AFFECTIVE COMPUTING AND INTERACTION IN VR

Corso Realtà Virtuale 2022/2023

susanna.brambilla@unimi.it

WITH UNITY V2020.3.0

AFFECTIVE COMPUTING

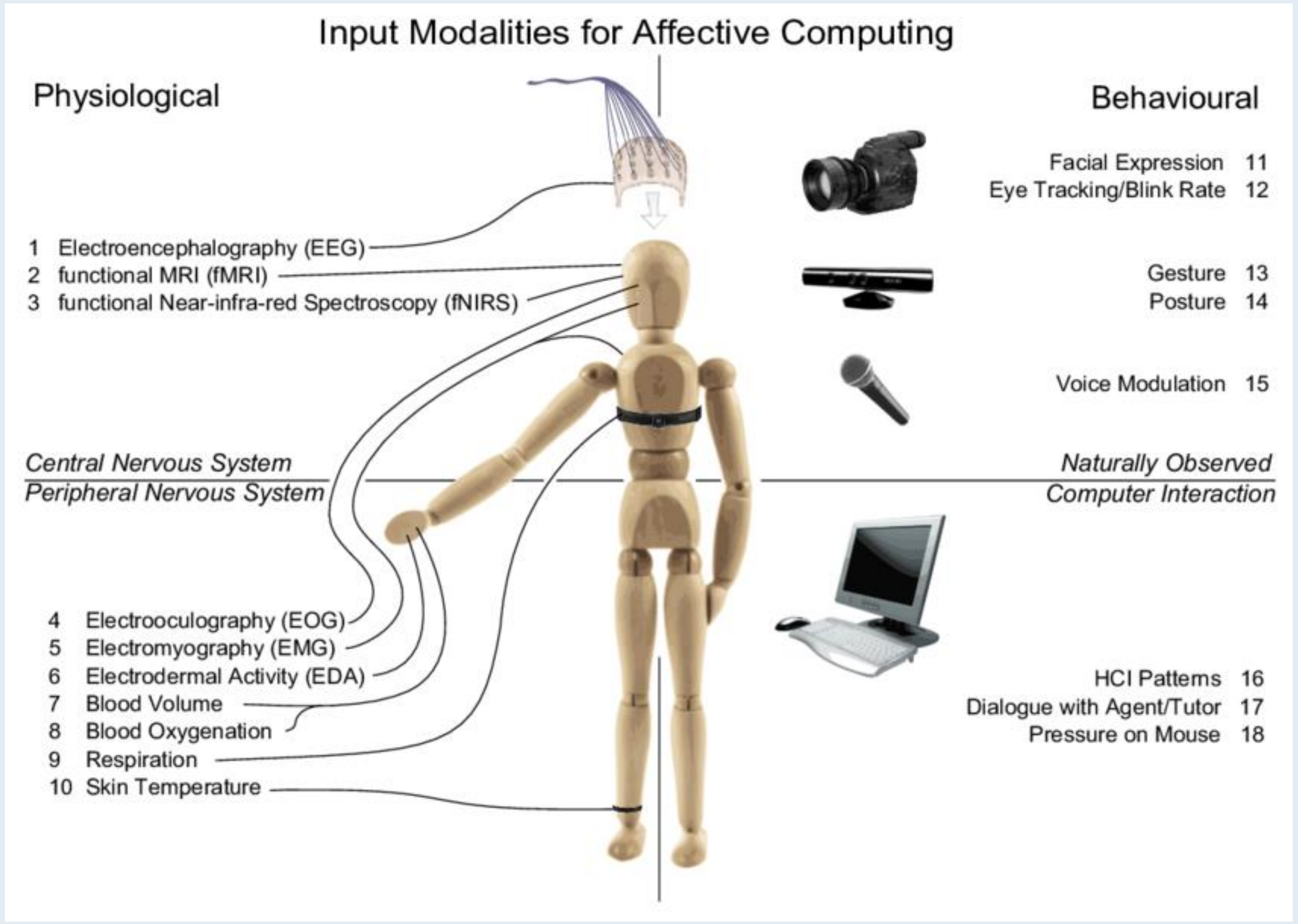
AFFECTIVE COMPUTING (AC)

Picard R. (1995)

Affective Computing (AC) aims at developing systems which can automatically recognize emotions to give adequate responses

Standard methods:

- Face tracking via cameras
- Voice tracking via microphones
- Physiological signals via sensors
- …



AC APPLICATION

- **Education:** cameras or microphones can be used to understand students' emotional states during lessons helping teachers to adapt themselves to tailor class load
- **Healthcare:** bots can monitor physical and emotional well-being of patiens or help doctors in counseling sessions
- **Marketing**: AC can analyze what makes customers engaged or their reactions to new products and companies could organize accordingly
- **Entertainement:** gaming companies can use AC for testing their games monitoring players' satisfaction level, but AC can be also useful to support the game adaptation to players' mental state
- …

AC AND VR

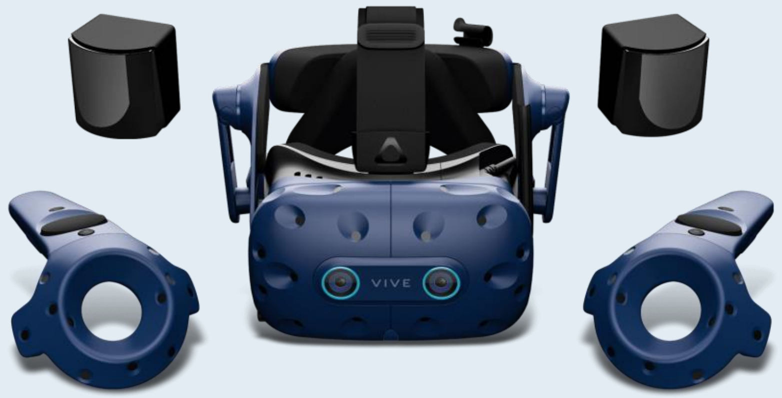
Why VR?

- High degree of immersion
- Elicit stronger emotional reactions
- Gather different types of data
- Realistic experience in a controlled and safe setting

It provides simulated experiences that create the sensation of being in the real world. Thus, VR makes it possible to simulate and evaluate spatial environments under controlled laboratory conditions

AC TECHNIQUES IN VR

- Face tracking
- Eye tracking
- Voice signal
- Motion-behavioral data
- …



UNITY XR AND INPUTS

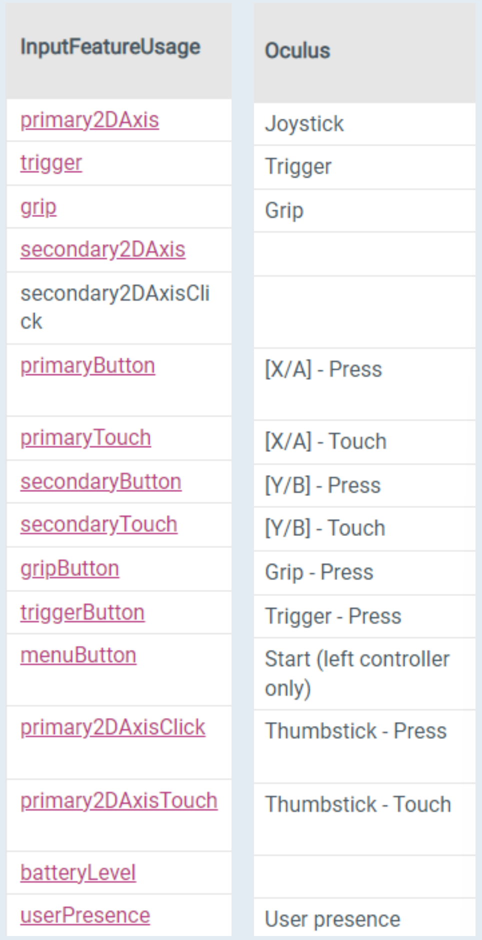
XR INPUT

With OpenXR it is possible to track input features that you can take advantage of when you design user interactions

Data:

- Positions
- Rotations
- Touch
- Buttons
- Joysticks

[Unity - Manual: Unity XR Input (unity3d.com)](unity3d.com)



InputFeatureUsage	Oculus
primary2DAxis	Joystick
trigger	Trigger
grip	Grip
secondary2DAxis	
secondary2DAxisClick	
primaryButton	[X/A] - Press
primaryTouch	[X/A] - Touch
secondaryButton	[Y/B] - Press
secondaryTouch	[Y/B] - Touch
gripButton	Grip - Press
triggerButton	Trigger - Press
menuButton	Start (left controller only)
primary2DAxisClick	Thumbstick - Press
primary2DAxisTouch	Thumbstick - Touch
batteryLevel	
userPresence	User presence

