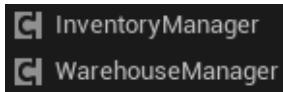


Inventory & Warehouse

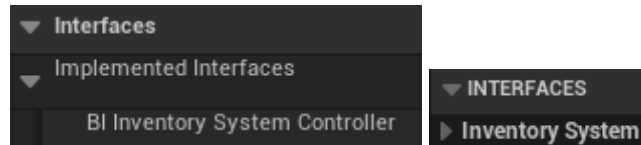
Slot Manager

AC_SlotManager is added to the **Player Controller**.

Slot Manager is used to implement both Inventory and Warehouse.

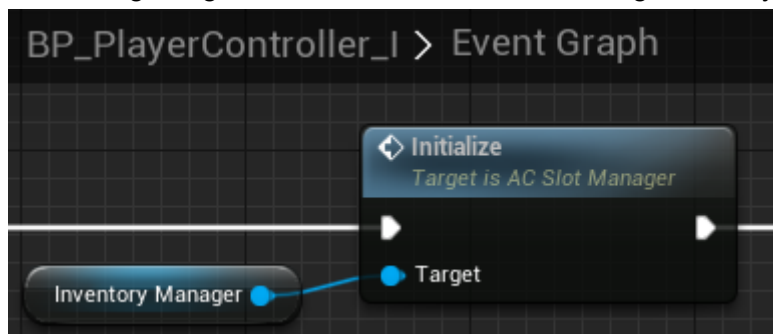


For the system to access managers without hard reference to controller, a special **BI_InventorySystem_Controller** is implemented in the controller too.

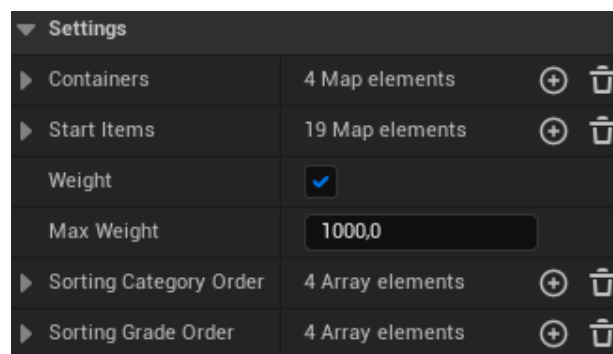


Warehouse Slot Manager is initialized automatically.

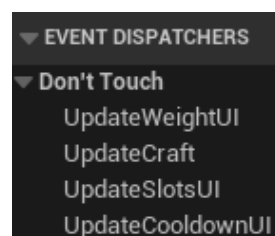
Inventory Slot Manager is **initialized** manually on **Begin Play**. This is necessary because there are Quick Slots, which, when loading the game, should be bound to existing inventory slots.



Select each manager in the controller and set settings. Hover cursor on each parameter to see description.



I use dispatchers to update GUI. GUI is bound to these events.



To add item, use **f TryAddItem**.

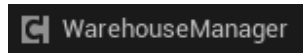
To remove item from a specific slot, use **f RemoveItemFromSlot**.

To remove item, use **f RemoveItem**. It is useful if the item can be in a few slots. I use it when I need to remove ingredients for Crafting or Incrustation.

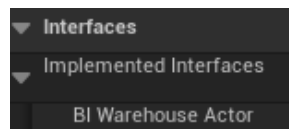
Warehouse

If SlotManager is added ONLY to Player Controller, **Global Warehouse** is used.

If you want all or some actors to have their own **Unique Warehouse**, you need to add Slot Manager to this actor.



And implement **BI_Warehouse_Actor** interface.



Don't forget set settings. Warehouse Slot Manager is initialized automatically.

At first, the system will try to get Slot Manager from the actor, and if it is not valid, it will try to get manager from Player Controller.

!!! Out of the box, Unique Warehouse is saved only for AI classes, since I use the spawn system and save all progress for it. And ofc, for AI save and load will work only if you use spawn system.

Weight

Enable the parameter in Slot Manager settings.

Weight	<input checked="" type="checkbox"/>
Max Weight	1000,0

Every time when an item is added or removed from inventory, `f CalculateWeight` is called.
I determine If Overweight effect should be applied/removed and call the next functions that are implemented in Player Controller.



In **InventorySystem** project I change Walk Speed using character movement component directly, but you can open the Player Controller and add any logic you want.

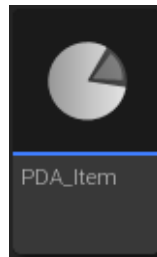
In **RPGTools** project I use Effect from Ability System.

!!! Since I use Player Controller to apply Overweight effect, it makes sense to use Weight only for Inventory.

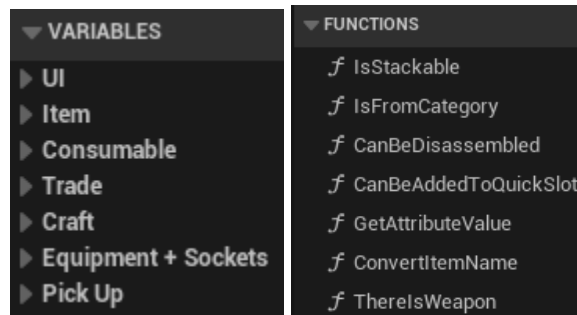
Item

Data

Data about the item is stored in a primary data asset **PDA_Item**. This is a template.

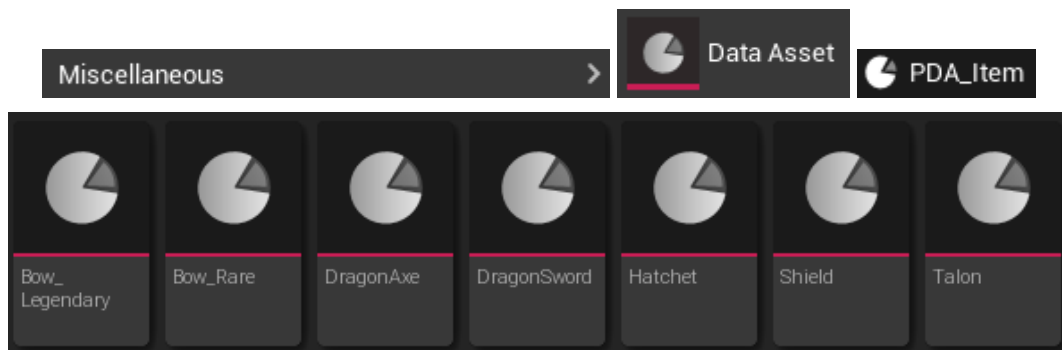


You can open it and add new parameters or support functions.



Real items are created from PDA_Item.

Click RMB on Content Browser -> Miscellaneous -> Data Asset -> PDA_Item.



In the previous iteration I used inheritance for data assets. It means that I created a child class from PDA_Item and added additional parameters (for example for equipment).

But in this iteration I decided not to use inheritance, because it complicates the project structure.

My demo is very simple and I don't need inheritance. But in your game you can use this feature.

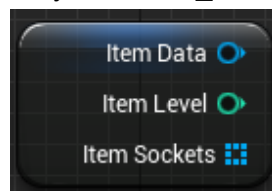
Why do I use Data Asset?

- (1) This is UObject. I can create supporting functions. It is easier to organize the code.
- (2) Inheritance. Some of the items can have much more parameters than others, in this case it makes sense to use inheritance. But this is a dangerous thing, since it complicates the architecture.
- (3) Reference instead a text. If all items are stored in a data table, you need to use the RowName to find Item from the data table. If you change the RowName during the production, it will be hell. Using a data asset, I set this data asset as a parameter. I can rename data assets, it will not be a problem, since I use object reference.

Item Structure

Items can have sockets, level, duration and other additional features.

So when we work with items, we work not only with PDA_Item, we work with S_ItemData structure.



Thus, before adding an item to any place, it needs to be created as a structure.

For this you need to use two functions from the library.

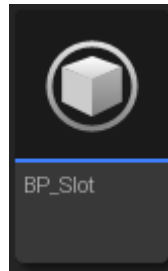


!!! Don't forget, if you want to move the item from one place to another, you need to move the structure.

Slot

Here we talk about Inventory and Warehouse slots.

Item is stored in **BP_Slot**.

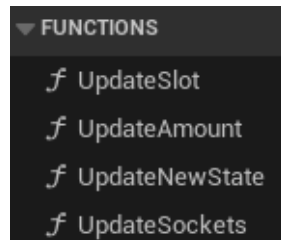


For example, when a player interacts with a Blacksmith, it is necessary to display a list of all equipment items from inventory. In this case we need to display a **list of slots**.

Remember that our item is S_Item structure. And it is stored in a slot. If we change S_Item directly, we will change its copy. The original S_Item (real item) always is in BP_Slot.

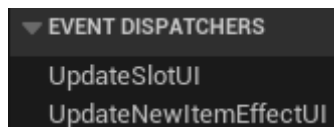
So that any operation with the item (level up, repairing, incrustation) should be done through the BP_Slot.

As for Blacksmith, when I use an incrustation feature I call a function **UpdateSockets** in BP_Slot.



As for the manager, the component creates and removes slots automatically, you don't need to think about it.

As for visual, I use dispatchers to update GUI. GUI is bound to these events.



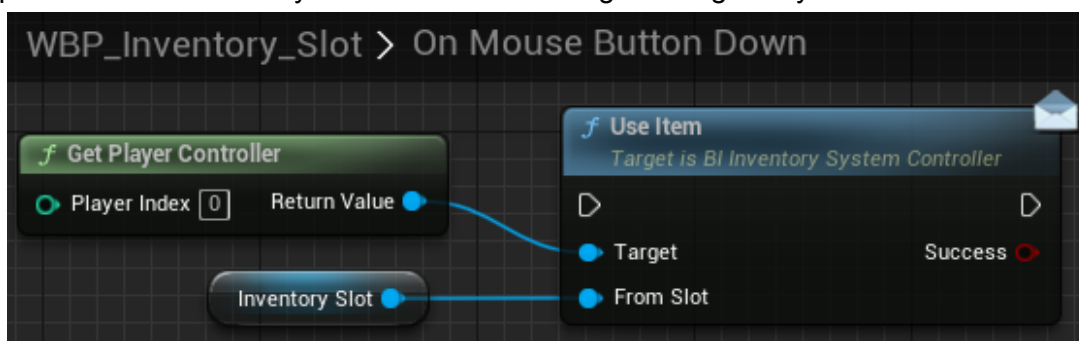
See my **WBP_Inventory_Slot** as an example.

Use Item

Currently, in build, the feature is implemented for Equipment and Consumable Items.

UseItem interface function is called when a player tries to use an item.

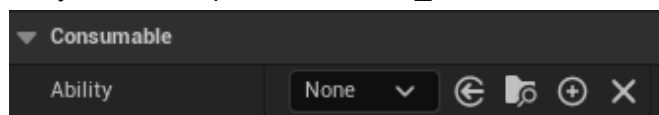
You can open this function in Player Controller and change the logic as you want.



Controller provides the access to other systems such as Ability System or Equipment System. We also save CurrentlyUsedItem so that we can access it from anywhere (for example, ability class) via interface.

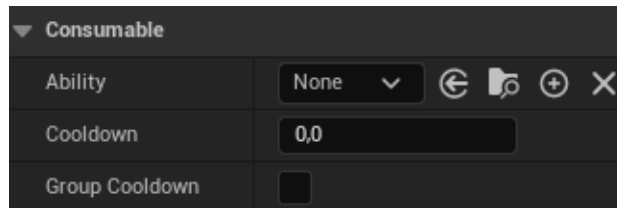
For Equipment, I don't use ability, and just call TryEquipItem

For Consumables, I use Ability, which is specified in PDA_Item.

