Wanted to use a function other than GetKeyDown when assigning button presses in case the player would like to modify the keybinds. Found out about the axis feature - Unity has a built in feature that allows you to assign names and buttons to various virtual axis and allows you to call them with the Input.GetAxisRaw function.
Example

Input.GetAxisRaw(“Horizontal”) //…

Had trouble applying the Input.GetAxisRaw function to an if statement to track player input. I had tried to use it in the same way you’d use Input.GetKeyDown, but it turns out GetAxisRaw works differently. Found out that Input.GetAxisRaw outputs either a positive or negative number correlating to its position on a virtual axis. If I wanted to use it in an if statement, I would do as follows:

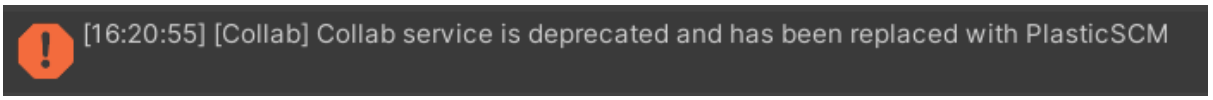
```
if (Input.GetAxisRaw("Horizontal") != 0f)
{
  // ...
}
```

Wanted to know how to access a private variable within the unity editor without changing it to public. Learned that adding [Serializefield] in front of the variable would allow me to do this.
Example:

[Serializefield] public GameObject player

Allows the ‘player’ gameobject to be accessed in the editor, but it cannot be accessed by other scripts.

Had an error appear in the console that seemed unrelated to the project, it was as follows



As it turns out, I was using an outdated version of unity. I installed a more recent version of unity and upgraded my project to that version, and this fixed the problem.

Wanted to have a camera following the player, however the camera would look rather jittery at times. The statement that allowed the camera to follow the player was placed inside the update function, however I found out that LateUpdate() would be more suited for the task. LateUpdate is called on every frame like update, however it’s called after all other functions on that frame have finished.
Example:

```
Void LateUpdate()
{
```

```
        //…
}
```

After adding a tilemap as a background, I found that some elements of my game were disappearing. The reason why is that they were all on the same Z level, and as such my player character and NPCs were being obscured behind the background.

To fix this, I set the backgrounds Z level to 1 to ensure that it stays behind all elements on Z level 0. Positive Z levels place an object farther from the top, while negative places them closer.

I was having issue with a script copied from a tutorial. It is as follows:

```
using UnityEngine;

//Made using Semag Games' Dialogue tutorial: https://www.youtube.com/playlist?list=PLCGaK2yqfY2IrJYnOnlgdmzWVUFXsRQXA

Unity Script (1 asset reference) | 1 reference
public class DialogueActivator : MonoBehaviour
{
    [SerializeField] private DialogueAsset dialogueAsset;

    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player") && other.TryGetComponent(out PlayerController player))
        {
            player.Interactable = this;
        }
    }

    Unity Message | 0 references
    private void OnTriggerExit2D(Collider2D other)
    {
        if (other.CompareTag("Player") && other.TryGetComponent(out PlayerController player))
        {
            if (player.Interactable is DialogueActivator dialogueActivator && dialogueActivator == this)
            {
                player.Interactable = null;
            }
        }
    }
    0 references
    public void Interact(PlayerController player)
    {
        player.DialogueUI.ShowDialogue(dialogueAsset);
    }
}
```

Line 13 was causing trouble and was returning the following error

```
Assets\Scripts\Dialogue Scripts\DialogueActivator.cs(13,35): error CS0266: Cannot implicitly convert type 'DialogueActivator' to 'IInteractable'. An explicit conversion exists (are you missing a cast?)
```

The issue was that I was supposed to have the script inherit from another script, IInteractable. Placing ‘IInteractable’ after MonoBehaviour fixed the issue and the script was able to function as normal.

Wanted to learn how to make a main menu, and in the process learned of SceneManager.LoadScene(). This can load a scene based off of either its name or its index number.
Example:

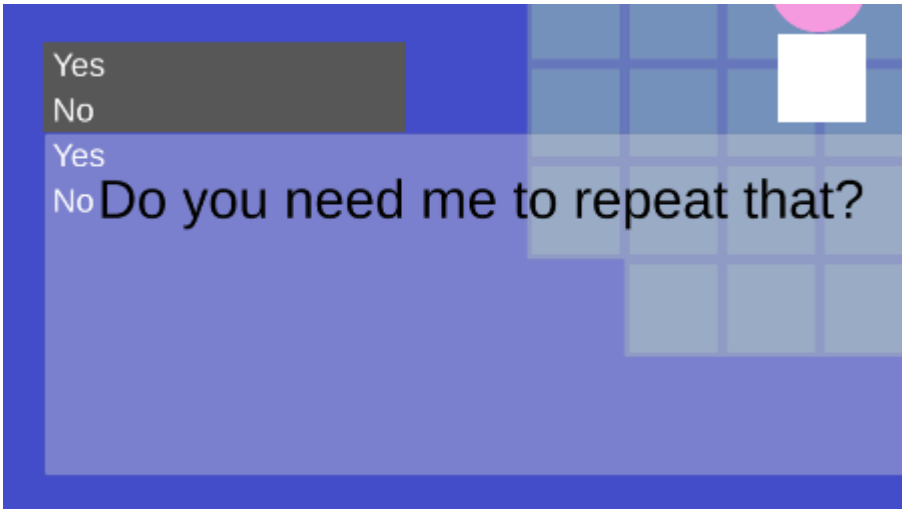
```
Void loadlevel()
{
```

```
        SceneManager.LoadScene(1);
}
```

Or

```
Void loadlevel()
{
        SceneManager.LoadScene("Level1");
}
```

I'm currently facing an issue with the dialogue tree system. After the first branch of the dialogue system the options presented duplicate themselves.



I added Debug.Log commands to the lines that were being called to make the response options appear and following where they were called in the console. This continued until we got two different debug logs from the same source but calling from two different lines. This led me to the ResponseHandler script.
I found out that the issue was that I had accidentally duplicated a line of code as follows:

```
if (response.DialogueAsset)
{
    dialogueUI.ShowDialogue(response.DialogueAsset);
}
else
{
    dialogueUI.CloseDialogueBox();
}

dialogueUI.ShowDialogue(response.DialogueAsset);
```

The issue was repaired by removing the final line in this screenshot.