**GAS 모듈의 사용**

**무기 사용, 소모품 사용 등의 캐릭터의 ‘행동’을 어빌리티로 구현**

**무기 사용 ::** 캐릭터에게 [무기를 사용] 한다라는 어빌리티를 줌

[무기를 사용] 하는 어빌리티는 캐릭터에게 있는 Current Weapon Actor에게 자신이 갖고있는 Ability를 사용 하라고 지시함. 이렇게 하는 이유

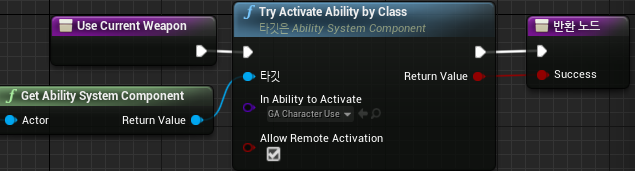
1. 무기 자체에도 어트리뷰트를 부여하는게 구조적으로 깔끔해짐. -> 탄창 관련, 어태치먼트와 같은 웨폰에게만 필요한 정보는 캐릭터와 별개로 돌아가야함.
2. 무기의 발사 방식 ( 주, 보조 발사) 같은 것을 구현 할 때의 편리함 )

**소모품 같은 아이템이 가지고 있는 어빌리티를 사용**할 때에는 Slotted Ability - 인벤토리와 연동되는 별도의 Map - 에 등록하여 인풋이 특정 슬롯의 어빌리티를 사용하게 하는 방식.



해당 함수를 특정 인풋에 바인드하여 사용

**모든 캐릭터가 기본으로 가지고 있어야 하는 스프린트, 웨폰 사용** 같은 어빌리티는 슬롯이 아닌 별도의 인풋에 바인딩 하여 어빌리티 클래스 단위의 사용을 하는것으로 구현.



해당 함수를 특정 인풋에 바인드하여 사용

~~--------------------------------------------------------------------------------------------------------------~~

이하 세부적 구현에 대한 설명

~~--------------------------------------------------------------------------------------------------------------~~

**GPGameAbility**

:: 게임 캐릭터들이 행하는 능력 들의 베이스 ::

캐릭터들의 어트리뷰트 ( GPGameAttributeSet )에 영향을 끼치는 요소를 중점적으로 다룸.

~~--------------------------------------------------------------------------------------------------------------~~

**PROPERTIES :**

TMap<FGameplayTag, FGPGameplayEffectContainer> EffectContainerMap

**키 값**은 “시전자의 몽타주 정보”를 태그로 기록하기 위한 용도

-> 후에 전담 태스크 ( UGPAbilityTask\_PlayMontageAndWaitForEvent )를 통해 처리

**밸류**에 이 어빌리티가 대상에게 가할 GameplayEffect에 대한 정보, 타겟을 구하는 방식을 저장.

FGPGameplayEffectContainer 스트럭쳐의 구성

: TSubclassOf<UGPTargetType> TargetType;

UGPTargetType이 가진 GetTargets 함수를 사용, 해당 클래스의 ClassDefaultObject의 것을 사용함.

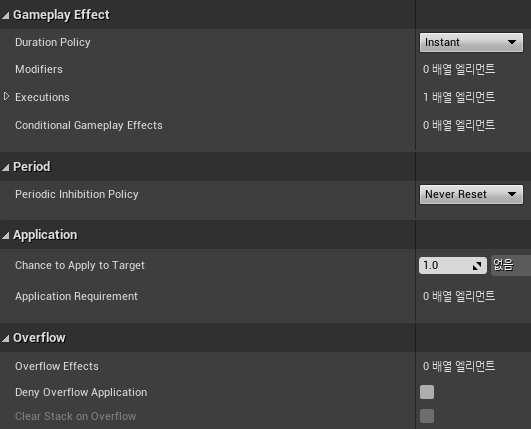
  
From GPGameplayAbility.cpp / line 22~24

타겟 타입은 CPP상에 만들어 둔 것 외에도 BP로 받아서 GetTargets를 오버라이딩 해서 하는 것으로 종류를 늘릴 수 있다.

