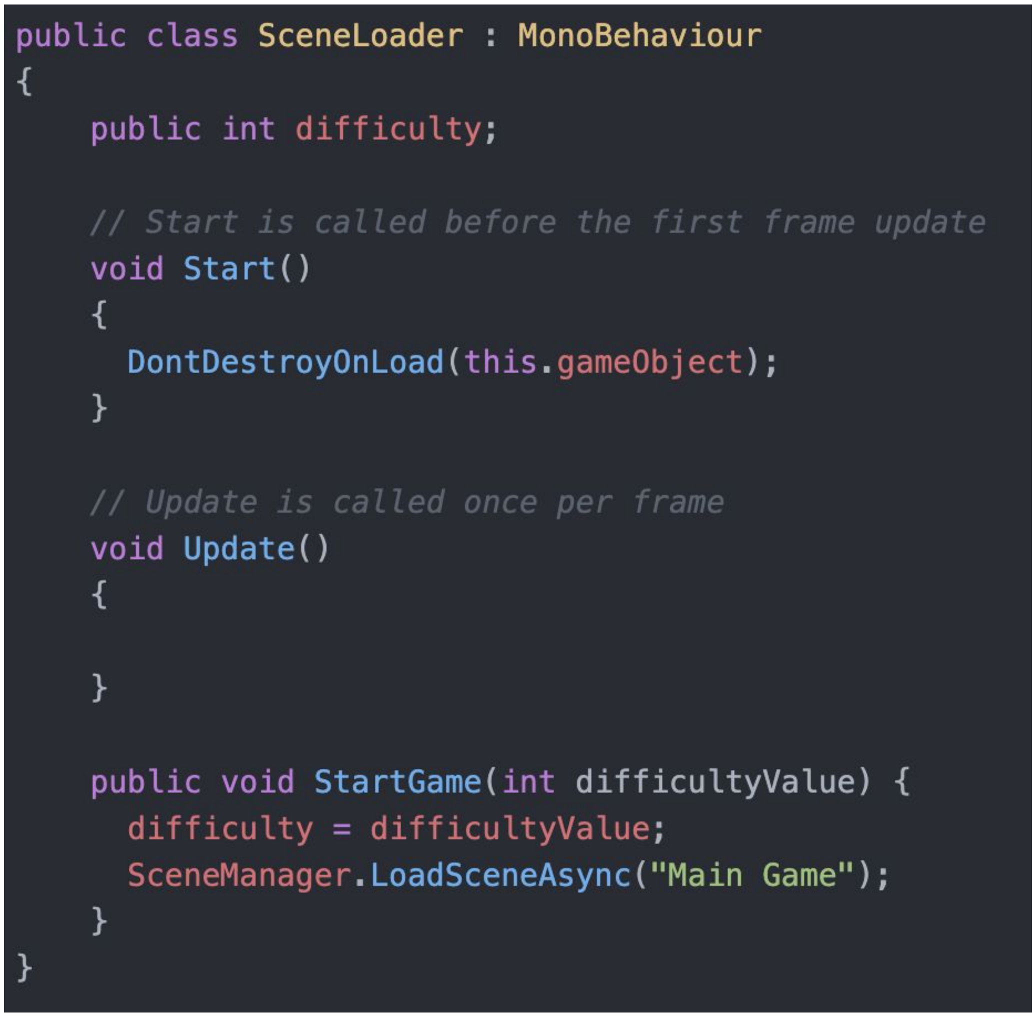
Intro

Welcome back everyone. Last time we created a score text that changes as you play the game and a game over screen with a working restart button. Today, we will start by finishing up our game with a main menu. Then we will show off another example of what you can do with UI elements and introduce you to a few more we have yet to show you.

