## Installation/ Setup

Here is the link to downloading Unity Hub for Mac/ Windows:
https://unity.com/download

- You must install an Unity editor before using Unity.
- Install both the Unity Hub and Unity editor in the same directory to avoid issues
- Click on new projects to get started and choose one of the following templates.

## Unity UI

**Scene View** - the center that shows the game/ perspective (located in the top left of the game screen).
**Game View** - the option that changes the screen into the camera/player POV (located next to the scene view option).
**Hierarchy** - Almost like an inventory list. In a game, the elements are rendered from top to bottom in a 2D game.
**Project Window** - Almost like a warehouse. Stashed away for future use (Located on the bottom).
**Inspector** - You can modify the zooming, transforms, etc (Located on the right side of the screen).

- Mesh Filter - Basically the type of shape.
- Mesh Renderer - Tells the GPU how to render the shape (whether it should receive shadows, etc.)
- Box Collider - Physics aspect of the game object.

**Game Object** - Everything in unity is a game object. It's like adding sprites in scratch. It is a collection of transforms and components.

- In the basic example, we right click on the hierarchy (left margin of the screen) and add in a 3d object called a cube. This cube is an example of a game object.

- This can be seen from the game view as any game object added is automatically updated to the game view.
- **Transform** - Position, rotation, and scale of a game object.

**Unity's Coordinate System**:

- 3D coordinate system with (x, y, z).
- Z - forward and the camera is always looking through the z-axis.

**Toolbar**:

- Located on the top left corner of the UI.
- There are many tools such as the move tool, scale tool, and rotate tool.
- With the keyboard:
  - W - move tool
  - E - rotation tool
  - R - scaling tool

## <u>Creating A Basic Game</u>

### Assets Folder

- Take care of the color, properties of game objects.
- To be organized, create a folder named materials to hold these properties.
- Right click, and create new material. Once you have changed some of the material's attributes, you must drag and drop the material from the project window onto the project window.
- Using the hierarchy, you can move a gameobject to be within another gameobject (child).
  - This will make the child gameobject be a part of the parent gameobject.
- **Prefabs** - Useful to reuse gameobjects for other projects.
  - Simply drag the gameobject from the hierarchy into the prefabs folder (you need to create a new folder in the project window.)

### Saving a copy

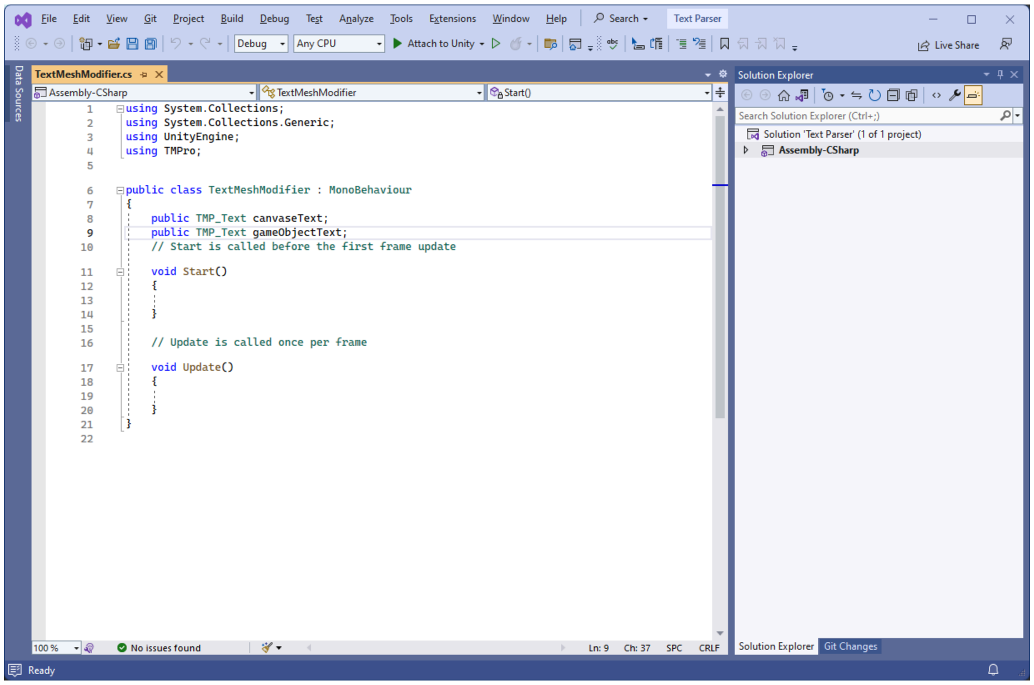
- Assets > Export Package
- Give this package folder a new name.

