**인벤토리의 구현**

**개요**

UGPPlayerControllerBase에 InventoryData가 존재하여 그 곳에 아이템 데이터를 수납하는 형식  
IGPInventoryInterface를 통해 인벤토리 데이터가 필요한 클래스들에게 각자 데이터를 얻는 방식을 구현하게 함

무기 장착이나 기능을 사용 가능하게 할 아이템은 SlottedItems 에 보관하여 사용 함  
SlottedItems : 기본적으로 세이브 데이터를 로드할 때 뚫어주는 형식으로 구현함

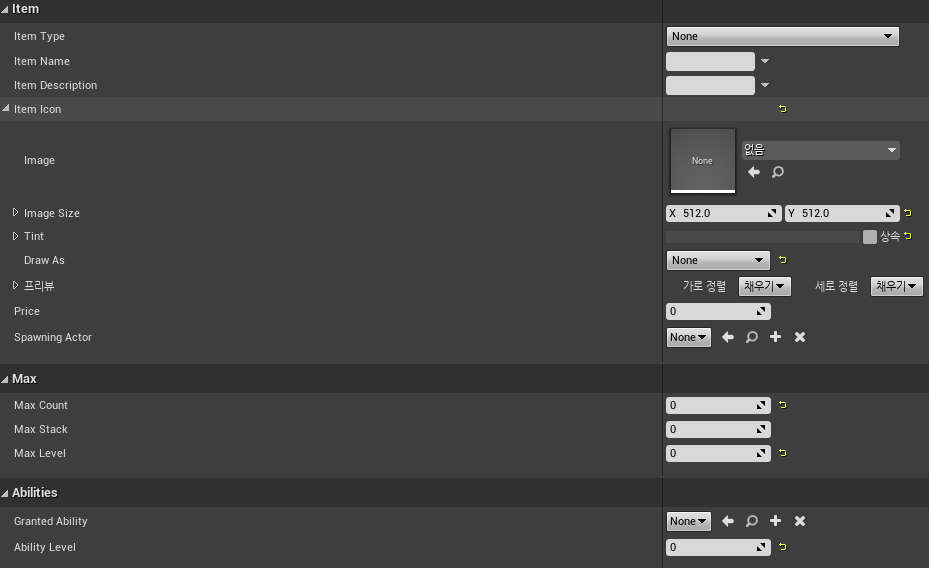
이하 각 클래스 설명

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

**class [UGPItem]**

:: 아이템의 정보를 저장하는 데이터 어셋 ::

해당 아이템의 정보들을 기록하는 데이터 어셋 이며 픽업 아이템이나 아이템 액터들은 이것을 참조함.  
주도적으로 뭔가 하는 클래스가 아니기 때문에 이미지로 프로퍼티 리스트를 대체



아이템 타입, 이름 등등의 정보가 있으며 아이템이 가진 어빌리티를 지정할 수 있다

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

**struct [FGPItemData]**

:: 아이템이 인벤토리에 들어가 있을 때의 데이터 ::

아이템 개수, 레벨등의 정보를 가지고 있으며 인벤토리를 업데이트 할 때 최대 수량을 체크하여 clamp 해주는 함수를 내장하고 있음. - 세이브, 시리얼라이즈를 하기 위한 데이터 래핑용 스트럭쳐 -

사용 할 때 어떻게 쓰이는지는 PlayerController 단에서 설명함.

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

**struct [FGPItemSlot]**

:: 아이템 슬롯의 정보 ::

참고 ) 타게임의 Equipmented 와 이 게임의 Slotted 가 비슷한 의미로 쓰인다고 생각 하면 됨

위에서 쓴 아이템 타입과 슬롯 넘버 정보를 가지고 있음

사용 할 때 어떻게 쓰이는지는 PlayerController 단에서 설명함.

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

**Class [IGPInventoryInterface]**

:: 인벤토리 데이터에 대한 방식을 구현 하게 하는 인터페이스 ::

**FUNCTIONS :**

**const virtual TMap<UGPItem\*, FGPItemData>& GetInventoryDataMap() const  
 = 0**

**const virtual TMap<FGPItemSlot, UGPItem\*>& GetSlottedItemMap() const  
 = 0**

각각 인벤토리 데이터와 슬롯된 아이템 맵을 가져오는 함수를 구현하게 함.

