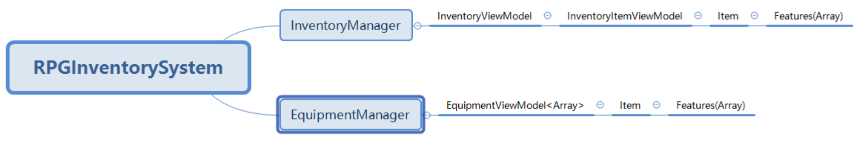
Documentation
RPGInventorySystem(MVVM)

Feature

1. Using UMG View Model (MVVM) framework
2. Network replication using Iris
3. Use Feature to process the information and logic of item assets, which is highly scalable. The Demo uses GAS skill system and CommonUI. You can also expand your own skill system through Feature subclasses.

Summary

RPGInventorySystem uses MVVM ViewModel as the item instance and binds it with the UMG view by introducing ViewModel.



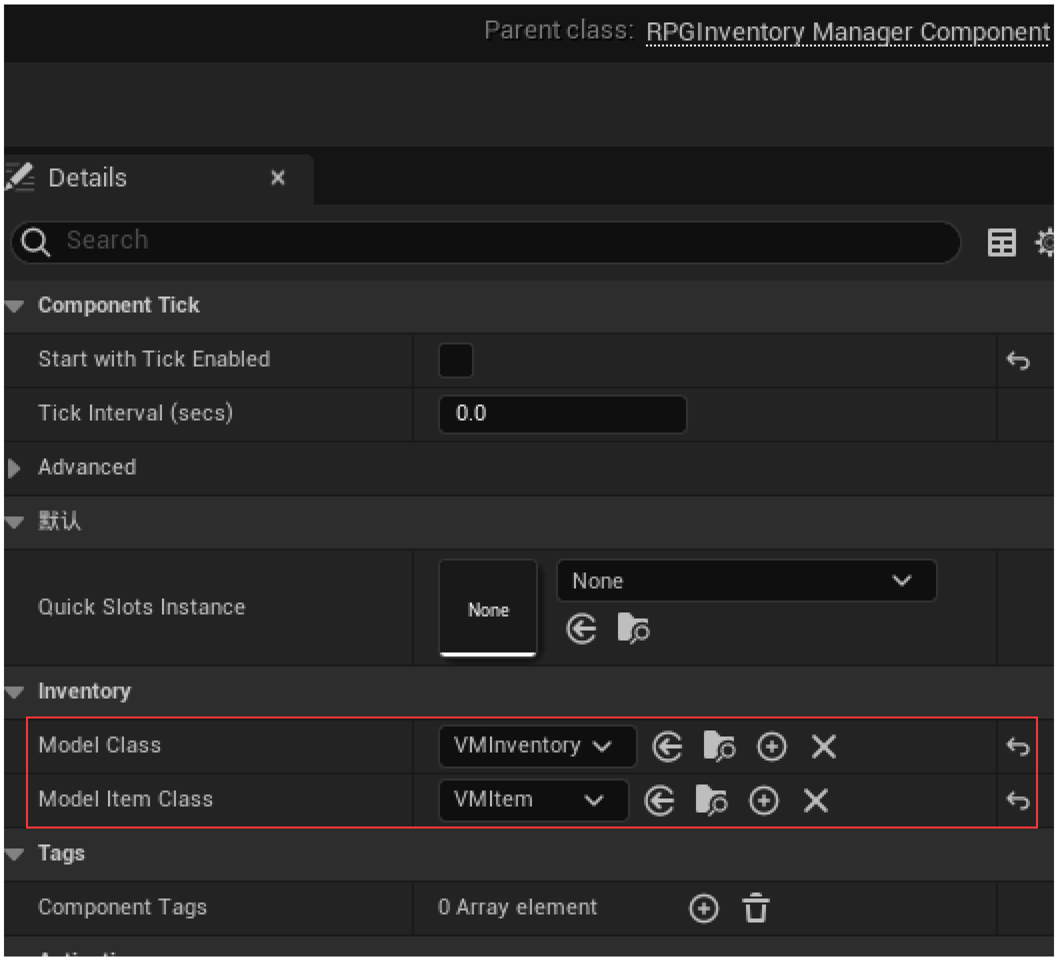
1. InventoryManager

1.3 InventoryItemViewModel,InventoryViewModel

InventoryManager

InventoryManagerIt is a component mounted on PlayerController and can be mounted directly or added through GameFeature.

Modify the ModelClass and ModelItemClass on the InventoryManager to use the ViewModel subclass derived from the blueprint to extend more functions.


Parent class: RPGInventory Manager Component

Details

Search

Component Tick

Start with Tick Enabled

Tick Interval (secs) 0.0

Advanced

默认

Quick Slots Instance None None

Inventory

Model Class VMInventory

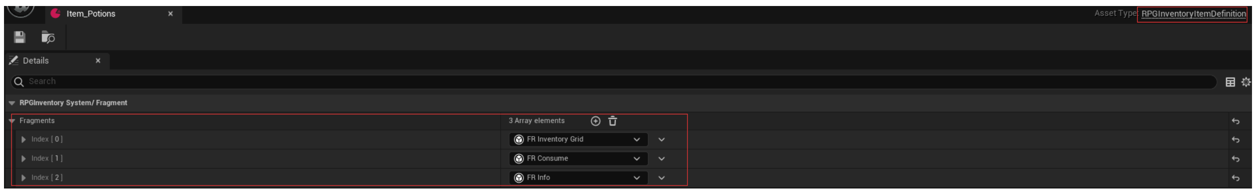
Model Item Class VMItem

Tags

Component Tags 0 Array element

- **RPGInventoryItem**

 This is a DataAsset type. Each RPGInventoryItem represents an item and information can be added by adding Feature.


