Universidad Rey Juan Carlos

VIRTUAL REALITY

PROJECT REPORT: SPACESHIP BATTLE COMMANDER SIMULATOR

Jesús Bastante López

Index

Throughout this report, the work completed for the course project will be thoroughly explained. In this case, the project involves creating a space combat simulator. To provide clarity, the explanation will be divided into several key sections essential to the project's development.

1. Project Concept.

The project, as previously mentioned, is a space combat simulator, specifically one focused on commanding a fleet of spaceships. The idea was inspired by various scenes from movies and video games where the protagonist manages a fleet within a virtual reality system or, alternatively, through an extended reality graphical interface.

Figure 1: References.

The main idea of the project is not exactly a video game, as no victory or defeat conditions have been incorporated (at least for now). Instead, it is a simulator where the user can position fleets as desired and then simulate the combat from that setup. Therefore, it allows modifying the existing ships at any time, should there be a need to alter the battlefield during the simulation.

This entire project was developed in *Unity* (Unity Technologies, 2005), with the assistance of its XR Plugin Management plugin.

2. The User as an Entity within the Simulator.

For the user's avatar, a free first-person camera has been chosen, with hands as the sole representation of the body. This choice was made because the hands are the only elements needed for interacting within the program, and they enhance immersion.

Within the simulator, the user can pause time to place and reposition ships, as well as to command allied ships on where to go or which targets to attack. The details of how this is accomplished will be explained later.

To set up the user, a "Player_XR_Origin" has been created with a tracking origin mode set to floor level. Within this object are located the components that simulate the hands, the cameras required for the user to properly view the virtual environment, and all the necessary interactors for proper functionality

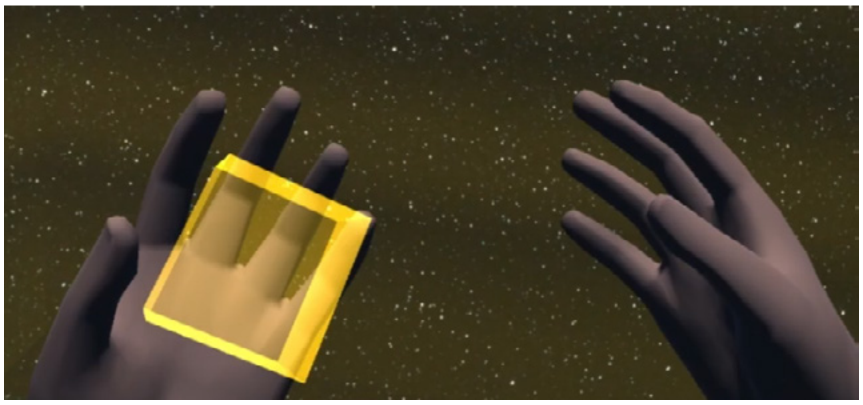


Figure 2: Virtual hands

3. Interaction with the Ships.
The ships are the main element of the simulator, as they are what the player primarily interacts with. The project includes two types of ships: the Gunship and the Cruiser. Since the inner workings of these ships are not the focus of this project, we will not delve into their programming. It is enough to note that the Gunship is slow and fragile, with higher firepower but a lower rate of fire, while the Cruiser is fast, more durable, but with lower firepower.

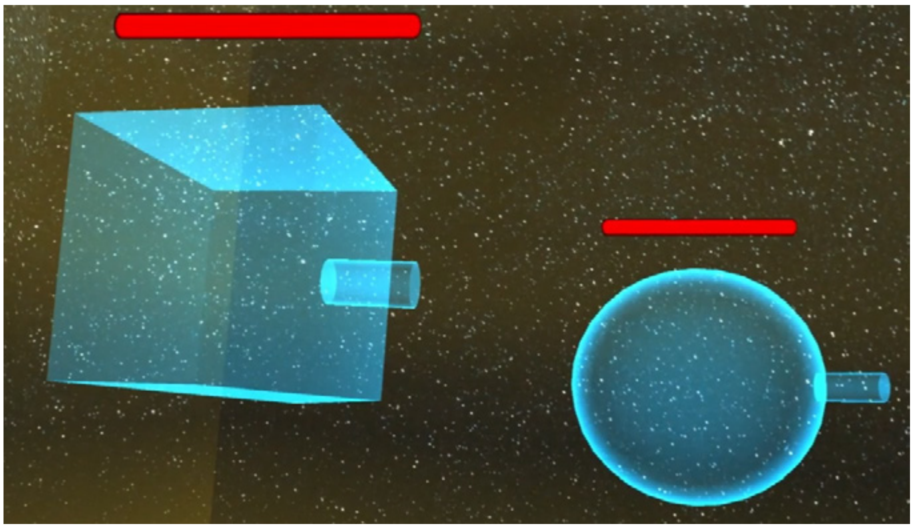


Figure 3: Spaceship types.