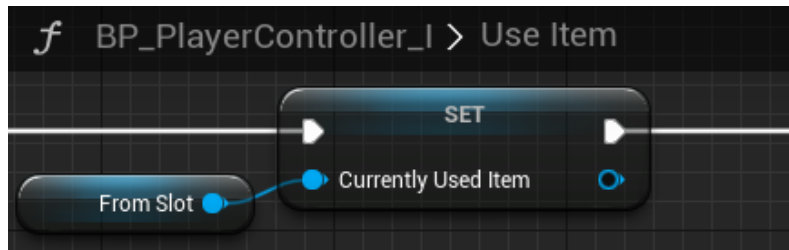


Consumable + Cooldown

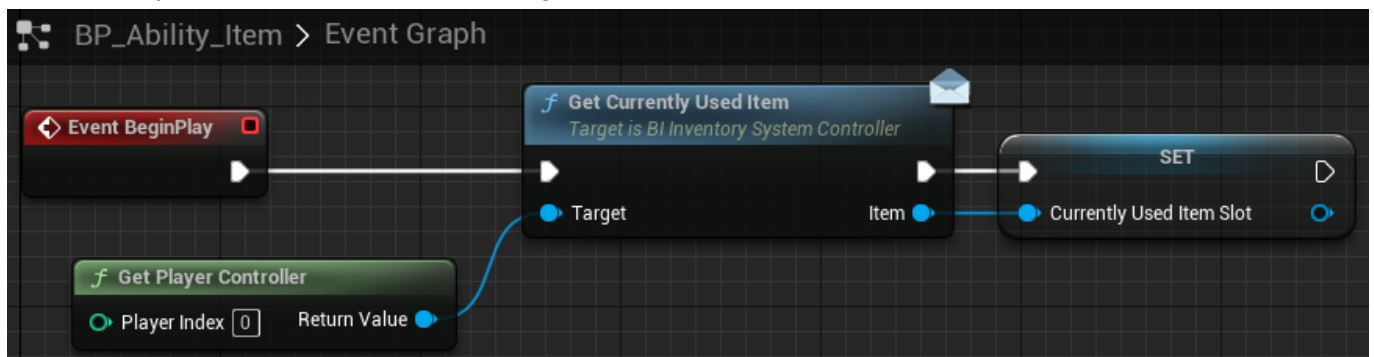
There are a few parameters for Consumable item in PDA_Item.



In Player Controller, I save currently used item (slot) in a special variable.



In BP_Ability_Item (parent class) we can get this item (slot).



In the demo, for all items I use BP_Ability_Consumable.

Why do I use Cooldown Timer actor class?

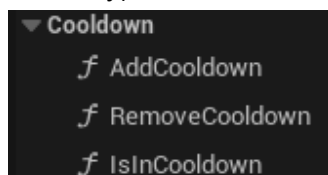
If we keep a timer inside the item (slot), when a player uses the last item, item is removed, the slot is cleared, and the timer is cleared too. When new item will be added to inventory, player can use it, despite the cooldown should be still applied to the item.

That's why I think we need to use this class. Even if a player hasn't any item in inventory, cooldown is still counted and applied to the item or item's group.

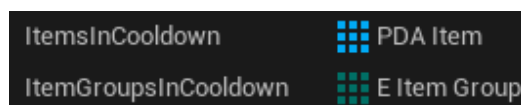
When the game is saved, we save all timers. When the game is loaded, we need to load Timers and give "Remaining Time" to them.

How does the timer work?

Timer is spawned in BP_SlotManager (Inventory).



Items or groups are added to special variables in SlotManager. These variables are used to determine if other items can be used.



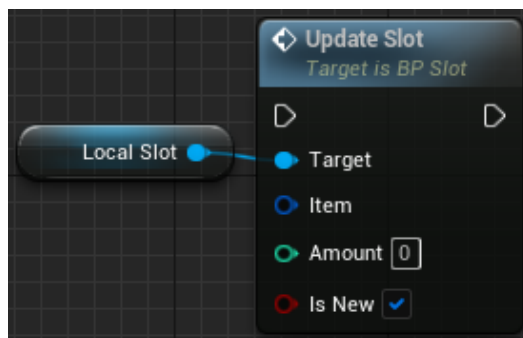
Timer plays the timeline and on each tick calls **UpdateCooldownUI** dispatcher event from SlotManager. See **WBP_Inventory_Slot** and **WBP_QuickSlot_Inventory**. They are bound to this event.

If the item or group are the same, cooldown effect is shown in the widget slot.

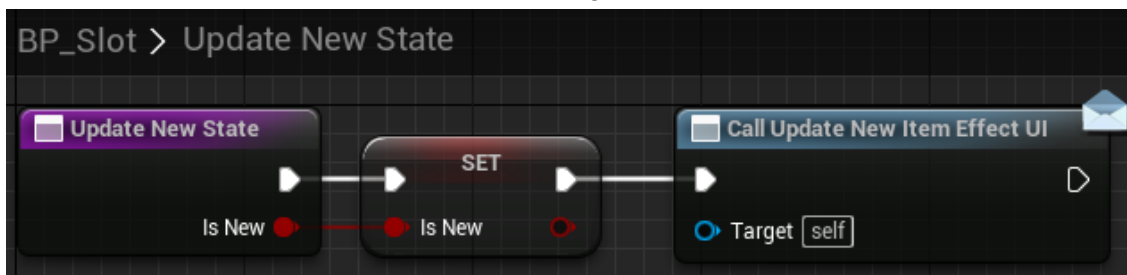
When the timeline is finished, item or group is removed from SlotManager.

New Item

When the manager adds item to slot, it calls the next function from slot.
This function is also used when item amount is increased.



BP_Slot saves current state and sends update message to GUI.



WBP_NewItem is bound to this event.

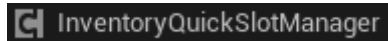
When player hovers cursor on inventory slot widget, the new state is changed to 'false'.

See **Event On Mouse Enter** in WBP_Inventory_Slot.

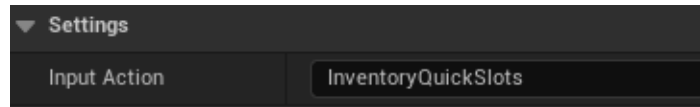
If you don't want to display this visual effect, just remove WBP_NewItem from WBP_Inventory_Slot.

Quick Slots

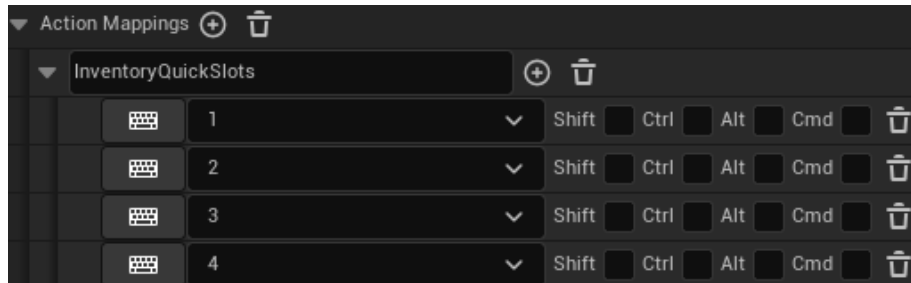
AC_QuickSlotManager_Inventory is added to the **Player Controller**.



You can select the manager in Player Controller and set Action Mapping name in settings.

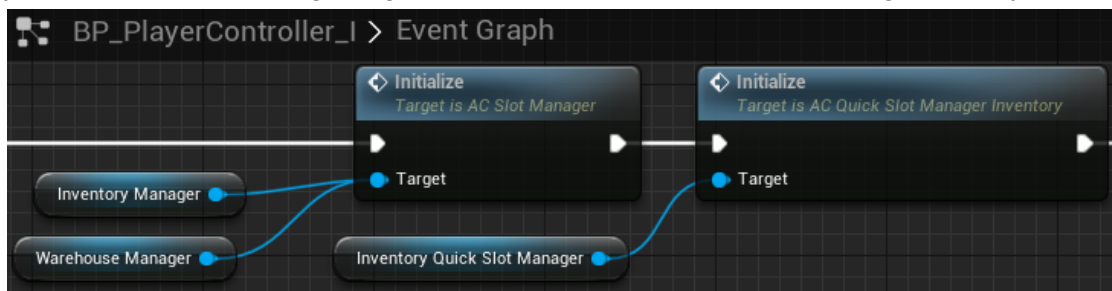


Project Settings -> Inputs.



On **Begin Play** the manager creates empty **BP_QuickSlot_Inventory** for each key from a specific Action Mapping.

Also manager should be **initialized** manually in Player Controller on **Begin Play**, after Inventory. This is necessary because, when loading the game, slots should be bound to existing inventory slots.



BP_QuickSlot_Inventory keeps not item data, but inventory slot. Don't worry about swapping items in inventory, because when you swap items, in fact, you swap slots.

You can swap items between quick slots. It is handled in QuickSlotManager.

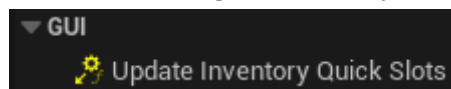
WBP_QuickSlot_Inventory is placed on HUD. In settings you can set Slot Index.

On Construct, the widget is bound to the quick slot by index.

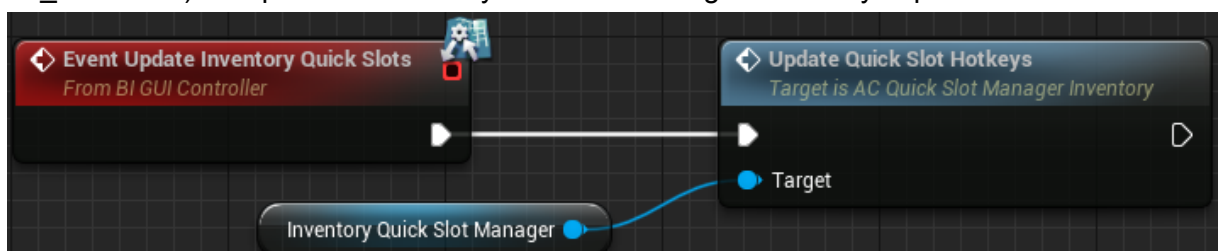


Not all items can be dropped on widget. It is handled in OnDrop function.

In **RPGTools** I have a **Menu** with **Input Settings** where player can change inputs.



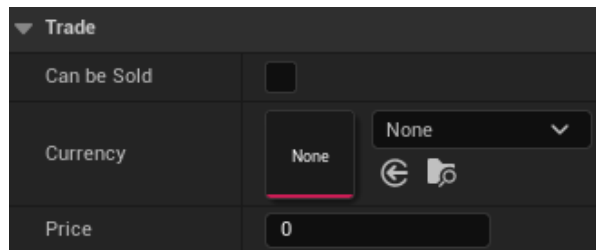
To Update inputs you can use **Update Inventory Quick Slots** interface function. This interface (BI_GUI_Controller) is implemented in Player Controller. Logic is already implemented.



Trade

Currency

There are a few parameters in PDA_Item.

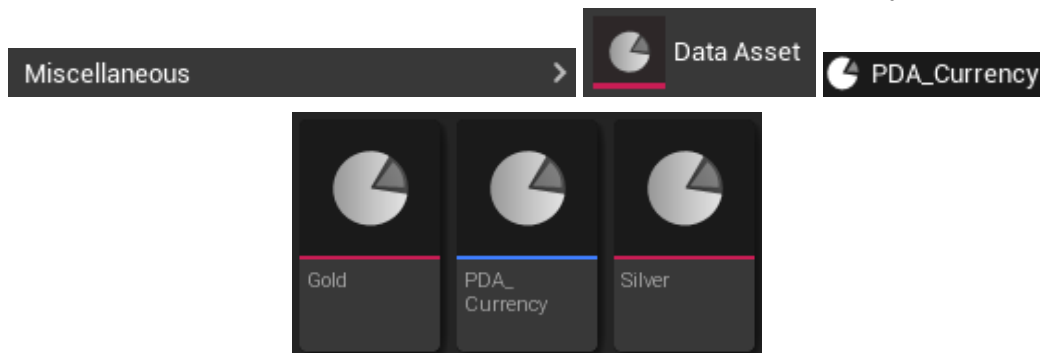


Data about the currency is stored in a primary data asset **PDA_Currency**. This is a template.

In section **Item -> Data** I explain why I use **data assets**.

Real currency is created from this template.

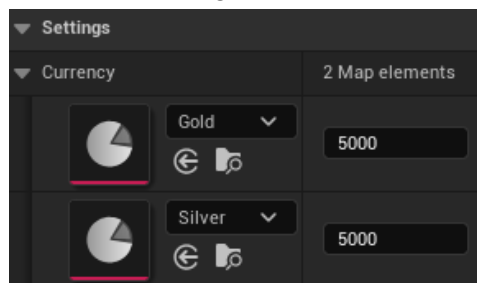
Click RMB on Content Browser -> Miscellaneous -> Data Asset -> PDA_Currency.



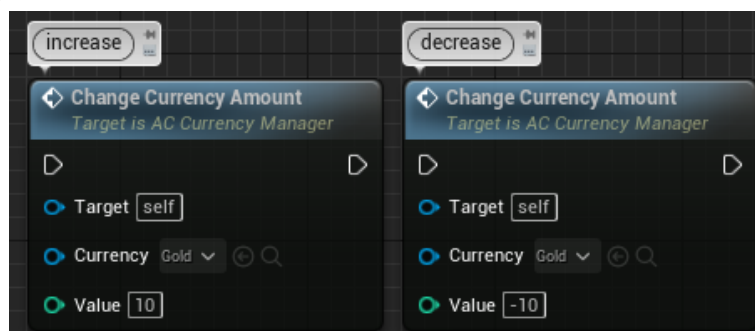
AC_CurrencyManager is added to the **Player Controller**.