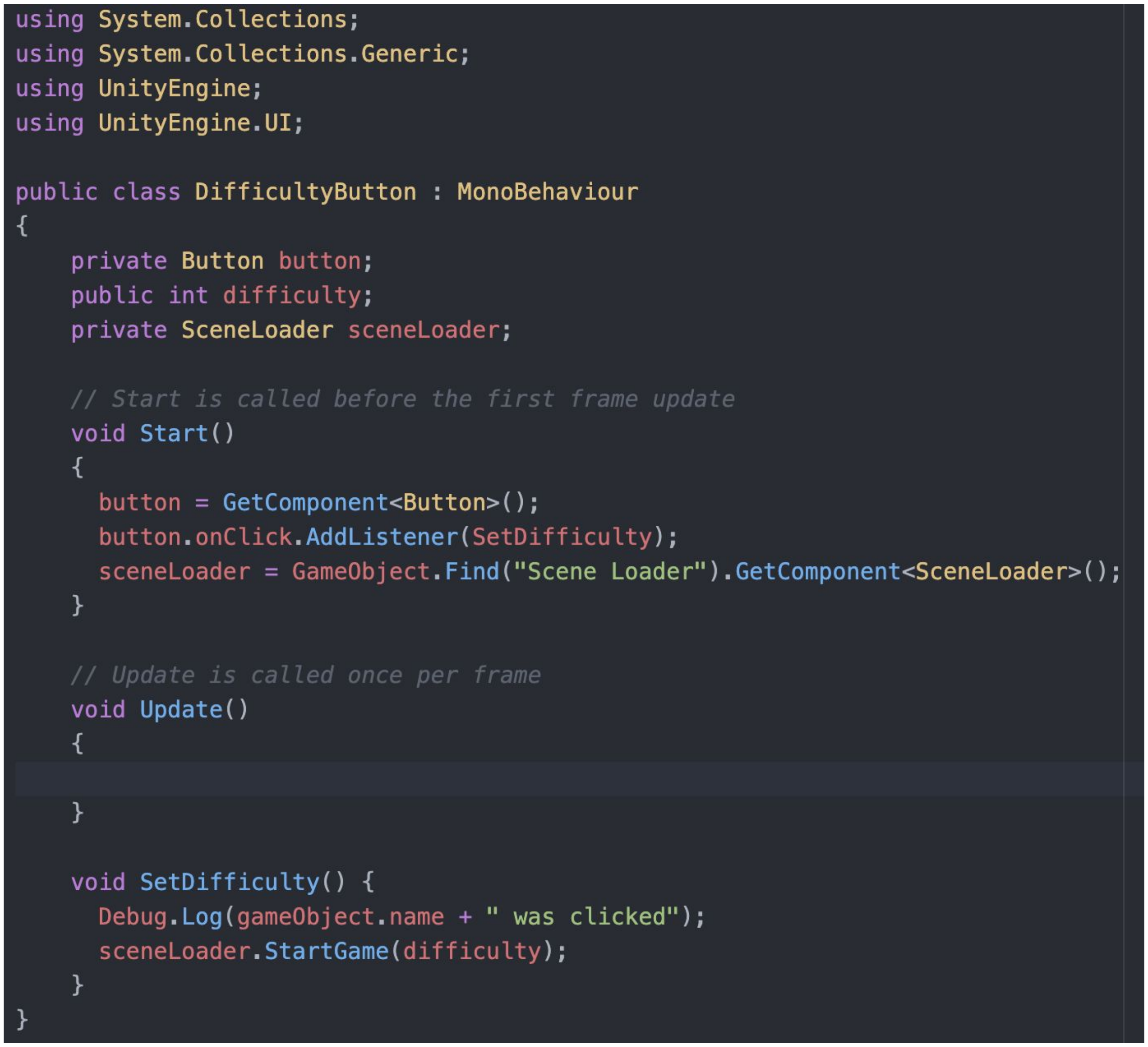
Part 3 - Main Menu

- Now it’s a little strange to be thrust into the game as soon as you start up the application. So we are going to create a main menu.
- Start by opening up a new scene and saving it as “Main Menu”
- Right click in the Hierarchy pane and click → UI → Text - TextMeshPro and rename this as Title
- Change the text itself to be Crates and the position to (0, 0)
- Change the color, font, and size if you wish
    - You may have to disable wrapping to get the Title on one line
- Now let’s create some buttons to choose the difficulty for our game
- Right click in the Hierarchy pane and click → UI → Button - TextMeshPro
- Change the name to Easy Button and its text to Easy.
    - Change the size if you wish
- We’ll also reduce the width to make the button look nicer and move it to the left.
- In the Hierarchy panel click the button and hit Ctrl-D twice to duplicate it twice
- Select the new buttons in the Hierarchy pane and change the position so that all of our buttons aren’t overlapping one another.
- Change the name and text of the new buttons to Medium Button and Medium for the second one and Hard Button and Hard for the third
- Next we are going to go to our Game Over scene and make some changes to our Scene Loader object
- First go into your SceneLoader.cs script to add a difficulty variable and update the Start() and StartGame() functions:



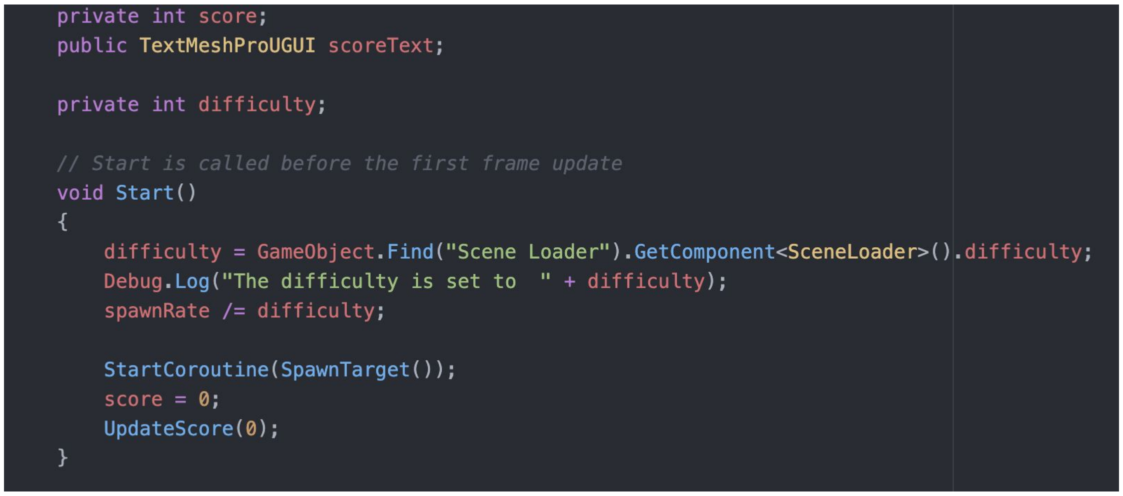
```
public class SceneLoader : MonoBehaviour
{
    public int difficulty;

    // Start is called before the first frame update
    void Start()
    {
      DontDestroyOnLoad(this.gameObject);
    }

    // Update is called once per frame
    void Update()
    {

    }

    public void StartGame(int difficultyValue) {
      difficulty = difficultyValue;
      SceneManager.LoadSceneAsync("Main Game");
    }
}
```

- The DontDestroyOnLoad will make it so that our Scene Loader game object will persist through different scenes. Through this we can transfer the difficulty value between scenes
- In the Project panel navigate to Assets > Prefabs and drag the Scene Loader from the Hierarchy panel to the Project panel
- Then delete Scene Loader from the Hierarchy and then drag the new prefab to the Restart Button’s On Click event. Don’t forget to reselect the StartGame(int) function too
- Go back to the Main Menu scene and add this new prefab to the Hierarchy
- Create a DifficultyButton.cs script and attach this to our three difficulty buttons
- In this new script:



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

public class DifficultyButton : MonoBehaviour
{
    private Button button;
    public int difficulty;
    private SceneLoader sceneLoader;

    // Start is called before the first frame update
    void Start()
    {
      button = GetComponent<Button>();
      button.onClick.AddListener(SetDifficulty);
      sceneLoader = GameObject.Find("Scene Loader").GetComponent<SceneLoader>();
    }

    // Update is called once per frame
    void Update()
    {

    }

    void SetDifficulty() {
      Debug.Log(gameObject.name + " was clicked");
      sceneLoader.StartGame(difficulty);
    }
}
```

- Go to the inspector for each of the buttons and you should see a difficulty field in the script section. Assign Easy a difficulty of 1, Medium 2, and Hard 3
- Now if we click the play button from our Main Menu scene, we should be able to play the game and still use the restart button on the Game Over screen
    - A message should appear in the console telling you which button you pressed too
- However, we aren’t actually changing the difficulty yet
- Go to your GameManager.cs script and add a difficulty variable and update the Start() function:



```
private int score;
public TextMeshProUGUI scoreText;

private int difficulty;

// Start is called before the first frame update
void Start()
{
    difficulty = GameObject.Find("Scene Loader").GetComponent<SceneLoader>().difficulty;
    Debug.Log("The difficulty is set to  " + difficulty);
    spawnRate /= difficulty;

