Demo: Inventory

Material in this demo courtesy of

Unity Inventory System Tutorial (2022) by GameAssetWorld https://youtu.be/PdSLNpeTRTA

I renamed a few things; I say "AmmoPickup" instead of "ManaGlobe," and "HealthIcon" instead of "HealthGlobeButton."

Many games allow players to collect and carry around a large number of items. An Inventory is a table of elements that provides quick access to player items and a simple way to organize them.

Hands-On

Let's implement an Inventory in Unity.

Setting the Scene

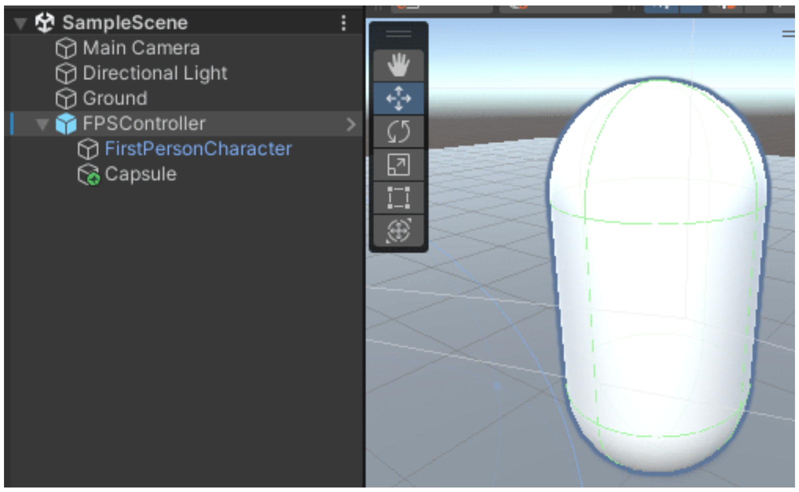
1. Make a new 3D project in Unity.

2. It is recommended that to stay organized you **make folders** for Prefabs, Materials, and Scripts.

3. Have a **Ground** to walk on – try creating a Plane and rename it "Ground," Position 0,0,0; Scale x=3 y=3 z=3, and give it a colored material and a Mesh Collider.

4. Have a Player that walks around. I used the **FPS Controller** from Hour 6 of the textbook *Sams Teach Yourself Unity Game Development in 24 Hours, 4th edition*. Starting files are located at http://fixbyproximity.com/Downloads/4th_Edition/UnityBook.html. (I had problems clicking the links, and had to right-click each link and choose "Save Link as", then when Chrome told me the link was not secure, I had to choose "Keep" so it finished the download).

   a. Because I wanted to see a physical representation of my player, I created a Capsule (GameObject -> 3D Object -> Capsule) and dragged it into the FPSController to become a child object of it (see screen shot). After that I gave the Capsule the Position x=0 y=0 z=2 to put it in front of the Camera, and made sure it had a Collider and Rigidbody. If it tends to fall over as you move, freeze its Rigidbody axes.



5. Select the FPSController (or whatever your Player is) and in the Inspector set its **Tag = "Player."** (In my case, instead I gave that Tag to the Capsule within).

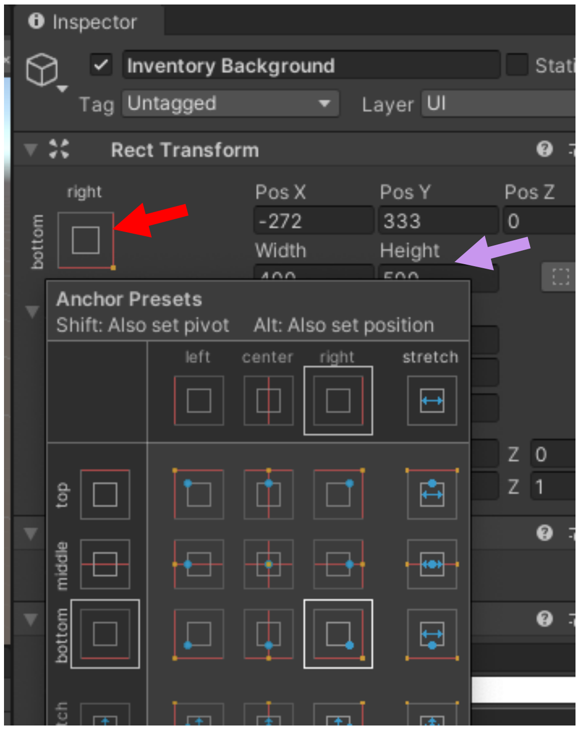
6. I also had to disable the Audio on the Main Camera, since the FPSController had Audio.

