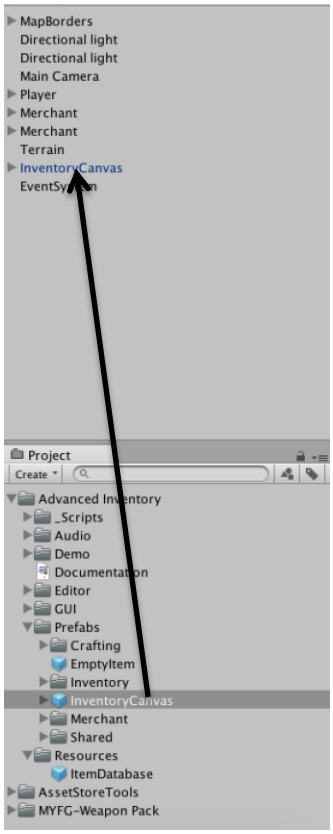
Advanced C# Inventory

Table of Contents

Setting up the inventory system

To set up the inventory you'll first need to drag the InventoryCanvas prefab into the hierarchy.



Now the inventory should be ready to use right off the bat, but you can customize it to suit your needs.

Inventory

For instance you can change the colors of the quality of the items. This can be done under the InventoryCanvas child called Inventory. This is used by all of the other scripts too.

You can also change the amount of slots in the inventory by altering the width and height variables of the inventory.

The **Drag Swap** option will determine whether or not the player can swap items in the inventory. Example: when the player drags an item and clicks on another item in the inventory they'll swap places, so that the one being dragged is added to the inventory and the item in the inventory is the one being dragged.

Auto Find Equipment Slot determines whether the items should find the appropriate slot by itself. Example: when the player drag and drop a shield and drops it onto the weapon slot the shield will automatically be equipped in the shield slot.

The **Right Click To Unequip** makes the player able to unequip equipped items by right clicking them. The equipped item will then return to the inventory.

Close If Merchant Open determines whether the inventory should also close when the merchant window is open.

Merchant

On the merchant you can also alter some functionality.

You can change the colors for when a tab is not selected and selected.

You can change the amount of slots, the size of the slots and also the size of the tab icons.

Remove Stackable Items When Bought will determine whether a stacked item should be removed once it's bought.

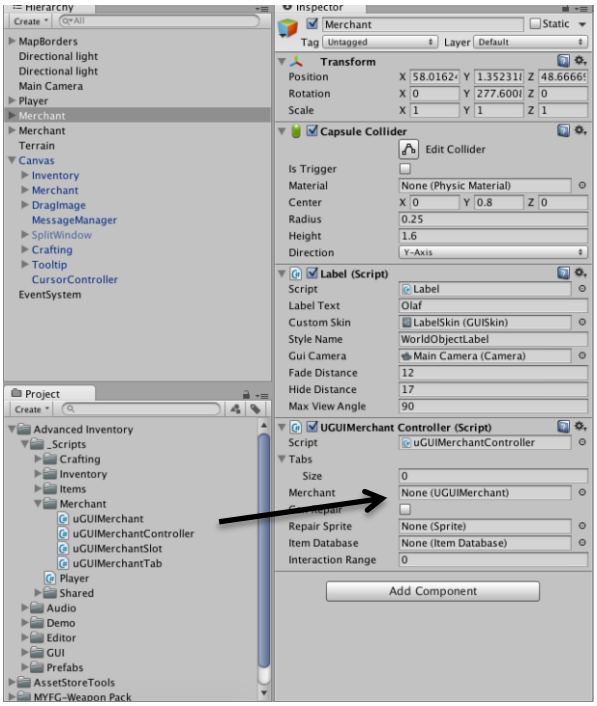
Tooltip

You can change the fade time on the tooltip. Fade time determines how long it takes for the tooltip to pop up. This is a useful feature if you don't want the tooltip to pop up immediately.