Item_Potions

Asset Type RPGInventoryItemDefinition

Details

Search

RPGInventory System/ Fragment

Fragments 3 Array elements

Index [0] FR Inventory Grid

Index [1] FR Consume

Index [2] FR Info

- **InventoryItemViewModel**

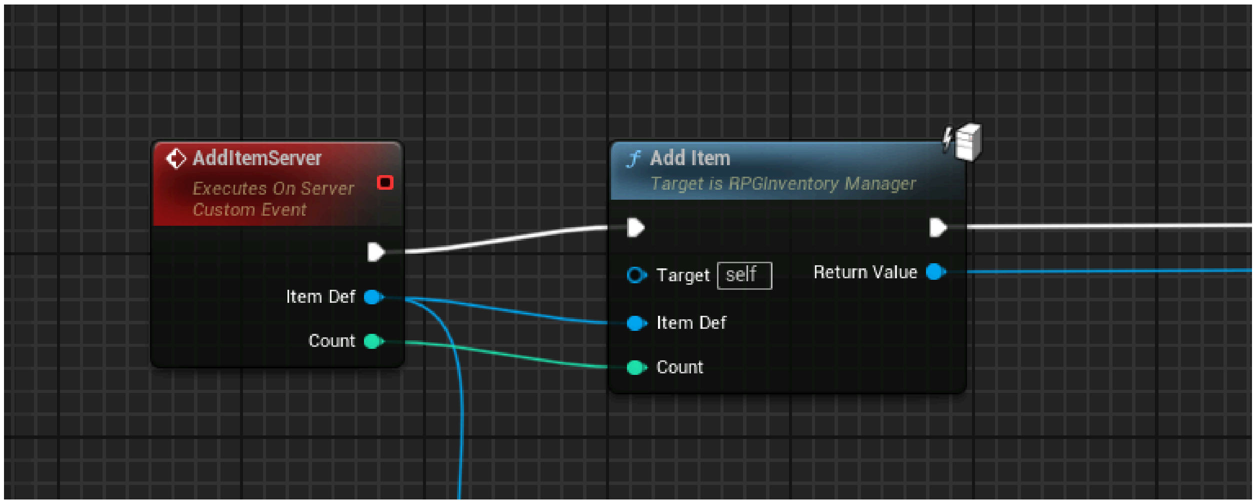
InventoryItemViewmodel is a derived class of MVVMViewModel. It is an instance of an item. Its member variables include Item, Count (quantity), and Index (the position of the item in the grid in the Inventory). It can be bound to UMG through the MVVMViewModel feature. CheckAddCondition andConsume will traverseItemThe Feature on executes the corresponding method.

InventoryViewModelIt is also a derived class of MVVMViewModel and is a collection of InventoryItemViewmodel.

Function description:

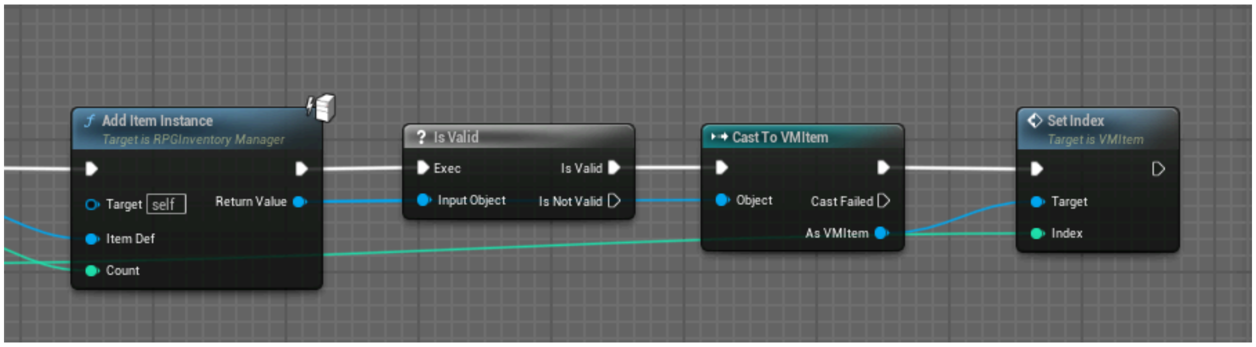
AddItem(URRPGInventoryItem* Item, int32 Count = 1): Add or subtract items. When Count>0, it is added. If the item does not exist in the Inventory itself or the current item ItemInstance cannot be added (exceeds the maximum number), a new item ItemInstance will be created.

Count<0 means consumption. If the quantity after consumption is <=0, the ItemInstance of the item will be removed.



