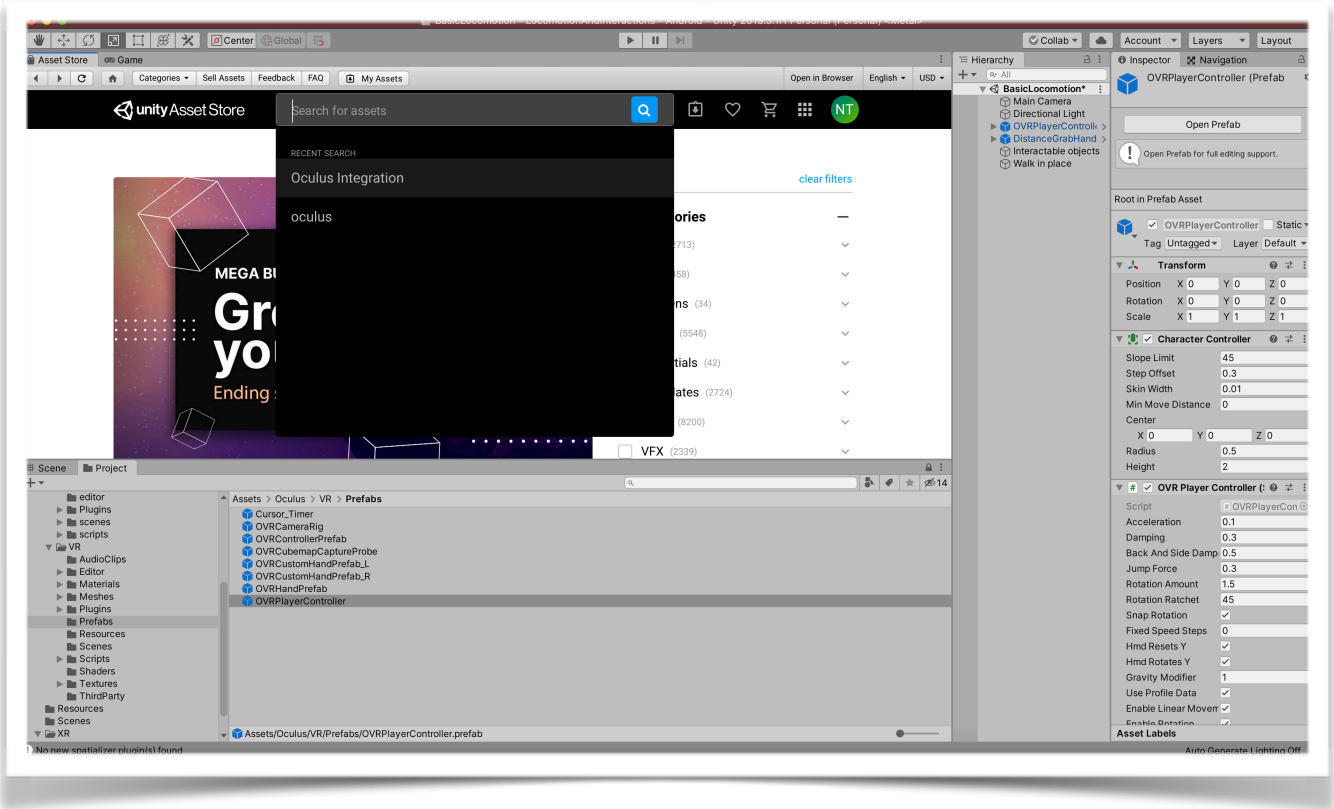
Tutorial locomotions and Interactions:

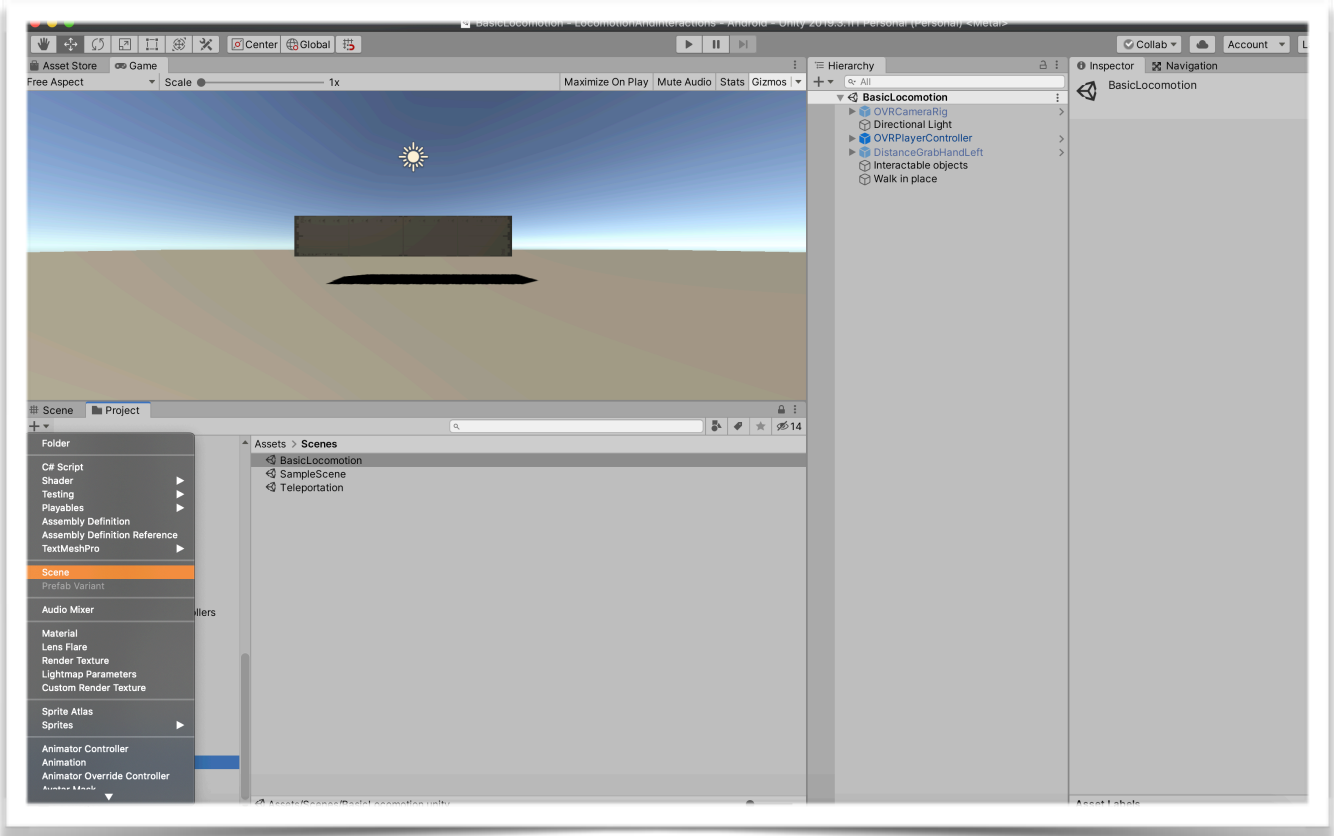
1. Linear Locomotion

Step 1:

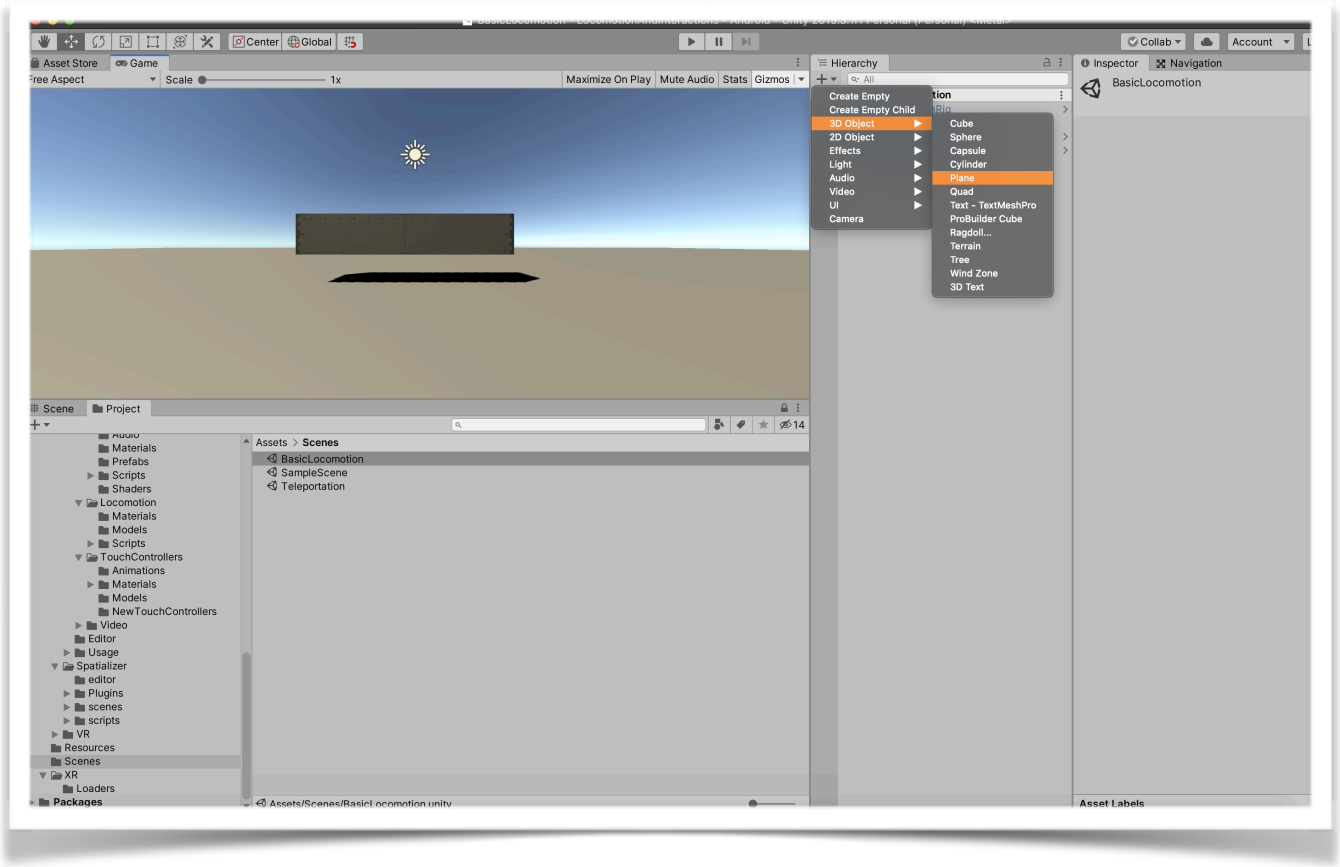
The linear locomotion techniques basically performs linear translational and rotational movements. It can be easily achieved by the Oculus integration SDK without coding. Let's get started! First, please download the **Oculus integration SDK In the asset store:**

