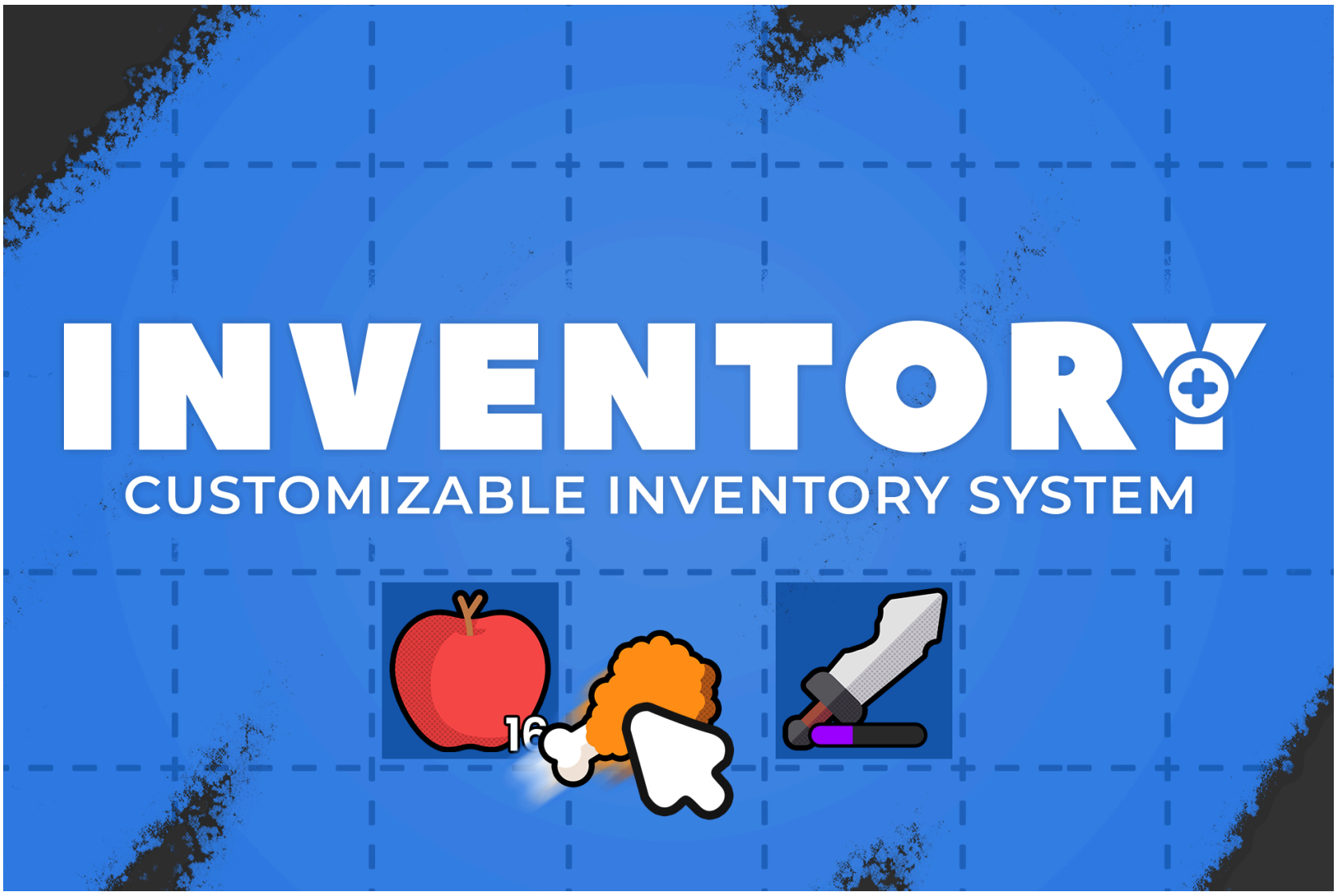


InventoryPlus

*Support email: **nappin.1bit@gmail.com***

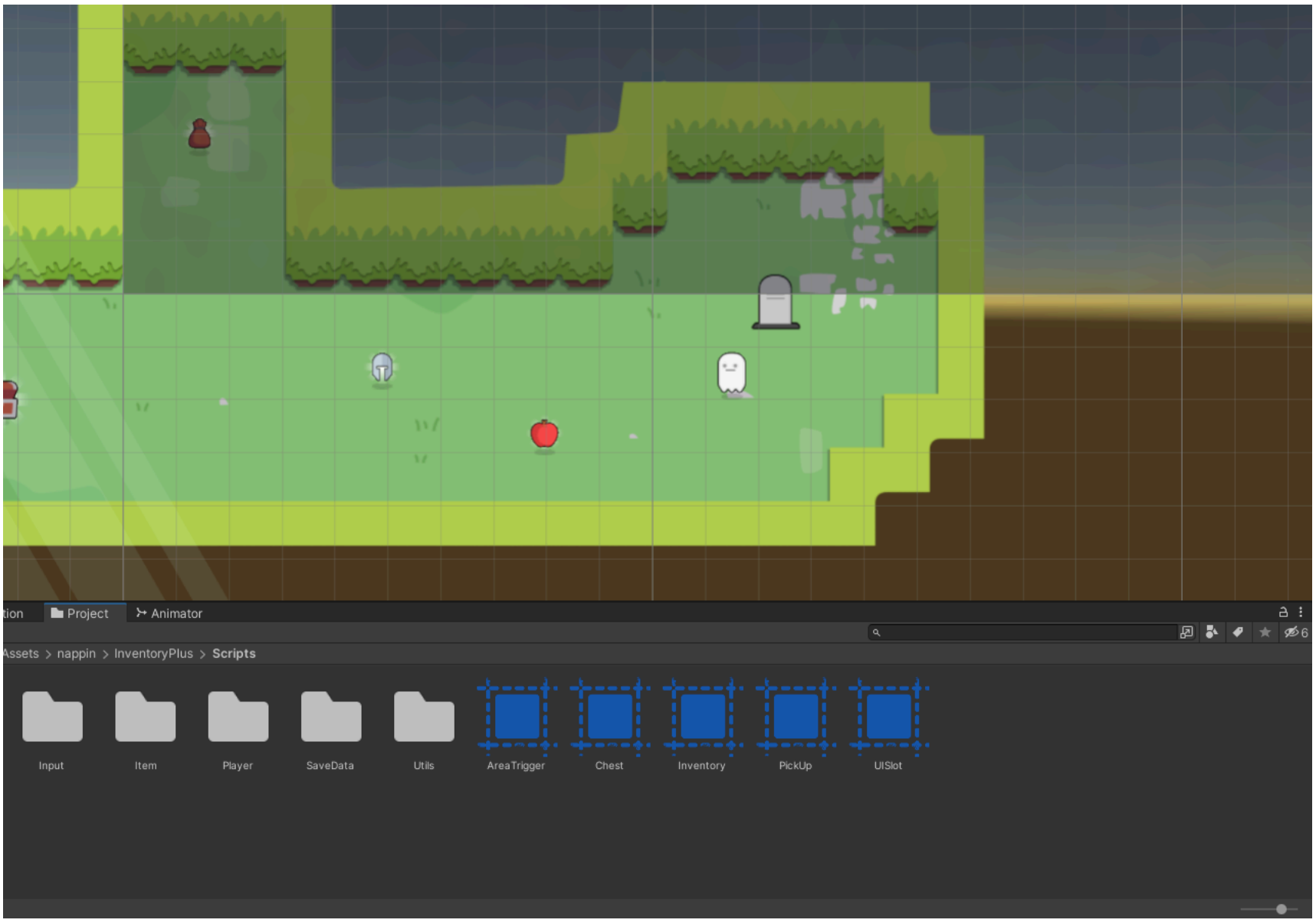
Table of Contents

Asset Content

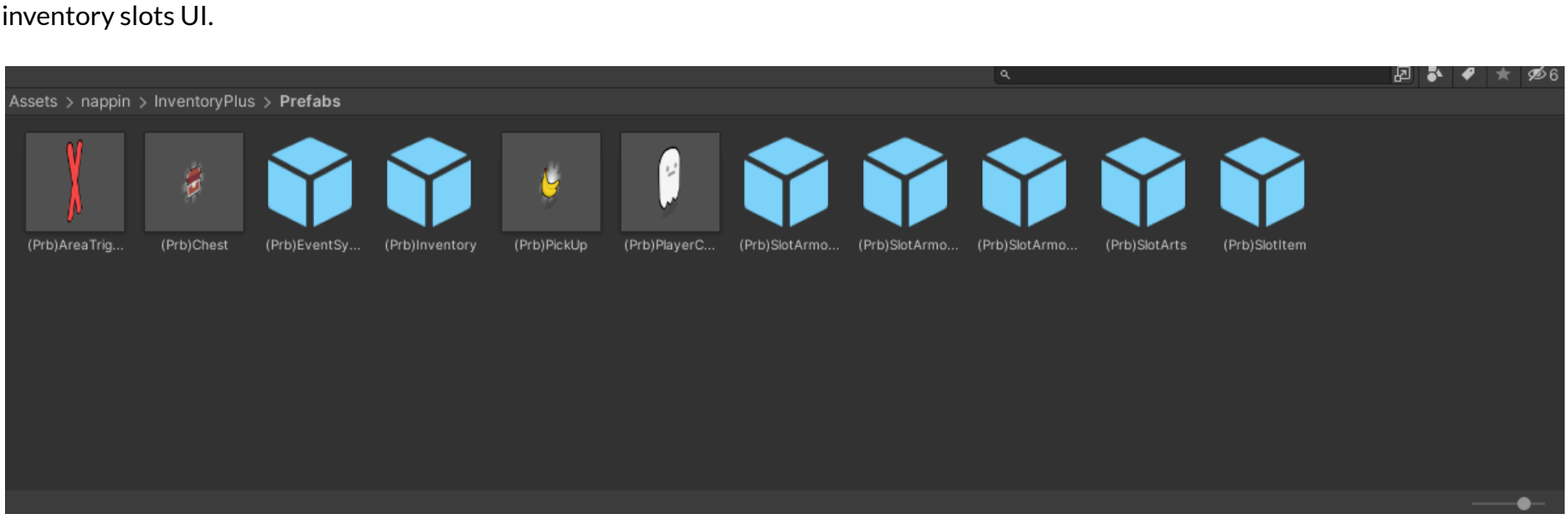
InventoryPlus contains multiple assets but the core content can be found in the *Prefabs* folder and in the root of the *Scripts* folder. The following documentation will touch on the core content of the InventoryPlus asset and give an overview of the additional scripts / assets / prefab that support it (SaveSystem, MouseInput etc..).

In the *Scripts* folder you can find the **Inventory** and **Chest** scripts which are the main components used to manage in-game inventories.
You can also find the **UISlot** script, which is a component associated with each individual slot in an inventory and chest.
In the folder there is also the **Pickup** script that is used to handle pickuppable items as well as the **AreaTrigger** script, which is used to trigger inventory events.