XR INPUT SCRIPT

1. Open the script called ‘XRinput_demo’
2. In order to use the XR functions, you have to add the XR namespace:

```
using UnityEngine.XR;
```

3. We need a private list to store the connected devices (in our case, controllers and headset):

```
List<InputDevice> devices = new List<InputDevice>();
```

4. We also need some variables to store the reference of the target device:

```
InputDevice targetLeftHand;
InputDevice targetRightHand;
```

```
InputDevice targetHead;
```

INITIALIZATION 1/2

We need to constantly check the presence of the controllers (both left and right):

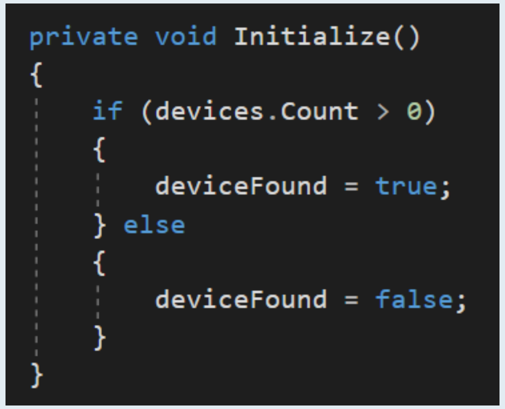
- If left/right controller found -> buttons and inputs can be checked
- If left/right controller not found -> buttons and inputs can’t be checked

1. We need a private boolean variable which checks if the target device has been found:

```
private bool deviceFound = false;
```

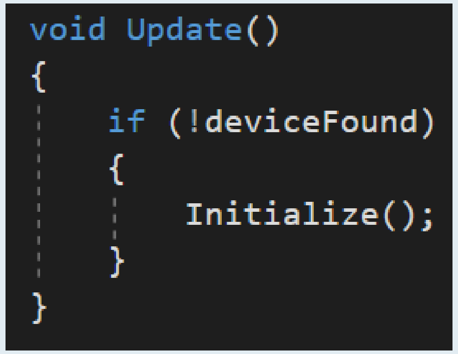
INITIALIZATION 2/2

2. We need a function to search for the target device:
   - If the device is found the boolean variable is set to true
   - Else the boolean variable is set to false



```
private void Initialize()
{
    if (devices.Count > 0)
    {
        deviceFound = true;
    } else
    {
        deviceFound = false;
    }
}
```

3. The function Initialize has to be called:
   - In Update, only if the boolean deviceFound is set to false
   - Else we will retrieve the desired value

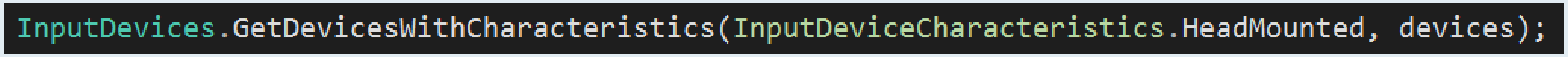


```
void Update()
{
    if (!deviceFound)
    {
        Initialize();
    }
}
```

HEAD

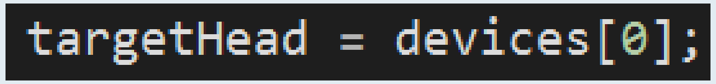
Each device has its own characteristics

1. If we want to find the HMD in the list, we need to check all the devices’ characteristics in order to find the one that matches with the characteristics ‘HeadMounted’, in the Initialize() function



```
InputDevices.GetDevicesWithCharacteristics(InputDeviceCharacteristics.HeadMounted, devices);
```

2. Then, we need to assign the device found to the targetHead variable in the Initialize() function, only if the devices.Count > 0:



```
targetHead = devices[0];
```

FEATURES

You can get different values from controllers:

- Velocities and accelerations -> Vector3
- Button (grip/trigger) pressure -> float (range 0-1)
- Button (grip/trigger) pressed -> int [0-1]
- Thumbstick position -> Vector2

You can get different values from HMD:

- Velocities and accelerations -> Vector3

TryGetFeatureValue()

If you want to access the current state of the device, you have to call the function
`InputDevice.TryGetFeatureValue()`