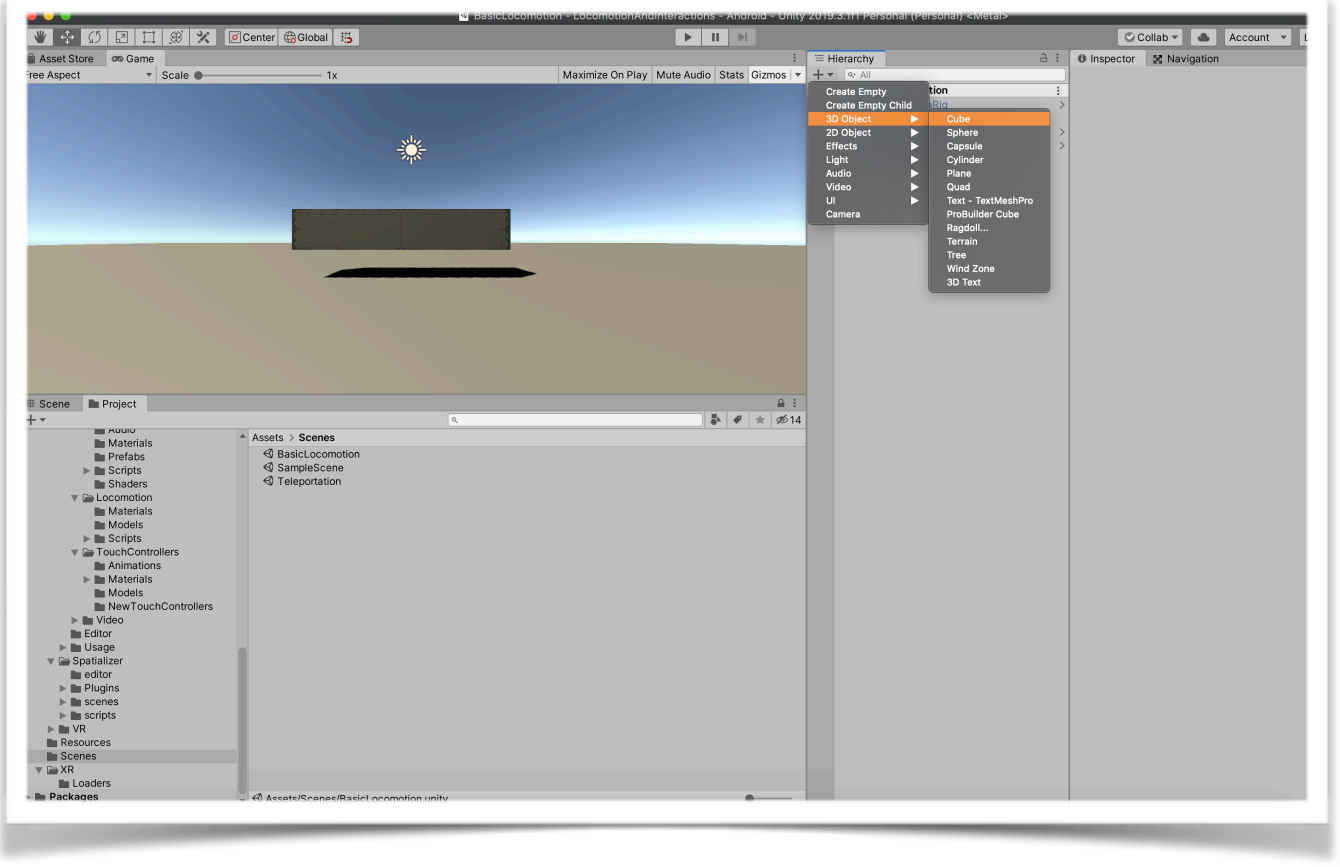


Step 2: Create an empty scene with the + button from the project panel. Name whatever you want.

