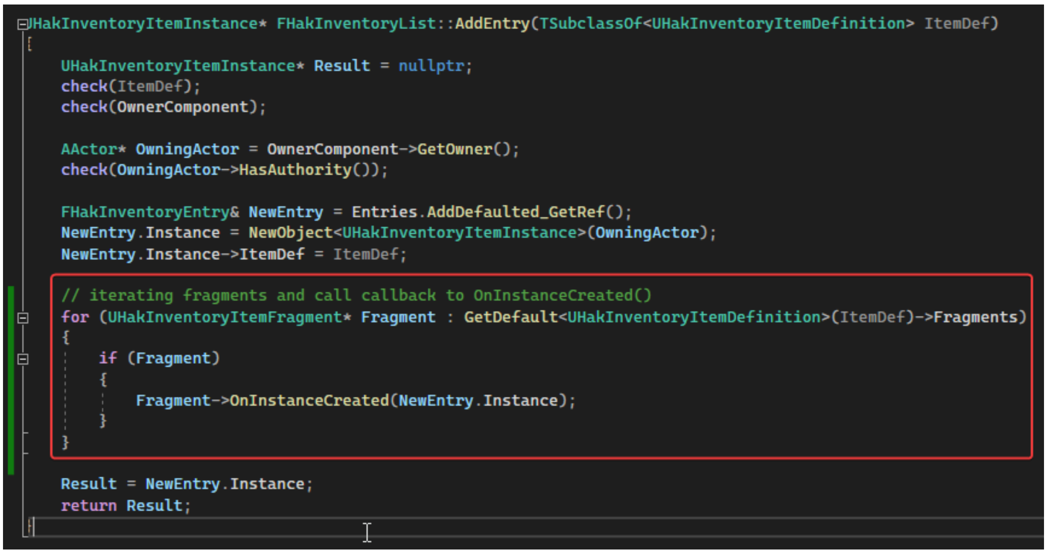
□ OnInstanceCreated():

- HakInventoryItemFragment:



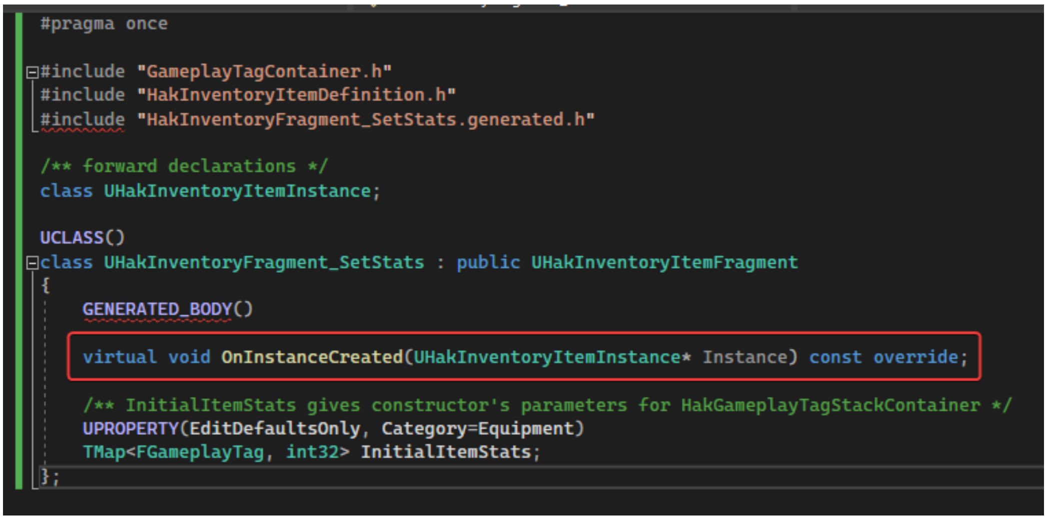
```cpp
#pragma once

#include "HakInventoryItemDefinition.generated.h"

/** forward declaration */
class UHakInventoryItemInstance;

/**
 * Inventory에 대한 Fragment은 확 와닫지 않을 수 있다:
 * - Lyra에서 사용하는 예시를 통해 이해해보자:
 *   - ULyraInventoryFragment_EquippableItem은 EquipmentItemDefinition을 가지고 있으며, 장착 가능한 아이템을 의미한다
 *   - ULyraInventoryFramgent_SetStats는 아이템에 대한 정보를 가지고 있다
 *     - Rifle에 대한 SetStats으로 총알(Ammo)에 대한 장착 최대치와 현재 남은 잔탄 수를 예시로 들 수 있다
 *   - 등등...
 */
UCLASS(Abstract, DefaultToInstanced, EditInlineNew)
class UHakInventoryItemFragment : public UObject
{
	GENERATED_BODY()
public:
	/** interface to call when inventory item instance is added to UHakInventoryManagerComponent's InventoryList */
	virtual void OnInstanceCreated(UHakInventoryItemInstance* Instance) const {}
};

UCLASS(Blueprintable)
class UHakInventoryItemDefinition : public UObject
{
	GENERATED_BODY()
public:
```

- HakInventoryManagerComponent:

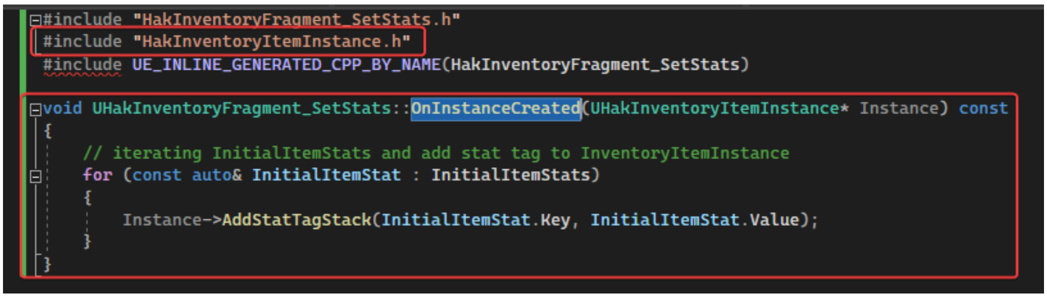


```cpp
UHakInventoryItemInstance* FHakInventoryList::AddEntry(TSubclassOf<UHakInventoryItemDefinition> ItemDef)
{
	UHakInventoryItemInstance* Result = nullptr;
	check(ItemDef);
	check(OwnerComponent);

	AActor* OwningActor = OwnerComponent->GetOwner();
	check(OwningActor->HasAuthority());

	FHakInventoryEntry& NewEntry = Entries.AddDefaulted_GetRef();
	NewEntry.Instance = NewObject<UHakInventoryItemInstance>(OwningActor);
	NewEntry.Instance->ItemDef = ItemDef;

	// iterating fragments and call callback to OnInstanceCreated()
	for (UHakInventoryItemFragment* Fragment : GetDefault<UHakInventoryItemDefinition>(ItemDef)->Fragments)
	{
		if (Fragment)
		{
			Fragment->OnInstanceCreated(NewEntry.Instance);
		}
	}

	Result = NewEntry.Instance;
	return Result;
}
```

- Now, override OnInstanceCreated() from HakInventoryItemFragment_SetStats:

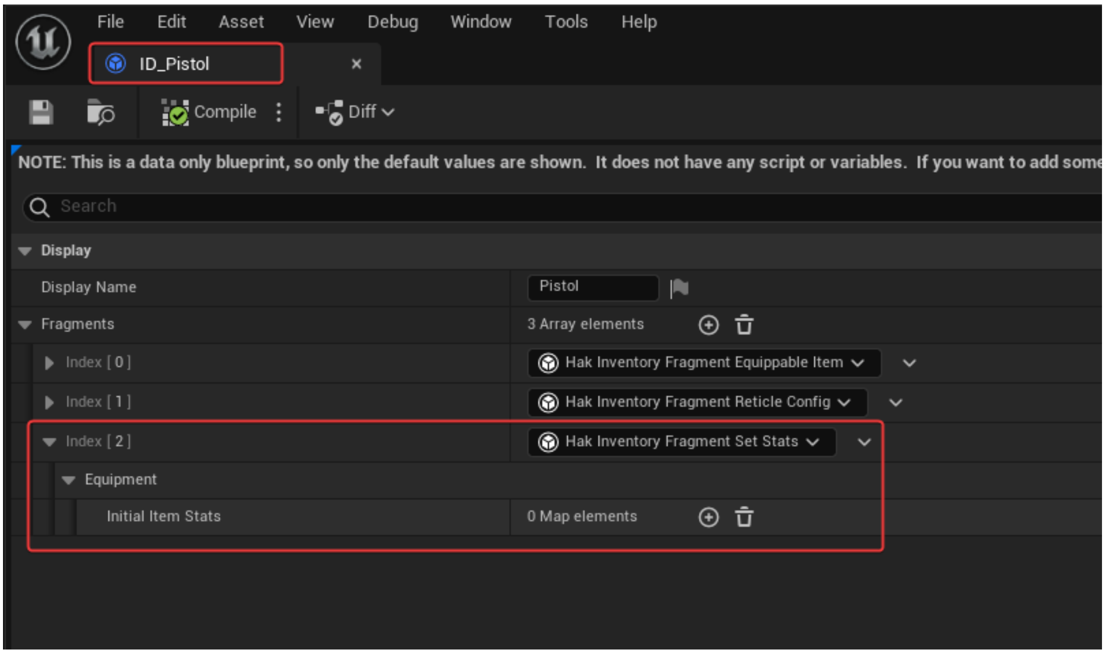


```
#pragma once

#include "GameplayTagContainer.h"
#include "HakInventoryItemDefinition.h"
#include "HakInventoryFragment_SetStats.generated.h"

/** forward declarations */
class UHakInventoryItemInstance;

UCLASS()
class UHakInventoryFragment_SetStats : public UHakInventoryItemFragment
{
    GENERATED_BODY()

    virtual void OnInstanceCreated(UHakInventoryItemInstance* Instance) const override;

    /** InitialItemStats gives constructor's parameters for HakGameplayTagStackContainer */
    UPROPERTY(EditDefaultsOnly, Category=Equipment)
    TMap<FGameplayTag, int32> InitialItemStats;
};
```



```
#include "HakInventoryFragment_SetStats.h"
#include "HakInventoryItemInstance.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakInventoryFragment_SetStats)

void UHakInventoryFragment_SetStats::OnInstanceCreated(UHakInventoryItemInstance* Instance) const
{
    // iterating InitialItemStats and add stat tag to InventoryItemInstance
    for (const auto& InitialItemStat : InitialItemStats)
    {
        Instance->AddStatTagStack(InitialItemStat.Key, InitialItemStat.Value);
    }
}
```

□ Add HakInventoryFragment_SetStats to ID_Pistol:

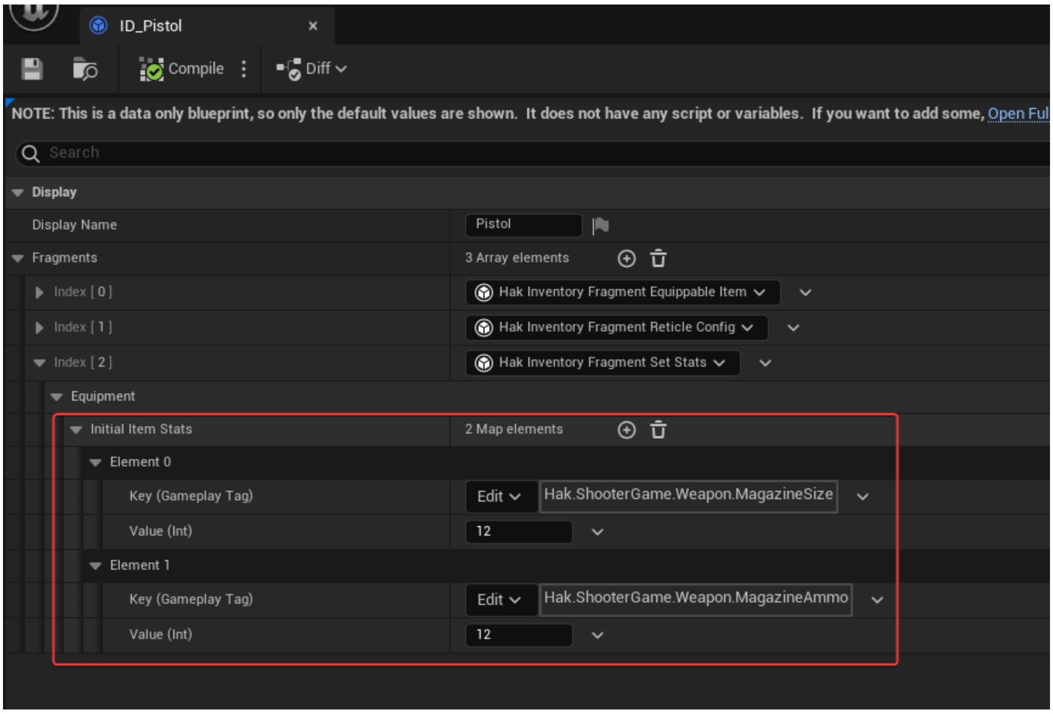


NOTE: This is a data only blueprint, so only the default values are shown. It does not have any script or variables. If you want to add some

Display
Display Name: Pistol
Fragments: 3 Array elements
Index [0]: Hak Inventory Fragment Equippable Item
Index [1]: Hak Inventory Fragment Reticle Config
Index [2]: Hak Inventory Fragment Set Stats
Equipment
Initial Item Stats: 0 Map elements

- ID_Pistol inherits from HakInventoryItemDefinition

□ Add Two StatTags:



NOTE: This is a data only blueprint, so only the default values are shown. It does not have any script or variables. If you want to add some, Open Ful

Search

Display

Display Name: Pistol

Fragments: 3 Array elements

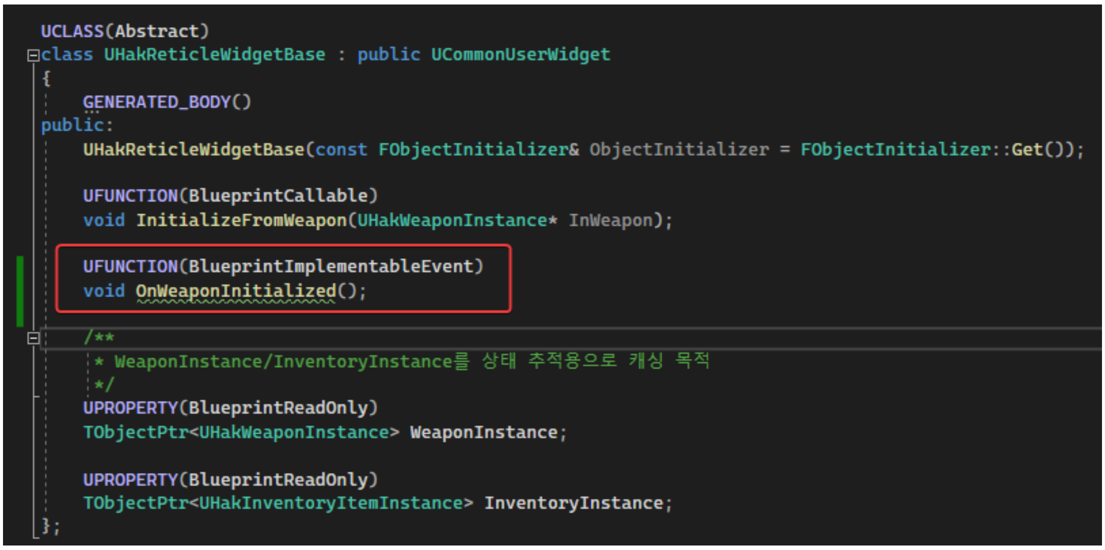
- Index [0]: Hak Inventory Fragment Equippable Item
- Index [1]: Hak Inventory Fragment Reticle Config
- Index [2]: Hak Inventory Fragment Set Stats
  - Equipment
    - Initial Item Stats: 2 Map elements
      - Element 0
        - Key (Gameplay Tag): Edit Hak.ShooterGame.Weapon.MagazineSize
        - Value (Int): 12
      - Element 1
        - Key (Gameplay Tag): Edit Hak.ShooterGame.Weapon.MagazineAmmo
        - Value (Int): 12