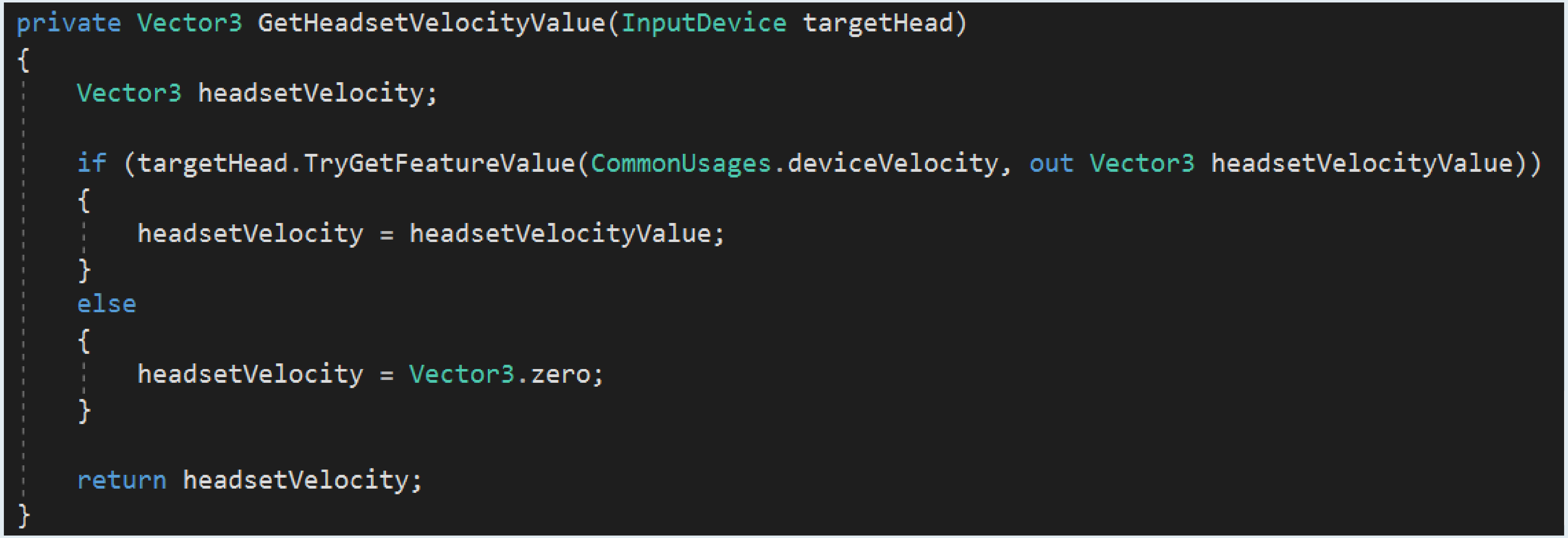
To get a particular button value or input, you must use the **CommonUsages** class:

CommonUsages defines static variables used to retreive input freatures from TryGetFeatureValue

[Unity - Scripting API: CommonUsages (unity3d.com)]

HEAD VELOCITY

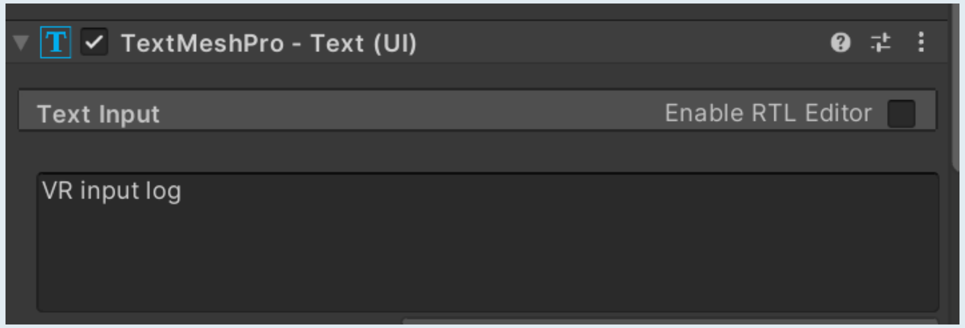
For example, if we want to get the head velocity value:



```csharp
private Vector3 GetHeadsetVelocityValue(InputDevice targetHead)
{
    Vector3 headsetVelocity;

    if (targetHead.TryGetFeatureValue(CommonUsages.deviceVelocity, out Vector3 headsetVelocityValue))
    {
        headsetVelocity = headsetVelocityValue;
    }
    else
    {
        headsetVelocity = Vector3.zero;
    }

    return headsetVelocity;
}
```