Inside the other folders there are additional support scripts, the most important one is the Input folder that contains **InputReader.** It's a script that interprets inputs and converts them into inventory events. We will dive into the specifics in the Input chapter.



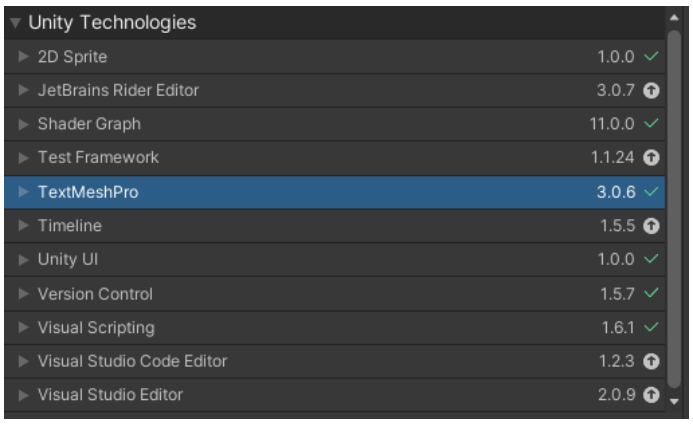
In the *Prefab* folder are located the assets, sprites and characters used in the Sample scene. Here there are also variations of the inventory slots UI.



Only 1 element is necessary to utilize InventoryPlus and that is the **Inventory** script. To access all the additional inventory features (like Toggle, Use, Drop etc…) the **InputReader** script **(**or a custom script that listens to player inputs) is needed.

External Packages

The asset doesn't require any additional package. TextMeshPro is used in the InventoryPlus package to render texts but the asset is not required and the default Unity Text component can be used instead (or any other text renderer).

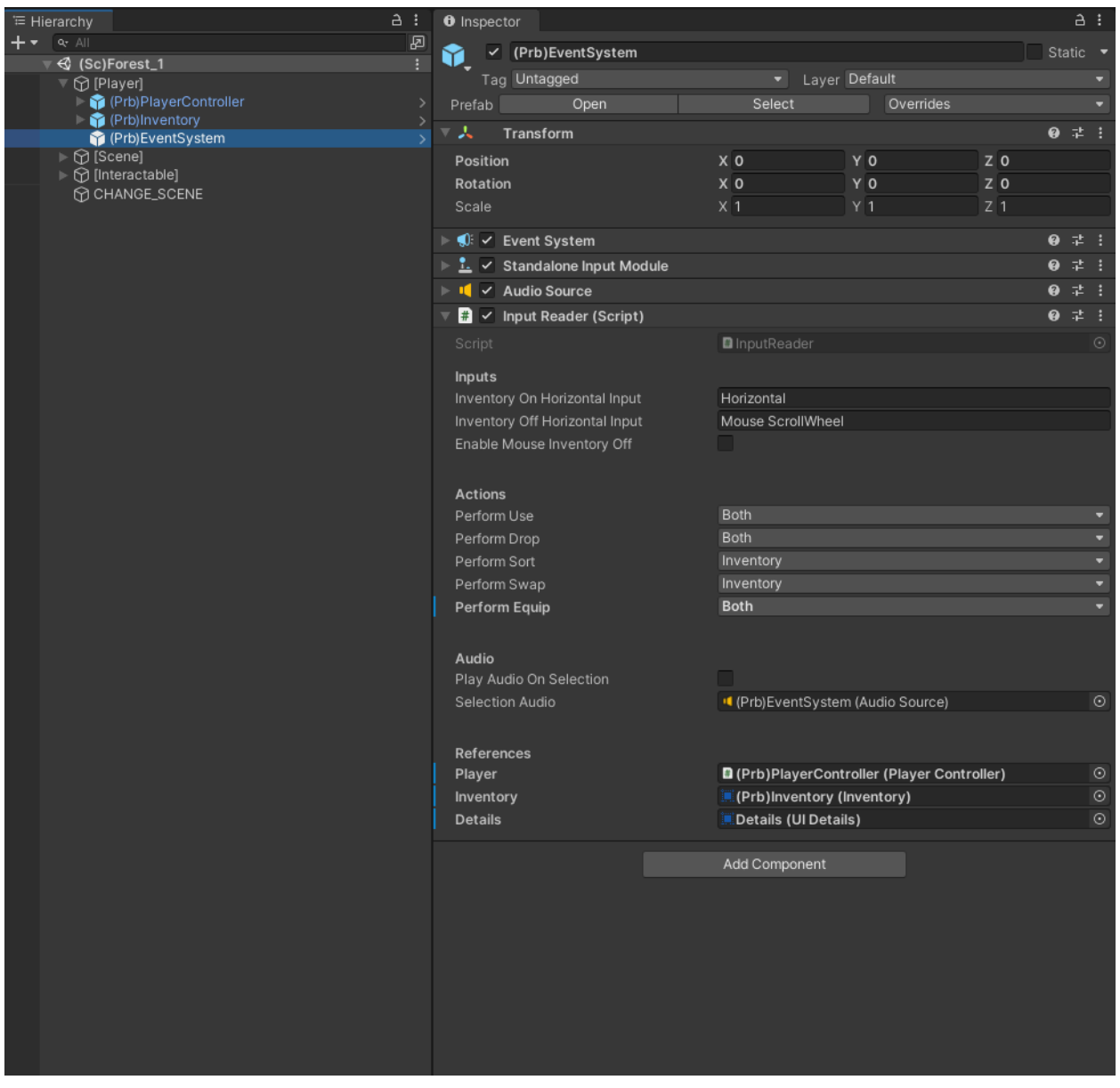


N.B. If you intend to use TextMeshPro, there is no need to install it before downloading the asset. When the InventoryPlus package is imported the install prompt for TextMeshPro will be displayed automatically.

Input and InputReader

In the sample scenes there are 3 elements that rely on player input:

- ***Inventory:*** the inventory navigation relies on UI buttons.
- ***InputReader:*** this script is required to trigger additional inventory events (like Toggle, Use, Equip, Drop, etc..). What the script does is simply tie a button press to an event and decide when certain actions can be performed. The component uses the Old InputSystem and the IPointer events. Support for the NewInputSystem out of the box is being developed, if your project requires it, it's recommended to run both Inputs.



- ***PlayerController:*** the player top down character controller. The top down controller is used only for demonstrative purposes and is not tied to the inventory management.

N.B. If you intend to use a GamePad in your project it is recommended to assign an InputManager entry to each action (you can find the code below inside the InventoryActions method in the InputReader)

```
1 reference
private void InventoryActions()
{
    // Reference to the Input for using an Item, it is recommended to swap to an InputManager entry: https://www.youtube.com/watch?v=DB5xkz-3nHk
    // Example of a InputManager entry: (Input.GetButtonDown("UseItem"))
    if (Input.GetKeyDown(KeyCode.U) && currentSelectedObj != null && ((performUse == ActionState.Inventory && inventoryOn) || (performUse == ActionState.HUD && !inventoryOn) || (performUse == ActionState.Both)))
    {
        inventory.UseItem(currentSelectedObj.GetComponent<UISlot>());
        if (details != null && inventoryOn) details.UpdateDetails(currentSelectedObj.GetComponent<UISlot>(), false);
    }
```

Why isn't the gamepad supported by default? In the Old Input System gamepad support requires an InputManager entry. Unfortunately the InputManager is project specific so the setup can't be imported when downloading the tool from the asset store.

Do you need additional support when setting up the InputManager entries? Do you need help setting up the new Input System? Reach me via email, I'm always happy to help!

Item Setup

Items are the type of object that make up inventories and chests. In the InventoryPlus package are included 24 Item Types showcasing what can be created with the asset. In this chapter we'll explore how to create an Item Type from scratch and all the available options.

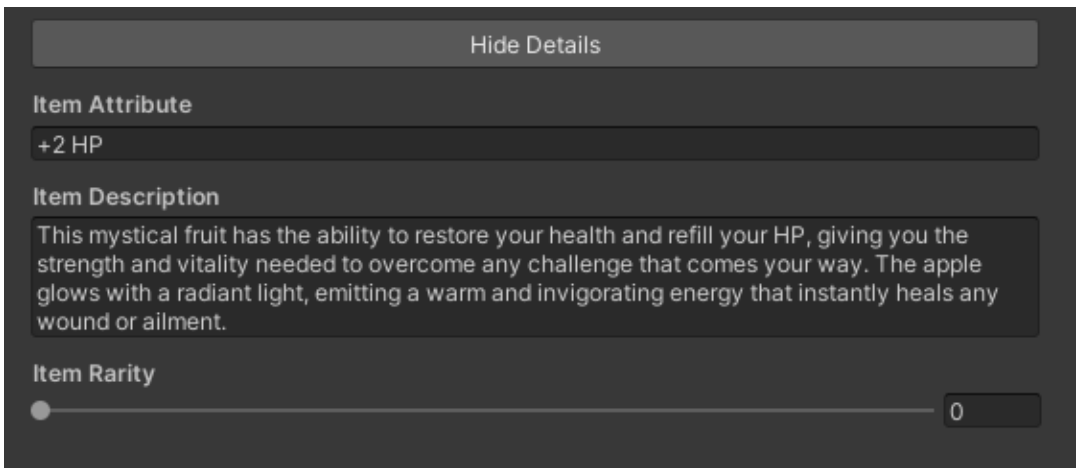
To create an Item Type right click in the **Project Window -> InventoryPlus -> Item**. Let's now explore all the available options. The options listed above can be easily customized by tweaking the *Item* and *ItemEditor* scripts.

Core Attributes

Here you can assign the core property of the item as well as its default visual.

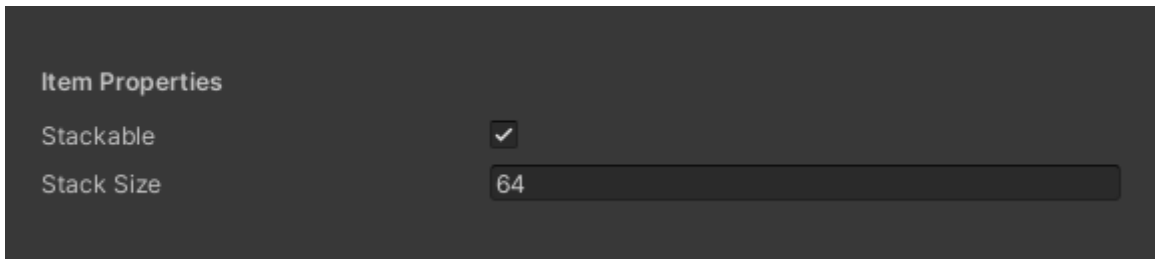
Its variables are:

- ***Sprite***: the default visual of the item that is referenced in the inventory, chests and wherever a preview of the item is displayed.
- ***Name:*** the name of the item. The name can be non unique and multiple Item Types can share the same.
- ***ID***: the ID of the item. This text should be unique for each item because it's the text used to identify the Item Type via code.
- ***Category***: the text is used to restrict inventory slots and to perform specific actions in the inventory.
- ***Show / Hide Details***: when the option is toggled additional properties are available. Remember that these details can be easily added / removed tweaking the Item and the ItemEditor script.