**virtual FOnInventoryItemChangedNative& GetInventoryItemChangedDelegate()  
 = 0**

**virtual FOnSlottedItemChangedNative& GetSlottedItemChangedDelegate()  
 = 0**

**virtual FOnInventoryLoadedNative& GetInventoryLoadedDelegate()  
 = 0**

각각의 델리게이트들을 가져오는 부분을 구현하게 함.

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

**class [GPPlayerControllerBase]**

:: 플레이어 컨트롤러 – 인벤토리 데이터를 보관 ::

인벤토리 관련 부분만 중점적으로 설명

**PROPERTIES :**

**TMap<UGPItem\*, FGPItemData> InventoryData**

인벤토리 아이템을 저장하는 Map

Map 안의 각 element들은  
Key의 데이터 어셋에 해당하는 아이템을   
Value로 들어간 아이템 데이터가 갖고있는 수량 정보 만큼 갖고있고, 레벨 정보 만큼 강화 되었다 라는 의미.

**TMap<FGPItemSlot, UGPItem\*> SlottedItems**

슬롯에 장착된 아이템을 보관하는 Map

Map 안의 각 element들은  
Key의 아이템 슬롯에   
Value로 들어간 아이템이 있음을 의미

**FOnInventoryItemChanged OnInventoryItemChanged**

**FOnInventoryItemChangedNative OnInventoryItemChangedNative**

인벤토리 내에 변화가 생겼을 때 Broadcast할 델리게이트 ( Native는 C++용 이라 생각하면 된다. )

**FOnSlottedItemChanged OnSlottedItemChanged**

**FOnSlottedItemChangedNative OnSlottedItemChangedNative**

슬롯에 장착된 아이템에 변화가 생겼을 때 Broadcast할 델리게이트

BP로 GE를 구현했을 때 노출되는 프로퍼티들  
GameplayEffect.h 참조

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

**FUNCTION :**

**void InventoryItemChanged**( UGPItem\* Item, FGPItemData ItemData )

**void SlottedItemChanged**( FGPItemSlot ItemSlot, UGPItem\* Item )

둘다 BlueprintImplementable로 되어있으며 BP에서의 간단한 구현을 위해 만들어 놓은 함수

**bool AddInventoryItem**(   
 UGPItem\* NewItem,  
 int32 ItemCount = 1,  
 int32 ItemLevel = 1,  
 bool bAutoSlot = true  
)

**bool RemoveInventoryItem**( UGPItem\* RemoveItem, int32 RemoveCount = 1 )

아이템 추가, 제거를 담당하는 함수  
내부적인 구현이 새 아이템 데이터를 덮어쓰는 형태로 되어있음. -> 현재 구조에서 레벨이 다른 같은 종류의 아이템을 두개 가질수는 없음 ( 리팩토링 요소 )  
두 함수를 실행하면 각각의 Nofity 함수를 실행 ( 노티파이 함수는 각자의 델리게이트들을 Broadcast하고 Changed 함수를 부름 )

**void GetInventoryItems**  
(TArray<UGPItem\*>& Items, FPrimaryAssetType ItemType)

**int32 GetInventoryItemCount**  
(UGPItem\* Item) const

**bool GetInventoryItemData**  
(UGPItem\* Item, FGPItemData& ItemData) const

**UGPItem\* GetSlottedItem**  
(FGPItemSlot ItemSlot) const

**void GetSlottedItems**  
(TArray<UGPItem\*>& Items, FPrimaryAssetType ItemType, bool bOutputEmptyIndexes)

정보 접근을 위한 함수들, 별도의 설명은 생략

**bool SetSlottedItem**  
( FGPItemSlot ItemSlot, UGPItem\* Item )

슬롯에 아이템을 장착 하는 함수 = **슬롯을 새로 생성 하는 함수가 아님!**

**void FillEmptySlots()**

**void FillEmptySlotWithItem**(UGPItem\* NewItem)

비어있는 슬롯을 자동으로 채우는 함수

인자 없는 버전은 인벤토리내의 모든 아이템을 가지고 실행