언리얼 페스트 2024 서울

게임플레이 어빌리티 시스템을 사용한 프로토타입 제작 사례

박준수
CTO
㈜넥스트스테이지

강현우
대표이사
㈜넥스트스테이지

강연제목과 목차

Section 1
게임플레이 어빌리티 시스템의 기본 개념과 적용 사례

Section 2
Common UI를 통해 HUD와 메뉴 시스템 적용 사례 공유

Section 3
프로토타입 제작 팁과 사례 공유

발표에 앞서

㈜넥스트스테이지의 CTO 박준수

- 2021년 콘솔 게임 '울트라 에이지' 개발
- 신규 차세대 콘솔 프로젝트 진행 중
- UDN 컨설턴트 파트너

게임 플레이 어빌리티 시스템의 기본 개념과 적용 사례

Gameplay Ability System 란?

고도로 유연한 프레임워크
능력, 상태, 어트리뷰트를 구축하고 복잡한 상호작용과 시각적인 효과를 유연하게 관리할 수 있는 시스템

손쉬운 구성과 재사용
GAS 는 다양한 능력을 쉽게 구성하고 재사용할 수 있도록 도와주며, 복잡한 능력을 구현할 수 있습니다.

뛰어난 확장성과 유연성
확장성과 유연성이 뛰어나 다양한 게임 장르에 맞게 적용 가능하며, 멀티플레이 게임에서의 동기화 문제를 최소화 할 수 있습니다.

Gameplay Ability System 사용하는 이유

복잡한 Ability 관리

게임 내 캐릭터가 사용할 액티브, 패시브 능력을 체계적으로 관리
활성화 및 비활성화, Cooldown 및 Cost 쉽게 구현

구조화된 Attribute Set

버프나 디버프를 유연하게 적용가능 하며 재구현을 통해 Attribute 값 커스터마이즈

체계적이면서 유연한 상호작용

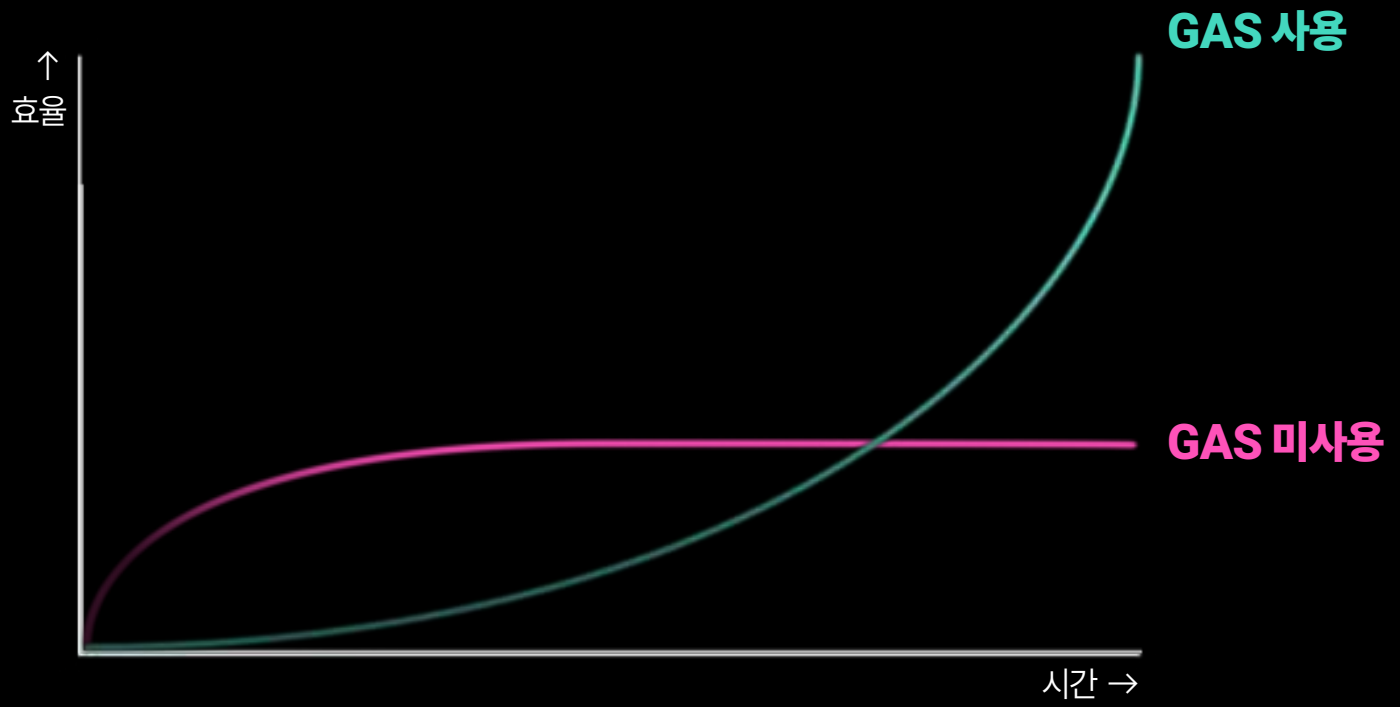
게임 내 액터 간의 상호작용을 Gameplay Effect Class를 활용하여 전달

성능 최적화

에픽게임즈에서 제작한 프레임워크로 대규모 게임에서도 안정적인 성능을 제공

개발 생산성 향상

모듈화 되어 있어 재사용성과 확장성이 뛰어나
복잡한 게임 플레이 메커니즘을 빠르게 프로토타이핑


GAS 사용
GAS 미사용
↑
효율
시간 →

Gameplay Ability System 구성 요소

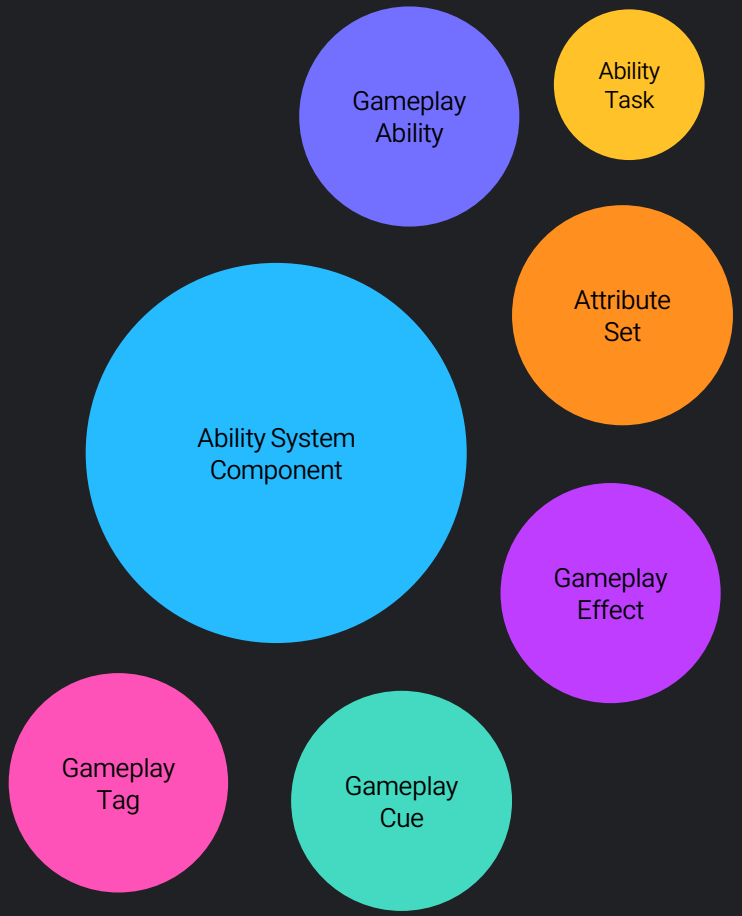
Ability System Component
Ability, Attribute Set, Gameplay Effect 관리 하는 컴포넌트
Ability System Component 간의 상호작용

Ability
캐릭터가 수행할 수 있는 행동

Gameplay Effect
캐릭터에 적용되는 상호작용 및 효과 정의

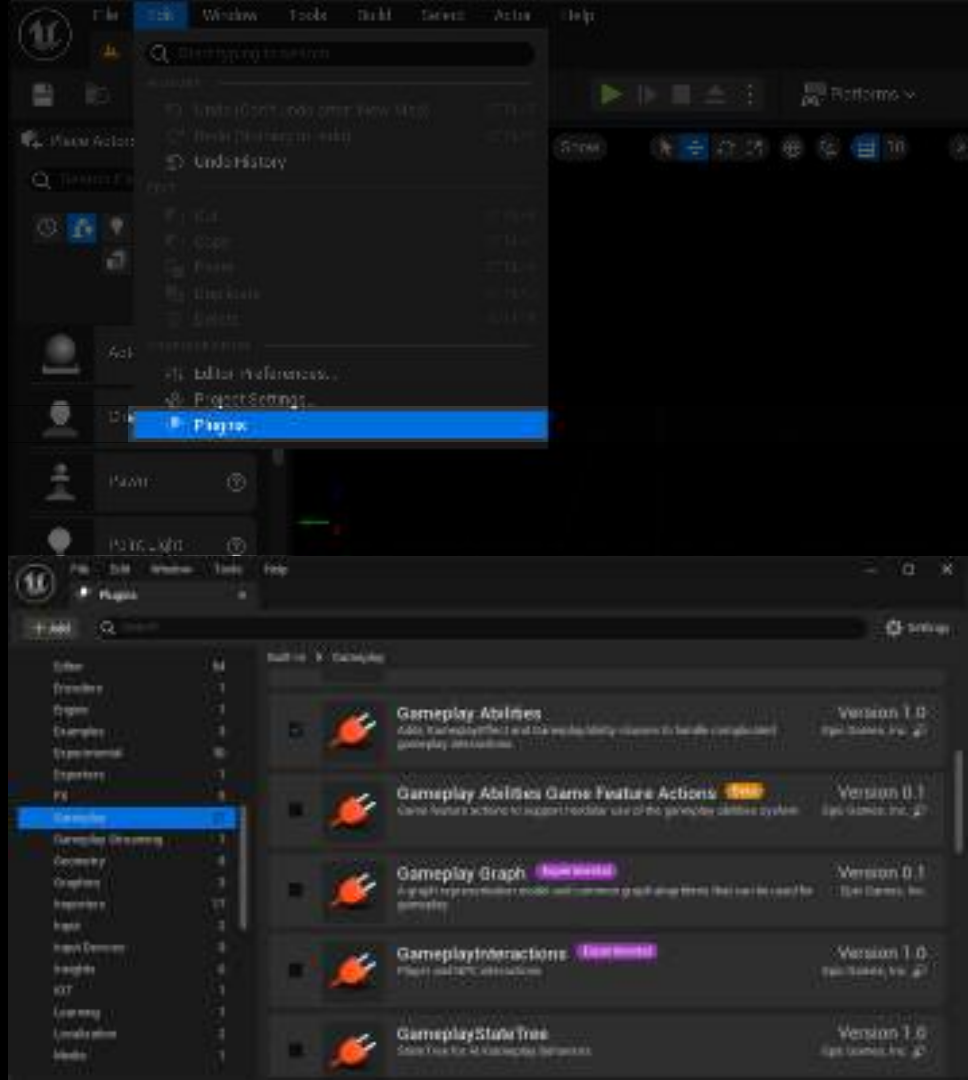
Attribute Set
캐릭터 스탯 값의 모음

Gameplay Cue
시각적, 청각적 피드백



Gameplay Ability System 활성화 방법

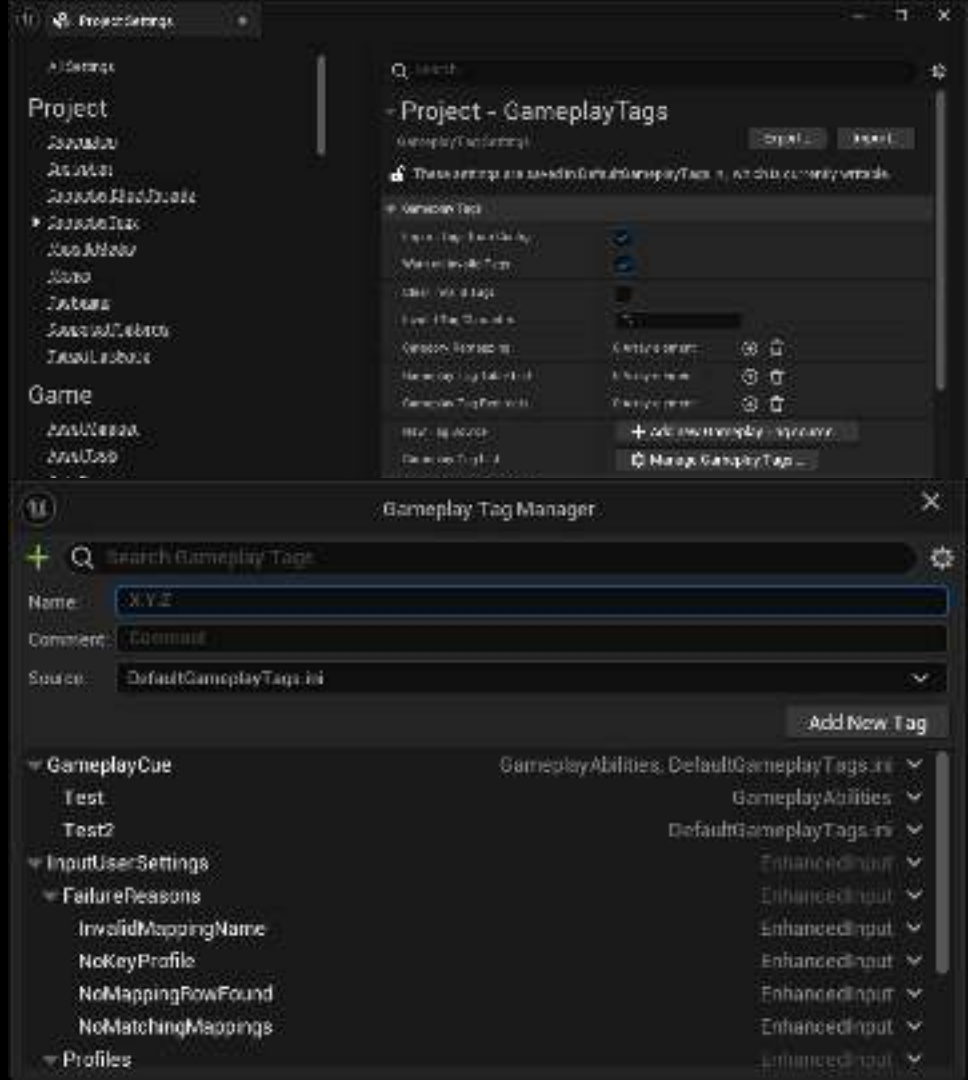
- Edit → Plugins 창 열기
- Gameplay 카테고리 → Gameplay Abilities 플러그인 활성화



Gameplay Tag

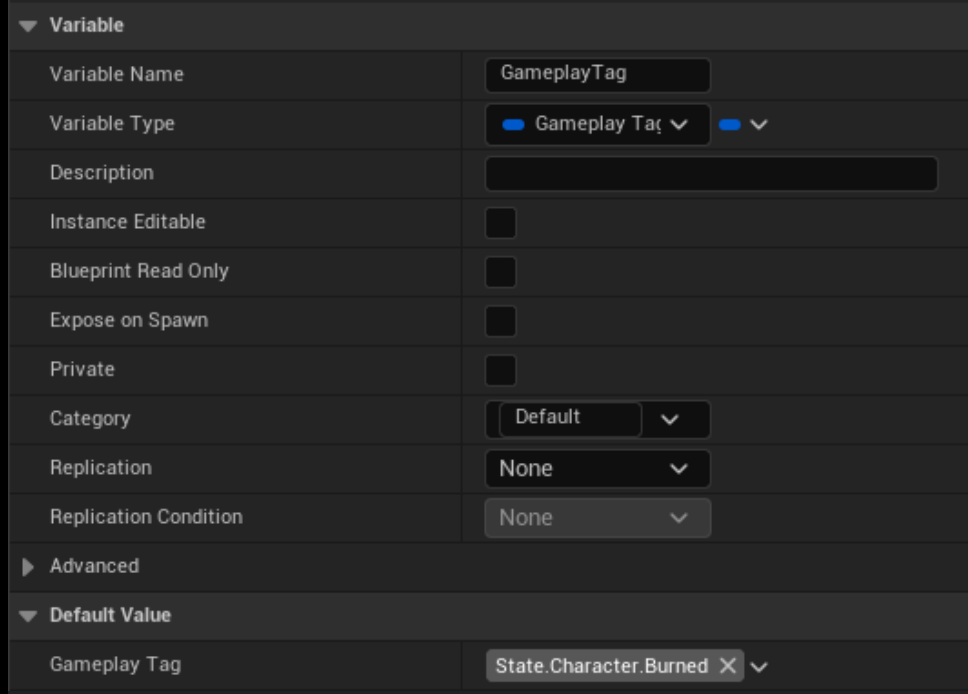
Gameplay Tag Manage 방법

- Edit → Project Settings 창 열기
- Project
  → GameplayTags
  → GameplayTag List
  → Manage Gameplay Tags…



Gameplay Tag

- FName을 가지고 있는 구조체
- Gameplay Tags Manager에서 관리
- Gameplay Tags Manager에 의해 관리되는 Gameplay Tag Node를 이용하여 태그 일치 확인




```
FGameplayTag StateBurn = FGameplayTag::RequestGameplayTag(
    FName(TEXT("State.Character.Burned")));
```

Gameplay Tag 특징

Enum이나 Bool기반이 아닌 Tag를 사용하여 다양한 상태를 표현할 수 있으며,
상태를 사람이 이해하기 쉬운 문자로 나타낼 수 있습니다.

계층적 트리 구조

태그는 계층적 트리 구조로 구성됩니다.

- State.Character.Element.Fire
- State.Character.Element.Water

상태 추적 및 관리

캐릭터의 상태를 유연하게 관리 할 수 있습니다.

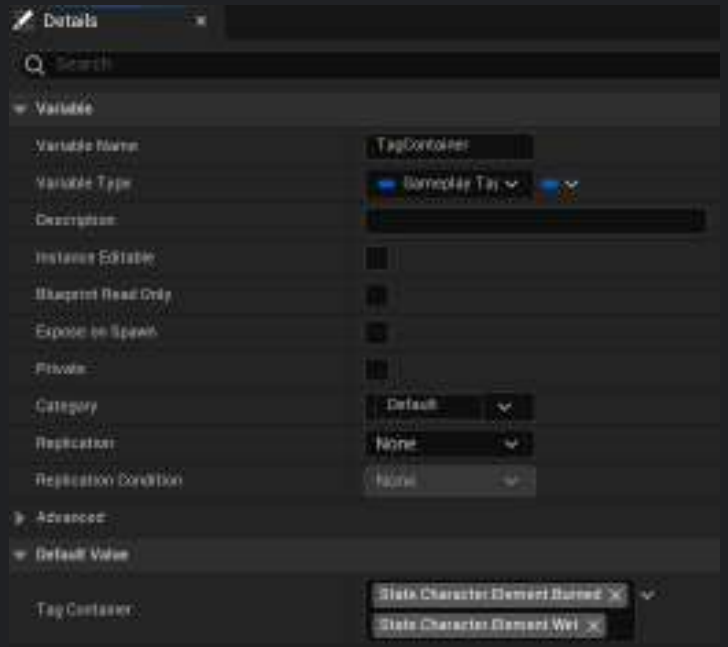
- State.Character.Alive
- State,Character.Dead

관리 용이성

다양한 상태를 태그로 관리함으로써 코드의 복잡성을 줄이고, 쉽게 확장할 수 있습니다.

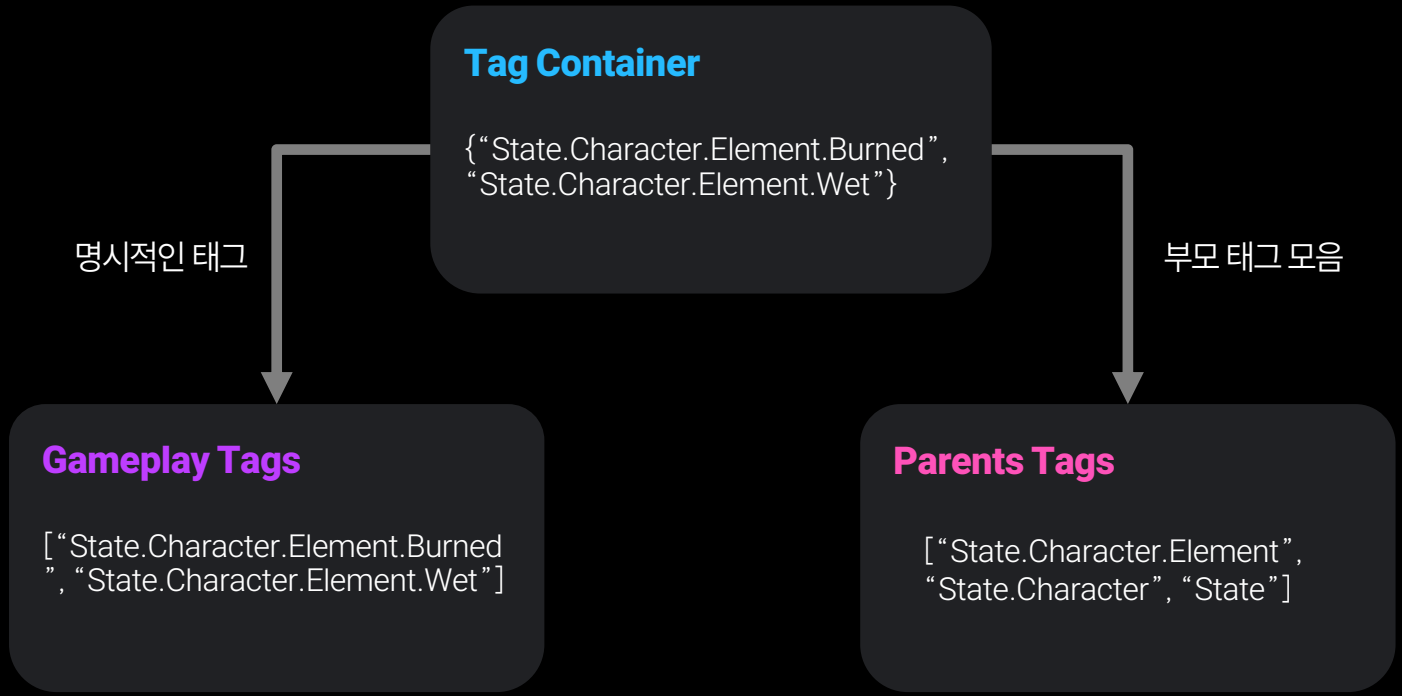
Gameplay Tag Container

- 여러 개의 Gameplay Tag 구조체
- 컨테이너 내의 Gameplay Tag를 제거 및 추가 가능
- 컨테이너 간의 태그 비교
- “Gameplay Tags” 배열에 명시적으로 추가된 Tag들을 저장
- “Parents Tags” 배열에 태그들의 부모 Tag들을 저장




```
FGameplayTagContainer TagContainer;

TagContainer.AddTag(
    FGameplayTag::RequestGameplayTag(
    FName("State.Character.Element.Burned")));

TagContainer.AddTag(
    FGameplayTag::RequestGameplayTag(
    FName("State.Character.Element.Wet")));
```

Gameplay Tag Container 간단한 예제



Gameplay Tag 주요 함수

- Tag와 Tag 비교 함수
- Tag와 Tag Container 비교 함수
- Tag Container와 Tag Container 비교 함수

Has Tag

Has Tag

Tag Container에서 단일 Gameplay Tag 부분적 포함 유무

{"Insect.Ant", "Mammal.Cat"}.HasTag("Insect") → **true**

{“Insect, Mammal"}.HasTag("Insect.Ant") → **false**

Has Tag Exact

Tag Container에서 단일 Gameplay Tag 명시적 포함 유무

{"Insect.Ant","Mammal.Cat"}
.HasTagExact(“Insect.Ant”) → **true**

{"Insect.Ant", "Mammal.Cat"}
.HasTagExact(“Insect”) → **false**

Has All

Has All

Tag Container에 모든 Gameplay Tag 부분적 포함 유무

{"Insect.Ant", "Mammal.Cat"}
.HasAll({"Insect"," Mammal"}) → **true**

{"Insect.Ant", "Mammal.Cat"}
.HasAll({"Insect"," Bird"}) → **false**

Has All Exact

Tag Container에 모든 Gameplay Tag 명시적 포함 유무

{"Insect.Ant", "Mammal.Cat"}
.HasAllExact({"Insect"," Mammal"}) → **false**

{"Insect.Ant", "Mammal.Cat"}
.HasAllExact({" Insect.ant"}) → **true**

{"Insect.Ant", "Mammal.Cat"}
.HasAllExact({" Insect.ant"," Bird.Sparrow"}) → **false**

Has Any

Has Any

Tag Container에 어떤 것 하나라도 Gameplay Tag
부분적 포함 유무

{"Insect.Ant", "Mammal.Cat"}
.HasAny({" Insect"," Bird"}) → **true**

{"Insect.Ant", "Mammal.Cat"}
.HasAny({“Reptile"," Bird"}) → **false**

Has Any Exact

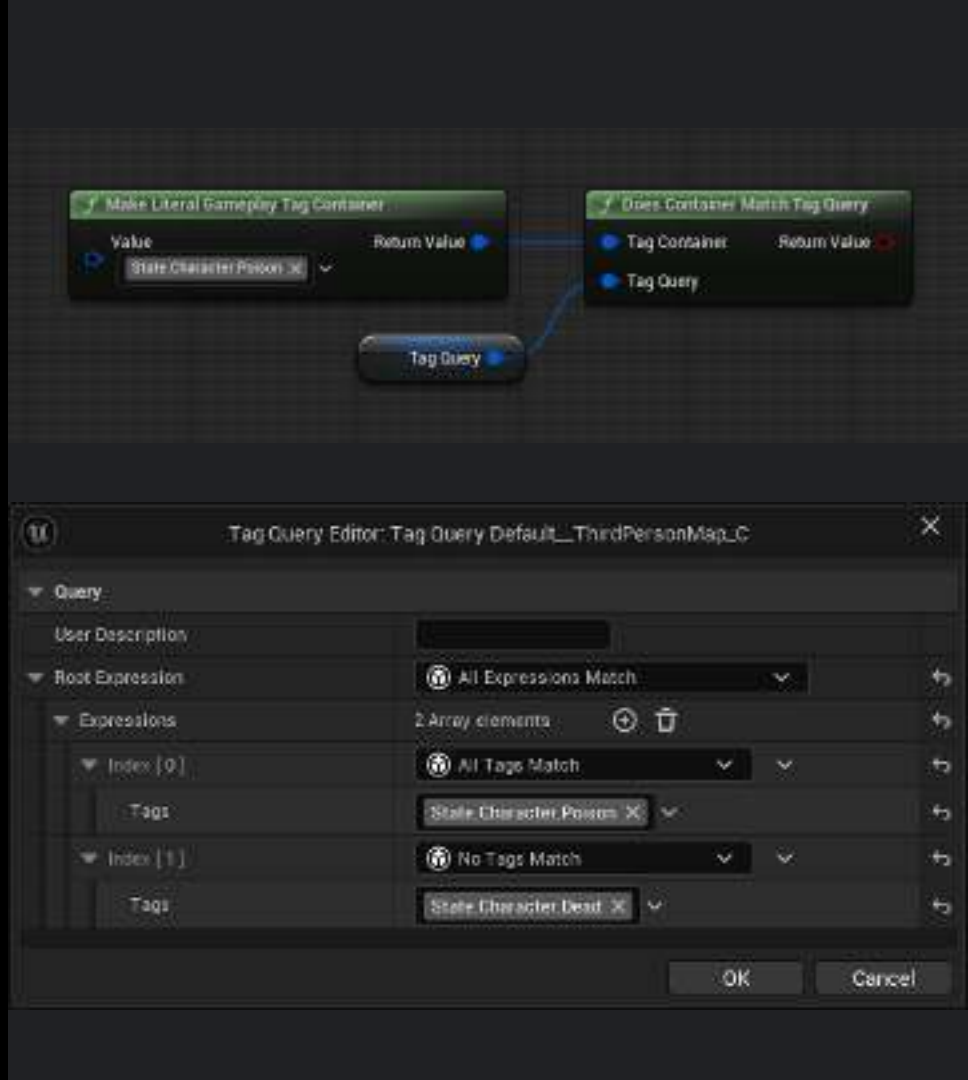
Tag Container에 어떤 것 하나라도 Gameplay Tag
명시적 포함 유무

{"Insect.Ant", "Mammal.Cat"}
.HasAnyExact({" Insect"," Bird”}) → **false**

{"Insect.Ant", "Mammal.Cat"}
.HasAnyExact({" Insect.ant“, “Bird.Sparrow”}) → **true**

Gameplay Tag Query

- 복잡한 논리적 연산을 수행하기 위해 태그를 쿼리하는 방법
- AND, OR, NOT 연산자를 사용하여 조건을 조합하고, 태그 컨테이너가 이 조건에 맞는지 여부를 True 또는 False로 반환
- 캐릭터의 상태에 대한 복합적인 쿼리가 가능
- 예제
  (캐릭터 독 중독) 이면서
  (캐릭터 죽지 않은 상태)



Gameplay Tag Query
C++ 예제

```
FGameplayTag PoisonTag =
	FGameplayTag::RequestGameplayTag(
	FName("State.Character.Poison"));

FGameplayTag DeadTag =
	FGameplayTag::RequestGameplayTag(
	FName("State.Character.Dead"));

FGameplayTagQuery TagQuery;

TagQuery.Build(
	FGameplayTagQueryExpression()
	.AllExprMatch()
	.AddExpr(
		FGameplayTagQueryExpression()
		.AllTagsMatch()
		.AddTag(PoisonTag)
	)
	.AddExpr(
		FGameplayTagQueryExpression()
		.NoTagsMatch()
		.AddTag(DeadTag)
	)
);
```

C++ Native Gameplay Tag 추가

- 태그 시스템에서 컴파일 타임에 정의된 태그로 등록할 수 있습니다.
- C++에서 정의된 태그를 사용할 수 있습니다.