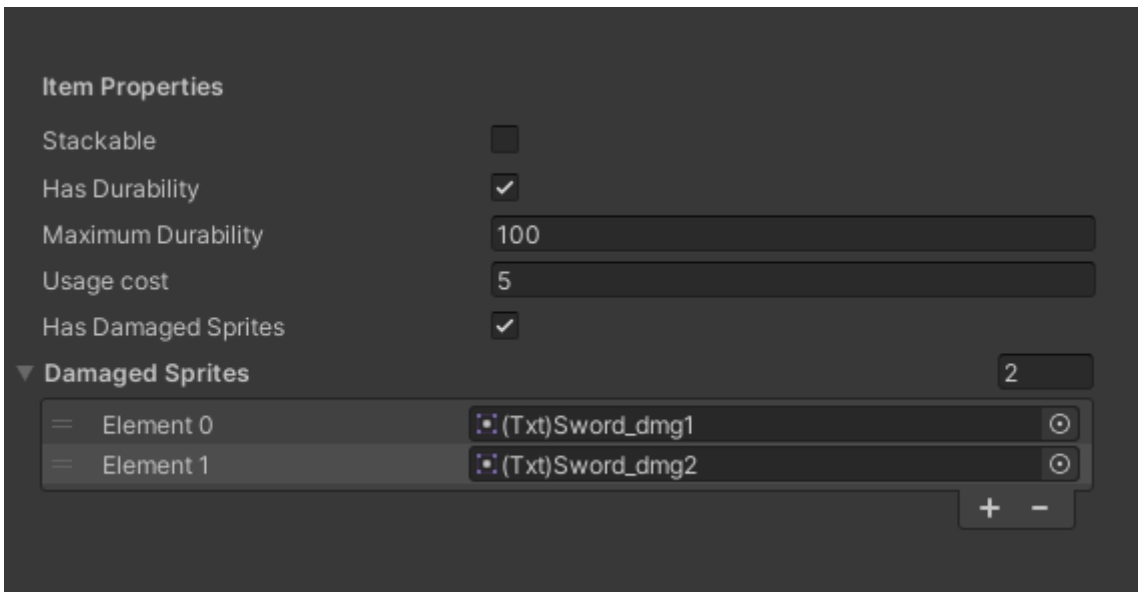
Item Properties

The Item Properties define how the item will be displayed in the inventory and chests and how it will behave when the inventory actions are performed.



Its variables are:

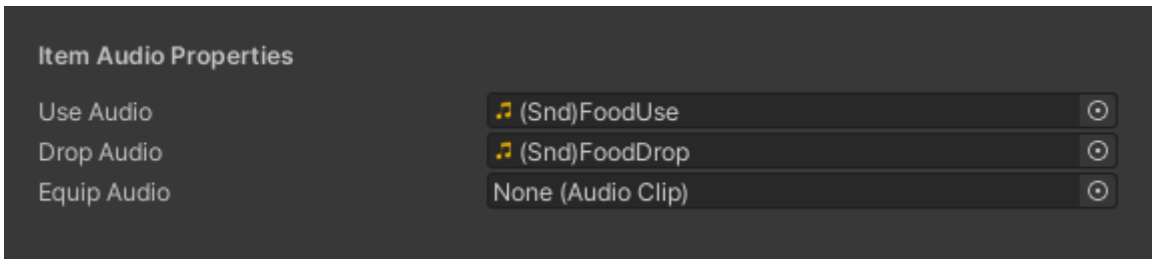
- ***Stackable:*** if the Item Type is stackable it means that more than one instance of that item can occupy the same slot.
    - ***Stack Size:*** define how many items can occupy the same slot.



- ***Stackable:*** if the item is not stackable it means that only one instance of the item can occupy the same slot.
    - ***Has Durability:*** if the option is true it means that the item can be used.
        - ***Maximum Durability:*** represents the maximum durability of the item.
        - ***Usage Cost***: represents how much the durability is reduced when using the item.
    - ***Has Damaged Sprites:*** when true a different visual is used for the item sprite based on its durability.
        - ***Damage Sprites:*** the sprites used when the item is damaged.

Item Audio Properties

The audio played on item events. If the fields are empty no audio is reproduced.



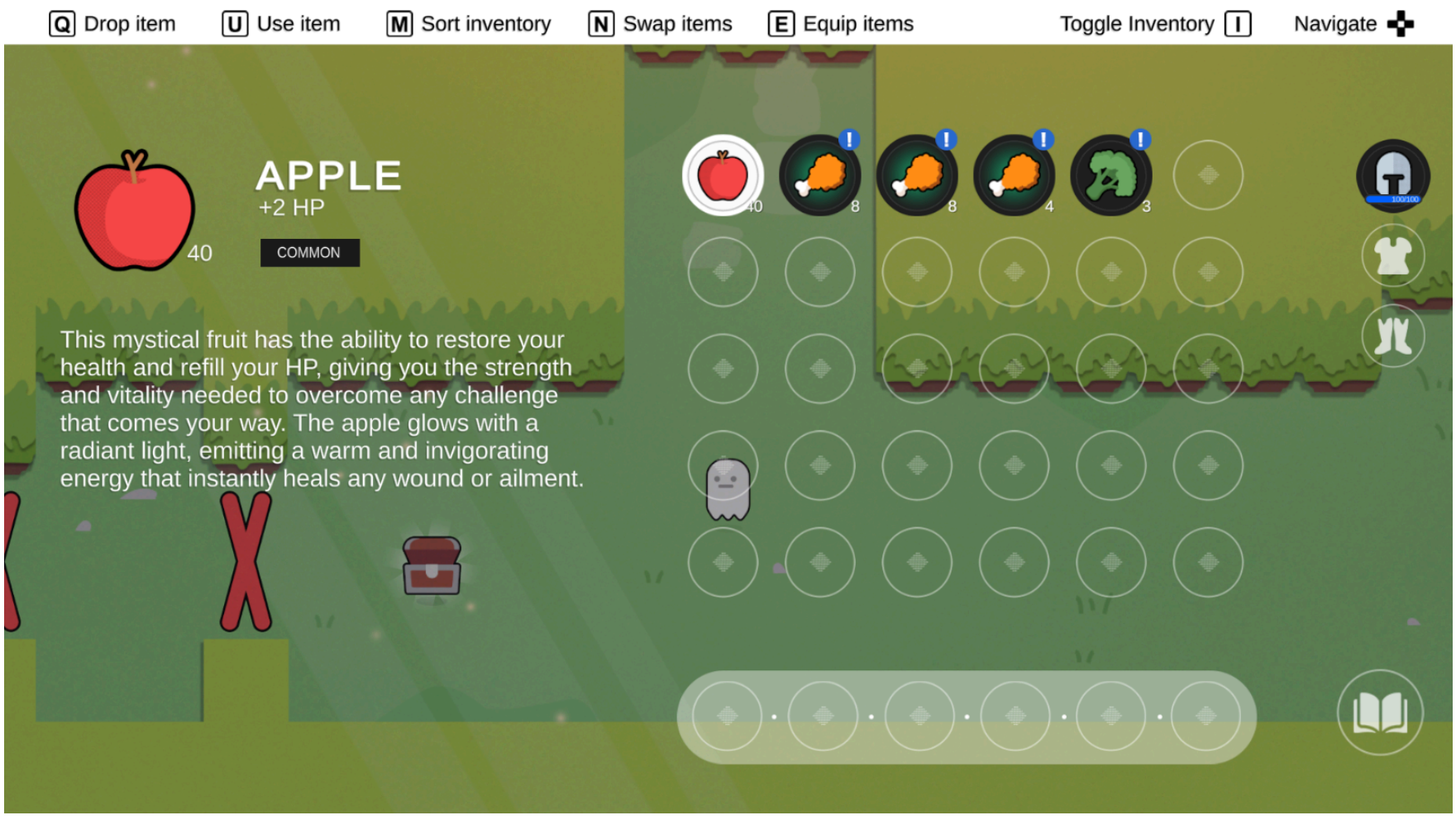
Its variables are:

- ***Use Audio***: audio reproduced when the item is used.
- ***Drop Audio***: audio reproduced when the item is dropped.
- ***Equip Audio***: audio reproduced when the item is equipped.

Inventory

The inventory gameObject contains a few visual elements and additional content (like the UIDetails), but to work it only needs: the **Inventory** component itself and at least one **UISlot**.

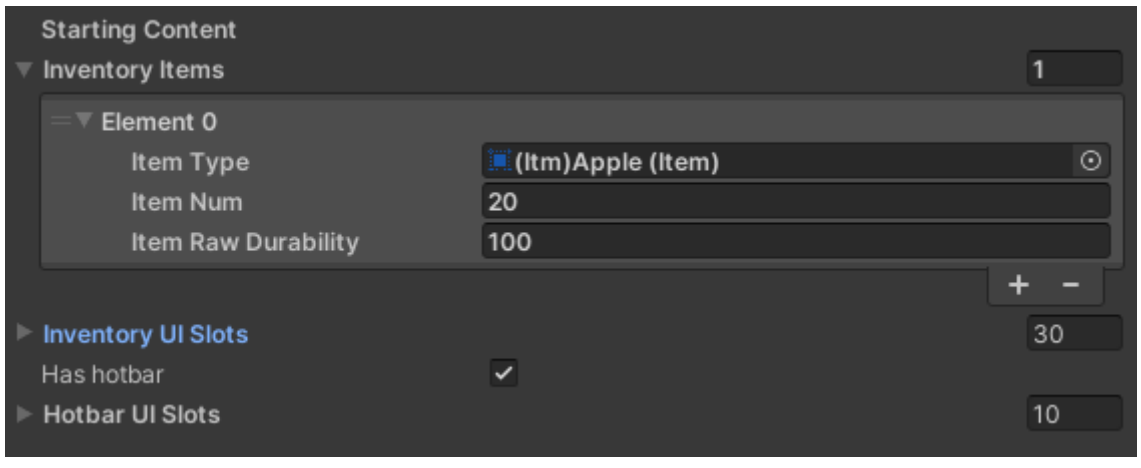
Let's explore the properties available for the **Inventy** component. We'll dive into the specifics of the [UISlot script in this chapter].



> NB: the **Inventory** component relies on the Inventory script and the InventoryEditor script, if you want to tweak it remember to edit both of them.