    StartCoroutine(SpawnTarget());
    score = 0;
    UpdateScore(0);
}
```

- Now you should have buttons that change the difficulty for the game and restarting the game will put you at the same difficulty too
    - A message should also appear in you console log that says what the difficulty is set to

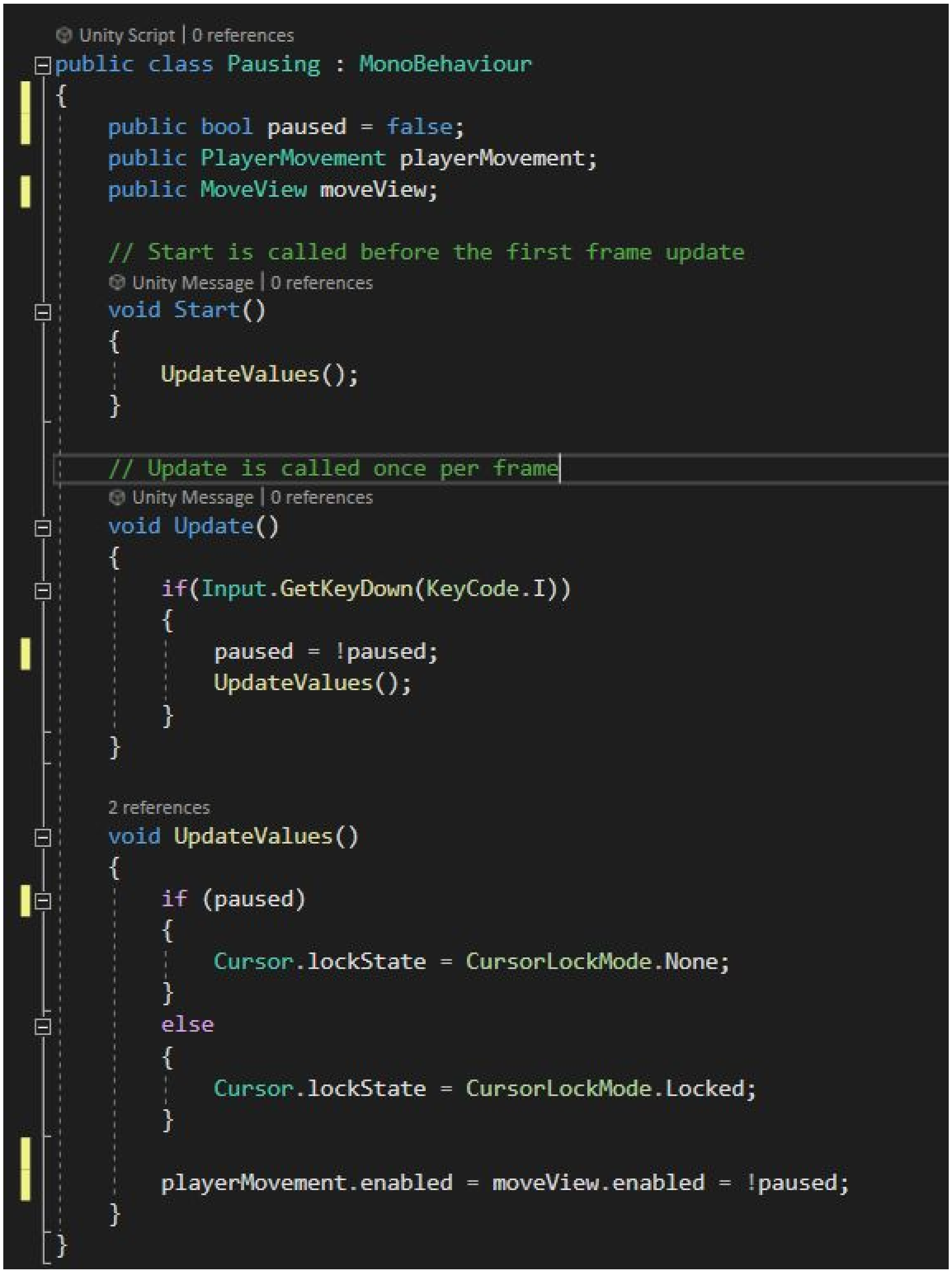
Part 4 - Inventory System

Sources: Brackeys from YouTube and Ahninniah from the Unity Asset Store
Intro:

This tutorial is going to cover ways to pause in Unity and is also going to involve using the UI grid layout component. We are going to be using a prebuilt game that already includes player movement, item interaction and an inventory. This inventory, however, has no representation visually and the HeldItem child object of the player can't have any items assigned to it in the game's current state.

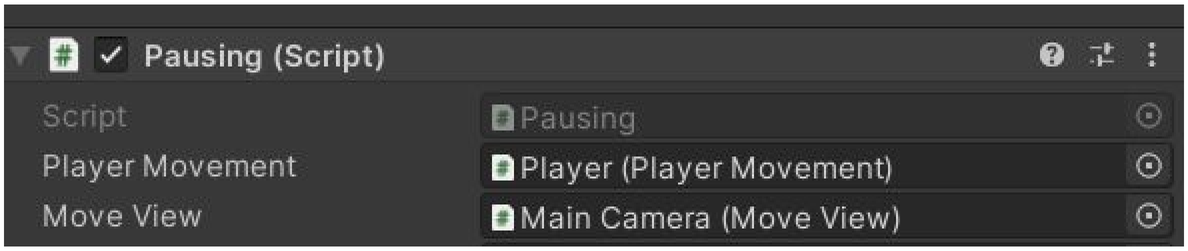
- Create Pause script
- Choosing what and what not to disable can vary depending on the game, but here are a few things to keep in mind:
    - Disabling objects by tag can be difficult, because objects can be re-enabled directly using a tag
    - Things like the player controller are obvious, but keep in mind other objects' operations that will be running
    - In the case of larger games, using parent objects is the best way to go about this
- Luckily, for this tutorial, we're only concerned about restricting player control while pausing, which means retrieving and disabling the MoveView and PlayerMovement scripts according to a boolean value

- The cursor is set to be locked and invisible in the middle of the screen at startup by the MoveView script, but to allow it to be used on the pause screen, we're going to be changing that variable based on the boolean as well