BI_Trade_Controller interface is also implemented in Player Controller.

Select the manager in the controller and set settings.



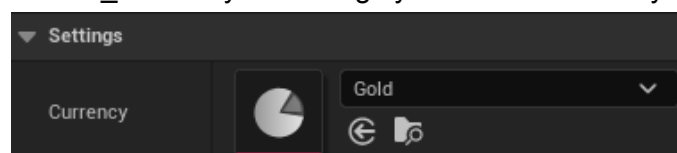
To change the amount, use the next function.



To get currency amount, use `f GetCurrencyAmount`.

I use dispatcher event to update GUI. **WBP_Currency** is bound to this event.

WBP_Currency is placed in WBP_Inventory. In settings you can set currency to which widget bound.



Assortment

There are two managers: **AC_AssortmentManager** and **AC_BuybackManager**.

Why do I use two managers?

Most often in games, merchant has two slot containers: assortment and buyback slots.

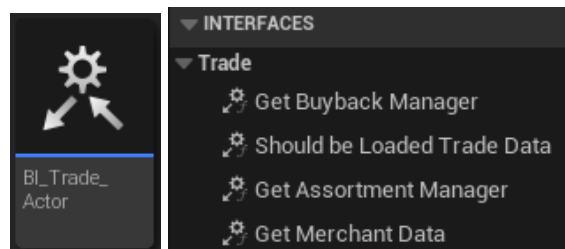
Assortment is different for each actor. It means Assortment Manager is added to Actor.

As for **Buyback Slots**, if they are the same for all actors, the manager is global and connected to Player Controller. If some actors have their own unique buyback slots, the manager is added to Actor.

In fact, game design can be different. I just found it necessary to divide the trading mechanics into two different components.

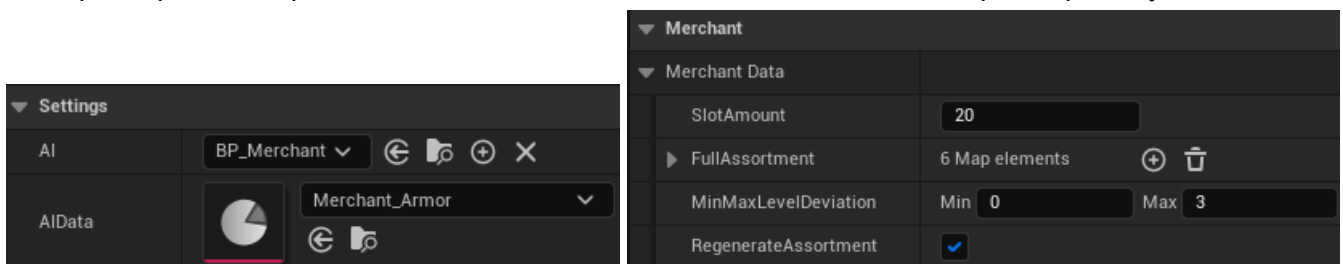
AC_AssortmentManager is added to **Actor**. There are no settings for the manager.

BI_Trade_Actor interface is also implemented in Actor. See **BP_Merchant**.



Manager automatically generates the assortment at start game. For this it needs to get merchant data from actor via interface. See BeginPlay event in Assortment Manager.

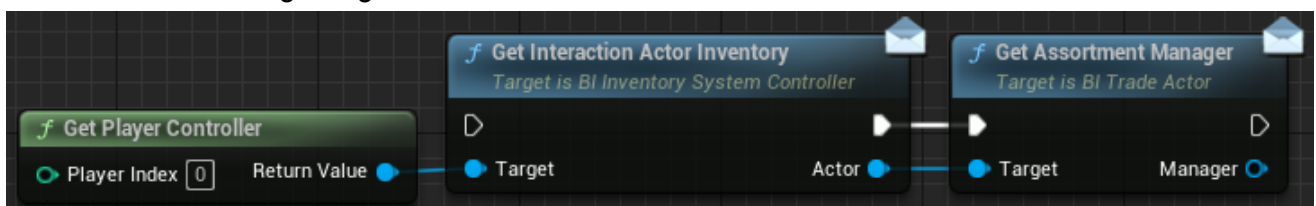
I use spawn points to spawn AI to the world. All data about AI is stored in a special primary data asset.



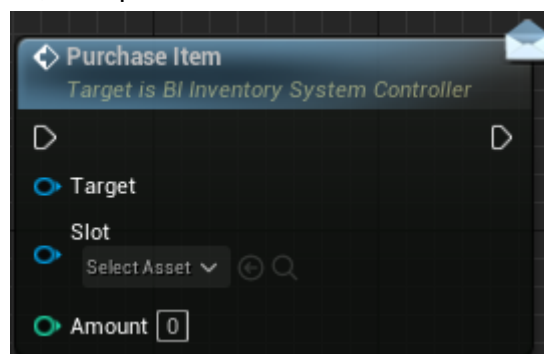
Assortment is saved for each AI in the world. See **BP_SpawnPoint**.

Assortment item level is updated when player levels up.

In GUI I use the next logic to get correct data.



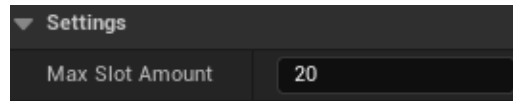
When a player purchases an item, the special interface function is called from the Player Controller.



Buyback

First, **Global** Buyback Slots.

AC_BuybackManager is added to the **Player Controller**. Select the manager in the controller and set settings.

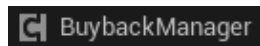


BI_Trade_Controller interface is also implemented in Player Controller.

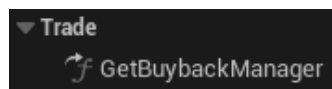
Slots are saved. Manager determines if slots should be loaded via **BI_InventorySystem_GI** interface.

!!! If BuybackManager is added ONLY to Player Controller, Global Buyback is used.

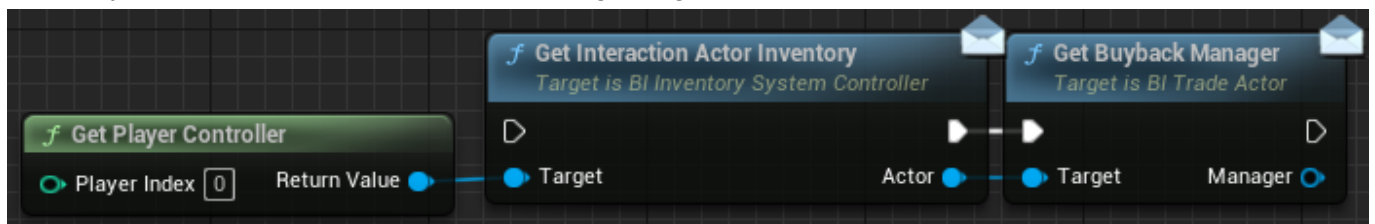
If you want all or some actors to have their own **Unique** Buyback Slots, you need to add Buyback Manager to this actor. See **BP_Merchant_UniqueBuyback**.



And override the interface function.



At first, the system will try to get Manager from the actor, and if it is not valid, it will try to get manager from player controller. In GUI I use the next logic to get correct data.

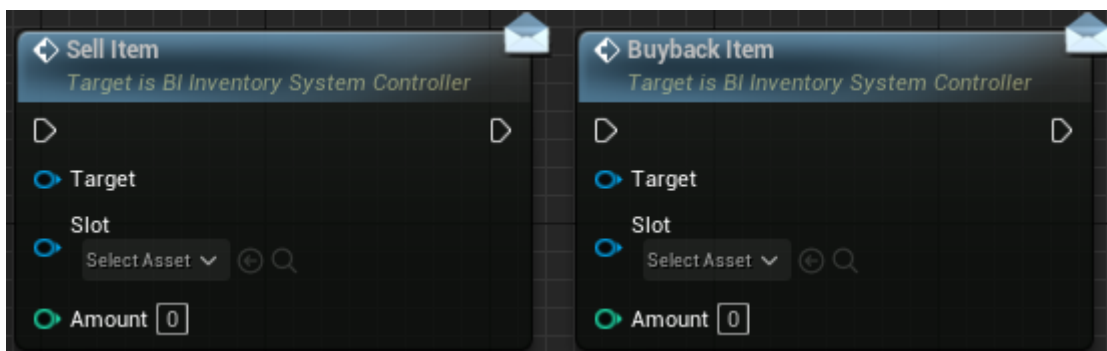


Buyback Slots are saved for each AI in the world. See **BP_SpawnPoint**.

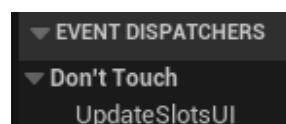
Manager determines if the assortment should be loaded via **BI_Trade_Actor** interface.

!!! Out of the box, Unique Buyback Slots are saved only for AI classes, since I use the spawn system and save all progress for it. And ofc, for AI save and load will work only if you use spawn system.

When a player sells or buybacks an item, the special interface functions are called from the Player Controller.



I use dispatcher from Buyback Manager to update trade slots for both Assortment and Buyback Managers. GUI is bound to this event. It doesn't matter which manager you will use to update slots.

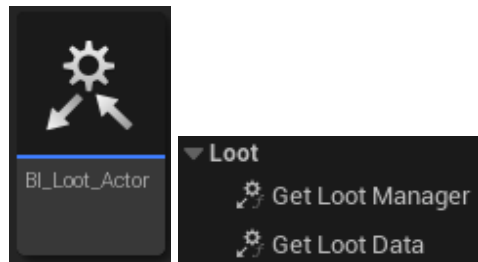


Loot

Loot Actor

AC_LootManager is added to **Actor**. There are no settings for the manager.

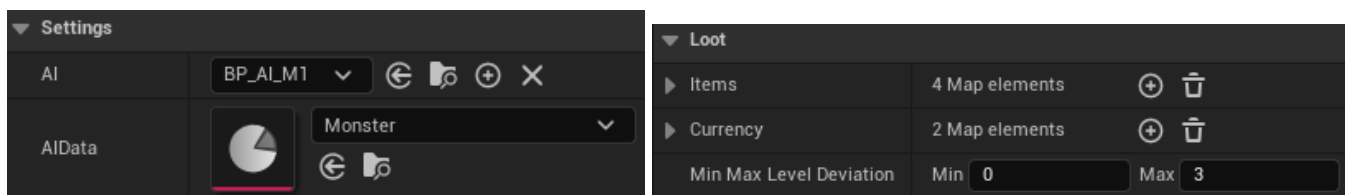
BI_Loot_Actor interface is also implemented in Actor. See **BP_Chest** or AI parent class.



Manager gets loot data via the interface.

For **Static** actors such as Chest, loot data is set in settings when actor is placed on the scene.

For **AI**, I use spawn points to spawn AI to the world. All data about AI is stored in a special primary data asset.



For **Static** actors, loot is saved. See BP_Chest. On BeginPlay I determine if the loot should be loaded.

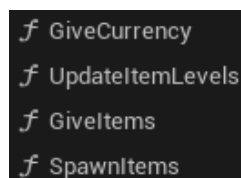
For **AI**, loot is not saved since killed AI is not spawned to the world after loading the game. But you can spawn items on the ground. PickUp actors are saved.

To generate loot, use **f GenerateLoot**.

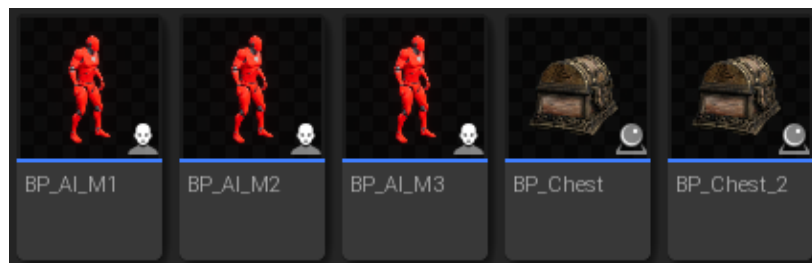
For **Static** actors such as Chest, manager automatically generates the loot at start game, to be able to interact with them.

For **AI**, manager generates the loot when AI is killed.

Use the next functions to implement different ways of giving the reward to player: interaction, interaction + window, spawn.



See my examples.



For those who own RPG Tools, see BP_AI and BP_Chest.

