Semra Usanmaz

Programming Tutorial: Making an inventory in Unity

NOTE: This is not part of the four tutorials, it is a tutorial I worked on but couldn't finish – it is mentioned in the learning journal)

How to make an inventory in Unity:

1. Set up the scene

Create a new scene and name it "Inventory".

Add a 'Canvas' to the scene by right-clicking in the Hierarchy, hovering over '*UI*', and selecting 'Canvas'. Name this Canvas "theInventory".

Double click on the newly placed Canvas to put it in full view (or you can press F which will do the same thing).

Add a 'Panel' to the canvas and decrease the size to a size that you think would be good for an inventory. (Ensure that if you want the inventory to be exactly in the middle you will need to move it until you see the middle of the panel have a blue line running through it).

Rename the new panel "*Inventory*".

2. Enabling the inventory panel in-game using both/either a specific key-press or a button-press.

Add a new script and name it "inventoryScript", drag this script onto the Canvas named "theInventory".

You will need to first of all get the Canvas component of the Canvas, to do this, type

"Canvas canvas;"

just above the Start function of the script and then type

"canvas = GetComponent<Canvas>();"

into the Start function. Under that, type

"canvas.enabled = false;"

(make sure the c is in lower case), this ensures that when the game starts, the Canvas is turned off.

Key press to enable:

If you would like to enable the inventory by pressing a random key on the keyboard e.g. the "I" key, read below. If you would like to enable it through button press, skip to the "Button press to enable" section of this tutorial.

In the Update function of the script, type in:

```
if(Input.GetKeyDown(KeyCode.I) && canvas.enabled == false)
    {

        canvas.enabled = true;

    }
```

This will enable the “theInventory” canvas once the I button is pressed in-game, only if the canvas is already disabled.

After this you would also probably want the same key to disable the Canvas when pressed again.

To do this, type:

```
else

if(Input.GetKeyDown(Keycode.I) && canvas.enabled == true)

{

canvas.enabled = false;

}
```

This checks whether the “I” key has been pressed and also whether the canvas is already enabled or not, then disables it according to if the statements within the if() brackets are true.

You can test this out in the Game view.

Button press to enable:

If you would prefer the player press a button to enable the inventory –

Create another Canvas and name this one “UI”.

Create a button on the new Canvas by right clicking it, hovering over “UI” and selecting “Button”.

Rescale the button to a size you prefer and change the text in it to “I” for inventory or just “Inventory” if your button is long enough to fit it in.

Create a new script and name it “inventoryButton”, drag this script into the canvas named “UI” and open this script.

Remove the ‘Start’ and ‘Update’ functions (void Start, void Update) as they are not needed and create a new function named buttonEnable by typing:

```
public void buttonEnable ()

{

}
```

This will be linked to the button and once the button is clicked, whatever is within the two function brackets will be done.

To actually get the function to work, you must type in a command, in this case the command is to enable the canvas/disable it.

Firstly, you must make the script link to the canvas that you want to enable/disable. To do this, type “public Canvas Inventory” just above the newly created function. This will create a box which you should click and drag the “theInventory” Canvas into (back in the project scene) after saving so that the script knows it is supposed to interact with that particular object.

After this go back to the script and type within the function brackets:

```
 if(Inventory.enabled == false )
      {
         Inventory.enabled = true;
      }
      else
         if(Inventory.enabled == true)
      {
         Inventory.enabled = false;
      }
```

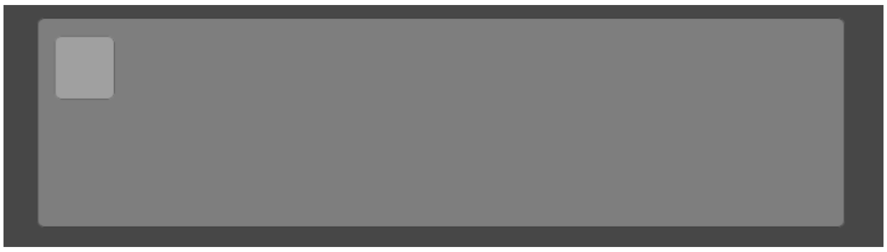
This will ensure that if the Inventory canvas is disabled, once you press the button it will be enabled, and when it is enabled, a button press will disable it.

Finally, you will need to make the button react to the script. To do this, click the button then go to the "On Click ()" section in the Inspector and press the "+" button. Drag the "UI" canvas into the box and in the drop-down list on the button next to the button labelled "Runtime" select the "inventoryButton" script and the function "buttonEnable()".

Test the button in Game view.

3. Inventory Slots

Go to your "theInventory" canvas and add another panel as the child of the "Inventory" panel, this will be one of the inventory slots so rename it "inventorySlot". Resize this to a much smaller size than the Inventory panel and place it in the top left corner, but not too close to the sides (as shown in the image below).



Once that is done, add a button component to the inventory slot.

Duplicate the inventory slot by pressing CTRL-D as many times as you need. After this, go to the Inventory panel and add a new component called "Grid Layout Group", this will place each of the inventory slots in a grid format with equal spacing - if you prefer to not have them in a grid format or if you want to take the long route, you can move each slot yourself around the panel.

Once they are in the grid format, you can adjust their scale and spacing within the Inspector so that they fit well in the Inventory panel.