Creating merchants

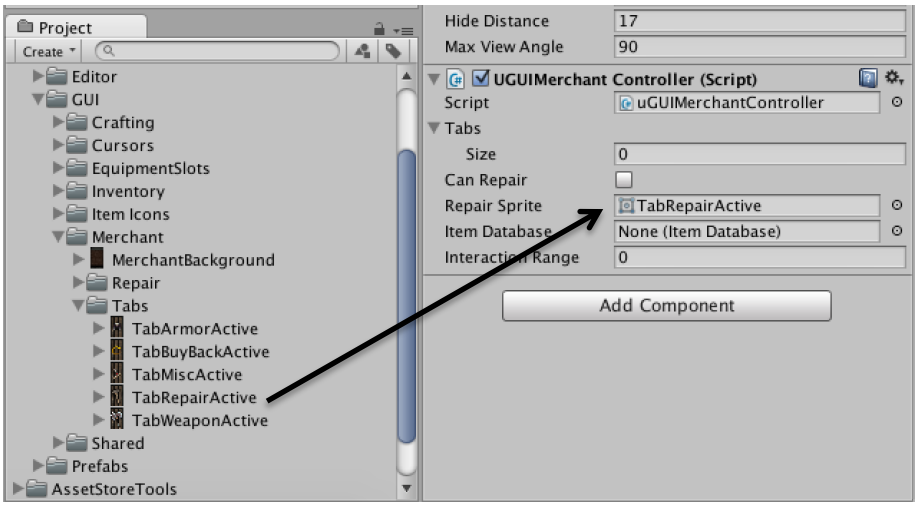
Let's create a merchant.

First you'll have to make a gameobject to be the merchant. This can be done by creating a cube or by dragging a model into the world. Once you've made the gameobject you'll have to drag the uguiMerchantController script onto the gameobject.

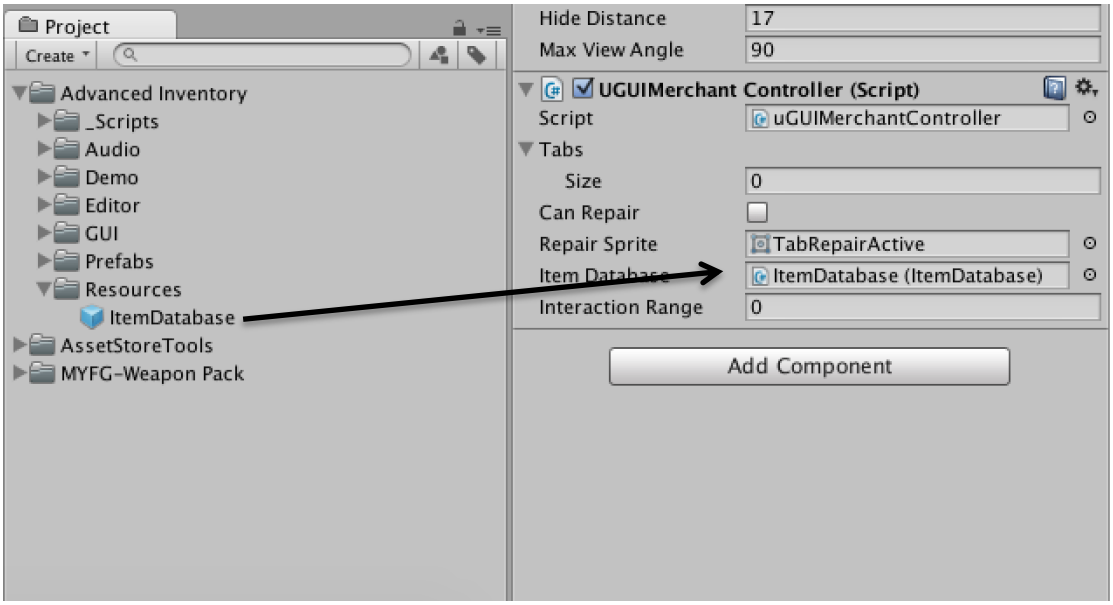


Now you have to decide what kind of items this merchant can sell. You'll have to option to have him sell weapons, armors and miscellaneous. Also remember to make a buyback tab for the merchant.
Now decide if the merchant has the ability to repair the player's gear. Simply tick the toggle Can Repair if you wish for the merchant to be able to repair the gear.

