UNIVERSITAT JAUME•I

Application of virtual reality to the Rubik's cube learning

José Fenollera Faustino

Final Degree Work
Bachelor's Degree in
Video Game Design and Development
Universitat Jaume I

May 25, 2022

Supervised by: Miguel Chover Sellés

To my mother, my father, my brother, my friends, my family and all the people who have supported me during these years far away from home.

To my uncle Pedro, who has helped and supported me when I have had problems in a subject.

Acknowledgments

First of all, I would like to thank my Final Degree Work supervisor, Miguel Chover Sellés, for his support on the project from the beginning and during its development.

I also want to thank the professor José Martínez Sotoca, who sadly passed away this year, for his help and support.

Abstract

This document is the Final Degree Project report of José Fenollera Faustino of the Video Game Design and Development degree.

The project consists of a virtual reality video game tutorial that teaches the user how to solve the Rubik's cube step by step. There are various modes, including a free one and tutorial.

Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

Contents

This chapter describes the motivation to start with the idea of creating a Rubik's cube game tutorial for a virtual reality environment and its different objectives achieved during these months. And also the working environment with the professors and the initial state.

1.1 Work Motivation

The idea of this FDP came because in the Game Engines subject, the professor Miguel Chover Sellés asked the students that if any of them wanted to experience the virtual reality or propose a project to develop with it they could ask him to do that, so he accepted to supervise the purpose of a FDP the next course about the Rubik's cube.

Moreover, it is meaningful to develop such a game tutorial, not only because of its usefulness to implement a tutorial in a virtual reality environment, but also because it would be a challenge to program something like this.

1.2 Objectives

The main objectives of this project are the following:

- Analyse how to program a Rubik’s cube from zero and create a virtual cube that was able to rotate its faces independently of the position and rotation in the computer.
- Learn how virtual reality works in a game engine and get that cube to work in such an environment.
- Make the menus.
- Create a way to make the controls of the game for some controllers with very few buttons.
- Analyse how to create a tutorial for the cube, based on the book "El Cubo Mágico" [1] and implement it.
- Make a list of the cube algorithms to be used in the tutorial and link them to the movements of the user, to check if the movement is good or not.

1.3 Environment and Initial State

As it is said in the Work Motivation section, the idea of this FDP started the previous course, so months later a first project was finally finished. A continuation of the project is planned for the future, but this will be commented on in section 6.2 Future Work.

It is important to mention that the project started from zero and the work environment has been significant due to the the new technologies used, the new ideas to be developed and all the process and experience obtained with it.

CHAPTER 2

PLANNING AND RESOURCES EVALUATION

Contents

In this chapter the planning proposed for the project is exposed and all the resources needed to complete it are listed. This is important to have an estimation of the cost of the project.

2.1 Planning

The planning proposed for this project is the following:

- **Learning:**

  - Task 1 (3 hours): Learn about basic virtual reality programming for a game engine.

- **Interaction:**

  - Task 2 (20 hours): Make the selection of the faces and rotate them with hand controllers.

  - Task 3 (10 hours): Adapt the cube to virtual reality.

  - Task 4 (20 hours): Make the selection of the layers.

  - Task 5 (35 hours): Rotate the selected layer.

  - Task 6 (35 hours): Rotate the cube.

- **Interface:**
  - Task 7 (10 hours): Make the menus.
  - Task 8 (40 hours): Make a tutorial with indications at the top.
  - Task 9 (40 hours): Make an AI to solve the cube.
- **Control of the game:**
  - Task 10 (2 hours): Create a model of the virtual Rubik's cube in a game engine.
  - Task 11 (20 hours): Create the movements of the cube.
  - Task 12 (5 hours): Create the logic for when the cube is completed or a bad movement is done.
  - Task 13 (10 hours): Complete the remainder of the game.
- **Memory and presentation:**
  Task 14 (40 hours): Final memory.
  Task 15 (10 hours): Final presentation.

The total hours is 300.

There are different group tasks, so it is possible that tasks from different groups are made at the same time and before others of the same or different groups.

2.2 Resource Evaluation

This project has been possible thanks to the material offered by the Universitat Jaume I and my own, but in case it had to be done without any of them, this is an approximation of its cost.

- A laptop Acer with Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz 2.40 GHz, 8 GB of RAM and 512 GB of Hard Disk Storage Capacity with Windows 10. 750 €
- Oculus Quest, the device where the game is shown. 349 €.
- Cable Link, to connect the Oculus Quest into the computer. 99 €.
- Unity 2020.2.2f1 Personal, used to create the game [2]. Free.
- Blender, used to create some of the models [3]. Free.
- Visual Studio 2019, used to program the game [4]. Free.
- Inkscape, used to make the arrows [5]. Free.

Chapter 3

Game Design Document

Contents

This is the game design document, where the specifications of the game are described to understand how is it designed before more detailed functionalities be written.

3.1 Introduction

The title of this game tutorial is "Rubik kocka", which means Rubik's cube in Hungarian. That name was decided in homage for the creator of this 3D puzzle, Ernő Rubik, who is Hungarian.

In fact, this is really a game tutorial, whose aim is to help the user to solve the Rubik's cube with a virtual reality device like the Oculus Quest.

The main features are the central cube, that the user rotates, and the tutorial up to it, where the movements are shown to solve properly the cube.

3.2 Level Design

There are only two modes to play: a *Free* mode and a *Tutorial* mode. In the first mode initially there is a solved cube, and the user can rotate its faces and the cube itself, and also make random scrambles. In the other one the initial cube is previously scrambled, and the user can do the same except to scramble it. In addition, there is a tutorial to teach the user how to solve the cube.