```
Unity Script | 0 references
public class Pausing : MonoBehaviour
{
    public bool paused = false;
    public PlayerMovement playerMovement;
    public MoveView moveView;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        UpdateValues();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if(Input.GetKeyDown(KeyCode.I))
        {
            paused = !paused;
            UpdateValues();
        }
    }

    2 references
    void UpdateValues()
    {
        if (paused)
        {
            Cursor.lockState = CursorLockMode.None;
        }
        else
        {
            Cursor.lockState = CursorLockMode.Locked;
        }

        playerMovement.enabled = moveView.enabled = !paused;
    }
}
```

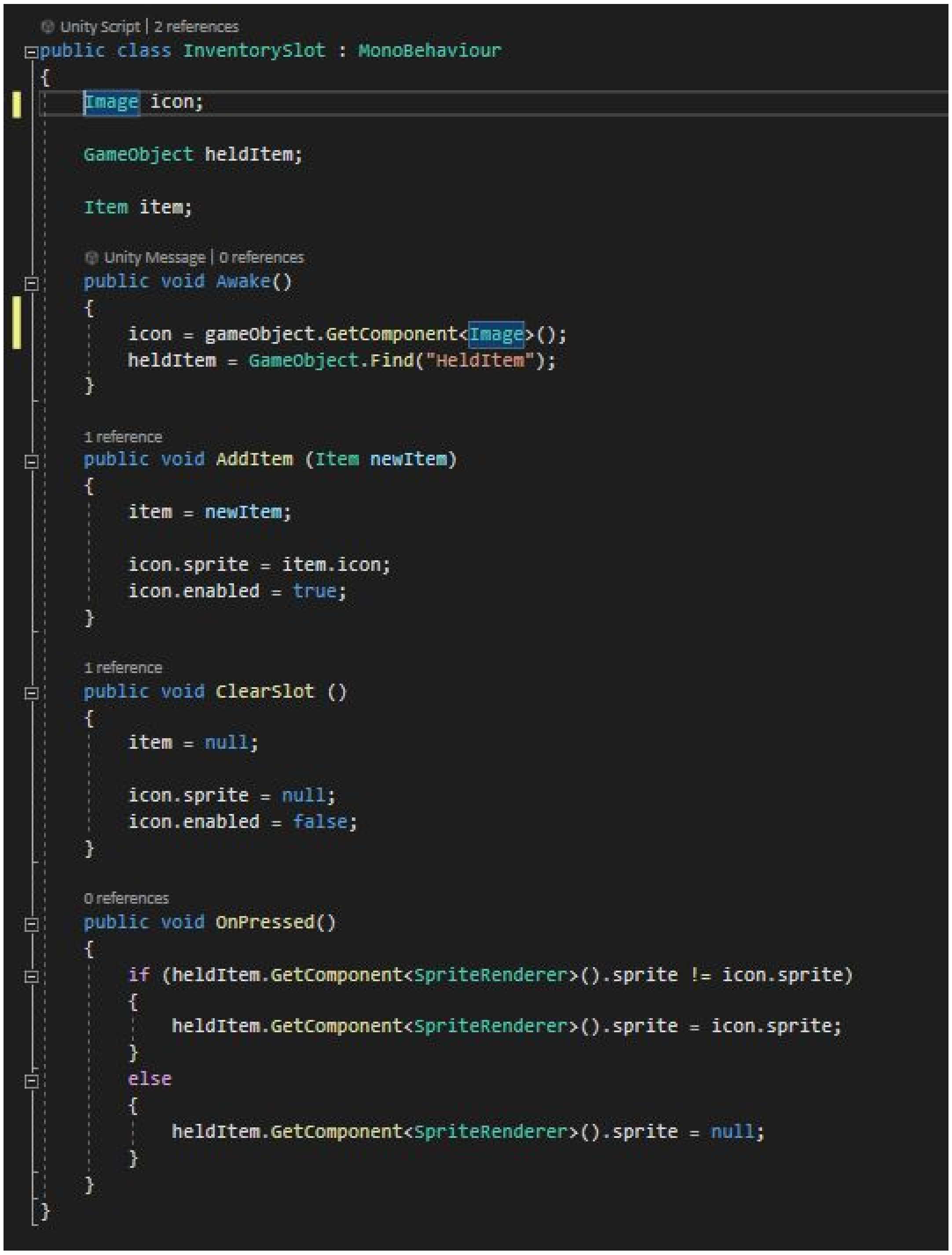
- We’ll then put this script in the GameManager object and assign its public variables:



Pausing (Script)

Script	Pausing
Player Movement	Player (Player Movement)
Move View	Main Camera (Move View)

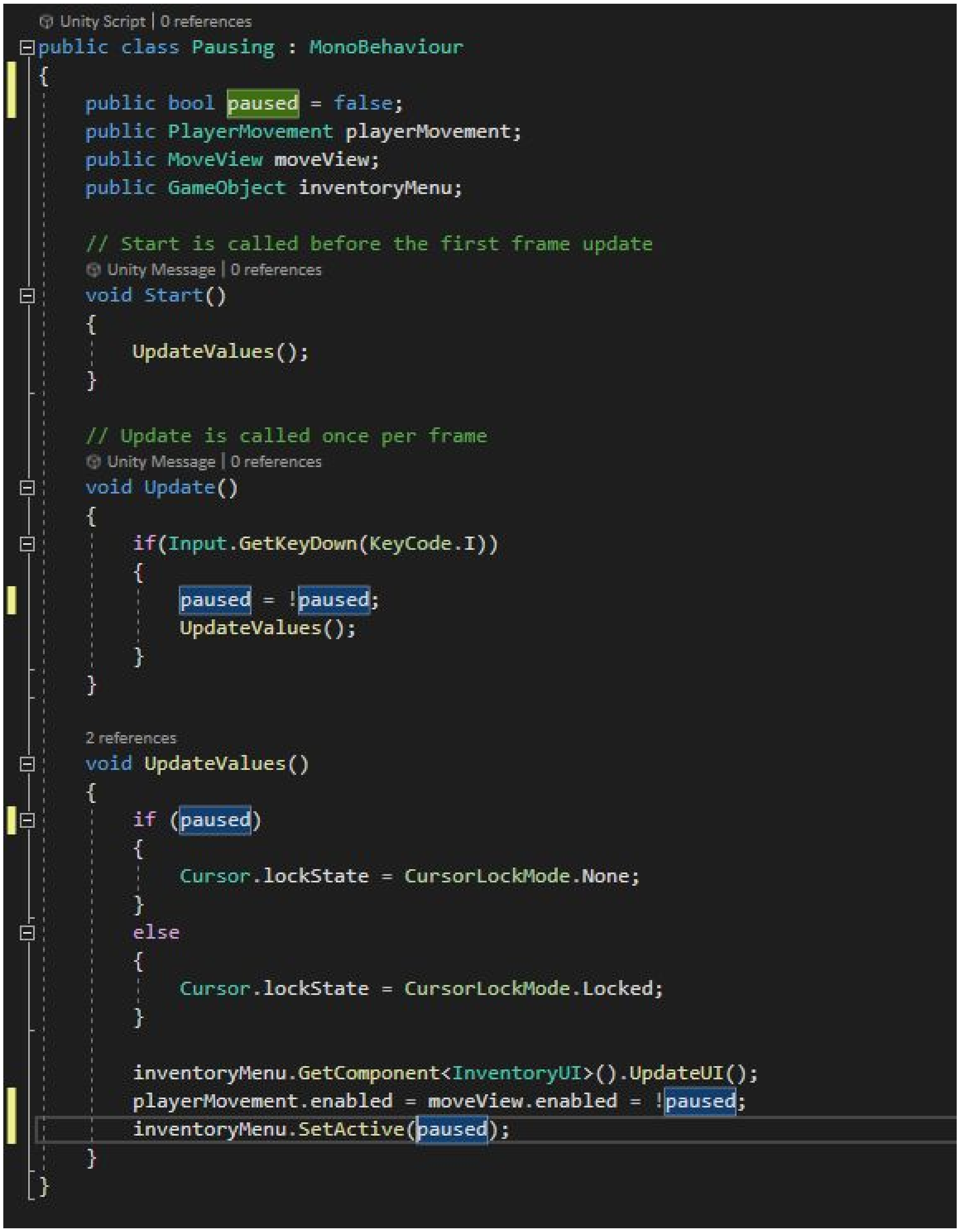
- Now we’re going to start setting up the menu the player will be able to interact with
- In the Canvas object, we’re going to create a UI>Panel object as its object that will be called InventoryMenu and will be sized according to the values in my final game
    - We’re going to briefly cover some panel and canvas attributes in Unity and some common beginner pitfalls
    - How to anchor panels by place on screen rather than dimensions to have consistency across aspect ratios
- We’re going to add a small panel as a child of InventoryMenu with a TextMeshPro child object that will indicate that this is an inventory menu
- A child of the InventoryMenu will also be created which is going to be an empty game object called InventoryGrid
    - In this game object we’re going to add a Grid Layout component which is a quick, easy way to format UI that has things like repeating buttons or displays
    - We’re going to briefly go over some of the components of the Grid Layout component
- We’re then going to create a child of this InventoryGrid called InventorySlot that is a Button
- This InventorySlot object will be copied and pasted 24 times to give the grid the 25 components to represent the inventory slots. This will allow us to play more with the Grid Layout options and get our inventory to look nice
- Now we can delete all of the buttons except for the first InventorySlot, as we’re going to be making changes to it and we’ll copy it again later, but we’re firstly going to create a script called InventorySlot to define the behavior of this object
- This script is going to use a small amount of Image component functionality, but we’re going to do more once we attach it to the object