- Think of MagazineSize as **Total Ammo**
- Think of MagazineAmmo as **Current Ammo**

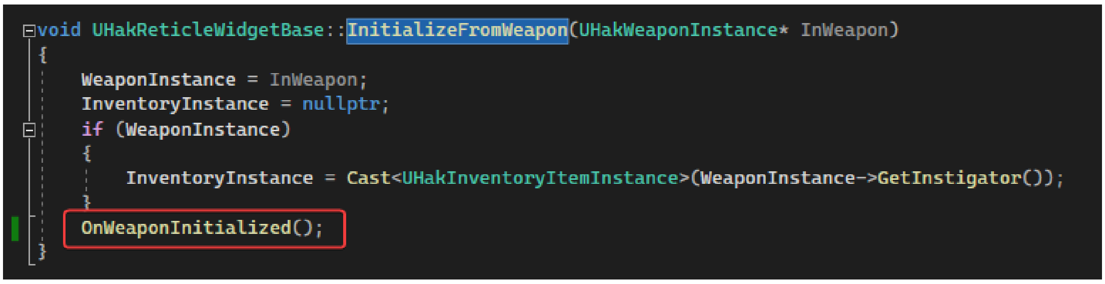
W_AmmoCounter_Pistol - 2:

▼ Details

□ Add BlueprintImplementableEvent for OnWeaponInitialized():

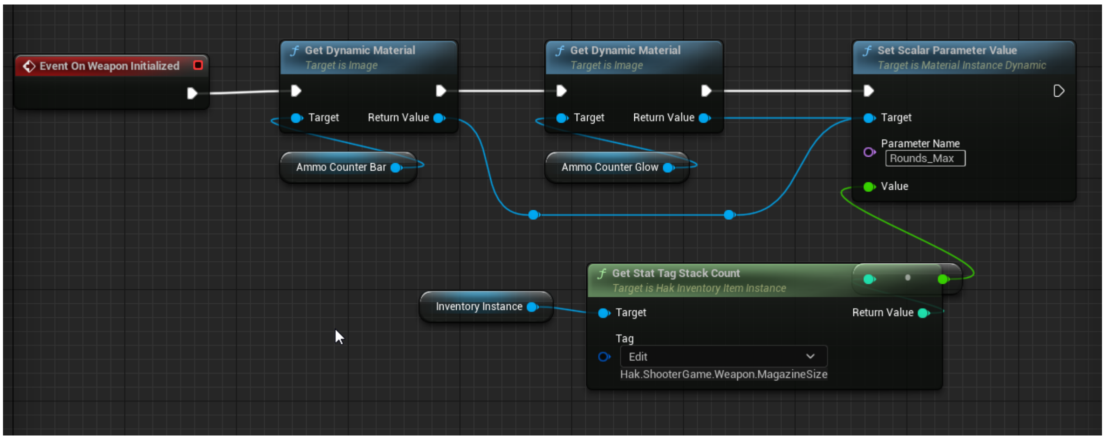


```
UCLASS(Abstract)
class UHakReticleWidgetBase : public UCommonUserWidget
{
    GENERATED_BODY()
public:
    UHakReticleWidgetBase(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

    UFUNCTION(BlueprintCallable)
    void InitializeFromWeapon(UHakWeaponInstance* InWeapon);

    UFUNCTION(BlueprintImplementableEvent)
    void OnWeaponInitialized();

    /**
     * WeaponInstance/InventoryInstance를 상태 추적용으로 캐싱 목적
     */
    UPROPERTY(BlueprintReadOnly)
    TObjectPtr<UHakWeaponInstance> WeaponInstance;

    UPROPERTY(BlueprintReadOnly)
    TObjectPtr<UHakInventoryItemInstance> InventoryInstance;
};
```

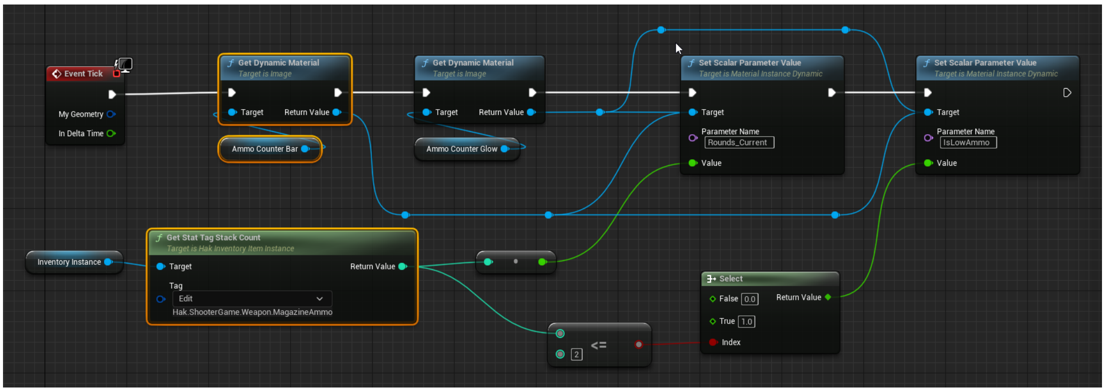


```
void UHakReticleWidgetBase::InitializeFromWeapon(UHakWeaponInstance* InWeapon)
{
    WeaponInstance = InWeapon;
    InventoryInstance = nullptr;
    if (WeaponInstance)
    {
        InventoryInstance = Cast<UHakInventoryItemInstance>(WeaponInstance->GetInstigator());
    }
    OnWeaponInitialized();
}
```

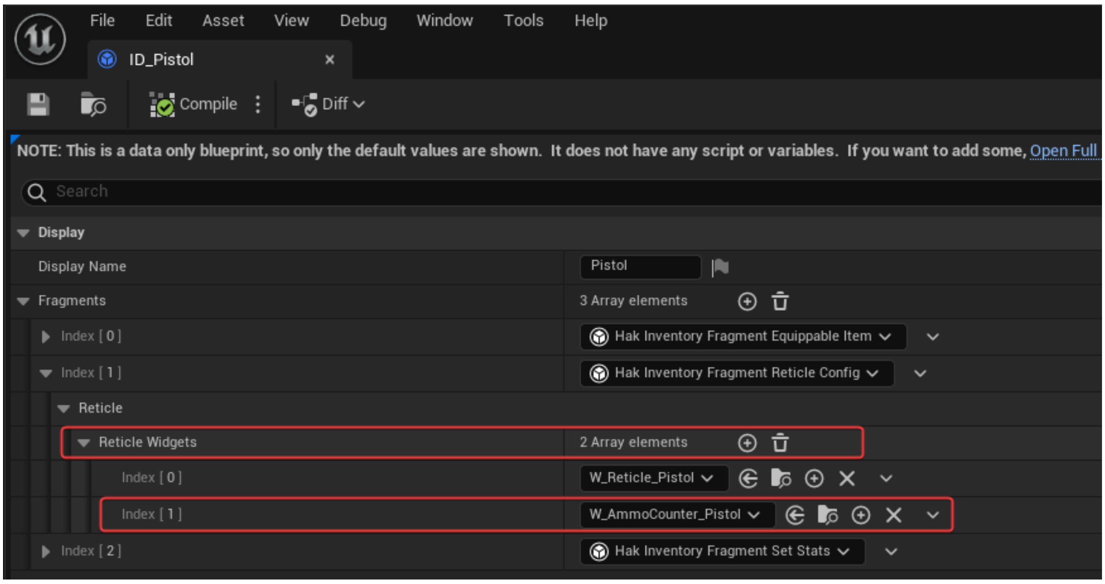
□ Implement BP Event, OnWeaponInitialized() in W_AmmoCounter_Pistol:



□ Implement Event Tick:



□ Add W_AmmoCounter_Pistol with HakInventoryFragment_ReticleConfig in ID_Pistol:



NOTE: This is a data only blueprint, so only the default values are shown. It does not have any script or variables. If you want to add some, Open Full

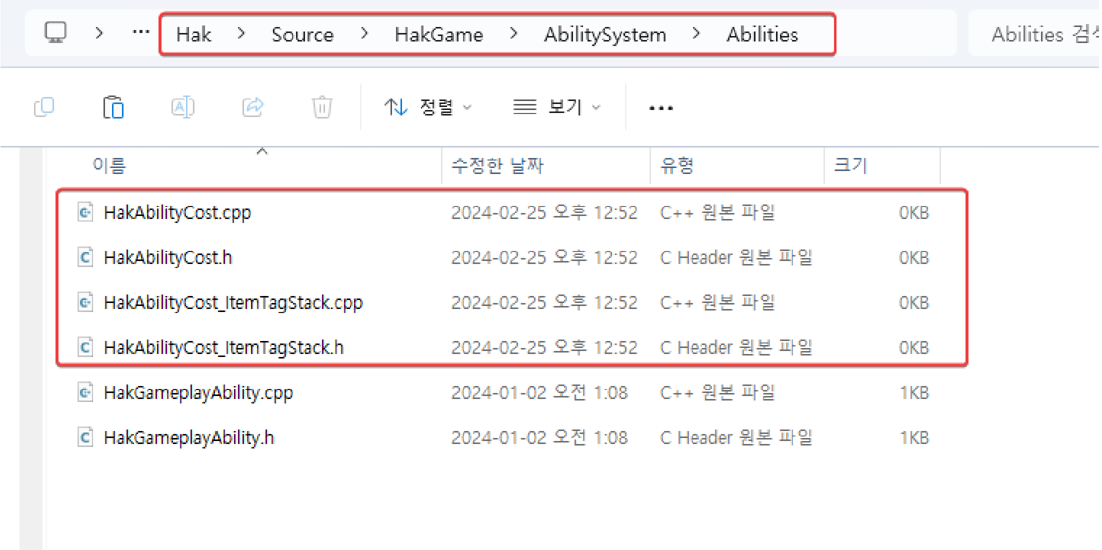
Display
Display Name: Pistol
Fragments: 3 Array elements
Index [0]: Hak Inventory Fragment Equippable Item
Index [1]: Hak Inventory Fragment Reticle Config
Reticle
Reticle Widgets: 2 Array elements
Index [0]: W_Reticle_Pistol
Index [1]: W_AmmoCounter_Pistol
Index [2]: Hak Inventory Fragment Set Stats

HakAbilityCost_ItemTag:

▼ **Details**

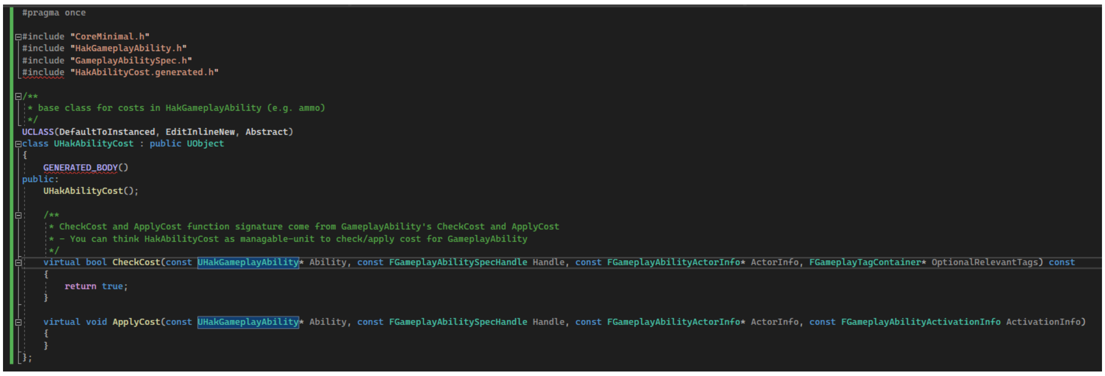
- We want to reflect ammo count to inventory item instance for pistol, and also it will apply current state of pistol's ammo to Widget, W_AmmoCounter_Pistol

□ Add files for HakAbilityCost.h/.cpp and HakAbilityCost_ItemTagStack.h/.cpp:

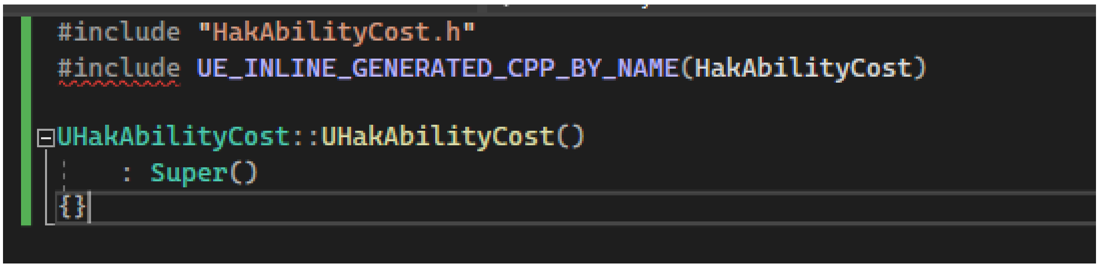


이름	수정한 날짜	유형	크기
HakAbilityCost.cpp	2024-02-25 오후 12:52	C++ 원본 파일	0KB
HakAbilityCost.h	2024-02-25 오후 12:52	C Header 원본 파일	0KB
HakAbilityCost_ItemTagStack.cpp	2024-02-25 오후 12:52	C++ 원본 파일	0KB
HakAbilityCost_ItemTagStack.h	2024-02-25 오후 12:52	C Header 원본 파일	0KB
HakGameplayAbility.cpp	2024-01-02 오전 1:08	C++ 원본 파일	1KB
HakGameplayAbility.h	2024-01-02 오전 1:08	C Header 원본 파일	1KB