.h

```
UE_DECLARE_GAMEPLAY_TAG_EXTERN(
        TAG_Mammal_Cat);
```

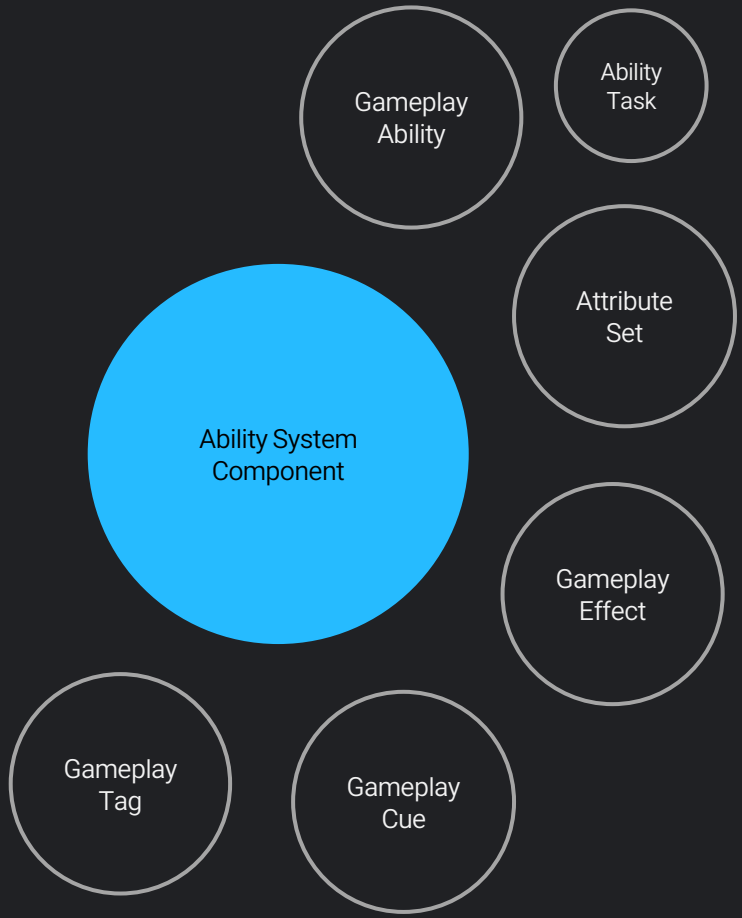
.cpp

```
UE_DEFINE_GAMEPLAY_TAG(
        TAG_Mammal_Cat,
        "Mammal.Cat");

UE_DEFINE_GAMEPLAY_TAG_COMMENT(
        TAG_Mammal_Dog,
        "Mammal.Dog",
        "Dog in the Mammal category");

UE_DEFINE_GAMEPLAY_TAG_STATIC(
        TAG_Mammal_Elephant,
        "Mammal.Elephant");
```

Ability System Component

Ability System Component

- Gameplay Ability System 의 핵심 구성 요소
- 캐릭터가 보유하는 Ability를 관리
- 캐릭터의 Attribute Set을 관리
- 캐릭터에 적용되는 Gameplay Effect 관리
- Gameplay Tag를 활용하여 상태 표현
- Ability System Component 간의 상호작용



Ability System Component 추가 방법

.h

```
class AMyActor : public AActor, public IAbilitySystemInterface
{
	UPROPERTY(VisibleDefaultsOnly, BlueprintReadOnly, Category = "Abilities")
	UAbilitySystemComponent* AbilitySystemComponent;
	virtual UAbilitySystemComponent* GetAbilitySystemComponent() const override;
}
```

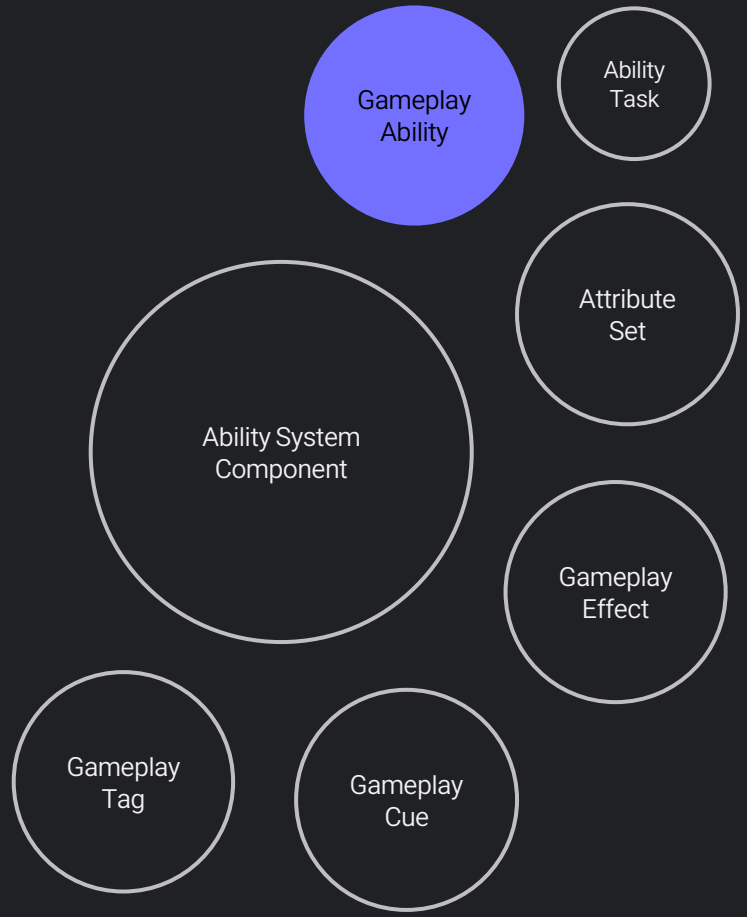
.cpp

```
UAbilitySystemComponent* AMyActor::GetAbilitySystemComponent() const
{
	return AbilitySystemComponent;
}
```

Gameplay Ability

Gameplay Ability

- 캐릭터가 수행할 수 있는 특정 행동이나 동작 정의
- Gameplay Ability 클래스를 상속하여 구현
- Enhanced Input과 커스텀하여 연동
- 활성화 시 태그 부여
- Can Activate Ability 함수를 통한 실행 가능 판단
- Ability Task를 활용한 지연되는 로직 제작 용이



Gameplay Ability
구현 가능한 능력

점프

포션 사용

검을 휘두르는 공격

총을 이용한 총알 발사

탄창 재장전

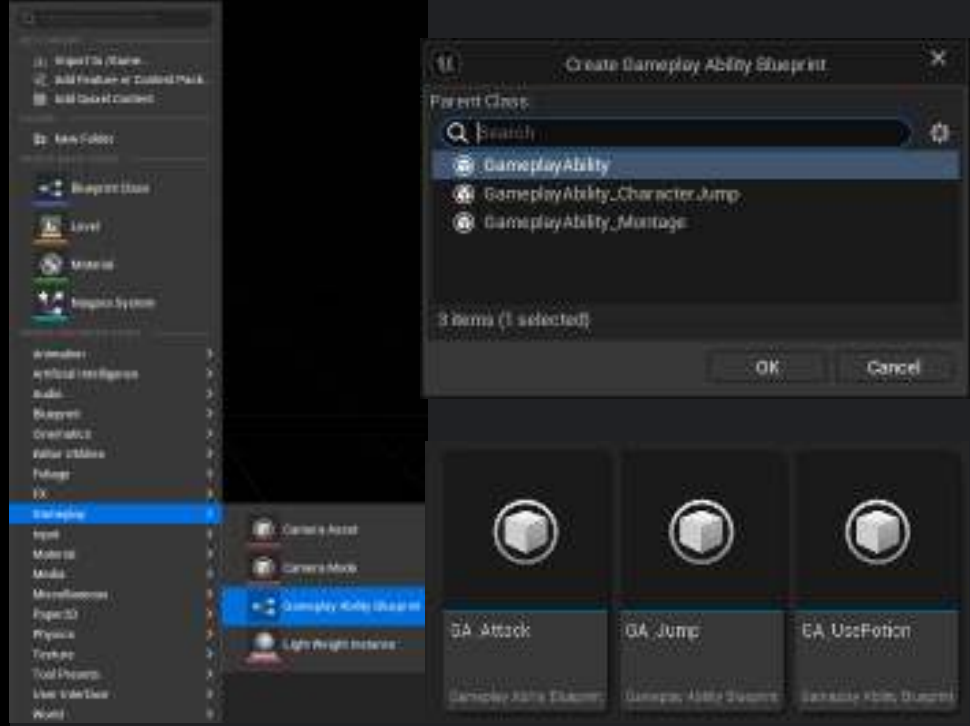
마법 사용

액터와 상호작용

Gameplay Ability 생성 방법

- Content Browser
  → Gameplay
    → Gameplay Ability Blueprint

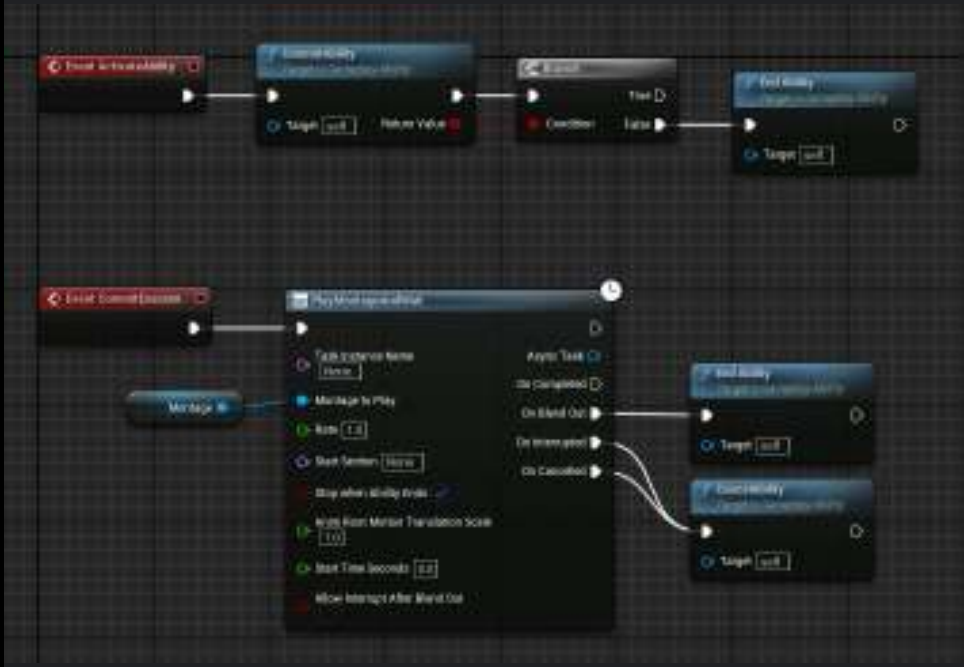
- C++ 에서는 Gameplay Ability 클래스를 상속하여 제작할 수 있습니다.




```
UCLASS()
class ABILITYSAMPLE_API UMyGameplayAbility :
        public UGameplayAbility
{
        GENERATED_BODY()
};
```

간단한 Ability 생성 예제

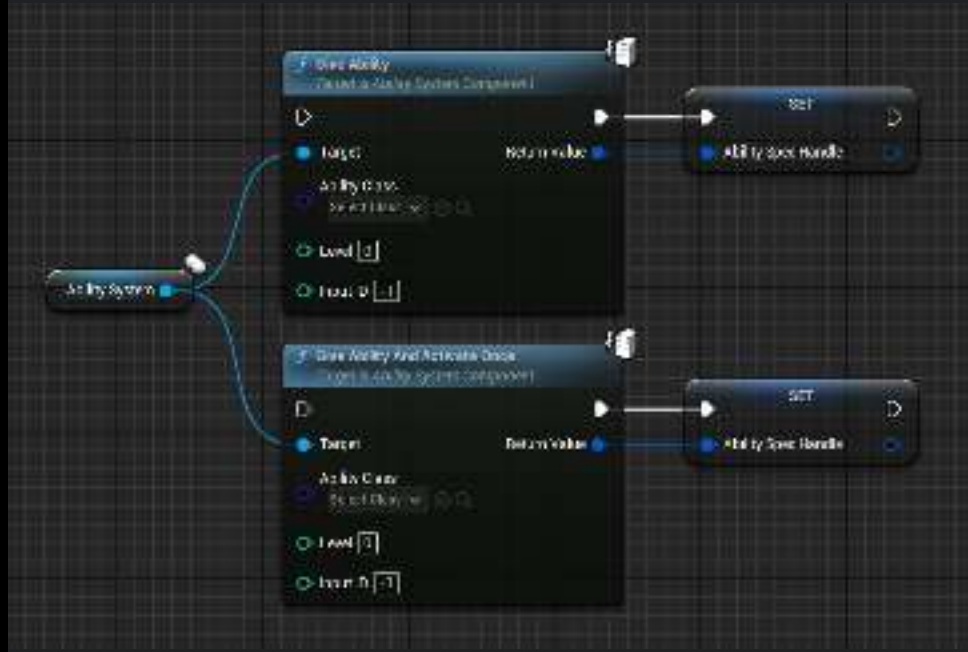
- **Event Activate Ability**
  Ability가 활성화될 때 호출
- **Commit Ability**
  Ability를 Commit하여 필요한 자원(마나, 기력 등)이 소비되고 Cooldown이 시작됩니다.
- **Play Montage And Wait**
  몽타주를 재생하고 완료 될 때까지 기다립니다.
- **EndAbility**
  Ability를 종료합니다.



Ability 부여

- **Give Ability**
  Ability System Component에 Ability Class 능력을 부여합니다.
  추가된 Ability는 Gameplay Ability Spec Handle을 반환합니다.

- **Give Ability And Activate Once**
  Ability 부여한 후 바로 실행합니다.
  만약 실행 조건을 충족하지 못하면 Ability는 자동으로 제거 됩니다.

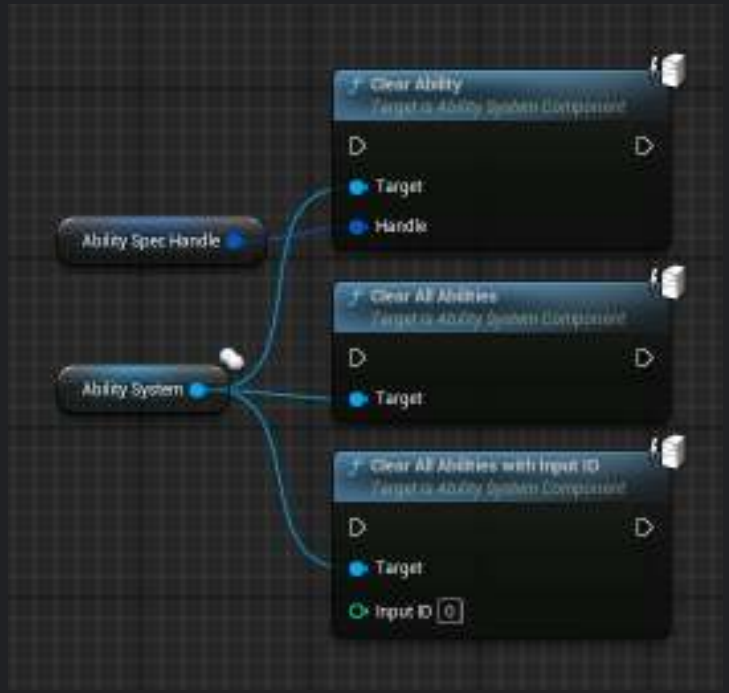


Ability 제거

- **Clear Ability**
  부여시 반환된 Gameplay Ability Spec Handle을 사용하여 지정된 Ability를 Ability System Component 에서 제거합니다.

- **Clear All Abilities**
  Ability System Component에 등록된 모든 Ability를 제거합니다.

- **Clear All Abilities with Input ID**
  입력 받은 Input ID를 사용하는 모든 Ability를 제거합니다.

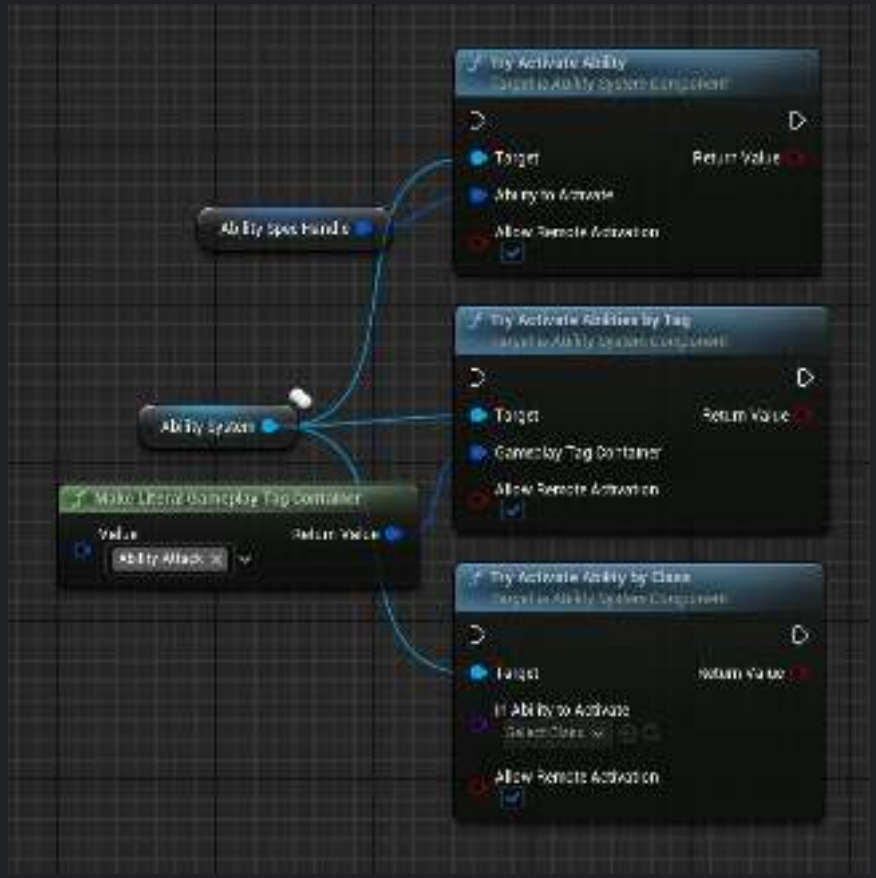


Ability 활성화

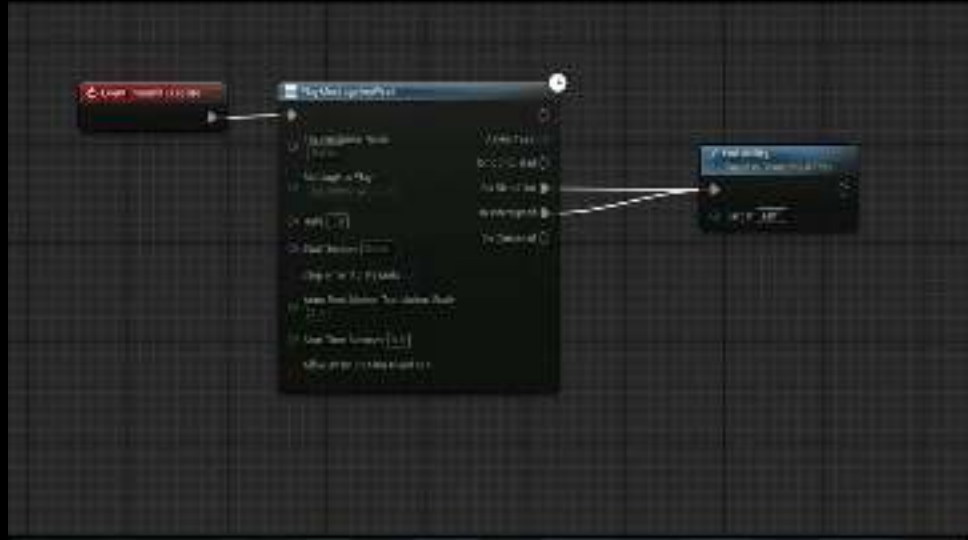
- **Try Activate Ability**
  Give 후 받은 Ability Spec Handle 을 사용하여 Ability를 활성화합니다.

- **Try Activate Abilities By Tag**
  지정된 태그를 사용하여 Ability를 활성화합니다.

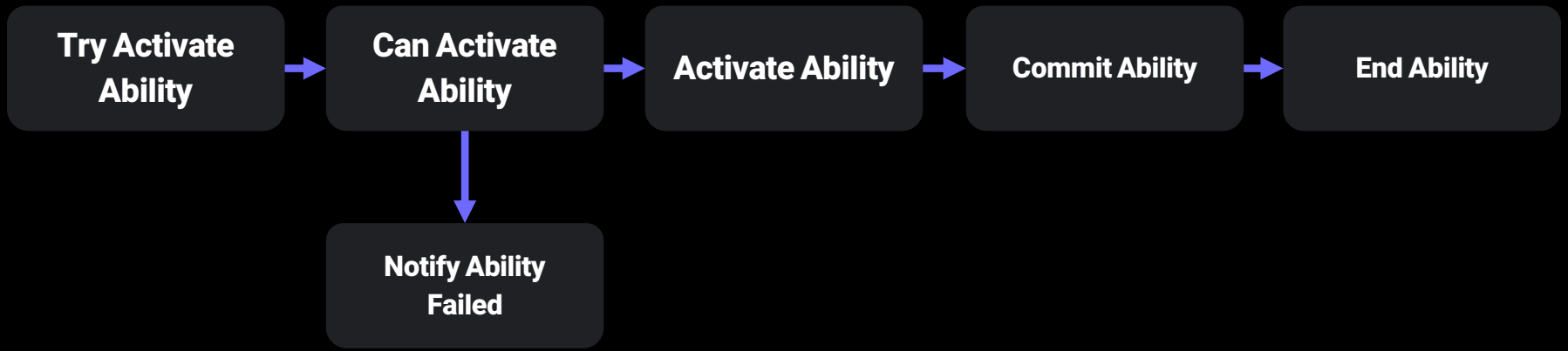
- **Try Activate Ability By Class**
  지정된 Ability 클래스를 사용하여 Ability를 활성화합니다.



간단한 Ability 실행 결과



Ability 실행 흐름도



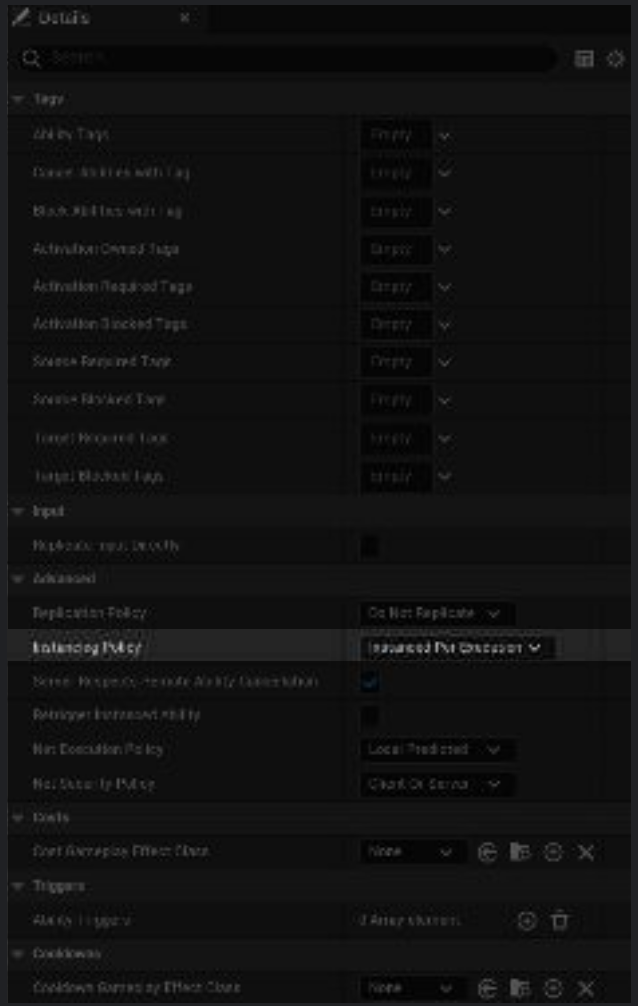
Enhanced Input 연동

Ability를 Give할 때 Input ID를 명시하여, 해당 Ability가 어떤 Input Action에 결합될지 미리 정의 한 후,
Enhanced Input의 Bind Action 콜백을 사용하여 Ability를 실행합니다.

```
void UMyAbilitySystemComponent::BindAbilityActivationToEnhancedInputComponent(
    UEnhancedInputComponent* EnhancedInputComponent, TMap<UInputAction*, int32> IndexByInputAction)
{
   if (EnhancedInputComponent)
   {
      for (const auto& Item : IndexByInputAction)
      {
         // Triggered event
         EnhancedInputComponent→BindAction(
            Item.Key, ETriggerEvent::Triggered, this, &UAbilitySystemComponent::AbilityLocalInputPressed, Item.Value);

         // Cancel event
         EnhancedInputComponent→BindAction(
            Item.Key, ETriggerEvent::Canceled, this, &UAbilitySystemComponent::AbilityLocalInputReleased, Item.Value);

         // Complete event
         EnhancedInputComponent→BindAction(
            Item.Key, ETriggerEvent::Completed, this, &UAbilitySystemComponent::AbilityLocalInputReleased, Item.Value);
      }
   }
}
```

Ability Instancing Policy

- Non Instanced
- Instanced Per Actor
- Instanced Per Execution

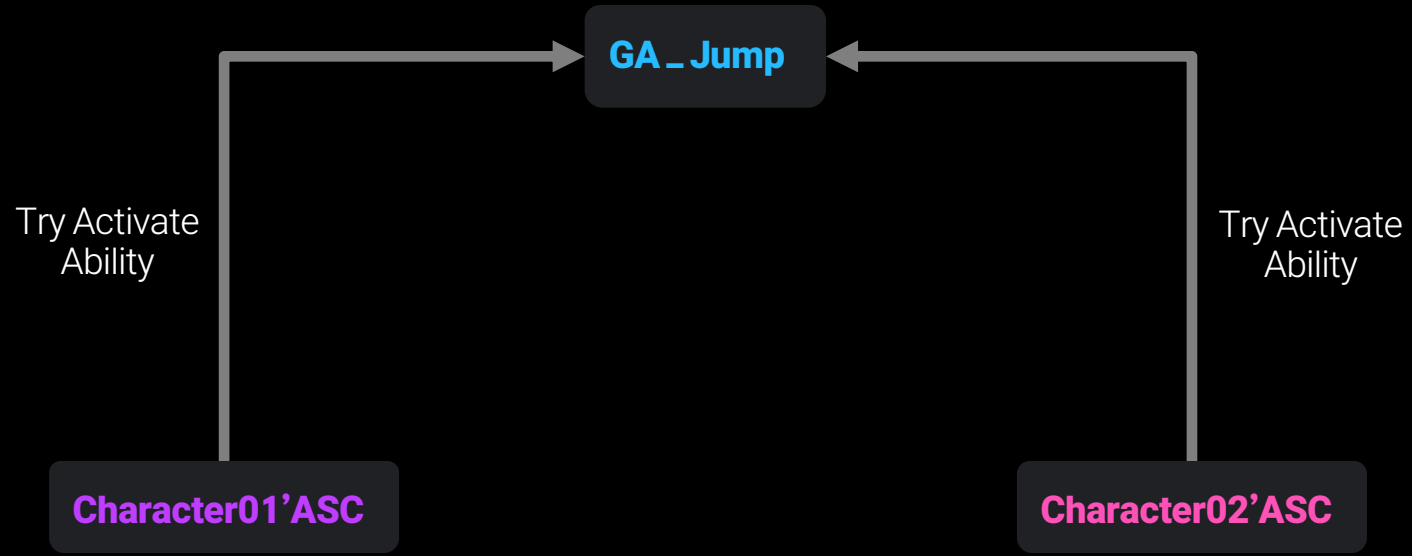


Non Instanced

어빌리티가 Instancing 되지 않고 CDO를 사용합니다.

즉, 어빌리티의 모든 실행이 동일한 클래스 인스턴스를 공유합니다.

주로 상태를 저장할 필요가 없는 간단한 어빌리티에서 사용합니다.

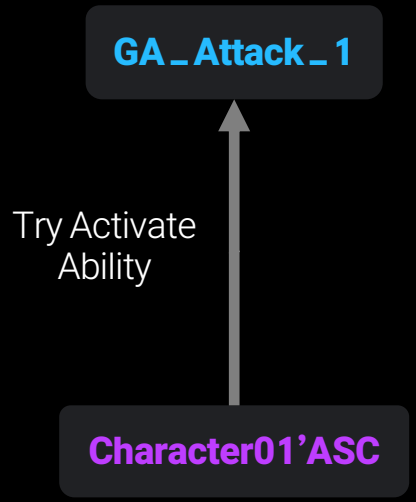


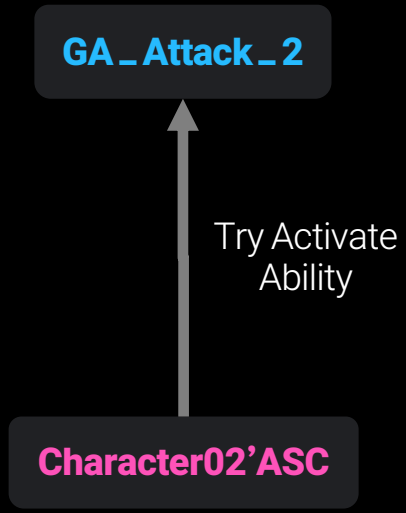
Instanced Per Actor

액터마다 하나의 어빌리티 인스턴스를 생성합니다.