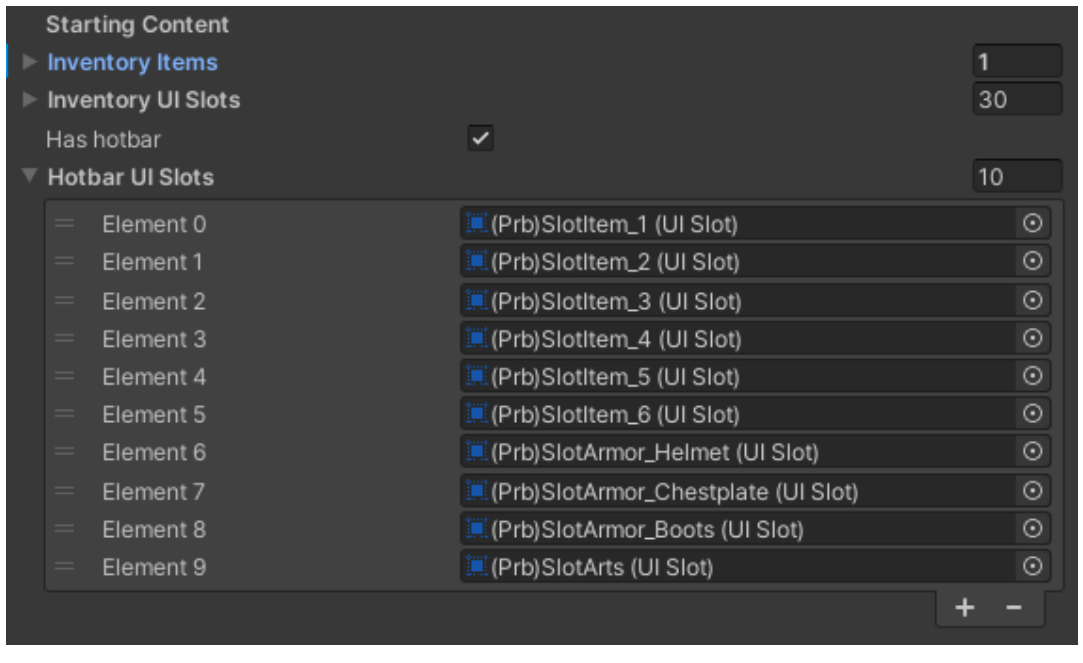
Starting Content

Here we assign the inventory UISlots as well as the hotbar UISlots (if the hotbar is enabled). We can also assign an "initial inventory" that will be given to the player when the scene is launched.



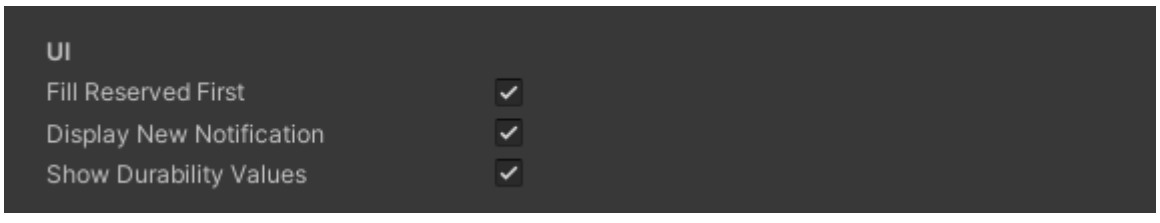
Its variables are:

- ***Inventory Items***: the items assigned to the inventory when the scene is launched. All the items in the list will be added to the inventory until it reaches capacity. For each element we define:
    - ***Item Type:*** the Item Type that is added to the inventory.
    - ***Item Num***: number of items.
    - ***Item Raw Durability:*** durability of each item, the value is not relevant if the item doesn't have durability (like in the screenshot above).
- ***Inventory UI Slots:*** reference to all the UISlots used in the inventory
- ***Has Hotbar***: if is enabled an hotbar will be added to the inventory. The hotbar element is also displayed when the inventory is closed. Remember to also remove the visual of the hotbar if you are not interested in having it in you scene.
    - ***Hotbar UI Slots:*** reference to all the UISlots used in the hotbar.



UI

Global properties about the behaviour of the UISlots.



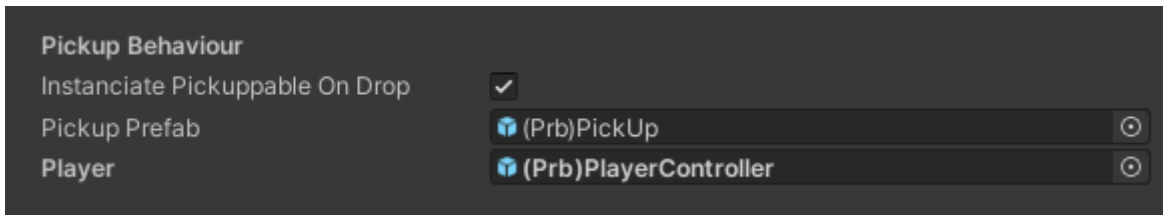
Its variables are:

- ***Fill Reserved First:*** when the option is turned on, if an item is added to the inventory and the instory has a reserved slot for that item category, that slot will be filled automatically (unless it's already filled).
- ***Display New Notification***: when the option is turned on, a balloon with an exclamation mark will be displayed on top of item instances just added to the inventory. The notification disappears when the item is selected, hovered or an inventory operation is performed with it.

- ***Show Durability Values***: when the option is turned on, the items that have durability will show the value in addition to the progression bar.

Pickup Behaviour

This section contains properties relevant to the items when items are dropped from the inventory.

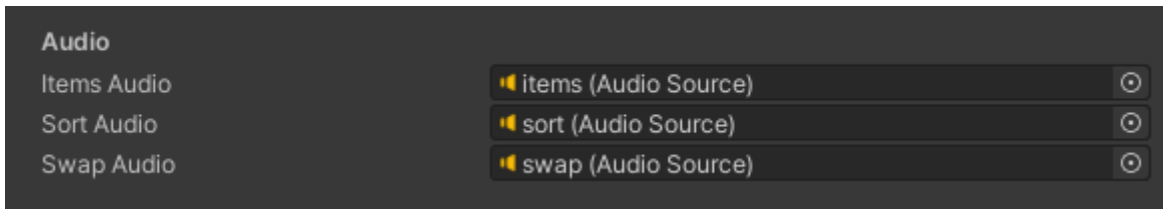


Its variables are:

- ***Instantiate Pickuppable On Drop:*** when the option is turned on, if an item is dropped from the inventory a pickup prefab is instantiated in the world. If the option is off, when an item is dropped it just gets destroyed.
    - ***Pickup Preb:*** reference to the pickup prefab.
    - ***Player***: reference to the Player. This reference is needed to know where in the world the pickup should be instantiated.

Audio

The audio that is played by the inventory when specific inventory events occur. If the section is blank no audio is played.

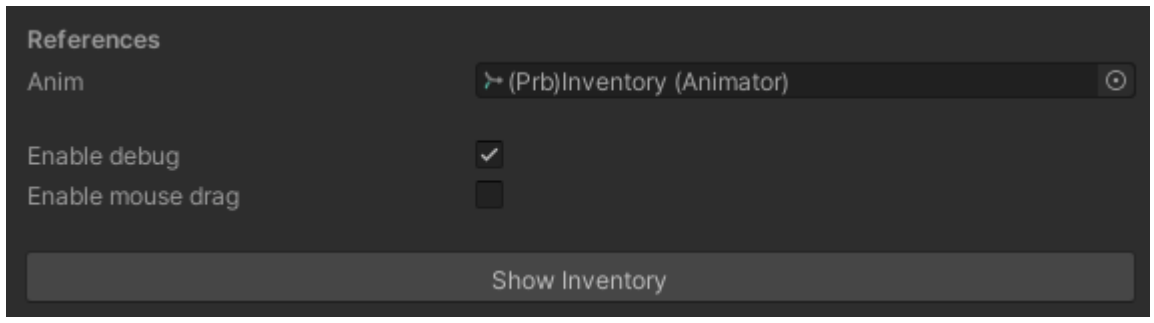


Its variables are:

- ***Items Audio:*** audio played when an item is selected.
- ***Sort Audio:*** audio played when the inventory is sorted.
- ***Swap Audio:*** audio played when 2 items are swapped.

References and Debug

In this section we reference the Animator used for the inventory animations, as well as a couple of debug options.



Its variables are:

- ***Anim:*** the animator used for inventory animations.
- ***Enable Debug:*** when it's on, it displays messages in the console for inventory events.
- ***Enable Mouse Drag:*** when it's on, it enables the usage of the mouse for the drag action.
- ***Show / Hide Inventory:*** we display the current content of the inventory without the need to open it in the scene. The content of this section is automatically updated when updating the editor (meaning when clicking or moving the cursor wherever in the except the scene or game window)

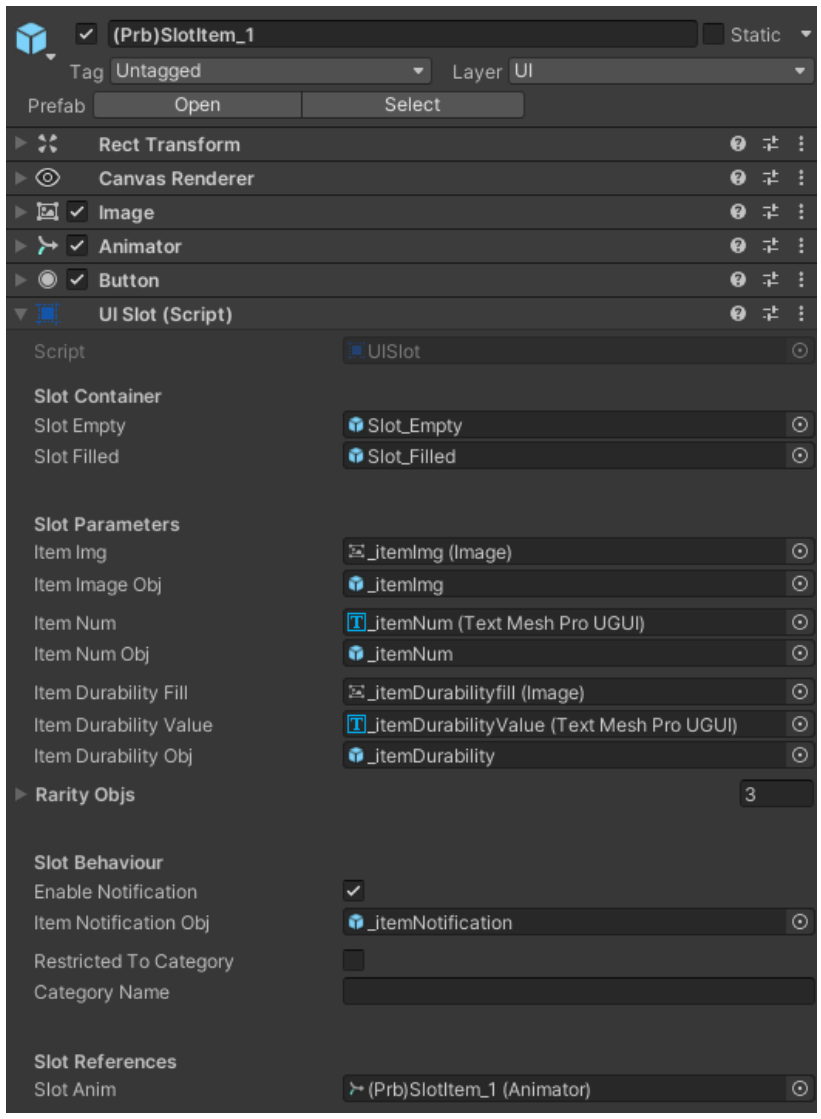
Inventory actions

Let's now dive into all the actions that can be performed in the inventory. I'll list all the inputs used in the sample scene but of course everything can be very easily changed:

- ***By pressing "I"*** the player can toggle the inventory. In this sample scene I have a hotbar and a couple of slots for equipment that I always display. I can of course customize what I want to display via animation and remove the hotbar if I want to.
- ***By pressing "N"*** or by using the mouse (if the option is enabled) I can move stuff inside the inventory. The swap of course respects the attributes of the UISlots so for example if I move a non-helmet item inside an helmet slot, the action is not performed.
- ***By pressing "E"*** the player can equip items. What the action does is look for an empty restricted slot, whose restriction matches the item category. If it finds one, the selected item is placed there. Of course the action doesn't work if the slot is already occupied or if there is no UISlot of that category.
- ***By pressing "U"*** the player can use items. If the item is stackable its number is reduced by one. If it is not stackable it's durability is reduced. When the durability or the number of an item reaches 0, the item is destroyed.
- ***By pressing "Q"*** the player can drop items. Multiple options are available for the drop action, for example you might want to destroy the item when dropped or maybe add it to the world to be picked up again. If a single item is dropped, a pickuppable object is generated with that item preview. If multiple different items are dropped a pickuppable object is generated with a grouped preview.
- ***By pressing "M"*** the player can sort the inventory. When sorting, items of the same type are added together and the inventory content is reorganized removing empty slots.

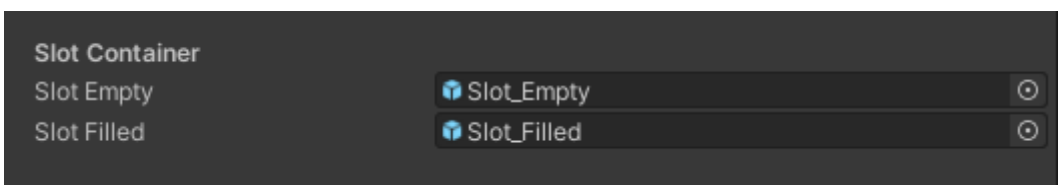
UISlots

The **UISlots** component is associated with the individual Slots that make up a chest and inventory (including the hotbar). Let's explore what's the content of the **UISlot** component and what it can do.



Slot Container

A Slot can be empty (if it doesn't contain an item) or Filled (if it contains an item). Each one of them also supports multiple states (hovered, clicked, normal etc…) based on inventory events.

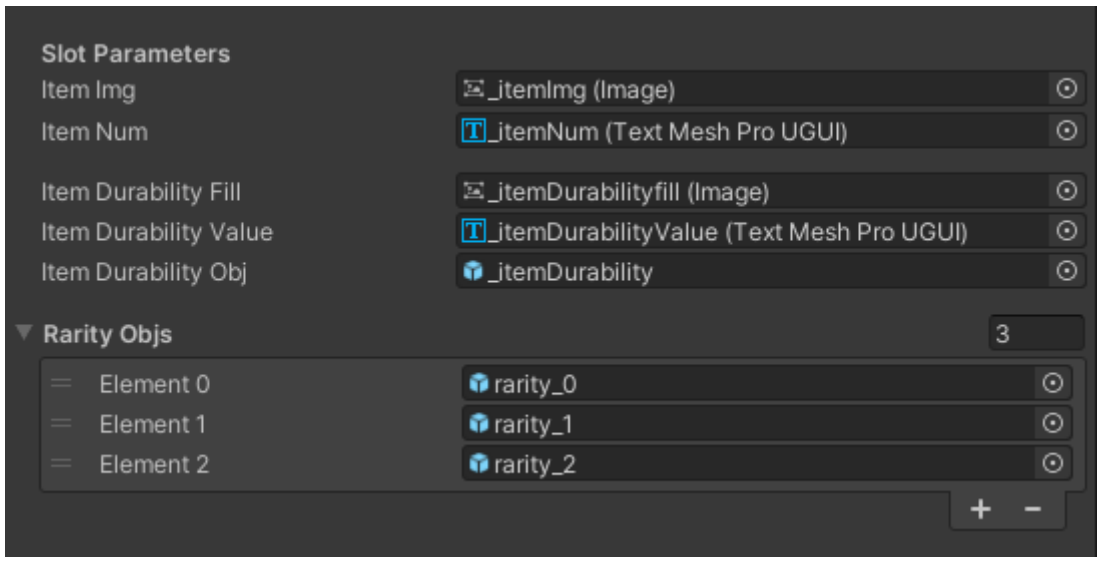


Its variables are:

- ***Slot Empty:*** reference to the gameobject parent of the Empty Slot.
- ***Slot Filled:*** reference to the gameobject parent of a Filled Slot.

Slot Parameters

Reference to text and strings that need to be filled when an item populates a Slot.

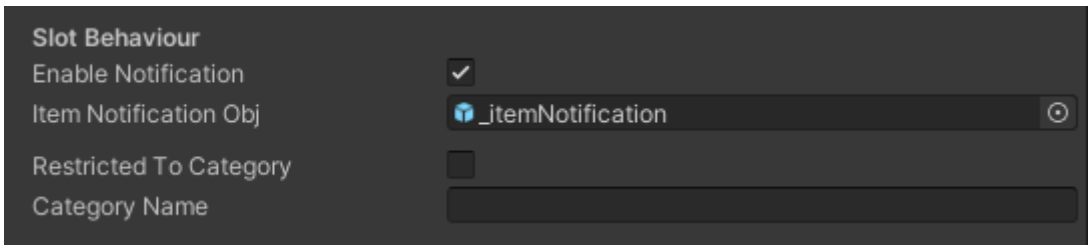


Its variables are:

- ***Item Img:*** reference to the SpriteRender that displays the item sprite.
- ***Item Num:*** reference to the TextMeshPro that displays the number of items.
- ***Item Durability Fill:*** reference to the Image that represents the durability filler of the items that have durability.
- ***Item Durability Value:*** reference to the TextMeshPro that displays the durability value of the item.
- ***Item Durability Obj:*** reference to parent GameObject of the durability component (this reference is needed for stackable items, in that case the ItemDurability Obj needs to be collapsed).
- ***Rarity Objs:*** the Item Types in the InventoryPlus asset have a durability value in a range from 0 to 2. A different visual is displayed in the slot based on the item rarity.

Slot Behaviour

The behaviour of the Slot when it comes to notifications and the restrictions.



Its variables are:

- ***Enable Notification:*** when the option is on items recently added to the inventory will have a “new” notification bubble.
- ***Item Notification Obj:*** reference to parent GameObject of the notification bubble. The GameObject will be collapsed or not based on the option above.
- ***Restricted to Category:*** when the option is on, only items of a specific category can fill this Slot.
- ***Category Name:*** the name of the category that is used if the Slot is Restricted (option above). The text must be the same one used in the category section of the Items (Food, Armor, Book…).

Slot References

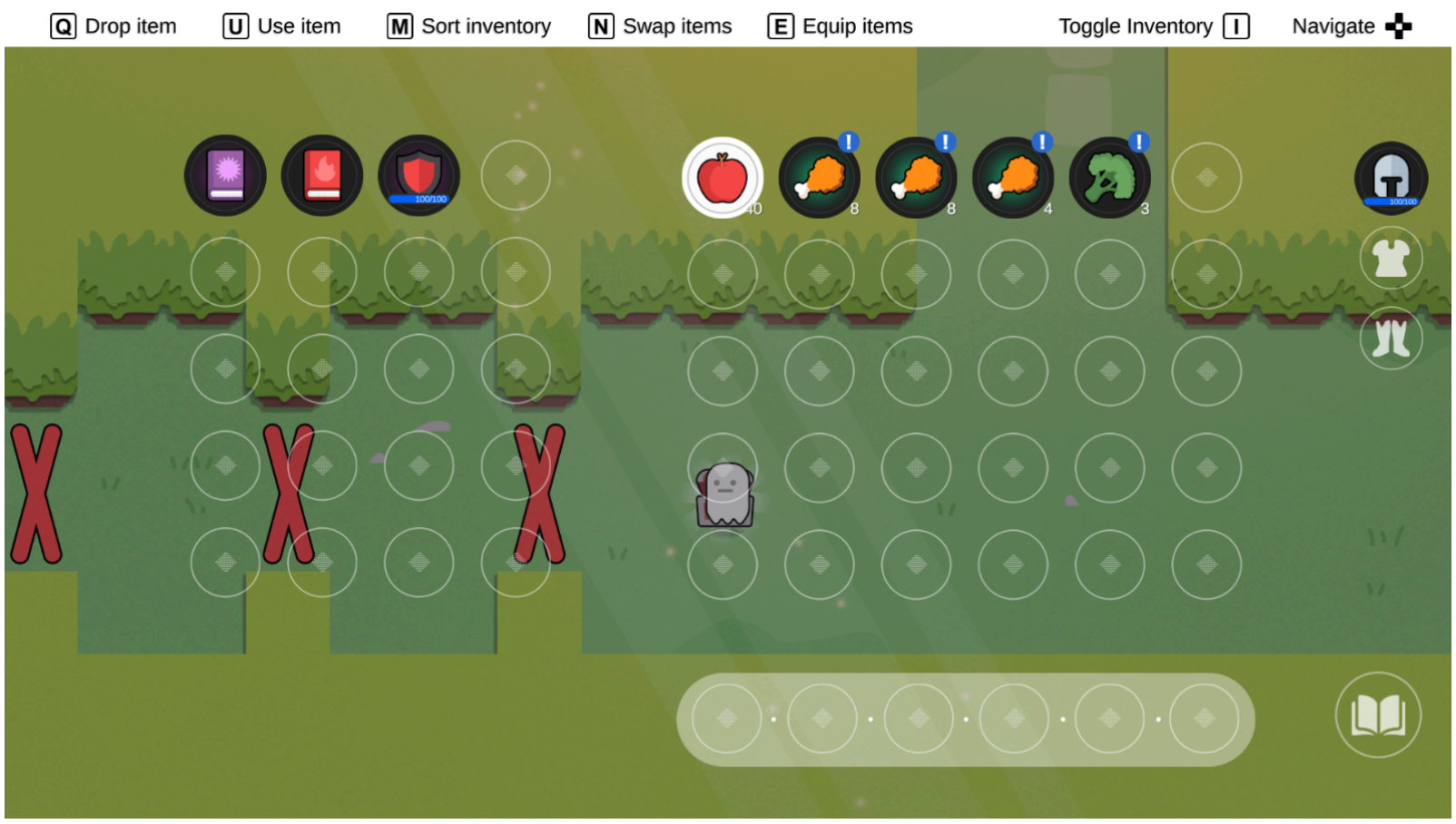
Reference to the Animator that handles the Slot animations in its various states.