□ Implement HakAbilityCost.h/.cpp

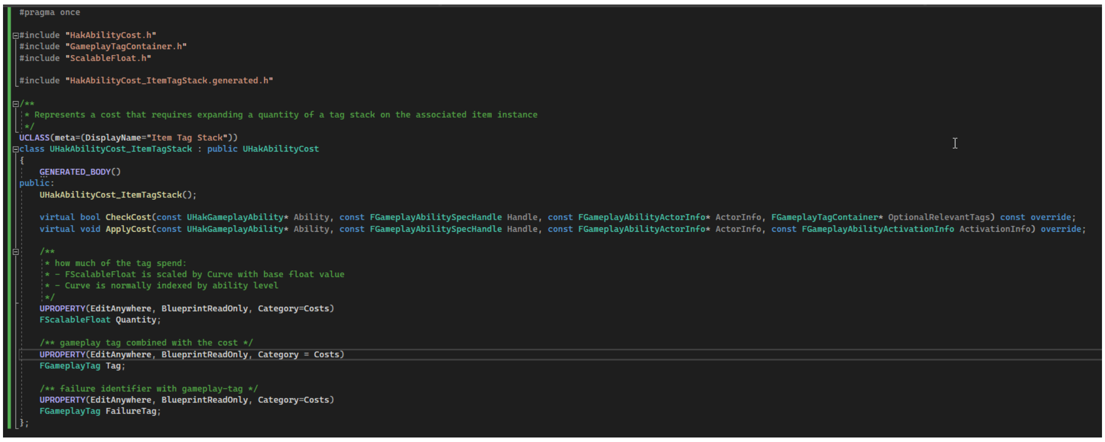


```cpp
#pragma once

#include "CoreMinimal.h"
#include "HakGameplayAbility.h"
#include "GameplayAbilitySpec.h"
#include "HakAbilityCost.generated.h"

/**
 * base class for costs in HakGameplayAbility (e.g. ammo)
 */
UCLASS(DefaultToInstanced, EditInlineNew, Abstract)
class UHakAbilityCost : public UObject
{
	GENERATED_BODY()
public:
	UHakAbilityCost();

	/**
	 * CheckCost and ApplyCost function signature come from GameplayAbility's CheckCost and ApplyCost
	 * - You can think HakAbilityCost as managable-unit to check/apply cost for GameplayAbility
	 */
	virtual bool CheckCost(const UHakGameplayAbility* Ability, const FGameplayAbilitySpecHandle Handle, const FGameplayAbilityActorInfo* ActorInfo, FGameplayTagContainer* OptionalRelevantTags) const
	{
		return true;
	}

	virtual void ApplyCost(const UHakGameplayAbility* Ability, const FGameplayAbilitySpecHandle Handle, const FGameplayAbilityActorInfo* ActorInfo, const FGameplayAbilityActivationInfo ActivationInfo)
	{
	}
};
```

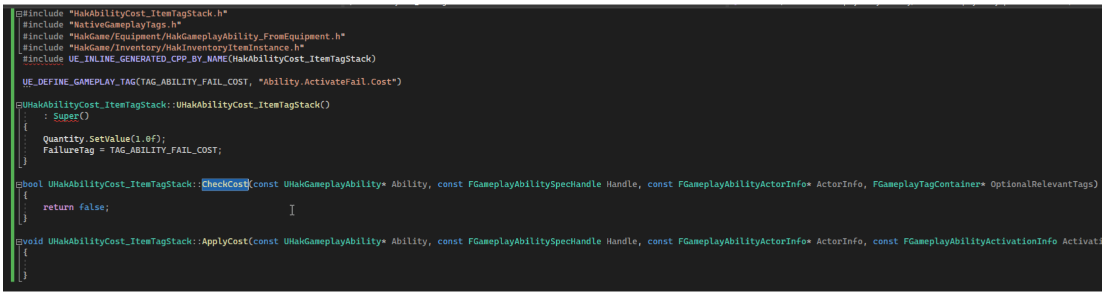


```cpp
#include "HakAbilityCost.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakAbilityCost)

UHakAbilityCost::UHakAbilityCost()
	: Super()
{}
```

□ Implement HakAbilityCost_ItemTagStack.h/.cpp:



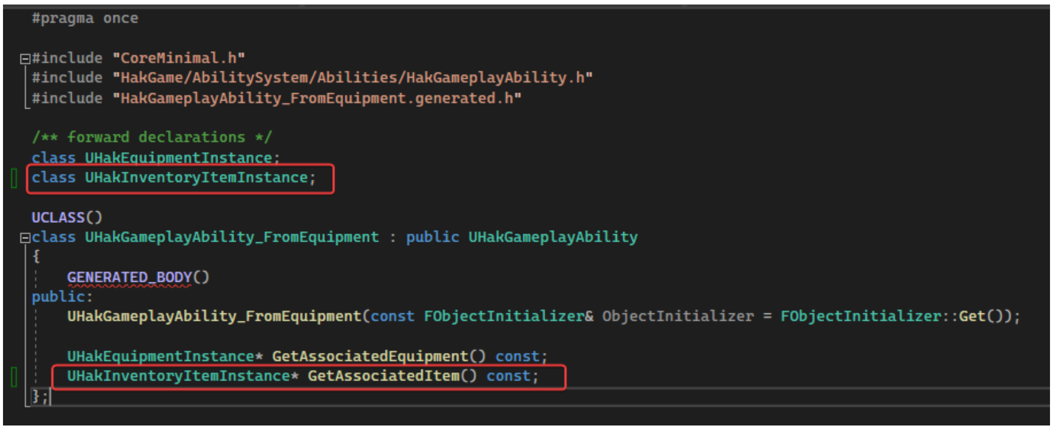
```cpp
#pragma once

#include "HakAbilityCost.h"
#include "GameplayTagContainer.h"
#include "ScalableFloat.h"

#include "HakAbilityCost_ItemTagStack.generated.h"

/**
 * Represents a cost that requires expanding a quantity of a tag stack on the associated item instance
 */
UCLASS(meta=(DisplayName="Item Tag Stack"))
class UHakAbilityCost_ItemTagStack : public UHakAbilityCost
{
	GENERATED_BODY()
public:
	UHakAbilityCost_ItemTagStack();

	virtual bool CheckCost(const UHakGameplayAbility* Ability, const FGameplayAbilitySpecHandle Handle, const FGameplayAbilityActorInfo* ActorInfo, FGameplayTagContainer* OptionalRelevantTags) const override;
	virtual void ApplyCost(const UHakGameplayAbility* Ability, const FGameplayAbilitySpecHandle Handle, const FGameplayAbilityActorInfo* ActorInfo, const FGameplayAbilityActivationInfo ActivationInfo) override;

	/**
	 * how much of the tag spend:
	 * - FScalableFloat is scaled by Curve with base float value
	 * - Curve is normally indexed by ability level
	 */
	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Costs)
	FScalableFloat Quantity;

	/** gameplay tag combined with the cost */
	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Costs)
	FGameplayTag Tag;

	/** failure identifier with gameplay-tag */
	UPROPERTY(EditAnywhere, BlueprintReadOnly, Category=Costs)
	FGameplayTag FailureTag;
};
```



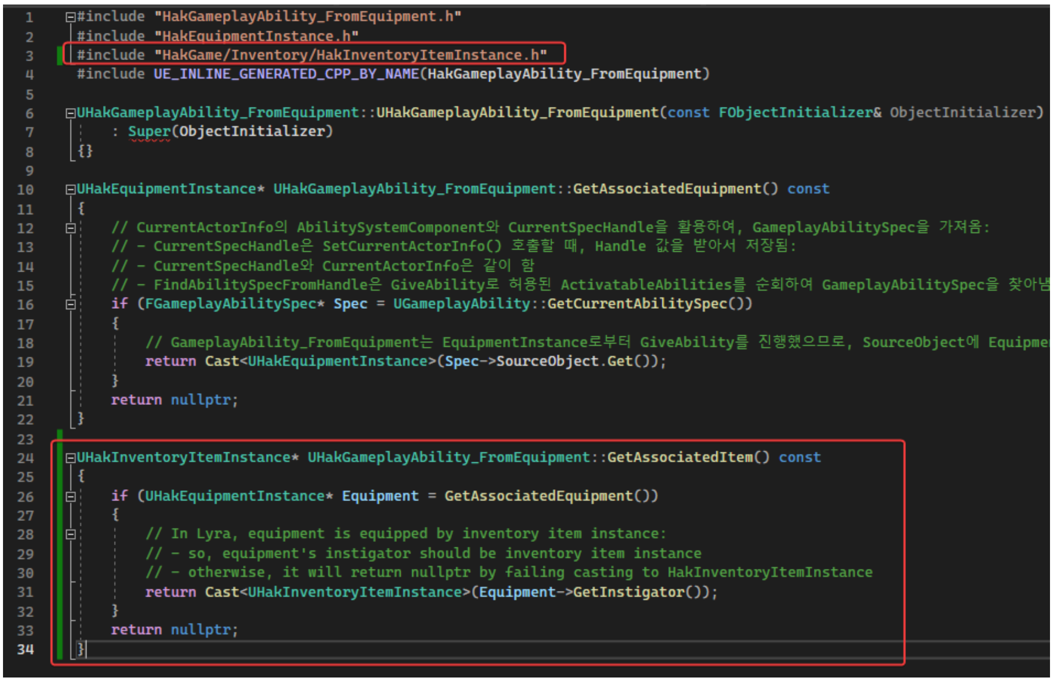
```cpp
#include "HakAbilityCost_ItemTagStack.h"
#include "NativeGameplayTags.h"
#include "HakGame/Equipment/HakGameplayAbility_FromEquipment.h"
#include "HakGame/Inventory/HakInventoryItemInstance.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakAbilityCost_ItemTagStack)

UE_DEFINE_GAMEPLAY_TAG(TAG_ABILITY_FAIL_COST, "Ability.ActivateFail.Cost")

UHakAbilityCost_ItemTagStack::UHakAbilityCost_ItemTagStack()
	: Super()
{
	Quantity.SetValue(1.0f);
	FailureTag = TAG_ABILITY_FAIL_COST;
}

bool UHakAbilityCost_ItemTagStack::CheckCost(const UHakGameplayAbility* Ability, const FGameplayAbilitySpecHandle Handle, const FGameplayAbilityActorInfo* ActorInfo, FGameplayTagContainer* OptionalRelevantTags)
{
	return false;
}

void UHakAbilityCost_ItemTagStack::ApplyCost(const UHakGameplayAbility* Ability, const FGameplayAbilitySpecHandle Handle, const FGameplayAbilityActorInfo* ActorInfo, const FGameplayAbilityActivationInfo Activati
{

}
```

