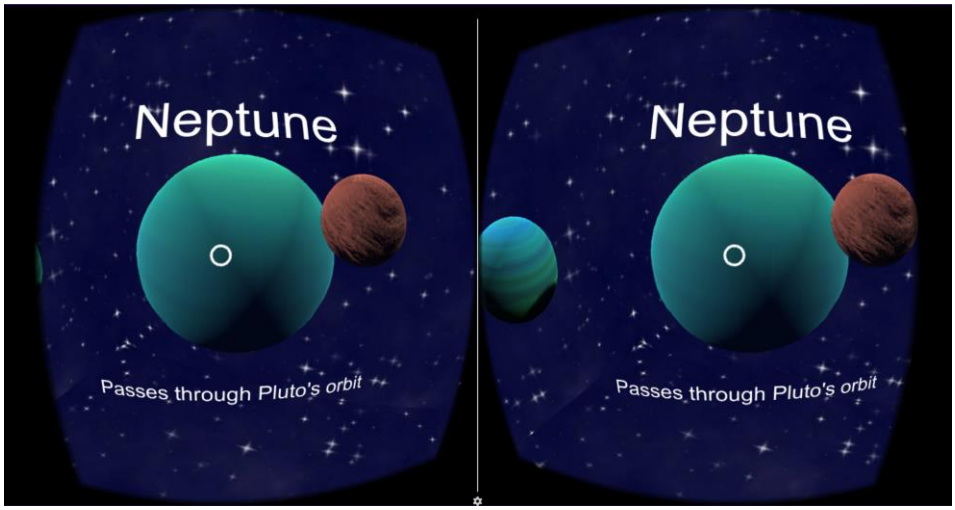
LAB #5 – PROJECT NOTES

Commented [AS1]: Alex: copyediting/styles later

INTRODUCTION TO C# AND SCRIPTING

This is a companion document to the Lab #5 document that will cover step by step how to go through the process of creating C# scripts to control different behaviors in our experience.

After completing this lab, we will be able to switch between our menu scene and our main scene, and show/hide our UI text on our planets when a viewer triggers a click on the side of their device.



Before completing this lab, you should have completed Lab 4 and added Text elements to all of the planets in your scene, or have downloaded the zip folder for Lab 5.

EVENT SYSTEM MECHANICS

The first thing we are going to do with our main menu scene is add the Google VR Gaze Input Module to our Event System. The Event System object listens for things going on in the scene and tells them when to respond accordingly.

1. Open up the UI_Scene by double clicking on it in the Asset window.
2. Select the Event System in the Hierarchy and click the “Add Component” button in the Inspector.
3. Type in “Gaze Input Module” and hit enter to add the component to the Event System.

4. Click the small gear next to the Gaze Input Module and click “Move Up”. This will have gaze events trigger before normal input is processed.

GIVING OUR OBJECTS TRIGGERS

Now that we have a way to manage the input we receive from the user’s gaze, we need to add triggers to our UI objects to add behaviors to them. In this scene, we will be adding a ‘Button Script’ to handle the interaction of looking at and “clicking” by triggering the hardware button – in our case, a magnet pull.

1. Expand the Canvas item in the Hierarchy and select the Button element.
2. In the Inspector for the Button, click the ‘Add Component’ button and search for “Event Trigger”. Hit enter to add the component to the button.
3. Click the ‘Add Component’ button again and type in ‘ButtonScript’. When prompted, choose the option to Add New Script > C# Script. Name the script ‘ButtonScript’ and press enter to create and add the new script.
4. Double click on the greyed out ButtonScript element to open up the script. By default, your script will launch in MonoDevelop or Visual Studio, depending on what you installed when you installed Unity.
5. We are going to create a one line function that will launch our main menu scene when our button is “pressed”. Replace the contents of the ButtonScript.CS file with the following:

ButtonScript.cs

```
using UnityEngine;
using System.Collections;

public class ButtonScript : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }

    // Navigate to main scene
    public void StartScene()
    {
        // We will load the first scene (this is scene 0)
        UnityEngine.SceneManagement.SceneManager.LoadScene(1);
    }
}
```

6. Back in Unity, open up the Build Settings by going to File > Build Settings. Add the UI_Scene to the Build Settings dialog by clicking “Add Open Scenes”. Make sure that there is a 0 next to the UI_Scene.
7. Switch over to your Main Scene and repeat step 6, making sure that the main scene has a number 1 next to it in the Build Settings. Save and return to UI_Scene.
8. In the UI_Scene, select the Button object again and look at the Event Trigger component. Click the “Add New Event Type” button and select “PointerClick” from the menu that appears.
9. Select the circle to the right of the box that appears directly below the “Runtime Only” drop down. Choose the ‘Button’ object from the box that appears.
10. In the drop down next to the “Runtime Only” drop down, choose “ButtonScript > StartScene”.