The Inventory UI

7. Create a new **UI** -> **Image** and name it "**Inventory Background**." It appears in the Hierarchy as a child of a Canvas object.

8. Select the Canvas and in the Inspector find the "Canvas Scaler" area. There, set "**UI Scale Mode" = "Scale with Screen Size**" and set "**Reference Resolution" = x 1920, y 1080**.

9. **Position the Inventory where you wish**: Zoom and pan in the Scene until you can see the Canvas, and drag the white square **Inventory Background** Image near the **bottom right** corner of the canvas rectangle. In the Inspector, find the "Rect Transform" area. There, click to top left square graphic (see **red arrow** in the screen shot) for Anchor Presets, and anchor it to the **bottom** and **right**. Afterward, set Width to **400** and Height to **500** (see **purple arrow** in the screen shot).

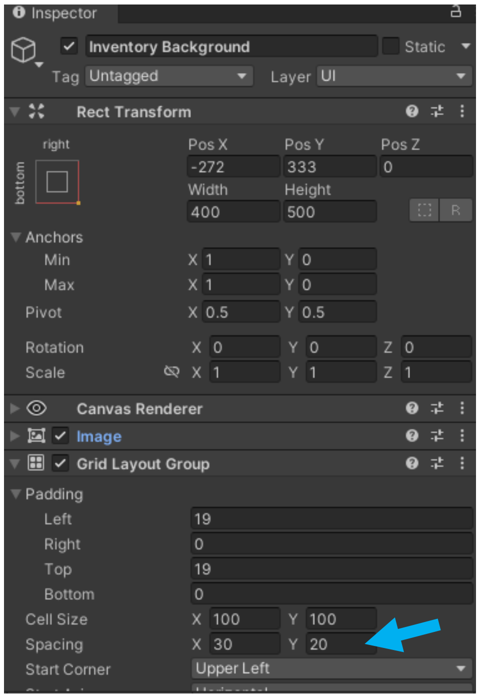


10. In the Inspector, give the Inventory Background a new Component: search for "**Grid Layout Group**" and add it. This will cause all the slots we add to arrange nicely.

11. In the Hierarchy right click "Inventory Background" and create a UI -> **Image** and name it "**Slot (1)"** and be sure it appears in the Hierarchy as a child of Inventory Background.

12. Position the Slot near the upper-left corner of Inventory Background. Give it a medium-gray color.

13. In the Hierarchy, temporarily copy Slot (1) to make a total of 12 Slots. Select the Inventory Background and adjust its **Spacing** properties until the slots are arranged nicely in 4 rows of 3 Slots per row (see settings near the **blue arrow** in the screen shot). Below that, also set **Child Alignment = "Middle Center."** Then delete the 11 copies because we will make Slot 1 a Prefab later.