: TArray<TSubclassOf<UGameplayEffect>> TargetGameplayEffectClasses;

UGameplayEffect는 GameplayAbilitySystem 모듈에 정의 되어있음.

이 안에 단순 어트리뷰트 모디파이, 연산해야 할 부분 ( 데미지를 줄 때 상대의 방어력 같은 부분에 대한 ) 에 대한 연산 방식을 클래스로 구현하여 지정하기. 등등 다채로운 기능을 제공.

  
BP로 GE를 구현했을 때 노출되는 프로퍼티들  
GameplayEffect.h 참조

~~--------------------------------------------------------------------------------------------------------------~~

**FUNCTION :**

**FGPGameplayEffectContainerSpec** **MakeEffectContainerSpecFromContainer**  
(   
 const FGPGameplayEffectContainer& Container,   
 const FGameplayEventData& EventData,   
 int32 OverrideGameplayLevel = -1  
)

인자로 받은 EffectContainer를 통해 타겟을 구하고 적용할 이펙트들을 등록한 외부 전달용 FGPGameplayEffectContainerSpec을 만들어 리턴   
( EventData는 타겟 타입중 UseEventData를 쓰는 것 들을 위함 -> GA 를 만들 때 특수 한 타겟이 필요한 경우 별개의 연산으로 타겟을 구해 EventData에 정보들을 넣어서 전달 하는 식으로 구성 )

FGPGameplayEffectContainerSpec은 FGPGameplayEffectContainer의 외부 전달용

: 타겟 타입이 아닌 직접적인 타겟 액터 정보와 이펙트 정보들을 묶어둠

-> FGameplayAbilityTargetDataHandle, FGameplayEffectSpecHandle 사용  
각각은 FGameplayAbilityTargetData , FGameplayEffectSpec 의 Handle

**FGPGameplayEffectContainerSpec MakeEffectContainerSpec**(  
 FGameplayTag ContainerTag,   
 const FGameplayEventData& EventData,  
 int32 OverrideGameplayLevel = -1  
)

MakeEffectContainerSpecFromContainer 함수를 게임 플레이 태그로 호출 하는 방법  
: 본 클래스의 프로퍼티인 ‘EffectContainerMap’에서 ContainerTag 인자로 Find -> 저장 되어 있는 FGPGameplayEffectContainer 밸류로 위의 MakeEffectContainerSpecFromContainer 를 호출

**TArray<FActiveGameplayEffectHandle> ApplyEffectContainerSpec**(  
 const FGPGameplayEffectContainerSpec& ContainerSpec  
)

리턴 타입 FActiveGameplayEffectHandle은 해당 단일 이펙트가 성공적으로 대상에게 영향을 끼쳤는지를 기록 하는 것 ( GameEffect는 태그 조사를 통해 해당 이펙트가 무시 될지 같은 것도 결정 가능 함 )과 int32 ID 를 가지고 있음.

인자로 받은 ContainerSpec 에 저장된 타겟 액터 정보들과 이펙트 정보들을 가지고 실제 영향을 끼치는 함수를 실행 하고 그 결과로 나오는 FActiveGameplayEffectHandle들을 TArray 형태로 묶어 리턴함

**TArray<FActiveGameplayEffectHandle> ApplyEffectContainer**  
(  
 FGameplayTag ContainerTag,  
 const FGameplayEventData& EventData,  
 int32 OverrideGameplayLevel = -1  
)

위의 Make 함수와 비슷 하게 Tag를 이용하는 방식임

~~--------------------------------------------------------------------------------------------------------------~~

**Handle? )**

Windows 프로그래밍에선 OS 커널단 오브젝트를 직접 사용 하지 못하게 하고 대신 Handle을 제공하여 이용 할 수 있도록 함.   
Windows 프로그래밍에서 이것을 이용 하는 이유로는 관리적 측면과 프로세스가 직접 커널 오브젝트에 접근해 사용하게 된다면 어떤 프로세스가 커널 오브젝트를 사용하는지, 프로세스가 접근해서는 안될 커널 오브젝트에 접근하는지 등을 OS가 파악하기 위해서이다.