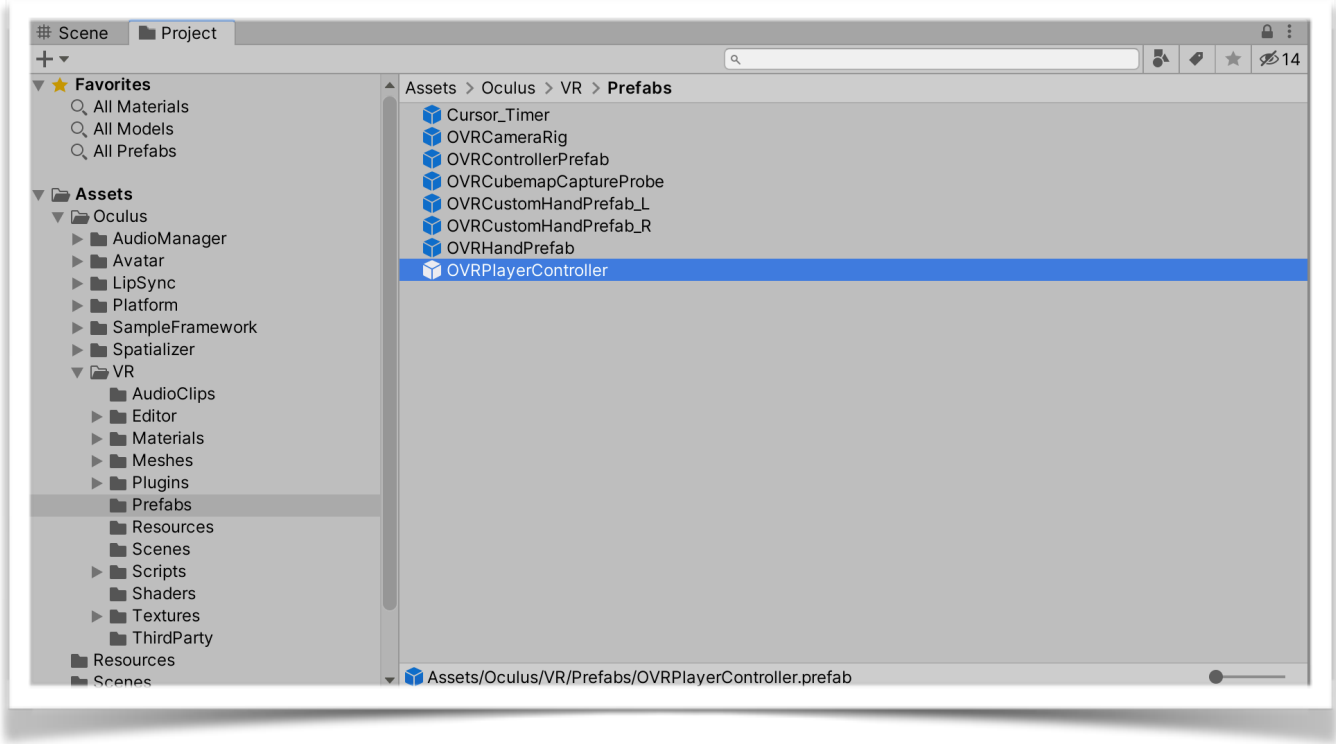


Step 3: In the hierarchy panel, use the + button to create the floor and place some object as indicators in the scene. You can be creative.

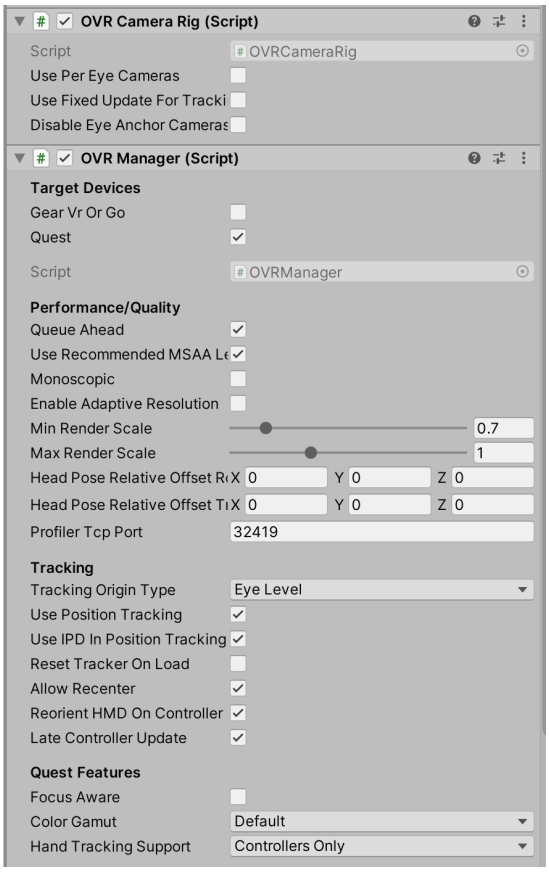




Step 4: If we check the Oculus SDK package-> VR - > Prefabs: You can found the prefabs that we can use to create basic interactions and locomotions. The most basic two are OVRCameraRig and OVRPlayerController.