Its variables are:

- ***Slot Anim:*** reference to the Animator component.

Chest

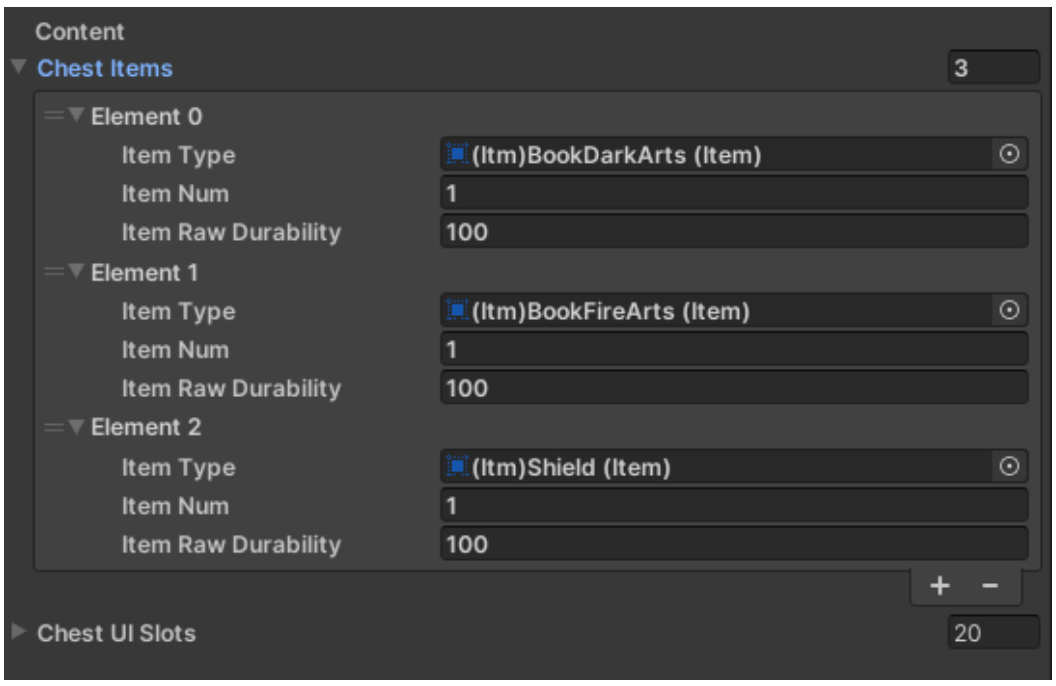
Chests behave in a very similar way to inventories. Chests are made up of **UISlots** components, when the player interacts with a chest its content is opened next to the inventory (via animation). In this state the player can perform the usual actions in the inventory and they can also swap content from inventory to chest and vice versa.



Let's explore the content of the **Chest** component.

Content

In this section of the component we reference the chest content and its Slots.

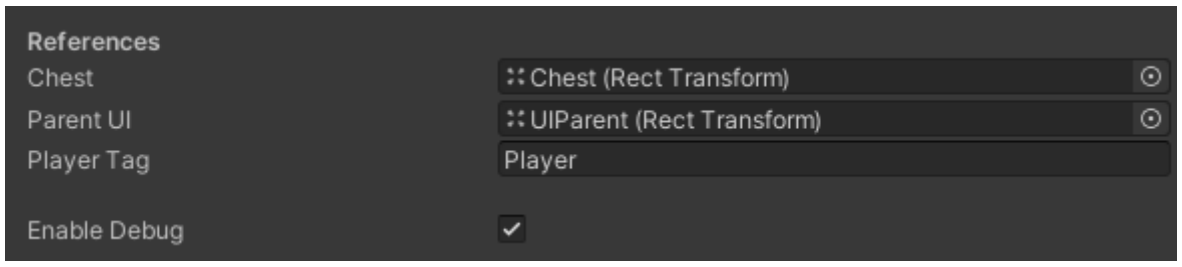


Its variables are:

- ***Chest Items***: the items assigned to the chest when the scene is launched. For each element we define:
    - ***Item Type:*** the Item Type that is added to the chest.
    - ***Item Num***: number of instances of that item.
    - ***Item Raw Durability:*** durability of each item, the value is not relevant if the item doesn't have durability (like in the screenshot above).
- ***Chest UI Slots:*** reference to all the Slots in the chest.

References and Debug

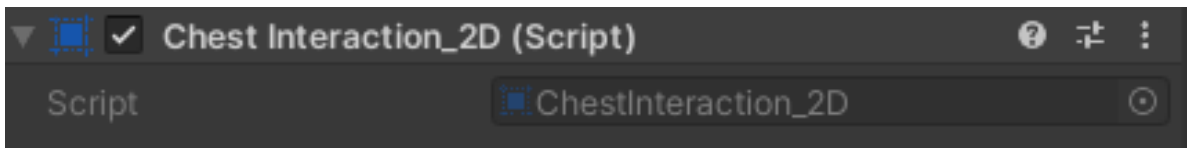
In this section we reference the chest UI parent as well as a couple of debug options



Its variables are:

- ***Chest:*** the parent of the UISlots.
- ***Parent UI:*** the parent of the chest UI.
- ***Player Tag:*** tag assigned to the player(s), the chest interaction is triggered when its collider touches a GameObject with that tag.
- ***Enable Debug:*** when it's on, it displays messages in the console for chest events.

To handle the ***Chest*** trigger events an additional script is required on the same GameObject as ***Chest***, ***ChestInteraction_2D*** for 2D triggers and ***ChestInteraction_3D*** for 3D triggers.



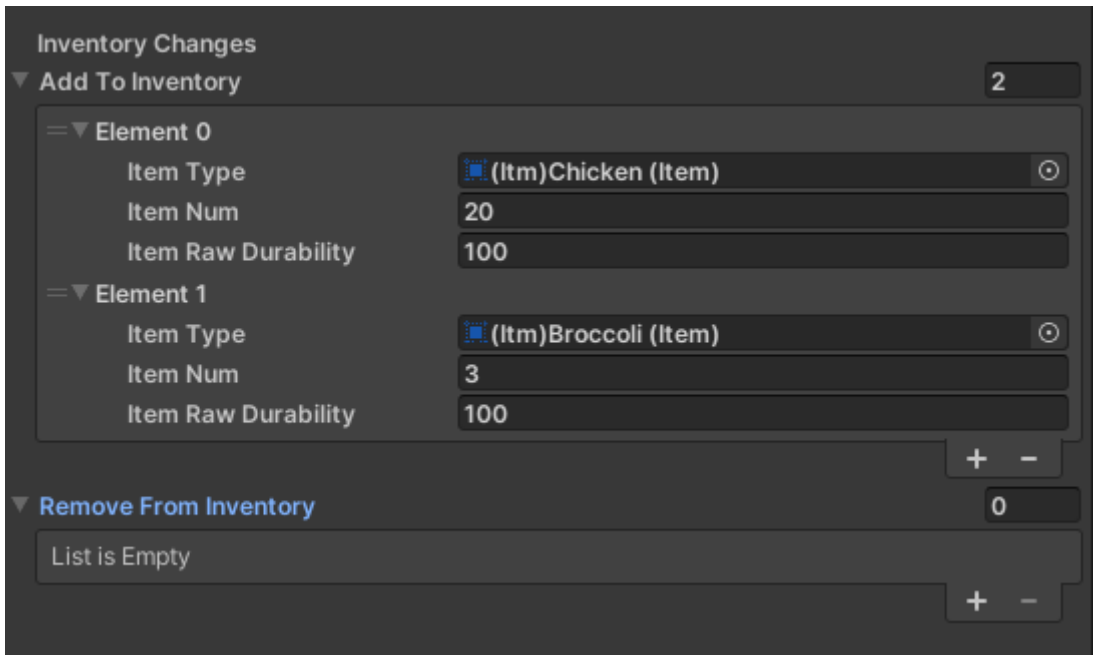
PickUp

A PickUp is a gameObject instantiated in the world that when interacts with the player (via collision) can add / remove items from their inventory.

Let's explore the content of the **PickUp** component.

Inventory Changes

In this section we list the item that we want to add and remove from the player inventory when an interaction occurs.

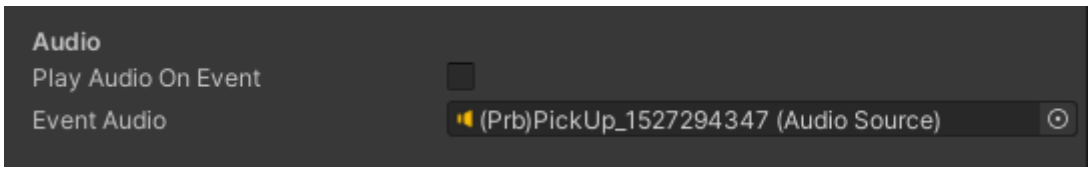


Its variables are:

- ***Add to Inventory***: the items that are added to the player inventory.
  - ***Item Type:*** the Item Type that is added to the inventory.
  - ***Item Num***: number of instances of that item.
  - ***Item Raw Durability:*** durability of each item, the value is not relevant if the item doesn't have durability (like in the screenshot above).
- ***Remove from Inventory:*** the items that are removed from the player inventory.
  - ***Item Type:*** the Item Type that is removed to the inventory.
  - ***Item Num***: number of instances of that item.
  - ***Item Raw Durability:*** durability of each item, the value is not relevant if the item doesn't have durability (like in the screenshot above).

Audio

In this section we list the audio that is triggered when an interaction occurs.

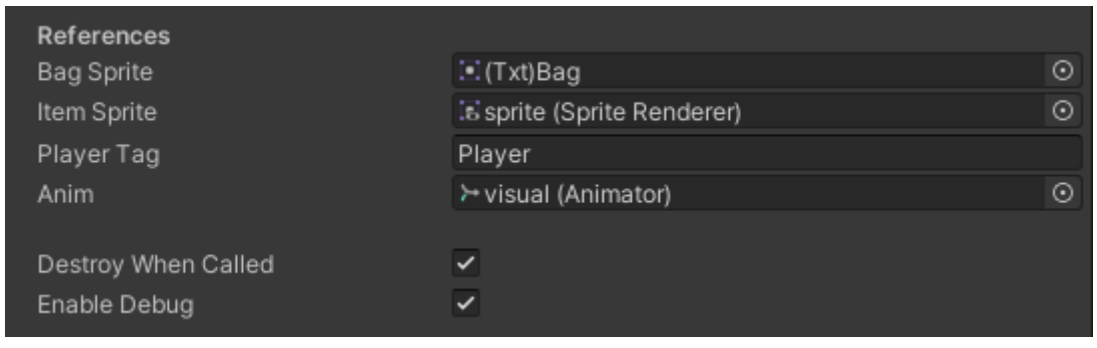


Its variables are:

- ***Play Audio On Event***: when the option is true, we reproduce an audio when an interaction occurs.
- ***Event Audio:*** reference of the SourceClip that will be reproduced when an interaction occurs (if the option above is true).