여기서 말하는 GameplayAbility 의 Handle들도 비슷한 목적에 의해 사용된다.

Handle for Targeting Data. This servers two main purposes:

-Avoid us having to copy around the full targeting data structure in Blueprints

-Allows us to leverage polymorphism in the target data structure

-Allows us to implement NetSerialize and replicate by value between clients/server

-Avoid using UObjects could be used to give us polymorphism and by reference passing in blueprints.

-However we would still be screwed when it came to replication

-Replication by value

-Pass by reference in blueprints

-Polymophism in TargetData structure

제작자의 코멘트

내용물에 대한 직접 접근 제한 ( 넷시리얼라이즈 관련도 포함 ), 다형성 보장, 레플리케이션 으로 요약 가능하다

~~--------------------------------------------------------------------------------------------------------------~~

내부 구현? )

래핑할 대상을 TSharedPtr 로 감싸고 유틸리티 함수 ( Clear, IsVaild, NetSerialize ) 등을 같이 넣음

TSharePtr은 C++의 shared\_ptr과 비슷한 역할

- 쉐어드 포인터는 참조하는 오브젝트를 소유

- 참조 횟수를 계산함

- 원시 포인터 하나를 여러 소유자에게 할당하려고 할 경우 사용

언리얼에서의 몇몇 추가 피쳐 :

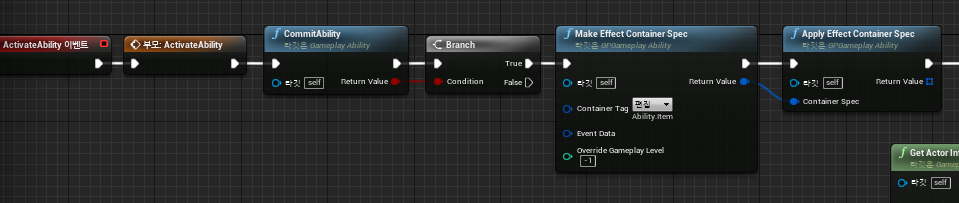
- 무기한으로 오브젝트의 소멸을 방지

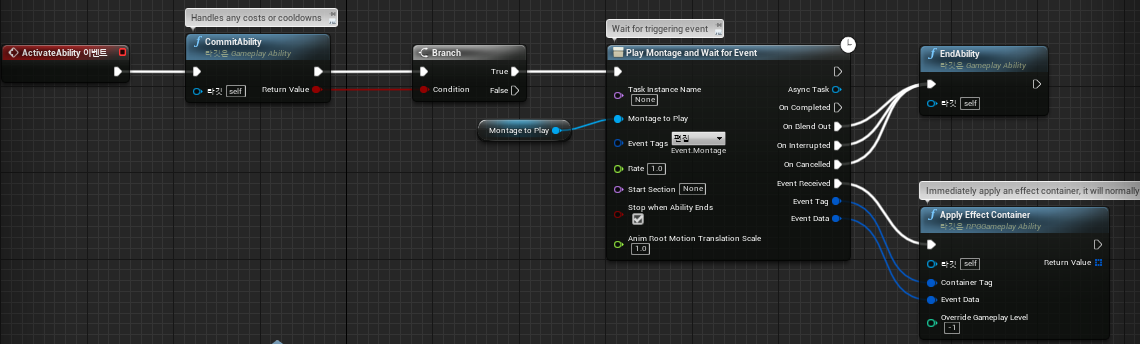
- 해당 스마트 포인터에 대한 참조 횟수가 0일 때 내부 raw data 에 대한 GC 처리에 들어감

<https://docs.unrealengine.com/ko/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/SmartPointerLibrary>  
참조 :: UE4 스마트 포인터에 대한 문서

~~--------------------------------------------------------------------------------------------------------------~~

