Aimee Valladares

CAP 4122

25 February 2022

# VR User Interface Report

## Specifications

* Unity Version:
* VR Hardware: Oculus Quest 2 headset + controllers
* VR Software: Oculus Unity SDK and OpenXR
* Certain File Locations:
  + Scene Build – Assignment 2/Build/Assignment 2
  + Button Script – Assignment 2/Assets/ButtonChange.cs

## User Interface Design

The design I chose was to be reminiscent of a more modern layout for an expansive menu that is sleek, easy to understand, and clean. Specifically, the canvas is comprised of multiple panels which are separate components as opposed to a single image with all the elements simulating a typical menu. All the panels utilize a dark image for the panel background to make it much more distinguishable from the brighter environment. The title panel utilizes a different material that makes it seem a little more transparent to help distinguish it from the other more interactable panels. The title panel also utilizes a different heading style from the other panels by using a light blue font color. The other panels that contain interactable elements utilize a solid black image as the background while using light text and buttons to create contrast and make the elements more readable for the user. For the subpanels, I chose a different heading style with the orange text color to illustrate a hierarchy with the panels.

Additionally, the VR environment utilizes a form of software and hardware basedlocomotion that makes movement smoother for the user as they can simply walk slightly forward in real-life to move in the environment due to headset sensors. Though this can be an issue with running into obstacles in the real world, the process of setting up and connecting the Oculus Quest 2 reduces this issue. Through the process of setting up the headset, the system has the user *manually* create a designated area for movement with the headset that creates a clear boundary for the user. This enables the headset to track the controllers as well as even track the user’s hands to ensure that the user is not going past the boundary they created when immersed in the environment.

For my scene, I created a small and simple environment to help make the environment a little more immersive and give the user a frame of reference with common objects that the user can recognize. Specifically, the green plane serves as the floor to help give the user a sense of how big they are in the scene. Meanwhile, the 3D objects in the scene are meant to be a table, a stack of cans, and a few balls. Having the scene in the background gives the user more of a sense of scale and distance without being disorienting. The user interface design can be seen in **Figure 1**.

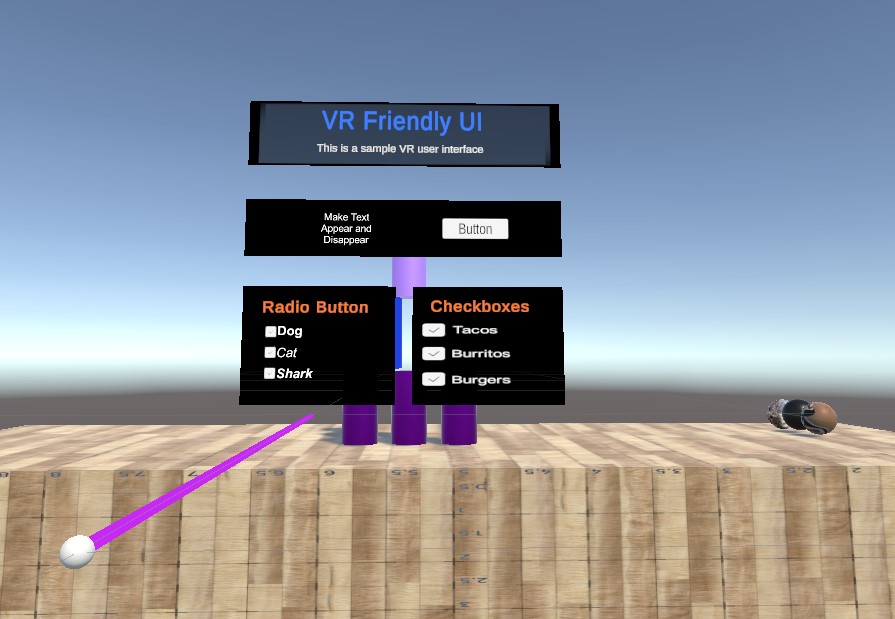
## User Interface Elements

The interactable elements in my UI involve a button, multiple radio buttons, multiple checkboxes, and a laser pointer. On one panel, the VR button has a standard light gray layout that is slightly resized to be larger to allow more room for the user to press it with the reticle. The button functionality is to have the text on the left side of the button appear and disappear when it is pressed.