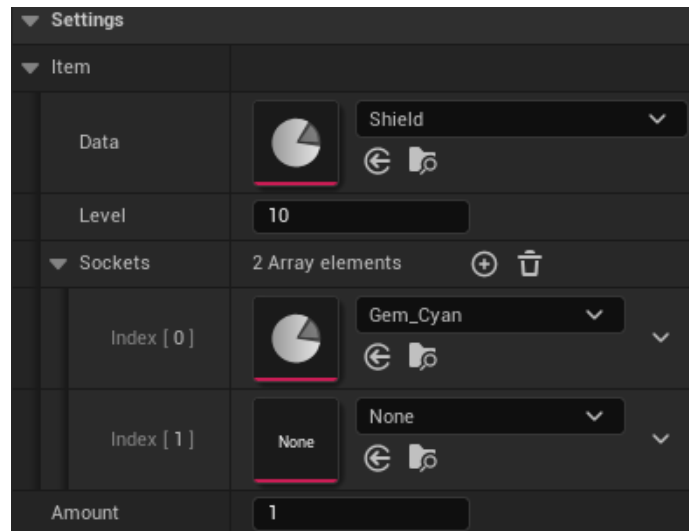
PickUp Item Actor

There are two actors.



BP_PickUp_Item is placed on the scene in the editor.

You can select it and set settings. Sockets array is automatically resized. See ConstructionScript.

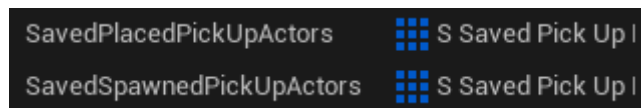


BP_PickUp_Item_Spawned is used when Loot Manager spawns items on the ground.

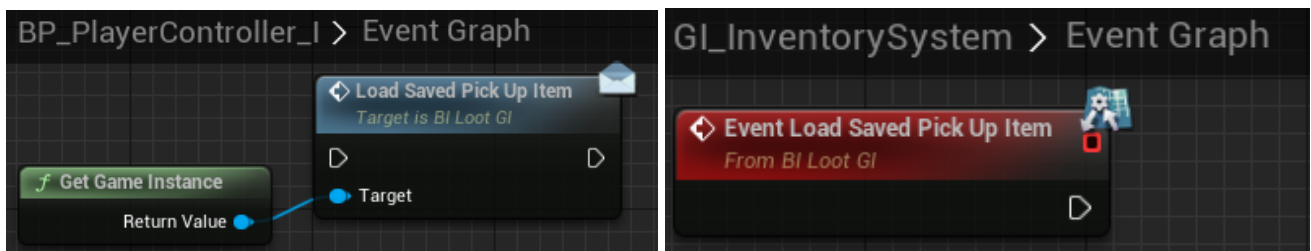
All PickUp Items are saved. For this I use Game Instance with implemented **BI_Loot_GI** interface.

But save and load are implemented a little differently for Item and Item_Spawned.

At first, these actors are saved to different arrays.



When the game is loaded, I spawn only Item_Spawned actors. The next event is called from Player Controller.



I don't need to spawn the actors that are placed in the editor since they are already on the scene every time when we use Open Level node. I just need to determine if they were saved.

See **Begin Play** in **BP_PickUp_Item**. I determine if actor was saved, and if it was not saved, I destroy it.

I don't need to determine this for **BP_PickUp_Item_Spawned** since I spawn only saved actors.

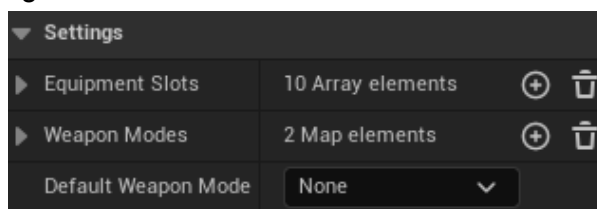
Also after game is loaded, item can be picked up. In this case we need to remove it from saved array in game instance, in order not to load it next time. See **Interaction** event in BP_PickUpItem.

Equipment

Core

AC_EquipmentManager is added to **Player Character** or **AI**.

You can select it and set settings.

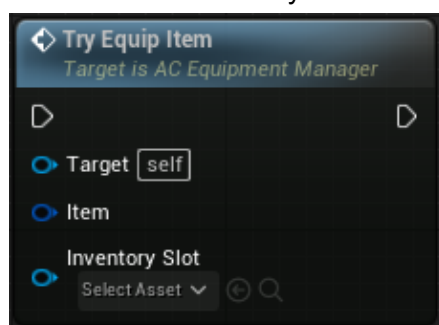


BI_Equipment_Actor is also implemented in Player Character or AI.

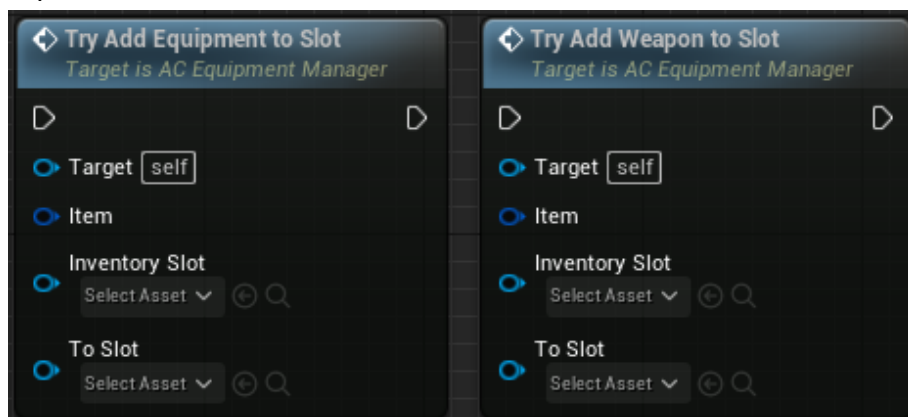
If the manager has to find the slot for the item, use the next node.

If manager is added to AI, you don't need to set InventorySlot, since AI hasn't inventory.

This function is used when player clicks RMB on inventory slot.



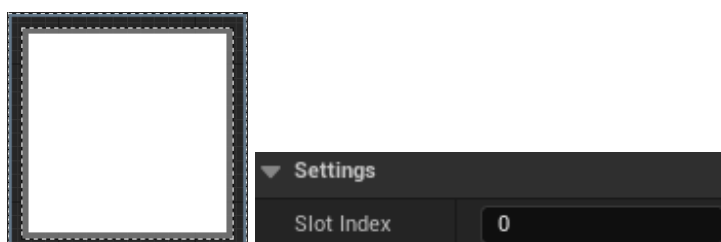
To equip item to a specific slot, use the next nodes.



I wrote separate logic for **weapon** and **armor + jewelry**. For weapon I spawn actor class WBP_Weapon and attach it to sockets. Also weapon slots can be activated/deactivated. For armor I use skeletal meshes.

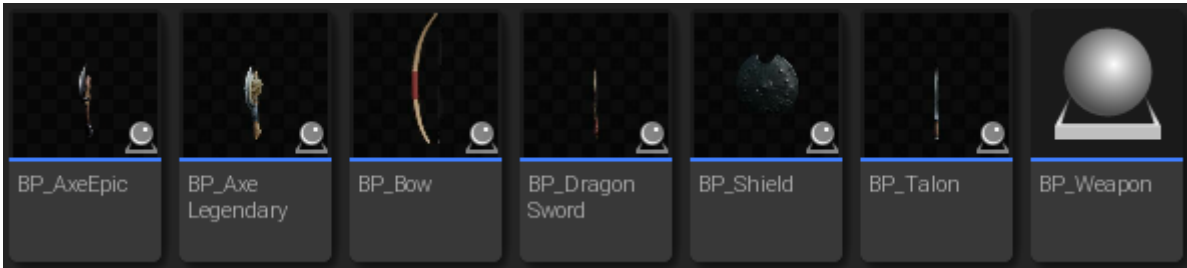
To switch weapon mode, use `f SwitchWeaponMode`.

As for GUI, see **WBP_Equipment** and **WBP_EquipmentSlot**. When you place a slot you need to set settings.

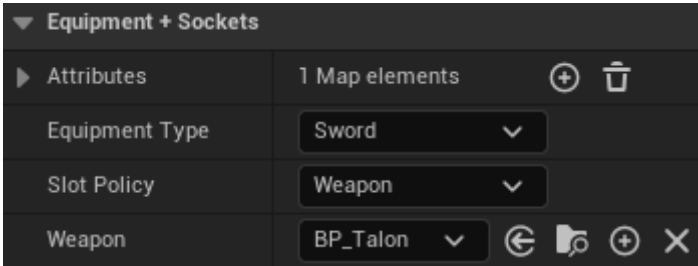


Weapon

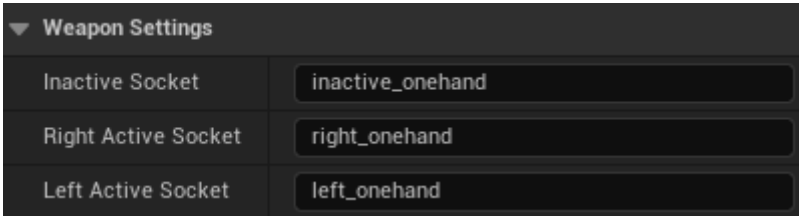
BP_Weapon is the parent class. You can create child classes for your custom weapons.
Actor is spawned and attached to sockets when weapon item is added to equipment slot.



This actor is set to **Weapon** parameter in **PDA_Item**.



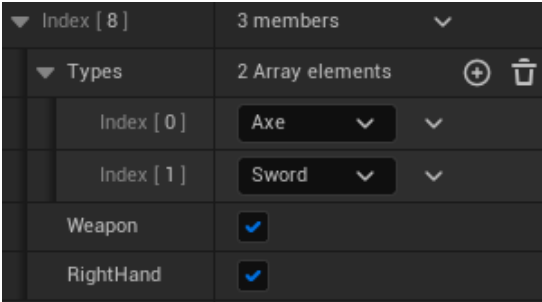
There are the settings in weapon class.



Sockets with these names should be added to the skeleton. See **SK_Mannequin_Skeleton**.

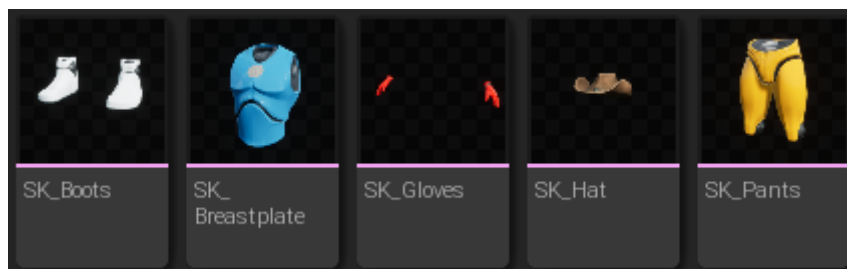
There are the settings in Equipment Manager.

If RightHand is enabled, the actor is attached to RightActiveSocket.

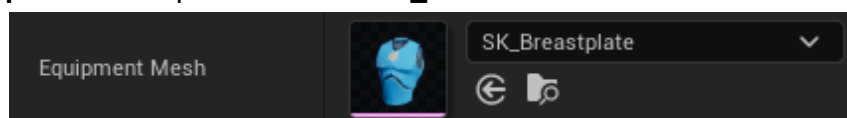


Equipment Mesh

For Armor I use skeletal meshes.



Mesh is set to **EquipmentMesh** parameter in **PDA_Item**.

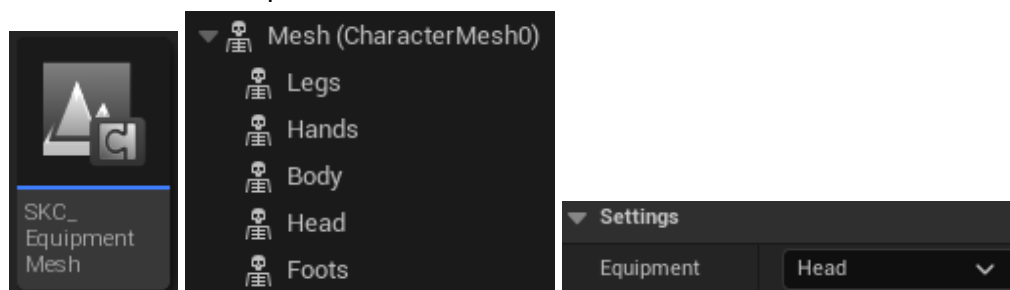


A special skeletal mesh component is added Player or AI Character class.

It must be attached to character mesh.

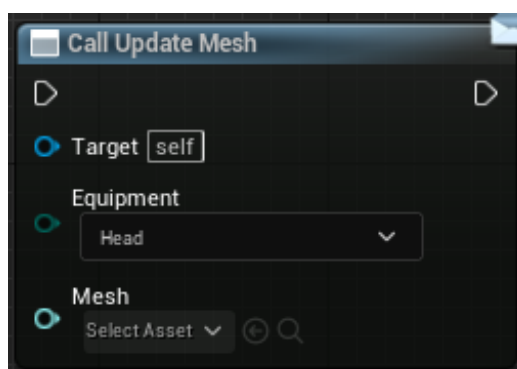
In settings you need to set equipment type.