The player can interact with the ships using two “ray interactors,” each located on one hand (LShipInteractor, RShipInteractor). An “XR Simple Interactable” component has been added to each ship to enable user interaction. Associated with this interactable component are functions to display and hide the attack range when the ship is in a hover state. When a ship is selected, it is designated for commands. Both the interactors and interactables are set to the “Ships” interaction layer mask.

The possible interactions with the deployed ships are as follows:

- **Selecting a Ship:** The selected ship will appear as highlighted in the selected ships menu (indicated by the ship's symbol).
- **Commanding a Ship to Attack:** First, an allied ship must be selected, followed by an enemy ship.
- **Commanding a Ship to Move to a Point:** A gesture must be performed with one hand (gestures are explained later) to create a navigation point at the hand's position. The selected ship will then move to this navigation point.

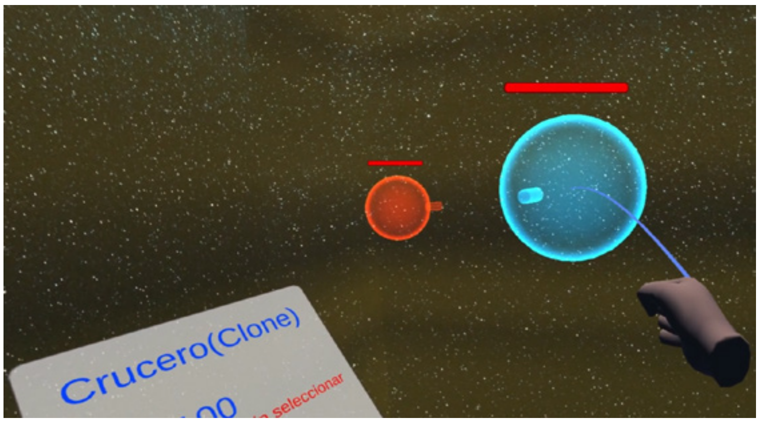


Figure 4: Spaceship selection.

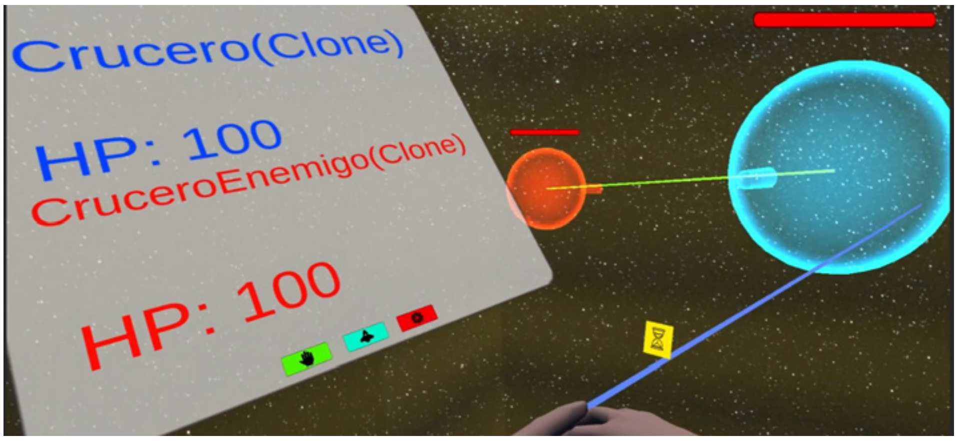


Figure 5: Attack an enemy ship.

To deploy ships, the user must first pause time by pressing the button located on their right hand. Once time is paused, the left hand displays a menu (hand symbol) from which the user can pick up cubes representing the different available ship types and place them as desired. Once deployed, and while time remains paused, the user can grab the deployed ships with their right hand to move or rotate them as needed.

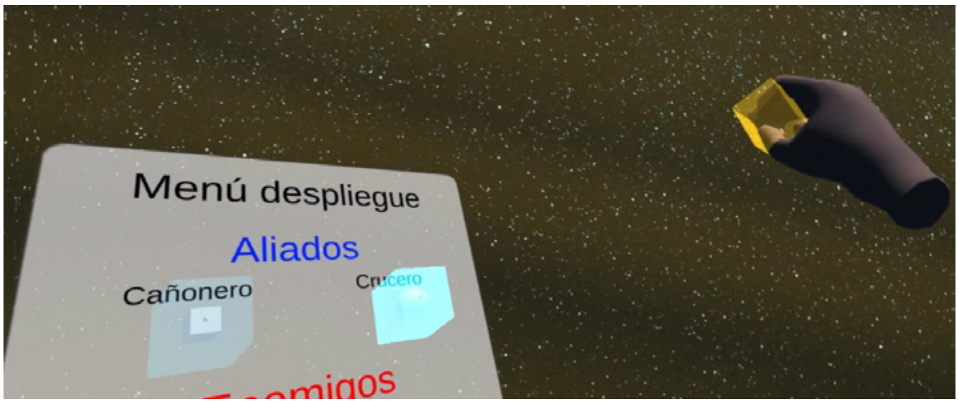


Figure 6: Spaceship deployment.