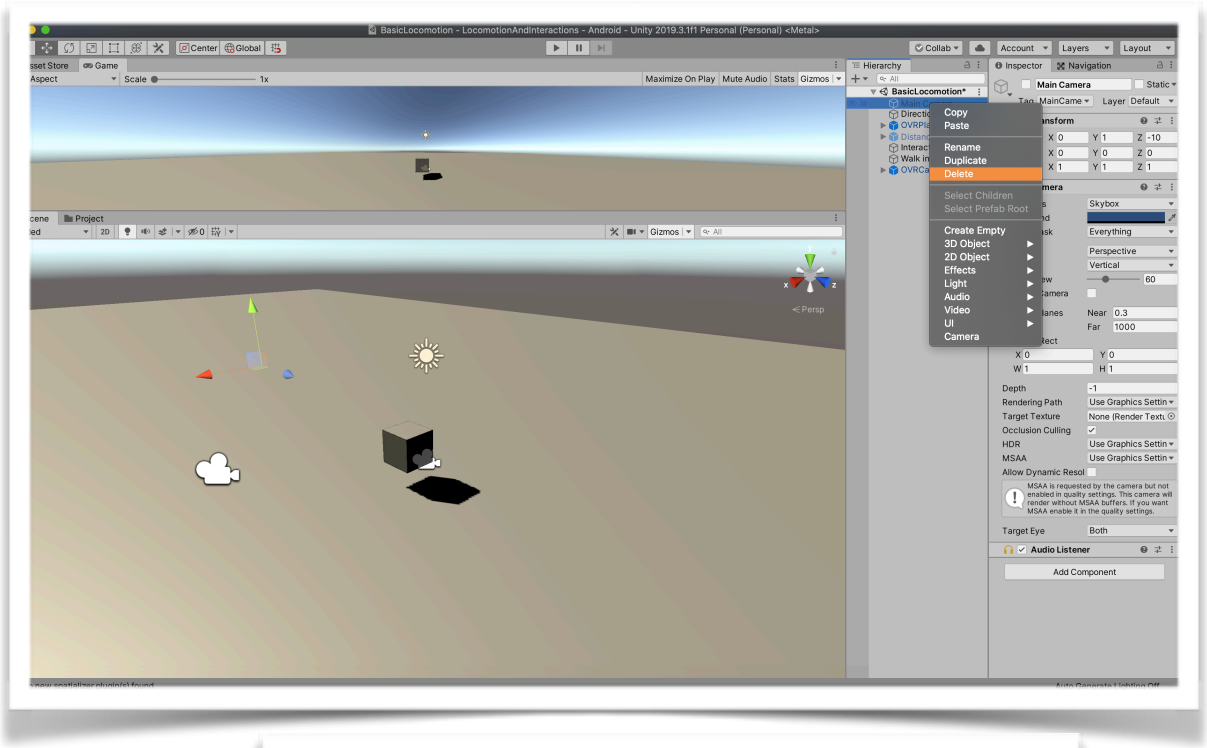
OVRCameraRig
is a custom VR camera that may be used to replace the regular Unity Camera in a scene. Drag an OVRCameraRig into your scene and you will be able to start viewing the scene.
The primary benefit to using OVRCameraRig is that it provides access to OVRManager, which provides the main interface to the VR hardware. For example, in the inspector, we can check if the device is Quest or Go.