Now drag the repair tab sprite onto the repair sprite field.



Now drag the ItemDatabase prefab onto the Item Database field.



Now adjust the interaction range. This determines how far away the player can interact with the merchant. It is measured in unity units. (meters).

That's it. The merchant should now be up and running. To add items see below in the Manage merchant items section on how to add items to the merchant.

Manage merchant items

You can manage the items that a merchant sells by selecting the merchant in the hierarchy and then go to Window -> Manage Selected Merchant Items.

From here you'll be able to add new items and remove items you don't wish the merchant to sell.

Creating items

Creating items in and crafting items can be made using the item editor window. You'll find the editor window by choosing Window -> Manage Items.

Here you be met with a window that 4 tabs.

Create items.
Manage items.
Create crafted items.
Manage crafted items.

Creating items for inventory

First you'll need to select what type of item you're going to create. You'll be able to select between Weapon, Armor and other. Weapon is of course weapons and armors are of course armors, but other defines items like potions and scrolls etc. Click here to see how to make custom scripts for consumables.

Now you'll have to give the item a name. Consider don't using the same name for multiple items since it could give you some problems later on.

Item width and height determines how much space the item occupies in the inventory in the grid. For MMORPGs you'll probably want 1x1 sized items and for ARPGs you might want to use more complex sizes like 2x3, 1x3, 1x4, 2x4 and so on.

Consider using an icon with the same ratio is the item width and height that you made earlier else the icon is going to be distorted when placed in the inventory.

For MMORPGs you might want to use the same min and max for the stats so that the item stats doesn't vary when the player acquires the item. But if you want the stats to vary you can just use a different min and max stat.

Attack speed is defined as attacks per second. 1.5 will be 1.5 attacks per second. Description text is displayed at the bottom of the tooltip. So for instance if you want to display a text for a weapon you could write something like:

Forged in the deepest pits of hell this sword was wielded by the evil himself.

If you wish to use line break \n you'll have to write the text in a text editor and then copy paste the text in the description text field. This is because the unity editor doesn't understand \n as a line break like it does in MonoDevelop.

Manage items

Once an item is created it'll be added to the ItemDatabase prefab that stores all of the items. Furthermore it'll be assigned a unique ID that you can use to find the item in the database.

The manage items tab displays a list of all the items that's added to the database. When clicking on the name of the item you'll be able to edit all of the variables on the item for easy editing.

You can also remove an item from the list of items by clicking the red X. You'll be met with a dialog box to make sure you don't accidentally delete an item unless you want to delete it.

Creating crafting items

Crafting items uses IDs to determine what you want to create and what items are needed to craft the item. First you have to choose the resulting item from the dropdown.

Then you must choose the IDs of the crafting materials. In the list beneath the list of materials you can choose the amount of the given material that's needed to craft the item.

Next you must choose how long the item takes to craft and then how much it costs to craft the item.

Lastly you must choose what base type the resulting item is. This is to determine what tab the crafting item goes into in the crafting window.

Manage crafting items

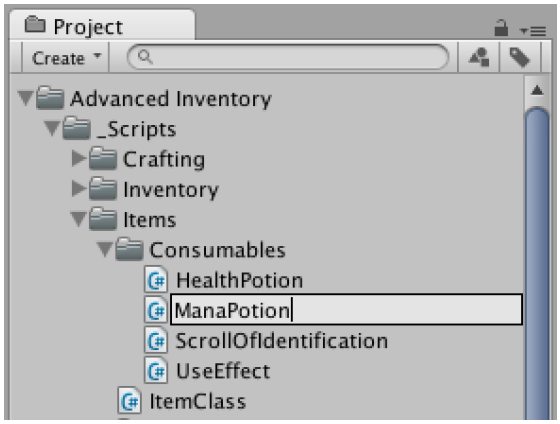
Same as the mange items tab in this tab you'll be able to manage the crafting items once they've been added to the list of crafting items for easy editing.

Custom consumables

Consumables use a custom script <UseEffect> as its base class. Whenever making a new consumable it must derive from this class.