To achieve the functionalities of grabbing cubes and ships, the hands are equipped with an “XR Direct Interactor,” which interacts with the “XR Grab Interactable” components of the cubes and ships (an additional collider is included within the ships to prevent interference with the ray interactors). Both the interactables and interactors are set within the “DropShips” interaction layer mask.

Finally, a “ship disposal” feature has been implemented in the left hand. This feature is activated when time is paused, allowing the user to remove deployed ships in two ways using a poke interactor in the “DropShips” interaction layer mask:

- **Dragging a Ship:** The user can grab a ship and place it in the disposal cube, which turns red to indicate that a ship is ready for deletion.
- **Touching a Ship:** The user can simply touch a ship with the hand holding the disposal cube..

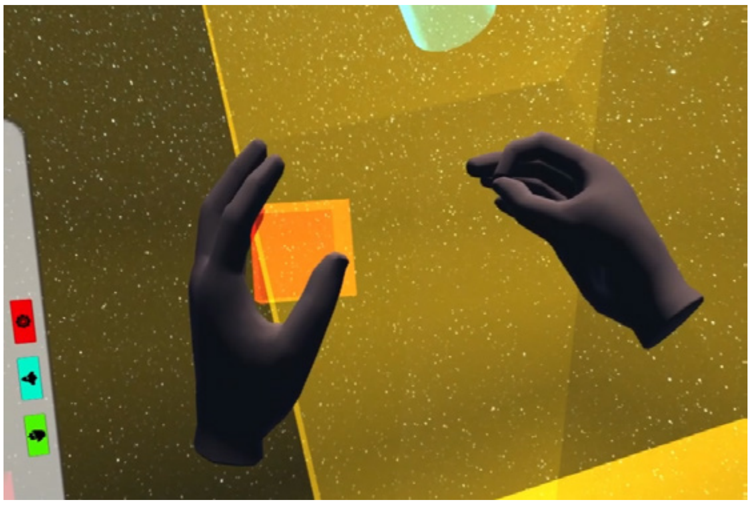


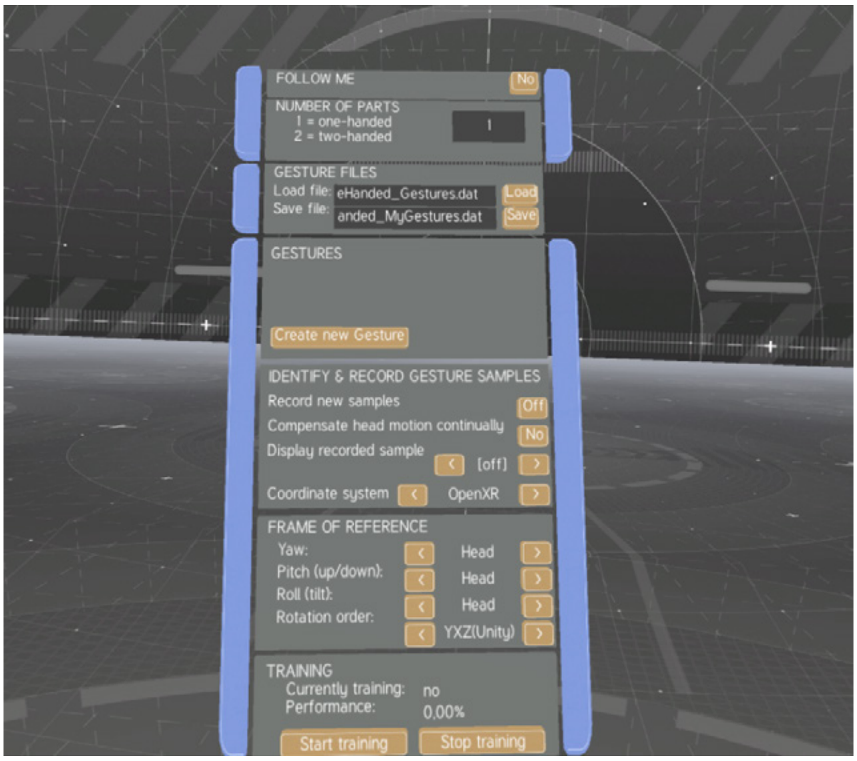
Figure 7: Delete a ship.

4. Gesture Recognition.

For the implementation of gesture recognition, the miVry, asset has been utilized, specifically the version for Unity. miVry allows for the creation of a gesture set, which can be saved and then loaded into our project to recognize the user’s gestures.

Regarding gesture creation, the plugin includes a “GestureManager” scene located within the “GestureRecognition” folder. In this scene, we can save our gestures using an AI that recognizes the patterns made with one or two hands. To do this, we need to activate recognition and press the “trigger” button.

Once we have created our gestures, they can be saved to a specific directory; in this case, it has been hardcoded because the process in the scene itself is not very intuitive. Additionally, we can edit our gesture set by reloading it from any desired directory.