Location and Rotation must be equal to 0.



When item is equipped, I call the next dispatcher event in Equipment Manager.

SKC_EquipmentMesh is bound to this event.

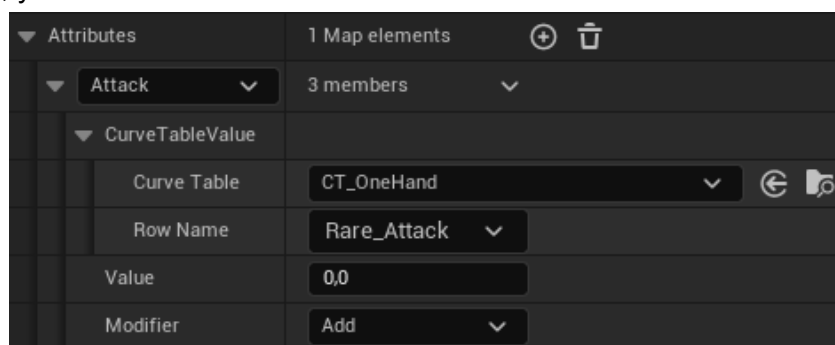


Attributes and Leveling

There is **Attributes** parameter in PDA_Item.

If item has no level, you can use **Value** and **Modifier**.

If the item has level, you need to use **CurveTableValue** and **Modifier**.



As for Curve Table, attribute values are taken from the table according to the current item level.

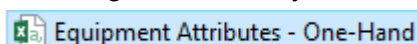
+ Curve	Filter	1	2	3	4	5	6	7	8	9	10
Common_Attack		20,0	40,0	60,0	80,0	100,0	120,0	140,0	160,0	180,0	200,0
Rare_Attack		25,0	50,0	75,0	100,0	125,0	150,0	175,0	200,0	225,0	250,0
Epic_Attack		30,0	60,0	90,0	120,0	150,0	180,0	210,0	240,0	270,0	300,0
Legendary_Attac		35,0	70,0	105,0	140,0	175,0	210,0	245,0	280,0	315,0	350,0

How to create Curve Table

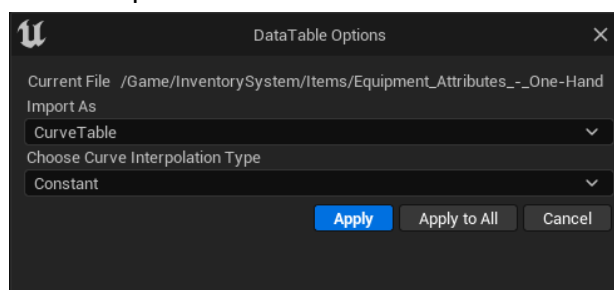
1. Curve Table is a CSV or JSON file
2. Let's create new table in Google Sheets

A	B	C	D	E	F	G	H	I	J	K
	1	2	3	4	5	6	7	8	9	10
Common_Attack	20	40	60	80	100	120	140	160	180	200
Rare_Attack	25	50	75	100	125	150	175	200	225	250
Epic_Attack	30	60	90	120	150	180	210	240	270	300
Legendary_Attack	35	70	105	140	175	210	245	280	315	350

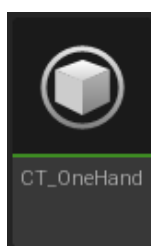
3. Export a sheet as CSV file
File -> Download -> CSV (current sheet)
You will get the file on your disk.



4. Drag and drop this file to any folder in content browser
The special window will appear.
5. Select Curve table and Constant options











6. Press Apply and rename asset as you want



Incrustation

There are the next parameters in PDA_Item.

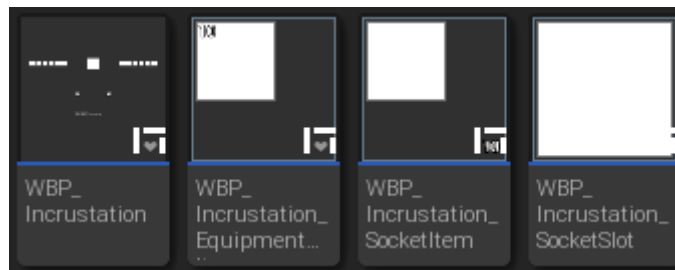
Socket Amount	<input type="text" value="2"/>
Insert Item	<div><div></div><div>Tools </div></div> <div> </div>
Insert Item Amount	<input type="text" value="3"/>
Extract Item	<div><div></div><div>Tools </div></div> <div> </div>
Extract Item Amount	<input type="text" value="1"/>

See **Item Structure** and **Slot** sections, to learn how item is created and exists at runtime.

How it works?

No component is needed to implement this mechanic.

You just need a GUI to generate and display all needed data. You can use my widgets or create your own, using my as the example.



The process is simple:

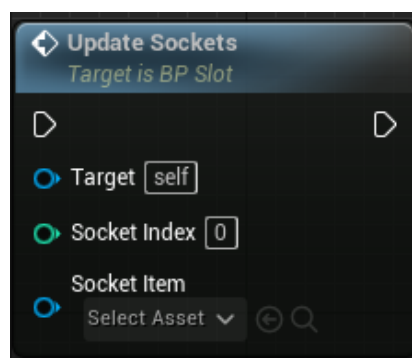
- insert: pick item -> pick socket slot -> pick socket item -> press button
- extract: pick item -> pick socket slot -> press button



I think for such a mechanic there's no need to create a manager, since we don't need to handle or keep any information. Widgets have references to inventory slots.

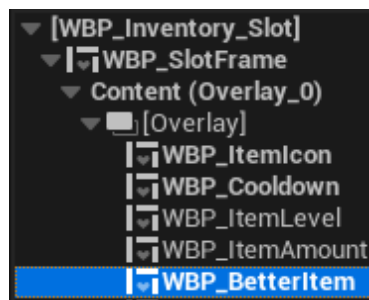
When buttons are pressed we just remove or add items from inventory, and update sockets.

Just keep in mind that when we update a socket we don't do it directly with the structure. We call a special function from inventory slot.

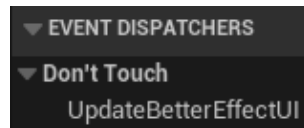


Better Item Effect

This is a visual feature that works for Inventory, Warehouse, Loot widget slots.
The special widget is added to the slot.



When item is added to equipment slot, Equipment Manager calls dispatcher event to update effect.



On Construct, **WBP_BetterItem** is bound to this event.
If target item is better than equipped item, the effect is shown.



Live Portrait

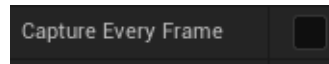
Scene Capture Component is added to Player Character.

I set Render Primitive Mode = Show Only List, in order to render only character mesh and attached weapons.

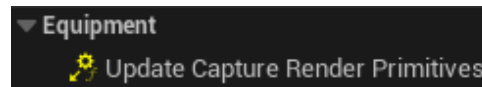


BI_Equipment_Actor interface is implemented in Player Character class.

Using interface we can enable/disable



When a new weapon class is created and attached to socket, I update render primitives calling the next event from Player Character.






Craft

Craft Data

I implemented one of the most popular craft systems: craft with known recipe.
Crafted Item level = Player level.
There are the next parameters in PDA_Item.

▼ Craft		
▼ Recipe	3 Array elements	⊕ 🗑
▶ Index [0]	3 members	▼
▶ Index [1]	3 members	▼
▶ Index [2]	3 members	▼
Craft Amount	<input type="text" value="1"/>	
Disassemble	0 Array elements	⊕ 🗑

There is **Alternative Items** feature.
One of the alternative items that can be used instead of the main item, in the same amount.

▼ Index [1]	3 members	▼
Item	 Gem_Blue	▼
Amount	<input type="text" value="4"/>	
▼ AlternativItems	2 Array elements	⊕ 🗑
Index [0]	 Gem_Cyan	▼
Index [1]	 Gem_Orange	▼

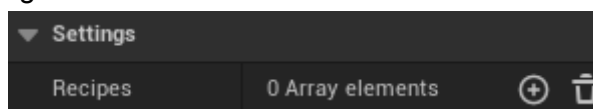
Widget is highlighted if there are alternative items.



Craft Manager

AC_CraftManager is added to **Player Controller**.

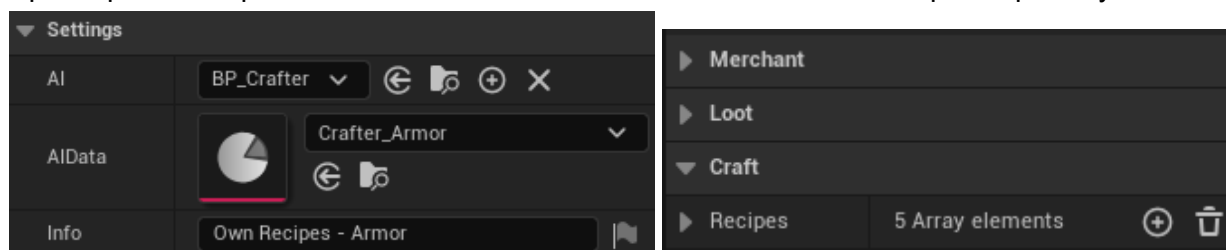
You can select it and set settings.



These recipes are global.

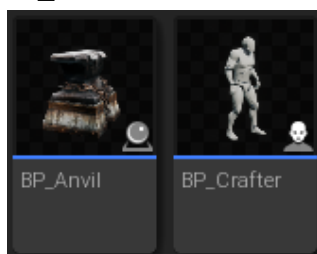
At the same time, each actor can have its own recipes.

I use spawn points to spawn AI to the world. All data about AI is stored in a special primary data asset.



You can use crafting for any actor you want: AI or static actor.

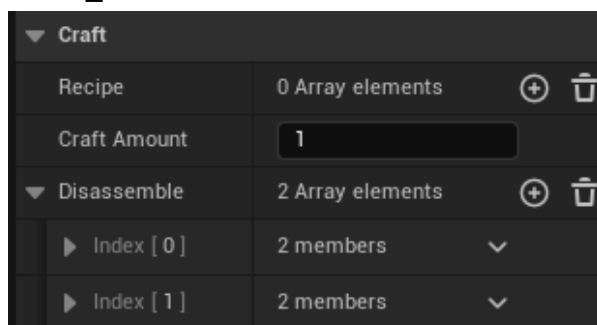
For this, it's enough to implement **BI_Craft_Actor** interface.



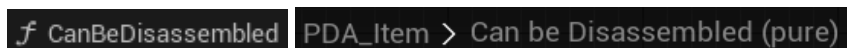
At first, the system will try to get Recipes from the actor, and if the array is empty, it will try to get Recipes from Player Controller.

Disassemble

There is the next parameter in PDA_Item.

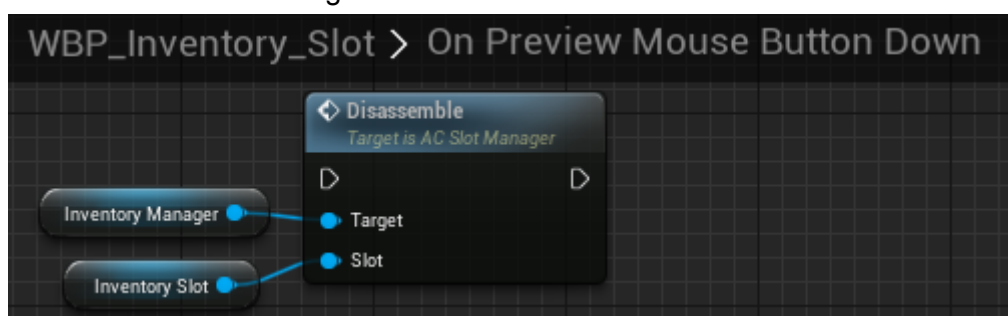


And the next function.



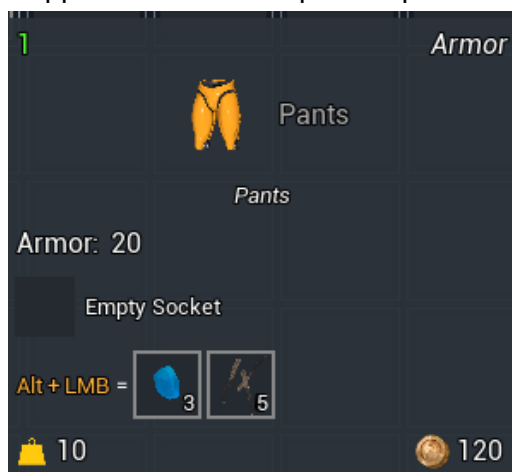
See **WBP_Inventory_Slot** or **WBP_Warehouse_Slot**.