Let's make an example of how to make a mana potion:

First let's create a new C# script called ManaPotion.cs



As said before this newly created script must derive from the UseEffect class.

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class ManaPotion : UseEffect {
5
6     // Use this for initialization
7     void Start () {
8
9     }
10
11    // Update is called once per frame
12    void Update () {
13
14    }
15 }
16
```

This will give us access to the item from which the effect script is called from, the player stats, the message manager and the inventory. All of these come from the UseEffect base class.

Along with the variables there's also a function on the base class called Use(). This is called whenever the player right clicks on the consumable. We're able to override the Use function of the base class to give it a whole new effect.

So in our ManaPotion script we want to delete the Start() and Update() functions and insert: public override string Use() {} so that the script now looks like this:

```
using UnityEngine;
using System.Collections;

public class ManaPotion : UseEffect {

    public override string Use() {

    }
}

```

Because we actually haven't got the references for the variables of the UseEffect script we must call the start function of the UseEffect script. This can be done by calling base.Start();

Now that we have the variables we need we can make the rest of our script.

We want to make sure that the player doesn't use any potion when the player already is at maximum mana we want to add a if statement to check for it.

```
using UnityEngine;
using System.Collections;

public class ManaPotion : UseEffect {

    public override string Use() {
        base.Start();

        if(player.curMana >= player.maxMana) {
            return "Already at maximum mana";
        }
    }
}

```

We return with a string saying "Already at maximum mana". This will be sent the to message manager and displayed as a warning to the player.

Now we can add the mana to the player.

```
using UnityEngine;
using System.Collections;

public class ManaPotion : UseEffect {

    public override string Use() {
        base.Start();

        if(player.curMana >= player.maxMana) {
            return "Already at maximum mana";
        }
        else {
            player.curMana += item.manaAmount;
        }
    }
}
```

But we also must make sure that the mana doesn't go above the player's maximum mana.

```
using UnityEngine;
using System.Collections;

public class ManaPotion : UseEffect {

    public override string Use() {
        base.Start();

        if(player.curMana >= player.maxMana) {
            return "Already at maximum mana";
        }
        else {
            player.curMana += item.manaAmount;

            if(player.curMana > player.maxMana) {
                player.curMana = player.maxMana;
            }
        }
    }
}

```

Now all that's left is to tell the inventory that a potion has been used. We do that by returning "Potion". When this is returned to the inventory 1 potion will be removed from the stack of potions.

```
using UnityEngine;
using System.Collections;

public class ManaPotion : UseEffect {

    public override string Use() {
        base.Start();

        if(player.curMana >= player.maxMana) {
            return "Already at maximum mana";
        }
        else {
            player.curMana += item.manaAmount;

            if(player.curMana > player.maxMana) {
                player.curMana = player.maxMana;
            }

            return "Potion";
        }
    }
}

```

And that's how you can create a potion.

To add this effect to an item you must add the script to the item when creating it. Open up the Create Items window and under "Item Type" select other. Now select type potion under the Consumable Type. Now you'll see a field called Use Effect Script Name. Here you'll have to write: ManaPotion. Now the inventory knows what script to call when the item is clicked.

Player variables

For everything to function correctly you'll need at least three variables for your player (or you can expand upon the included player script).

canDualWield;
canDualWieldTwoHanded;
canWieldTwoHanded;

You'll need the first variable to determine whether your character can dual wield weapons like a rogue using two daggers or a warrior using two swords.

The second variable determines whether the player can dual wield 2 two-handed weapon. Just like warrior talent Titan Grip gives the warrior the ability to dual-wield 2 two-handed weapons.

The third variable determines whether the player can use two-handed weapons at all. Like a rogue might not be able to equip two-handed weapons.

The mana and health variables are only used by the inventory when the player is using potions.

Last notes

There's still some features that you can customize on the other parts of the inventory, but they're very similar to the one already mentioned and should be somewhat straight forward, but if you need further explanation please feel free to contact me, and I'll do my best to answer your questions!

If you require support or have any questions, suggestions or just want to say hi please contact me at Asgerroed@me.com.