4. Adding items to the inventory slots

Firstly, create a new script, name it "Item" and open it. Change the "MonoBehaviour" part of the script to "ScriptableObject" which will allow you to not have to place the script on a gameobject and instead derive it from scriptable objects. Also remove both the Start and Update functions as they are not needed.

Then type in between the brackets “public string itemName;” for the name of the items and “public sprite Icon” for the icons of the sprites that will be displayed in the inventory. You will need to add a new class above the public class area – the area where the “using UnityEngine;” part is called “[CreateAssetMenu]”. This will allow you to create items by right-clicking in the project area.

Go back to the project and create a new folder named “Items” then right click in the project area and create a new “Item” and name it “An Item” – place it into the “Items” folder, add a sprite to the item.

Create a new script called “inventorySlots” and attach this script to every ‘inventorySlot’ and open the script. Add a reference to the newly created ‘Items’ by writing “public Item Items;” and a reference to the images by typing “[SerializeField] Image Images;” above the Start function and remember to add “Using UnityEngine.UI;” just below “Using UnityEngine;” otherwise unity cannot access the Image class. Also remove the Start and Update functions once again. [SerializeField] was used instead of public for the Images script line so that the Images are protected and cannot be changed by other codes but will still show up in the inspector.

In order to save time and not have to manually input every image, you can add an “OnValidate” script to the objects which will look like:

```
private void OnValidate()

{

}
```

And within the brackets write:

```
If(Images == null)

{

Images = GetComponent<Image>();

}
```

This code will ensure that if the item does not have an image manually placed, the image component of it will be found automatically.

Now the previously created “public Item Items;” script will need to be changed into a property instead so that you can make the system change/disable the icon depending on whether an item is active or not. Change the mentioned script to:

```
private Item _Items;
  public Item Items

  {
    get {return _Items; }
    set
    {
      _Items = value;
      if (_Items == null)
      {
        Images.enabled = false;
      } else
```

```
        {
            Images.sprite = _Items.Icon;
            Images.enabled = true;
        }
    }
}
```

The script essentially states that if there is no item, the image is disabled. But if there is an item, the image is enabled.

Now go back to the 'inventoryScript' and add three new lines of script within the upper part of the public class brackets. These include, a list for the items, a transform for the 'Inventory' panel (also known as the parent of the item slots) and an array of item slots. Write them with [SerializeField] rather than public much like this:

```
[SerializeField] List<Item> theItems;

[SerializeField] Transform inventoryPanel;

[SerializeField] inventorySlots[] itemSlots;
```

Create another 'OnValidate' function below the above lines to help fill in the array of itemSlots automatically by grabbing them from the inventoryPanel object. The code for this is:

```
private void OnValidate()
    {
        if(inventoryPanel != null)
        {
            itemSlots = inventoryPanel.GetComponentsInChildren<inventorySlots>();

        }
    }
```

You can check if this works by saving, going back to the project, and dragging the 'Inventory' Panel into the new box made for the panel which should be in the 'theInventory' canvas where the used script was placed. And all your inventory slots should automatically show up in the inspector.

In order to update the UI when a new item is placed and to give it information on what to do if there is no item in a particular place, you will need to create a new function below the above script called:

```
public void UpdateUI()

{

}
```

Within the brackets write:

```
  int i = 0;
  for(; i < theItems.Count && i < itemSlots.Length; i++)
{
 itemSlots[i].Items = theItems[i];
}
```

This script assigns every item into an inventory slot.

Below this write:

```
  for (; i < itemSlots.Length; i++)
{
 itemSlots[i].Items = null;
}
```

This ensures that if there is no item in an item slot, the slot is null.

After this, call the UpdateUI function by adding “UpdateUI();” within the ‘OnValidate’ function’s brackets. If you go back to the project after this the image components of the inventory slots should be turned off, as they do not have items in them.

Now you can add in your items that you created by right clicking in project and selecting items by dragging them into the ‘The Items’ section which is just above the area where the inventory slots were listed earlier – in the hierarchy of the ‘theInventory’ canvas. Add a sprite for the icon of each item and they should be visible in the inventory once you click play.

5. Equipping items into the inventory

You will first need the system to check whether the inventory is full or not before allowing more items in. To do this, create a bool under everything called AddItem(item theItems), it will look like:

```
public bool AddItem (Item theItems)

{

}
```

Within the brackets write:

```
if (IsFull())

{

return false;

}
```

This will deny the given command if the inventory is full.

Under “return false;”, add:

```
Item.Add(theItems);
UpdateUI();
return true;
```

This will add the item to the list if the inventory isn’t full.

At the moment most of the commands within that code show up as wrong, this is because they must still be created.

To check whether the inventory is full or not, you will need to create the ‘IsFull’ command, so create a new bool called “public bool IsFull()” and within the brackets write:

```
return theItems.Count >= itemSlots.Length;
```

What this does is that it checks whether the number of items in the inventory is larger than the amount of item slots or not.

After this, create a bool for the removal of items – "public bool RemoveItem(Item theItems)" and within those brackets write:

```
If(Item.Remove(theItems))

{

UpdateUI();

return true;


}

return false;
```

This will remove an item from the inventory, only if the 'Item.Remove' function is called.

Finally, add a 3D cube or sphere into the scene and add a collider to it with a trigger.