VR Humans and Systems

Project Documentation

Members:

- **Mazen Hassaan** **2307227**
- **Toseef Ahmed** **2307270**
- **Hung Trinh** **2307229**

Introduction

In our project, we chose to implement 4 different illusions in VR which are; Optical Illusion, Moon Illusion, Ponzo Illusion and Ebbinghaus Illusion.

The Optical Illusion: This scene is based on Chromatic Adaptation. In this scene, the outline of an object is presented in VR in two contrasting colors. These contrasting colors are presented between the object itself and the background with a black point in the middle. During the scene, the user should focus on the black dot in the middle for a period of time and then the real object with its distinct features will be suddenly shown. The user is expected to see a trace of both the previous presented colors imposed on the new scene.

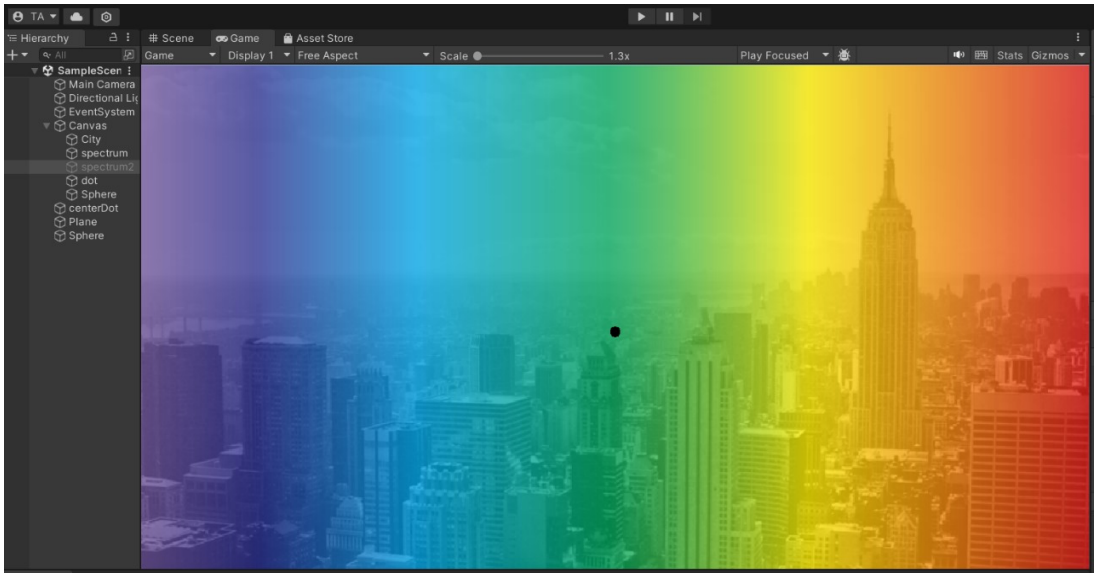
Moon Illusion: In this virtual reality setting, the moon illusion phenomenon which is experienced in our everyday life is explored. The user will stand and will be presented with two moons in the sky. One moon will appear at the horizon line near the outlines of mountains and the road, and the other one in the sky above. It's expected that the moon intersecting with the horizontal line should appear larger compared to the one higher in the sky above the user although, originally, they are of the same size.