On a separate panel, there are 3 different radio buttons that allow the user to only choose one option at a time. The options are Toggle objects that are placed in a toggle group. To illustrate different style options for text. One radio option is bolded, another radio option is italicized, and the final radio option is both bolded and italicized. When an option is pressed, the button will change color drastically to indicate which option has been selected.

On a separate panel, there are 3 different check boxes that allow the user to choose and enable multiple options at the same time. The buttons are also Toggle objects (not placed in a toggle group) so the button and text layout are like the radio buttons. The labels for the checkboxes are normal text that has not been decorated. Unlike the radio buttons, there is no huge shift in color when toggling options. Though it can still indicate the option state with the use of white and gray.

Finally, the reticle that tracks the user's controller for the user's movement is a key user interface element. The reticle is how the user can interact with the VR environment and all the button actions. The reticle has a sphere at the end to indicate where the user can interact with an object as well as a laser pointer to make it easier for the user to track their hand/controller position and navigate the menu. In the UI, the reticle can be seen when hovering over the four menu panels while the laser pointer can be seen all throughout the scene. The user interface design and elements can be seen in **Figure 1**.

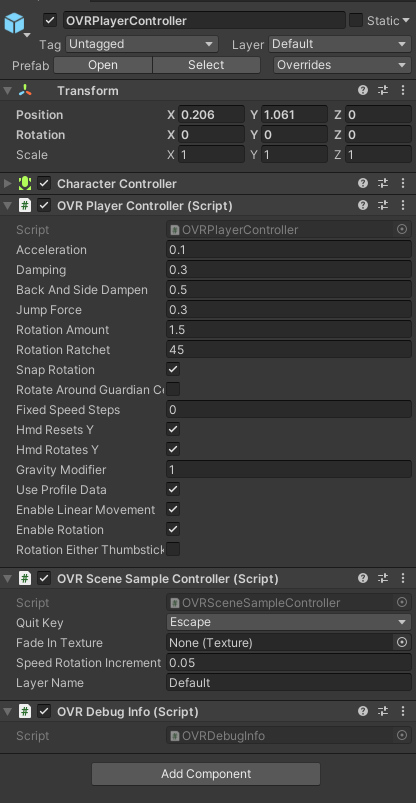


**Figure 1:** Complete User Interface Design and Elements in Scene

## Implementation

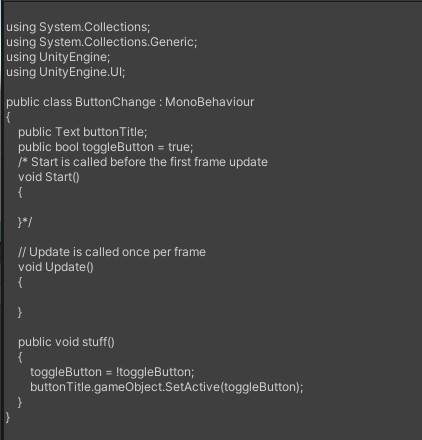
The scene in unity consists of 5 major components: the Directional light, Background Scene, OVRPlayerController, Canvas, and the UIHelper. The directional light and the main camera are automatically generated by Unity. However, I deleted the main camera to avoid issues with running the application. The directional light helps add dimension to the scene by adding shadows to objects in the background scene. The background scene is the collection of 3D objects and a plane in the environment that is meant to simulate a possible VR environment.

The OVRPlayerController is a prefab that allows the user to interact and view the environment in VR. In order to access and enable the prefab, I needed to install the Oculus SDK and integration plug-in. Then I needed to ensure that the various OVR components were enabled. This can be seen in **Figure 2**.