XR INPUT CONSOLE

1. In your scene, create a new XR UI with right mouse click in the Hierarchy > XR > UI Canvas
2. Select the Canvas object and right mouse click > UI > TextMeshPro, when promped click Import TMP Essentials
3. Scale the Canvas and the Text and place them in front of the XR Rig
4. Select the Text object and, in TextMeshPro - Text component, change ‘New Text’ into ‘VR input log’

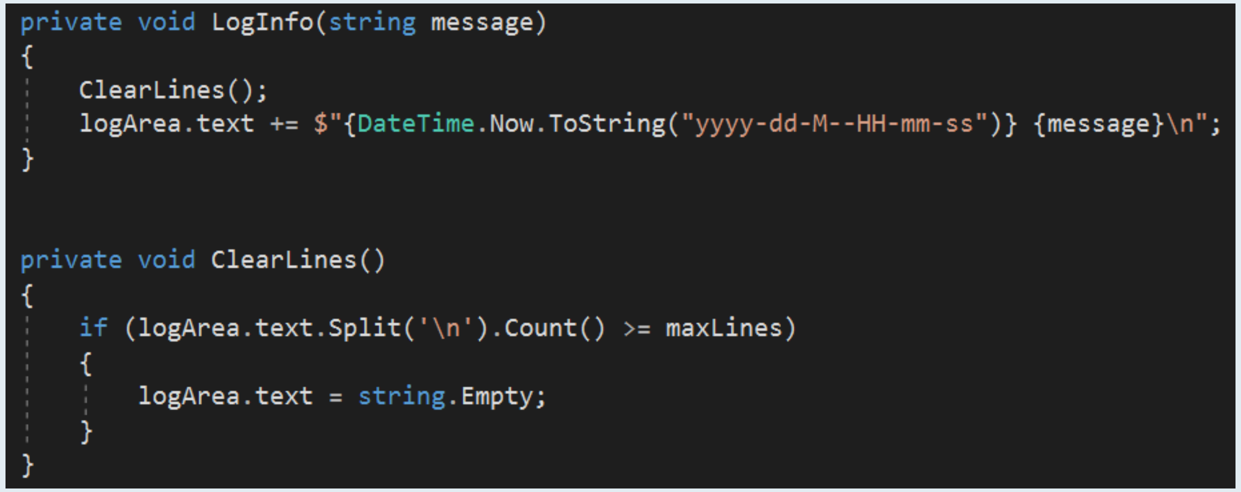


VR input log

LOGS

1. You need two fields:
   1. A public TextMeshProUGUI which defines the area where we want to display logs
   2. A private int to define the max num of lines allowed

2. You need also a function to print the logs and a function to clear the lines where the maximum allowed is reached:



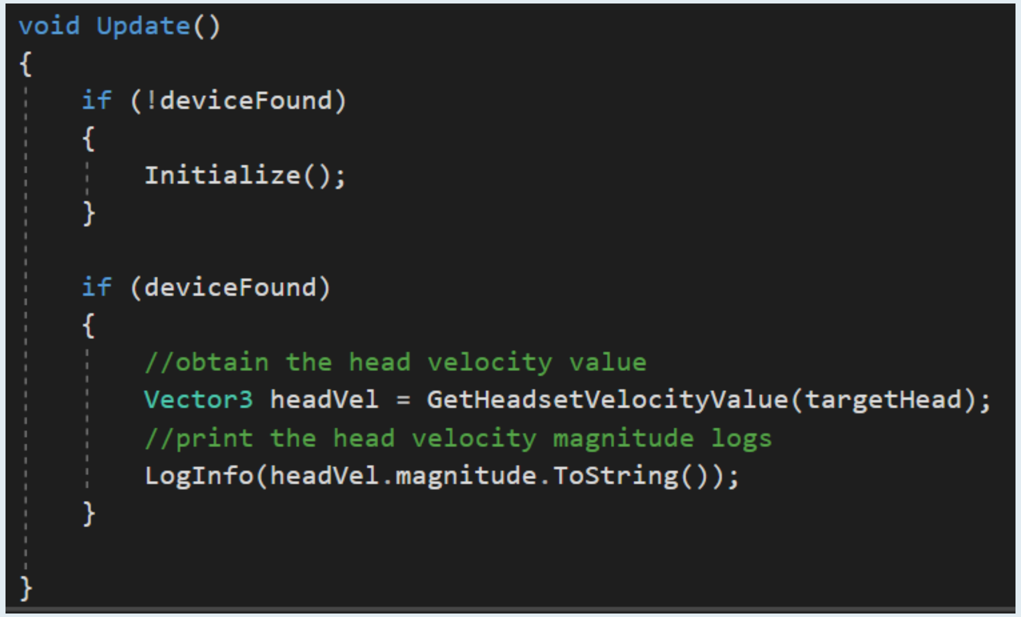
```
private void LogInfo(string message)
{
    ClearLines();
    logArea.text += $"{DateTime.Now.ToString("yyyy-dd-M--HH-mm-ss")} {message}\n";
}


private void ClearLines()
{
    if (logArea.text.Split('\n').Count() >= maxLines)
    {
        logArea.text = string.Empty;
    }
}
```

