3D User Interfaces Final Project SS22

Enrique Mesonero Ronco

1 General data

Name of the assessor: Dr. Jean-Luc Lugrin, Dr. Martin Fischbach, Chris Zimmerer, Ronja Heinrich, Andrea Bartl

Name of the author of the proof of work: Enrique Mesonero Ronco

Last Actualization: 20/09/2022

Index

2 Mandatory Requirements

In this section an explanation will be given about the different methods used to fulfil the different minimum requirements of this work.

2.1 Mandatory Requirement 1: Virtual World

For the development of this work, the project attached to this SS22 course, the *getting started* project, has been used as a basis.

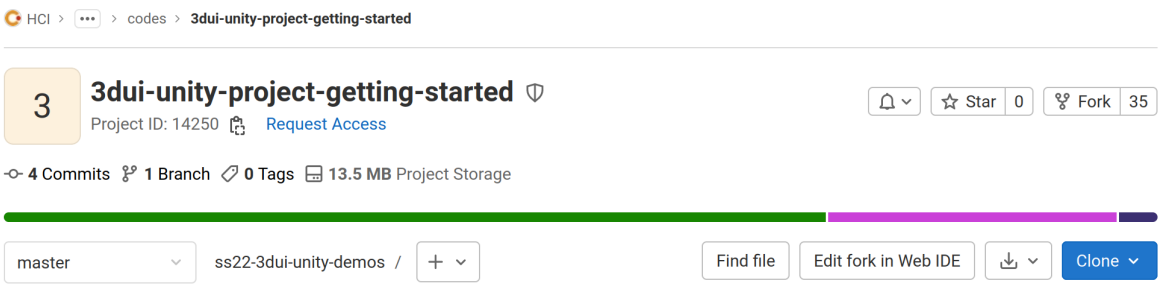

Abbildung 1: Project in GitLab

The project uses the Unity Game Engine Version: Unity: 2019.3.8f1.

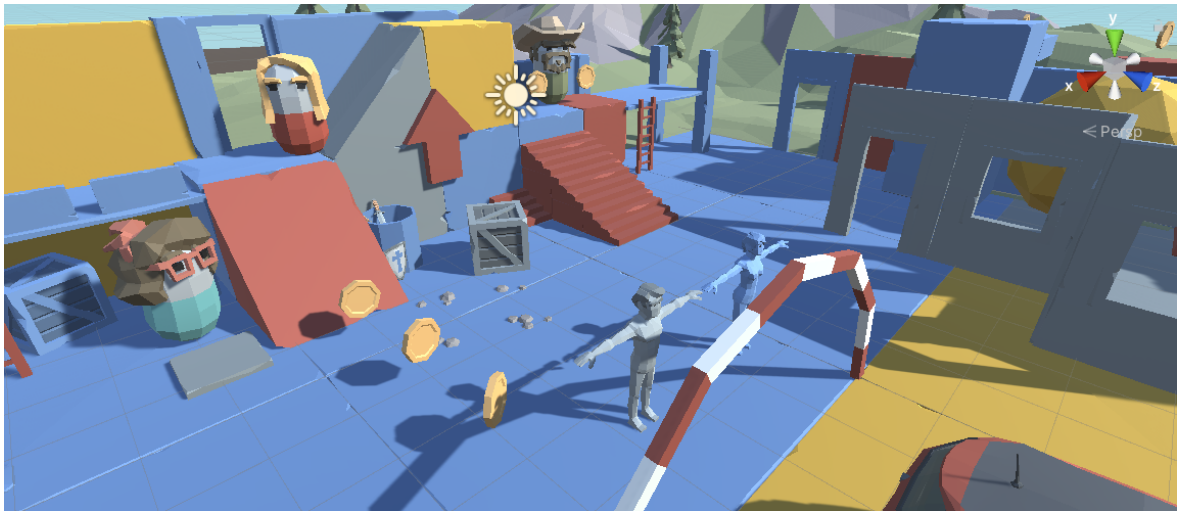

Abbildung 2: 3D Scenario

2.2 Mandatory Requirement 2: 3D Menu

Within the creation of menus within the environment, we have three different menus:

- The first one, a menu that allows us to create several objects (See O10) activated with the button: **Enter button**.