FOLLOW ME: No

NUMBER OF PARTS
1 = one-handed
2 = two-handed
1

GESTURE FILES
Load file: eHanded_Gestures.dat
Save file: anded_MyGestures.dat

GESTURES

IDENTIFY & RECORD GESTURE SAMPLES
Record new samples: Off
Compensate head motion continually: No
Display recorded sample: [off]
Coordinate system: OpenXR

FRAME OF REFERENCE
Yaw: Head
Pitch (up/down): Head
Roll (tilt): Head
Rotation order: YXZ(Unity)

TRAINING
Currently training: no
Performance: 0,00%

Figure 8: Gesture creation scene

In the project, we created an object (MiVRY), assigned the “Mivry” script to it, and specified the directory of our gesture set, the object containing the hands, and the function to be called when a gesture is recognized. Each gesture has an ID used for identification, which is passed as an input parameter to the calling function.

Additionally, a trail of the user’s gesture is created when gesture recognition is activated. This is accomplished using a “trail renderer” that activates simultaneously when the gesture identification button is pressed.

Finally, the recognized gestures are as follows:
- **Circle:** Move the ship to the hand.
- **Vertical Line:** Change the type of movement.

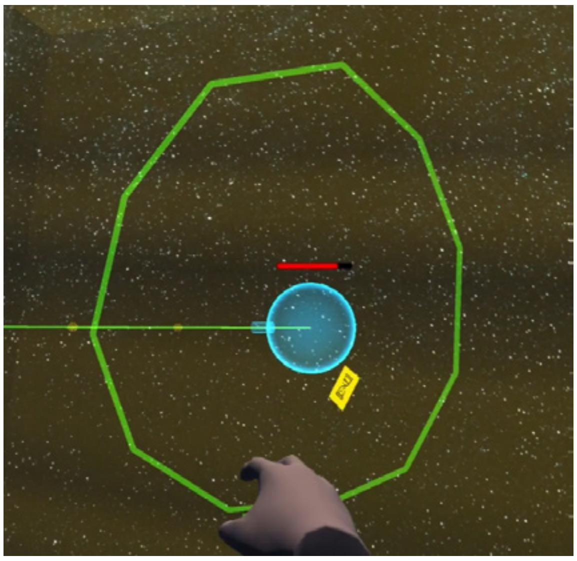


Figure 10: Move ship gesture

Figure11: Change movement type gesture

Problems: The free license of the plugin allows for the identification of only 100 gestures per session, although the project can be reloaded to reset the counter. A more significant issue discovered is that the plugin does not function in the build unless the license is purchased; therefore, the build does not work correctly.

5. Navigation.

For player navigation, both teleportation (for ships) and continuous movement, as well as smooth and snap turning, have been implemented. The type of turning and movement vignette variables can be selected from the "Movement" menu.