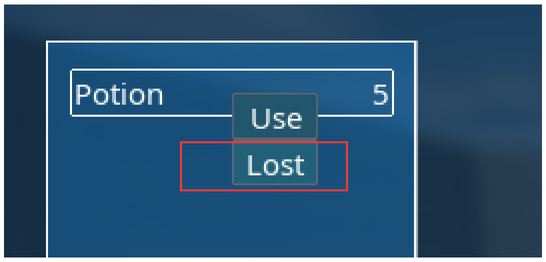
AddItemInstance

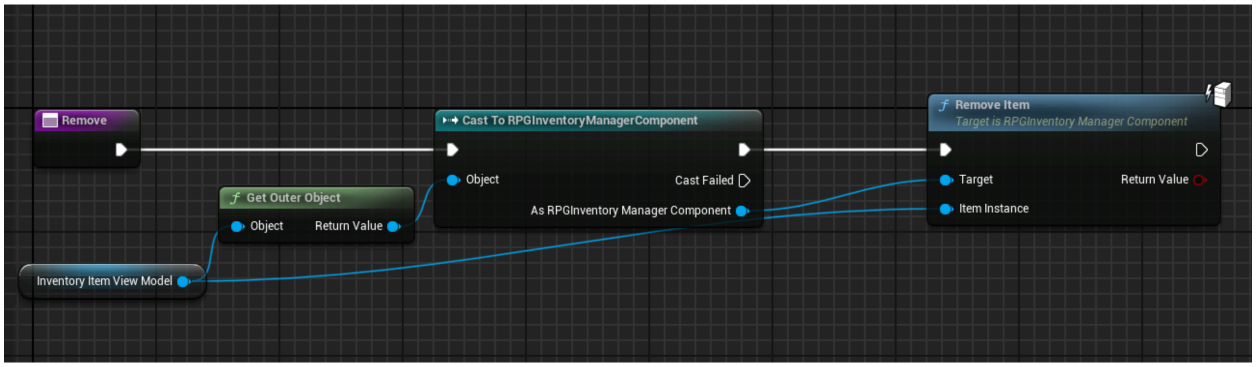
Add an ItemInstance instance



RemoveItemInstance(UInventoryItemViewModel* ItemInstance)

Remove an ItemInstance instance




Remove
Get Outer Object
Object
Return Value
Inventory Item View Model
Cast To RPGInventoryManagerComponent
Object
Cast Failed
As RPGInventory Manager Component
Remove Item
Target is RPGInventory Manager Component
Target
Item Instance
Return Value

CanAddItem

Calling this method will traverse all ItemInstances in the Inventory, then traverse the Feature array of RPGInventoryItem in the ItemInstance, and call the CheckAddCondition in the Feature (implemented in the blueprint). If the CheckAddCondition of a Feature returns false, the addition will fail.

If the traversal is unsuccessful and Count>0 (for addition, not consumption), CheckAddNewCondition in Feature will be called to check whether a new ItemInstance can be added.

GetAllItems()

Get all items in the Inventory

Consume(UInventoryItemViewModel* ItemInstance,FInventoryEventData Payload)：
Use items.When this method is called, thisin ItemInstancein RPGInventoryItemDefinition Feature array, calling Consume in all Features (implemented in blueprints).

Can be downloaded: https://github.com/GEF797/InventorySystem

In Content/Demo/Blueprint

BC_InventoryGrid

Extended multiple functions to implement network synchronization and grid Inventory, the data model uses VMInventory (a subclass of InventoryViewModel) and VMItem (a subclass of InventoryItemModel)

VMInventory

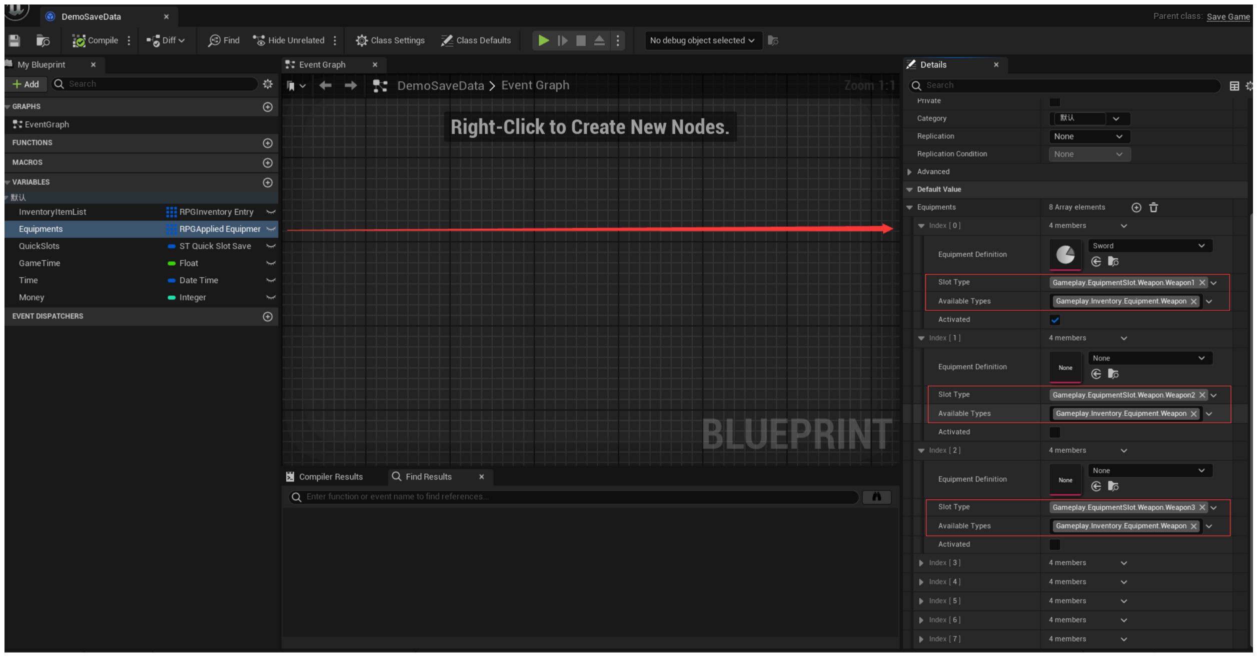
Added Total and GridFilled variables, added Fill functionTo realize plaid Inventory