- The next one, a menu that shows us the different interactions that we can perform with our controls, this will be displayed by pressing the **Menu button** on our **left control**.

1 – Interact with GUI
2 – Hand-Steering Speed (pointing = direction)
3 – Help Menu (this one)
4 – Jumping (pointing = Target location)
5 – Object Factory Menu
6 – FPS Counter Menu
7 – *Nothing*
8 – Pick and Drop Object on/from Ray
9 – Grab Object
10 – Move object attached to Ray
11 – *Nothing*
12 – *Nothing*

Abbildung 3: Menu controller

- Finally, we will have a menu that shows us the FPS, as well as other information about our hardware system (cpu consumption, ram, etc...). This menu will be activated by pressing the **B button** on the **left controller**.

2.3 Mandatory Requirement 3: Selection of virtual objects

For the manipulation of objects in this scenario, the **GrabObjectVirtualHand** script has been implemented, which is based on the bubble cursor technique (Manipulation 1 Lecture), in which a spherical collider is added to one of the knobs (in this case the right one).

For object detection, when this collider detects an object, it will turn black (blue, in case this object is previously black), and when we grab that object (by pressing the

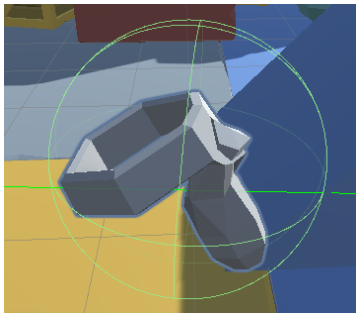

Abbildung 4: Spherical Collider

trigger of the right knob), the object will turn red and our stage knob will turn red.

In addition, we will have an audible feedback (clicking sound when we grab an object) as well as a vibration when we detect an object that we can pick up and as long as we hold on to that object.

2.4 Mandatory Requirement 4: Object Modify

To fulfill the M4 requirement, the **GrabObjectVirtualHand** script is also used, since it contains the code for modifying the position, rotation and scale of the object. To grab the object, you must first detect the object (it will turn black or blue, as mentioned in the previous section), and once detected, press the **right Trigger**. Once this is done, both the grabbed object and our controller will turn red, as well as we will hear a click, and we will notice a constant vibration. To release the object, release the **right Trigger**. If we want to modify its scale, we will have to, while holding the object, move the **right Joystick**.

WARNING: if we move an object with a **RigiBody**, we must take into account that this object has physics, so when we move the object to an elevated position, it will fall to the ground.

2.5 Mandatory Requirement 5: Navigation

For navigation through our scenario, the **HandSteering** script has been implemented. To be able to move around the map, we will simply move the **left Joystick**, and with this we will be able to move in a simple way. Another way to move is by **Jumping**, implemented in the **Jumping** script, explained in the major optional requirement O1. In addition, by pressing the left joystick, we will have SSnap Turningrrotation, rotating 45 degrees each time the left joystick is moved from left to right or vice versa.
If we press the joystick again, we return to smooth rotation.

2.6 Mandatory Requirement 6: Guide

To start the prototype, we open the project with Unity and click Play. To access a guide to the possible movements, press the **Menu button** on the **left knob**, where an image will be displayed, using the **HelpMenu** script. In addition, with the script

FPSMenuController we will have a menu with the statistics of our system, (FPS, CPU consumption, etc...), pressing the **button B** of the **left Joystick**, as previously mentioned.

2.7 Mandatory Requirement 7: Documentation

For the documentation of this work, the following document is attached.

2.8 Mandatory Requirement 8: Video

For the documentation of this work, the following video is attached.

3 Major Optional Requirements

3.1 Major Optional Requirement 1: Selection-based VR Traveling Technique

For the realisation of this objective, two prefabs have been created, a red sphere with a white cylinder that indicates the point where we are going to jump (with the grip button of the left joystick) and the rotation, indicated by the white cylinder, that is, where the cylinder points is where the camera points, (with the left joystick). The other prefab (blue sphere) will indicate the position where we were before. This requirement is fulfilled by using the AdvancedJumping script. Thus, we have pre-trip information (red sphere) and post-trip information (blue sphere).