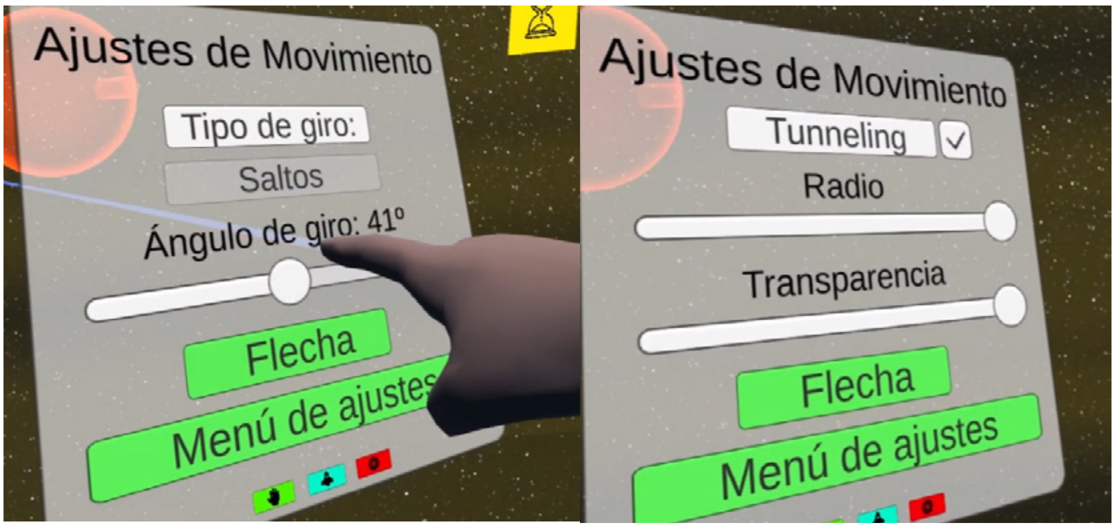
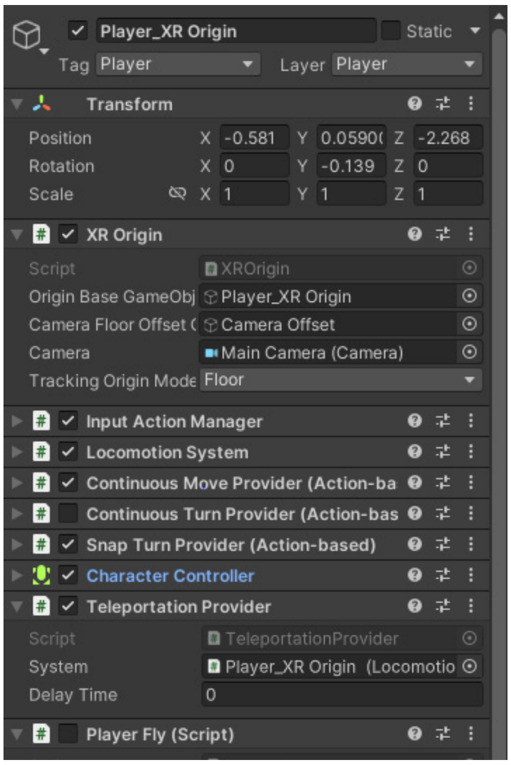
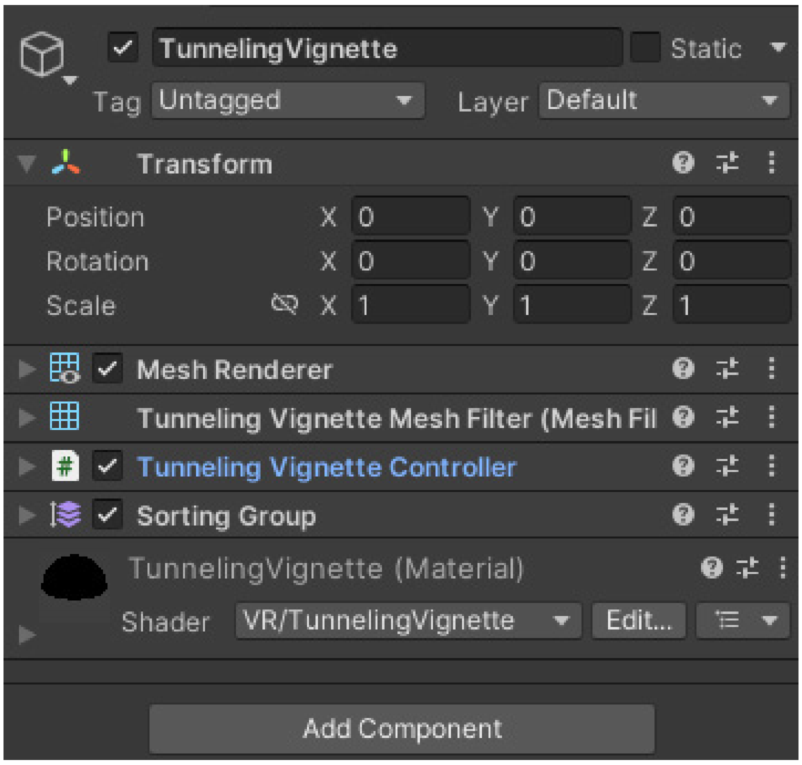


Figure 12: Movement settings.



```
✓ Player_XR Origin                         Static
Tag Player                 Layer Player
Transform
Position             X -0.581   Y 0.0590(  Z -2.268
Rotation             X 0        Y -0.139   Z 0
Scale                X 1        Y 1        Z 1
✓ XR Origin
Script                    XROrigin
Origin Base GameObj       Player_XR Origin
Camera Floor Offset (     Camera Offset
Camera                    Main Camera (Camera)
Tracking Origin Mode      Floor
✓ Input Action Manager
✓ Locomotion System
✓ Continuous Move Provider (Action-ba
  Continuous Turn Provider (Action-bas
✓ Snap Turn Provider (Action-based)
✓ Character Controller
✓ Teleportation Provider
Script                    TeleportationProvider
System                    Player_XR Origin  (Locomotio
Delay Time                0
  Player Fly (Script)
```

Figure 12: Type of movement jerar-chy



```
✓ TunnelingVignette                        Static
Tag Untagged               Layer Default
Transform
Position             X 0        Y 0        Z 0
Rotation             X 0        Y 0        Z 0
Scale                X 1        Y 1        Z 1
✓ Mesh Renderer
  Tunneling Vignette Mesh Filter (Mesh Fil
✓ Tunneling Vignette Controller
✓ Sorting Group
TunnelingVignette (Material)
Shader  VR/TunnelingVignette    Edit...
Add Component
```

Figure 13:Tunneling vignette jerar-chy

Regarding movement, the player can choose the size and transparency of the vignette, all of which is implemented by modifying the values in the “Tunneling Vignette Controller” script within the “TunnelingVignette” object. A “Continuous Move Provider” has also been implemented in the Player_XR Origin. Additionally, through the movement change gesture, the user can move along the vertical axis using the “PlayerFly” script, and can revert to horizontal movement using the same gesture whenever desired.

For turning, the “Continuous Move Provider” and “Snap Turn Provider” scripts from Unity’s XR package have been implemented. When snap turning is enabled, users can choose the turn range from 0º to 90º in the settings menu.