PRINT HEAD VELOCITY LOGS

1. In the Update() function, we need to call the GetHeadsetVelocityValue() function, giving the targetHead as InputDevice

2. We will print the velocity logs calling the LogInfo() function and giving it as input the magnitude of the head velocity

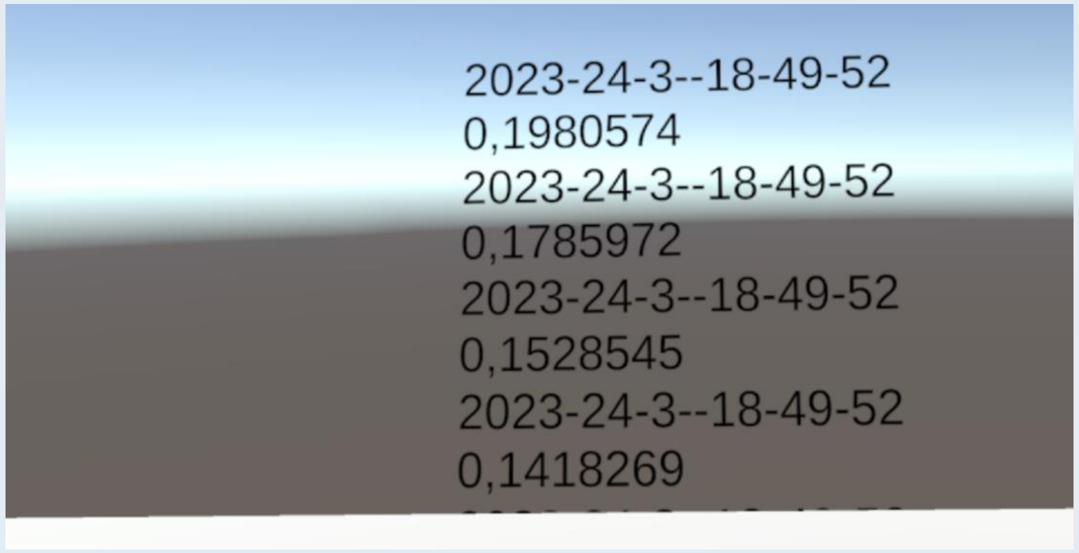
N.B. we call the functions only if the target device has been found



```
void Update()
{
    if (!deviceFound)
    {
        Initialize();
    }

    if (deviceFound)
    {
        //obtain the head velocity value
        Vector3 headVel = GetHeadsetVelocityValue(targetHead);
        //print the head velocity magnitude logs
        LogInfo(headVel.magnitude.ToString());
    }

}
```

RESULTS

- In the Hierarchy, create a new empty GO with: right mouse click > Create Empty and call it ‘Input Manager’
- Attach the ‘XRinput_demo script to the Input Manager GO by dragging it to the GO
- Assign the ‘Text (TMP)’ object (children of Canvas) to the ‘Log Area’ variable
- Press Play



```
2023-24-3--18-49-52
0,1980574
2023-24-3--18-49-52
0,1785972
2023-24-3--18-49-52
0,1528545
2023-24-3--18-49-52
0,1418269
```