즉, 동일한 액터가 어빌리티를 실행하더라도 동일한 인스턴스를 사용합니다.

어빌리티 내에서 변수를 관리해야 하는 어빌리티에 적합합니다.



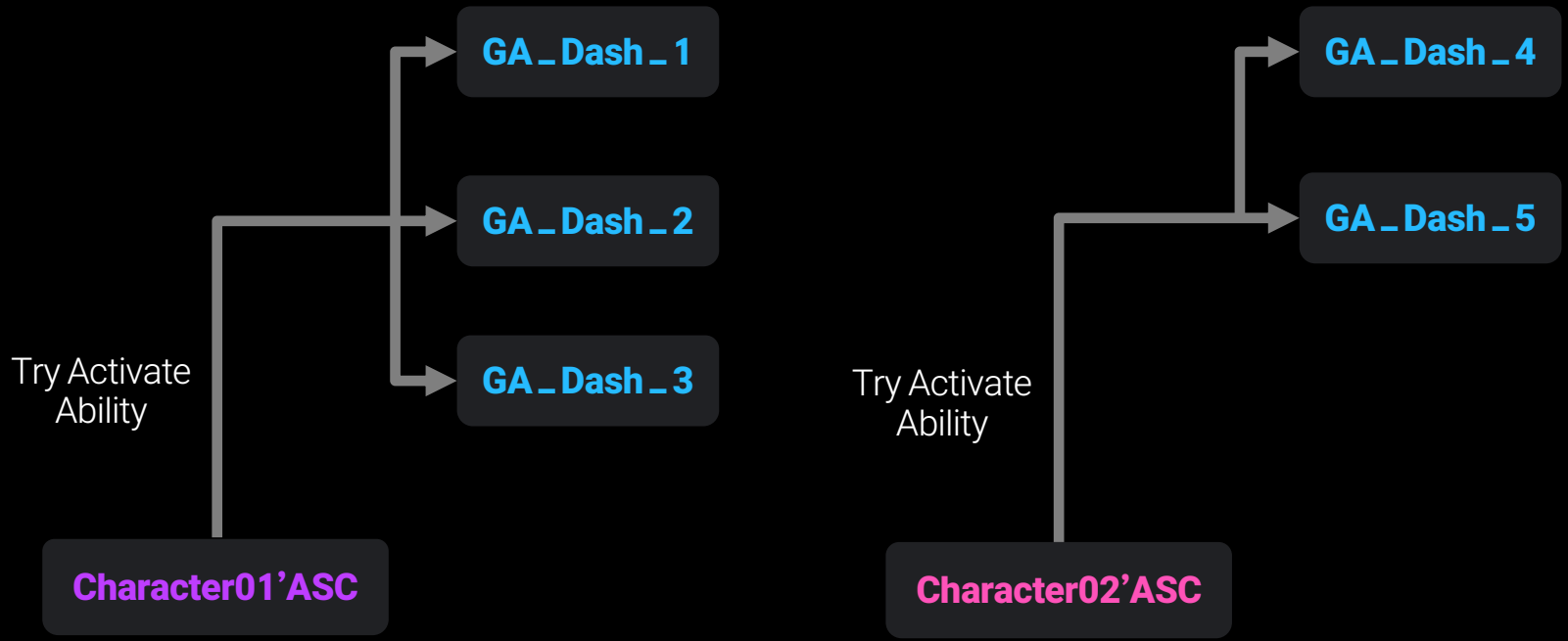


Instanced Per Execution

어빌리티가 실행될 때마다 새로운 인스턴스를 생성합니다.

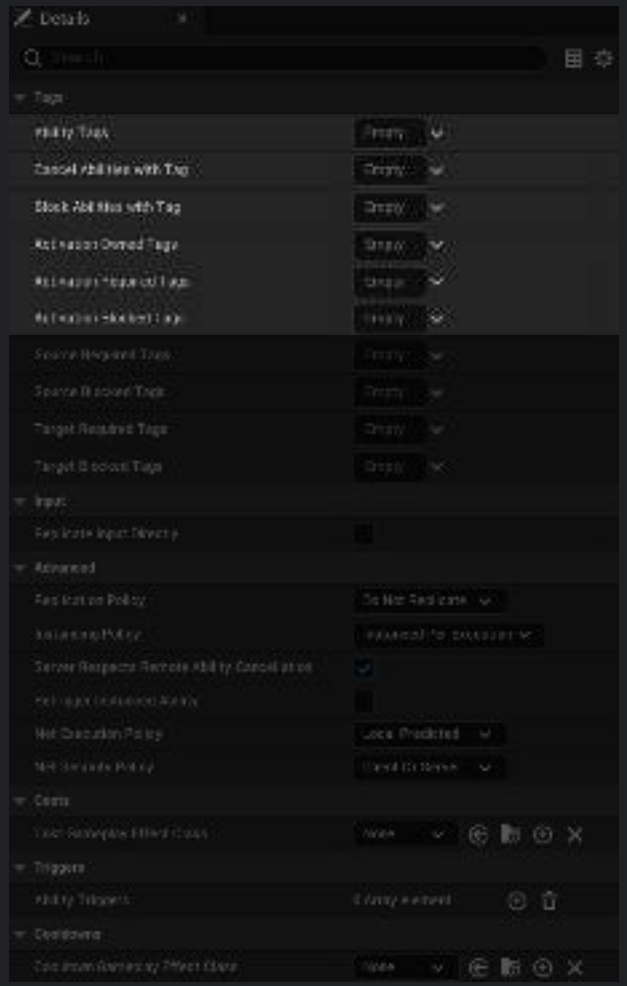
즉, 동일한 액터가 여러 번 같은 어빌리티를 실행하더라도 각 실행마다 별도의 인스턴스를 생성되고 사용합니다.

각 실행 간의 상태를 독립적으로 유지해야 하는 어빌리티에 적합합니다.



Ability 상세 항목

- Ability Tags
- Cancel Abilities With Tag
- Block Abilities With Tag
- Activation Owned Tags
- Activation Required Tags
- Activation Blocked Tags



Cancel Abilities With Tag

해당 Ability가 활성화 될 때, 지정된 태그를 가진 모든 활성화 된 Ability를 취소합니다.

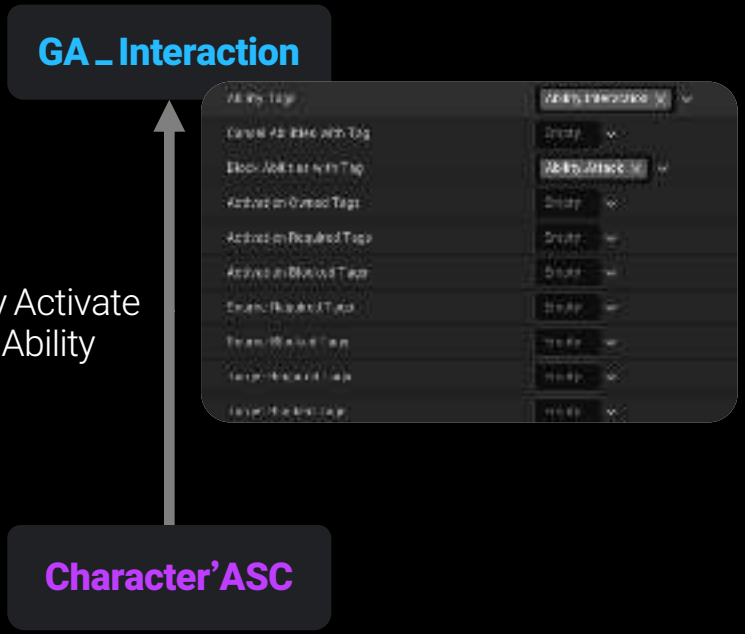
예시) “Ability.Dash” 태그를 가진 Ability가 활성화될 때, “Ability.Attack” 태그를 가진 모든 Ability를 취소


Tap [Apostrophe] to close or hold to select new Pawn (hold Shift+Apostrophe] to select local player). Use Numpad to toggle categories
Ctrl+Tilde:HUD Ctrl+Tab:DebugMessages Tab:Spectator
0.Normal 1.Net 2.BehaviorTree 3.EQS 4.Perception 5.PerceptionSystem 6.LCDebugger 9.Abilities
Debug actor: BP_LCPlayerCharacterBaseV3_C_0 ROLE_AutonomyProxy
VLog not recording on this
[CATEGORY: Abilities] Tags [Shift+1] Abilities [Shift+2] Effects [Shift+3] Attributes [Shift+4]
Owned Tags (0)
Gameplay Abilities Legend: Granted (2) Active (0)
94 Attack source: BP_LCPlayerCharacterBaseV3_C_0 level: 21
94 Dash source: BP_LCPlayerCharacterBaseV3_C_0 level: 61

Block Abilities With Tag

해당 Ability가 활성화 되어 있는 중에는 지정된 태그를 가진 Ability는 실행할 수 없습니다.

예시) “Ability.Interaction” 태그를 가진 Ability가 활성화 된 동안 “Ability.Attack” 태그를 가진 Ability 사용 차단

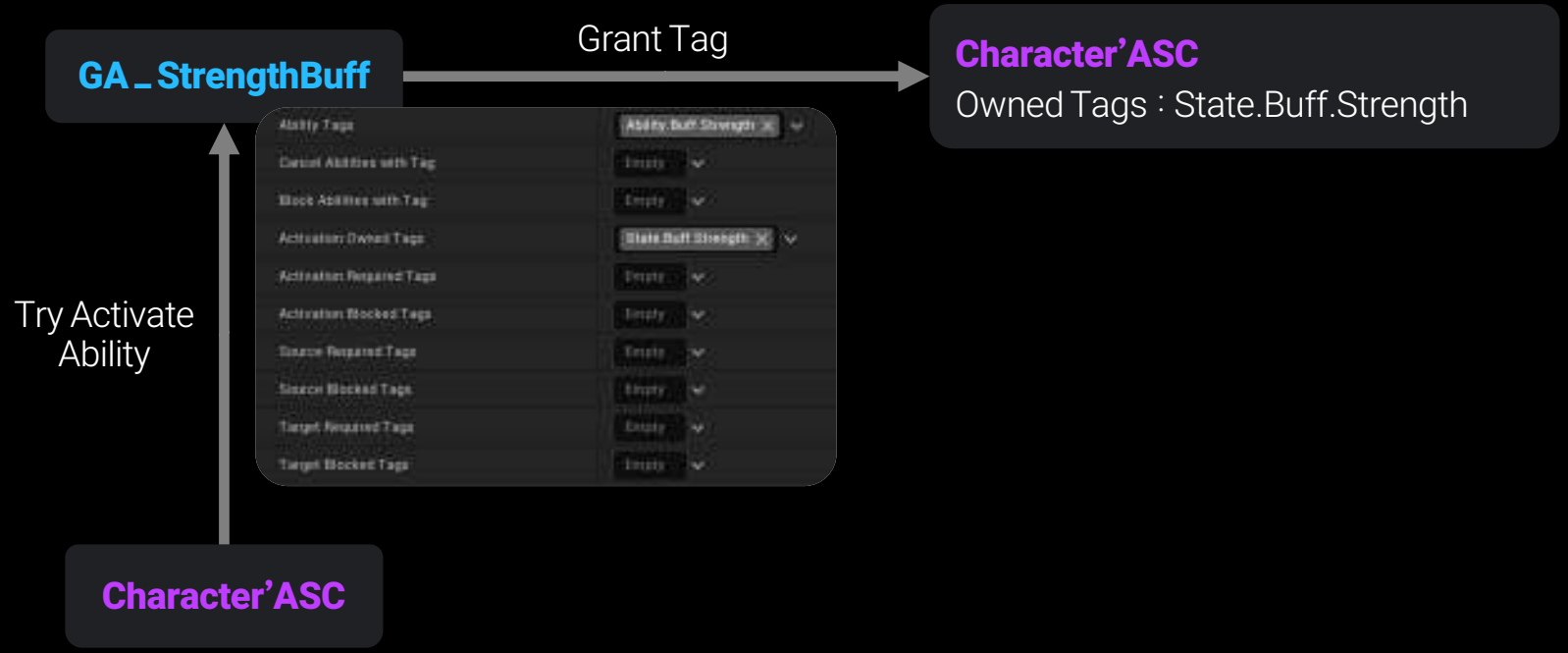



ABILITIES for avatar BP_LCPlayerCharacterBaseV3_C_0 (authority) for owner BP_PlayerState_C_0 (authority)
Showing Debug for BP_LCPlayerCharacterBaseV3_C_0. Press [PageUp] and [PageDown] to cycle between targets
LEEHOMIN 766F07BD41C
Location: V(0) Rotation: R(0)
Instigator: None Owner: BP_LCPlayerControllerV2_C_0
LEEHOMIN 766F07BD41C
Location: V(X=-253.17, Y=-1253.42, Z=-0.85) Rotation: R(Y=85.96)
Instigator: BP_LCPlayerCharacterBaseV3_C_0 Owner: BP_LCPlayerControllerV2_C_0
CONTROLLER BP_LCPlayerControllerV2_C_0 Pawn BP_LCPlayerCharacterBaseV3_C_0
STATE Playing
Tip: Use the GameplayDebugger for enhanced functionality
Owned Tags:
BlockedAbilityTags:
GA_Attack
GA_Interaction

Activation Owned Tags

Ability가 활성화될 때 소유하는 태그입니다. 이 태그는 Ability가 활성화되어 있는 동안 Ability System Component에 부여됩니다.

예시) 어빌리티를 실행하는 동안에는 힘 강화가 부여된다면 State.Buff.Strength 태그를 ASC에 추가합니다.

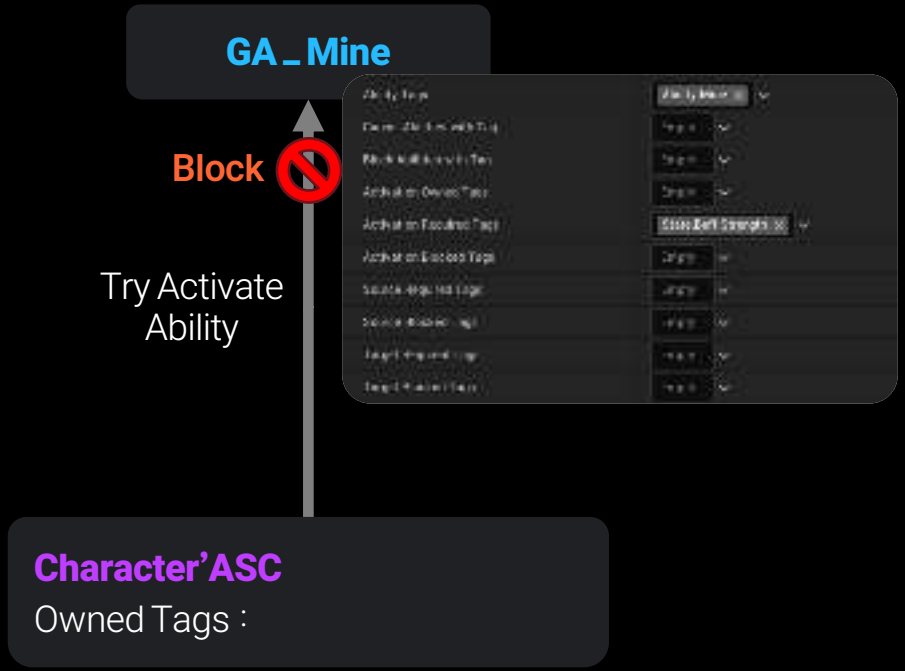


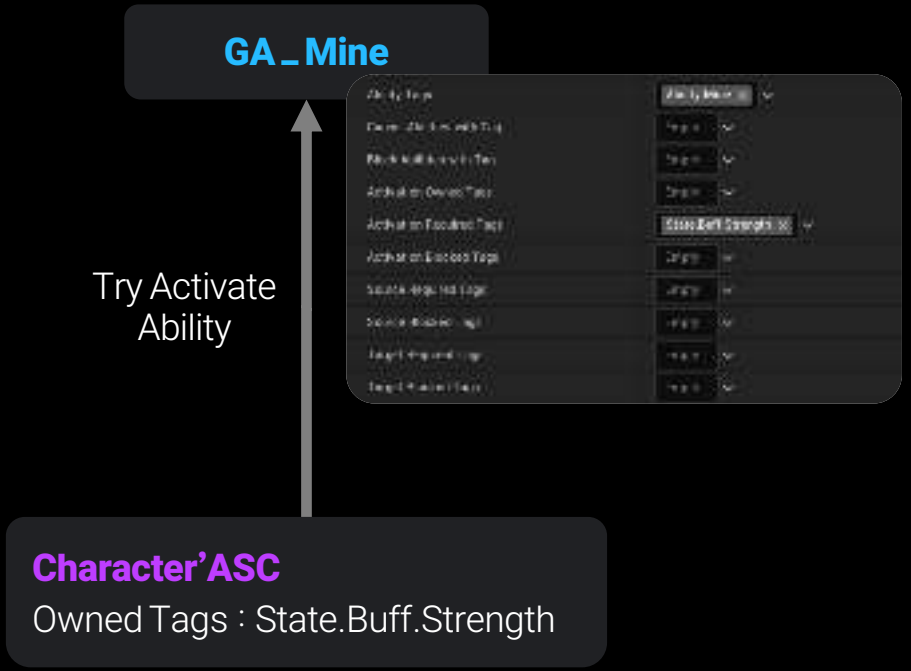

Strength Buff

Activation Required Tags

Ability를 활성화하기 위해 요구되는 태그입니다. 캐릭터가 이 태그를 가지고 있어야 해당 Ability를 사용할 수 있습니다.

예시) “State.Buff.Strength”태그가 요구되는 Ability.Mine는 힘 버프 상태에서만 활성화할 수 있습니다.

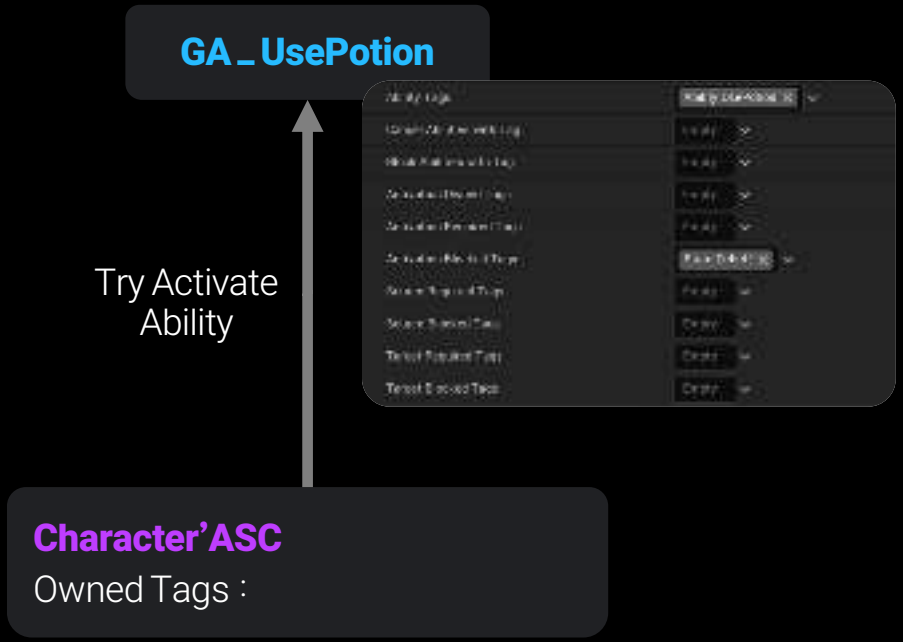


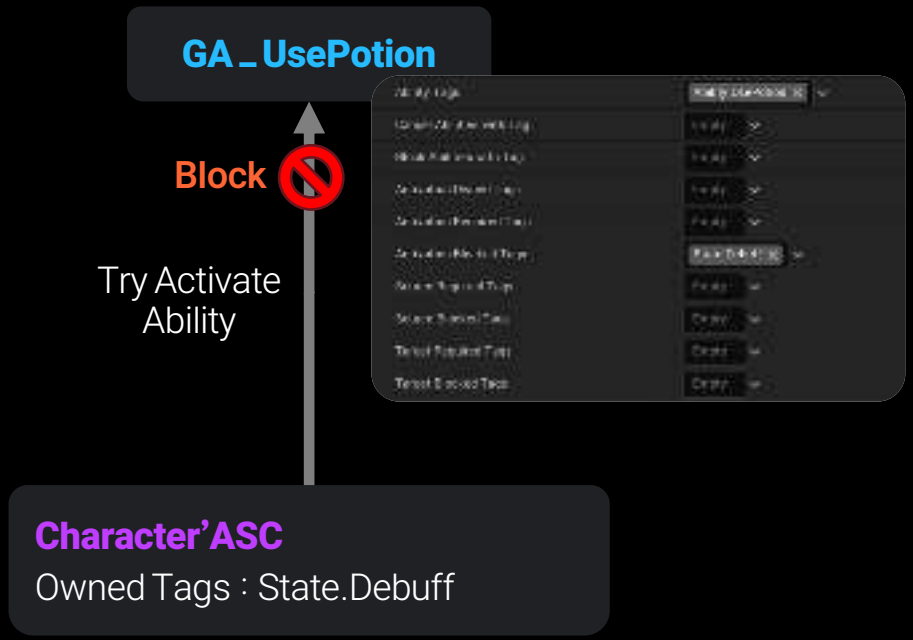


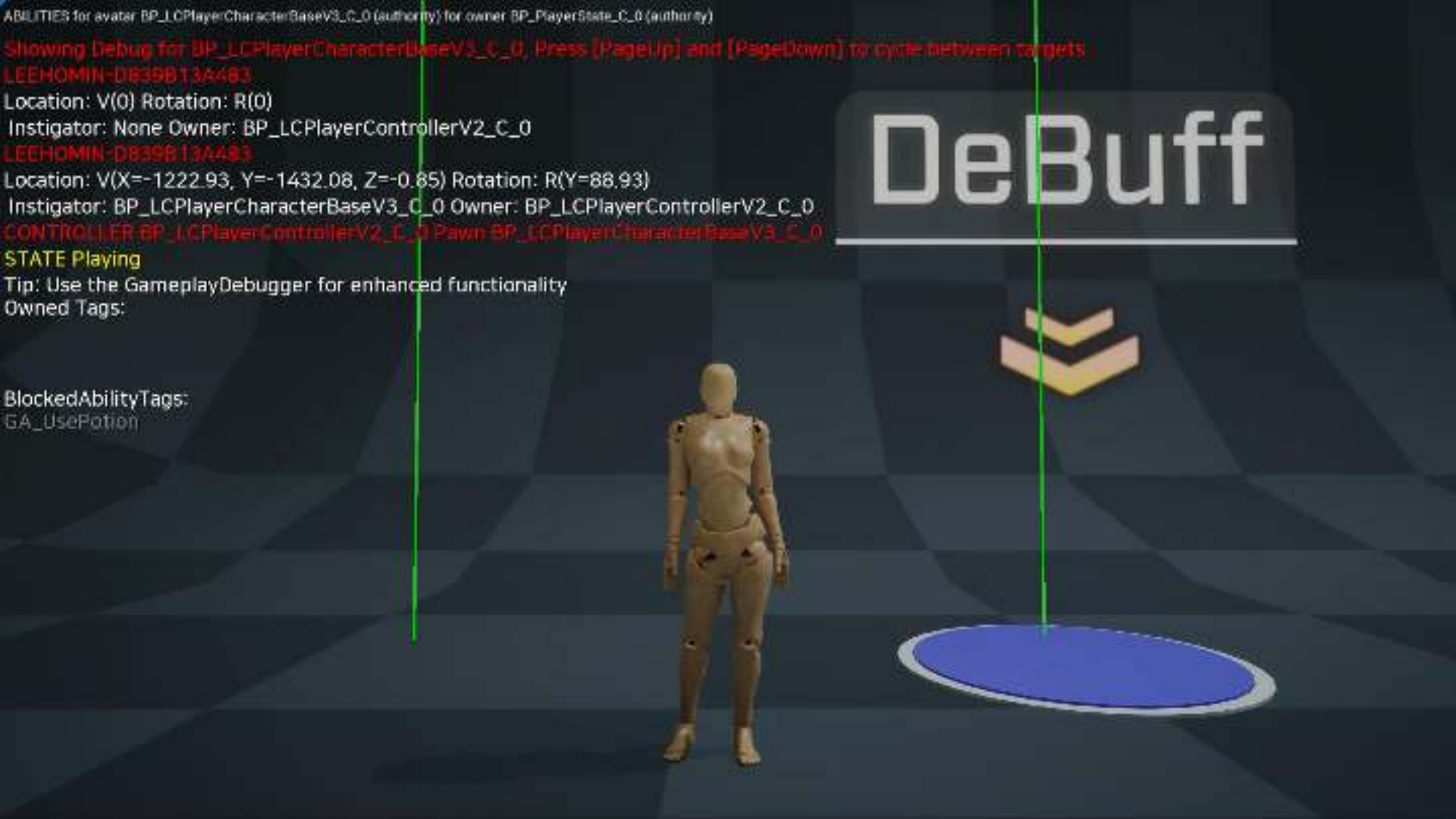

ABILITIES for avatar BP_LCPlayerCharacterBaseV3_C_0 (authority) for owner BP_PlayerState_C_0 (authority)
Showing Debug for BP_LCPlayerCharacterBaseV3_C_0. Press [PageUp] and [PageDown] to cycle between targets
LEEHOMIN-B2A29525485
Location: V(0) Rotation: R(0)
Instigator: None Owner: BP_LCPlayerControllerV2_C_0
Location: V(X=-734.07, Y=-408.67, Z=-0.85) Rotation: R(Y=-161.63)
Instigator: BP_LCPlayerCharacterBaseV3_C_0 Owner: BP_LCPlayerControllerV2_C_0
STATE Playing
Tip: Use the GameplayDebugger for enhanced functionality
Owned Tags:
BlockedAbilityTags:
Mine

Activation Blocked Tags

Ability를 활성화하는 것을 차단하는 태그입니다. 캐릭터가 이 태그를 가지고 있으면 해당 Ability를 활성화 할 수 없습니다.
예시) “State.Debuff” 태그가 있는 동안에는 Ability.UsePotion를 사용할 수 없습니다.





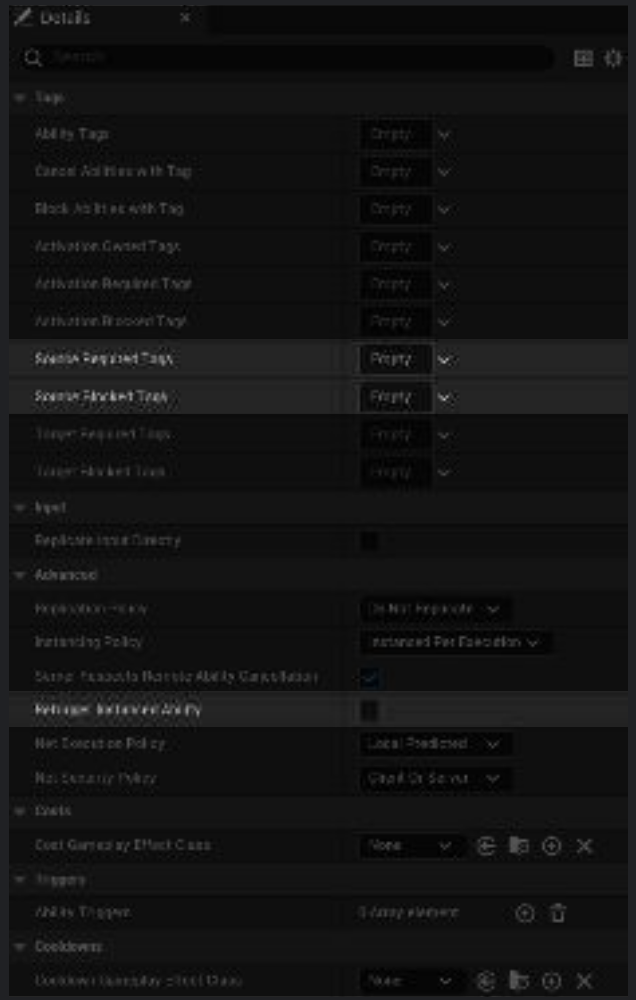

ABILITIES for avatar BP_LCPlayerCharacterBaseV3_C_0 (authority) for owner BP_PlayerState_C_0 (authority)
Showing Debug for BP_LCPlayerCharacterBaseV3_C_0. Press [PageUp] and [PageDown] to cycle between targets
LEEHOMIN-DESKTOP-D839B13A483
Location: V(0) Rotation: R(0)
Instigator: None Owner: BP_LCPlayerControllerV2_C_0
LEEHOMIN-DESKTOP-D839B13A483
Location: V(X=-1222.93, Y=-1432.08, Z=-0.85) Rotation: R(Y=88.93)
Instigator: BP_LCPlayerCharacterBaseV3_C_0 Owner: BP_LCPlayerControllerV2_C_0
CONTROLLER BP_LCPlayerControllerV2_C_0 Pawn BP_LCPlayerCharacterBaseV3_C_0
STATE Playing
Tip: Use the GameplayDebugger for enhanced functionality
Owned Tags:
BlockedAbilityTags:
GA_UsePotion
DeBuff

Ability 상세 항목

Source Required Tags
Event에 의한 트리거될 때만 유효합니다.
Event의 Instigator Tags가 모든 태그를 가지고 있을 경우에만 Ability를 활성화할 수 있습니다.