Try playing your scene and see how to simulate a trigger pull. You should see your application switch to your other scene!

ADD A RETICLE

We want to make it easier to see when we’re hovering over objects that can be selected, so we’re going to add a reticle to our camera that will show us when we’re looking at an interactive object.

1. In the Assets folder, navigate to GoogleVR > Prefabs > UI and select the GvrReticle prefab.
2. Drag the GvrReticle prefab onto your main camera object
3. Select your Main Camera in the Hierarchy. In the Inspector window, click “Add Component” and search for ‘GVR Gaze’ to add the Gaze script to the camera.

That’s it! Play your scene and take a look around. You should notice the reticle change when your gaze goes over the button.

UI ELEMENTS FOR 3D OBJECTS

To interact with a 3D object, we need to add an additional element to our camera – a physics raycaster. This will send out a “ray” from the camera to detect collisions with solid objects.

1. Open your main scene by double clicking it in the Assets folder.
2. Repeat the steps in “Add A Reticle” for your scene to add the pointer object to the camera.
3. Select the Main Camera and click the “Add Component” button. Search for ‘Physics Raycaster’ and add it to the Main Camera.

Once we have set up our camera to detect collisions with 3D objects, it's time to add a new script to our PlanetCanvas prefab. The following 'Billboard' script will set up our text so that it is always facing the viewer. This is helpful in that we don't have to worry about which way the text is oriented relative to the user.

1. Select the PlanetCanvas prefab in the Assets window and click "Add Component" > "New C# Script". Name the script "Billboard.cs"
2. Open up Billboard.cs and copy the following into the script:

Billboard.cs

```
using UnityEngine;
using System.Collections;

public class Billboard : MonoBehaviour {

	// Use this for initialization
	void Start () {

	}

	// Update is called once per frame
	void Update () {
	 transform.LookAt(transform.position +
	                  Camera.main.transform.rotation * Vector3.back,
	                  Camera.main.transform.rotation * Vector3.up);
	}
}
```

SHOWING AND HIDING UI ELEMENTS

The final part of our project is to set up the interactive element in our scene so that we can show and hide our UI when people interact with the planets. To do this, we are going to write a script that:

- Rotates our planets around in space
- Pauses when a viewer looks at them
- Shows the text we added when the viewer "clicks" with the magnet
- Auto-hides the text and resumes rotating when the viewer looks away

To do this:

1. Right click in the Assets window and create New > C# Script. Name the Script "PlanetScript.cs"
2. Copy the below code into PlanetScript.cs

PlanetScript.cs

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
/// <summary>
/// Variables for the planet script
/// Controls rotation
/// On gaze + trigger, show UI
/// When gaze Edits, hide UI
/// </summary>
///
public class PlanetScript : MonoBehaviour {

    // Public variables
    public float planetRotationSpeed; // For animating the planet movement

    // Private variables
    bool isFloating = true;

       // Use this for initialization
       void Start () {
       }

       // Update is called once per frame
       void Update () {
        if(isFloating)
        {
            gameObject.transform.Rotate(new Vector3(0.0f, planetRotationSpeed, 0.0f));
        }
    }
    /// <summary>
    /// Show or hide the UI based on whether or not it is showing already
    /// Will be called with EventTrigger OnPointerClick
    /// </summary>
    public void ShowHideUIDetails()
    {
        isFloating = false;
        GetComponentInChildren<Canvas>().enabled
= !GetComponentInChildren<Canvas>().enabled;

GetComponentInChildren<Canvas>().GetComponent<CanvasScaler>().dynamicPixelsPerUnit = 10;
    }

    /// <summary>
    /// Hide the UI for a planet when the user stops looking at it
    /// Will be called with EventTrigger OnPointerExit
    /// </summary>
    public void HideUIWhenGazeExit()
    {
        GetComponentInChildren<Canvas>().enabled = false;
        isFloating = true;
    }
}
```

3. Select all of your planets in the Hierarchy by holding down CTRL and click “Add Component”.
4. Type in “PlanetScript” and add it to your planets.
5. Like before, we will need to add in EventTriggers for our planets. You will have to do this individually for each planet:
    a. Select the planet and Add Component > Event Trigger
    b. Add the Pointer Click trigger, select the planet, and choose ShowHideUIDetails
    c. Click Add New Event Type and choose Pointer Exit
    d. Select the planet, and choose HideUIWhenGazeExit

Click Play and spend some time viewing your different planets!

NEXT TIME

For our final lab, we will be building our application into a format that can be stored on a phone and run in our headset!

Before you come to the next lab, finish this week’s lab by adding the show/hide script to the rest of the planets in your scene. Alternatively, download the zip folder for Week 6 before attending the next lab.