Next function is called from Slot Manager.



Inside I check if item can be disassembled.

If item can be disassembled, data appears in item tooltip description.



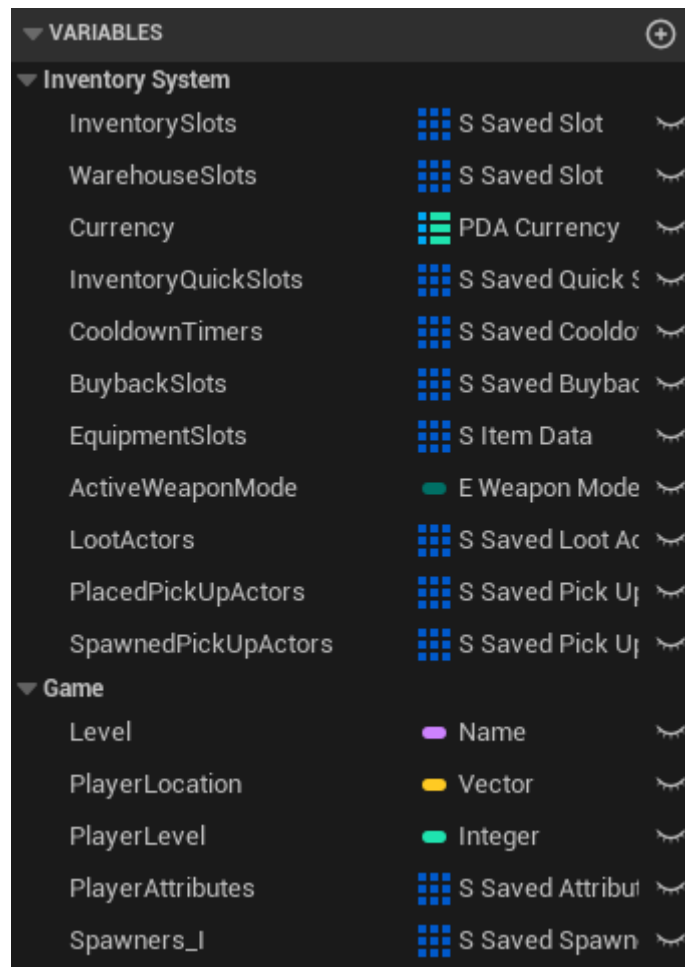
Save and Load

For saving and loading I use Game Instance and Save Game classes.

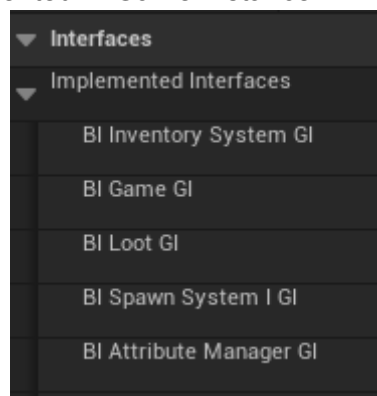
GI_InventorySystem is always used (teleportation between levels or saving).

SG_InventorySystem is used to save data to a file.

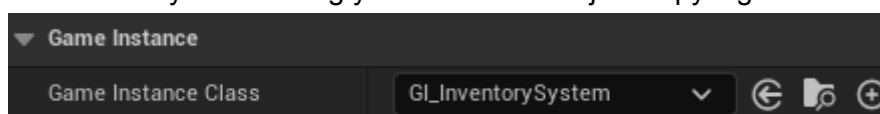
The following data is saved.



Each actor (Managers, Chest, PickupItem, Spawner) uses its own interface to contact the game instance class. Interfaces are implemented in Game Instance.



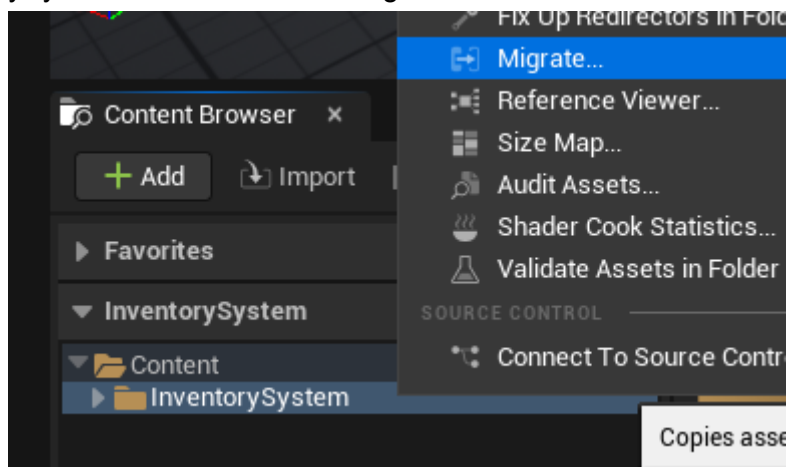
In order to use my Game Instance class in your game, open project settings -> Maps and Modes -> and select Game Instance class. If you are using your own class -> just copy logic.



How to Connect

Migrate

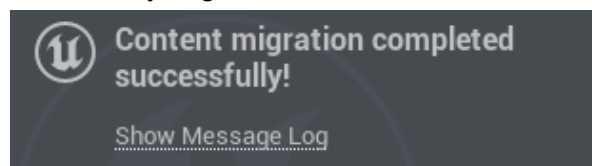
Find downloaded Inventory System project in your library, and create new project based on it. Click RMB on InventorySystem folder and select migrate.



Press OK and select Content folder in your project.

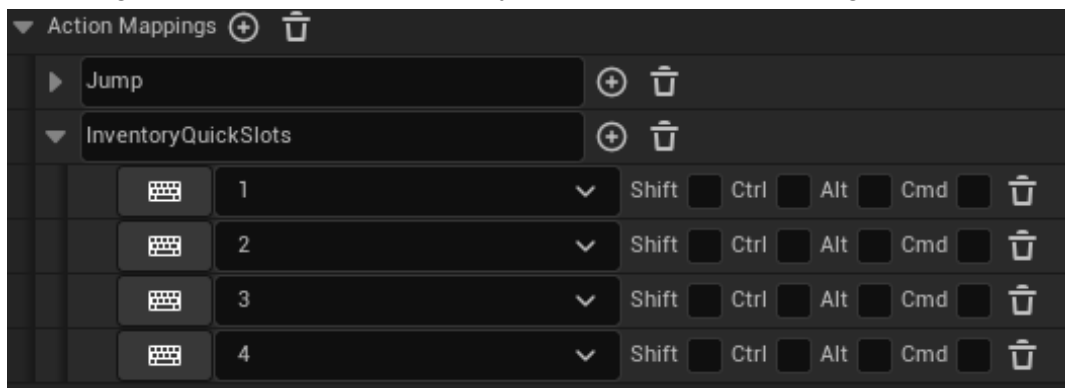
Media (E:) > YourProject > Content >

Press OK. Project should be successfully migrated with all files.



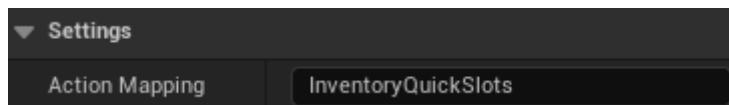
Inputs

Open Project Settings -> Input -> Create InventoryQuickSlots Action Mapping.



Open Player Controller -> Select InventoryQuickSlotManager.

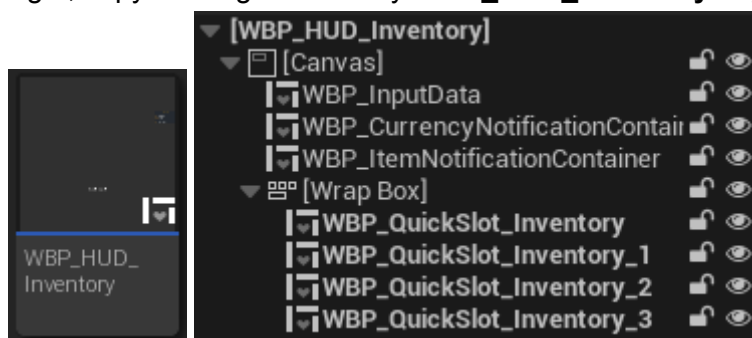
Make sure Action Mapping name in settings is the same.



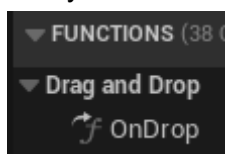
Slot widgets for each key are placed manually on HUD. See next section.

HUD

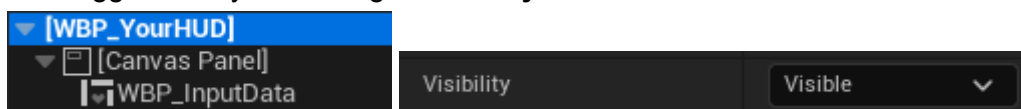
If you haven't HUD, just use my one. HUD widget is created in the Player Controller on BeginPlay.
If you have your own widget, copy all widgets from my **WBP_HUD_Inventory** to your HUD.



Override OnDrop function and copy logic from my HUD.

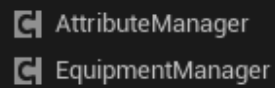


On Drop event is triggered only if the widget **Visibility = Visible**.



Player Character

Copy Managers.



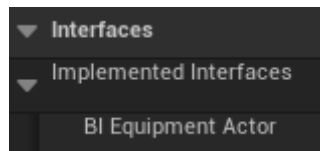
Copy Scene Capture Component.



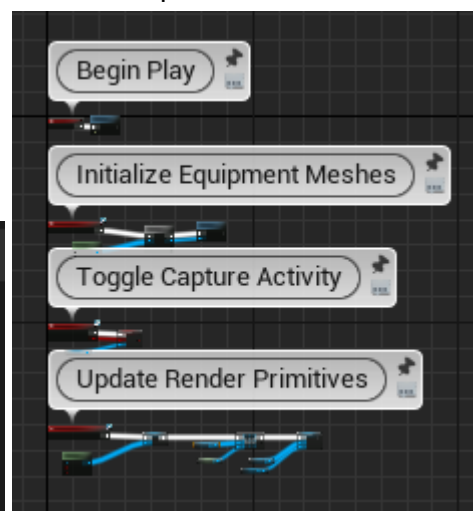
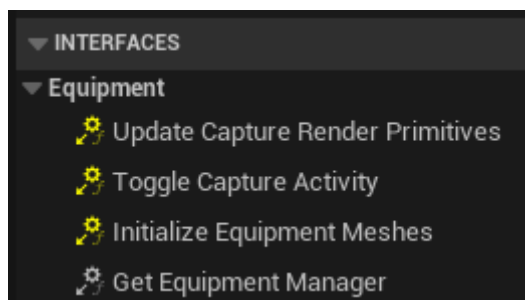
Copy Equipment Mesh Components.



Class Defaults -> Implement Interface.



Copy all logic for interface functions. Copy all logic from Event Graph.

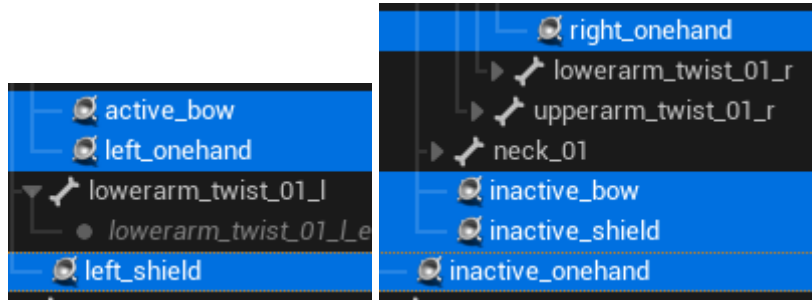


Player Character Skeleton

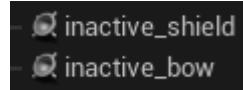
For demo content to work you need to add weapon sockets to your skeleton.

Open my **SK_Mannequin_Skeleton**.

Select and copy the next sockets to your character skeleton.



Adjust position for some sockets if necessary. Especially for



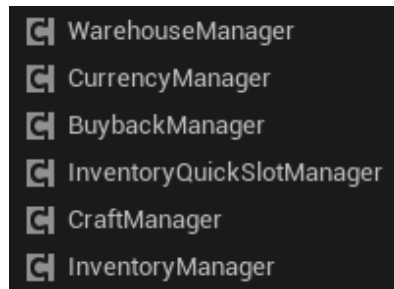
Player Controller

If you don't have a controller you can use my. Just open Game Mode and set it.