Source Blocked Tags
Event에 의한 트리거될 때만 유효합니다.
Event의 Instigator Tags가 지정된 태그 중 하나라도 가지고 있을 경우 Ability가 차단됩니다.

Retrigger Instanced Ability
Instanced Per Actor를 사용할 경우, Ability가 활성화 되어 있으면 종료 시키고 다시 활성화 시킬 수 있습니다.



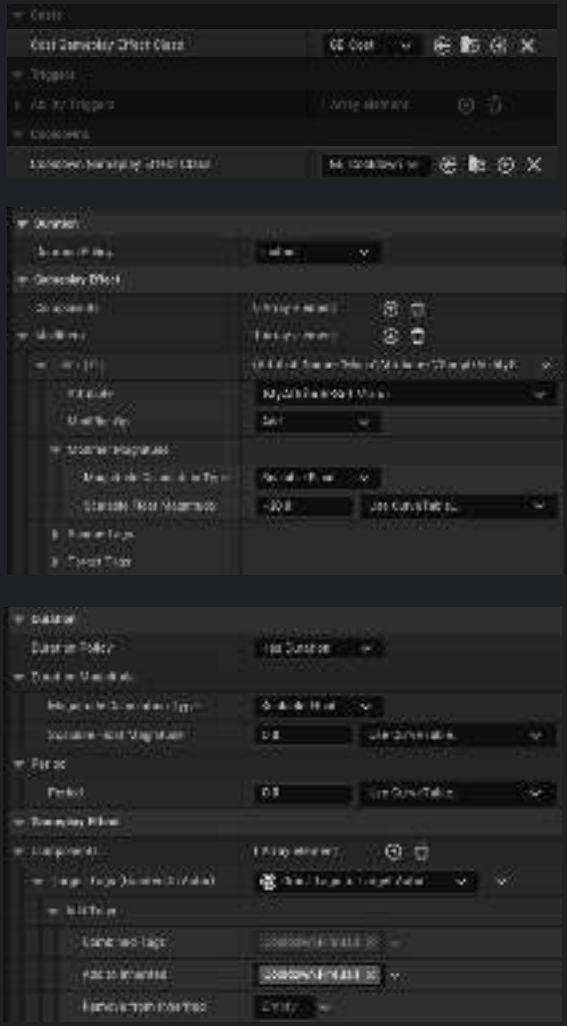
Cost & Cooldown

Cost Gameplay Effect Class

“Instant”GE를 적용하여 Commit Cost 호출 시 Attribute가 감소되도록 할 수 있습니다.
예시) 파이어 볼 사용 시 마나 30 감소

Cooldown Gameplay Effect Class

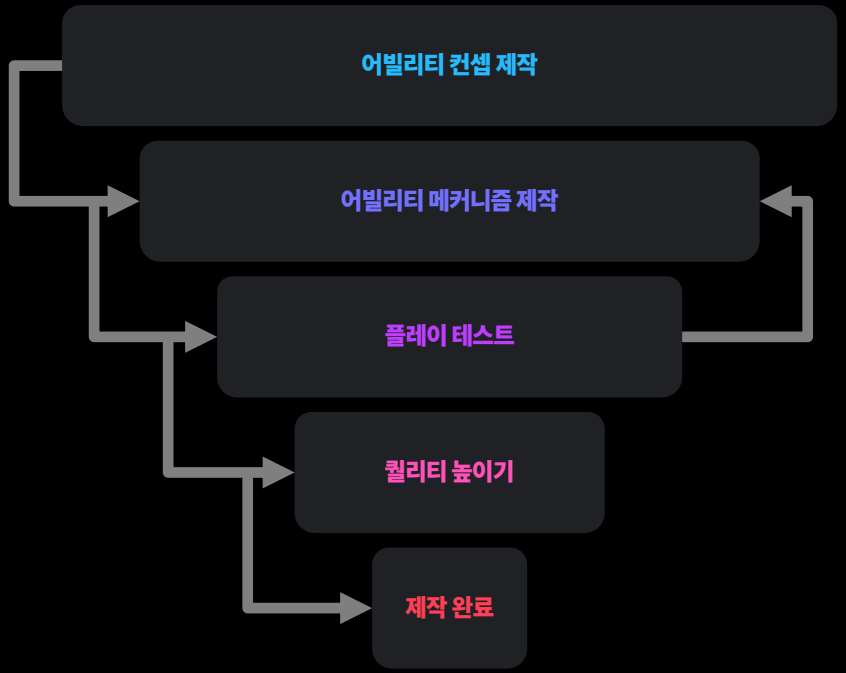
“Has Duration”GE를 적용하여 Commit Cooldown 호출 시 태그를 부여 합니다.
예시) 파이어볼 사용 시 “Cooldown.FireBall” 태그를 ASC에 부여하여 5초 동안 사용하지 못하도록 합니다.



게임플레이 이벤트로 트리거

- Send Gameplay Event To Actor를 사용하여 Data Payload를 포함하여 Ability 활성화 할 수 있습니다.
- Event로 트리거 되는 경우에는 Activate Ability From Event를 재구현해야 합니다.
- Gameplay Ability에 Trigger Tag가 Event Tag와 일치하는 태그를 포함해야 합니다.
- Wait Gameplay Event를 사용하여 이벤트를 받을 수 있습니다.

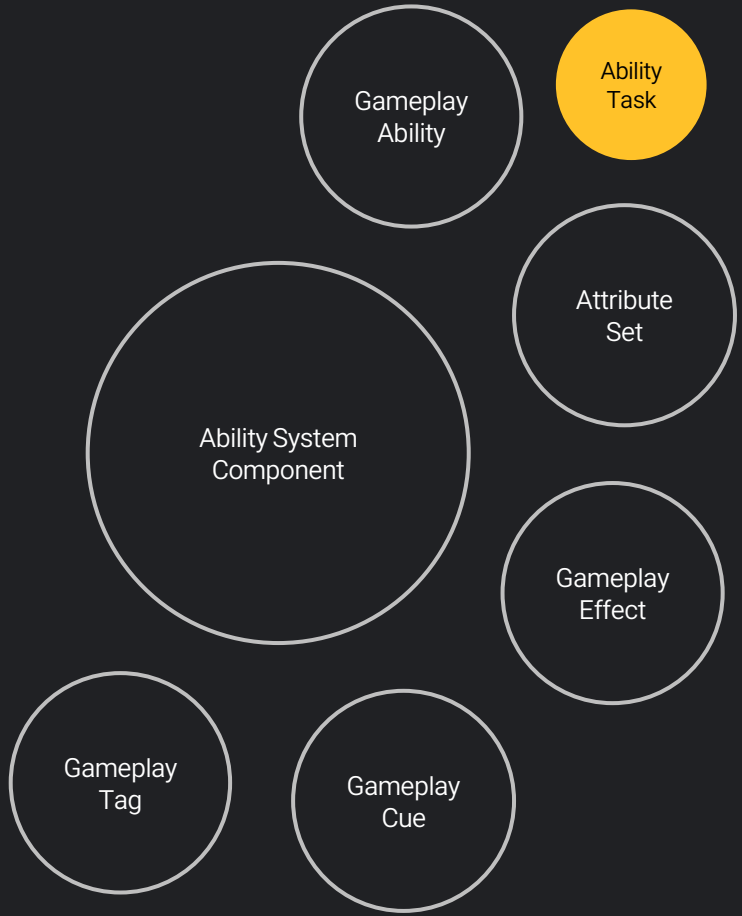
어빌리티 제작 과정



Ability Task

Ability Task

- 게임플레이 흐름 제어
- 코드 구조 간결화
- 유연한 게임플레이 구현



Ability Task 를 사용하는 이유

Ability 관리 측면에서의 효율성

Ability가 종료될 때, 해당 Ability 내부의 모든 Ability Task를 종료함으로써 안정적으로 관리 효율성을 높입니다.

게임 플레이 로직의 단순화

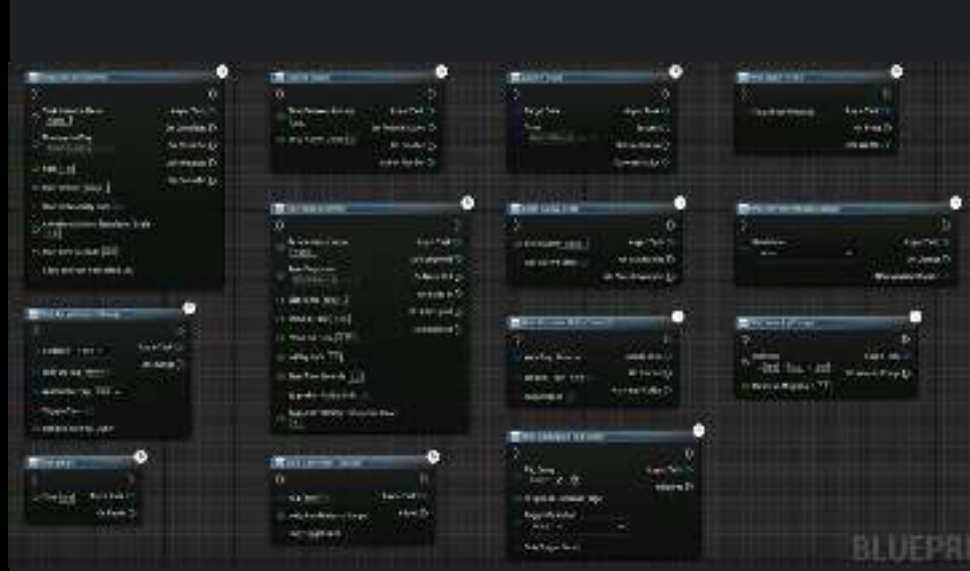
게임 플레이 로직에서 Set Timer를 사용한다면 중간에 끊길 경우 Timer를 Reset해야 하는 번거로움을 해결 해줍니다. Wait Delay (Ability Task) 를 대체 사용 가능합니다.

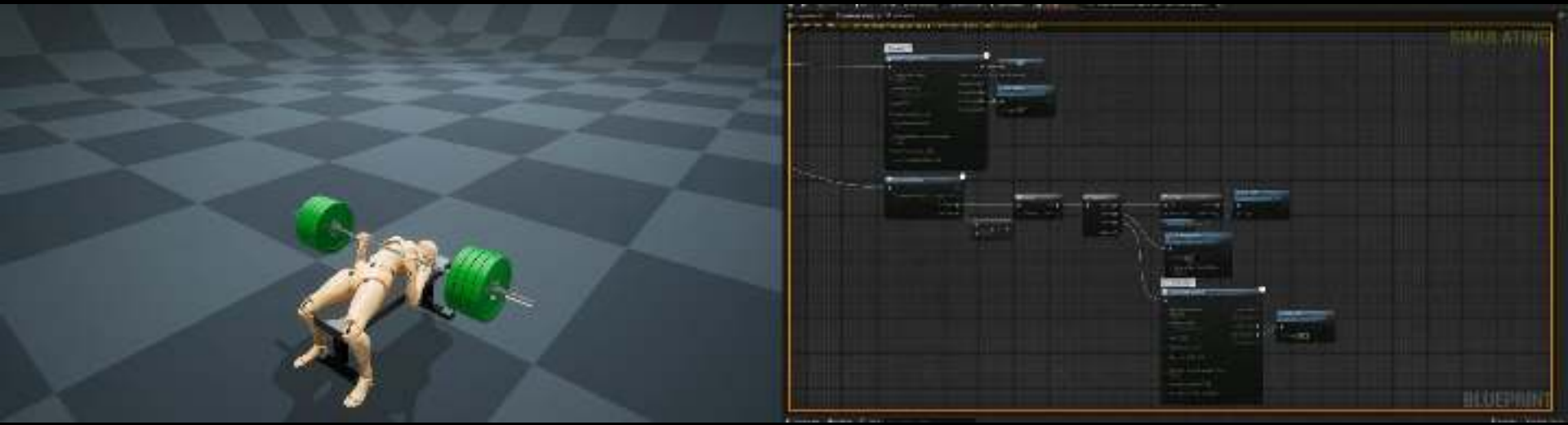
기능 모듈화 및 재사용성 향상

기능이 모듈화되어 처음에는 제작에 시간이 걸릴 수 있지만, 적절한 Ability Task를 제작해 둔다면 재사용성이 높아져 제작 효율성이 높아질 수 있습니다.

Ability Task 제작 방법

- 엔진 이미 제작된 다양한 Ability Task가 포함되어 있습니다.
- C++에서 Ability Task 클래스를 상속 받아 새로운 Task를 제작할 수 있습니다.
- Should Broadcast Ability Task Delegates() 함수를 적절히 확인하고 구현해야 합니다. 태스크가 이벤트를 브로드 캐스트 가능한 상태인지를 결정하는 함수입니다.

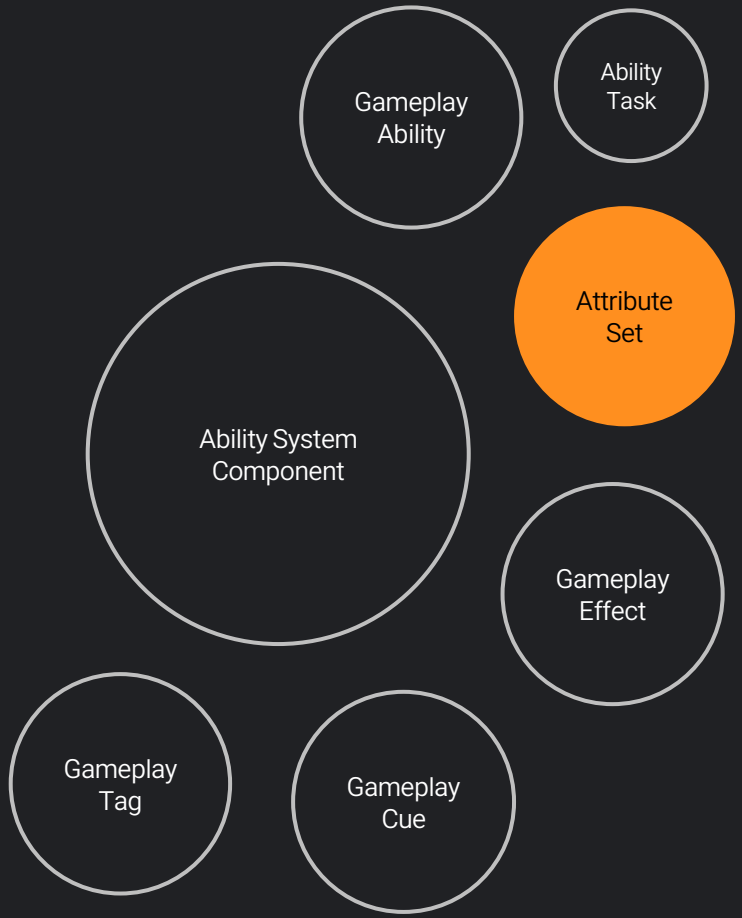



SIMULATING
BLUEPRINT

Attribute Set

Attribute Set

- 캐릭터의 스탯 모음
- 게임 플레이와 관련된 float 값을 계산 및 수정에 용이
- 아이템이나 스킬로 인한 능력치 변화 쉽게 관리
- FGameplay Attribute Data의 모음
- Attribute 변화 감지 및 처리를 위한 함수 제공



Attribute Set 정의 및 구성

- FGameplay Attribute Data 구조체를 사용하여 간단하게 필요한 Attribute 추가 가능
- 보편적으로 Max Health와 같은 최대 상한 Attribute를 사용하는 것이 필수는 아니지만 권장됩니다.
- 하나의 큰 Attribute Set를 만들 수도 있지만, 대표 그룹을 나누어 여러 개의 모듈 형태로 구성 할 수 있습니다.
- 하나의 Ability System Component가 다수의 Attribute Set을 가질 수 있지만, 각 Attribute Set의 클래스는 모두 달라야 합니다.

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

UCLASS()
class ABILITYSAMPLE_API UMyAttributeSet : public UAttributeSet
{
    GENERATED_BODY()

protected:
    UPROPERTY(BlueprintReadOnly, Category = "Health")
    FGameplayAttributeData Health;

    UPROPERTY(BlueprintReadOnly, Category = "Health")
    FGameplayAttributeData MaxHealth;

public:
    ATTRIBUTE_ACCESSORS(UMyAttributeSet, Health)
    ATTRIBUTE_ACCESSORS(UMyAttributeSet, MaxHealth)
};
```

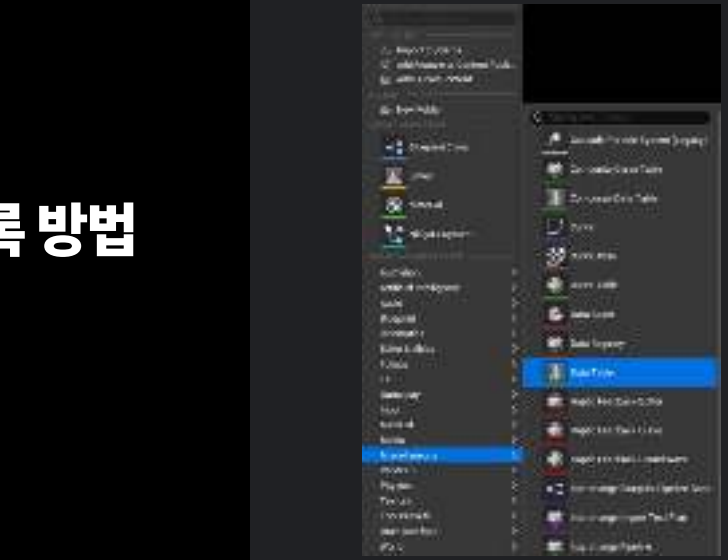
Attribute 초기 데이터 등록 방법

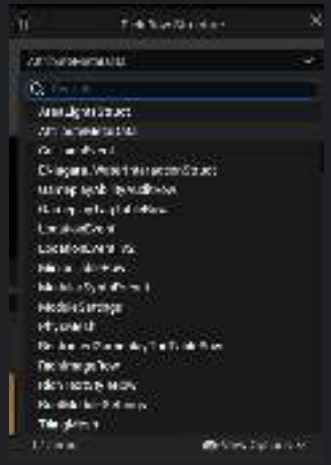
1. Contents Brower
   → Miscellaneous
   → Data Table

2. Attribute Meta Data 선택

3. Attribute Set에서 추가한 Attribute의 값 추가
   예시) MyAttribute.Health

4. Ability System Component의 Default Starting Data에 추가





Attribute Set C++ 추가 방법

- 액터의 블루프린트에서 Attribute Set과 초기값 데이터 테이블을 추가하였다면 코드 추가할 필요 없음
- 블루프린트에서 Attribute Set을 추가한 경우에는 ASC의 On Register 함수에서 자동으로 추가
- 생성자에 추가된 Attribute Set은 ASC의 Initialize Component 함수에서 추가
- AddSet 함수를 사용하여 메뉴얼하게 추가할 수 있습니다.

생성자에서 Attribute Set 추가

```
AAbilitySampleCharacter::AAbilitySampleCharacter()
{
    UMyAttributeSet* Attributes =
        CreateDefaultSubobject<UMyAttributeSet>(TEXT("AttributesSet"));
}
```

BeginPlay 함수에서 추가

```
void AAbilitySampleCharacter::BeginPlay()
{
    // Call the base class
    Super::BeginPlay();