- This can be readily imported into a brand new project by:
  - Assets > Import Package > Custom package

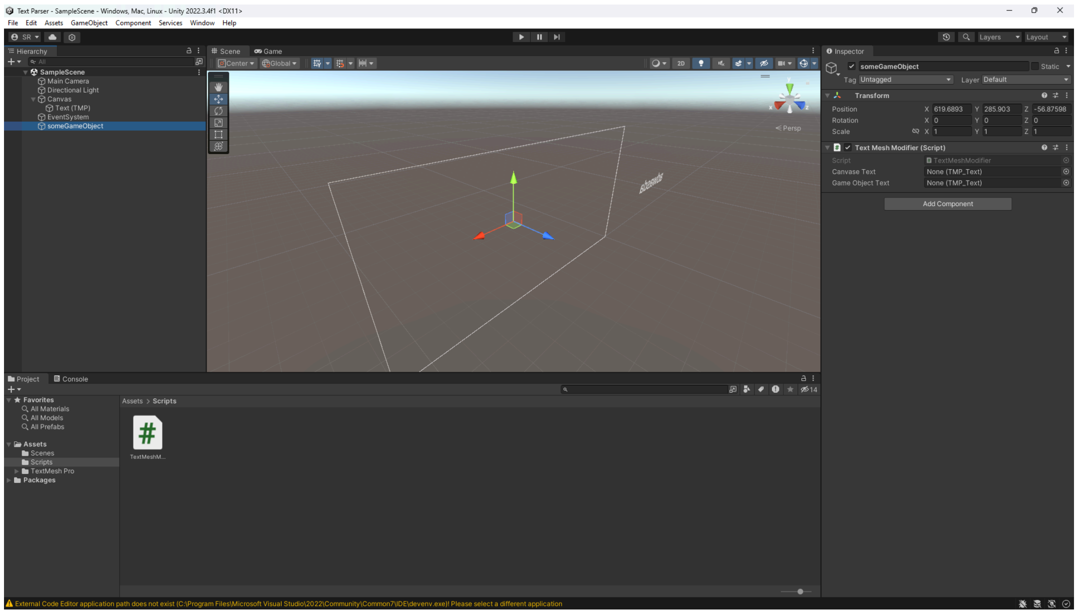
**Scripting**

- When coding a custom script for an object, you must drag the script onto the object in the hierarchy (class name must match file name in Unity).
- The Start() method only runs once when the play button is pressed.
- The Update() method is called on periodically per frame.
  - You should have a keystroke/ mouse click in this.
- The FixedUpdate() function runs at 100 hz -> meaning it will update 100 times per frame.
  - You should not put any keystrokes/ mouse clicks here.
  - You should place gameobject physics in this function.
  - You should create a boolean variable to keep track of key presses:
    - Update boolean variable in the Update() function and use this boolean function in the FixedUpdate() function to perform any physics related motion.
- Printing in the log for debugging purposes:
  - Debug.Log("Enter some string message")
- Device/ Keyboard input:
  - Input.GetKeyDown(KeyCode.UpArrow) -> GetKeyDown() means while the specific key is pressed. You need to provide which key in the parameter KeyCode.<choose key>
- Get gameObject:
  - GetComponent<RigidBody>()
  - Add a function after to make the object move/ perform physical motion.
  - GetComponent<RigidBody>().AddForce(provide a Vector3 attribute, provide a ForceMode):
    - Eg. Vector3.up, Vector3.down
    - Eg. ForceMode.Acceleration, ForceMode.VelocityChange
  - Vertical motion (jumping)
    - Get the rigidbody and then add a force with a Vector3.up parameter and a ForceMode.VelocityChange parameter.

    - Vertical motion
        - Better to use axes as we don't have to use specific keystrokes for horizontal motion.
- Using TextMeshPro (ARISE Lab Project):
    - Creating TextMeshPro Objects:
        - Right click on the hierarchy and press UI -> select Text-TextMeshBro
        - This will create a canvas with a child TextMeshPro object.
        - This uses the canvas renderer
    - Another way to do it is without the canvas:
        - Right click again in the hierarchy, create a 3d object.
        - Select TextMeshPro object.
        - This uses a mesh renderer like all 3d models.
    - Using the namespace TMPro
    - The type is (TMP_Text)
        - Give it public visibility as we are accessing the inspector.
        - If you copy and paste the code and the fields don't show up in the inspector, you should delete the fields and rewrite it again (to refresh).
        - Remember to drag the script into an empty gameobject first.
- Creating a Scrollable Text UI:
- Right click on the canvas object, hover over the UI, and select a scrollable view.
- Click content, which would be nested under the scrollable objects, and add a TextMeshPro UI component.
- Add this component to the text modifying script.

