**기술 기획 문서: C++/GAS 기반 아이템 조합 시스템 (상세 구현)**

**문서 버전:** 3.0 (GAS Integrated)

**작성일:** 2025년 10월 17일

**1. 시스템 개요**

본 문서는 \*\*게임플레이 어빌리티 시스템(GAS)\*\*을 핵심 백엔드로 사용하는 C++ 주축의 **클릭 기반 아이템 조합 시스템**의 기술적 구현 방안을 명세한다. 아이템이 부여하는 스탯은 GameplayEffect로, 고유 능력은 GameplayAbility로 추상화하여 시스템의 모듈성과 확장성을 극대화한다.

**2. C++/GAS 아키텍처 (C++/GAS Architecture)**

* **UAbilitySystemComponent (ASC):** 영웅의 모든 스탯, 어빌리티, 상태 효과를 관리하는 중앙 허브.
* **UAttributeSet:** 영웅이 가질 수 있는 모든 기본 능력치(공격력, 공속 등)를 C++로 정의하는 데이터 클래스.
* **UGameplayEffect (GE):** 아이템이 제공하는 **스탯 보너스**를 정의하는 데이터 에셋. 블루프린트로 생성하여 기획자가 수치를 조절한다.
* **UGameplayAbility (GA):** 아이템이 부여하는 **고유 능력**(예: 연쇄 번개)을 정의하는 로직 클래스. C++ 또는 블루프린트로 구현한다.
* **UInventoryComponent:** 인벤토리 아이템 목록을 관리하고, 아이템 장착/해제 시 ASC에 GE 적용/제거 및 GA 부여/해제를 **명령**하는 오케스트레이터 역할을 수행한다.
* **블루프린트:** 데이터 테이블 입력, UMG 디자인, C++ 클래스 상속 및 간단한 이벤트 연결에만 사용된다.

**3. 데이터 구조 설계 (C++ ItemTypes.h)**

아이템 데이터 구조(F\_ItemData)를 GAS와 직접 연동되도록 수정한다.

C++

// ItemTypes.h

#**pragma** once

#**include** "GameplayEffect.h"

#**include** "GameplayAbility.h"

#**include** "Engine/DataTable.h"

#**include** "ItemTypes.generated.h"

// ... EItemGrade, EStatType 열거형은 이전과 동일 ...

// 아이템 데이터 구조체 (GAS 연동)

USTRUCT(BlueprintType)

struct FItemData : public FTableRowBase

{

GENERATED\_BODY()

// ... ItemID, ItemName, Icon, Grade 등 기본 정보는 동일 ...

// 이 아이템이 제공하는 스탯 보너스를 담은 GameplayEffect 클래스

UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "GAS")

TSubclassOf<UGameplayEffect> StatEffect;

// 이 아이템이 부여하는 고유 능력을 담은 GameplayAbility 클래스

UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "GAS")

TSubclassOf<UGameplayAbility> GrantedAbility;

UPROPERTY(EditAnywhere, BlueprintReadOnly)

FText AbilityDescription;

};

// 조합법 구조체는 이전과 동일

USTRUCT(BlueprintType)

struct FRecipeData : public FTableRowBase { /\* ... \*/ };

**[블루프린트 작업]**

1. DT\_Items 데이터 테이블을 열면, 각 아이템 항목에 Stat Effect와 Granted Ability를 선택할 수 있는 드롭다운 메뉴가 생긴다.
2. 기획자는 스탯 보너스용 GameplayEffect 블루프린트와 고유 능력용 GameplayAbility 블루프린트를 미리 만들어두고, 이 테이블에서 해당 블루프린트를 선택하여 아이템에 기능을 할당한다.

**4. 핵심 로직 구현**

**4.1. AttributeSet.h**

영웅의 모든 스탯을 정의한다.

C++

// MyAttributeSet.h

#**pragma** once

#**include** "AttributeSet.h"

#**include** "AbilitySystemComponent.h"

#**include** "MyAttributeSet.generated.h"

#**define** ATTRIBUTE\_ACCESSORS(ClassName, PropertyName) // ... (Getter/Setter 매크로)

UCLASS()

class YOURPROJECT\_API UMyAttributeSet : public UAttributeSet

{

GENERATED\_BODY()

public:

virtual void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>& OutLifetimeProps) const override;

UPROPERTY(BlueprintReadOnly, Category="Attributes", ReplicatedUsing=OnRep\_AttackDamage)

FGameplayAttributeData AttackDamage;

ATTRIBUTE\_ACCESSORS(UMyAttributeSet, AttackDamage);

// ... AttackSpeed, CritChance, CritDamage 등 모든 스탯을 동일하게 정의 ...

protected:

UFUNCTION()

virtual void OnRep\_AttackDamage(const FGameplayAttributeData& OldAttackDamage);

// ... 모든 스탯에 대한 OnRep 함수 선언 ...

};

**4.2. InventoryComponent.h / .cpp**

아이템 장착/해제 시 ASC를 제어하는 로직이 추가된다.