4 Minor Optional Requirements

4.1 Minor Optional Requirement 9: Automatic velocity regulation

To meet this requirement, a **High Speed** option has been added to the HandSteering script, which is activated by pressing the left Trigger. When we are in the **High Speed** mode, the speed will increase, as the acceleration parameter will be added to the minimum speed value constantly until this value is equal to the maximum speed. When we stop moving, the speed will return to the minimum value. If we leave the **High Speed** mode, we will move at a constant minimum value.

4.2 Minor Optional Requirement 10: Creation Menu

In this section, I have created a menu with 9 buttons, 4 of them to create different figures represented by a figure on each button. 4 of the following buttons represent the crossed out figure on a red button, which indicates that these buttons are for deleting all figures of the same type. Finally, a last button is used to delete all the created figures. To open this menu, we must press button 1 (Y) on the left Joystick, and to select a button from the menu, we will use the Trigger buttons on both Joysticks.

5 Script Explanations

To avoid repetition when explaining the scripts, I'm going to explain here one of the most used functions by the scripts, which is basically to detect the devices (controllers) used, to basically be able to perform the different actions in our VR scenario. This code snippet is taken from the **GrabObjectVirtualHand script**.

```
private void GetRightHandDevice()
{
    var desiredCharacteristics = InputDeviceCharacteristics.HeldInHand
        | InputDeviceCharacteristics.Right
        | InputDeviceCharacteristics.Controller;

    var rightHandedControllers = new List<InputDevice>();
    InputDevices.GetDevicesWithCharacteristics(desiredCharacteristics, rightHandedControllers);

    foreach (var device in rightHandedControllers)
    {
        Debug.Log(string.Format("Device name '{0}' has characteristics '{1}'",
            device.name, device.characteristics.ToString()));
        righHandDevice = device;
    }
}
```

Abbildung 5: Getting Hand Devices

5.1 GrabObjectVirtualHand

Based on the principle of the bubble cursor, this script is implemented in order to be able to pick up objects, as well as to be able to drop them in a new place and modify their scale. The following functions are implemented for this purpose:

-**GetDevice()**: This function is in charge of getting the body of our controller, as well as its original color (using a Renderer for it).

-**OnTriggerEnter()**: his function defines the behavior when the Collider of our command detects another collider of another object, and saving that object in **Colliding Object**. That is to say, that if we detect an object, this one will change color to black.

-**OnTriggerExit()**: This function defines the behavior when the Collider of our controller stops detecting another collider of another object, returning the color of this one to its original color, indicating that we cannot pick up that object because it is not detected.Also setting **Colliding Object** to NULL.

-**GrabObject()**: The **GrabObject()** function will be used to:
- Grab the object.
- Once grabbed, modify its position and, in addition, its scale.
- In addition, GrabObject detects if the object to grab has a **RigiBody**, that is to say, if the object has physics, to be able to handle it correctly.

-**ReleaseObject()**: The **ReleaseObject()** function is simply used to release the grabbed object, taking into account, like **GrabObject()**, whether an object has physics or not.

In addition, there are the base functions, **Start()** and **Update()**. **Start()** will be in charge of the detection of our device (controller). **Update()** will take care of the detection of our object, changing its color and sound when it is detected. As well as the detection of if our Trigger is pressed or not, passing to the functions when this one is pressed or not, respectively.

5.2 HandSteering

-**GetHandDevice()**: With this function, we look for our device, in this case, the left controller, which will perform the action of moving, using the left Joystick.

-**GetHandControllerGameObject()**: With this function we obtain the representation of the device within our VR environment.

-**GetTrackingSpaceRoot()**: With this function we obtain our Rig, which is: The RRigïs used to refer to the base of the XR Rig. This is the that will be manipulated via locomotion.

-**MoveTrackingSpaceRootWithHandSteering()**: This function is in charge of the movement. To do this, it takes the values of the axis and the left Joystick, used to modify the value of the vector of our position, thus performing the action of movement.

The **Start()** function searches for the device and the XRRig. **Update()** is used for motion, as it is something that is constantly being updated.

5.2.1 AdvancedHandSteering

This is a modified HandSteering script in which two new options are added:

- If we press the Joystick, we will enter Snap Turning mode, in which, by moving the left Joystick in the Y axis (left-right) we will make 45 degree turns. If we press the left Joystick again, we will return to the smooth rotation mode.