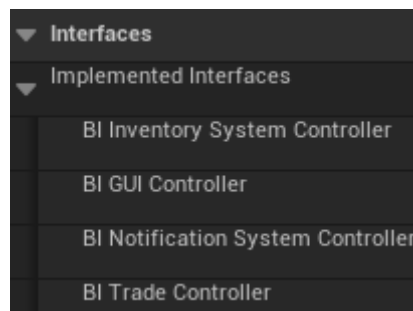
If you have your own player controller do the next.

Open my Player Controller and your Player Controller.

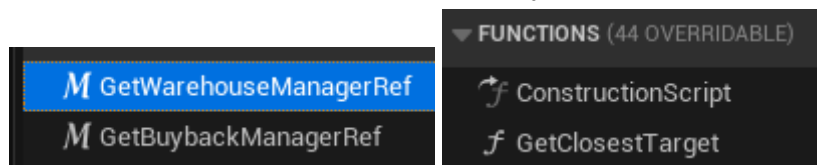
Copy Managers.



Class Defaults -> Implement Interfaces.

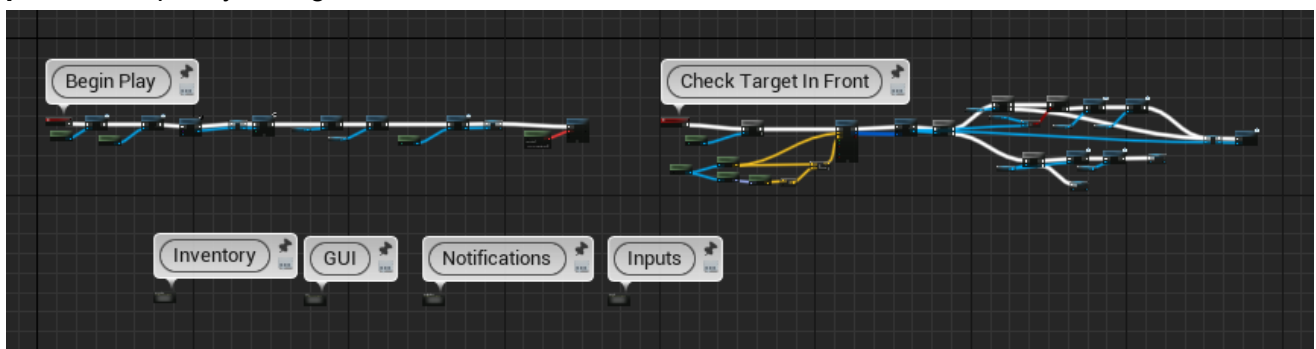


Copy macroses and one function. Select -> Ctrl + C -> Open your controller -> Ctrl + V.



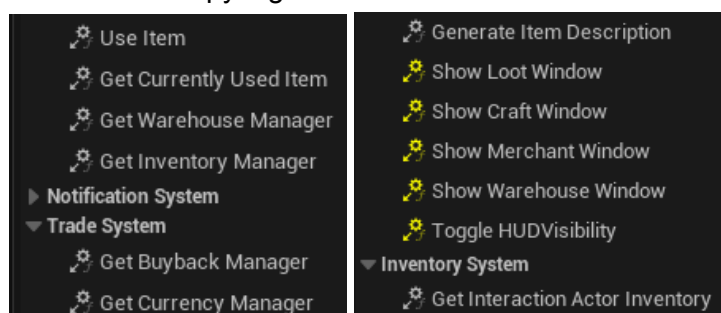
Copy all logic from the Event Graph.

Inputs can repeat your logic, so check it. And fix errors.



⚠ In use pin Q HUD no longer exists on node Q Get . Please refresh node or break links to remove pin.
⚠ In use pin Q HUD no longer exists on node Q Get . Please refresh node or break links to remove pin.

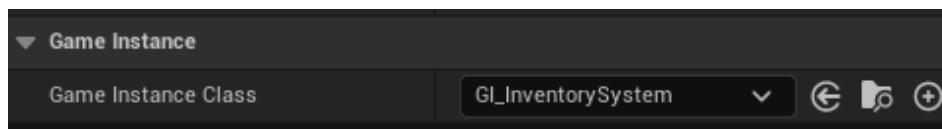
Open all gray interface functions and copy logic.



Save and Load

For saving and loading I use Game Instance and Save Game classes.

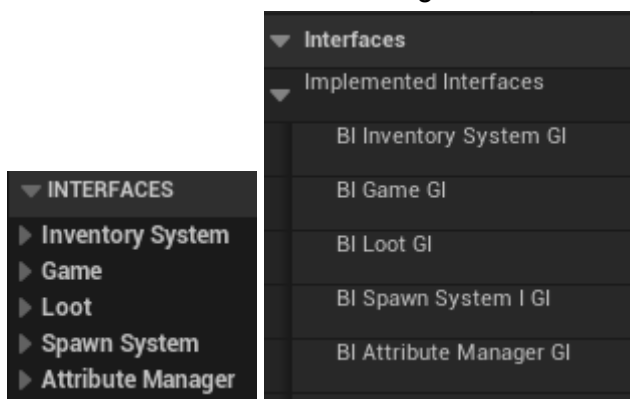
If you **don't have** your own classes, just open Project Settings -> Maps and Modes -> Set my Game Instance.



If you **have** your own classes, you need copy logic from my classes



Don't forget to implement interfaces in game instances. Each actor (Managers, Chest, PickUpItem, Spawner) uses its own interface to contact the game instance class.



Now, out of the box, if you will use my AI and other actors, everything will work.

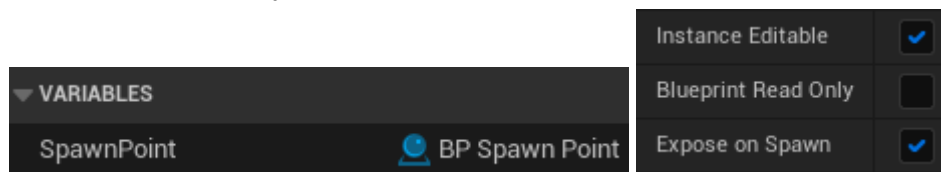
AI Parent

In your project, most likely, you are using your own AI class.

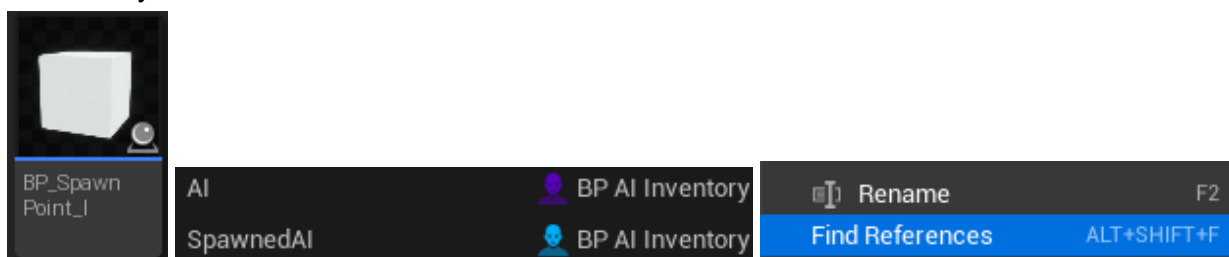
I don't know how exactly you spawn AI in the world. I do it via Spawners and Spawn Points. With them I can spawn AI when I really need it, and save all progress correctly.

You can use my spawn point.

First create **SpawnPoint** variable in your AI.

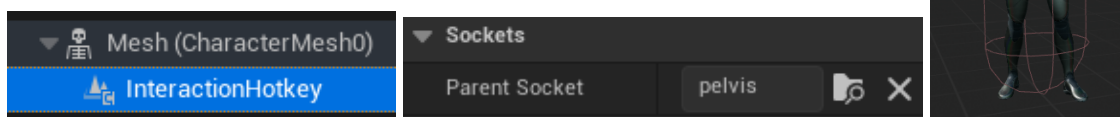


Then you need to replace AI references in BP_SpawnPoint. Use **FindReferences** to change all references correctly.

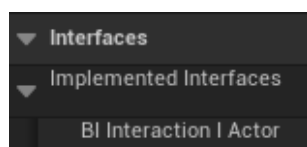


Copy WidgetComponent and attach it to Mesh to Pelvis bone.

Adjust location if necessary.



Class Default -> Implement Interface.



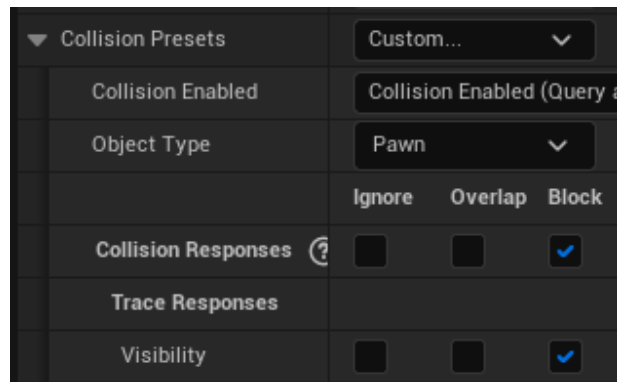
Copy logic from Event Graph.



Open **CanInteract** and set true.

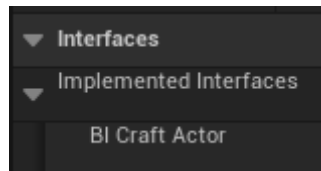


Make sure that AI capsule collision blocks trace responses by visibility channel. I use raycast by this channel in Player Controller, for checking targets in front of character.

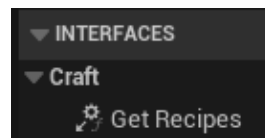


AI Crafter

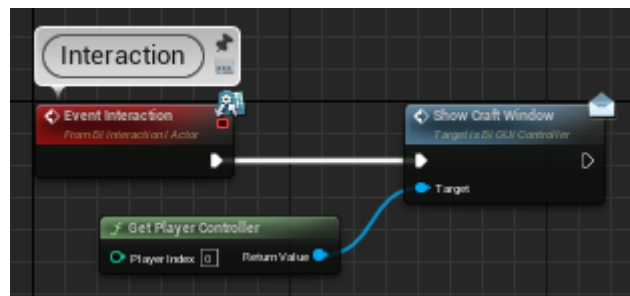
Class Defaults -> Implement Interface.



Copy logic from the function.

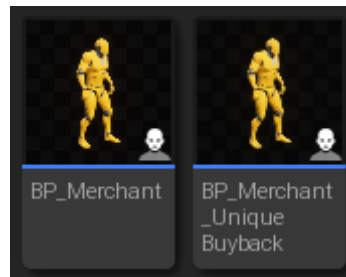


Copy logic from Event Graph.



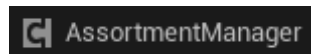
AI Merchant

I have two classes. Merchant_UniqueBuyback is the child of Merchant.

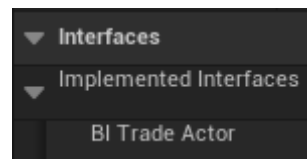


For **BP_Merchant**.

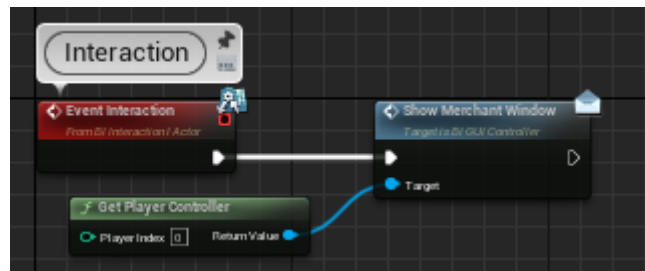
Add manager.



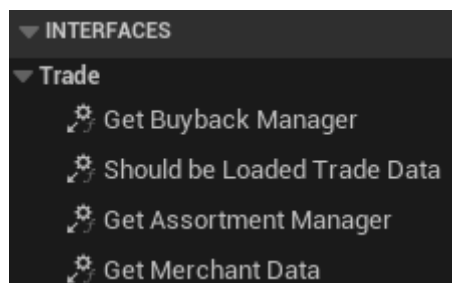
Class Defaults -> Implement Interface.



Copy logic from Event Graph.

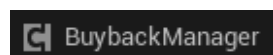


Copy logic for functions.

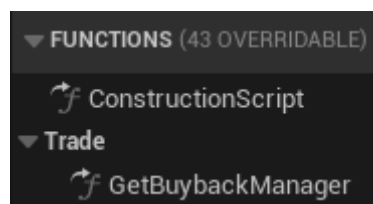


For **BP_Merchant_UniqueBuyback**.

Add manager.



Override interface function.