References and Debug

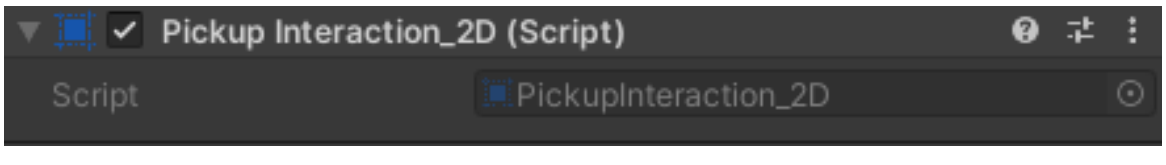
In this section we reference the UI used in the used in the pickup as well as a couple of debug options.



Its variables are:

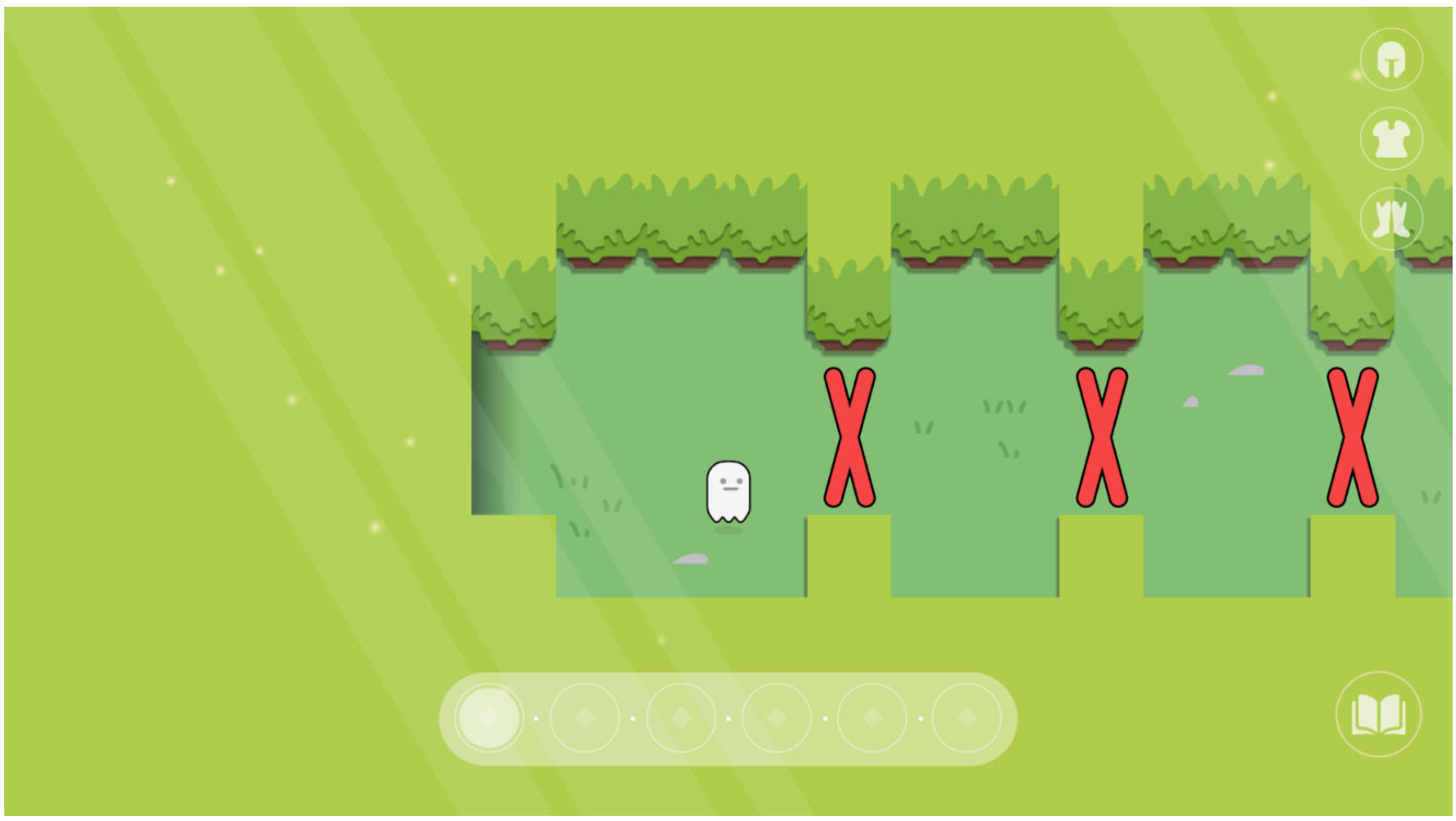
- ***Bag Sprite:*** the Sprite that is displayed when multiple items are dropped at the same time, otherwise if only an Item Type is dropped the PickUp will have that item sprite.
- ***Item Sprite:*** reference of the SpriteRenderer filled with the appropriate pickup image.
- ***Player Tag:*** tag assigned to the player(s), the pickup interaction is triggered when its collider touches a GameObject with that tag.
- ***Anim:*** the animator that handles the animation of despawn of the pickup.
- ***Destroy When Called:*** when the option is true the pickup instance is destroyed from the world when an interaction occurs.
- ***Enable Debug:*** when it's on, it displays messages in the console for pickup events.

To handle the ***PickUp*** trigger events an additional script is required on the same GameObject as ***PickUp***, ***PickUpInteraction_2D*** for 2D triggers and ***PickUpInteraction_3D*** for 3D triggers.



Area Trigger

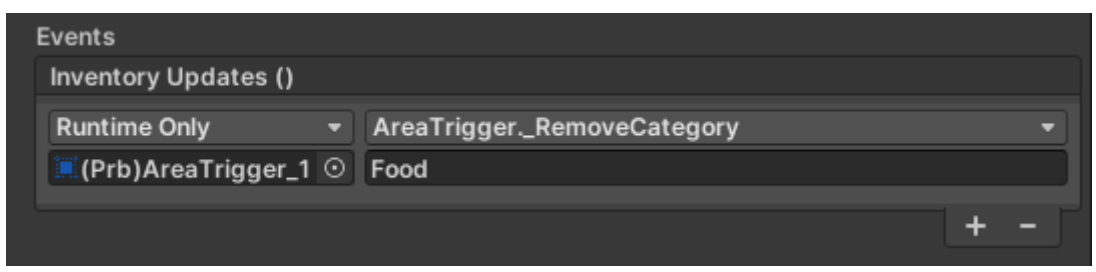
An AreaTrigger is a gameObject instantiated in the world that when interacts with the player (via collision) can call inventory methods. For example clearing a specific item category from the inventory, clear a specific item ID or clear the entire inventory.



Let's explore the content of the **AreaTrigger** component.

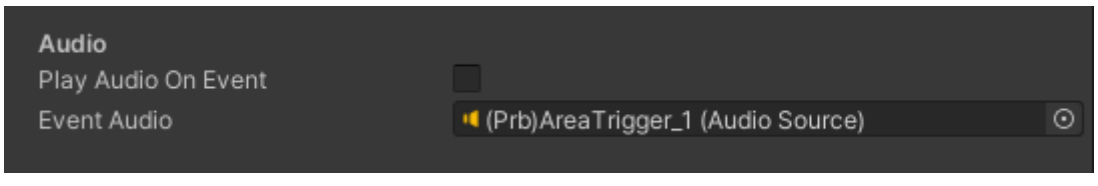
Events

This section of the component calls methods on interaction. The gameObject referenced in the Events is the AreaTrigger itself.



Audio

In this section we list the audio that is triggered when a interaction occurs with an AreaTrigger

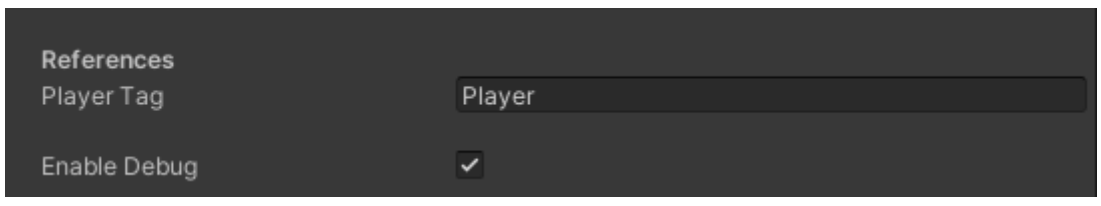


Its variables are:

- ***Play Audio On Event***: when the option is true, we reproduce an audio when an interaction occurs.
- ***Event Audio:*** reference of the SourceClip that will be reproduced when an interaction occurs (if the option above is true).

References and Debug

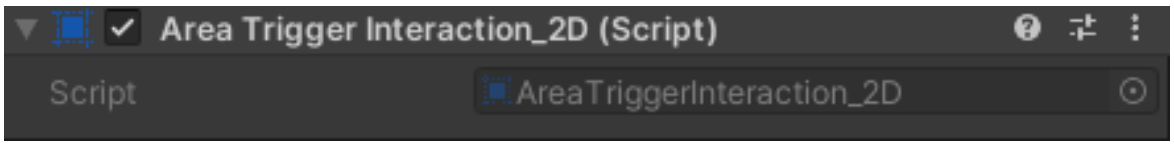
In this section we reference the UI used in the used for the AreaTrigger as well as a couple of debug options.



Its variables are:

- ***Player Tag:*** tag assigned to the player(s), the areaTrigger interaction is triggered when its collider touches a GameObject with that tag.
- ***Enable Debug:*** when it's on, it displays messages in the console for areTrigger events.