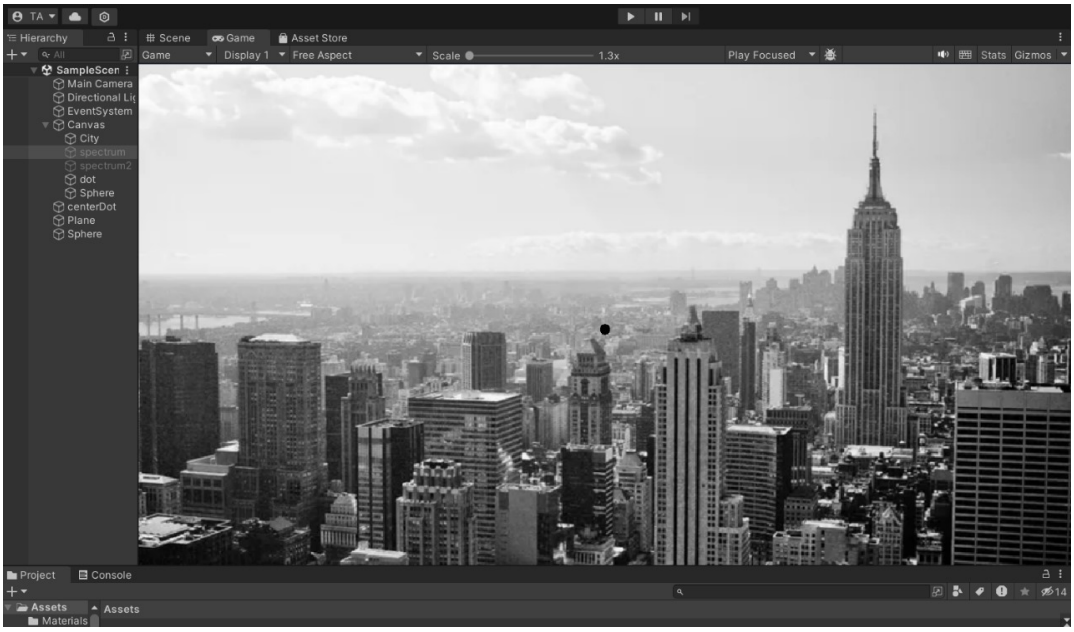
Ponzo Illusion: This is a visual illusion that involves the perception of the size of an object being affected by the background it's presented in. It illustrates how our perception of size is influenced by the depth cues and the contextual visual information. It's based on the idea of placing two similar objects on a pair of converging lines.

Ebbinghaus Illusion: This illusion involves two central spheres surrounded by different-sized spheres. The surrounding spheres around each sphere can create the illusion that both central spheres are different in sizes. This phenomenon plays with our perception of relative size and can be a fascinating subject to study in VR especially with 3D objects.

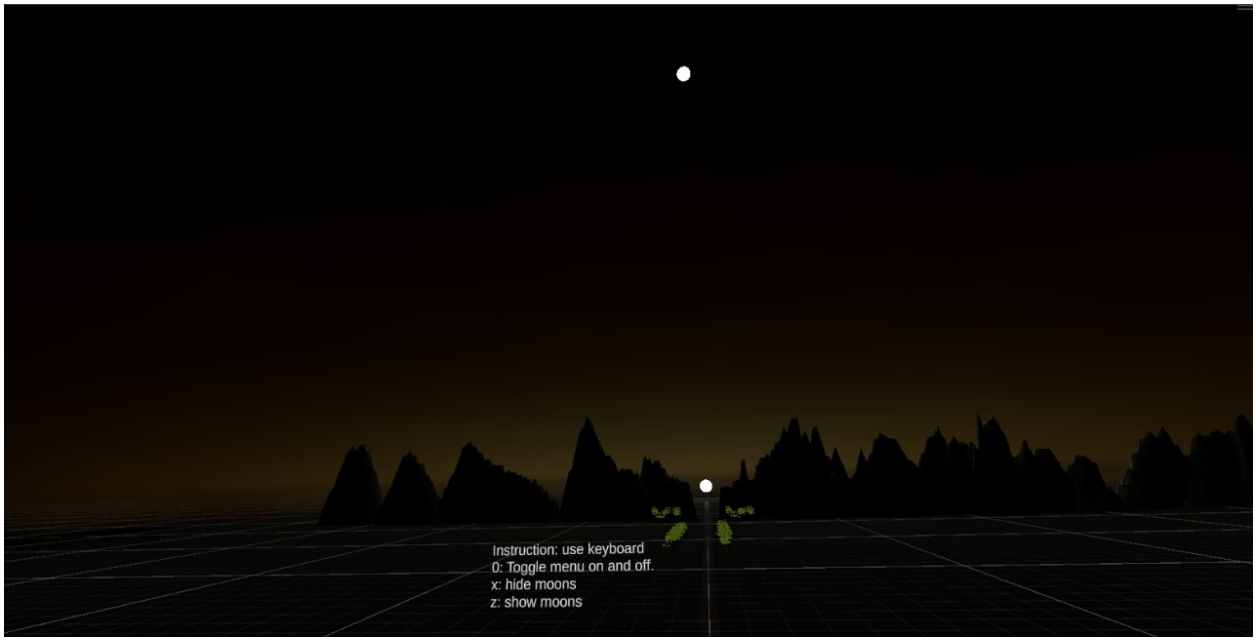
Snapshots of the implemented scenes:

The Optical Illusion:





Moon Illusion:



Ponzo Illusion:

Ebbinghaus Illusion:

Code Documentation

Optical Illusion:

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

public class ImageCanvas : MonoBehaviour
{
    public GameObject overlayImage; // Assign the first overlay image in
the inspector
    public GameObject spectrumImage; // Assign the second spectrum overlay
image in the inspector

    // Update is called once per frame
    void Update()
    {
        // Check if the space key is pressed
        if (Input.GetKeyDown(KeyCode.Space))
        {
            // Toggle the active state of both images
            overlayImage.SetActive(!overlayImage.activeSelf);
            spectrumImage.SetActive(!spectrumImage.activeSelf);

        }
    }
}
```

The script is designed to handle the visibility of overlay images in a scene. Here's a detailed documentation of the code:

Script Name
ImageCanvas

Dependencies
Unity Engine
System Collections

Purpose
This script toggles the visibility of assigned overlay images in a Unity scene when the space key is pressed.
Properties

- **public GameObject overlayImage:** This public variable is used to assign the first overlay image in the Unity Inspector. This image's visibility will be toggled.
- **public GameObject spectrumImage:** This public variable is used to assign a second overlay image, referred to as the spectrum image, in the Unity Inspector. This image's visibility will also be toggled.

Behavior

- **Update Method:** The script runs the Update method once per frame.
    - Within the Update method, it checks for the press of the space key (Input.GetKeyDown(KeyCode.Space)).
    - If the space key is pressed, the active state of the overlayImage and spectrumImage GameObjects is toggled.
    - The method SetActive is used for this purpose. The new active state is set to the opposite of the current state (!overlayImage.activeSelf).
    - This results in the images being visible when they were previously invisible, and vice versa.

show_hide script (used in both Ebbinghaus Illusion and Ponzo Illusion):

In this VR scene, the user can show or hide all the spheres surrounding the main spheres. The idea behind this capability is to allow the user to instantly break or go into the illusion. The script is presented below and an elaborate explanation will be given on the important parts of it.

What the script mainly does is create an addressable game object that will have the capability to be shown/hidden based on another addressable InputAction. Now, an elaborate explanation of each of the method's functionalities will be provided.

OnEnable: This function is to enable the input action when the script is enabled. The input action assigned through the inspector window.

OnDisable: This function is to disable the input action when the script is disabled.

Start: This function is just an initiation point for the running of the scene. So the call of the main function(ToggleObjectVisibility) is put there when the toggle action is performed.

ToggleObjectVisibility: This is the main function. It's very simple in its implementation. It starts out by checking if the targeted object is active or not, if it is, then it will be set to inactive if the toggle action is performed. Otherwise, if the object is not active, it sets it to active.

There was an issue here where if the object gets deactivated, it can't be activated back because a script can only be fetched if the object is enabled. This was avoided by making an empty game object to be used as only a script holder so that it would be independent of the target object being disabled.
It's also worth mentioning that all the spheres excluding the main ones, which are supposed to have the ability to be toggled off and on, are attached to a one empty parent object to allow them to be simultaneously deactivated or activated by connecting the script's functionality to this parent object.