    const UMyAttributeSet* AttributeSet = AbilitySystemComponent->AddSet<UMyAttributeSet>();
}
```

Attribute Set 값 확인


Tap [Apostrophe] to close or hold to select new Pawn Hold [Shift+Apostrophe] to select local player Use Numpad to
Ctrl+Tilde HUD Ctrl+Tab DebugMessages Tab Spectator
0 Navmesh 1 AI 2 Behavior Tree 3 EQS 4 Perception 5 PerceptionSystem 6 UIDebugger 9 Abilities
Debug actor: BP_Sandbox_V2_C_2 [ROLE_Authority]
VLog: not recording to file
[CATEGORY: Abilities] Tags [Shift+One] Abilities [Shift+Two] Effects [Shift+Three] Attributes [Shift+Four]
Attributes [4]
Damage: 0
MaxHealth: 200
DamageTaken: 0
Health: 200

Base Value

- 값이 일시적으로 변경되어도 유지되는 원본 값
- 버프나 디버프와 같은 효과에 의해 값이 변경되지 않고 유지됩니다.
- 보편적으로 “Instant” Gameplay Effect에 의해 변경됩니다.

Current Value

- Base Value에 효과로 인한 임시 변경이 적용된 현재 값
- 게임 플레이 중 버프나 디버프 등의 효과가 적용되어 변경됩니다.
- 보편적으로 “Has Duration”, “Infinite” Gameplay Effect에 의해 변경 됩니다.
- 실제 게임 로직에 사용되는 값

디버프로 인한 Attribute 변경

Attribute Set 함수

Pre Gameplay Effect Execute
"Instant" Gameplay Effect가 적용되기 직전에 호출
재구현하여 값 변경을 취소하여 무효화 시킬 수 있습니다.

Pre Attribute Base Change
Attribute의 Base Value값이 변경되기 전에 호출
Base Value 값을 제한할 때 유용함

Pre Attribute Change
Attribute의 Current Value가 변경되기 전에 호출
값을 제한할 때 유용함

Post Attribute Change
Attribute의 Current Value가 변경된 후 호출

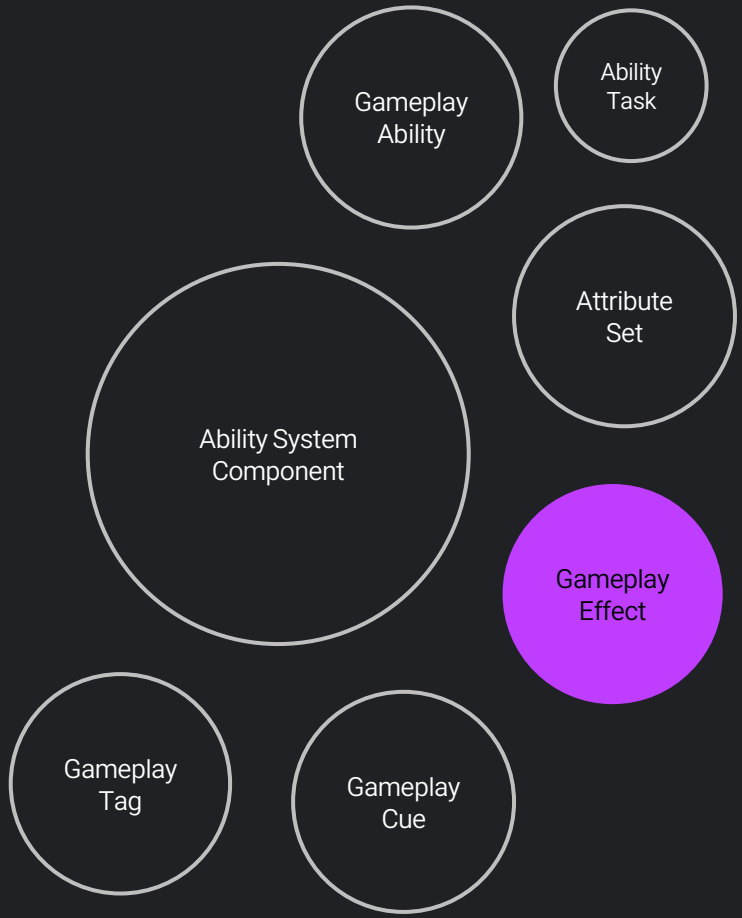
Post Attribute Base Change
Attribute의 Base Value가 변경된 후 호출

Post Gameplay Effect Execute
"Instant" Gameplay Effect가 호출된 후 호출
추가적인 로직 제가에 유용한 함수
Meta Attribute(임시값)를 활용한 Attribute 계산

Gameplay Effect

Gameplay Effect

- Attribute 변경
- 캐릭터의 버프, 디버프 상태 부여
- 일정 시간 동안 지속적인 피해
- Damage Calc를 이용한 대미지 적용
- 동일한 효과를 Stack으로 관리



기본 대미지 로직

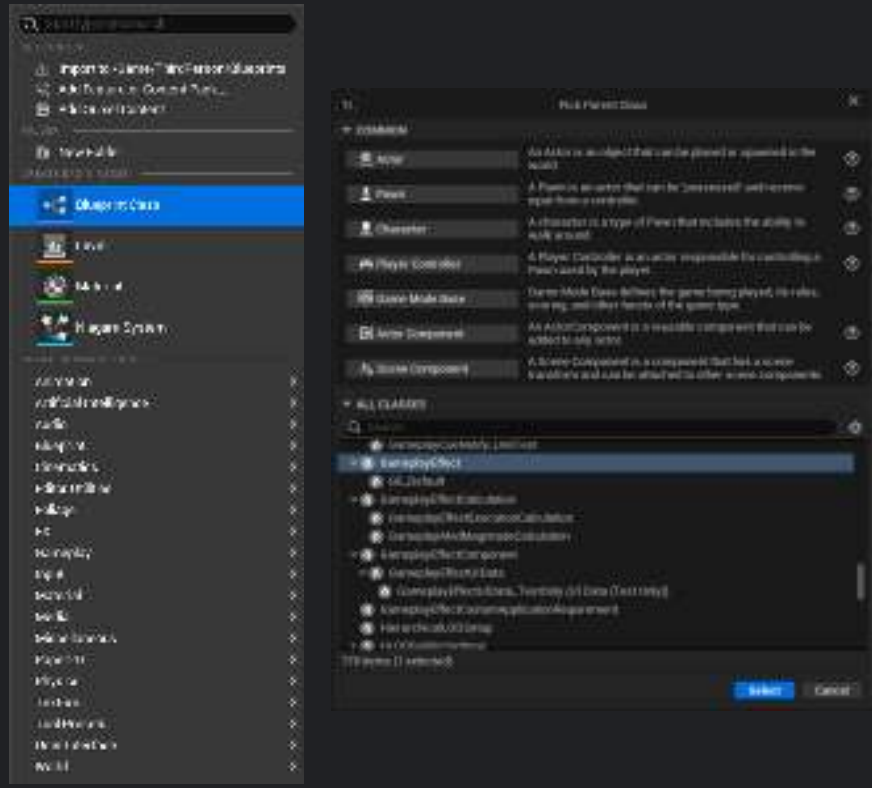
- Actor클래스의 Take Damage를 재구현하여 대미지 처리 로직 구현
- Apply Damage, Apply Point Damage 이벤트를 통해 전달
- 단순한 대미지 처리 로직을 제공하여 빠르게 적용
- 고도로 커스터마이징된 대미지 계산이나 복잡한 상태 효과 적용에 어려움
- Damage Type 클래스 사용

GAS 대미지 로직

- Ability System Component를 사용하여 대미지 처리
- Apply Gameplay Effect To Self 함수를 사용하여 대미지 전달
- 다소 높은 학습 곡선
- 커스터마이징 가능하여 복잡한 대미지 계산을 보다 쉽게 제작 가능
- Gameplay Effect 클래스 사용

Gameplay Effect 생성 방법

- Contents Browser → Blueprint Class
- Gameplay Effect 선택



Gameplay Effect 5.3~ 변경점

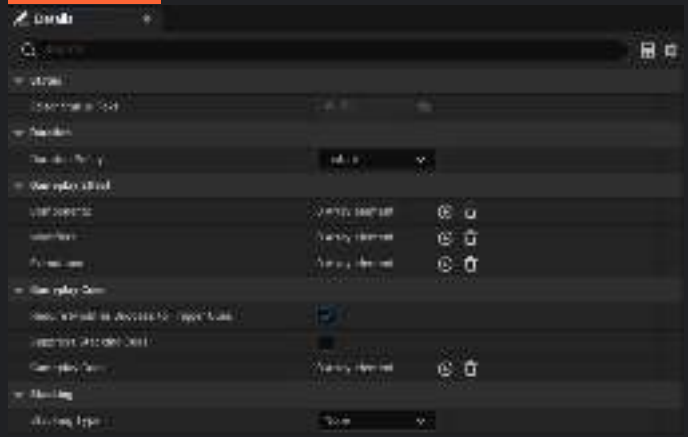
다자인 방식 변경
모놀리식 클래스 디자인 → 컴포넌트 디자인 방식

Gameplay Effect Component
GEComponent를 상속하여 원하는 기능을 모듈 방식으로 구현 가능

UX 개선
불필요한 정보를 덜 볼 수 있어서 작업에 용이

~ 5.2

5.3 ~

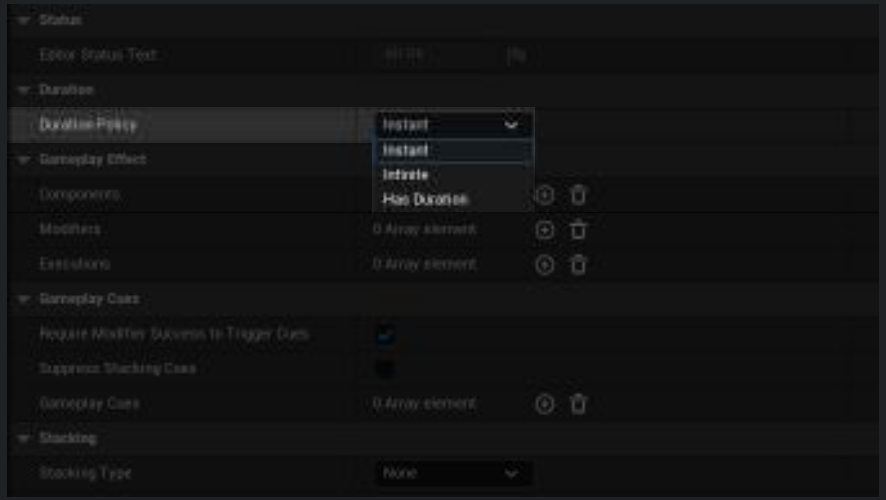


Gameplay Effect 적용 및 제거

- Apply Gameplay Effect Spec To Self
- Apply Gameplay Effect To Self
- Remove Active Gameplay Effect

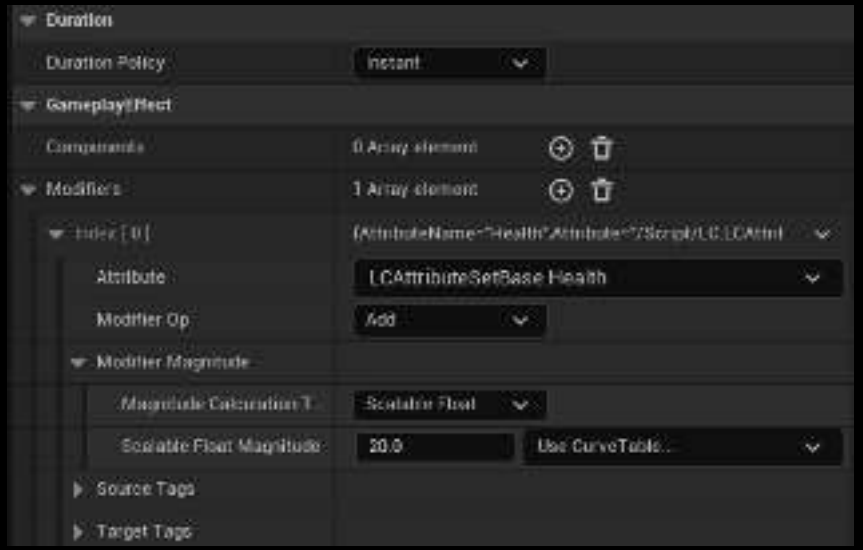
Duration Policy

- Instant
- Has Duration
- Infinite
- Period



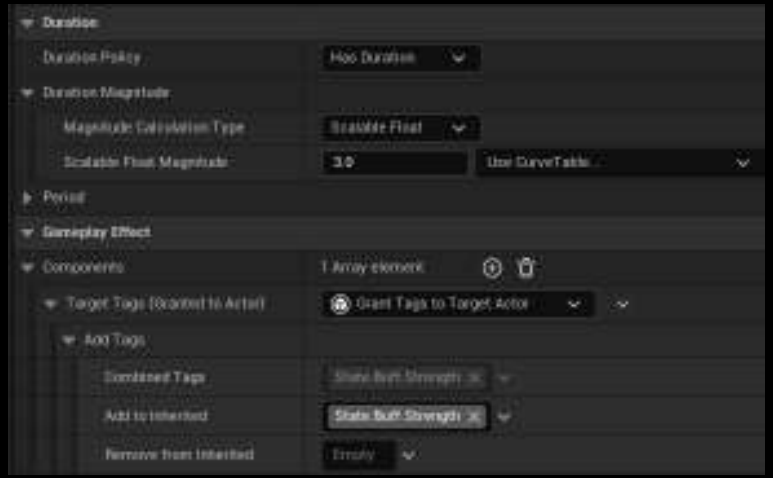
Instant

- 효과가 즉시 적용되고, 지속 시간이 없습니다.



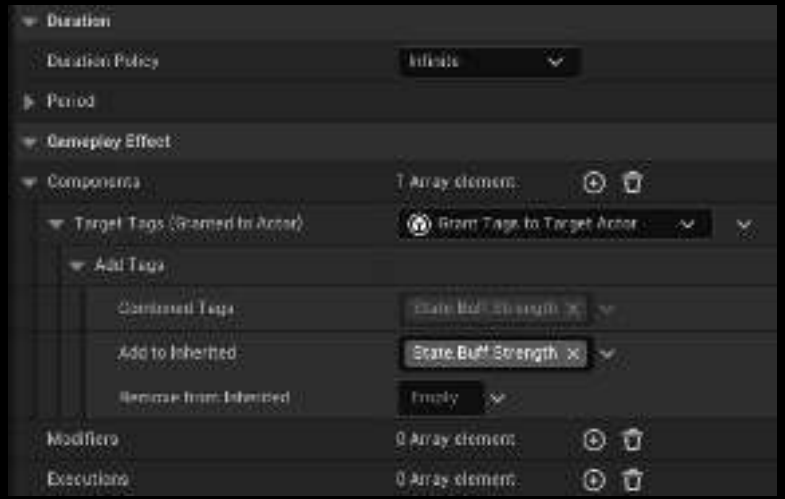
Has Duration

- 효과가 일정 시간 동안 지속됩니다. 이 경우, 지속 시간을 설정할 수 있으며, 이 시간이 지나면 효과가 자동으로 종료됩니다.



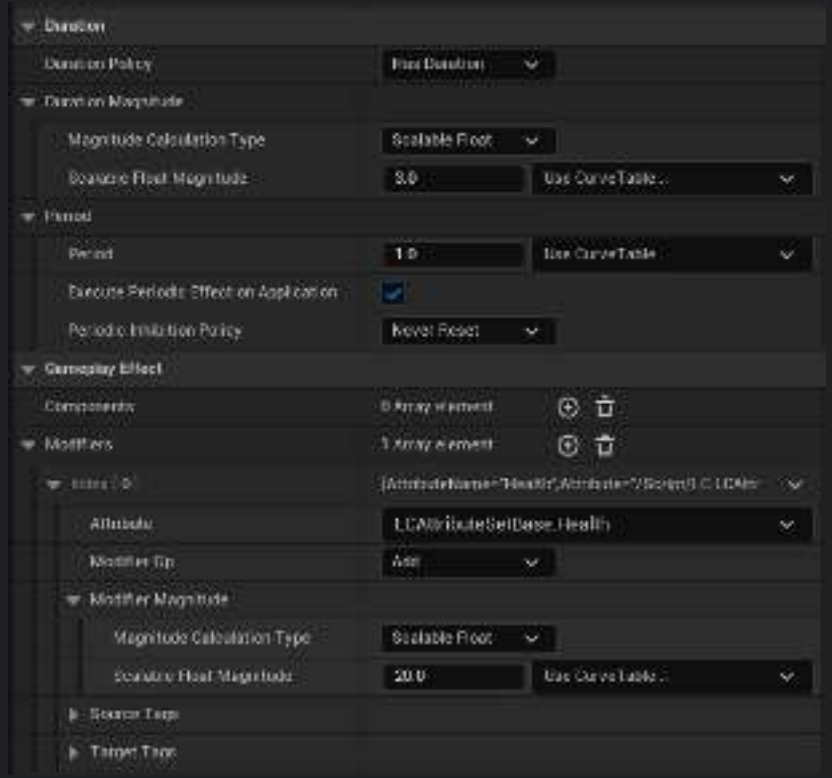
Infinite

- 효과가 무한히 지속됩니다. 한 번 적용되면, 명시적으로 제거되지 않는 한 영원히 유지됩니다.



Period

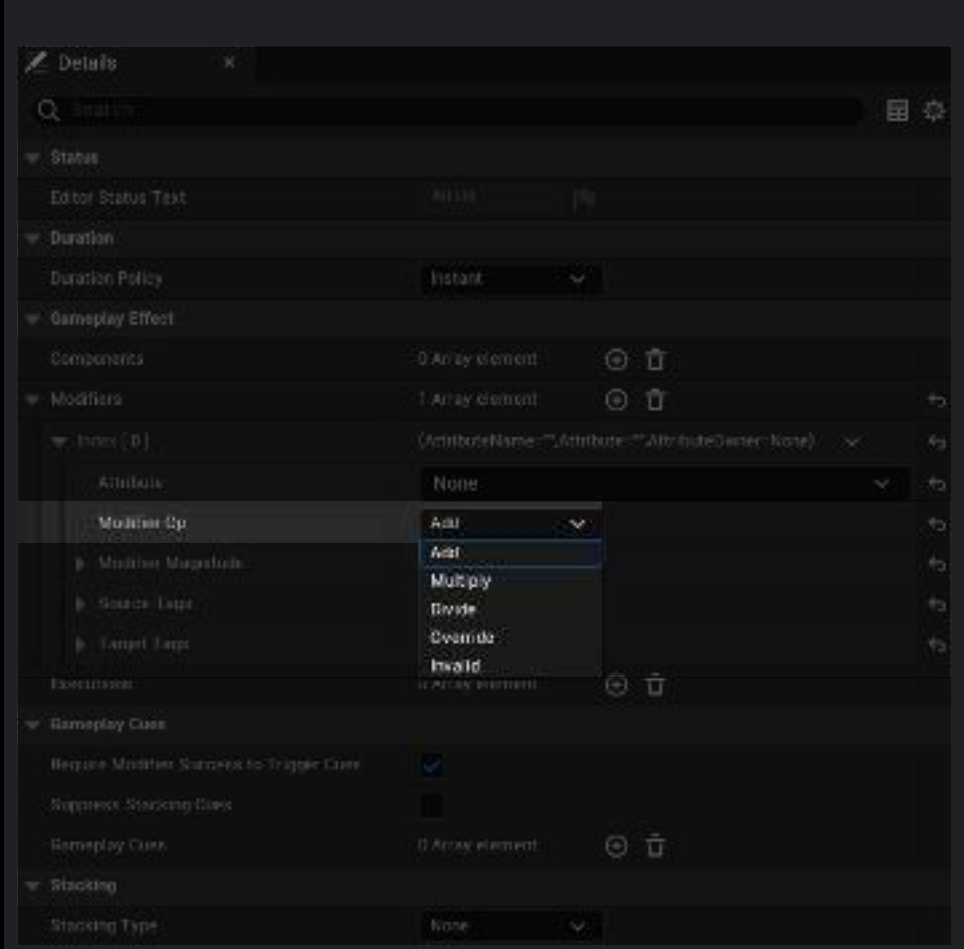
- “Has Duration”과 “Infinite” 정책에서 활성화 됩니다.
- 0.0f 초과인 경우에는 Periodic 이펙트가 되며 정해진 시간동안 주기적으로 이펙트를 실행합니다.
- 정해진 시간동안 주기적으로 대미지를 주는 메커니즘을 제작할 수 있습니다.



Periodic 적용 사례

Modifier Operation

- Add
- Multiply
- Divide
- Override

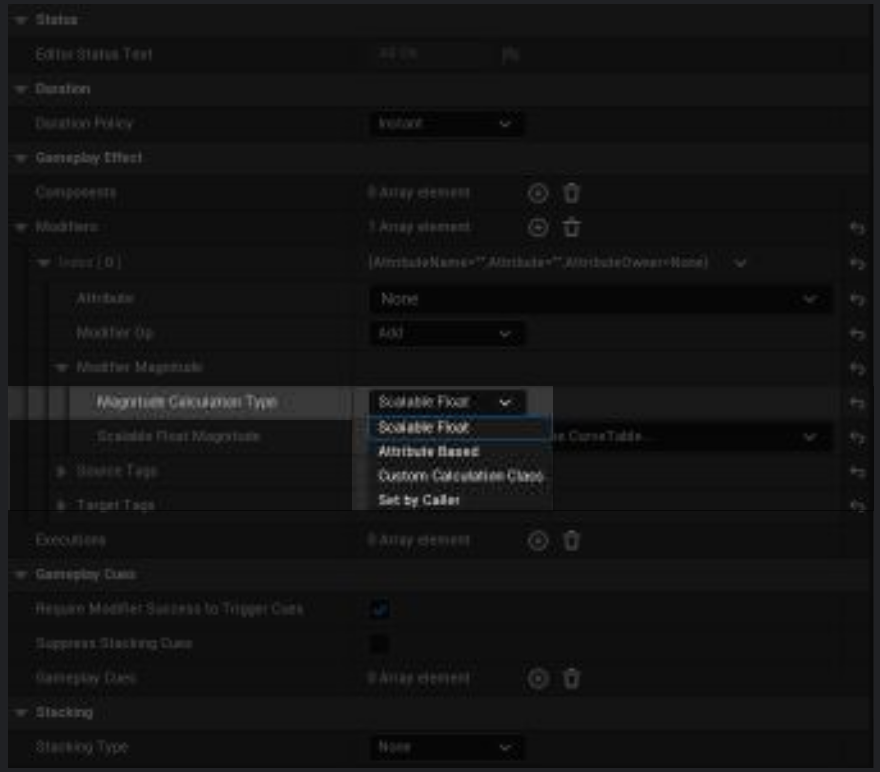


Modifier Operation

- Add
  Health 20 증가
- Divide
  Health 3 나누기
- Multiply
  Health 2배 증가
- Override
  Health 50으로 변경

Magnitude Calculation Type

- Scalable Float
- Attribute Based
- Custom Calculation Class
- Set By Caller



Scalable Float

- 가장 간단한 방법
- 단일 Float 값을 사용
- Curve 사용 시 Level에 따라 Curve Value 사용
  Float Magnitude * Curve[Level].Value
  예) Effect Level : 2
  20 * 1.5 = 30

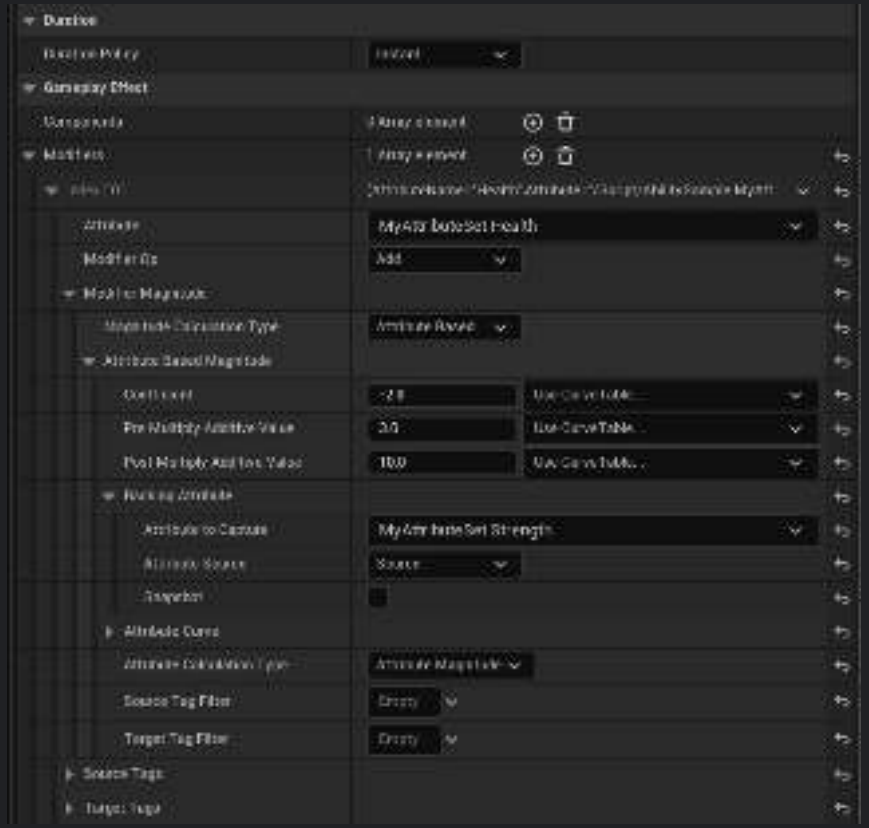
+ Curve	1	2	3
Level	1.0	1.5	2.0

정적인 값 사용

Curve Value 사용

Attribute Based

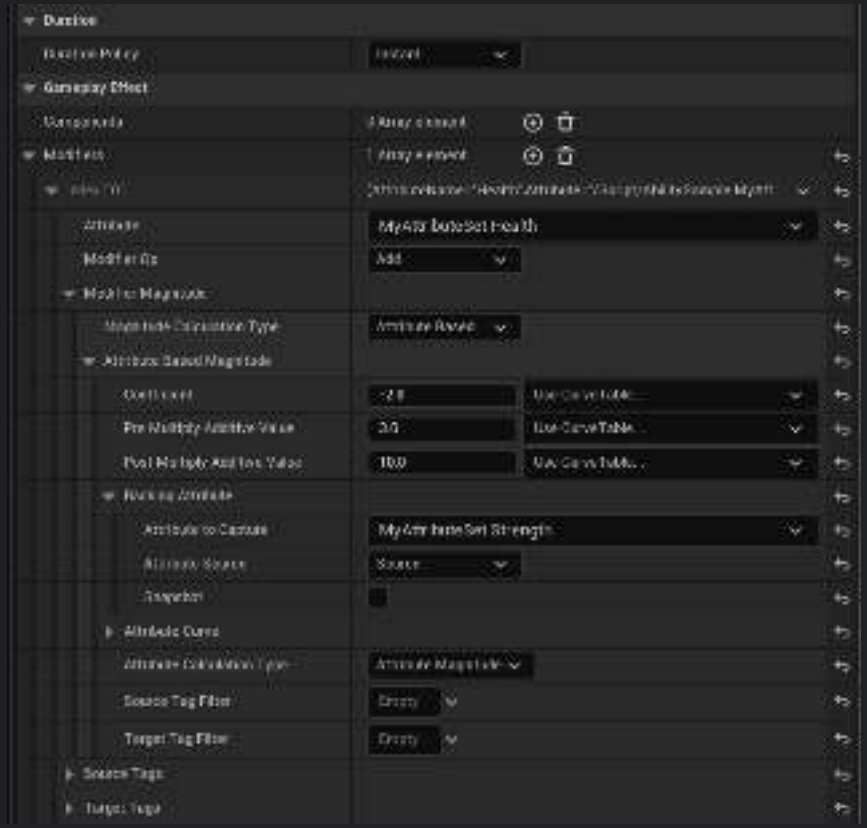
- 캐릭터의 Attribute 기반하여 동적으로 계산
- 간단한 대미지 공식 제작 가능
- Coefficient * (Attribute + Pre Value) + Post Value
- Backing Attribute
  공식에 사용될 Attribute
- Attribute Source
  Source : 발동한 자신의 Attribute 사용
  Target : 목표물의 Attribute 사용
- Snapshot
  Backing Attribute 값 캡처
- 예) Strength 10
  -2 * (10 + 3) + 10 = -16



Attribute Based

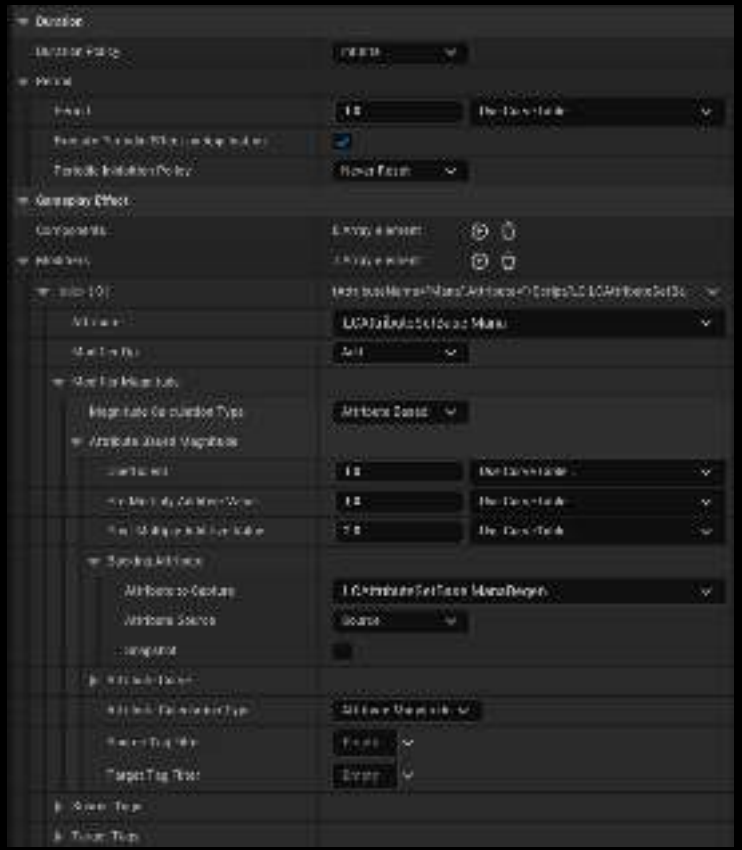
- Attribute Curve
  Lookup Table, 계산된 Attribute의 값을 매핑된 테이블의 값을 사용하도록 할 수 있음

- Attribute Calculation Type
  Backing Attribute의 Base Value나 Bonus Value를 선택적으로 사용 가능



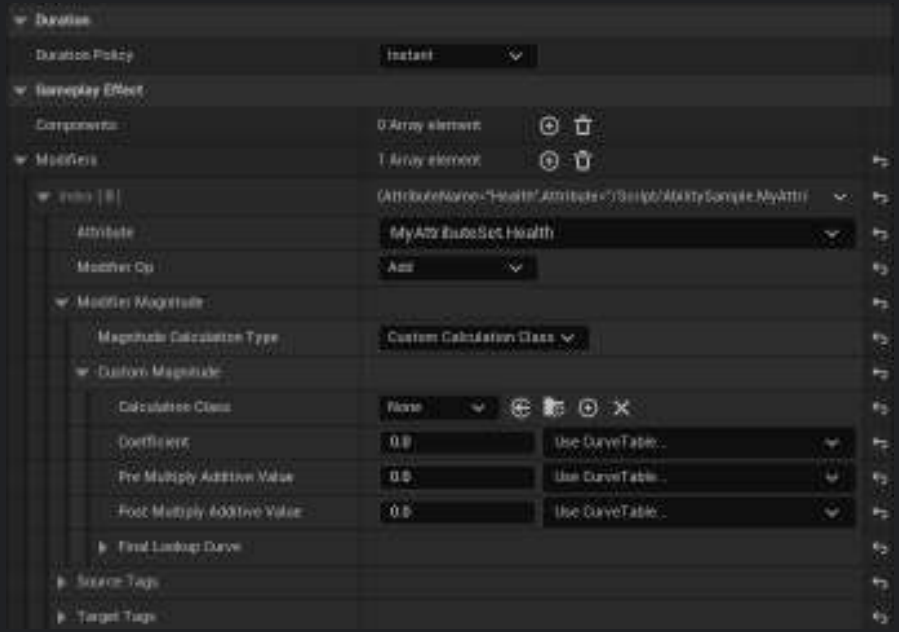
Attribute Based 적용 사례

- 1 * (ManaRegen + 1) + 2



Mod Magnitude Calculation

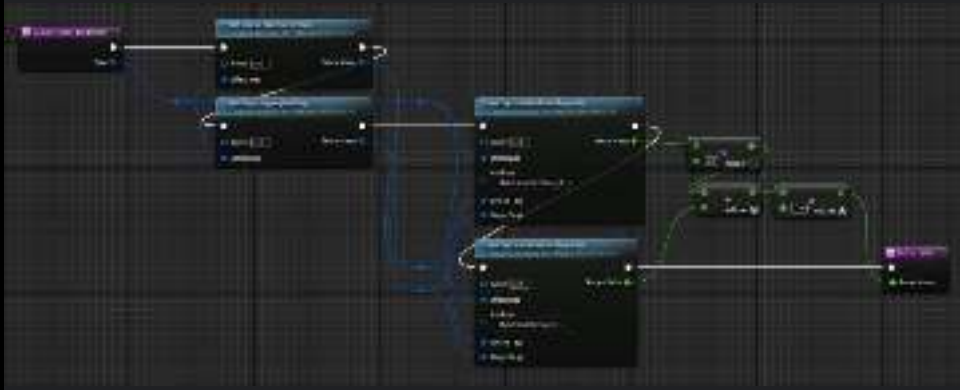
- Modifiers에서 사용되는 강력한 클래스
- Gameplay Effect Execution Calculation과 유사하지만 덜 강력함
- 복잡한 대미지 공식을 만들 수 있음
- Lookup Table, 계산된 Attribute의 값을 매핑된 테이블의 값을 사용하도록 할 수 있음

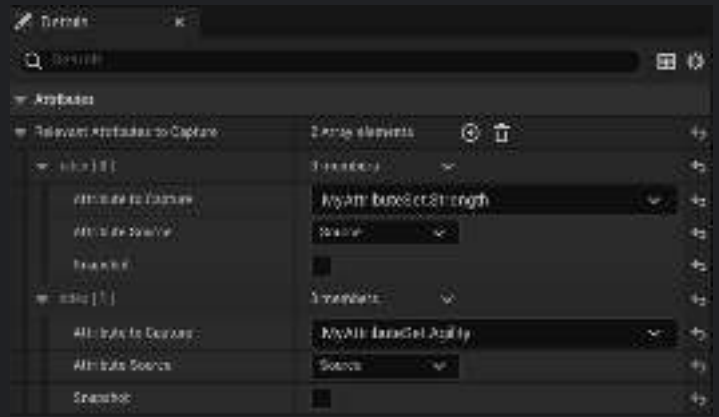


Mod Magnitude Calculation

- 예) Strength 5, Agility 10

  대미지 공식
  (Strength * 2) + Agility
  ((5 * 2) + 10) * -1 = -20





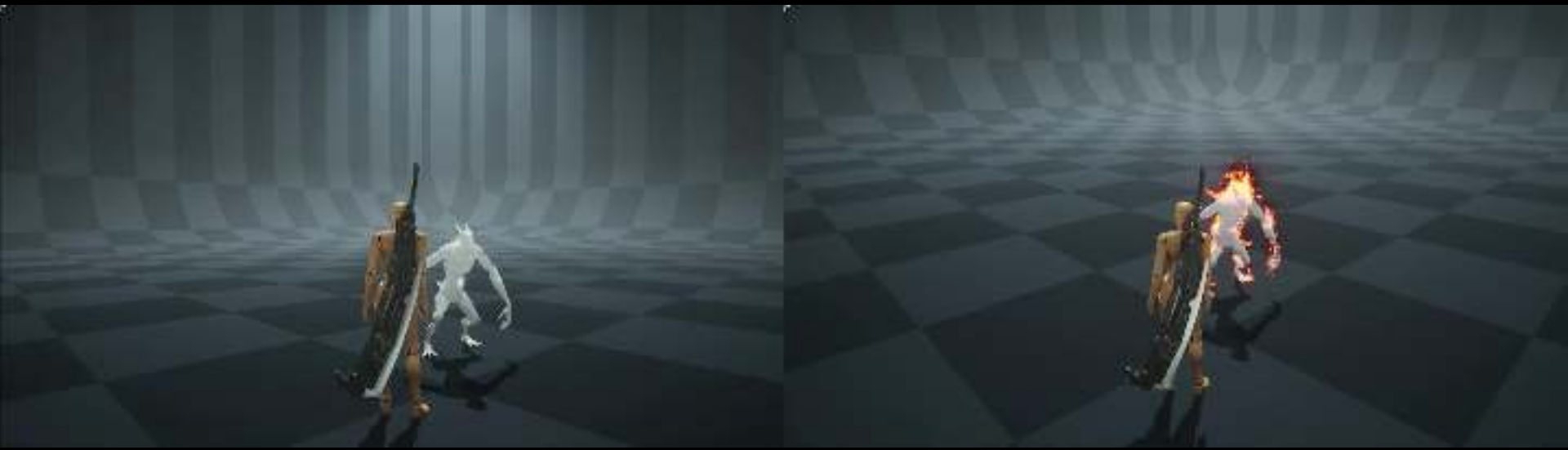
Set By Caller

- Gameplay Effect Spec에 Data Tag를 활용하여 값을 바인딩 할 수 있습니다.
- 런타임에서 값을 설정할 수 있습니다.
- Gameplay Effect에서 미리 정해진 값이 아닌 게임 안에서 가공된 값을 설정할 수 있습니다.
- Set By Caller라는 FName 버전이 있지만 Tag버전 사용을 권장합니다.

Gameplay Effect Execution Calculation

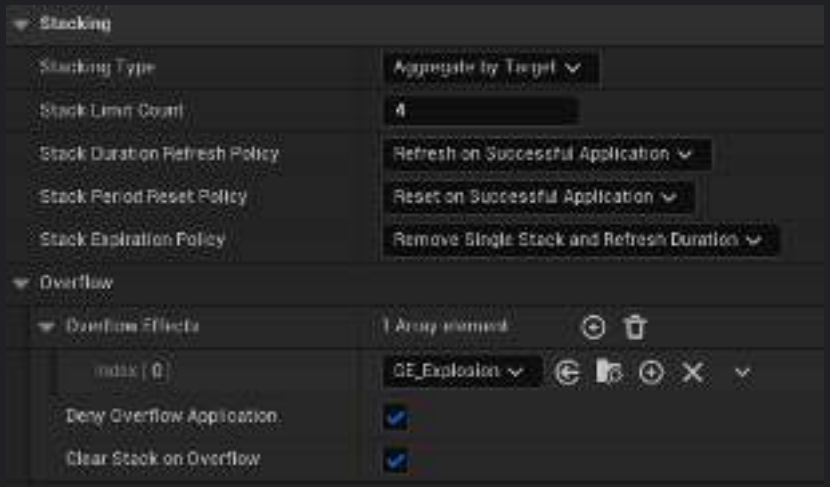
- 대미지 계산에 사용되는 가장 강력한 클래스
- Mod Mag Calc와 유사하며 여러 Attribute를 변경할 수 있습니다.
- Instant, Periodic 유형에서 적용
- C++에서만 구현 가능
- 복잡한 대미지 공식을 만들 수 있음

```cpp
void UMyDamageCalc::Execute_Implementation(
    const FGameplayEffectCustomExecutionParameters& ExecutionParams,
    FGameplayEffectCustomExecutionOutput& OutExecutionOutput) const
{
    const FGameplayEffectSpec& Spec = ExecutionParams.GetOwningSpec();
    FGameplayTagContainer AssetTags;
    Spec.GetAllAssetTags(AssetTags);

    const FGameplayTagContainer* SourceTags =
        Spec.CapturedSourceTags.GetAggregatedTags();

    const FGameplayTagContainer* TargetTags =
        Spec.CapturedTargetTags.GetAggregatedTags();

    FAggregatorEvaluateParameters EvaluationParameters;
    EvaluationParameters.SourceTags = SourceTags;
    EvaluationParameters.TargetTags = TargetTags;