**실제 사용 예시**

GA\_Consumable\_Base ( 몽타주 플레이 없는 GA의 사용 예시 )

  
ARPG 예제의 몽타주 플레이 태스크를 사용한 GA의 사용 예시

~~--------------------------------------------------------------------------------------------------------------~~

**GameplayEffect 세부 설명**

**어트리뷰트에 영향을 줄 수 있는 모든 효과들의 베이스**

어트리뷰트?) GAS 모듈 내에서 제공하는 무언가에 대한 ‘수치 / 값’에 관련된 것을 구현 하기 편하게 만들어 놓은 것 ex) HP, 탄창, 데미지, 방어력

여러 어트리뷰트들을 모아둔게 어트리뷰트 셋이며 어트리뷰트는 레플리케이션이 구현되어 있음 ( 사용 여부 또한 지정 가능 ). 또한 캐릭터에게 부착된 GAS 컴포넌트 안에 어트리뷰트의 실제 값이 존재함.

게임 플레이 이펙트는 어빌리티가 대상에게 주는 효과 / 어빌리티의 코스트 / 어빌리티의 쿨다운 등 여러가지 곳에 쓰일 수 있으며 캐릭터가 가지고 있는 태그에 따라 효과를 다르게 주는 등의 여러가지 기능이 구현되어 있음.

**프로퍼티 설명**

Duration Policy 를 설정하면 Peroid 관련이 활성화 - > Peroid 는 틱 이펙트를 가할 때 쓰임

Peroid 의 시간단위는 틱 interval 을 설정 하는것이고 execution 관련 불린은 틱 시작시에도 이펙트를 적용 할 것인지에 대한 설정임.

이펙트의 지속시간이 시간이 끝났을 때 실행할 이펙트는 Expiration 항목에 지정할 수 있음.

Infinite로 한번만 이펙트를 적용시키고 원하는때에 이펙트를 끄는 방식은 구현 불가 -> Duration을 매우 길게잡거나 별도의 구현 방식을 적용시킨다면 가능 할 지도.

Tags 에서 이 이펙트가 무슨 태그를 갖고 있는지, 액터에게 무슨 태그를 줄것인지 설정 가능.

태그에 따른 해당 이펙트의 삭제, 켜지고 꺼지는 것에 대한 구현까지 가능.

~~--------------------------------------------------------------------------------------------------------------~~

**PlayMontageAndWaitForEvent 작동 방식**

이 함수는 UGPAbilityTask\_PlayMontage 클래스 내의 함수 AbilityTask (부모) 클래스는 GameplayAbility에 종속적이며 어빌리티가 끝날 때 태스크도 같이 소멸 된다는 것이 보장되어있다.

이 함수는 UGPAbilityTask\_PlayMontage 내의 스태틱 함수로 함수 안에서 NewAbilityTask 함수를 통해 UGPAbilityTask\_PlayMontage를 인스턴스화 하여 주어진 몽타주를 재생하고, 재생된 몽타주 내의 애니메이션 스테이트 노티파이 (ASN) 가 보내는 이벤트를 받고, 받은 시점에서의 몽타주 상황에 따른 델리게이트를 실행하는 함수이다. 이벤트가 올 때까지 몽타주 진행 상황과는 비 동기적으로 다른 작업을 할 수 있다.

  
ANS\_FireWeapon과 해당 ANS를 쓰는 몽타주의 예시

~~--------------------------------------------------------------------------------------------------------------~~

**Projectile 관련 - Class [AGPProjectileBase]**

프로젝타일의 경우 GameplayAbility 가 제공하는 함수들의 직접적인 사용이 어려워 별개의 방식으로 대상에게 이펙트를 가하는데 이는 GPBPFuncLibrary에 정의되어있는 몇 개의 함수들을 사용한다.