VMItem

Added Index member variable to implement grid Inventory

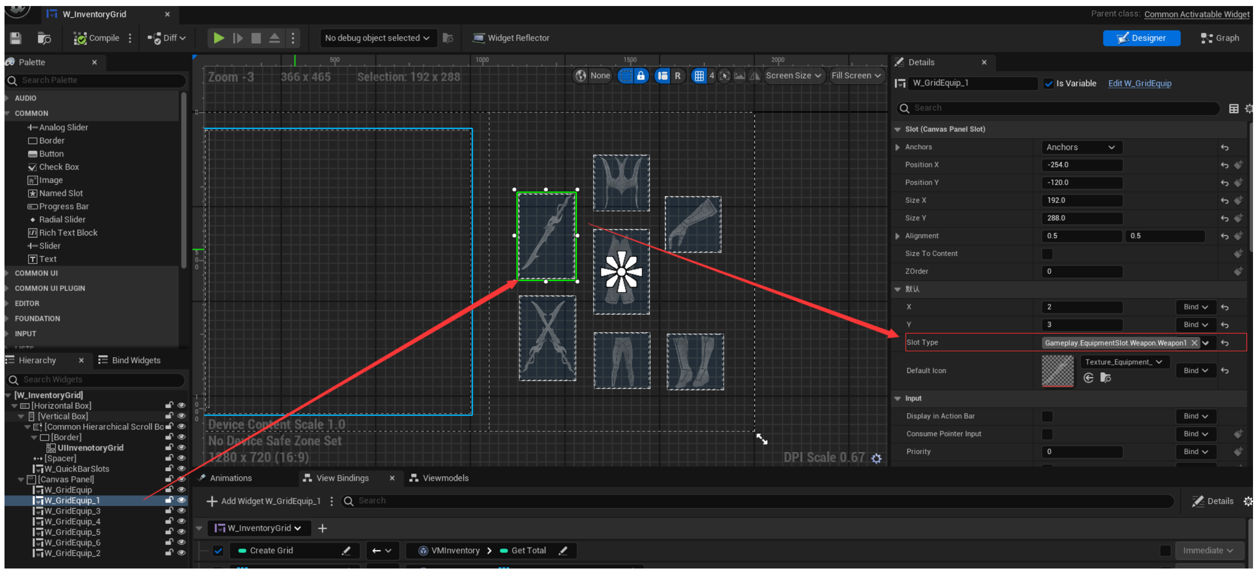
EquipmentManager

EquipmentManagerIt is a component that is mounted in PlayerState or Pawn (in my project, the monster's Controller does not need PlayerState, so EquipmentManager is directly added to Pawn). It can be mounted directly or added through GameFeature. EquipmentManager logic is different from InventoryManager. InventoryManager adds new instances when acquiring items.andEquipmentManager will call the AddSlot() method to add an Instance during initialization. It will only be modified and not added when it is equipped.



Variable	Type
InventoryItemList	RPGInventory Entry
Equipments	RPGApplied Equipmen
QuickSlots	ST Quick Slot Save
GameTime	Float
Time	Date Time
Money	Integer

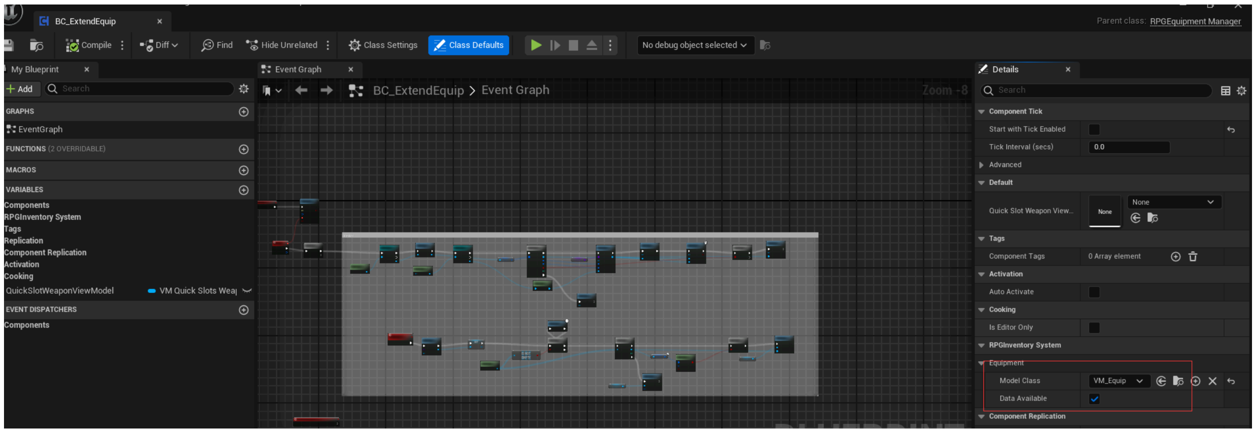
Private	
Category	默认
Replication	None
Replication Condition	None
Advanced	
Default Value	
Equipments	8 Array elements
Index [0]	4 members
Equipment Definition	Sword
Slot Type	Gameplay.EquipmentSlot.Weapon.Weapon1
Available Types	Gameplay.Inventory.Equipment.Weapon
Activated	✓
Index [1]	4 members
Equipment Definition	None
Slot Type	Gameplay.EquipmentSlot.Weapon.Weapon2
Available Types	Gameplay.Inventory.Equipment.Weapon
Activated	
Index [2]	4 members
Equipment Definition	None
Slot Type	Gameplay.EquipmentSlot.Weapon.Weapon3
Available Types	Gameplay.Inventory.Equipment.Weapon
Activated	
Index [3]	4 members
Index [4]	4 members
Index [5]	4 members
Index [6]	4 members
Index [7]	4 members



W_GridEquip_1 ✓ Is Variable Edit W_GridEquip

Slot (Canvas Panel Slot)	
Anchors	Anchors
Position X	-254.0
Position Y	-120.0
Size X	192.0
Size Y	288.0
Alignment	0.5 0.5
Size To Content	
ZOrder	0
默认	
X	2
Y	3
Slot Type	Gameplay.EquipmentSlot.Weapon.Weapon1
Default Icon	Texture_Equipment_
Input	
Display in Action Bar	
Consume Pointer Input	
Priority	0

Modify the ModelClass on EquipmentManager to use the ViewModel subclass derived from the blueprint to extend more functions.