```
Unity Script | 2 references
public class InventorySlot : MonoBehaviour
{
    Image icon;

    GameObject heldItem;

    Item item;

    Unity Message | 0 references
    public void Awake()
    {
        icon = gameObject.GetComponent<Image>();
        heldItem = GameObject.Find("HeldItem");
    }

    1 reference
    public void AddItem (Item newItem)
    {
        item = newItem;

        icon.sprite = item.icon;
        icon.enabled = true;
    }

    1 reference
    public void ClearSlot ()
    {
        item = null;

        icon.sprite = null;
        icon.enabled = false;
    }

    0 references
    public void OnPressed()
    {
        if (heldItem.GetComponent<SpriteRenderer>().sprite != icon.sprite)
        {
            heldItem.GetComponent<SpriteRenderer>().sprite = icon.sprite;
        }
        else
        {
            heldItem.GetComponent<SpriteRenderer>().sprite = null;
        }
    }
}
```

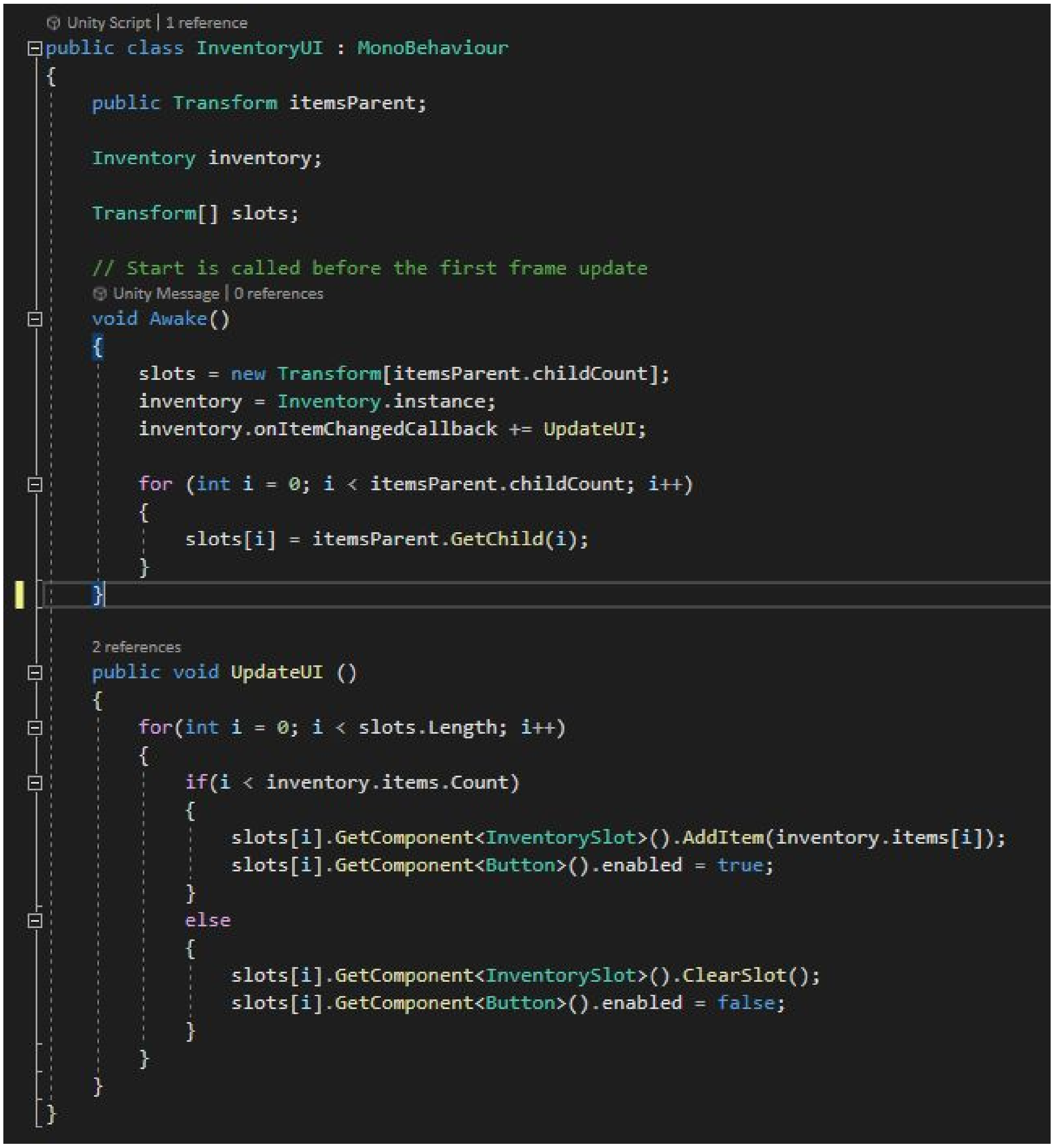
- We’re going to attach this script to the InventorySlot object, and then add the image component that this code references
- We can use a placeholder image to get the dimensions of this image looking nice inside the button, as this is where the image of the item in this slot is going to be displayed
- This script is also going to be interacting with the Button component, so we’re going to place OnPressed() into OnClick() in the inspector

- This object can now be copied 24 times. A prefab can be made if you'd like for easy recreation and access to the InventorySlot's variables. It's good practice if you want to make changes down the line, but it's not necessary for the scope of this tutorial
    - Copying a prefab in the hierarchy is an easy way to duplicate them and it still counts as a prefab
- The inventory menu can now be seen while playing the game, but it covers the screen regardless of whether the game is paused or not
- We're going to go back into the Pausing script in GameManager to add a reference to the InventoryMenu object and some lines of code that'll enable/disable it based on the paused boolean