    // Get the Source and Target attributes
    float LocalStrength = 0.f;
    ExecutionParams.AttemptCalculateCapturedAttributeMagnitude(
        StrengthDef, EvaluationParameters, LocalStrength);

    if (TargetTags->HasTag(
        FGameplayTag::RequestGameplayTag(FName("State.Debuff"))))
    {
        LocalStrength *= 2.0f;
    }

    // Set the new Health
    OutExecutionOutput.AddOutputModifier(FGameplayModifierEvaluatedData(
        UMyAttributeSet::GetHealthAttribute(), EGameplayModOp::Additive,
        -LocalStrength));
}
```



Stack

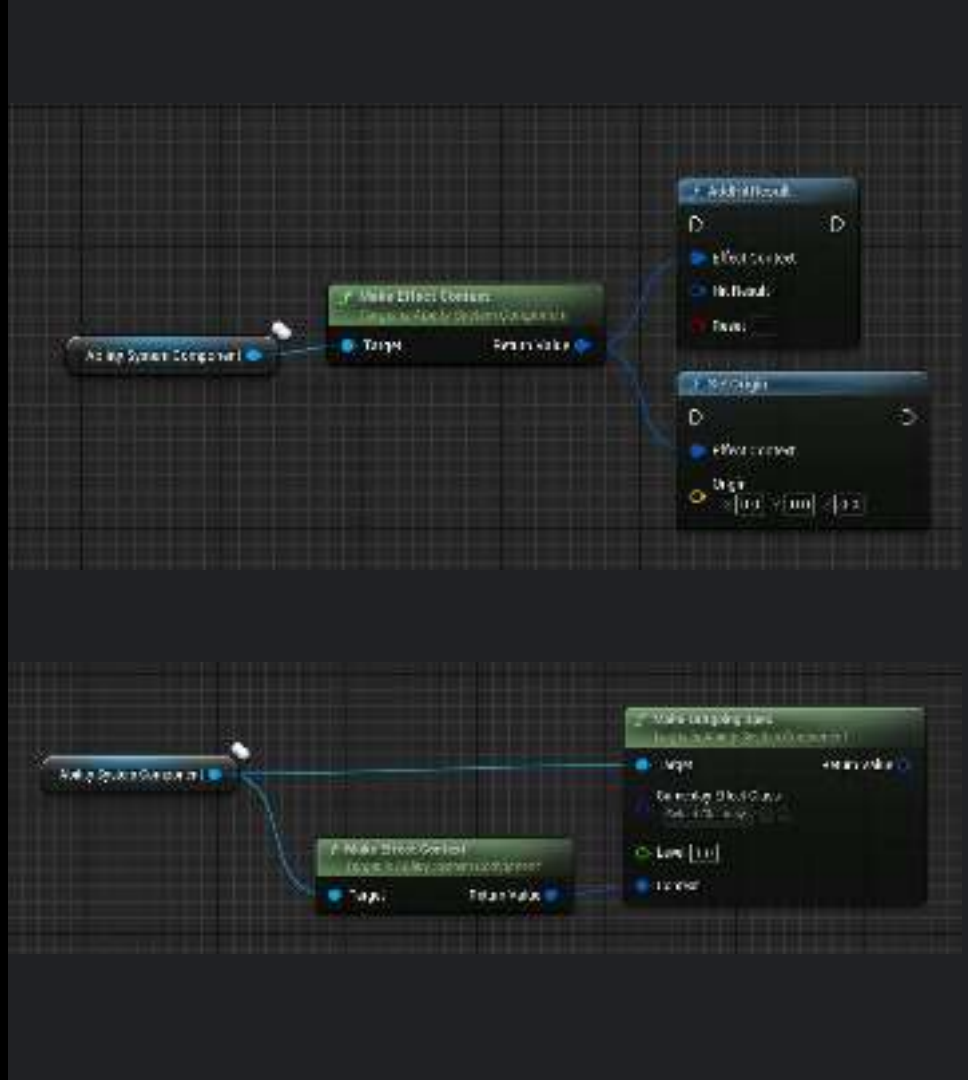
- 같은 게임 게임플레이 이펙트가 여러 번 적용 될 때 단일 이펙트로 관리 되지 않고 누적되는 형태의 이펙트로 관리 할 수 있도록 합니다.
- 스택은 보편적으로 “Has Duration”, “Infinite” 기간 정책을 사용합니다.
- Overflow Effects를 사용하여 스택이 가득 찬 경우에 추가 효과를 줄 수 있습니다.




Tab [Apostrophe] to close or hold to select new Pawn (hold [Shift] Apostrophe to select local player) Use Numpad to Upper Categories
Ctrl+Tilde HUD Ctrl+Tab DebugMessages Tab Spectator
0 Navmesh 1 AI 2 BehaviorTree 3 EQS 4 Perception 5 PerceptionSystem 6 UIDebugger 9 Abilities
Debug actor: BP_UCPlayerCharacterBase_C_0 [ROLE_Authority]
VLog: not recording to file
[CATEGORY: Abilities] Tags [Shift+One] Abilities [Shift+Two] Effects [Shift+Three] Attributes [Shift+Four]
Gameplay Effects [0] Legend: Both [0] Server [0] Local [0]

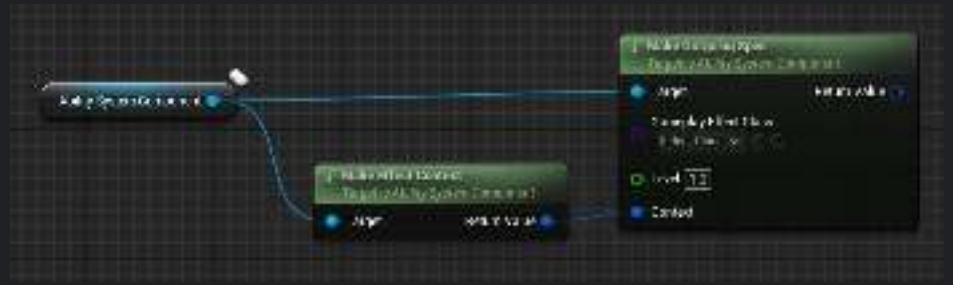
Gameplay Effect Context

- Instigator과 관련된 데이터를 전달하기 위해 구조체
- 게임에 맞게 서브 클래싱하여 다양한 정보를 추가 가능
- Gameplay Effect Spec 에 필수적인 데이터
- Gameplay Effect 상호작용에 필요한 일시적인 데이터를 저장
- Mod Mag Calc, Execution Calc, Attribute Set, Gameplay Cue 등에서 Effect Context의 데이터를 가져와서 필요한 연산 가능
- 예) Hit Result, 위치 정보, 타깃의 위치 정보, Origin 위치 정보 등



Gameplay Effect Spec

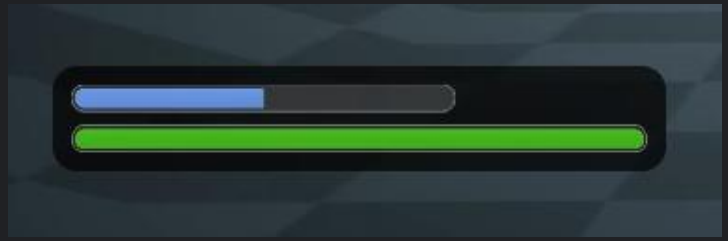
- Gameplay Effect의 CDO를 담고 있는 상호작용 사양
- Gameplay Effect의 Level이나 지속 시간은 기본적으로 같지만 값을 변경하면 달라질 수 있습니다.
- Gameplay Effect Context Handle을 가지고 있으며 Instigator나 Target의 정보를 포함합니다.
- Make Outgoing Spec 함수로 생성됩니다.
- Spec이 만들어질 때 Level, Effect Context 등의 데이터가 초기화 됩니다. Modifiers, Executions에 사용되는 데이터의 Source Attribute가 캡처 됩니다.
- Target Attribute는 Apply Gameplay Effect에서 적용 시 캡처 됩니다.



Post Gameplay Effect Execute 활용 사례

대미지를 받은 경우 Mana를 먼저 감소한 후 Health를 감소시키는 코드

```
if (Data.EvaluatedData.Attribute == UMyAttributeSet::GetDamageAttribute())
{
    const float Damage = Data.EvaluatedData.Magnitude;
    const float CurMana = UMyAttributeSet::GetManaAttribute().GetGameplayAttributeDataChecked(this)->GetCurrentValue();
    const float CurHealth = UMyAttributeSet::GetHealthAttribute().GetGameplayAttributeDataChecked(this)->GetCurrentValue();
    const float ManaDamage = FMath::Clamp(Damage, 0.0f, CurMana);
    const float HealthDamage = FMath::Max(Damage - ManaDamage, 0.0f);
    if (ManaDamage > 0.0f)
    {
        SetMana(CurMana - ManaDamage);
    }
    if (HealthDamage > 0.0f)
    {
        SetHealth(CurHealth - HealthDamage);
    }

    SetDamage(0.0f);
}
```



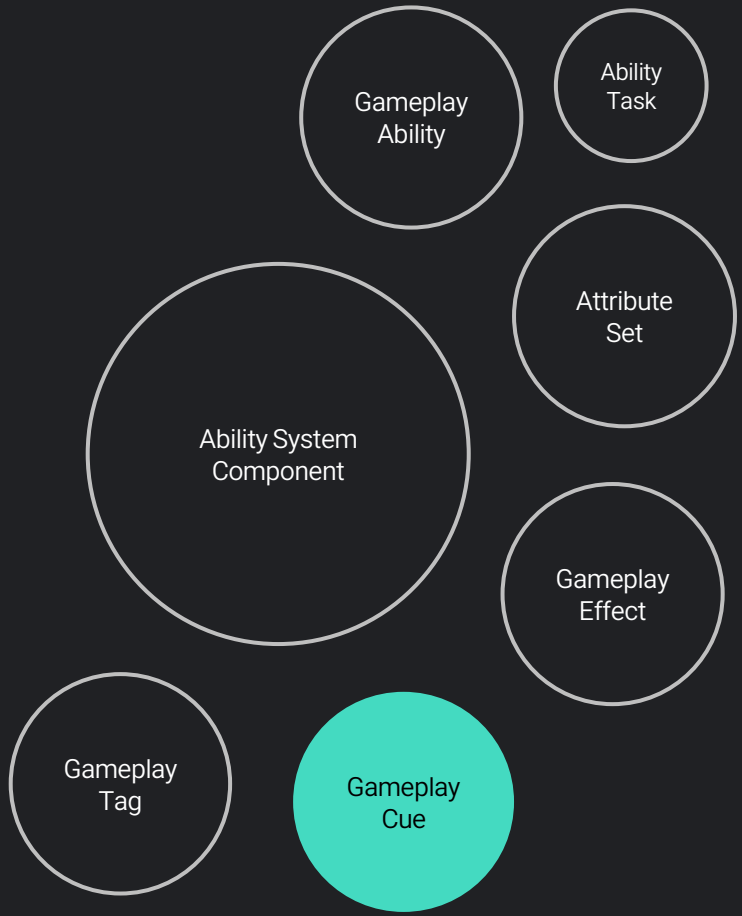
이외 Gameplay Effect 로 할 수 있는 것들

- Immunity (면역)
- Chance To Apply This Effect (확률에 의한 이펙트 적용)

Gameplay Cue

Gameplay Cue

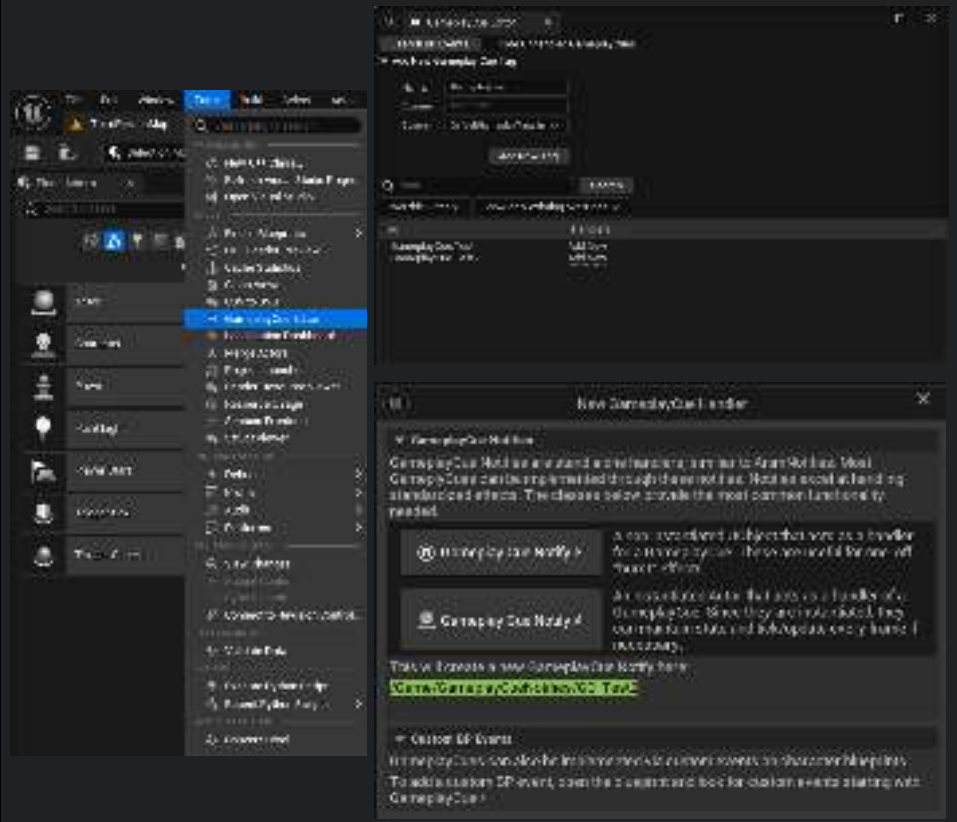
- 사운드 효과, 파티클, 카메라 연출을 묶어서 표현
- 태그를 사용하여 연출 효과를 트리거
- 다양한 효과를 쉽게 관리



Gameplay Cue 종류

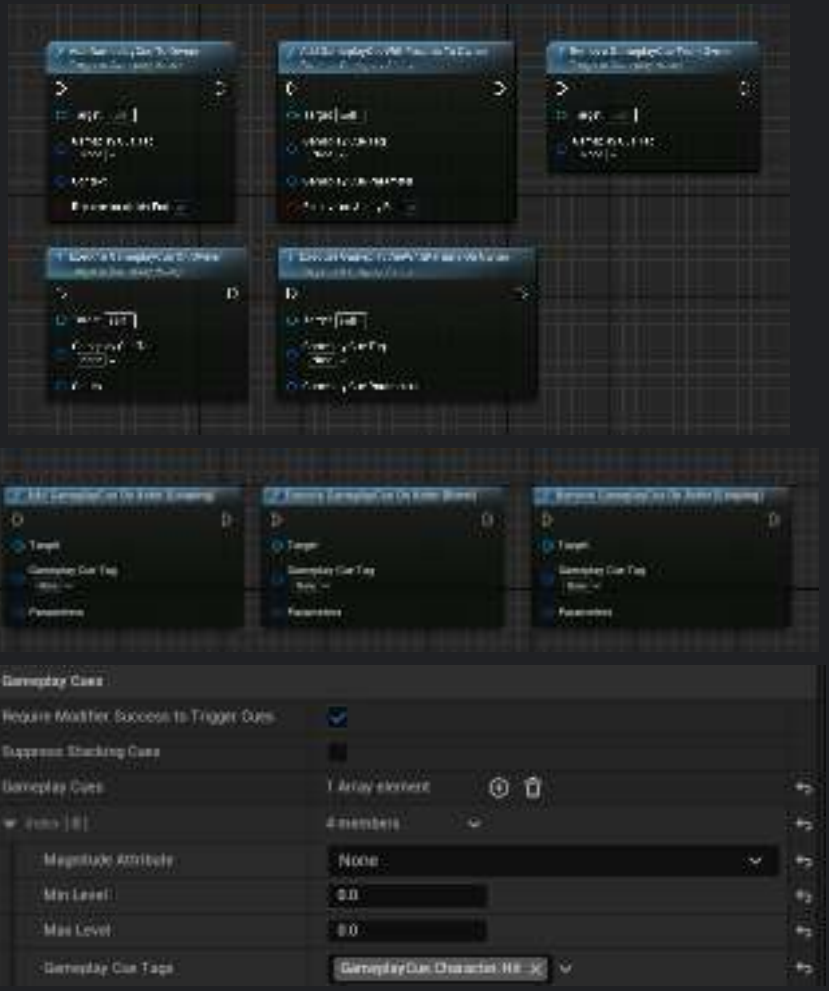
GameplayCueNotify_Static
개별 인스턴스가 없는 클래스 디폴트 오브젝트에서 작동
타격 이펙트와 같은 일회성 이펙트에 적합

GameplayCueNotify_Actor
인스턴스화되어 제거 될 때까지 계속 동작
라이프 사이클이 있는 이펙트에 적합
기본적으로 Actor는 재활용될 수 있습니다.



Gameplay Cue 실행 방법

- Gameplay Ability에서 함수를 사용하여 호출
- Gameplay Cue Function Library의 함수 호출
- Gameplay Effect에서 Gameplay Cue Tag를 추가



Gameplay Cue Event

On Active

Gameplay Cue가 처음 활성화될 때 한 번 호출됩니다. 초기 효과나 설정을 처리합니다.

예시) 기름에 불을 붙이면 폭발이 일어납니다.

WhileActive

Gameplay Cue가 활성 상태일 때 한 번 호출됩니다. 이미 활성 상태인 Gameplay Cue에서 실행이 되어야 하는 효과를 처리합니다.
이것은 Tick 연산이 아닙니다.

예시) 기름에 불을 붙인 후 지속적으로 활활 타오르는 이펙트

Executed

Gameplay Cue가 실행될 때 호출되며, 즉시 실행되는 효과나 이벤트를 처리합니다.

예시) Hit 이펙트의 Slash 이펙트

Removed

Gameplay Cue가 제거될 때 한 번 호출됩니다. 종료 효과와 정리 작업을 처리합니다.

예시) 활활 타오르는 불 이펙트를 Deactivate

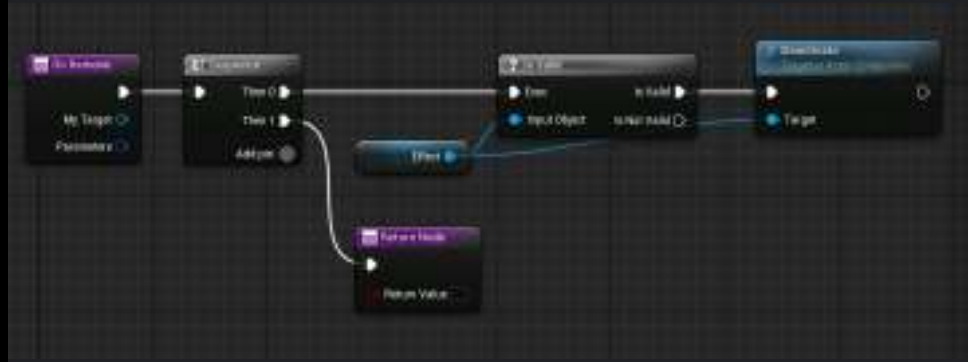
Gameplay Cue Static 예제

Gameplay Cue Static
적용 사례

Gameplay Cue Actor 예제

On Active

On Remove



Gameplay Cue Actor
적용 사례

그 외
알면 좋은 것들

Gameplay Cue Manager

- 게임 플레이 도중 발생하는 다양한 Gameplay Cue 이벤트를 효과적으로 관리하고 재구현 할 수 있는 매니저

- Gameplay Cue Manager를 상속하여 Scan 및 Load 전략을 변경할 수 있습니다.
  런타임에서 기본적으로 모든 Gameplay Cue를 비동기 로드 합니다.

```cpp
bool UGameplayCueManager::ShouldSyncScanRuntimeObjectLibraries() const
{
    // Always sync scan the runtime object library
    return true;
}
bool UGameplayCueManager::ShouldSyncLoadRuntimeObjectLibraries() const
{
    // No real need to sync load it anymore
    return false;
}
bool UGameplayCueManager::ShouldAsyncLoadRuntimeObjectLibraries() const
{
    // Async load the run time library at startup
    return true;
}
```

Ability System Globals

- Gameplay Effect Context 를 재구현 할 수 있습니다. Gameplay Effect 전달에 파라메터를 다양화 할 수 있습니다.
- 여러 공용 Tag를 설정할 수 있습니다.
- Ability System에서 공통적으로 사용되는 데이터
- Gameplay Cue Path를 지정합니다. Path가 지정되어 있지 않는 경우 /Game/ 를 사용합니다. 대규모 프로젝트에서 지정되어 있지 않는 경우 느릴 수 있습니다.

```
virtual FGameplayEffectContext* AllocGameplayEffectContext() const;


/** Global Tag */
UPROPERTY(config)
FName ActivateFailIsDeadName;

UPROPERTY(config)
FName ActivateFailCooldownName;

UPROPERTY(config)
FName ActivateFailCostName;

UPROPERTY(config)
FName ActivateFailTagsBlockedName;

UPROPERTY(config)
FName ActivateFailTagsMissingName;

UPROPERTY(config)
FName ActivateFailNetworkingName;