Parent class: RPGEquipment Manager

QuickSlotWeaponViewModel: VM Quick Slots Weap

Component Tick	
Start with Tick Enabled	☐
Tick Interval (secs)	0.0
Default	
Quick Slot Weapon View...	None
Tags	
Component Tags	0 Array element
Activation	
Auto Activate	☐
Cooking	
Is Editor Only	☐
RPGInventory System	
Equipment	
Model Class	VM_Equip
Data Available	✓
Component Replication	

Function description:

EquipToSlot()

Equip the InventoryItemDef to this EquipmentInstance (Slot). When calling this method, all Features on the InventoryItemDef will be traversed and the CheckEquipCondition of the Feature will be called. As long as one CheckEquipCondition returns true, the equipment is successful. OnEquip() will be called when the equipment is successful. If the EquipmentInstance is ActivateSlot, OnActivate() will be called, and the post-equipment logic can be implemented in the blueprint.

UnEquip

Unequip this EquipmentInstance and call OnUnEquip() of all Features.

ActivateSlot()

Switch the currently used weapon

GetEquipmentList()

Get all equipment instances

- **EquipmentViewModel**

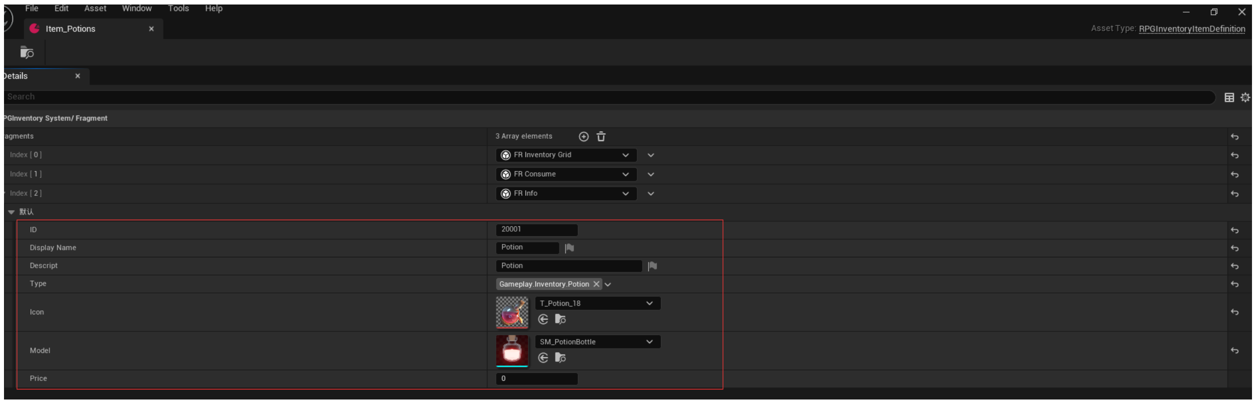
EquipmentViewmodel is a derived class of MVVMViewModel and is an instance of an equipment slot. The member variables include EquipmentDefinition, Activated (whether it is activated), SlotType (the label of the slot, which is a GameplayTag), and AvailableTypes (the types of items that can be equipped to this slot, is a FGameplayTagContainer) that can be bound to UMG through the MVVMViewModel attribute.

Feature

RPGInventoryItemFeature

Using the combination of Feature and RPGInventoryItem, this Inventory system can be adapted to most project needs.

Use Feature to add information to items:



PGInventory System/ Fragment

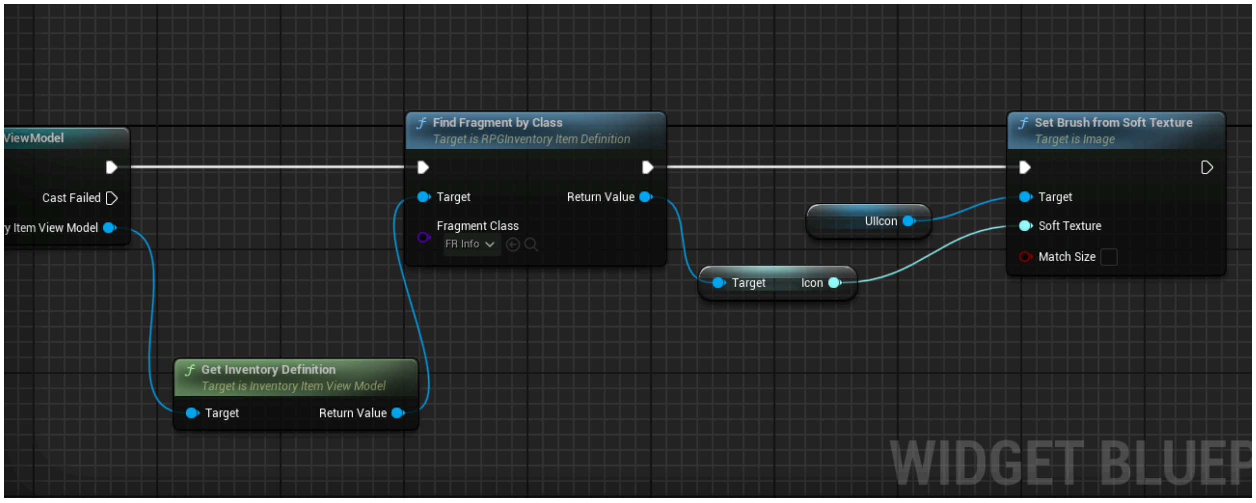
agments	3 Array elements
Index [0]	FR Inventory Grid
Index [1]	FR Consume
Index [2]	FR Info

默认

ID	20001
Display Name	Potion
Descript	Potion
Type	Gameplay.Inventory.Potion
Icon	T_Potion_18
Model	SM_PotionBottle
Price	0

You can create a Feature subclass according to your project needs, declare the required member variables, and add it to the item’s data assets.