□ HakAbilityCost_ItemTagStack::CheckCost:

- □ HakGameplayAbility_FromEquipment::GetAssociatedItem():

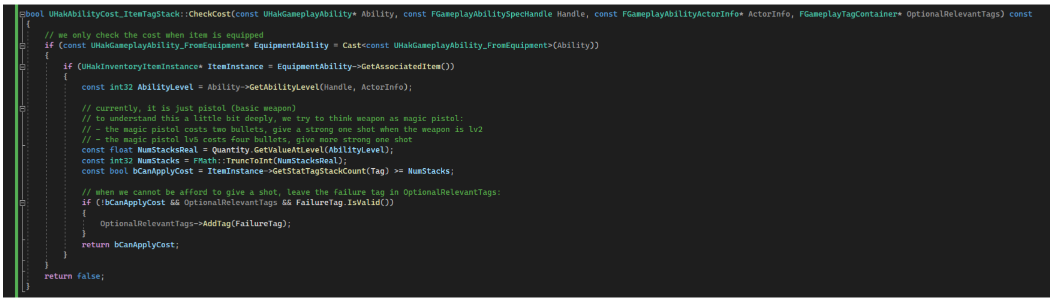


```cpp
#pragma once

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

/** forward declarations */
class UHakEquipmentInstance;
class UHakInventoryItemInstance;

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

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



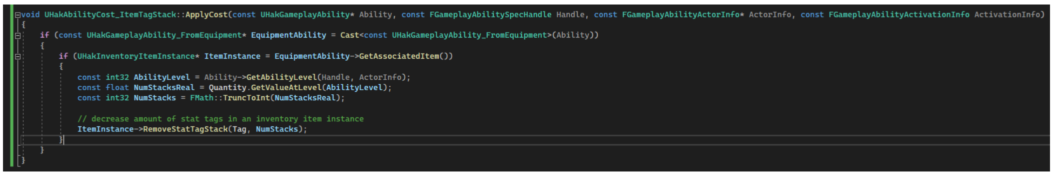
```cpp
#include "HakGameplayAbility_FromEquipment.h"
#include "HakEquipmentInstance.h"
#include "HakGame/Inventory/HakInventoryItemInstance.h"
#include UE_INLINE_GENERATED_CPP_BY_NAME(HakGameplayAbility_FromEquipment)

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

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

UHakInventoryItemInstance* UHakGameplayAbility_FromEquipment::GetAssociatedItem() const
{
    if (UHakEquipmentInstance* Equipment = GetAssociatedEquipment())
    {
        // In Lyra, equipment is equipped by inventory item instance:
        // - so, equipment's instigator should be inventory item instance
        // - otherwise, it will return nullptr by failing casting to HakInventoryItemInstance
        return Cast<UHakInventoryItemInstance>(Equipment->GetInstigator());
    }
    return nullptr;
}
```

☐ CheckCost():



```cpp
bool UHakAbilityCost_ItemTagStack::CheckCost(const UHakGameplayAbility* Ability, const FGameplayAbilitySpecHandle Handle, const FGameplayAbilityActorInfo* ActorInfo, FGameplayTagContainer* OptionalRelevantTags) const
{
    // we only check the cost when item is equipped
    if (const UHakGameplayAbility_FromEquipment* EquipmentAbility = Cast<const UHakGameplayAbility_FromEquipment>(Ability))
    {
        if (UHakInventoryItemInstance* ItemInstance = EquipmentAbility->GetAssociatedItem())
        {
            const int32 AbilityLevel = Ability->GetAbilityLevel(Handle, ActorInfo);

            // currently, it is just pistol (basic weapon)
            // to understand this a little bit deeply, we try to think weapon as magic pistol:
            // - the magic pistol costs two bullets, give a strong one shot when the weapon is lv2
            // - the magic pistol lv5 costs four bullets, give more strong one shot
            const float NumStacksReal = Quantity.GetValueAtLevel(AbilityLevel);
            const int32 NumStacks = FMath::TruncToInt(NumStacksReal);
            const bool bCanApplyCost = ItemInstance->GetStatTagStackCount(Tag) >= NumStacks;

            // when we cannot be afford to give a shot, leave the failure tag in OptionalRelevantTags:
            if (!bCanApplyCost && OptionalRelevantTags && FailureTag.IsValid())
            {
                OptionalRelevantTags->AddTag(FailureTag);
            }
            return bCanApplyCost;
        }
    }
    return false;
}
```

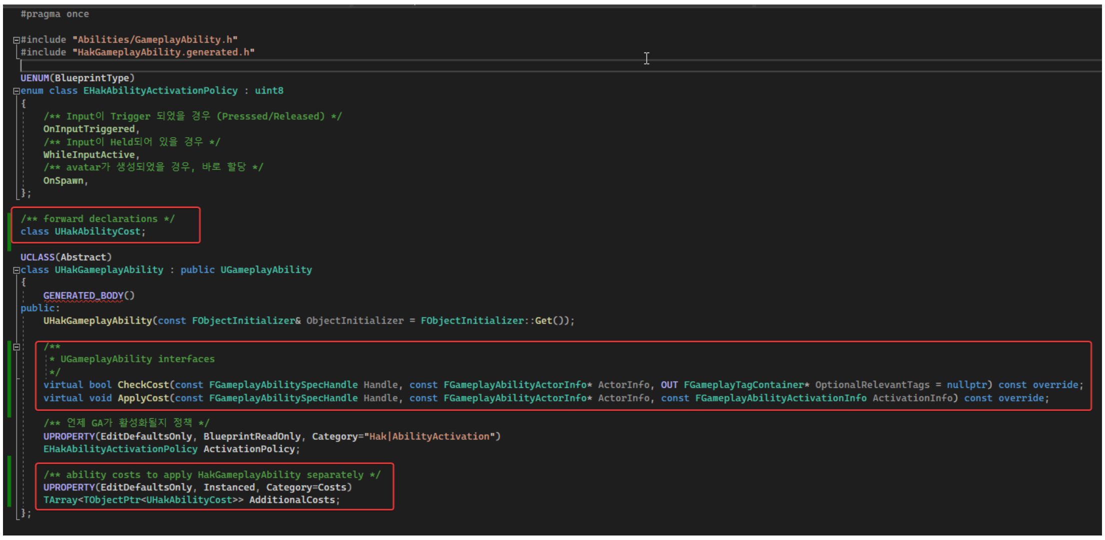
☐ ApplyCost():