Finally, a small teleportation movement has been implemented for the ships, which feature a “Teleportation Anchor” with its respective “TpObjective” to allow the user to appear above the ship.

Problem: The teleportation works well with allied Gunships; however, for some reason, it often fails to function with other types of ships. This is strange because they were created in the same manner, and extensive testing has been conducted over several hours without resolving the issue.

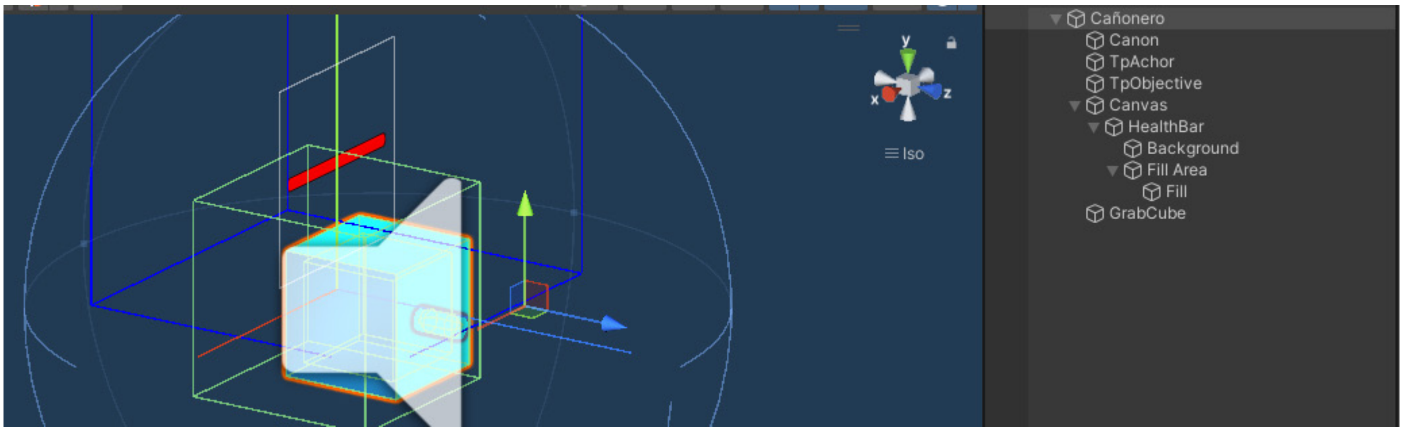


Figure14: Ship with TpAnchor

6. Settings and inputActions.

Although some adjustments have already been mentioned, this section will provide more detailed information and a deeper explanation of all the settings in the virtual environment.

To begin with, there are general settings that allow users to adjust brightness and volume using sliders. Next, we have the movement settings mentioned in the previous section. Lastly, there are control settings, which include options to change the selection buttons, enable rays, activate gestures, and turn on the teleportation ray.

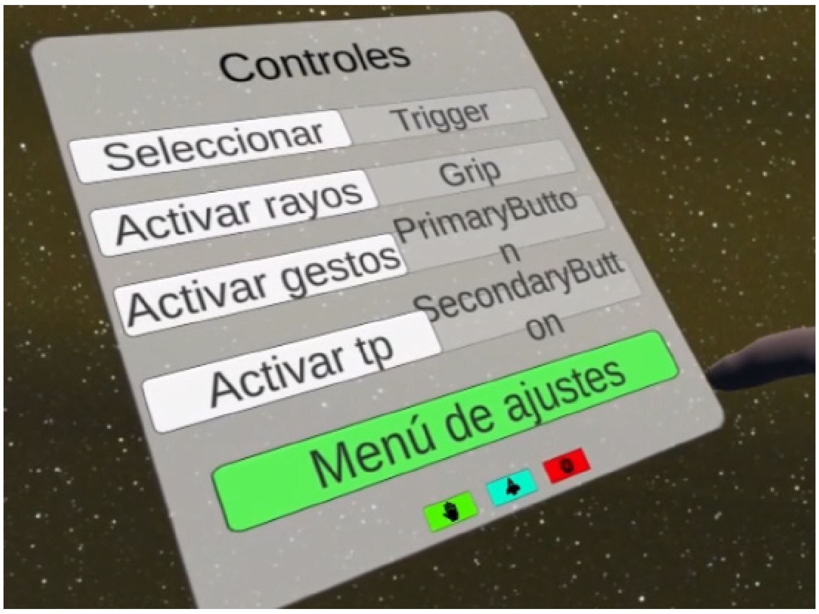


Figure 15: Control settings

To implement these final adjustments, the respective input actions were created in the “XRI MyInputActions” asset located in the “CustomAnimations” folder, which were associated with the default buttons. Next, input actions were established for each hand to identify when one of the available buttons was pressed. Finally, the “ControlSettingManager” script was created, which activates the identification of these input actions when a button in the menu is pressed and manages the swapping of one button for another.