FindFeature

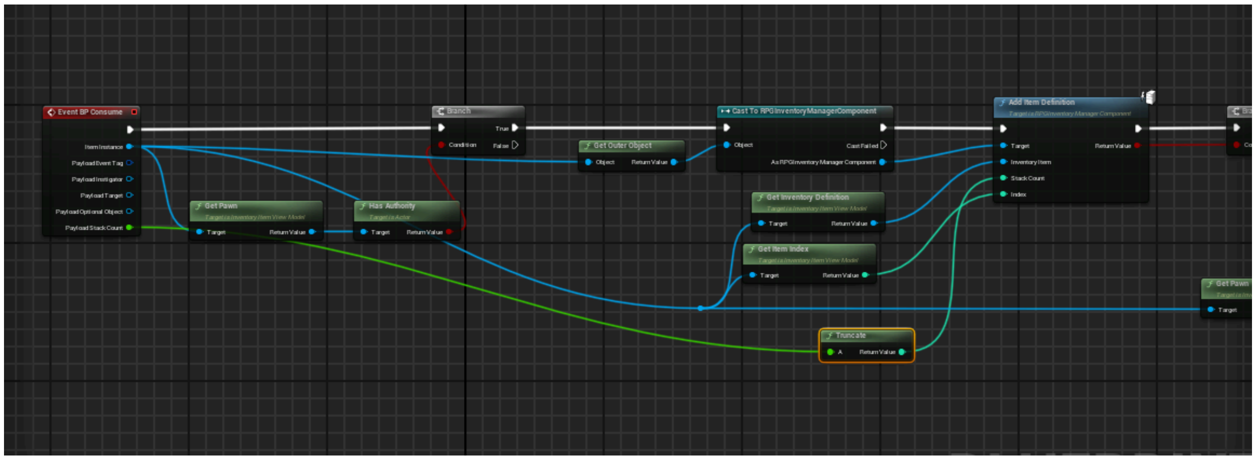


CheckAddCondition is triggered when ItemInstance calls AddItem to check whether ItemInstance meets the conditions for adding this quantity. The default is true.

CheckAddNewCondition will be triggered when Inventory calls AddItem to check whether the conditions for adding a new ItemInstance to the Inventory are met. The default is true.

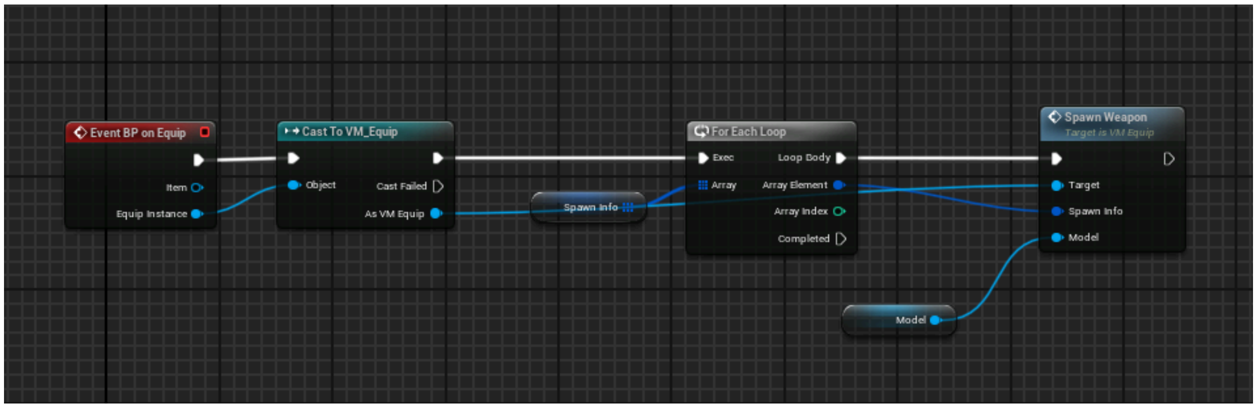
In Plugins, there is **FR-InventoryGrid**, which declares MaxCount and XY, and implements the item attributes of the grid Inventory in conjunction with the conditional restrictions of CheckAddCondition and CheckAddNewCondition

Consume When calling the Consume method of InventoryManager, it willimplementAll of this ItemFeature The Consume event can be extended in Blueprints.

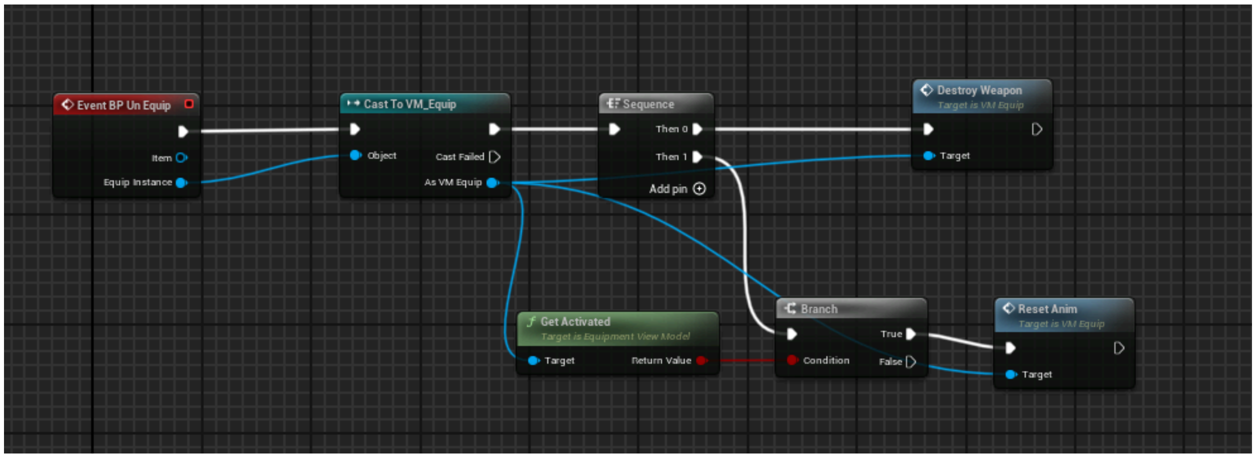


- **URPGEquipFeature** yes **RPGInventoryItemFeature** A subclass of , the following events are added:

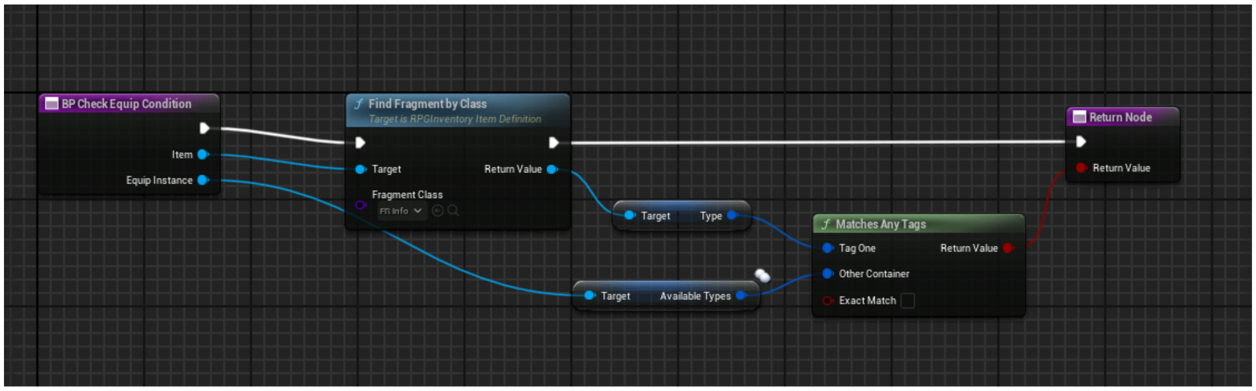
OnEquip



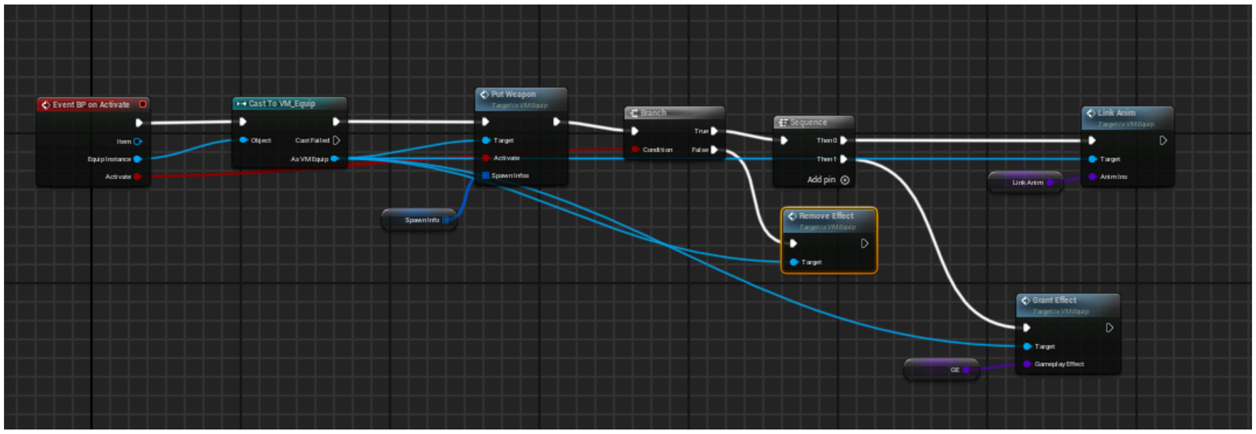
UnEquip



CheckEquipCondition returns false by default



OnActivate



The above is the usage in the case. You can also customize your own Feature according to your own project needs. Feature cannot store data, but you can save the data in the instance that calls Feature. Available for download:https://github.com/GEF797/InventorySystem

In Content/Demo/Blueprint,

VM_Equip

VMEquip has created multiple functions for Feature to call to add skills, increase values, generate equipment, change animations, etc. Since the skill system used by each project is different, you can create a subclass of EquipmentViewModel and extend the required functions in the blueprint. logic.