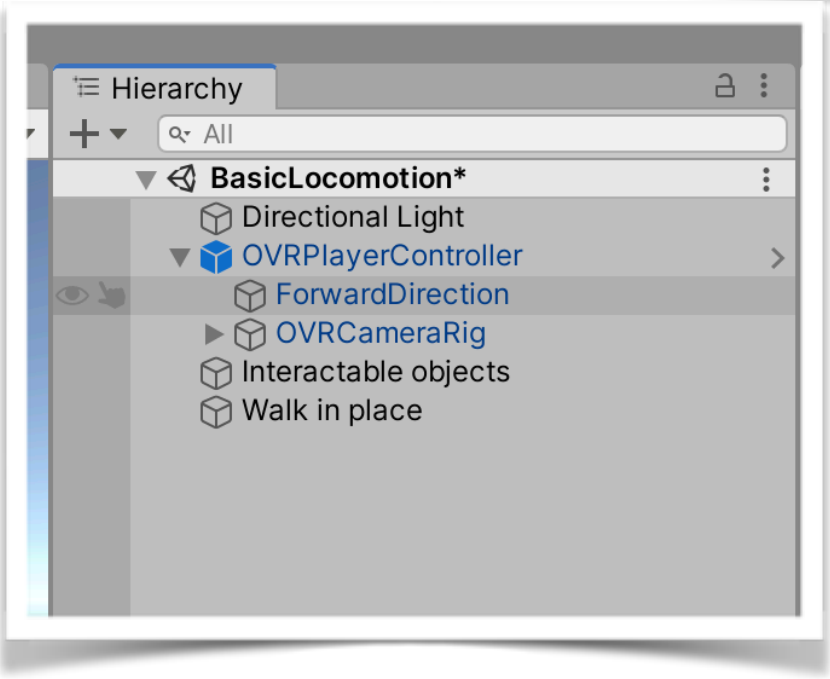


OVRPlayerController

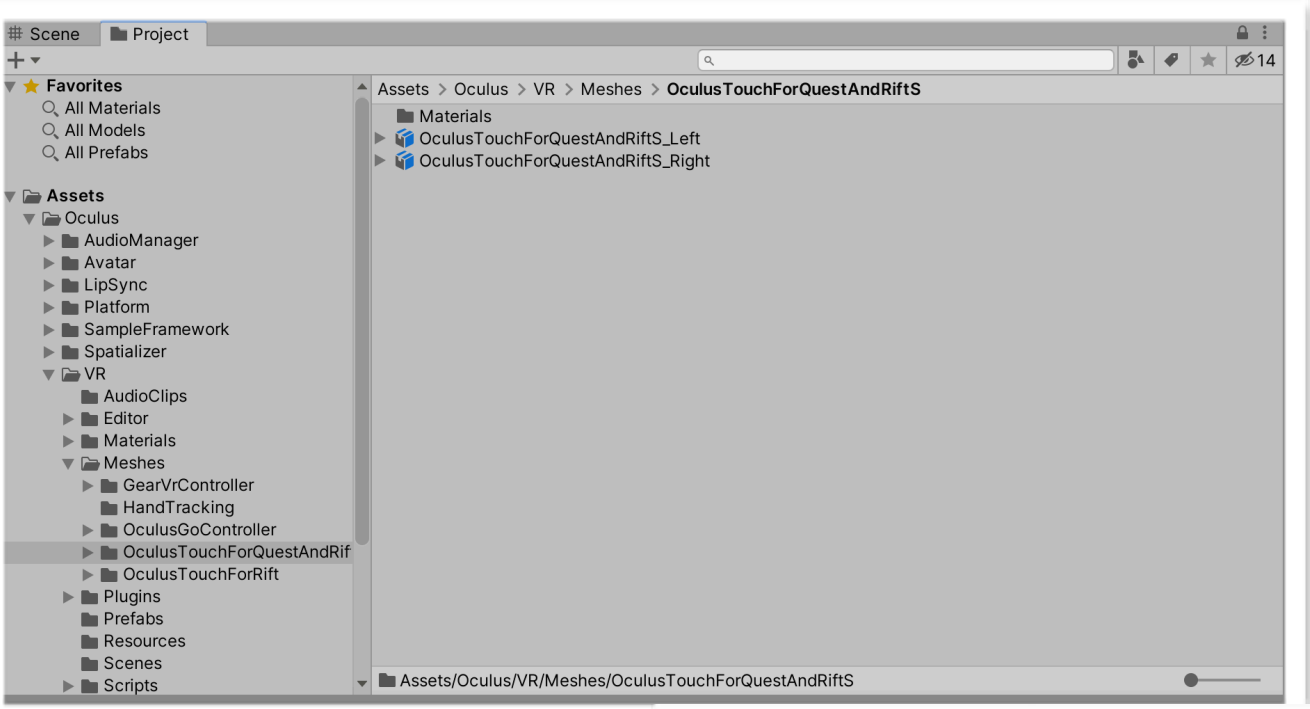
The OVRPlayerController is the easiest way to start navigating a virtual environment. **It is basically an OVRCameraRig prefab attached to a simple character controller.** It includes a physics capsule, a movement system, a simple menu system with stereo rendering of text fields, and a cross-hair component. **Hence, we can simply drag the prefab in the scene, and move around with the controllers.**

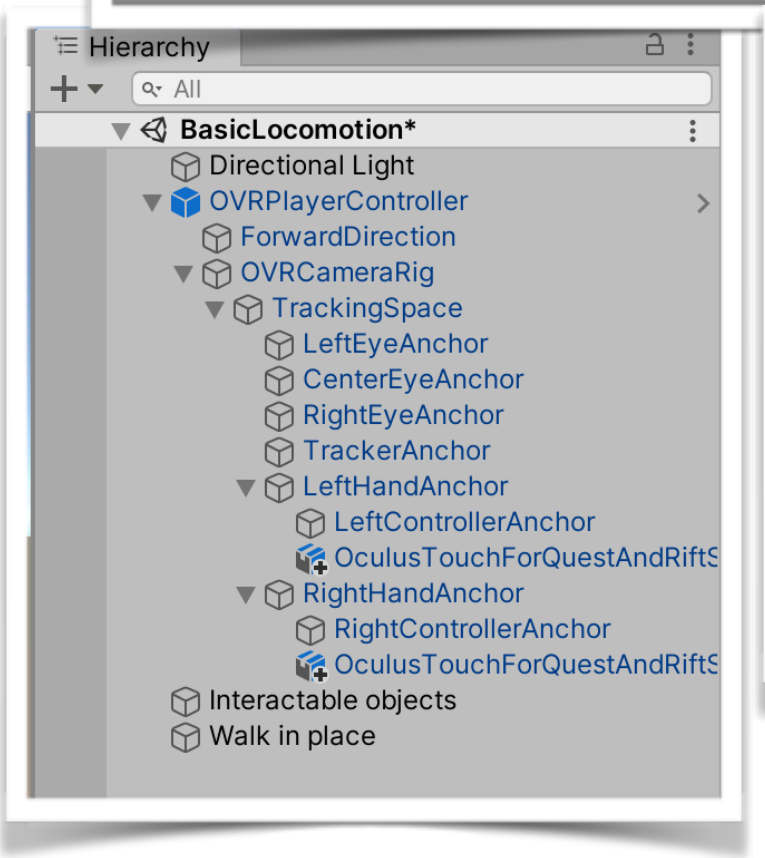
Note: Since the OVRPlayerController already have a camera with it. You can delete the original MainCamera in the scene.
Make sure you only have one camera in the scene(Unless you have other usages like 2D mini maps)