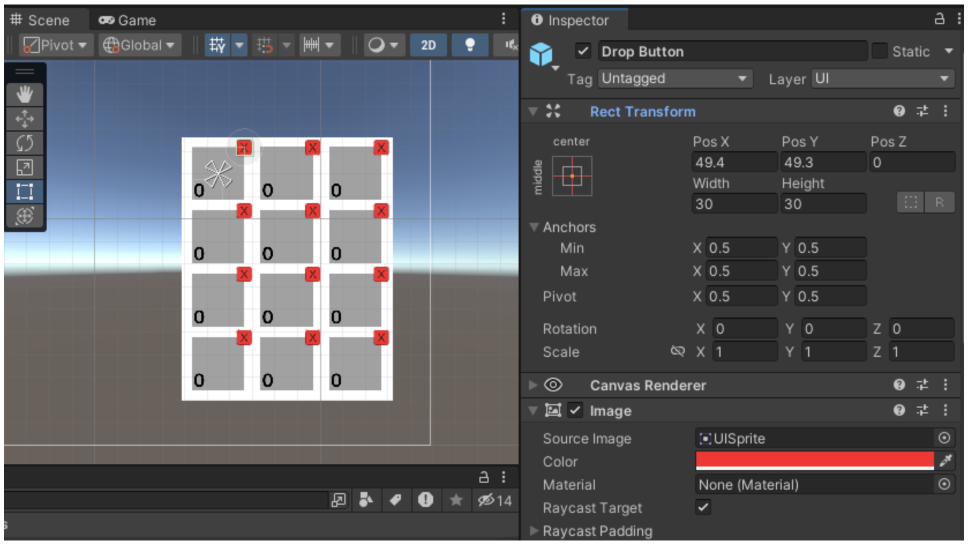

Inspector
Inventory Background — Static
Tag Untagged — Layer UI
Rect Transform
right / bottom
Pos X -272, Pos Y 333, Pos Z 0
Width 400, Height 500
Anchors
Min X 1, Y 0
Max X 1, Y 0
Pivot X 0.5, Y 0.5
Rotation X 0, Y 0, Z 0
Scale X 1, Y 1, Z 1
Canvas Renderer
Image
Grid Layout Group
Padding
Left 19
Right 0
Top 19
Bottom 0
Cell Size X 100, Y 100
Spacing X 30, Y 20
Start Corner Upper Left

14. Next, give the Slot a text display for Amount of Items: In the Hierarchy right click "Slot (1)" and create a UI -> **Text - TextMeshPro** and name it "**Amount"** and be sure it appears in the Hierarchy as a child of Slot (1).

15. Select the Amount and in the Inspector find the “Main Settings” area. There, set “**Vertex Color**” = black and beside “**Text Style**” click B (bold). Above that, in the big box under **TextMeshPro – Text (UI)**, type a zero so that number appears in the field to start with. Drag it to the bottom-left corner inside the gray Slot.

16. Next, give the Slot a drop button: In the Hierarchy right click "Slot (1)" and create a UI -> **Button-TextMeshPro** and name it "**Drop Button"** and be sure it appears in the Hierarchy as the second child of Slot (1).

17. Select the Drop Button and in the Inspector give it **Width = 30** and **Height = 30**. Below, under “Image”, set **Color = red**. Drag it to the upper-right **corner** of the gray Slot. (The screen shot here shows what the inventory grid will look like once all 12 slots are present in step 21.)



Drop Button ✓ Static
Tag Untagged Layer UI
Rect Transform
center / middle
Pos X 49.4 Pos Y 49.3 Pos Z 0
Width 30 Height 30
Anchors
Min X 0.5 Y 0.5
Max X 0.5 Y 0.5
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Image ✓
Source Image UISprite
Color 🟥
Material None (Material)
Raycast Target ✓

18. In the Hierarchy, select the **Text** that’s a child of the Drop Button and in the Inspector in the big box under **TextMeshPro – Text (UI)**, type an “X” as a symbol that clicking it will remove the item.

19. Create a **script named Slot** and attach the script to Slot (1). We won’t add code to it yet.

20. In the Hierarchy, drag **Slot (1)** into the Project panel to become a **Prefab**.

21. In the Hierarchy, **copy and paste the Slot (1)** instance until there are 12 Slots inside the Inventory Background.

The Controller

22. Create a new empty GameObject and name it "**Controllers**"

23. Create a new empty GameObject and name it "**InventoryController**" then drag it into the Controllers Empty Object to become a child object of Controllers.