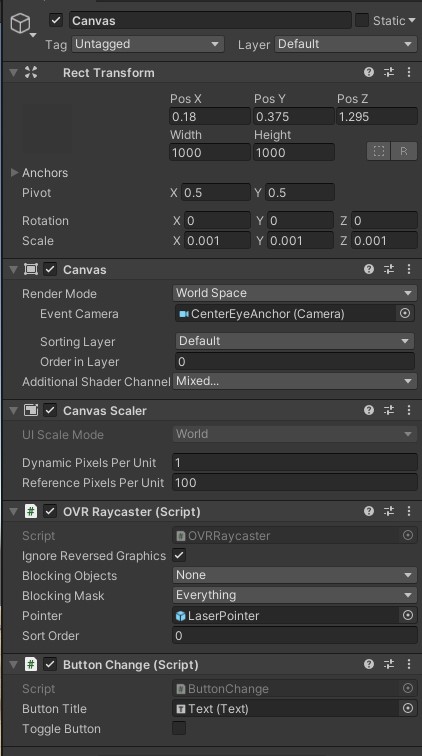


**Figure 2:** OVRPlayerController Inspector View

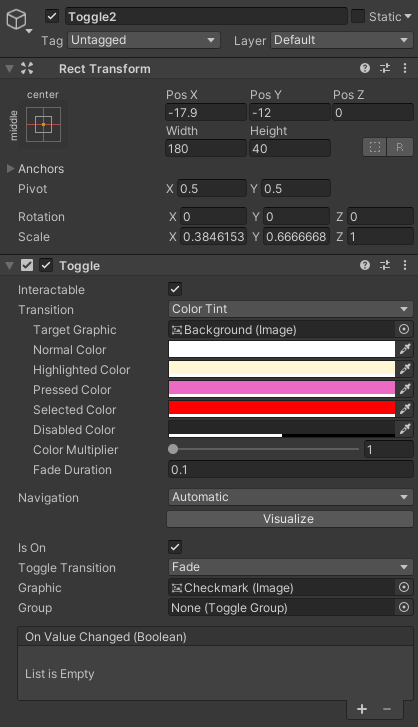
The Canvas contains all the components and game objects for the user interface. For all the panels, I added an image to house all the other objects in the user interface. For the button, I created a script named *ButtonChange.cs* to enable the text to appear/disappear functionality. The button would call on the *stuff()* method that would change the bool value for *toggleButton* variable and then have the game object activate or deactivate based on the bool value. In the inspector view for the Canvas, I would set the Button Text to be the object as the text that appears and disappears. These can be seen in **Figures 3 and 4**. For the Radio buttons, I added various *Toggle* objects into the panel and added the Toggle objects (Toggle1, Toggle2, and Toggle3) into a toggle group to ensure only one option can be chosen. For the Checkbox buttons, I added more *Toggle* objects into the panel. However, I did not add Checkbox objects (Check1, Check2, and Check3) to a Toggle group to ensure that multiple buttons could be enabled. This can be seen in **Figures 5 and 6**.