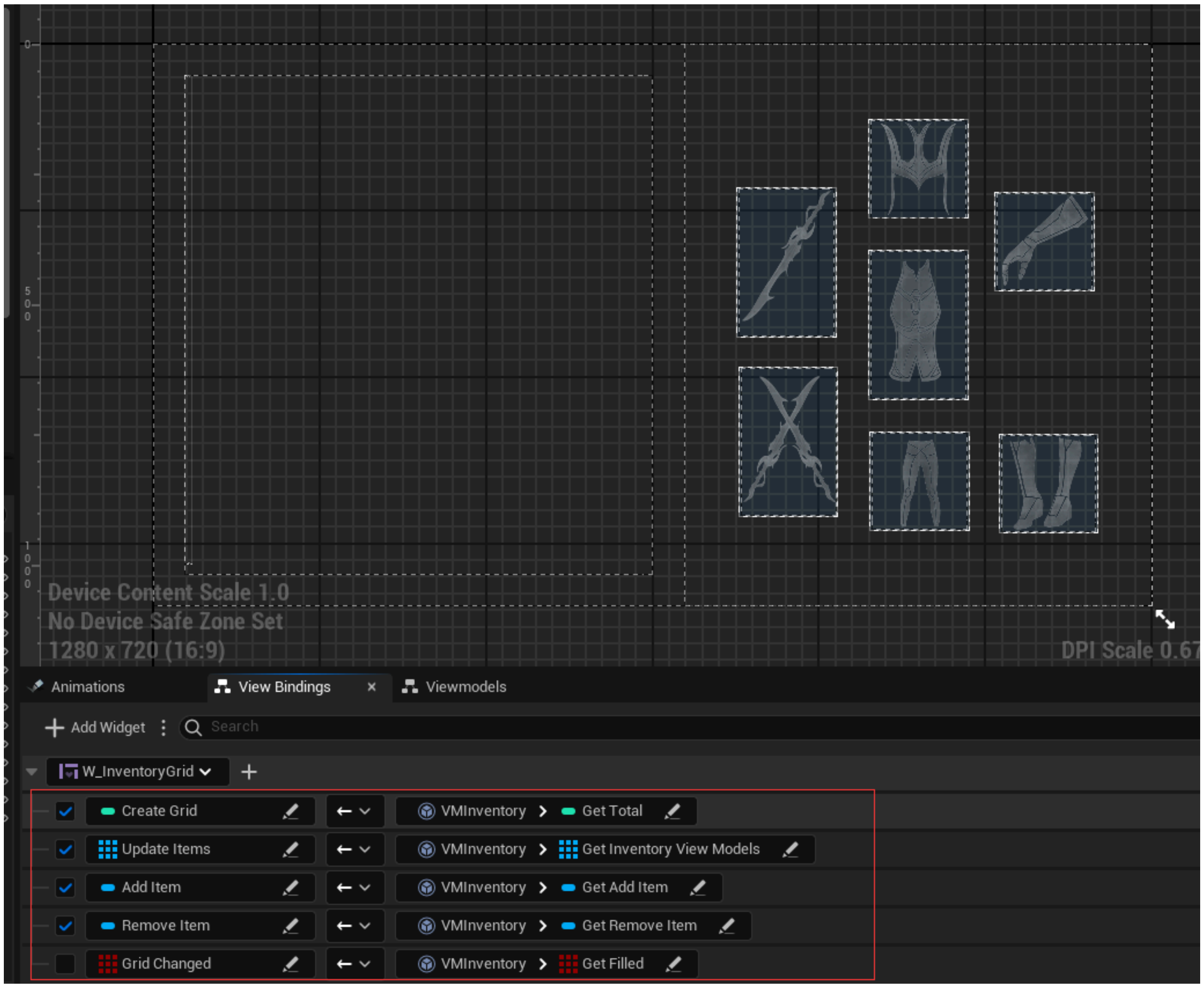
UI

Can be downloaded: https://github.com/GEF797/InventorySystem

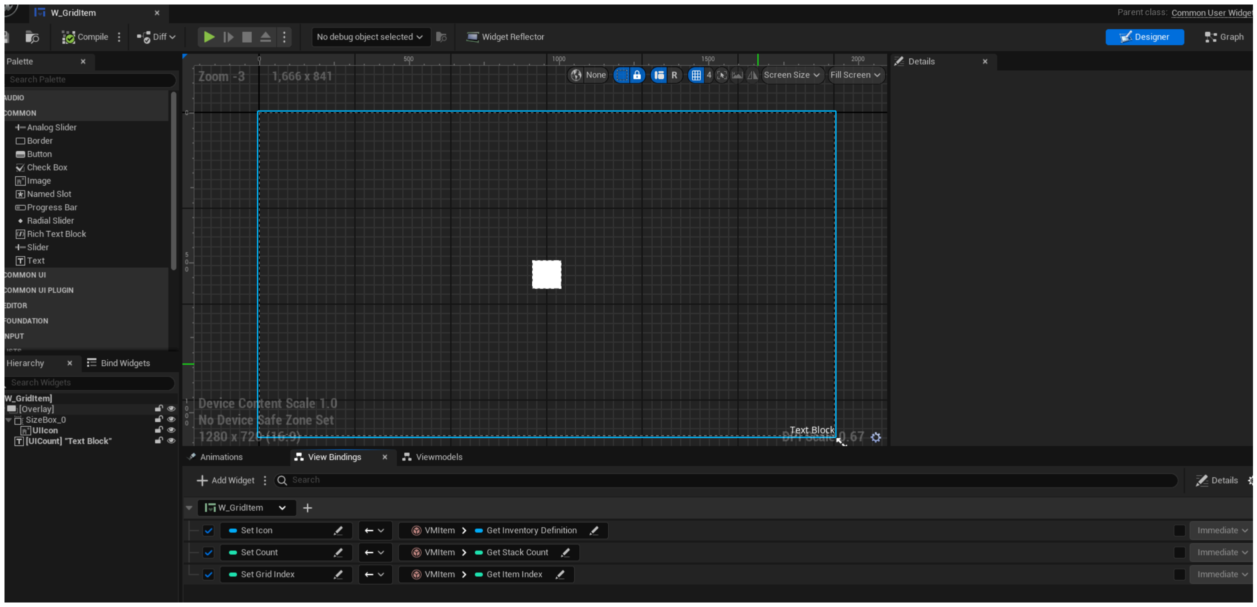
There are examples of RPGUI in Plugins

You can customize various UI styles according to your own needs. Just use the features of the MVVM View Model to bind data to the UI and display the required information.

Official documentation:
https://dev.epicgames.com/documentation/en-us/unreal-engine/umg-viewmodel



✓	Create Grid	←	VMInventory > Get Total
✓	Update Items	←	VMInventory > Get Inventory View Models
✓	Add Item	←	VMInventory > Get Add Item
✓	Remove Item	←	VMInventory > Get Remove Item
	Grid Changed	←	VMInventory > Get Filled


W_GridItem

Hierarchy

```
[W_GridItem]
  [Overlay]
    SizeBox_0
      UIIcon
    [UICount] "Text Block"
```

View Bindings

W_GridItem

✓	Set Icon	←	VMItem > Get Inventory Definition	Immediate
✓	Set Count	←	VMItem > Get Stack Count	Immediate
✓	Set Grid Index	←	VMItem > Get Item Index	Immediate