24. Create a **script named InventoryController** and attach the script to your "**InventoryController**" Object.

25. Inside the script use the code shown here:

The InventoryController.cs script (attached to InventoryController Empty Game Object)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class InventoryController : MonoBehaviour
{
	//This array will contain a true or false value for each of the 12 slots in our array
to keep track if each is full
	public bool[] isFull;
	//This array will contain the 12 slots that are in our array
	public GameObject[] slots;
}
```

26. In the Hierarchy, select the InventoryController and in the Inspector find where the Script is attached and notice some properties need to be filled. At the top of the Inspector click the lock so the InventoryController properties remain visible. Shift-click in the Hierarchy to select **all 12 Slots, then drag them over onto the word "Slots"** in the Inspector where the Script is attached. They now appear in the Inspector. Type a **12** to the right of the phrase **"Is Full"** in the Inspector where the Script is attached. At the top of the Inspector click the lock again so the InventoryController properties are no longer locked in view.

The Pickup Items

27. Your example pickup items could be anything, but for now create a new Cube or other primitive and name it "**HealthPickup**". Give it a red Material. Be sure it has a Collider with "**is Trigger**" checked.

28. And create a new Cube or other primitive and name it "**AmmoPickup**". Give it a blue Material. Be sure it has a Collider with "**is Trigger**" checked.

29. Create a **script named Pickup** and attach the script to both the pickup objects. We won't add code yet.

30. In the Hierarchy, drag each of the pickup objects into the Project panel to become a **Prefab**.

31. Next, make a pickup icon for the Slot: In the Hierarchy right click "Slot (1)" and create a UI -> **Image** and name it "**HealthIcon"** and be sure it appears in the Hierarchy as the third child of Slot (1).

32. Select the **HealthIcon** and in the Inspector give it **Width = 50** and **Height = 50**. Below, under "Image", set **Color = red**. Leave it centered inside the gray Slot. Normally, you'd use an image of the 3D object, not a red box.

33. [The instructions I was following said to take this step but it seems unnecessary: Select the **HealthIcon** and in the Inspector add a Component: Button.]

34. Create a **script named Spawn** (which eventually will be coded to create a 3D pickup when the icon is dropped from inventory) and attach the script to the **HealthIcon** object. We won't add code yet.

35. In the Hierarchy, drag the **HealthIcon** object into the Project panel to become a **Prefab**.

36. Next, convert the HealthIcon in the Hierarchy into another pickup item: In the Hierarchy, right-click the HealthIcon and choose **Prefab** > **Unpack** so it's disconnected from its master prefab. Rename the HealthIcon in the Hierarchy "**AmmoIcon**". In the Inspector under "Image", set Color = blue.

37. In the Hierarchy, drag the **AmmoIcon** object into the Project panel to become a **Prefab**.

38. In the Hierarchy, delete the **AmmoIcon** object. The Hierarchy should now have neither HealthPickup nor AmmoPickup; these icons will get added by code when something goes into the inventory.

39. Inside the **Slot** script use the code shown here:

The Slot.cs script (attached to the Slot Prefab)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class Slot : MonoBehaviour
{
         public InventoryController inventory;
         public int i;
         public TextMeshProUGUI amountText;
         public int amount;

         void Start()
         {
                  inventory = FindObjectOfType<InventoryController>();
         }
         void Update()
         {
                  // have this slot display the amount
                  amountText.text = amount.ToString();

                  //once the amount is zero, hide the text box, otherwise display it.
                  if (amount > 1)
                  {
                           transform.GetChild(0).GetComponent<TextMeshProUGUI>().enabled = true;
                  }
                  else
                  {
                           transform.GetChild(0).GetComponent<TextMeshProUGUI>().enabled = false;
                  }
                  //if this slot does not have an icon as third child, register that it's no longer full
                  if (transform.childCount == 2)
                  {
                           inventory.isFull[i] = false;
                  }
             }

         public void DropItem()
         {
                  if (amount > 1) // If more than one are "stacked" in the same slot
                  {
                           amount -= 1;
                           // make the Spawn script (attached to the Icon) create a 3D instance of the object in the game
                  environment.
                           transform.GetComponentInChildren<Spawn>().SpawnDroppedItem();
                  }
                  else if (amount == 1)
                  {
                           amount -= 1;
                           transform.GetComponentInChildren<Spawn>().SpawnDroppedItem();//gets a red underline, but that's OK
                           //destroy the game object (Icon) to which the Spawn script is attached
                           GameObject.Destroy(transform.GetComponentInChildren<Spawn>().gameObject);
                  }
         }
}
```