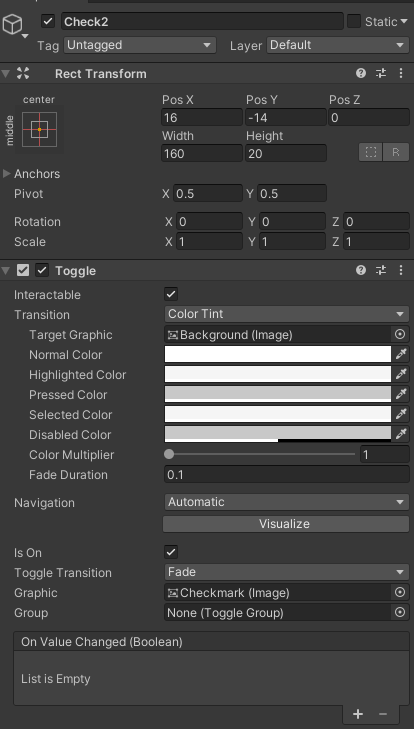
**Figure 3:** ButtonChange Script



**Figure 4:** Canvas Inspector View

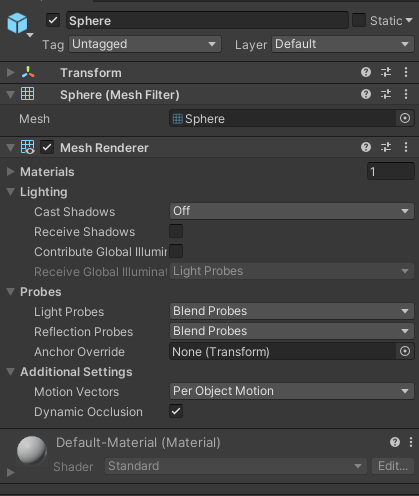


**Figure 5:** A Radio Button Option Inspector View

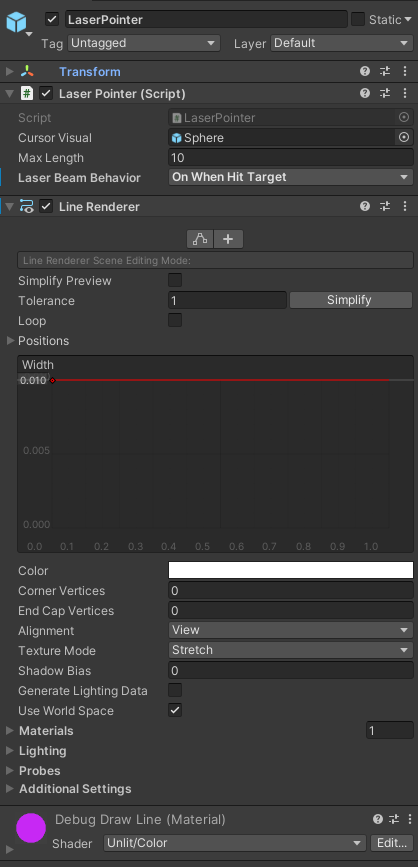


**Figure 6:** A Checkbox Option Inspector View

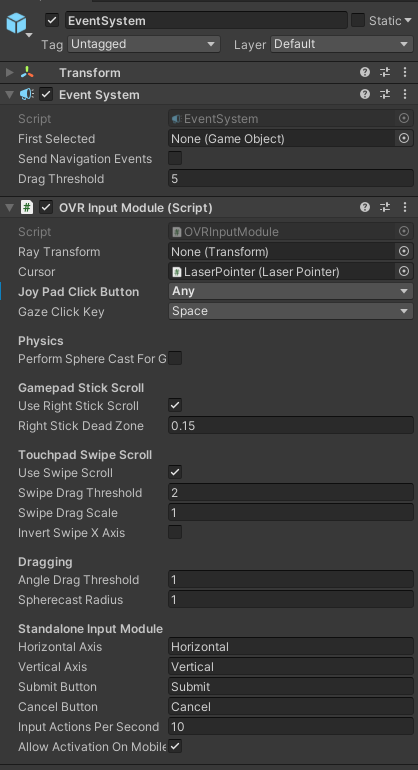
Finally, the UIHelper is a prefab that is meant to serve as an indicator for the player hand movement with the controller. Additionally, this enables the user to interact with the virtual elements in the environment. The UIHelper consists of a sphere, laser pointer, and event system. The sphere serve as a reticle for the user and helps indicate what elements in the environment are interactable. For instance, in the scene, the sphere should only appear in the panels in the UI and not in the background scene. The laser pointer is a cursor that tracks the controller’s movements and is set in the EventSystem. Finally, the EventSystem is responsible for handling events and inputs from the user. This makes the environment interactable. Any other event system that automatically generated was deleted to ensure the program would run appropriately. This can be seen in **Figures 7, 8, and 9**.



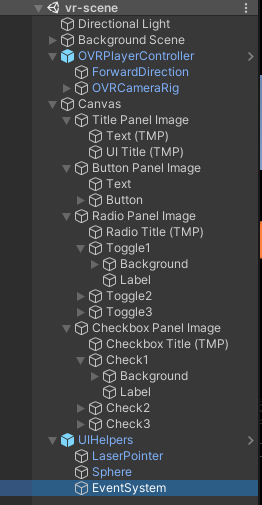
**Figure 7**: UIHelper – Sphere Inspector View



**Figure 8:** UIHelper – Laser Point Inspector View



**Figure 9:** UIHelper - EventSystem Inspector View



**Figure 10:** Overall Scene Components Hierarchy