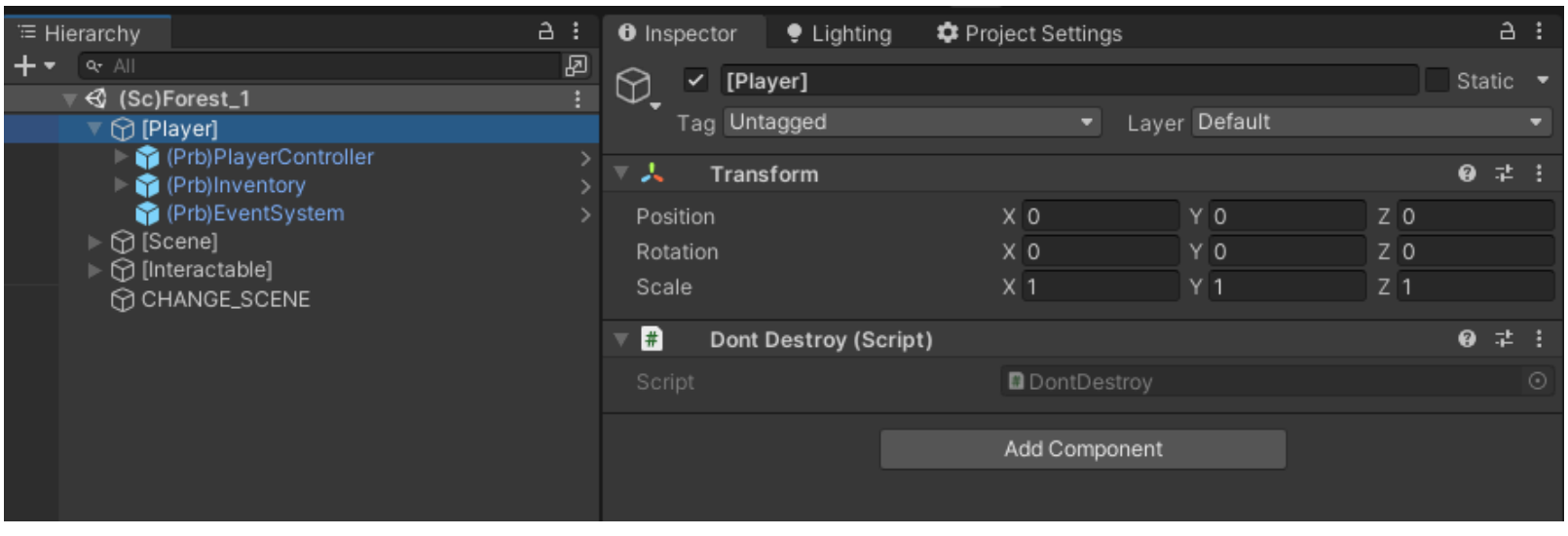
To handle the ***AreTrigger*** trigger events an additional script is required on the same GameObject as ***AreaTrigger***, ***AreTriggerInteraction_2D*** for 2D triggers and ***AreTriggerInteraction_3D*** for 3D triggers.



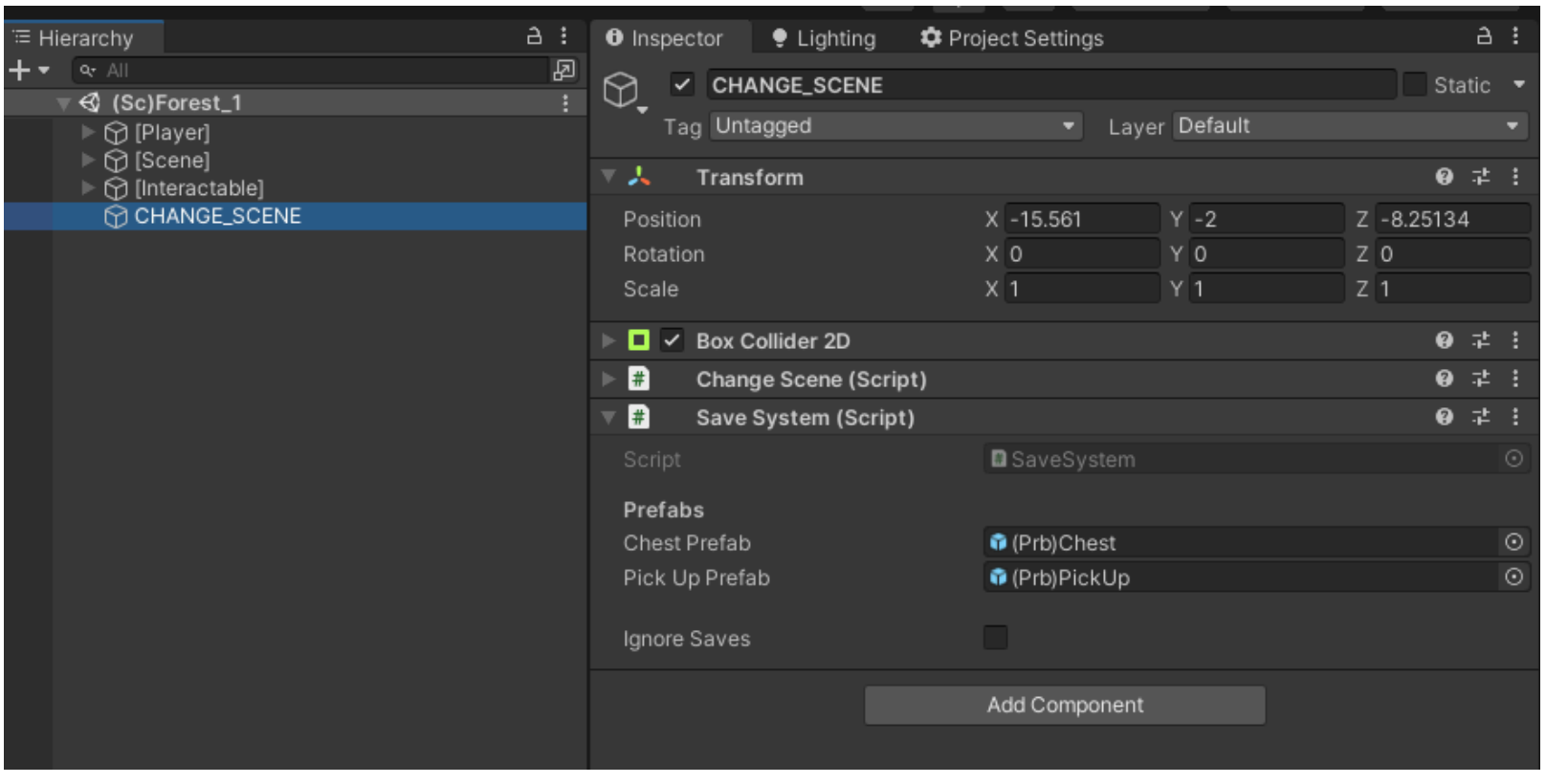
Additional Content - SaveSystem

The asset includes a simple SaveSystem that allows you to save the inventory content as well as chests and pickups when moving through scenes. As mentioned before this SaveSystem is studied to help you get started but a **custom solution that better fits your needs is highly recommended.**

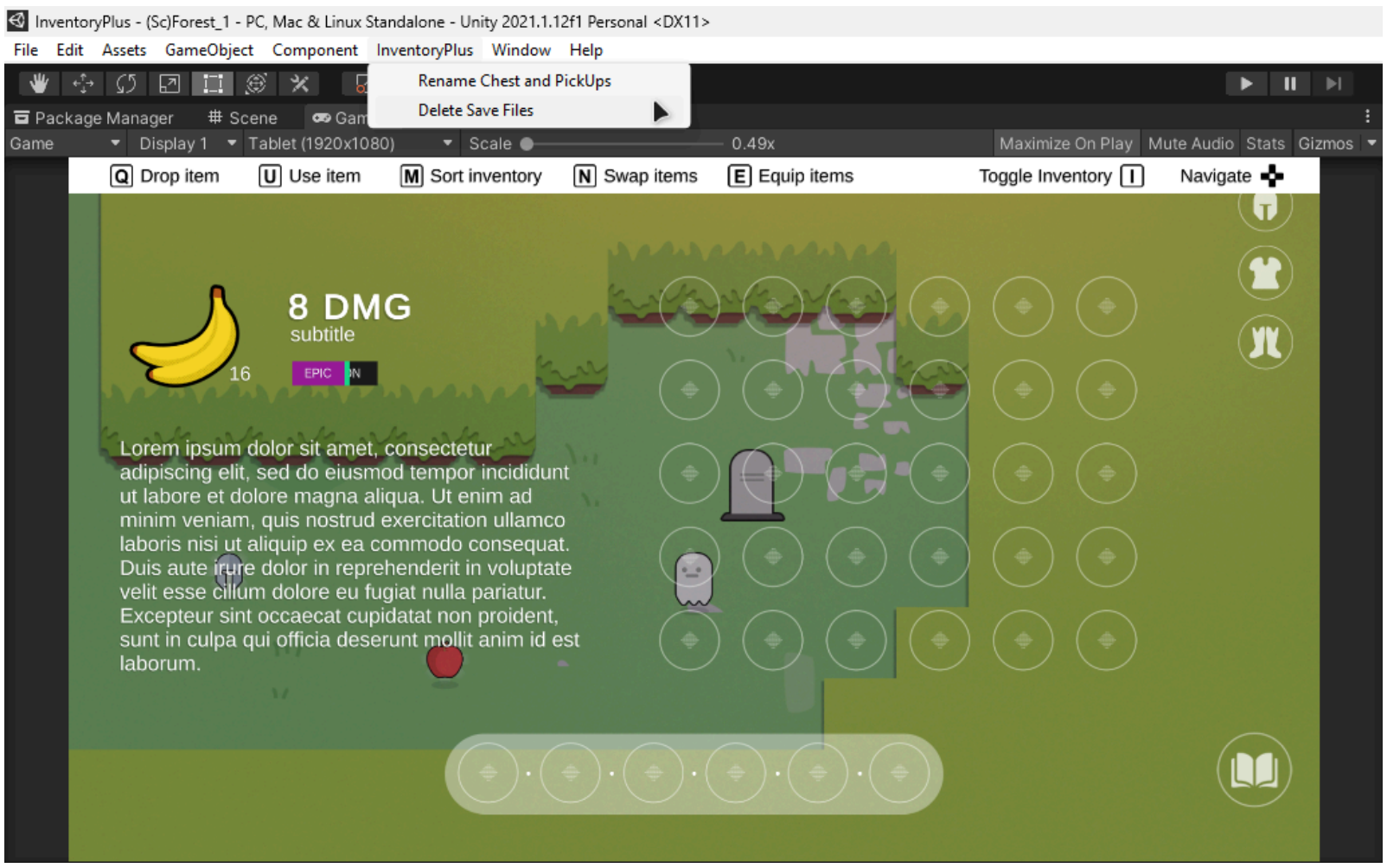
- ***Permanence of the Inventory:*** it's obtaining by making the parent of the inventory (as well as PlayerController and InputReader) not be destroyed on load.



- ***Creation of Saves:*** is obtained with the CHANGE_SCENE gameObject and its attached script SaveSystem.
    - When the scene is launched a **Load()** is called and various pickups and chests are instantiated according to the Saves (if they exist).
    - When a new scene is opened a **Save()** is performed.



You can decide if you want to use these Saves, ignore them or completely delete them via Menu. Via menu you can also assign a random ID to chest and Pickup, which is essential to perform a **Save()**



Contact

If you found this guide useful but need further help feel free to contact me at the email ***nappin.1bit@gmail.com***
P.S. A positive review of the asset would help a lot!

Cheers