- If we press the Left Trigger we will enter the **High Speed]** mode, where, starting from a minimum speed, we will automatically increase the speed as we move until we reach the maximum speed value. To exit this mode, press the left joystick again.

5.3 Jumping

The Jumping Script is based on the Jumping technique, and is used for the movement through jumps, and for this it implements the following functions:

-**getRightHandDevice()**: With this function, we look for our device, in this case, the left controller, which will be the device that will perform the jump action, by pressing the grip button.

-**getRighHandController()**: With this function we obtain the representation of the device within our VR environment.

-**getTrackingSpaceRoot()**: With this function we obtain our Rig, which is: The RRigïs used to refer to the base of the XR Rig. This is the that will be manipulated via locomotion.

-**createObject()**: With this function we will create the object that will show us the rotation and final position when we make the jump.

-**getPointCollidingWithRayCasting()**: with this function we will obtain the point where the RayCast of our device hits, which we will use to transform our position to the position of the point of contact of the RayCast with a surface, using the function -**MoveTrackingSpaceRootWithJumping()**.

-**MoveTrackingSpaceRootWithJumping()**: with this function we will transform our position to the position of the contact point between the RayCast and the surface, thus performing the jump function.

With the **Start()** function we will start the functions to get the device and create the orientation object. With the **Update()** function, to get the point of collision of the RayCast with the surface, in an updated form. Also the function to make the jump.

5.3.1 AdvancedJumping

This is a modified Jumping script where the displacements of the prefabs are added, (red pre-travel, blue post-travel) and where the rotation of the character is modified by copying the rotation of the red prefab, which is indicated by the white cylinder added to this red prefab.

5.4 HelpMenuController

-**GetLeftHandDevice()**: With this function, we look for our device, in this case, the left controller, which will be the device that will perform the menu action, by pressing the menu button.

-**GetXRRigMainCamera()**: With this function we obtain the position of our camera, which will help us in the future, at the time of generating our help menu, to be able to visualize it in a correct position, that is to say, in front of us.

-**OpenOrCloseHelpMenu()**: This function will be in charge of checking if the menu button on the left knob is pressed or not, to open and close the help menu. If it is pressed, it will open the menu, calling the **Open()** function creating a prefab with an

image with the actions that we can perform with our two commands. If it is pressed again, the menu will close, calling the **Close()** function.

-**Open()**: The open function will call **CreateMenuFromPrebab()** and **AttachCameraToMenuCanvasAndDisplayMenu()** to create the menu in front of us.

-**CreateMenuFromPrebab()**: It will create the prefab which is our menu.

-**Close()**: It will destroy the prefab that is our menu.

-**AttachCameraToMenuCanvasAndDisplayMenu()**: function in charge of the correct positioning of our menu, making use of the position of our camera.

Start() function to obtain the left device and the position of our camera. **Update()** function for the creation and destruction of our menu.

5.5 FPSMenuController

It is a script the same as **MenuHelpController**, only with another prefab that, in this case, shows the system statistics, and instead of using the **Menu button** on the **left knob**, it uses the **B button** on the **left knob**.

5.6 PlayerModeController

With this script, if we press the **B button** on the **right knob**, we will change the mode in the Unity development environment. Pressing this button, we will end the game mode in the environment.

5.7 ObjectFactoryMenuController

It is a script like **MenuHelpController**, only with another prefab that, in this case, shows a menu for the creation and destruction of objects in our 3D environment, and instead of using the **menu button** in the left knob, it uses the **button A** in the left knob.

5.8 ObjectSpawner

With this script, clicking with the **trigger** of the **right knob** in our menu created by **ObjectFactoryMenuController**, we will create an instance of an object, depending on the option that we select in the menu.

5.9 RayPicking

RayPicking is based on the same functions as Jumping to create a RayCast, but focused on grasping and moving objects.

- OutlineObjectCollidingWithRay() AttachOrDetachTargetedObject() are used - to detect and grab the detected object respectively; by pressing button 1 (A) on the right controller.

MoveTargetedObjectAlongRay() RotateTargetedObjectOnLocalUpAxis() with these two functions we can move the object along the RayCast and rotate it on its own axis with the right joystick, respectively.

- In addition, with the functions GenerateVibrations() and GenerateSound(), which serve to add vibration and sound feedback, respectively.