```
Unity Script | 0 references
public class Pausing : MonoBehaviour
{
    public bool paused = false;
    public PlayerMovement playerMovement;
    public MoveView moveView;
    public GameObject inventoryMenu;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        UpdateValues();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if(Input.GetKeyDown(KeyCode.I))
        {
            paused = !paused;
            UpdateValues();
        }
    }

    2 references
    void UpdateValues()
    {
        if (paused)
        {
            Cursor.lockState = CursorLockMode.None;
        }
        else
        {
            Cursor.lockState = CursorLockMode.Locked;
        }

        inventoryMenu.GetComponent<InventoryUI>().UpdateUI();
        playerMovement.enabled = moveView.enabled = !paused;
        inventoryMenu.SetActive(paused);
    }
}
```

- Our final script of this tutorial is going to be the InventoryUI script which is going to be placed in the InventoryMenu game object



```
Unity Script | 1 reference
public class InventoryUI : MonoBehaviour
{
    public Transform itemsParent;

    Inventory inventory;

    Transform[] slots;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Awake()
    {
        slots = new Transform[itemsParent.childCount];
        inventory = Inventory.instance;
        inventory.onItemChangedCallback += UpdateUI;

        for (int i = 0; i < itemsParent.childCount; i++)
        {
            slots[i] = itemsParent.GetChild(i);
        }
    }

    2 references
    public void UpdateUI ()
    {
        for(int i = 0; i < slots.Length; i++)
        {
            if(i < inventory.items.Count)
            {
                slots[i].GetComponent<InventorySlot>().AddItem(inventory.items[i]);
                slots[i].GetComponent<Button>().enabled = true;
            }
            else
            {
                slots[i].GetComponent<InventorySlot>().ClearSlot();
                slots[i].GetComponent<Button>().enabled = false;
            }
        }
    }
}
```

- This is going to involve discussing the inventory script logic and how we can use it to be displayed by the UI
- We then will assign InventoryGrid as the itemsParent variable in the inspector
- And we’re done!
- If time permits, I’ll answer questions and go over some of the more specific workings of the base game components