C++

// InventoryComponent.h

// ... 기존 코드에 추가 ...

private:

// 아이템 장착/해제 시 ASC를 제어하기 위한 참조

UPROPERTY()

TWeakObjectPtr<class UAbilitySystemComponent> AbilitySystemComponent;

// 적용된 GE와 부여된 GA의 핸들을 저장하여 나중에 제거할 수 있도록 함

TMap<FName, FActiveGameplayEffectHandle> AppliedStatEffectHandles;

TMap<FName, FGameplayAbilitySpecHandle> GrantedAbilityHandles;

public:

void Initialize(UAbilitySystemComponent\* InASC);

virtual void AddItem(FName ItemID) override; // override 키워드로 재정의 명시

virtual void RemoveItem(FName ItemID); // 단일 아이템 제거 함수 추가

C++

// InventoryComponent.cpp

void UInventoryComponent::Initialize(UAbilitySystemComponent\* InASC)

{

AbilitySystemComponent = InASC;

}

// 아이템 추가(장착) 로직

void UInventoryComponent::AddItem(FName ItemID)

{

if (!AbilitySystemComponent.IsValid()) return;

InventoryItems.Add(ItemID);

// DT\_Items에서 아이템 데이터 가져오기 (FindItemDataByID는 직접 구현해야 할 헬퍼 함수)

const FItemData\* ItemData = FindItemDataByID(ItemID);

if (!ItemData) return;

// 1. 스탯 효과(GE) 적용

if (ItemData->StatEffect)

{

FGameplayEffectContextHandle EffectContext = AbilitySystemComponent->MakeEffectContext();

EffectContext.AddSourceObject(this);

FGameplayEffectSpecHandle SpecHandle = AbilitySystemComponent->MakeOutgoingSpec(ItemData->StatEffect, 1.0f, EffectContext);

if (SpecHandle.IsValid())

{

FActiveGameplayEffectHandle ActiveHandle = AbilitySystemComponent->ApplyGameplayEffectSpecToSelf(\*SpecHandle.Data.Get());

AppliedStatEffectHandles.Add(ItemID, ActiveHandle); // 나중에 제거할 수 있도록 핸들 저장

}

}

// 2. 고유 능력(GA) 부여

if (ItemData->GrantedAbility)

{

FGameplayAbilitySpec AbilitySpec(ItemData->GrantedAbility);

FGameplayAbilitySpecHandle AbilityHandle = AbilitySystemComponent->GiveAbility(AbilitySpec);

GrantedAbilityHandles.Add(ItemID, AbilityHandle); // 핸들 저장

}

OnInventoryUpdated.Broadcast();

}

// 아이템 제거(해제) 로직

void UInventoryComponent::RemoveItem(FName ItemID)

{

if (!AbilitySystemComponent.IsValid() || !InventoryItems.Contains(ItemID)) return;

// 1. 스탯 효과(GE) 제거

if (AppliedStatEffectHandles.Contains(ItemID))

{

AbilitySystemComponent->RemoveActiveGameplayEffect(AppliedStatEffectHandles[ItemID]);

AppliedStatEffectHandles.Remove(ItemID);

}

// 2. 고유 능력(GA) 제거

if (GrantedAbilityHandles.Contains(ItemID))

{

AbilitySystemComponent->ClearAbility(GrantedAbilityHandles[ItemID]);

GrantedAbilityHandles.Remove(ItemID);

}

InventoryItems.RemoveSingle(ItemID);

OnInventoryUpdated.Broadcast();

}

**4.3. 영웅 캐릭터 C++ (능력 발동 트리거)**

영웅의 기본 공격 로직에 **Gameplay Event**를 보내는 코드를 추가하여, 아이템의 고유 능력(GameplayAbility)이 반응할 수 있는 신호를 보낸다.

C++

// YourCharacter.cpp

void AYourCharacter::PerformAttack() // 기본 공격을 수행하는 함수 예시

{

// ... 실제 공격 로직 (라인 트레이스, 애니메이션 재생 등) ...

// 공격이 적중했을 때, "Event.Attack.Hit" 태그로 게임플레이 이벤트를 ASC에 보냄

if (AbilitySystemComponent)

{

FGameplayEventData Payload;

Payload.Target = HitActor; // 맞은 대상을 페이로드에 담아 전달

AbilitySystemComponent->HandleGameplayEvent(FGameplayTag::RequestGameplayTag(FName("Event.Attack.Hit")), &Payload);

}

}

**[블루프린트 작업]**

1. 아이템의 고유 능력 GameplayAbility 블루프린트를 열고, ActivateAbility가 아닌 **ActivateAbilityFromEvent** 이벤트를 구현한다.
2. 클래스 디폴트에서 Ability Tags -> Activation Required Tags에 **Event.Attack.Hit** 태그를 추가한다.
3. 이제 PerformAttack에서 이벤트가 발생할 때마다, 이 태그를 가진 모든 아이템의 고유 능력이 자동으로 발동된다.

**5. UI 구현 (GAS 데이터 연동)**

UI 위젯은 더 이상 InventoryComponent로부터 스탯을 계산하지 않는다. 대신 AbilitySystemComponent로부터 직접 현재 스탯 값을 가져온다.

* **WBP\_StatDisplay (예시: 스탯창 위젯):**
  + 이 위젯은 AbilitySystemComponent의 참조를 가진다.
  + 스탯을 표시해야 할 때, ASC->GetFloatAttribute(UMyAttributeSet::GetAttackDamageAttribute()) 와 같은 함수를 호출하여 현재 최종 합산된 스탯 값을 직접 가져와 UI에 표시한다.
  + 또한, ASC->GetGameplayAttributeValueChangeDelegate()를 사용하여 특정 스탯이 변경될 때마다 UI를 자동으로 업데이트하는 이벤트를 바인딩할 수 있다.