40. Inside the **Pickup** script use the code shown here:

The Pickup.cs script (attached to each Pickup Item's Prefab)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Pickup : MonoBehaviour
{
	private InventoryController inventory;
	public GameObject itemIcon;
	public string itemName;

	void Start()
	{
		inventory = FindObjectOfType<InventoryController>();
	}

	private void OnTriggerEnter(Collider other)
	{
		if(other.CompareTag("Player"))
		{
			for (int i = 0; i < inventory.slots.Length; i++)
		{

		//Here, we set '2' as the max # of the same item allowed in a slot. Additional
	ones will spill over into a new slot
		if(inventory.isFull[i] == true &&
	inventory.slots[i].transform.GetComponent<Slot>().amount < 2)
		{
		//if another item of the same kind is already in this slot
			if (itemName ==
		inventory.slots[i].transform.GetComponentInChildren<Spawn>().itemName)
			{
				Destroy(gameObject); //destroy this 3d representation from the game
			environment
				inventory.slots[i].GetComponent<Slot>().amount += 1;
				break;
			}
		}
		// otherwise put one of this item into the slot
		else if(inventory.isFull[i] == false)
		{
			inventory.isFull[i] = true;
			Instantiate(itemIcon, inventory.slots[i].transform, false);
			inventory.slots[i].GetComponent<Slot>().amount += 1;
			Destroy(gameObject);
			break;
		}
	}
}
```

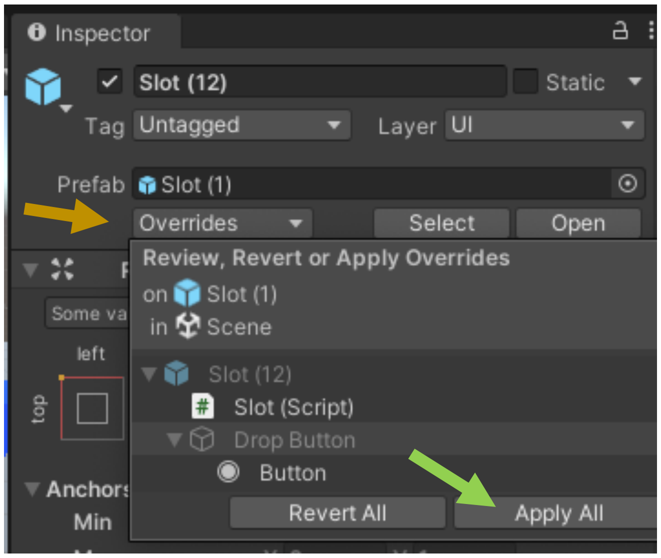
41. Inside the **Spawn** script use the code shown here:

The Spawn.cs script (attached to DropButton, child of Slot Prefab)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Spawn : MonoBehaviour
{
        public GameObject itemPrefab;
        private Transform player;
        public string itemName;

        void Start()
        {
                player = GameObject.FindGameObjectWithTag("Player").transform;
        }

        public void SpawnDroppedItem()
        {
                // create a 3D instance of the object in the game environment.
                Vector3 playerposition = new Vector3(player.position.x, player.position.y,
        player.position.z + 6);
                Instantiate(itemPrefab, playerposition, Quaternion.identity);
        }
}
```