프로젝타일이 직접적으로 GameplayAbility를 갖게 하는 방법도 생각해 볼 수 있겠지만 오버헤드가 커질 가능성이 있으므로 앞서 말해둔 GameplayEffectContainerSpec이라는 미리 계산 방식을 구해둔 외부 전달용 이펙트 스트럭쳐를 프로젝타일 액터에게 전달 함으로써 프로젝타일이 대상에게 가할 이펙트를 저장하게 하고 자식 블루프린트에서 각자 이펙트를 가할 대상을 구하도록 함 ( 단일 대상, Radial, 기타 등등 )

GPBPFuncLibrary에 정의되어있는 함수 중 AddTargetsToEffectContainerSpec과 ApplyExternalEffectContainerSpec를 사용하게 된다.

**AddTargetsToEffectContainerSpec** : GameplayEffectContainerSpec 안의 타겟 데이터에 직접적으로 Actor를 넣는 함수  
**ApplyExternalEffectContainerSpec** : GameplayEffectContainerSpec 안의 타겟들에게 같이 저장된 이펙트 컨테이너를 적용시키는 함수

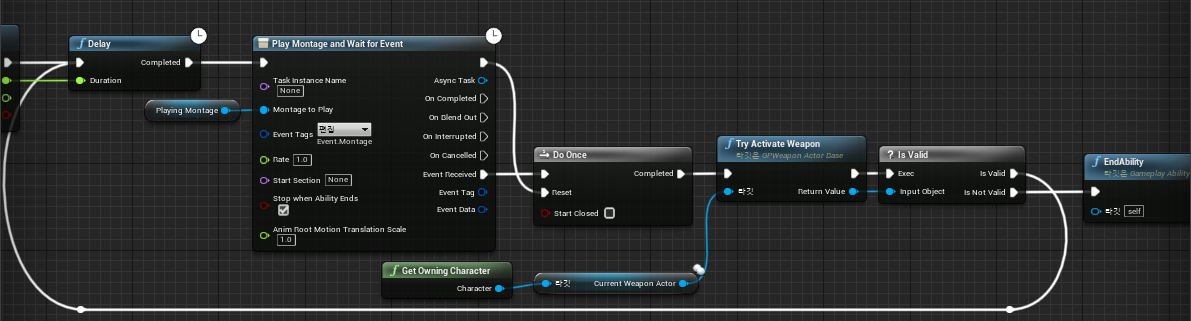
  
BP\_RadialProjBase의 Hit – 스피어 트레이스를 해서 범위 내의 Pawn 에게 이펙트를 가하는 부분

~~--------------------------------------------------------------------------------------------------------------~~

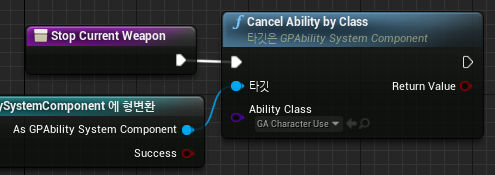
**토글형 어빌리티**

토글형이 될 어빌리티는 EndAbility를 내부적으로 호출하지 않고 계속 사용하는 부분을 리커시브하게 만드는 식으로 구현.

EndAbility를 인풋단이나 별도의 시점에서 호출하여 끝냄.



무기 발사 방식이 FullAuto일 때 의 방식 ( 일종의 토글형 이라고 볼 수 있음 )



강제로 EndAbility가 호출되면서 루프가 종료됨

~~--------------------------------------------------------------------------------------------------------------~~

**GameplayTag**

~~이벤트 그래프에서 태그를 얻어오는 함수를 호출할시 문제 발생 -> 퓨어 펑션으로 불러오는 기능을 별도로 만드는게 바람직함.~~

GAS 컴포넌트를 액터에게 부착할 시 게임 플레이 태그로 어빌리티의 실행 조건, 버프/디버프의 상태 등을 관리 할 수 있도록 태그 컨테이너가 내부적으로 미리 만들어져 있음.

게임 플레이 이펙트에서 이와 관련되어 액터에게 태그를 부여할 수 도 있고 제거 할 수도 있으며 이펙트를 가할 때 특정 태그가 있어야만 이펙트를 가하는 등의 여러가지가 가능함.

~~--------------------------------------------------------------------------------------------------------------~~