```
using System.Collections;
using System.Collections.Generic;
using System.Security.Cryptography.X509Certificates;
using UnityEngine;
using UnityEngine.InputSystem;
public class show_hide : MonoBehaviour
{
    // Reference to the GameObject you want to show/hide
    public GameObject targetObject;
    // Define the input action for toggling visibility
    public InputAction toggleAction;
    void OnEnable()
    {
        toggleAction.Enable();
    }
    void OnDisable()
    {
        toggleAction.Disable();
    }
    void Start()
    {
        // Register the method to be called when the toggle action is performed
        toggleAction.performed += _ => ToggleObjectVisibility();
    }
    void ToggleObjectVisibility()
    {
        if (targetObject != null && targetObject.activeInHierarchy==true)
        {
            targetObject.SetActive(false);
        }
        else if(targetObject.activeInHierarchy==false)
        {
            targetObject.SetActive(true);
        }
        else
        {
            Debug.LogWarning("TargetObject not assigned!");
        }
    }
}
```

QuitGame script (used in all scenes):

This is a simple Unity script that allows the user to quit the scene by pressing the “Q” key

At first, a public variable **action is declared** of type **InputActionReference**. This variable is meant to hold a reference to an Input Action, which can be triggered by player input.

In the **Start** function, the **action** is enabled, and a callback is registered for when the action is performed. When the action is performed (e.g., a button press), it triggers the provided lambda expression. Inside the lambda expression, there's code to quit the game. If running in the Unity Editor, it stops play mode (**UnityEditor.EditorApplication.isPlaying = false;**), and if running as a standalone build, it quits the application (**Application.Quit();**).

The **Update** method continuously checks whether the "Q" key is pressed. If it is, the same code for quitting the game is executed.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;
public class QuitGame : MonoBehaviour
{
    public InputActionReference action;
    // Start is called before the first frame update
    void Start()
    {
        action.action.Enable();
        action.action.performed += (ctx) =>
        {
#if UNITY_EDITOR
                UnityEditor.EditorApplication.isPlaying = false;
#else
            Application.Quit();
#endif
        };
    }
    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Q))
        {
#if UNITY_EDITOR
                UnityEditor.EditorApplication.isPlaying = false;
#else
            Application.Quit();
#endif
        }
    }}
```

MovingTheMoon script:

This Unity script, named MovingTheMoon, appears to be designed to control the movement of a moon object in the scene. The moon can be toggled between two positions (moonAbove and moonBelow) based on user input. The script uses an input action and a coroutine to animate the moon's movement.

The variables:

- **moonAbove** and **moonBelow** are references to two GameObjects representing different positions of the moon.
- **startAction** is meant to hold an input action that triggers the moon movement. However, the registration of this action is missing in the provided code.
- **period** is an integer variable representing the time period for the moon movement animation.

The functions:

X_calculation: This function calculates the x-coordinate of the moon's position based on a given y-coordinate. It seems to be used to determine the moon's position along a circular path.

Coroutine: This coroutine function moves the moon object gradually from an initial position to another over a specified period of time. It uses the x_calculation function to determine the x-coordinate based on the current y-coordinate. The movement occurs in discrete steps, with a wait of 4 seconds between each step.

Update: The **Update** function checks if the "T" key is pressed. If so, it stops any ongoing coroutines (if there are any) and starts the **Coroutine** coroutine, initiating the moon's movement animation.

```
using System.Collections;
using System.Collections.Generic;
using System.Security.Cryptography.X509Certificates;
using UnityEngine;
using UnityEngine.InputSystem;
public class MovingTheMoon : MonoBehaviour
{
    // Reference to the GameObject you want to show/hide
    public GameObject moonAbove;
    public GameObject moonBelow;
    // Define the input action for toggling visibility
    public InputAction startAction;
    public int period = 5;
    void Start()
    {
        // Register the method to be called when the toggle action is performed
    }
    float x_calculation(float y)
    {
        Debug.Log(y + " " + y * y + " " + (Mathf.Pow(moonBelow.transform.position.x - 250f, 2f) +
Mathf.Pow(moonBelow.transform.position.y, 2f) - Mathf.Pow(y, 2f)));
        return 250f - Mathf.Sqrt(Mathf.Pow(moonBelow.transform.position.x - 250f, 2f) +
Mathf.Pow(moonBelow.transform.position.y, 2f) - Mathf.Pow(y, 2f));
    }
    IEnumerator Coroutine()
    {
        int temp = period;
        float init_y = 761f;
        moonAbove.transform.position = new Vector3(x_calculation(init_y), init_y, 0);
        // start the action
        while (temp >= 0)
        {
            //yield on a new YieldInstruction that waits for 4 seconds.
            moonAbove.transform.position = new Vector3(x_calculation((init_y - 40) / period * temp +
40), (init_y - 40) / period * temp + 40, 0);
            yield return new WaitForSeconds(4);
            temp--;
        }
    }
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.T))
        {
            //Start the coroutine we define below named Coroutine.
            StopAllCoroutines();
            StartCoroutine(Coroutine());
        }
    }
}
```

The Project Build:

The project is made of 4 scenes and each scene has its own build. So, 4 builds are attached to the submission. The reason 4 scenes are separate is that we assign different scene for each member so we don’t have the same project.