/** Look in these paths for GameplayCueNotifies.*/
UPROPERTY(config)
TArray<FString> GameplayCueNotifyPaths;
```

Ability Async

- Blueprint Latent 함수를 쉽게 제작 할 수 있는 클래스
- 액터 클래스에서 Ability Task와 유사하게 사용 가능
- 재사용이 높은 콜백 함수를 블루프린트 노드로 만들어 생산성을 향상시킬 수 있습니다.
- 예시 클래스)
  AbilityAsync _ WaitAttributeChanged
  AbilityAsync _ WaitGameplayTagAdded
  AbilityAsync _ WaitGameplayTagRemoved

Ability Task 종료 후의 콜백 경고

- Ability 가 종료 되었음에도 불구하고 콜백을 받은 후 이벤트를 전달하려는 Ability Task를 감지 하기 좋습니다.

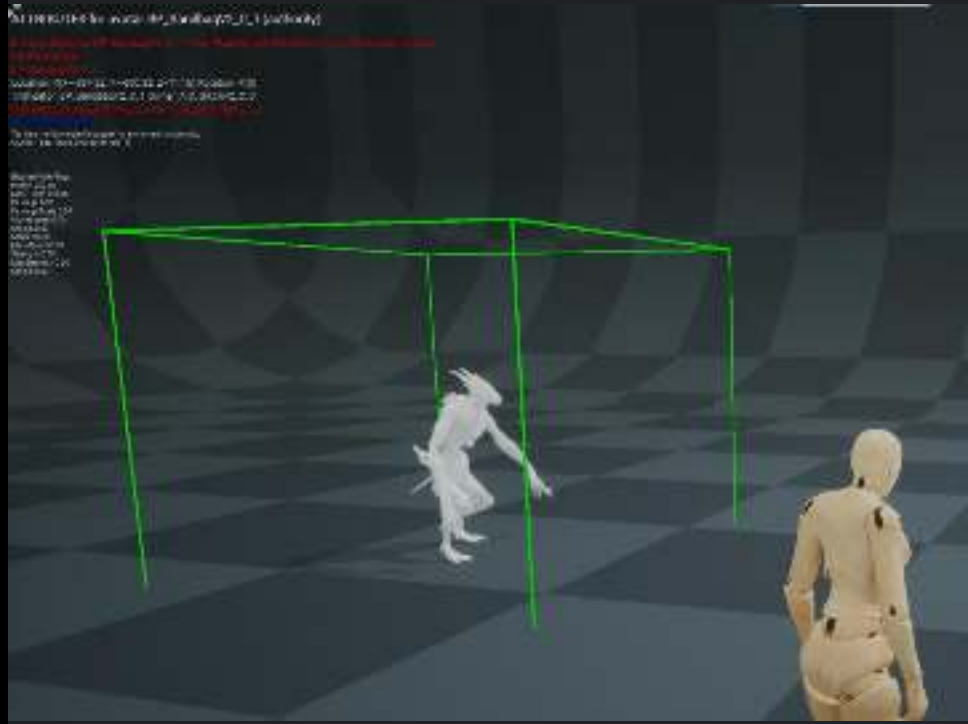
DefaultEngine.ini

[SystemSettingsEditor]
AbilitySystem.AbilityTaskWarnIfBroadcastSuppress=1

[SystemSettings]
AbilitySystem.AbilityTaskWarnIfBroadcastSuppress=1

Debugger

- ShowDebug AbilitySystem
  AbilitySystem.Debug.NextCategory
  AbilitySystem.Debug.PrevCategory

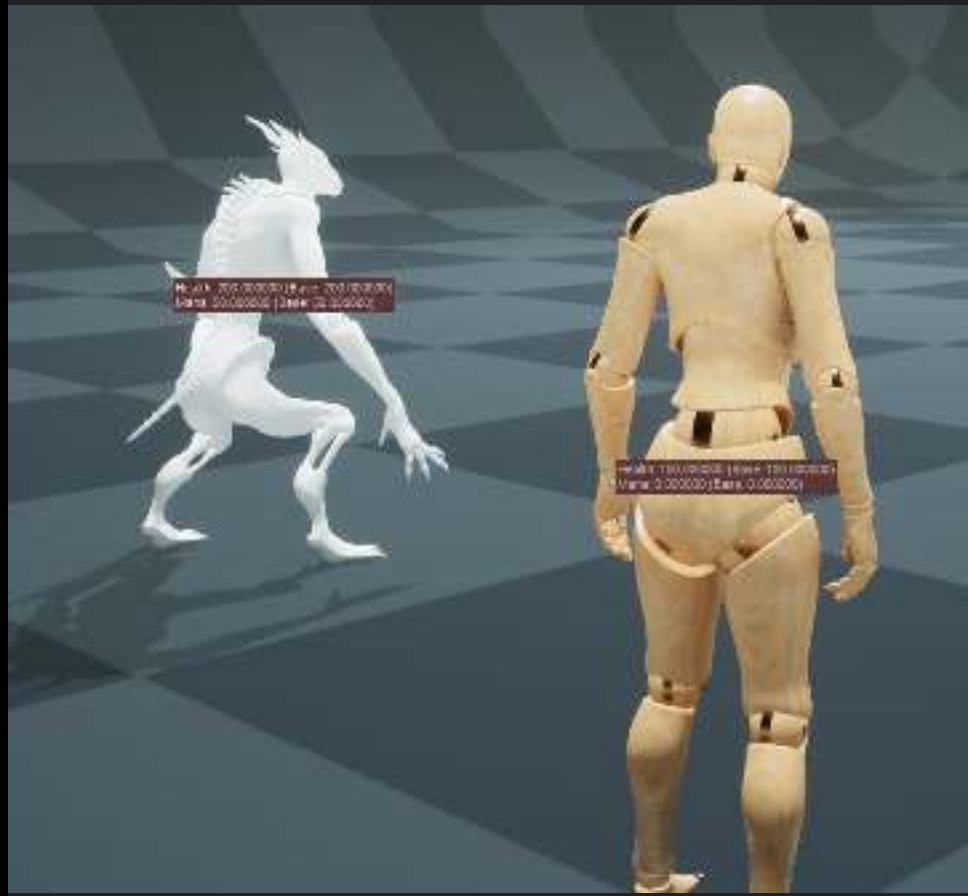


Debugger

- AbilitySystem.DebugAbilityTags

Debugger

- AbilitySystem.DebugAttribute
  [Attribute1] [Attribute2]
- 예시)
  AbilitySystem.DebugAttribute Health Mana



Debugger

- ' (어퍼스트로피)

참고 자료

언리얼 엔진 공식 문서
https://dev.epicgames.com/documentation/ko-kr/unreal-engine/gameplay-ability-system-for-unreal-engine

Lyra Game
https://www.unrealengine.com/marketplace/ko/learn/lyra?lang=ko

Tranek/GASDocumentation
https://github.com/tranek/GASDocumentation

액션 RPG로 알아보는 게임플레이 어빌리티 시스템 Fest 2022
https://youtu.be/vknzNVYJjr4?si=OxRvuvYysqFshNcY

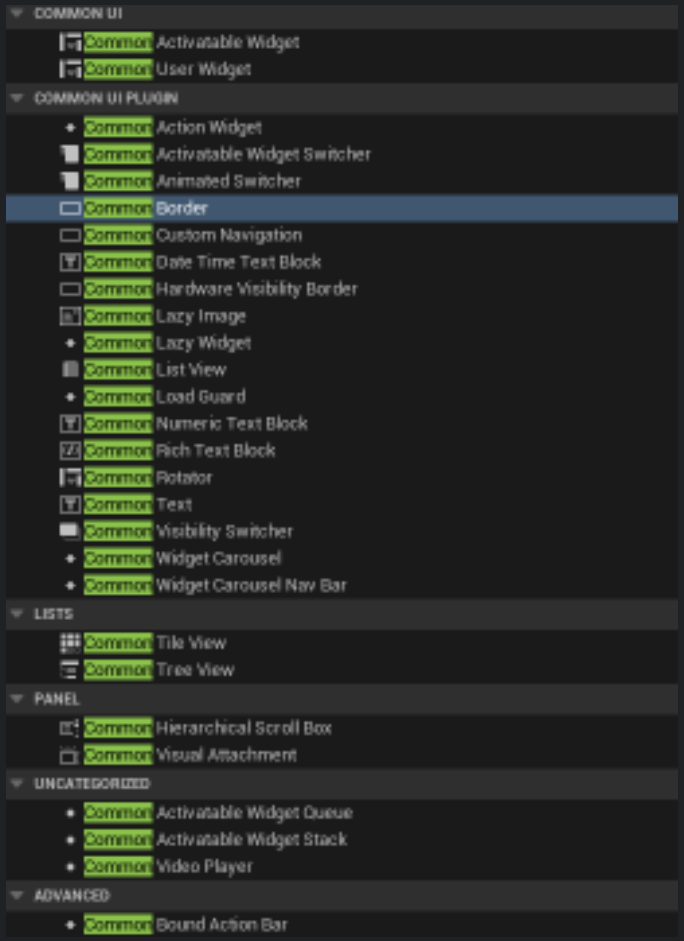
Common UI

Common UI 플러그인 활성화 방법

- Edit → Plugins
- UI → Common UI Plugin

Common UI

- 보편적으로 사용할 수 있는 UMG 패널 추가
- 스타일링 일관성을 쉽게 유지

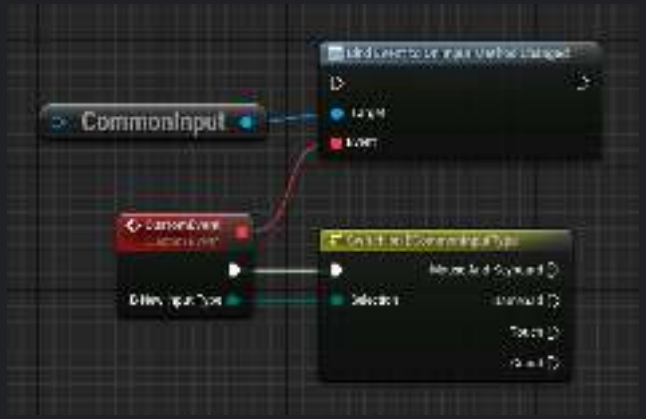


Common UI

- 멀티 레이어 메뉴 탐색
- Common Activatable Widget 을 활용하여 적합한 네비게이션 포커싱
- Stack을 활용한 뒤로 가기 동작 구현 용이

Common UI 장점

- Common Input Module 을 활용한 게임 인풋 타입에 유연한 대응
- Controller Data 를 사용하여 게임패드 및 키보드, 마우스 UI 리소스 관리 용이

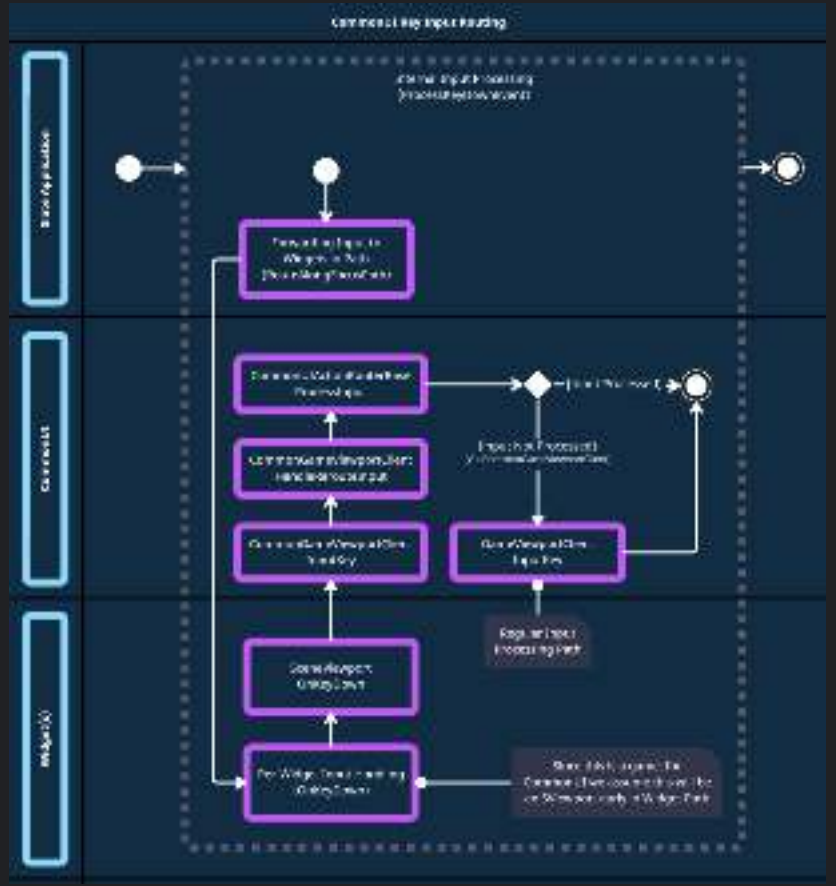


Common UI 사용 사례

- Carousel 기능 구현 가능

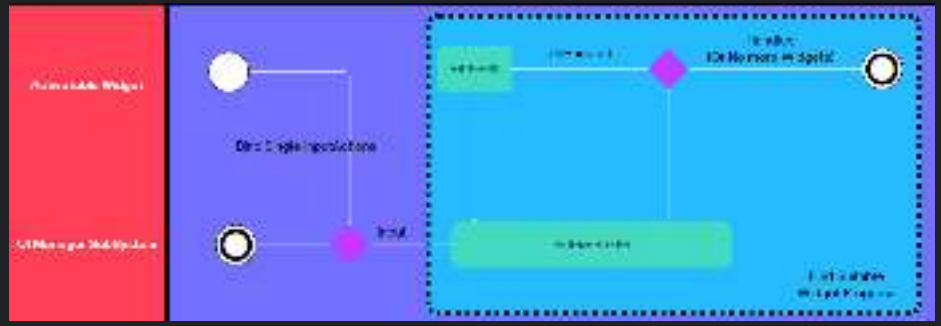
Common UI 입력 처리 방식

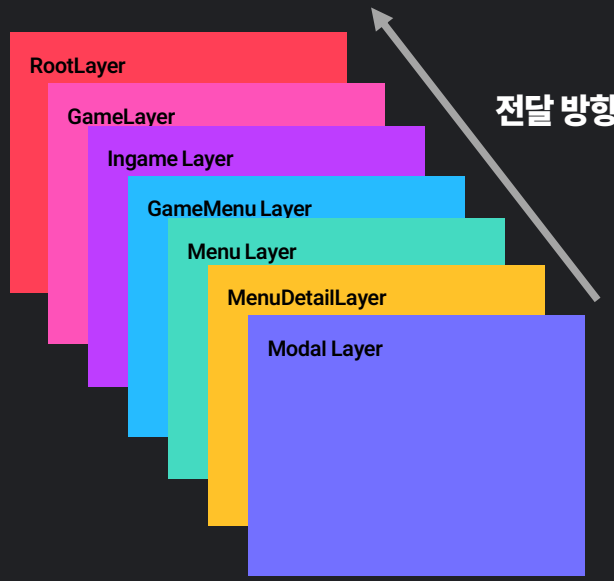
- 게임 패드 입력을 마우스 입력으로 변환하여 UI와 상호작용합니다. 즉, 보이지 않는 마우스 커서가 클릭하는 것뿐입니다.
- 모든 입력 경로를 하나의 입력 경로로 통합하여 크로스 플랫폼 입력의 흐름을 단순화합니다.
- CommonUI Input Routing
  입력 장치로부터 입력을 통합하여 적절한 UI 위젯으로 라우팅합니다. UI 위젯이 없는 경우에는 기본적인 게임 입력 흐름을 따릅니다.
- Action Router 는 “Activatable Widget” 루트 노드로 입력 이벤트를 전달합니다. 각 노드는 입력을 처리할 수 있으면 처리하고 그렇지 않다면 자식 노드에게 전달합니다.



Common UI 응용

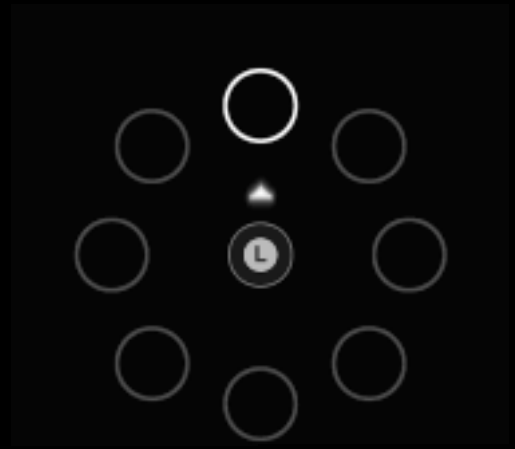
- Common Activatable Widget과 Enhanced Input 결합하여 사용
- Enhanced Input의 콜백을 Widget이 받아 UI 움직임 구현
- 포커스에서 구현하기 까다로운 UI 구현이 용이
- Game And UI 모드를 활용하여
  마우스, 키보드 ↔ 게임패드 간의 자연스러운 변경





Common UI 응용 사례

- Input Action을 게임패드 특정 키를 매핑 후, 해당 Input Value를 받아 동작 구현
- Input Action의 Modifier를 활용하여 리스트의 항목을 이동할 때 점진적으로 속도가 빨라지는 기능 구현
- Input Action의 특정 키를 Hold 하여 게이지를 올리는 애니메이션 제작
- 엔진 기본 포커스의 기본 동작을 수정하지 않고 제작



프로토타입 제작
팁과 사례 공유

여러 팁과 사례를 매우 빠르게 공유

발표에 앞서

㈜넥스트스테이지의 대표 강현우

- **앞의 내용이 더 중요하기 때문에 이 섹션은 10분**
- **10분 안에 핵심 내용만 빠르게 전달**
- PPT 내에 핵심 키워드 작성

자료 제작에 도움을 준 UDN 컨설턴트 맴버

Junsu Park
CTO
Next Stage inc.

Homin Lee
Gameplay, AI, Animation
Next Stage inc.

Jeahoon Jang
World Build
Next Stage inc.

Yongmin Choi
System
Next Stage inc.

Minyeol Park
GamePlay
Next Stage inc.

Seunghwan Lee
Rendering
Next Stage inc.

질문과 답변을 기반으로

넥스트스테이지는

UDN 컨설턴트 파트너 로 진행한 많은 답변

사내 **신규 프로토타입 제작**하며 얻은 사례

다양한 업체들의 **컨설팅 진행** 경험과 기반 지식

을 바탕으로 공개 가능하며 다수가 궁금해 하는 부분들을
지식과 사례로 공유해서 언리얼 엔진을 사용하는 모든 분들이
함께 발전할 수 있도록 짧지만 도움이 되도록 준비 했습니다.

기본 팁
사례 공유

언리얼 엔진 기본 팁 알아보기

아는 분들이 적지만 생산성이 높은 팁 위주로

LLVM을 사용한 정적 분석

정적 분석을 통해 코드의 잠재적인 버그, 보안 취약점, 코드 품질 문제를 사전에 발견하고 수정

사용할 수 있는 정적 분석 툴

- PVS-Studio (유료)
- **LLVM Clang** (무료)
- VisualCPP (무료)

사용 방법

- LLVM 16.0.0(UE5.4.2기준) 설치
- https://github.com/llvm/llvm-project/releases
- 언리얼 빌드 툴로 빌드를 진행할 때 StaticAnalyzer 옵션 추가
- -StaticAnalyzer=Clang
- 로그 파일의 내용을 확인하고 수정

```
"%UBT_PATH%" “ProjectEditor" Win64 Development
-Project= "%PROJECT_PATH%"
-WaitMutex -FromMsBuild
-StaticAnalyzer=Clang
-Log="%LOG_FILE%"
```

```
[441/682] Analyze [x64] LCSoundManager.cpp
C:\JenkinsBuildPC01AgentA\workspace\LC542\Svn\Source\LC\Sound\LC
SoundManager.cpp(269,6): note: Assuming the condition is false
  269 |         if (BeforeTop == TopSoundField())
      |             ^~~~~~~~~~~~~~~~~~~~~~~~~~~~
1 warning generated.
```

장점과 한계

- 코드 내에 잠재적 위험 확인
- CI/CD에 연결하면 좋음
- 만능은 아니고 잘못 잡는 경우도 있음

ParallelFor 를 사용하자

코드의 Loop 내부의 높은 비용이 발생하는 연산을 ParallelFor 를 사용하여 최적화

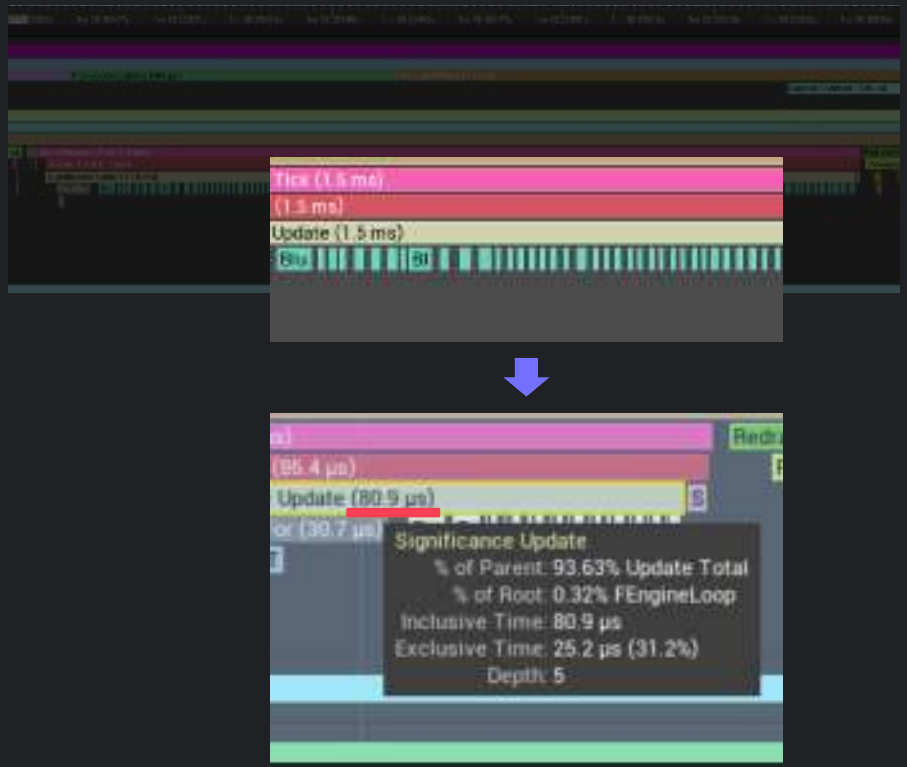
ParallelFor를 사용하면

- Loop 내 작업을 병렬로 처리
- 기존 For 보다 매우 빠르게 Loop 내 작업 처리
- 게임 로직에서 높은 비용을 줄일 수 있음