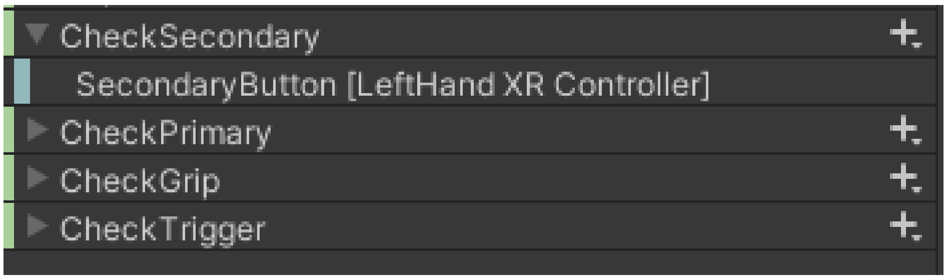

CheckSecondary
SecondaryButton [LeftHand XR Controller]
CheckPrimary
CheckGrip
CheckTrigger

Figure 16: InputActions example

All of these settings windows have been implemented using a "UI Canvas" from Unity's XR Plug-In.

7. Aesthetic Aspects.
In terms of aesthetics, the goal was to emulate the holographic appearance typical of such simulators in fiction. To achieve this, the Sci-Fi Hologram Shader asset by Sci-Fi Hologram Shader by Zololgo was used. From this package of various shaders, the basic "Halloween Ghost" shader was selected and modified to suit each type of ship and cube, primarily focusing on color adjustments.

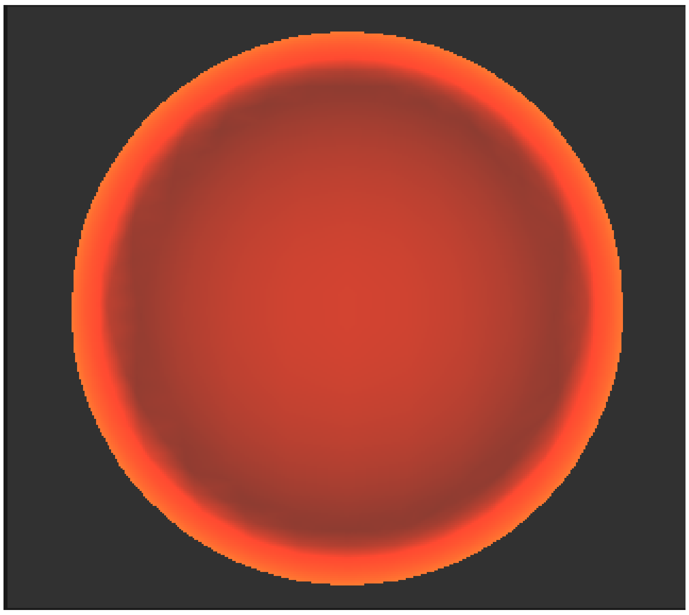


Figure 16: Material example.

Regarding the shape of the ships, basic shapes have been used since the focus needed to be on interaction, leaving this aspect somewhat neglected.

Additionally, a skybox featuring an image of a starry background has been created, and a cube was designed to serve as the map boundary, utilizing another variation of the shader applied to the ships.

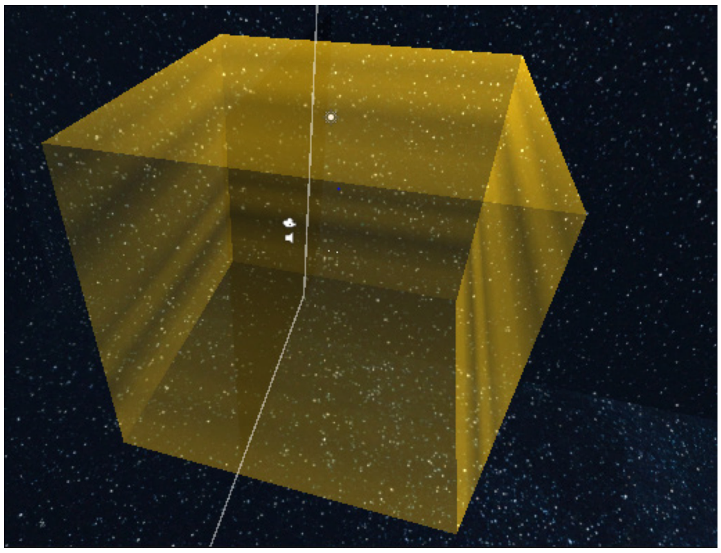


Figure 17: Skybox and limits.

Regarding the menus, the aesthetic of Unity's buttons and menus has been maintained, although different icons and colors have been used for each type of menu. The icons were created using the online icon creation tool Recraft.ai. Additionally, it's worth noting that a secondary camera has been created within the main camera, which captures everything related to the UI to prevent other objects from obstructing it.

Lastly, a new animation for the hands has been created, which is used when activating the selection rays for ships. These two animations, one for each hand, are called "l_OnlyIndex" and "r_OnlyIndex," located in the "CustomAnimations" folder. These animations involve closing the fist while keeping the index finger extended. They were developed from the default animations, modified to achieve the final result, and then implemented in the "animator."

