**Class [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.

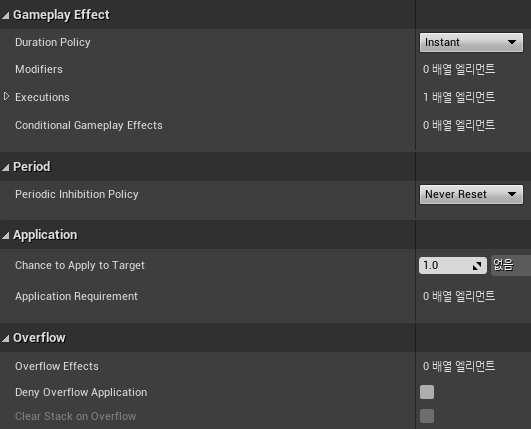
  
From GPGameplayAbility.cpp / line 22~24

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 :

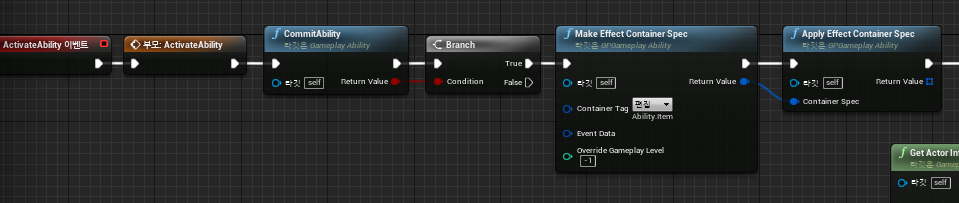
- 무기한으로 오브젝트의 소멸을 방지

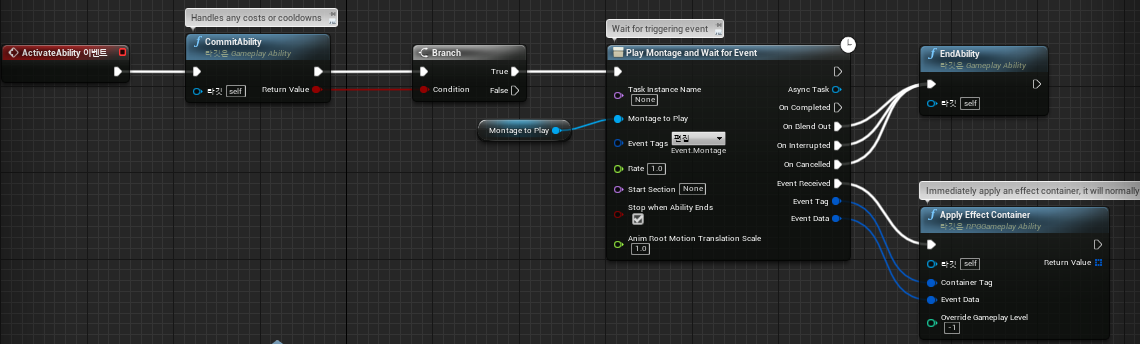
- 해당 스마트 포인터에 대한 참조 횟수가 0일 때 내부 raw data 에 대한 GC 처리에 들어감

<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