42. **Put values in the Inspector for Slot scripts**: In the Hierarchy, select **Slot (1)** and in the Inspector find where the **Slot Script is** attached and notice some properties need values. From the Hierarchy drag the **InventoryController** over to the Inspector to fill the "Inventory" field where the Script is attached. From the Hierarchy drag the slot's **"Amount"** child object over to the Inspector to fill the "Amount Text" field where the Script is attached. Type a **zero** to the right of the phrase "Amount" in the Inspector where the Script is attached.

43. That takes care of Slot (1) but there are 11 more slots. In the Hierarchy, select **Slot (1)** and in the Inspector near the top click the dropdown that says "Overrides" (see **gold arrow** in the screen shot) and choose "Apply All" (see **green arrow** in the screen shot) so that all 12 slots get their Inspector properties filled similarly. Double-check that they were filled properly. Where the script is attached, the **Amount** fields should all be set to zero. You will probably have to manually change the **'i' value** for each: use 0 for Slot (1), then 1 for Slot (2), then 3, then 4, and so on up to 11.

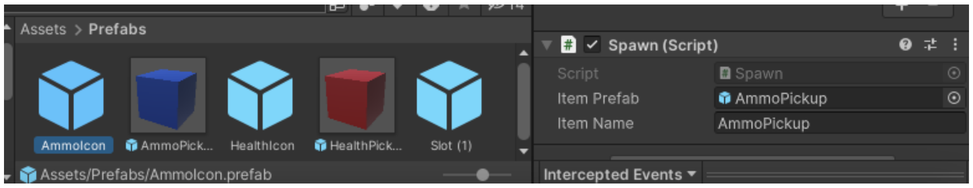


44. **Put values in the Inspector for Pickup scripts:** In the Project panel, select the **HealthPickup** Prefab and in the Inspector find where the **Pickup** Script is attached and notice some properties need values. From the Project panel drag the **HealthIcon** Prefab over to the Inspector to fill the "Item Icon" field where the Script is attached. Type "**HealthPickup**" to the right of the phrase "Item Name" in the Inspector where the Script is attached.

    a. Similarly, select the **AmmoPickup** Prefab and in the Inspector fill the two fields with the AmmoIcon Prefab and the phrase "AmmoPickup."

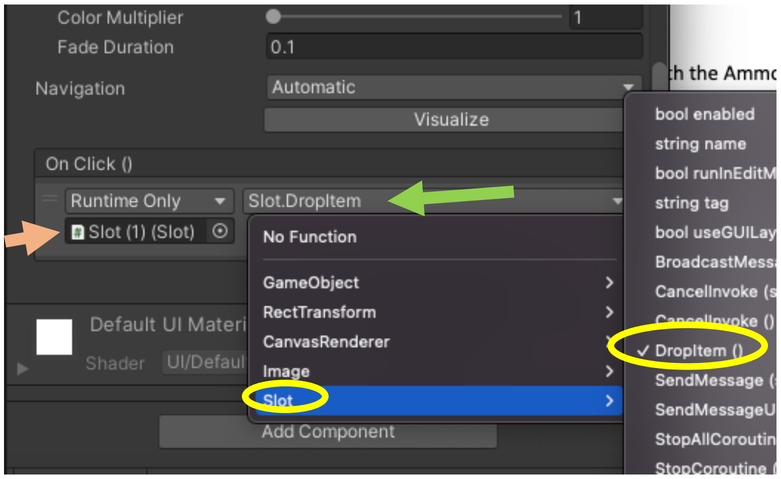
45. **Put values in the Inspector for Spawn scripts:** In the Project panel, select the **HealthIcon** Prefab and in the Inspector find where the **Spawn** Script is attached and notice some properties need values. From the Project panel drag the **HealthPickup** Prefab over to the Inspector to fill the "Item Prefab" field where the Script is attached. Type "**HealthPickup**" to the right of the phrase "Item Name". IT IS IMPORTANT THAT THIS NAME MATCHES THE ONE ON THE HEALTHPICKUP OBJECT.