```
inline void ParallelFor
(int32 Num, TFunctionRef<void(int32)> Body,
bool bForceSingleThread,
bool bPumpRenderingThread=false)
```

ParallelFor가 필요한 곳

- 게임 쓰레드에서 비싼 비용의 반복 작업이 이뤄지는 곳
- 코드를 분석했는데 싱글 쓰레드에서 비용을 줄이기 어려운 경우
- 실행 순서가 중요하지 않은 루프문
- EX) 주변의 액터 탐색과 마커 업데이트



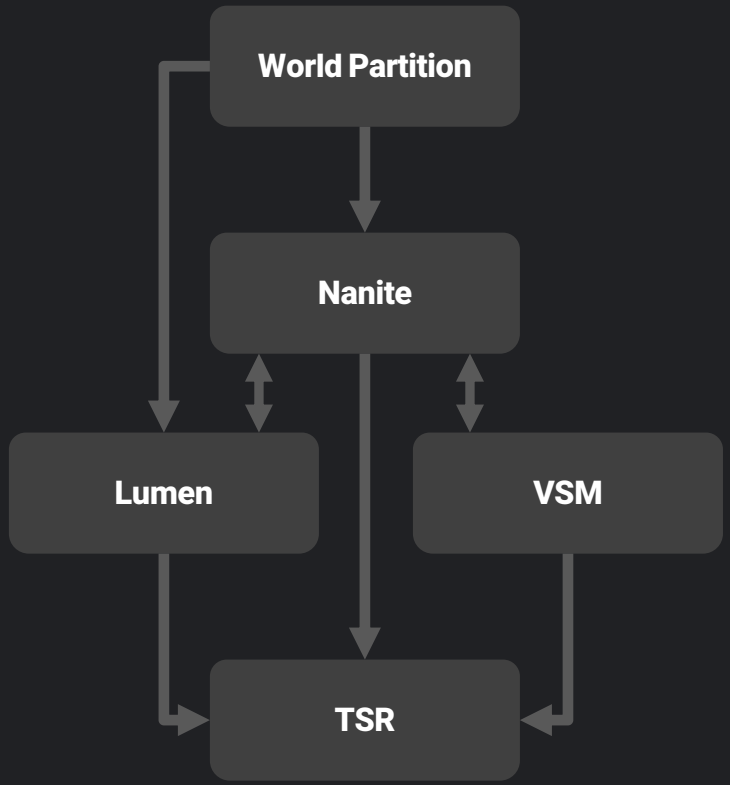
어디에 사용할지 모르겠다면

- 엔진 내에서 사용되는 코드들을 확인하고 학습하여 적절하게 사용
- 시그니피컨스 매니저를 분석하는 것을 추천
  내부에서 ParallelFor 로 작동
- `USignificanceManager::Update`

월드 파티션, 나나이트, 루멘, VSM, TSR 은 연결되어 있다

관계성을 알고 있어야 효율적으로 최적화를 할 수 있음

- UE5의 렌더링은 모든 것들이 연관되어 있음
- 월드 파티션과 HLOD는 나나이트 성능 최적화
- 나나이트는 루멘과 VSM의 성능과 연결
- 나나이트, 루멘 ,VSM의 비용이 TSR에 영향



더 알고 싶다면

- 더 이야기하고 싶지만...
- 언리얼 페스트 2023에 상세하게 설명
- https://www.youtube.com/watch?v=Cb63bHkWkwk&t=4799s
- https://www.youtube.com/watch?v=8eO2xdrDms8&t=872s

HLOD 빌드는 무겁다

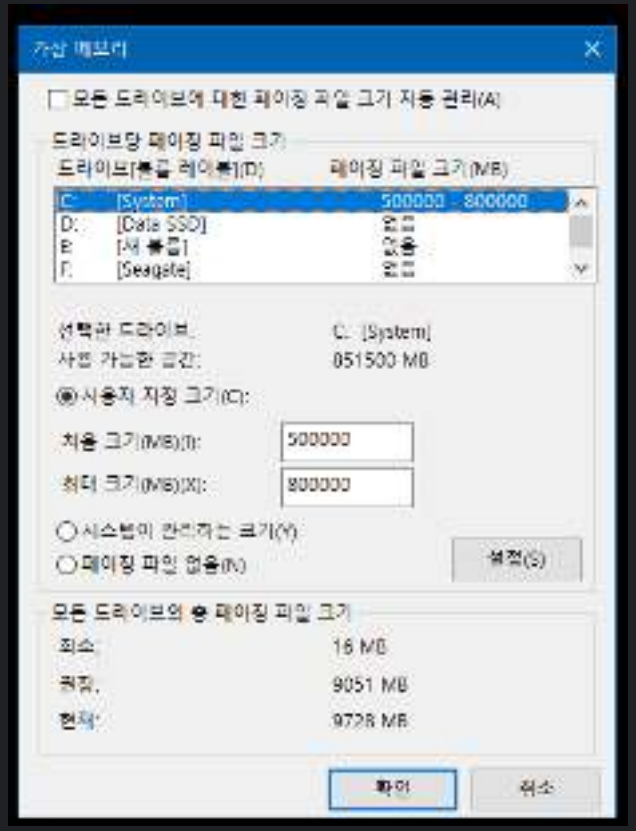
커맨드릿을 돌려서 하는 HLOD 빌드는 여러분들의 생각보다 더 많은 메모리를 요구함

새벽에 돌려 놓은 HLOD 빌드가 항상 실패하는 이유

- 프로젝트 초반에는 성공하던 HLOD 빌드
- 프로젝트가 진행되면서 월드 파티션으로 제작된 레벨의 크기도 매우 커지며 실패를 하기 시작함
- HLOD에 머징 옵션으로 된 액터가 늘어나면 문제 발생
- 크기가 커진 만큼 엄청난 메모리가 필요함
- 64GB ? 128GB? 800GB 이상
- 프로젝트에 따라 다르지만 상상 이상으로 많은 메모리가 필요함

HLOD 빌드에 가상 메모리를 사용

- 가상 메모리를 수동으로 설정
- 넉넉하게 1TB
- 속도가 빠른 SSD를 사용
- HDD를 사용하면 너무 오랜 시간 걸림
- HLOD 빌드 중에 가상 메모리로 세팅한 드라이브에 데이터를 추가하면 오류 발생

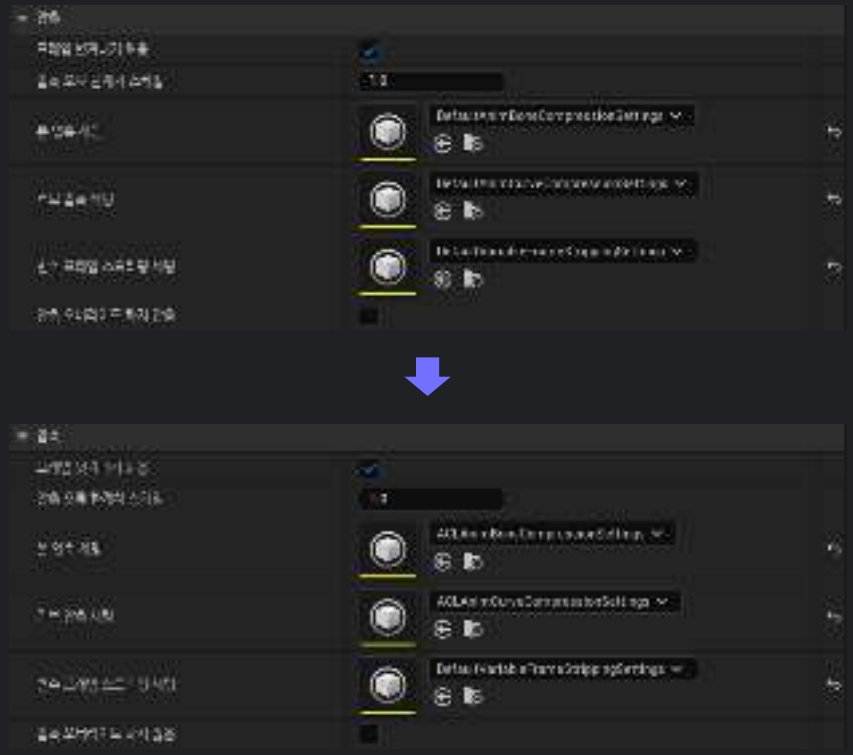


ACL을 사용한 애니메이션 압축

ACL(Animation Compression Library)를 사용하여 애니메이션을 효율적으로 압축하고 빠르게 디코딩할 수 있음

ACL이 각각의 애니메이션에 적용되었는지 확인하기

- 5.4.3 기준 ACL은 UE5의 기본 애니메이션 압축으로 설정되어 있음
- 이전 버전부터 엔진을 여러번 업데이트 했거나 마켓 플레이스에서 예전 버전의 애니메이션을 이주를 통해서 가지고 온 경우 ACL이 기본 애니메이션 압축 설정이 아님
- 애니메이션을 체크하고 애니메이션 압축 설정이 ACL이 아닌 애니메이션을 재설정



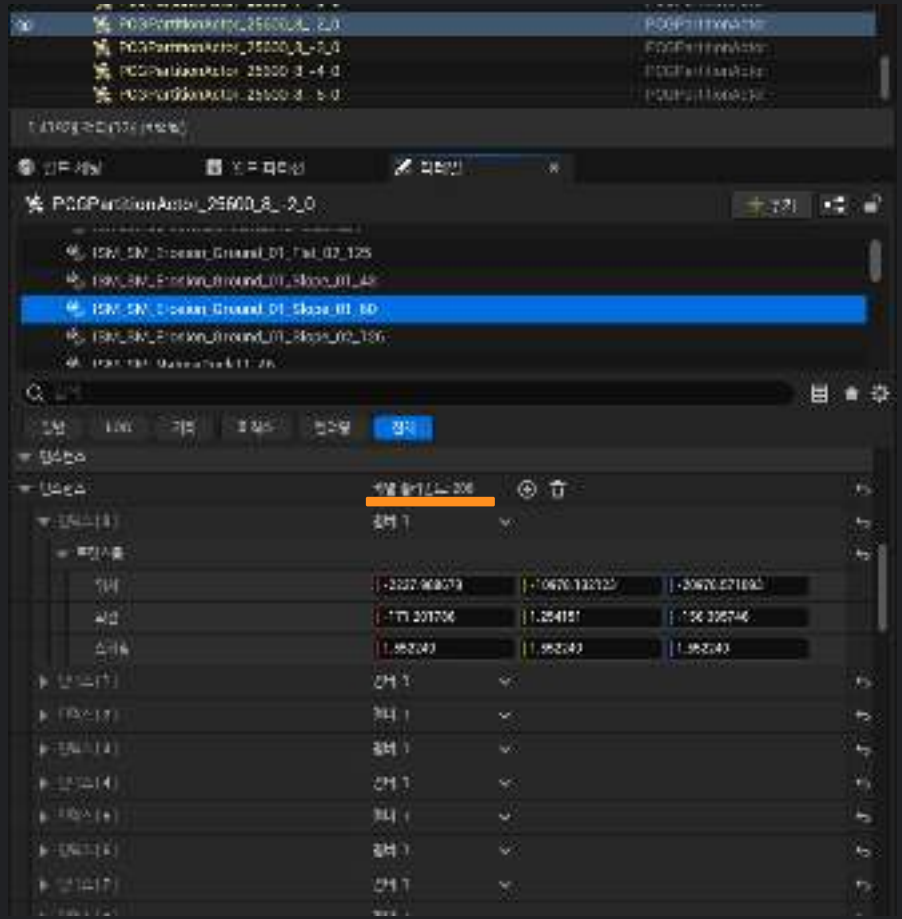
다양한 사례 공유

넥스트스테이지의 사례, 여러분들의 사례, 공유를 통해 서로 발전할 수 있도록

PCG 에서 너무 많은 인스턴스를 넣어서 Hitch가 발생한 경우

사례 분석

- 언리얼 인사이트를 통해서 확인
- 다수의 Actor, Component는 여러 틱에 걸쳐서 비동기 로딩
- 하나의 HISM에 수 많은 인스턴스가 세팅되어 있음
- 하나의 Component에 수 많은 인스턴스는 한 틱에 모두 로딩하기 때문에 Hitch 발생
- 매쉬에 따라 다르지만 20000~30000이 넘어가면 문제가 될 가능성이 높음



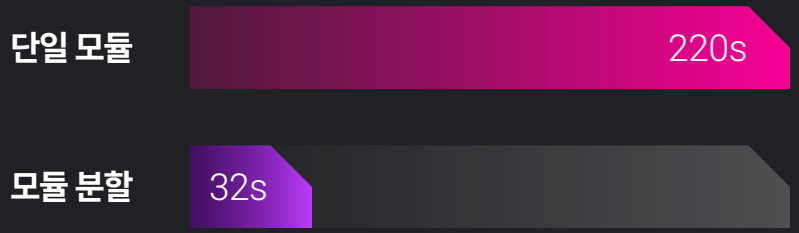
사례 분석

- 대부분의 원인은 월드 파티션의 그리드 별로 PCGActor가 만들어지는 옵션 세팅이 누락
- PCG를 제작할 때 계층형 생성 사용
- 레벨의 PCG Volume에서 Is Partitioned를 True로 설정
- PCG 로직을 제작할 때 부터 여러 액터, 컴포넌트에 나눠서 적용하도록 로직 개선

우리 프로젝트의 컴파일 시간을 줄이는 방법

사례 분석

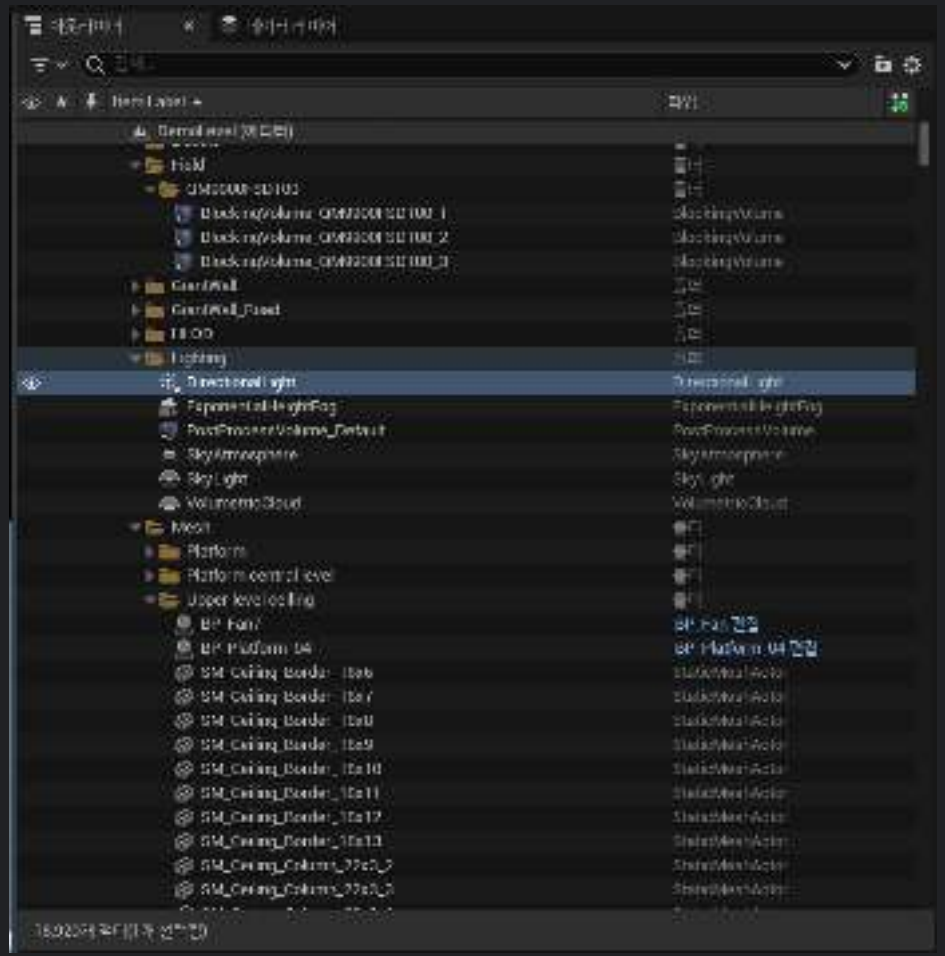
- 프로젝트가 장기화 되면서 높은 컴파일 시간이 문제
- C++ 프로젝트는 기본적으로 오래 걸린다는 생각
- 하나 수정했는데 4~5분씩 걸리면 문제라고 인식
- 대부분 프로젝트에 하나의 단일 모듈을 사용하고 있음
- 플러그인을 적극적으로 사용하여 모듈 분할
- 프로젝트마다 케이스가 다르지만 프로젝트를 진행하면서 플러그인으로 나눌 수 있는 부분들은 나눠서 관리
- 게임 피처를 분석해보면 좋음



월드 파티션에서 여러 액터를 관리하는 방법

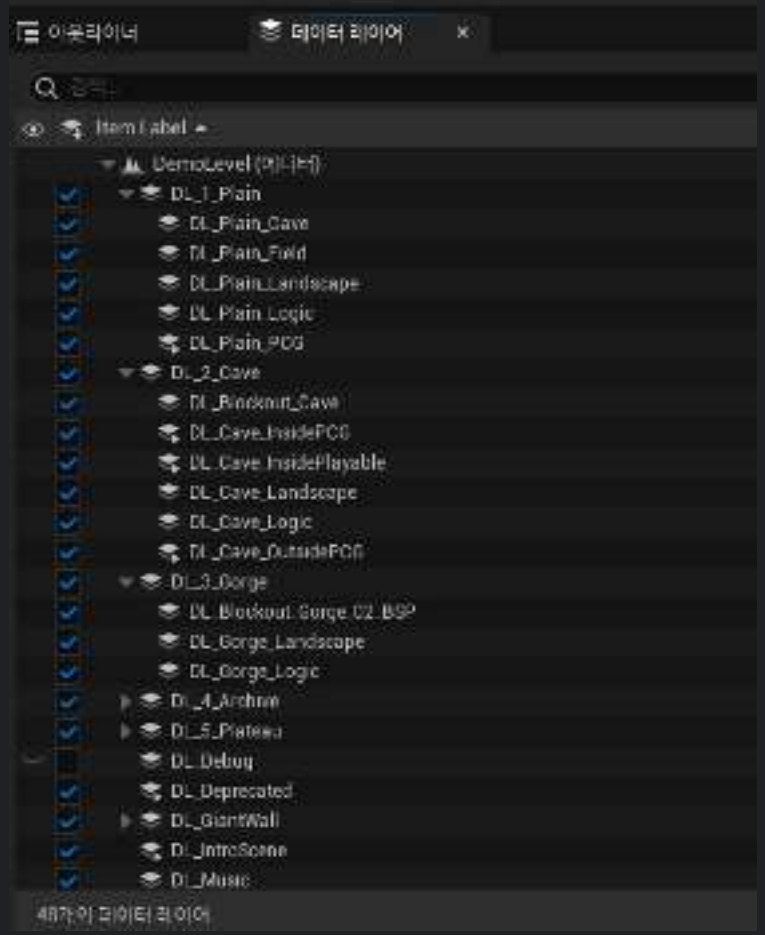
사례 분석

- 레벨의 아웃라이너에 폴더를 만들 수 있어서 많은 분들이 처음에 폴더로 관리해보려고 함
- 에디터에서는 문제가 없지만…
- 패키징된 상태에서 런타임에서 폴더를 통해 이 에셋들을 컨트롤 하려고 하면 문제 발생
- 패키징하면 폴더로 구분할 수 없음



사례 분석

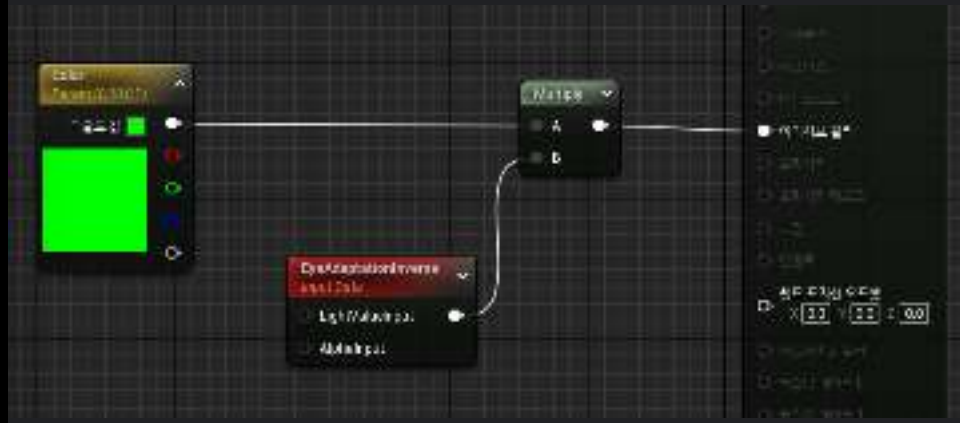
- 데이터 레이어 사용
- 여러 데이터 레이어를 연관되도록 만들 수 있음
- 런타임에서 데이터 레이어를 컨트롤 할 수 있는 서브 시스템
- 데이터 레이어와 별개로 런타임에 컨트롤 해야 되는 경우에는 Actor Tag로 구분하여 사용



밝은 라이팅 환경에서 이펙트가 보이지 않는 문제

사례 분석

- 월드 빌딩을 하게 되면 포스트 프로세스에서 눈 적응성 세팅(Eye Adaptation)을 통해서 밝기 제어
- 이펙터가 대응해야 하는 밝기의 범위가 너무 벌어짐
- 머테리얼에서 EyeAdaptationInverse을 사용하여 눈적응성에 대응
- 눈적응성의 영향을 받지 않도록 하는 이펙트 머테리얼에 사용
- 프로젝트에 따라서 머테리얼을 다양하게 컨트롤 할 수 있음



메타휴먼을 만드는 방법과 사례 공유

사례 분석

- 메타휴먼 에디터를 통해 기본적인 메타휴먼 제작
- 매시 투 메타휴먼을 통해서 좀 더 커스터마이징 가능
- 리얼리티한 캐릭터를 빠르게 제작할 수 있음
- 우리 만의 캐릭터를 만들기에는 1% 부족

사례 분석

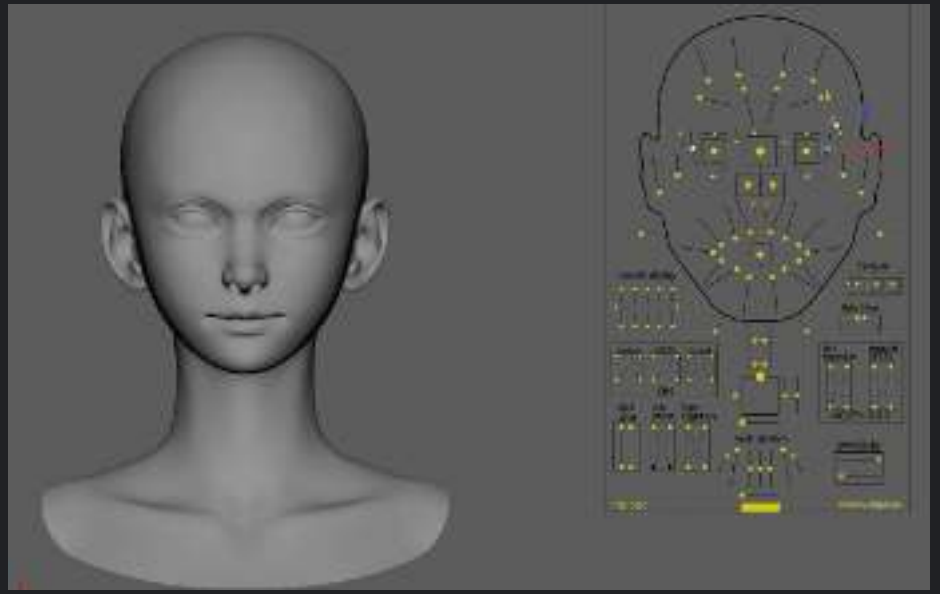
- 넥스트스테이지의 커스텀 메타휴먼
- DNA Calibration 사용
- 눈의 크기에 초점
- 눈이 커져도 자연스러운 방법을 R&D
- 메타휴먼의 모든 기능을 이용하면서도 매력과 개성이 있는 캐릭터를 목표

사례 분석

- 메타휴먼 베이스
- 메타휴먼 애니메이터 사용 가능
- 메타휴먼의 높은 생산성은 그대로
- 기존 메타휴먼에서 할 수 없었던 커스터마이징

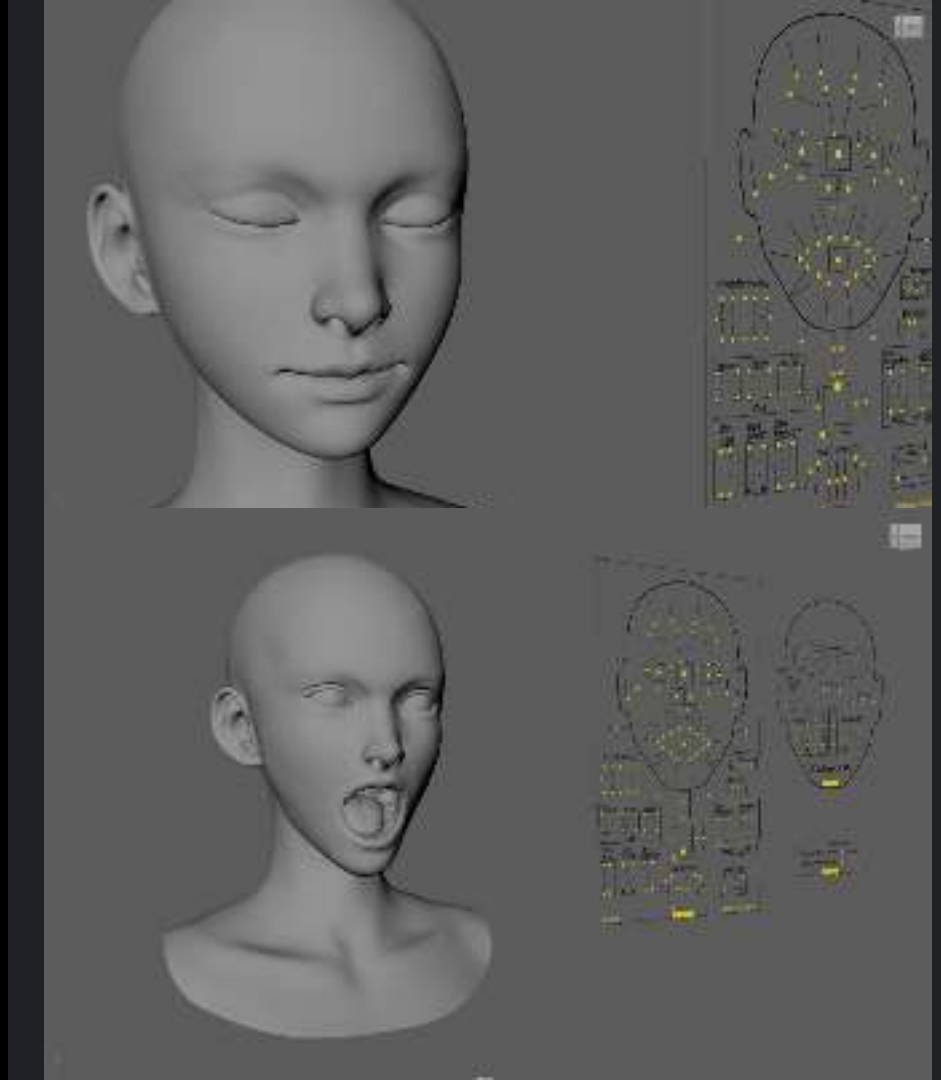
사례 분석

- 넥스트스테이지에서는 DNA Calibration을 통해서 기존의 메타휴먼에서 하지 못한 안구 스케일 조정
- 눈 크기에 따라서 표정의 어색함을 줄이기 위한 몇가지 모프 수정



사례 분석

- 모든 부분을 수정하지 않음
- 메타휴먼 애니메이터로 테스트 후 어색한 부분만 수정
- 최소한의 수정으로 최대의 효과
- 수정과 확인의 반복



사례 분석

- 모프를 수정한 컨트롤러 리스트
- 기본 조형을 기준으로 하나씩 수정하면서 확인
- 모델러가 엔진에 넣어보고 확인까지 하는 파이프 라인
- 빠르게 제작하고 피드백을 통해서 발전

CTRL_L(R)_brow_down
CTRL_L(R)_eye_blink
CTRL_L_eye,CTRL_R_eye
CTRL_L_eye,CTRL_R_eye
CTRL_L(R)_eye_blink
CTRL_L(R)_eye_squintInner
CTRL_C_jaw
CTRL_L(R)_mouth_upperLipRaise
CTRL_L(R)_mouth_sharpCornerPull
CTRL_L(R)_mouth_cornerPull
CTRL_L(R)_mouth_dimple
CTRL_L(R)_mouth_stretch
CTRL_L(R)_mouth_lowerLipDepress
CTRL_L(R)_funnel
CTRL_L(R)_funnelUD
CTRL_L(R)_mouth_pressU
CTRL_L(R)_mouth_pressD
CTRL_L(R)_mouth_towards
CTRL_L(R)_mouth_towardsD

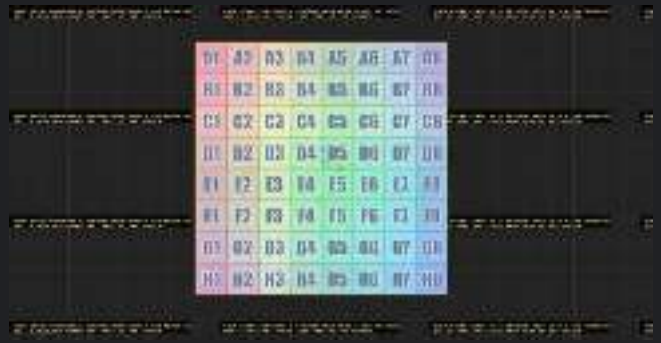
사례 분석

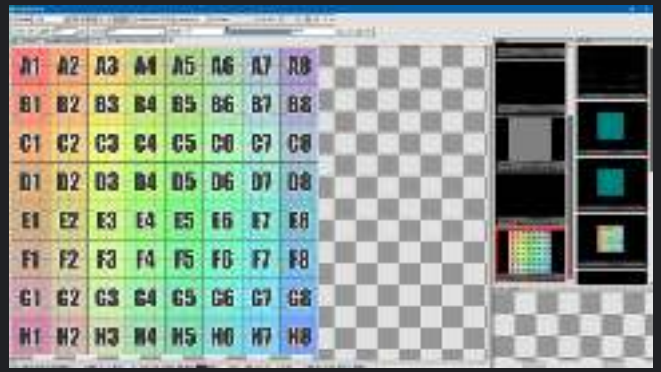
- R&D 할 부분이 많지만 목표에 다가가는 중
- 아직 정답은 없지만 메타휴먼은 매우 좋은 솔루션

Virtual Texture 사용 사례

사례 분석

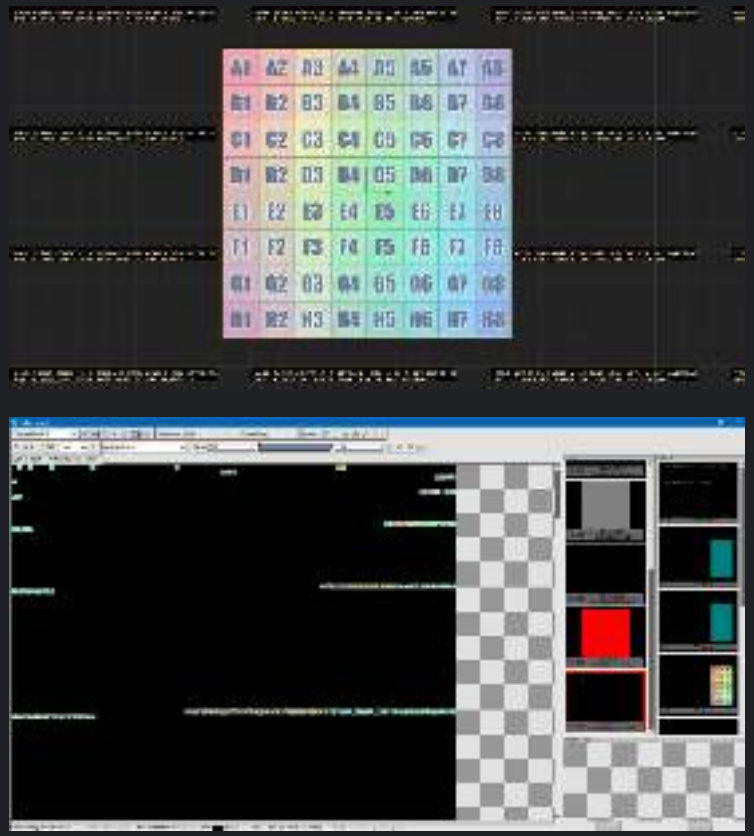
- VT가 좋다고 해서 세팅은 하는데 어떤게 좋은지 명확하지 않고 언제 VT 텍스처를 사용해야될지 애매한 경우
- NonVT 텍스처는 한번 로드되면 언로드 될 때까지 텍스처 전체를 메모리에 할당
- 일부분만 보이더라도 전체 텍스처가 메모리에 할당





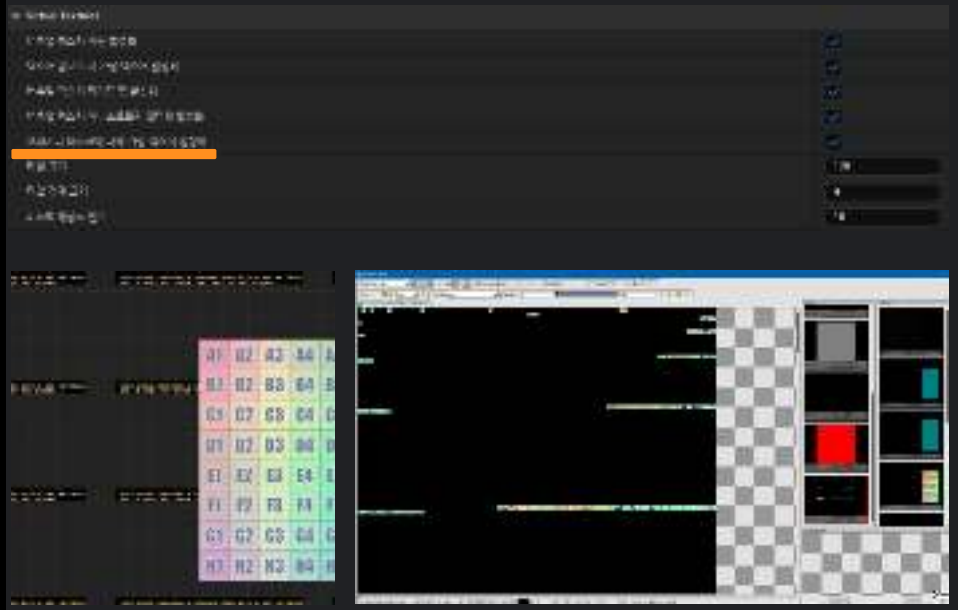
사례 분석

- Virtual Texture는 화면상에 그려지는 부분만 메모리에 할당
- 정해진 Virtual Texture Pool 내에서 NonVT보다 효율적으로 메모리 관리 가능



사례 분석

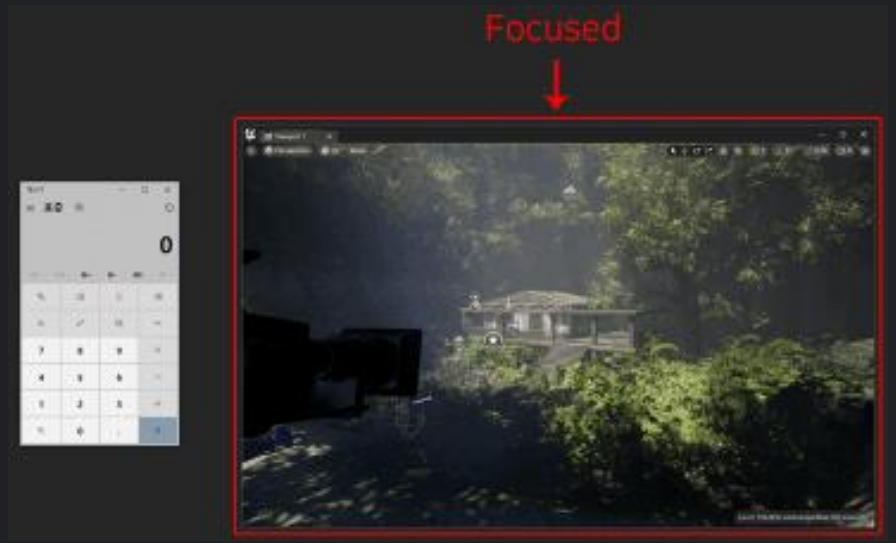
- 넥스트스테이지에서는 대부분의 텍스처를 Virtual Texture로 사용
- 화면에 즉시 표시해야 하는 경우에 NonVT 텍스처를 사용
- 예전 버전에서는 마스크 텍스처는 VT를 사용할 수 없었지만 [프로젝트 세팅]-[오퍼시티 마스크에 대해 가상 텍스처 활성화] 를 통해 사용 가능
- Virtual Texture에서 무슨일이 벌어지는지 쉽게 확인하기 위해서 r.VT.DumpPoolUsage 사용
- RenderDoc을 사용해서 확인 가능

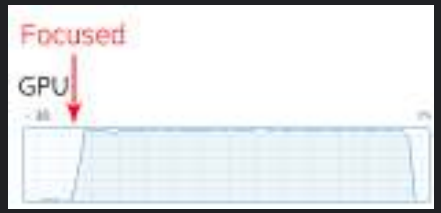


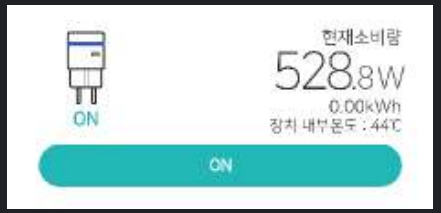
ESG 적용 사례, 전기요금을 절감하는 방법

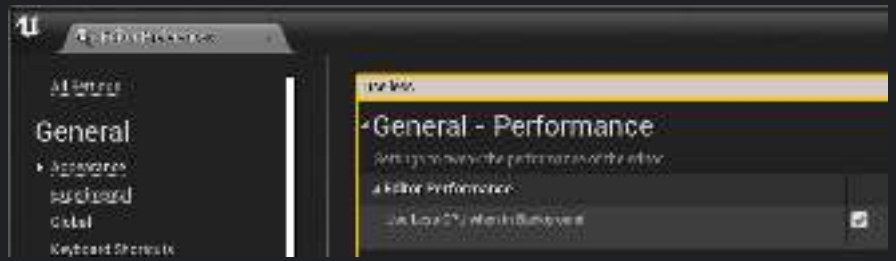
사례 분석

- 언리얼 엔진은 그래픽 카드를 항상 많이 사용
- 엔진 기본 세팅 중에 [Use Less CPU when in Background] 옵션을 통해 창이 포커스 되어 있지 않은 경우 전체 사용량을 줄이는 옵션
- 다만 엔진 내에서 ML(Machine Learning) 를 통해 학습하는 경우에는 해당 옵션을 사용하지 않아야 함



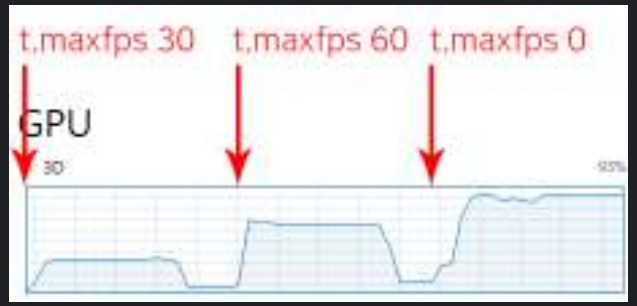






사례 분석

- 에디터에서 120fps가 필요한 사람은 얼마 없음
- 모니터 출력주파수보다 높은 프레임은 사실상 낭비
- 기본 세팅을 60fps로 줄이면 비용 감소
- t.maxfps를 통해 에디터에서 프레임 제한
- 하나 하나의 사용량 차이가 모이면 큰 차이가 됨



사례 분석

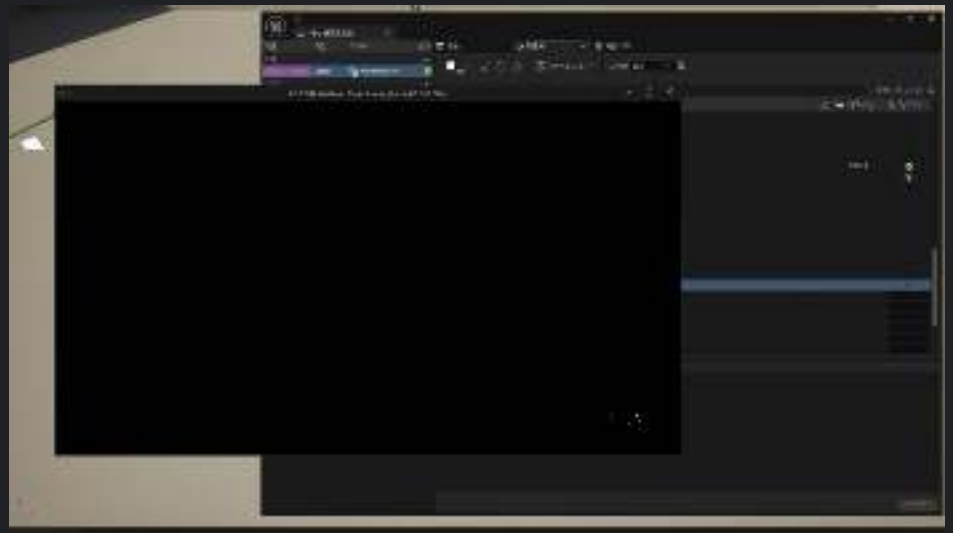
- 에디터를 실행할 때 마다 명령어 입력의 귀찮음
- 에디터를 수정하긴 귀찮음
- 내부에서 임시로 만들어 사용하는 플러그인 공유

https://github.com/Next-Stage-Inc/NSEnergySaver

Functional Test를 사용하여
액션 QA 자동화

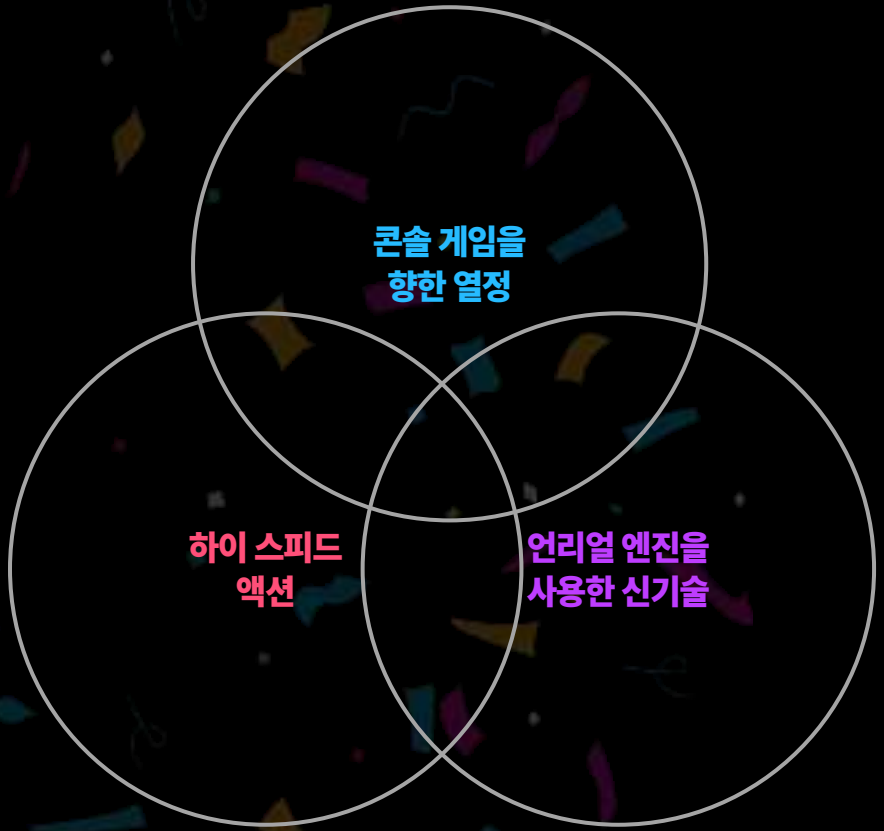
Functional Test

- 엔진에 내장된 테스트 도구
- 넥스트스테이지에서는 각각의 액션들이 잘 돌아가는지 확인하는데 사용
- GAS로 만들어진 액션들을 기반으로 간단한 매크로를 사용하여 액션이 제대로 작동하는지 검사
- 수많은 액션과 콤보를 자동으로 확인하는데 사용
- 액션이 항상 안정적으로 돌아가는지 매일 자동으로 검사



WE ARE HIRING!

새롭고 재미있는 하이 스피드 액션을
같이 만들 분들을 찾고 있습니다!



감사합니다.

— (주)넥스트스테이지 박준수, 강현우