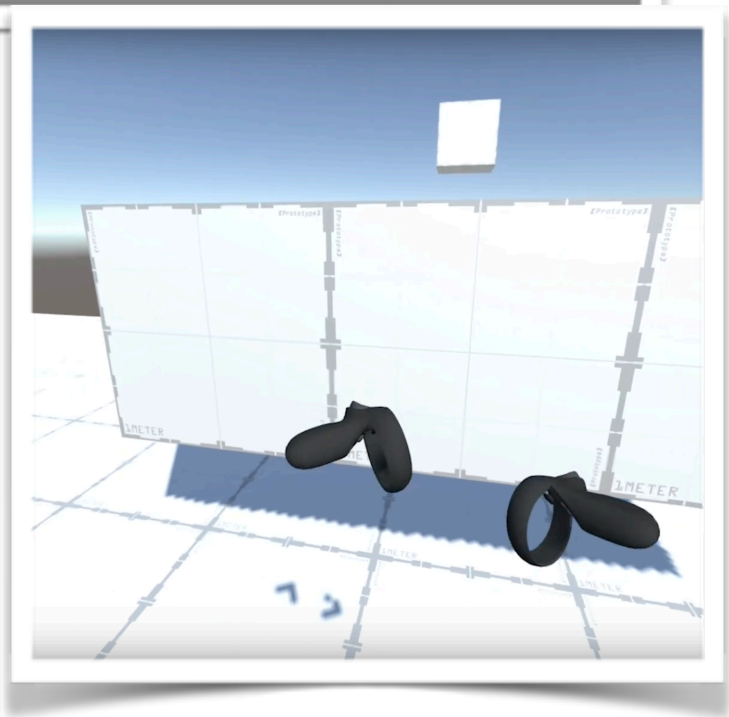




Step 5: To make the oculus quest controllers visible in the scene, You can add the model prefabs attached to the Left and right hand anchor as a Child. See the pictures below:



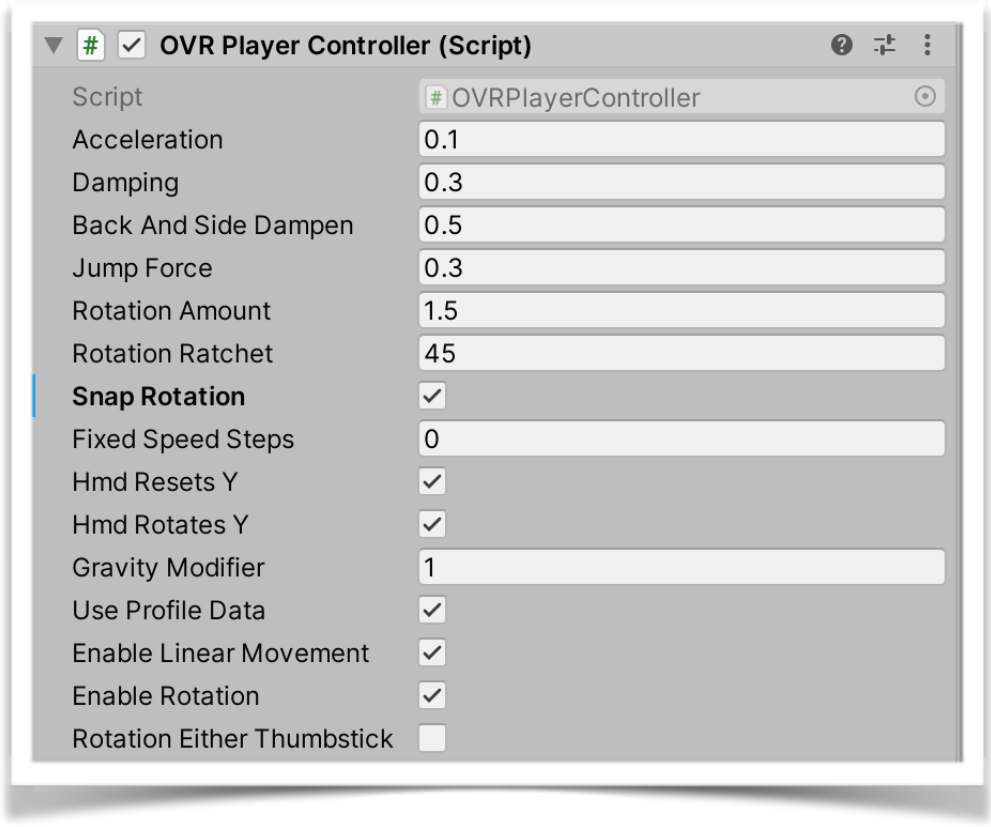




Now, you should be all set!!!