    a. Similarly, select the **AmmoIcon** Prefab and in the Inspector fill the two fields with the AmmoPickup Prefab and the phrase "AmmoPickup" as shown here:



Script Spawn
Item Prefab AmmoPickup
Item Name AmmoPickup

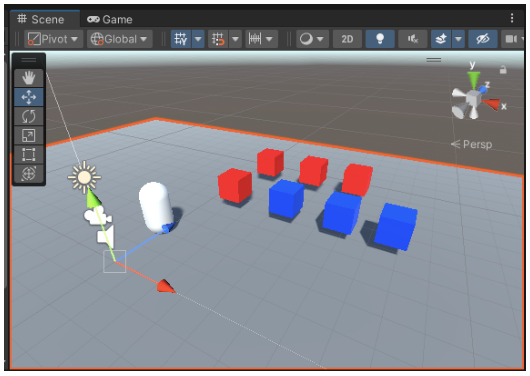
46. **Add Onclick listeners to the Drop Buttons:** In the Hierarchy, select Slot (1)'s **Drop Button child** and in the Inspector find the **Button** info area, and the sub-section entitled **OnClick().** Click the **plus +** to add a listener. You'll see three fields. From the Hierarchy drag **Slot (1)** over to the Inspector to fill the bottom left field (see orange arrow in the screen shot). Then click the upper-right dropdown (see green arrow in the screen shot) and choose **Slot** > **DropItem()** so that clicking the button will activate that function.