```cpp
void UHakAbilityCost_ItemTagStack::ApplyCost(const UHakGameplayAbility* Ability, const FGameplayAbilitySpecHandle Handle, const FGameplayAbilityActorInfo* ActorInfo, const FGameplayAbilityActivationInfo ActivationInfo)
{
    if (const UHakGameplayAbility_FromEquipment* EquipmentAbility = Cast<const UHakGameplayAbility_FromEquipment>(Ability))
    {
        if (UHakInventoryItemInstance* ItemInstance = EquipmentAbility->GetAssociatedItem())
        {
            const int32 AbilityLevel = Ability->GetAbilityLevel(Handle, ActorInfo);
            const float NumStacksReal = Quantity.GetValueAtLevel(AbilityLevel);
            const int32 NumStacks = FMath::TruncToInt(NumStacksReal);

            // decrease amount of stat tags in an inventory item instance
            ItemInstance->RemoveStatTagStack(Tag, NumStacks);
        }
    }
}
```

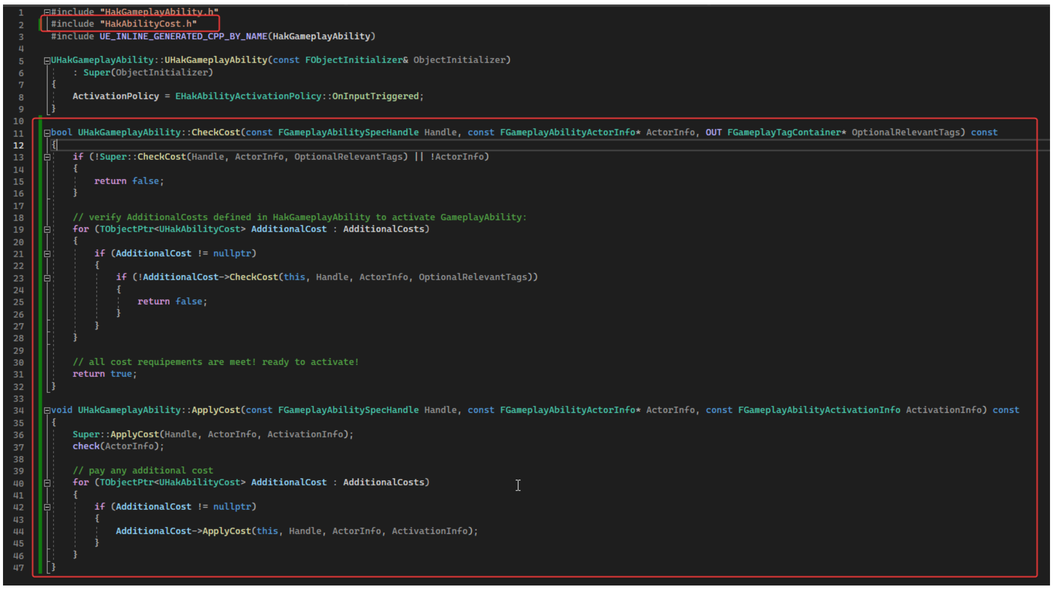
- Now we define HakAbilityCost_ItemTagStack, we need to define AbilityCost in HakGameplayAbility to process Ability Cost calculation correctly, by overriding methods

☐ UHakGameplayAbility:



```cpp
#pragma once

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

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

/** forward declarations */
class UHakAbilityCost;

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

	/**
	 * UGameplayAbility interfaces
	 */
	virtual bool CheckCost(const FGameplayAbilitySpecHandle Handle, const FGameplayAbilityActorInfo* ActorInfo, OUT FGameplayTagContainer* OptionalRelevantTags = nullptr) const override;
	virtual void ApplyCost(const FGameplayAbilitySpecHandle Handle, const FGameplayAbilityActorInfo* ActorInfo, const FGameplayAbilityActivationInfo ActivationInfo) const override;

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

	/** ability costs to apply HakGameplayAbility separately */
	UPROPERTY(EditDefaultsOnly, Instanced, Category=Costs)
	TArray<TObjectPtr<UHakAbilityCost>> AdditionalCosts;
};
```

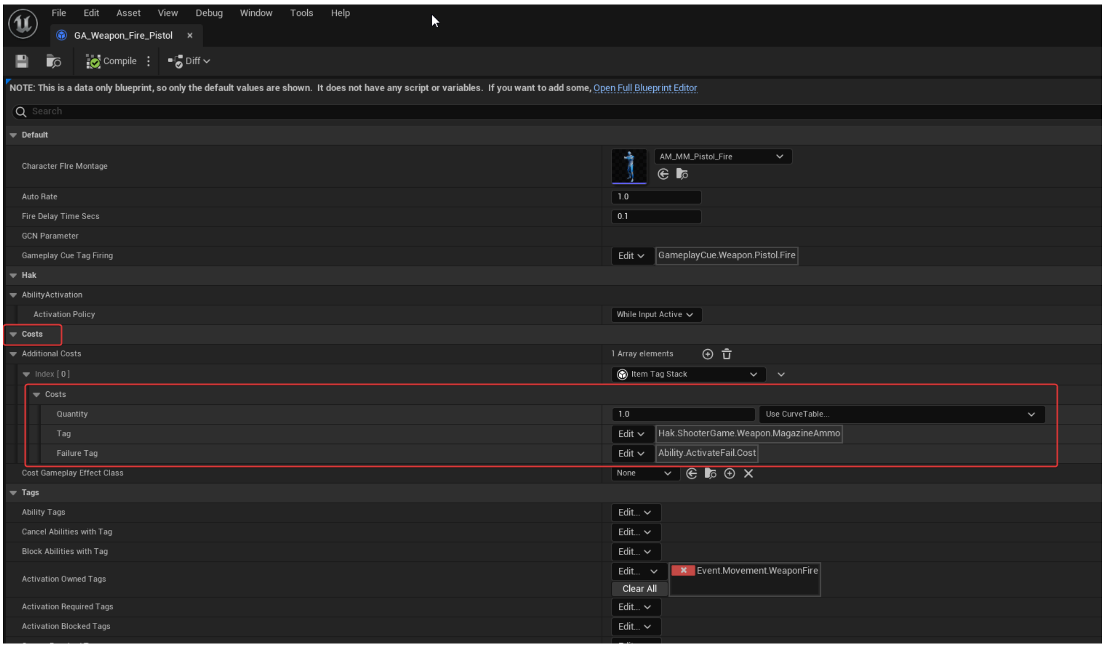
☐ CheckCost() and ApplyCost():



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

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

bool UHakGameplayAbility::CheckCost(const FGameplayAbilitySpecHandle Handle, const FGameplayAbilityActorInfo* ActorInfo, OUT FGameplayTagContainer* OptionalRelevantTags) const
{
	if (!Super::CheckCost(Handle, ActorInfo, OptionalRelevantTags) || !ActorInfo)
	{
		return false;
	}

	// verify AdditionalCosts defined in HakGameplayAbility to activate GameplayAbility:
	for (TObjectPtr<UHakAbilityCost> AdditionalCost : AdditionalCosts)
	{
		if (AdditionalCost != nullptr)
		{
			if (!AdditionalCost->CheckCost(this, Handle, ActorInfo, OptionalRelevantTags))
			{
				return false;
			}
		}
	}

	// all cost requipements are meet! ready to activate!
	return true;
}

void UHakGameplayAbility::ApplyCost(const FGameplayAbilitySpecHandle Handle, const FGameplayAbilityActorInfo* ActorInfo, const FGameplayAbilityActivationInfo ActivationInfo) const
{
	Super::ApplyCost(Handle, ActorInfo, ActivationInfo);
	check(ActorInfo);

	// pay any additional cost
	for (TObjectPtr<UHakAbilityCost> AdditionalCost : AdditionalCosts)
	{
		if (AdditionalCost != nullptr)
		{
			AdditionalCost->ApplyCost(this, Handle, ActorInfo, ActivationInfo);
		}
	}
}
```