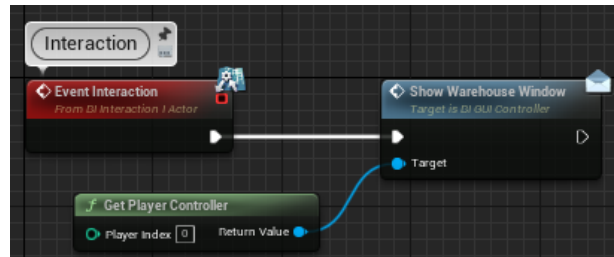


AI Warehouse Keeper

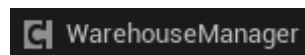
I have two classes. WarehouseKeeper_UniqueWarehouse is the child of WarehouseKeeper.



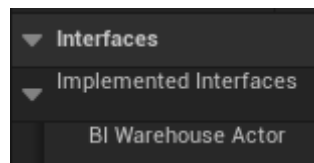
For **BP_WarehouseKeeper**.
Copy logic from Event Graph.



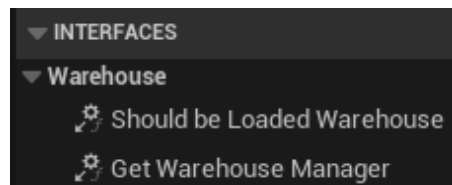
For **BP_WarehouseKeeper_UniqueWarehouse**.
Copy Manager.



Class Defaults -> Implement Interface.

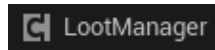


Copy logic for functions.

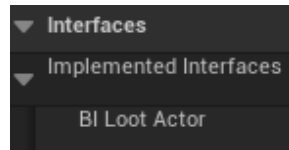


Add Loot Manager to AI

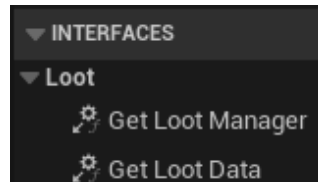
Add Manager.



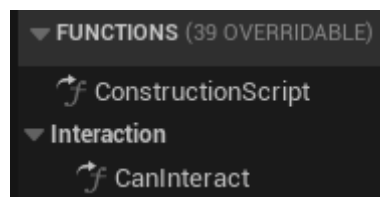
Class Defaults -> Implement Interface.



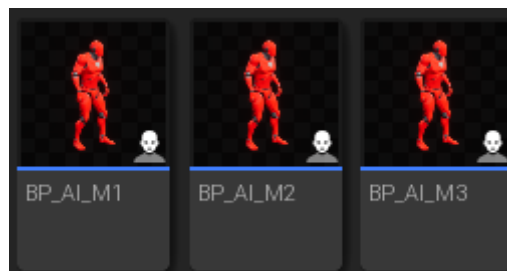
Copy logic for functions.



Override interface function.



Then you can open my AI examples and decide how you want to give loot to the player, and copy the appropriate logic.



Changelog

2.0

15 October 2020

The main goal of this update is to improve the architecture of the project in order to support new functionality. So for example, now you can connect Warehouse and Merchant managers to AI, if you need to make separate warehouses, or merchants with their own buyback slots.

Also now you can add more tabs to your inventory or warehouse. Storing information about items is improved with data assets.

- one large component divided on several small ones
 - slot manager (Inventory or Warehouse)
 - currency manager
 - inventory hotkey manager
 - merchant manager
 - equipment manager
- Tabs
 - before, you only had 2 tabs (overall + quest)
 - now, you can create any amount tabs you want, for each item category
- Storage info about items improved
 - data tables are replaced with Data Assets
- Equipment Leveling are added
- Loot manager
 - 3 methods are added: spawn, interaction, interaction + window
 - pickup objects are added
- Documentation is rewritten
- Videos are updated

2.1

25 December 2020

- crafting is added
- BP_ItemAction is added
 - The logic of using the item is described in children of this class
- “Dismantle” function is added for item
- several bugs is fixed
- UI_Manager is removed. All logic is moved to HUD

2.2

25 February 2021

- merchant has been reworked
 - Assortment Generator is added (assortment is updated when the player starts to interact with NPC. Next update will be available after N seconds)
- minor improvements in code
- bug is fixed
 - if item amount was equal to 1, it was just removed from inventory and not inserted to socket
- description tooltips are added to variables over entire projects

3.0

14 August 2021

- I completely rewrote all logic in project
 - Now there is only one data asset for Item (PDA_Item).
In the previous iteration I used inheritance for data assets. It means that I created a child class from PDA_Item and added additional parameters (for example for equipment). But in this iteration I decided not to use inheritance, because it complicates the project structure. This demo is very simple, but for real game it can be a good idea to use inheritance.
 - Containers for each item category are created automatically, you need just set settings in SlotManager
 - Equipment logic has been improved. Now there are active and inactive sockets
 - the settings for LootManager are removed. Now just call the necessary functions depending on how you want to give loot to the player (see my Demo scene)
 - Bow is added. Now there are one slot for Bow and one slot for one-handed weapon like Sword, Axe
- All GUI has been reworked (logic and visual)
- Documentation has been updated
- No new mechanics

I just improved the project. Now I hope it is cleaner and more understandable.

I removed a 'grid' crafting system. I decided to focus on only one system (with known recipes). Creating and supporting two crafting systems is too difficult.

3.1

02 December 2021

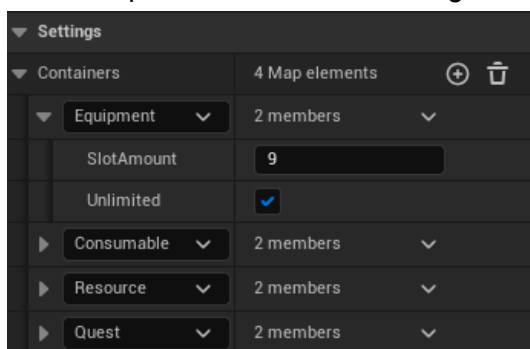
- **InventoryHotkeys** is renamed into **InventoryQuickSlots**
- Now **Equipment Manager** loads progress only for Player
- Now **Equipment Manager** accepts the item first, and then the inventory slot
Useful if you want to use the manager with AI, just adding the item. I do this in RPGTools
- Now when using **DetermineItemLevel** function, item's level is clamped in range (1;MaxLevel)

3.2

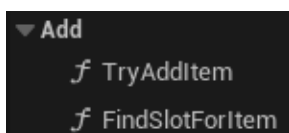
9 June 2022

Slot Manager

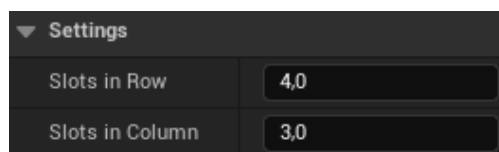
- Now **SlotAmount** and **Unlimited** parameters in Slot Manager can be set for each container



- Add Item** logic is reworked
Now you need to use **TryAddItem**.
Before finding the slot for item we can make some checks, and also, at the end, get total added amount and show notification.

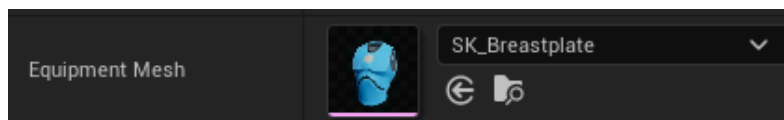


- Now **Warehouse Manager** (SlotManager) can be added to Actor class
See Warehouse section.
- Now when you place **WBP_Inventory_Container** or **WBP_Warehouse_Container**, you can set settings



Equipment

- **Equipment Mesh** parameter is added to PDA_Item

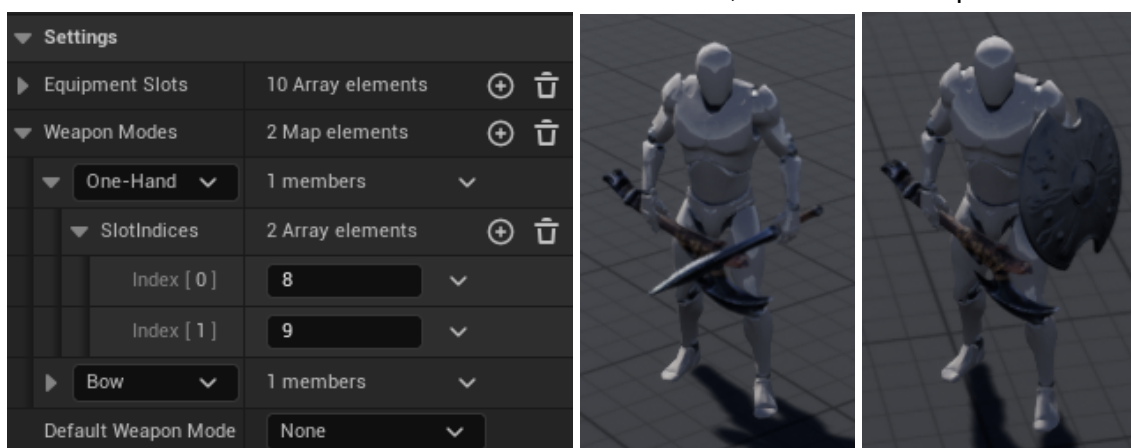


The appropriate logic is created.

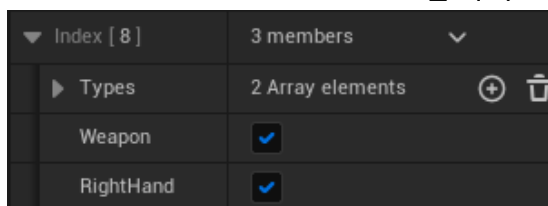


- **Weapon Mode** feature is implemented

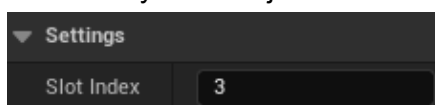
Now one or more slots can be activated at the same time, that allows to implement dual weapons



- **Weapon** and **RightHand** parameters have been added to S_EquipmentSlot

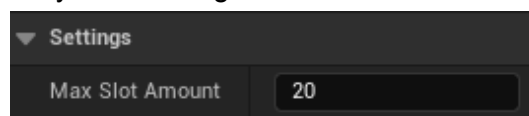


- **SlotPolicy** parameter is added to PDA_Item
- Now **Weapon** parameter in PDA_Item is soft reference
It is useful for RAM optimization. Actors will take place in memory only when spawned and attached to sockets.
- Now when placing **WBP_EquipmentSlot** you need just set Slot Index

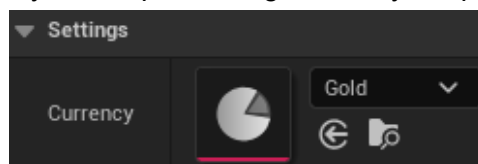


Trade

- **Assortment** is slightly reworked
 - Now assortment is generated at start game
 - Assortment item level is updated when player levels up
 - Update timer is removed. It was assigned with AI and stopped working when AI was unloaded from the world, which breaks the rules of the game, when we tell the player the assortment is updated every N minutes.
 - Now assortment can be regenerated when sold out. See S_MerchantData.
 - Now Item can be sold in unlimited amount. If Min and Max are equal to 0, amount is unlimited.
- Now **Buyback Manager** can be added to Actor class
See Buyback section.
Now Merchant can have its own buyback slots. If manager is not added to actor, global buyback slots will be used.
- Now **Assortment** and **Buyback** slots are **saved** for each AI
- Now **MaxSlotAmount** in Buyback Manager can be unlimited

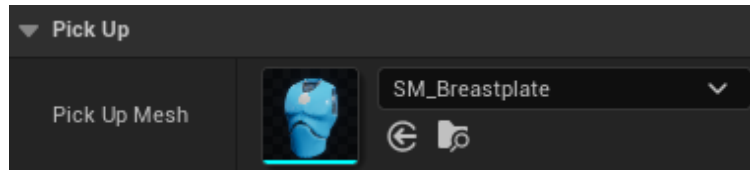


- Now **WBP_Currency** is placed manually
WBP_Currency_Container is removed.
When you place WBP_Currency you need to set settings.
It is more flexible solution since you can place widgets in any shape



PickUp Item

- Now all **BP_PickUp_Item** actors are saved and loaded
- Now each item can have its own mesh
New parameter is added to PDA_Item. If None, gray cube is shown.



- On **Scan**, item amount is displayed

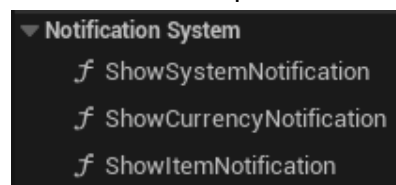


- Item Description appears on interaction



Overall

- **Notifications** have been added
BI_NotificationSystem_Controller interface is implemented in Player Controller.



- **Live Character Portrait** works in **ShowOnlyActors** mode
See Live Portrait section.
- Issue is fixed where full purchase amount was removed from trade slot, even if some amount was not added