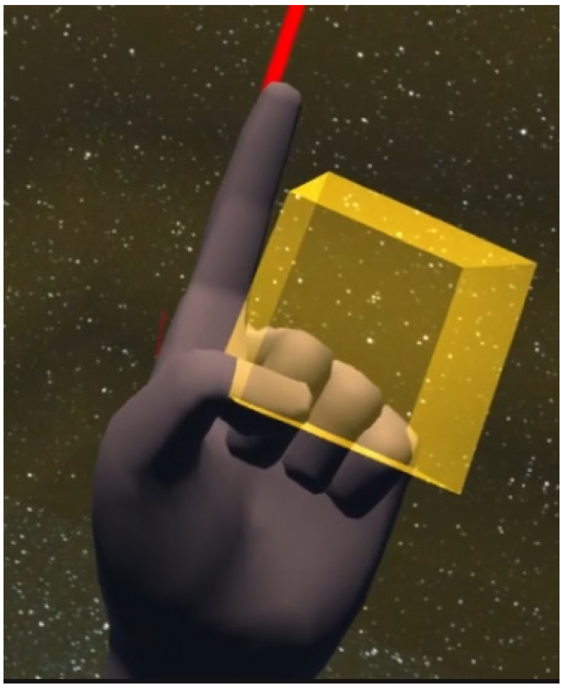


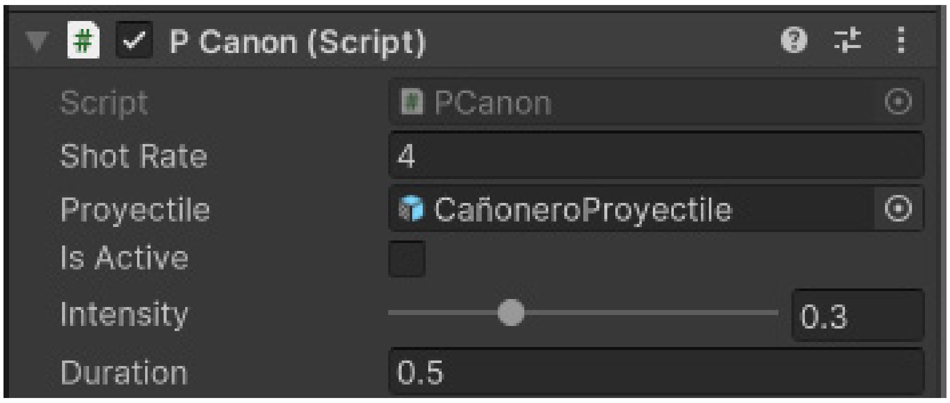
Figure 18: New animation.

8. Sound.

In terms of sound, an “Audio Reverberation Zone” of the “Hangar” type has been implemented within a sphere that is parented to the Player_XR Origin. This zone adds a reverberation effect that makes the gunfire and the AI voice sound as if the user were inside a ship. Unfortunately, only one AI sound and one shooting sound have been implemented, both of which are referenced later.

9. Haptic Feedback.

The only haptic feedback implemented is the vibration of the user’s controllers when they are near a ship and fire their weapon. This vibration can be adjusted for each type of cannon, allowing for easy modification of intensity and duration directly from the editor.



P Canon (Script)

Script	PCanon
Shot Rate	4
Proyectile	CañoneroProyectile
Is Active	☐
Intensity	0.3
Duration	0.5

Figure 19: Canon script.

10. Conclusion.

Throughout the project, we have learned how to harness the great potential of Unity’s XR Plug-In, exploring the various options for interaction, locomotion, and haptic feedback it provides. Additionally, several decisions had to be made regarding how to facilitate user interaction, making it more immersive and enjoyable to use.

We also explored various aspects of Unity, such as input actions, special cameras for UI, the animator, and the audio systems available in the game engine. This included extensive work on implementing ships and their functionalities, such as health bars, navigation, and firing mechanisms.

Finally, we delved into the gesture recognition plugin, addressing various challenges along the way and deciding which gestures to implement and what functionalities could be covered by them.

11. References.

- Shaders://assetstore.unity.com/packages/vfx/shaders/sci-fi-hologram-shader-66455
- Canon shot: https://pixabay.com/es/users/pixabay-1/
- Text to voice AI: https://elevenlabs.io/text-to-speech
- Voice filter: https://voicechanger.media.io/app/male?_gl=1*1s18ecv*_gcl_au*NDMzNjAwNTQ1LjE3MTQzOTg2OTM.*_ga*MTE1NzI2MzIxNC4xNzE0Mzk4Njk0*_ga_24WTSJBD5B*MTcxNjU3ODIxMC40LjAuMTcxNjU3ODIxMC42MC4wLjEwODM4Njg4MjQ.&_ga=2.247834437.398013520.1716578210-1157263214.1714398694
- Slow audio speed: https://audiotrimmer.com/audio-speed-changer/
- ASingleton cript: David María Arribas.