- Now we should add Pistol's GA to AbilityCost

☐ GA_Weapon_Fire_Pistol:



NOTE: This is a data only blueprint, so only the default values are shown. It does not have any script or variables. If you want to add some, Open Full Blueprint Editor

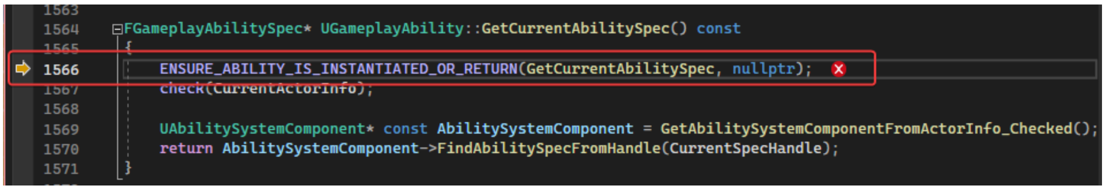
Default
- Character FIre Montage: AM_MM_Pistol_Fire
- Auto Rate: 1.0
- Fire Delay Time Secs: 0.1
- GCN Parameter
- Gameplay Cue Tag Firing: Edit GameplayCue.Weapon.Pistol.Fire

Hak
- AbilityActivation
  - Activation Policy: While Input Active

Costs
- Additional Costs: 1 Array elements
  - Index [0]: Item Tag Stack
    - Costs
      - Quantity: 1.0 Use CurveTable...
      - Tag: Edit Hak.ShooterGame.Weapon.MagazineAmmo
      - Failure Tag: Edit Ability.ActivateFail.Cost
- Cost Gameplay Effect Class: None

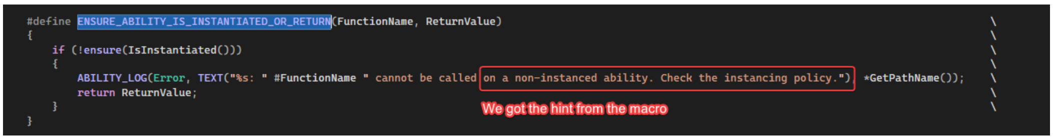
Tags
- Ability Tags: Edit...
- Cancel Abilities with Tag: Edit...
- Block Abilities with Tag: Edit...
- Activation Owned Tags: Edit... Event.Movement.WeaponFire Clear All
- Activation Required Tags: Edit...
- Activation Blocked Tags: Edit...

□ We get the ERROR (excecption):



```cpp
FGameplayAbilitySpec* UGameplayAbility::GetCurrentAbilitySpec() const
{
    ENSURE_ABILITY_IS_INSTANTIATED_OR_RETURN(GetCurrentAbilitySpec, nullptr);
    check(CurrentActorInfo);

    UAbilitySystemComponent* const AbilitySystemComponent = GetAbilitySystemComponentFromActorInfo_Checked();
    return AbilitySystemComponent->FindAbilitySpecFromHandle(CurrentSpecHandle);
}
```

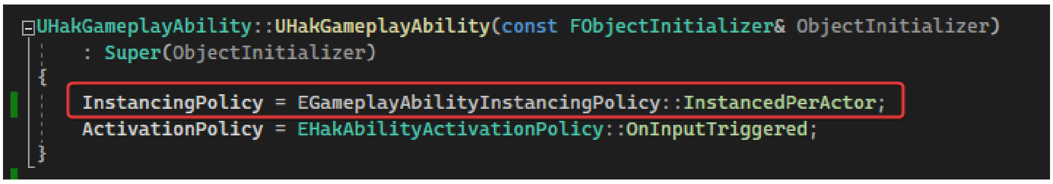
□ Let’s examine ENSURE_ABILITY_IS_INSTANTIATED_OR_RETURN:



```cpp
#define ENSURE_ABILITY_IS_INSTANTIATED_OR_RETURN(FunctionName, ReturnValue)                                                                          \
{                                                                                                                                                    \
    if (!ensure(IsInstantiated()))                                                                                                                   \
    {                                                                                                                                                \
        ABILITY_LOG(Error, TEXT("%s: " #FunctionName " cannot be called on a non-instanced ability. Check the instancing policy."), *GetPathName());  \
        return ReturnValue;                                                                                                                          \
    }                                                                                                                                                \
}
```

We got the hint from the macro

□ We are going to set default InstancingPolicy as InstancedPerActor like Lyra:



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

- As we show the video for describing the goal of today’s lecture, we get exact same result:

https://prod-files-secure.s3.us-west-2.amazonaws.com/ecba3054-6b52-40da-ba34-e88eb287722c/b405c3e5-cd25-43fe-a4fc-27ace917440e/UnrealEditor_vV6mNn0arB.mp4

정리:

▼ **펼치기**

□ 왜 GameplayEffect가 필요할까?

- 기본적으로 `GAS` 는 **이벤트 기반 게임 로직을 위한 프레임워크로 이해**해야 한다
- 단순 BP를 통해 AttributeSet의 GameplayAttribute를 매-틱마다 `+,-,*,/` 연산을 하는 것은 **굉장한 insturction 낭비**이다.
    - (참고로 BP의 노드 실행은 인터프리터로 생각하면 되기 때문에, 성능이 매우 안좋다)
        - 예로 들어, BP 곱셈을 실행하기 위해 C++에서 거의 100배이산 연산을 낭비하는 느낌으로 생각하면 된다
- 그럼 **GameplayEffect를 통해 단순히 주기적이든 단발적이든 이벤트 등록만 했다면?**
    - **연산이 대부분 C++에서** 돌기 때문에 BP보다 빠르다
    - BP는 단순히 여러분들이 분기를 시키기 위한 단순 업무만 하고, 중요한 업무는 C++에 넘긴다고 생각하면 된다.
    - 그에 대한 철학으로 **GameplayEffect는 매우 필요**하다

□ `GAS` 는 **BP와 C++을 더 잘 사용하기 위한 프레임워크**로 기존 과거 BP Nativization의 대체로 나오지 않았는가 유추해본다:

- 게임 플레이 로직을 짤 때, BP로 큰 게임 디자인을 만들지만, 실제 성능적 중요한 코드는 8:2 법칙에 의해 20% 수렴한다:
    - 이 20%를 담당하는 것이 GAS로 볼 수 있지 않을까 생각해본다
- GAS가 다소 복잡해보일 수도 있으나, **BP의 역활을 대신하여 Block을 제공하는 것이라고 생각하면, C++이 훨씬 빠르니 이해 가능**하다.
- 이번 Lyra를 분석해서 필자는 그렇게 판단하고 있다:

- 그래서 필자가 만약 프로젝트를 리드한다면 GAS와 Lyra는 꼭 쓸 거 같다:
    - 처음부터 만들기 시간과 비용이 크다
    - `GAS 프레임워크` 는 **BP를 통해 디자이너에게 잘 쓰도록 유도**만 한다면, 생산성과 성능을 챙길 수 있다고 본다.

- 마지막으로, 여러분들이 꼭 가져야 할 스킬은 시니어가 **어떻게 엔진 코드를 읽고 어떻게 이해**하며, **엔진의 철학을 어떻게 맞추어나가는지**에 대해 인사이트를 키워야 한다:
    - 왕도는 없다
    - 코드를 많이 읽고 깊게 이해해보며, 많은 생각을 해보아야 한다

  → 이를 위한 엔진 코드 분석 강의를 준비할 예정이다

- 유용한 추가적인 GAS 문서 링크:

  Building on Lyra
  One simulant attempts to share insight with others.
  https://x157.github.io/

  https://github.com/tranek/GASDocumentation

    - 이제 여러분들은 해당 문서를 읽으며 **더 깊이 GAS와 Lyra를 이해할 준비**가 되었다고 생각한다.