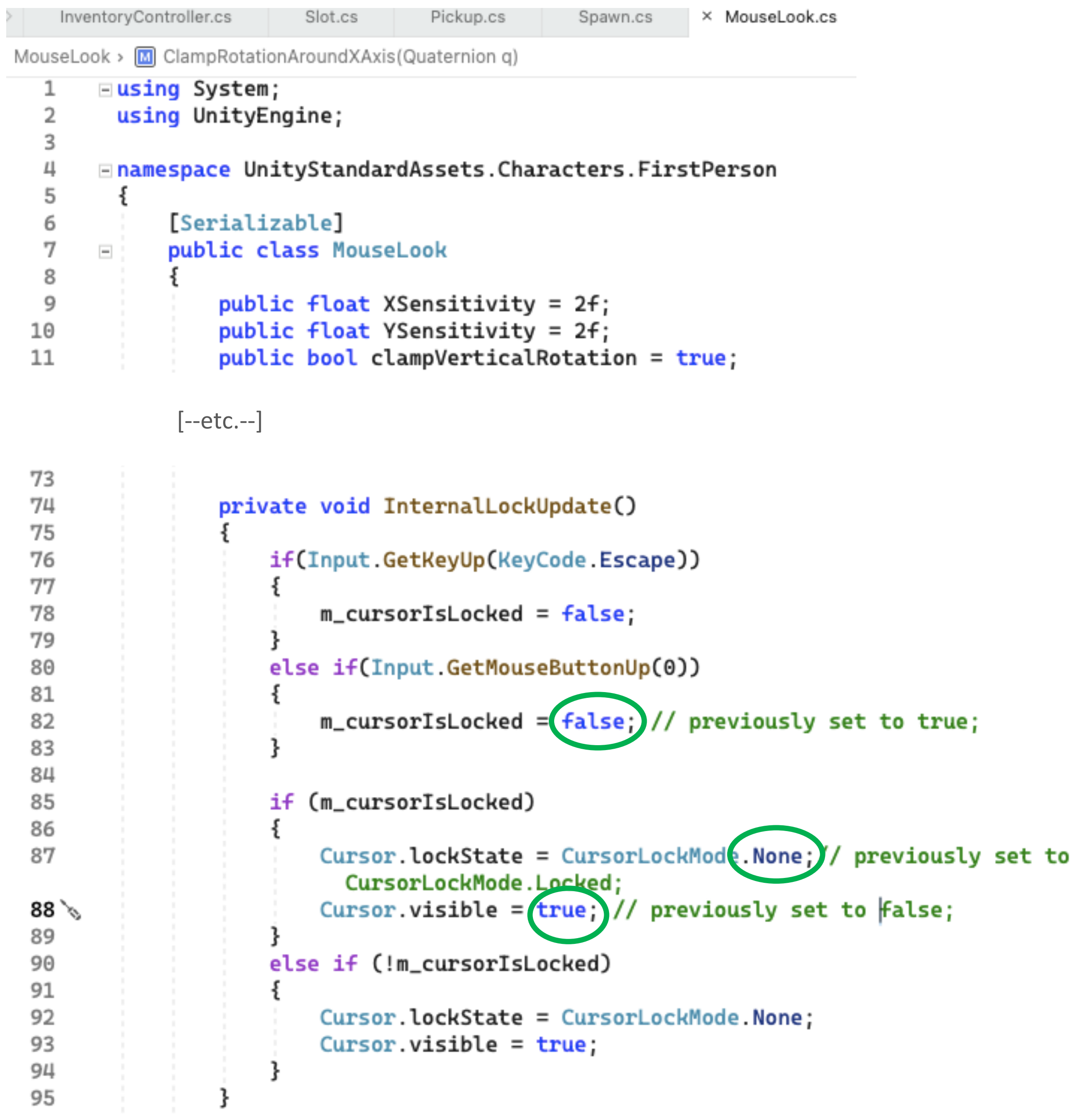
Color Multiplier 1
Fade Duration 0.1
Navigation Automatic
On Click ()
Runtime Only Slot.DropItem
Slot (1) (Slot)

47. That takes care of Slot (1)'s drop button but there are 11 more slots. In the Hierarchy, select **Slot (1)** and in the Inspector near the top click the dropdown that says "**Overrides**" and choose "**Apply All**" so that all 12 slots get their Inspector properties filled similarly. Double-check that they were filled properly.

48. Put several of each Pickup Prefab onto the Ground in the game. Play the game and practice running into the pickups to put them into your inventory. Also practice clicking the "X" drop buttons in the inventory to put the pickups back onto the Ground. My "X" drop buttons were a bit flaky… next, I'll show how I solved that…



PROBLEMS DROPPING? Each time I clicked a Slot's "Drop Button," my cursor vanished and it seldom dropped the item. That was because my FPSController had a script attached to it called "MouseLook" making the cursor vanish. I had to edit that file and make these changes, and the drop started working correctly:



```
using System;
using UnityEngine;

namespace UnityStandardAssets.Characters.FirstPerson
{
    [Serializable]
    public class MouseLook
    {
        public float XSensitivity = 2f;
        public float YSensitivity = 2f;
        public bool clampVerticalRotation = true;
```

[--etc.--]

```

        private void InternalLockUpdate()
        {
            if(Input.GetKeyUp(KeyCode.Escape))
            {
                m_cursorIsLocked = false;
            }
            else if(Input.GetMouseButtonUp(0))
            {
                m_cursorIsLocked = false; // previously set to true;
            }

            if (m_cursorIsLocked)
            {
                Cursor.lockState = CursorLockMode.None; // previously set to
                  CursorLockMode.Locked;
                Cursor.visible = true; // previously set to false;
            }
            else if (!m_cursorIsLocked)
            {
                Cursor.lockState = CursorLockMode.None;
                Cursor.visible = true;
            }
        }
```

END of DEMO