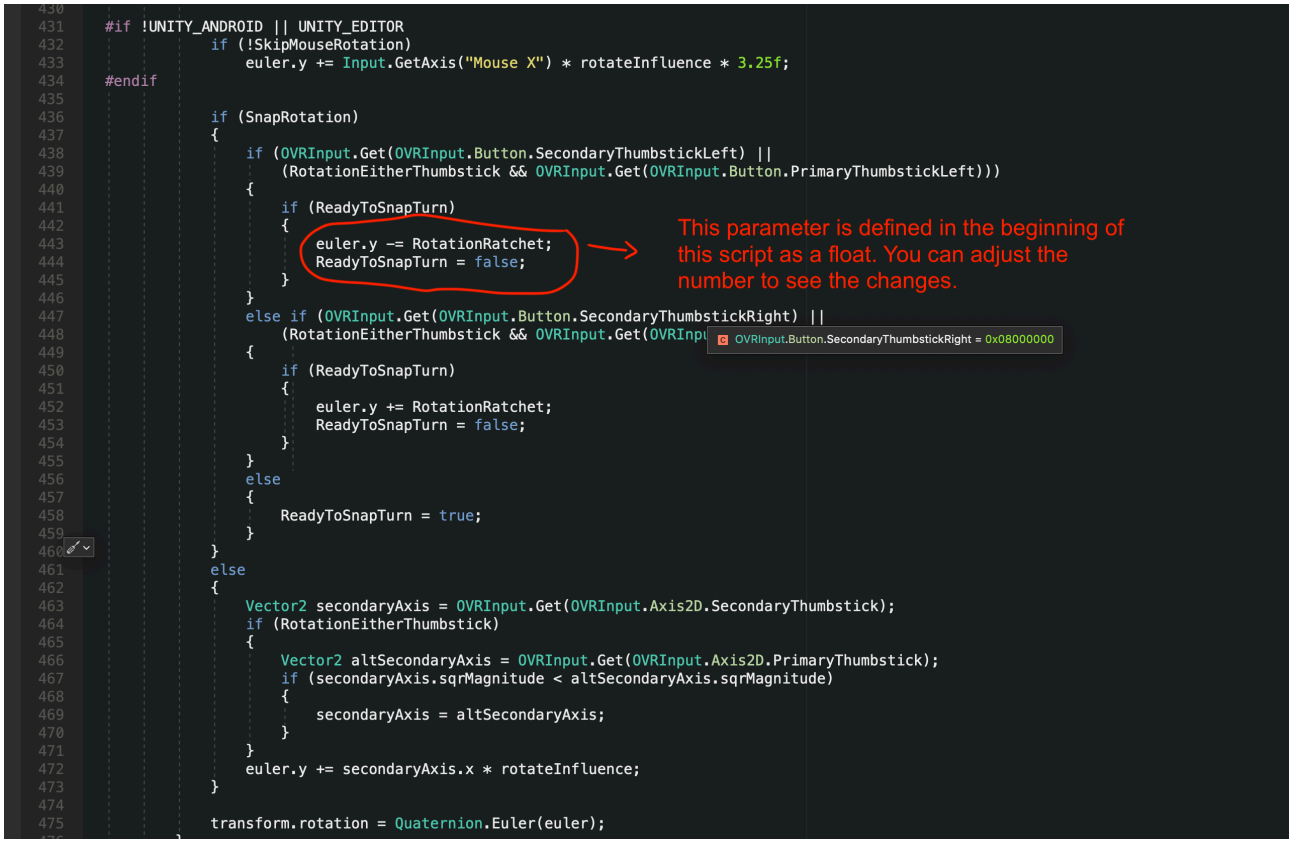
The right controller is to turn and the left one is to move.

Step 6: Finally, let's take a closer look at the details in the main script: OVRPlayerController. There're a lot of parameters you can modify and try freely. One specific parameter we mentioned here is the SnapRotation.



With this parameter toggled off. You will have the Smooth Turn, which could offer you more detailed information of the scene during turning. However, to user who is sensitive of cybersickness, SnapTurn is much better than the Smooth Turn. With this parameter toggled on, you can have the so-called Snap Turn, Similar to the next locomotion techniques that we will implement, it has a discrete nature. Try to modify the RotationRatchet Parameter to play.

Open the script OVRPlayerController.cs. The code on this capture is the section handling the snap rotation set on the previous picture. The RotationRatchet Parameter defines the degree that you will turn each time you press the button.




```
#if !UNITY_ANDROID || UNITY_EDITOR
            if (!SkipMouseRotation)
                euler.y += Input.GetAxis("Mouse X") * rotateInfluence * 3.25f;
#endif

            if (SnapRotation)
            {
                if (OVRInput.Get(OVRInput.Button.SecondaryThumbstickLeft) ||
                    (RotationEitherThumbstick && OVRInput.Get(OVRInput.Button.PrimaryThumbstickLeft)))
                {
                    if (ReadyToSnapTurn)
                    {
                        euler.y -= RotationRatchet;
                        ReadyToSnapTurn = false;
                    }
                }
                else if (OVRInput.Get(OVRInput.Button.SecondaryThumbstickRight) ||
                    (RotationEitherThumbstick && OVRInput.Get(OVRInpu
                {
                    if (ReadyToSnapTurn)
                    {
                        euler.y += RotationRatchet;
                        ReadyToSnapTurn = false;
                    }
                }
                else
                {
                    ReadyToSnapTurn = true;
                }
            }
            else
            {
                Vector2 secondaryAxis = OVRInput.Get(OVRInput.Axis2D.SecondaryThumbstick);
                if (RotationEitherThumbstick)
                {
                    Vector2 altSecondaryAxis = OVRInput.Get(OVRInput.Axis2D.PrimaryThumbstick);
                    if (secondaryAxis.sqrMagnitude < altSecondaryAxis.sqrMagnitude)
                    {
                        secondaryAxis = altSecondaryAxis;
                    }
                }
                euler.y += secondaryAxis.x * rotateInfluence;
            }

            transform.rotation = Quaternion.Euler(euler);
```