- 

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

public class TextMeshModifier : MonoBehaviour
{
    public TMP_Text canvaseText;
    public TMP_Text gameObjectText;
    // Start is called before the first frame update

    void Start()
    {

    }

    // Update is called once per frame

    void Update()
    {

    }
}
```

- 

```
Inspector
someGameObject
Tag Untagged    Layer Default
Transform
Position   X 619.6893   Y 285.903   Z -56.87598
Rotation   X 0          Y 0         Z 0
Scale      X 1          Y 1         Z 1
Text Mesh Modifier (Script)
Script             TextMeshModifier
Canvase Text       None (TMP_Text)
Game Object Text   None (TMP_Text)
```

```
External Code Editor application path does not exist (C:\Program Files\Microsoft Visual Studio\2022\Community\Common7\IDE\devenv.exe)! Please select a different application
```

- The variables appear in the inspector because it's public.
- Drag the TextMeshPro Objects into the null fields of the empty game object you created.
- In VR, the text seems to be mirror-imaged. To fix this, change the x scaling attribute to negative 1. This flips the text.

## Virtual Reality Setup

- https://developer.vive.com/resources/viveport/sdk/documentation/english/viveport-sdk/integration-viveport-sdk/unity-developers/
- Items needed:
    - Vive Headset
    - Base Station
    - Not required, two VR controllers
- Import SteamVR library (Asset store -> type in SteamVR).
    - Crucial
    - Without this, we cannot get it to work, the Vive headset will not connect to Unity.
- Go to Unity
    - Open project settings, install XR-plugin management.
    - Install XR interaction toolkit (make sure to install the latest version as it has all the prefabs and updated libraries).
        - You can do this by providing the c# com.<package name>
    - Make sure to check the box for oculus in XR-plugin management.
- To get the camera, go to Assets -> search up [Camera Rig]
    - This is needed for perspective.
    - Without this, the whole scene along with the plane will move when the person turns their head.
- Add <Steam_VR_Play_Area> for motion.

## Data from Eye Tracker - Setup

- Open Edit > Project Settings > XR Plug-in > OpenXR
    - Add Eye Gaze Interaction profile.
- Must create an action map
    - Right click on the assets project window and press create input actions
- Under actions, create eyePose.
    - Click on eyePose and set Action Type to Pass Through
    - Click on Control Type to select Pose
- Now right click on the actions column and add an binding

- Set the path to pose [Eye Gaze {OpenXR}]
- Link to how to use input system: https://developer.vive.com/resources/openxr/openxr-mobile/tutorials/unity/basic-input-openxr-mobile/original-basic-input-openxr-mobile/
- Link to specific attribute keys:
- https://docs.unity3d.com/ScriptReference/XR.Eyes.html
- Link to explanation of functions that are used to access attributes in the XR interaction debugger tool:
- https://docs.unity3d.com/Manual/xr_input.html
- The eye tracking data shows up in the XR Interaction Debugger
    - Window > Analysis > XR Debugger
    - Now, we need to retrieve this data to be used for analysis
- Retrieving the data:
    - We need to create any empty game object to have a script component that successfully retrieves the data from the input device.

**Explanation of the code and certain keywords:**

```
InputDevice device = InputDevices.GetDeviceAtXRNode(XRNode.Head);
```

- The device object is created in order to access the VIVE headset

```
Vector3 leftEyePosition, rightEyePosition
```

- Both attributes are type Vector3 (0, 0, 0)
    - Vector3 type is also used for moving game objects up and down from prior examples.

```
 if (device.TryGetFeatureValue(CommonUsages.leftEyePosition, out leftEyePosition))
     {
        Debug.Log("Left Eye Position: " + leftEyePosition);
     }
```

- **TryGetFeatureValue()** attempts to access the current value of a feature, and returns:
    - true if it successfully retrieves the specified feature value
    - false if the current device doesn't support the specified feature, or the device is invalid (that is, the controller is no longer active)
- The Unity XR documentation says to use CommonUsages to retrieve any data regarding eyes.