3.3 Gameplay

In this game tutorial the player starts in the *Main menu* where there are four buttons. The first one is *Play*, and is to choose the *Free* mode or the *Tutorial* mode. The player can only interact directly with the cube, but according to the effected rotations, the tutorial can change erasing the solved steps. Moreover, there is a pause menu in both modes. Returning to the *Main menu*, the second button is *Controls*, that shows the indications of the controllers in an image. The third button is *Help*, and explains to the user the aim of each mode and some help in case of necessity. And the last button is *Exit*, with which the user can quit the game.

3.4 Art

As this is a game about the Rubik's cube, there is no need to add complex art and it is very geometric. There is a Rubik's cube with the six colours for the faces (white, red, blue, green, orange and yellow), and black for the inner parts of the cubelets (see Figure 3.1).

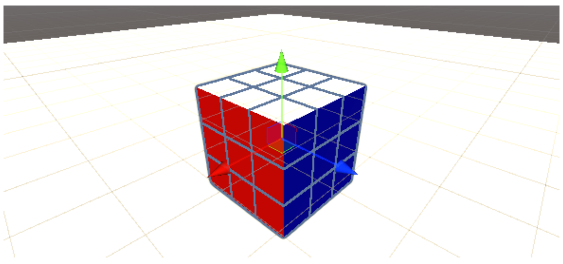


Figure 3.1: Rubik's cube.

For the tutorial the same colours are used for each square of the facelets (see Figure 3.2).

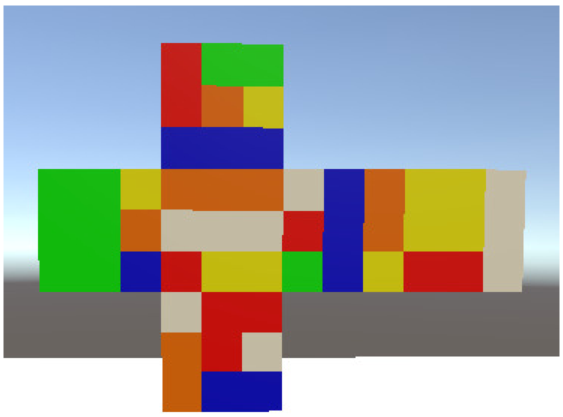


Figure 3.2: Example of the tutorial.

There are also some two different arrows indicating the direction of the rotation (see Figure 3.3).

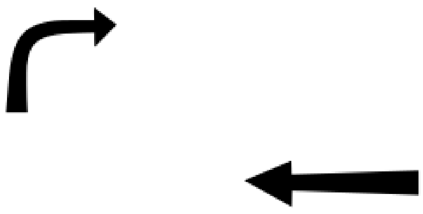


Figure 3.3: The two arrows used in the tutorial.

There are also two geometric models for the hands (see Figure 3.4).

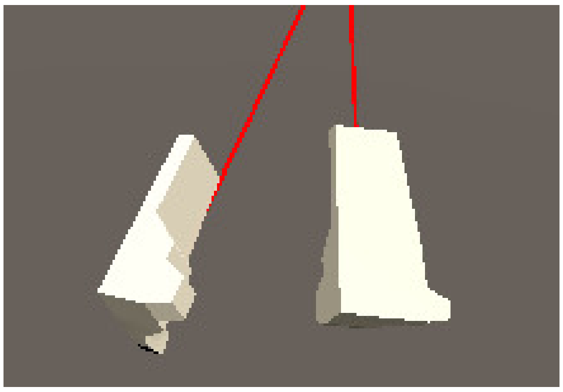


Figure 3.4: Two hands.

3.5 Sound and Music

There are only two pieces of music in this game, both for free use and without any cost: one for the menus, and the other one for the game. The first one is "Synthwave Vintage Future Synth 80s Retro Game Futuristic Music" of REDproductions [7], and the second one is "Cosmic Glow" of Andrewkn [8].

3.6 User Interface

The game starts with the main menu, where it shows four buttons that are *Play*, *Controls*, *Help* and *Exit* (see Figure 3.5).

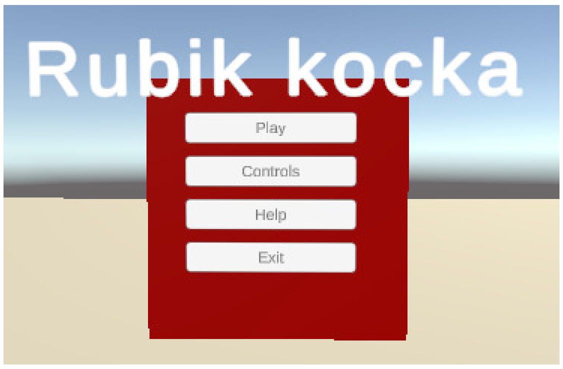


Figure 3.5: The *Main menu.*

Once the *Play* button is selected, there will be another menu that shows the mode to play (see Figure 3.6): *Free* and *Tutorial.* Also the *Go back* button. If each of those buttons are selected each game will start.

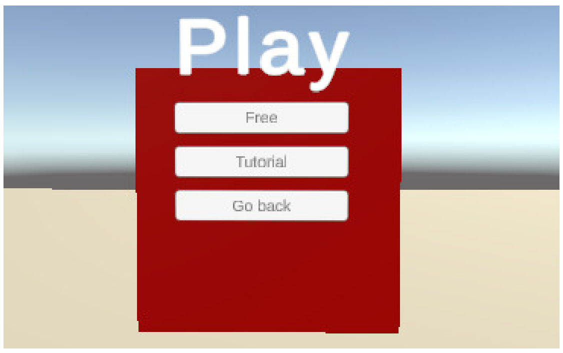
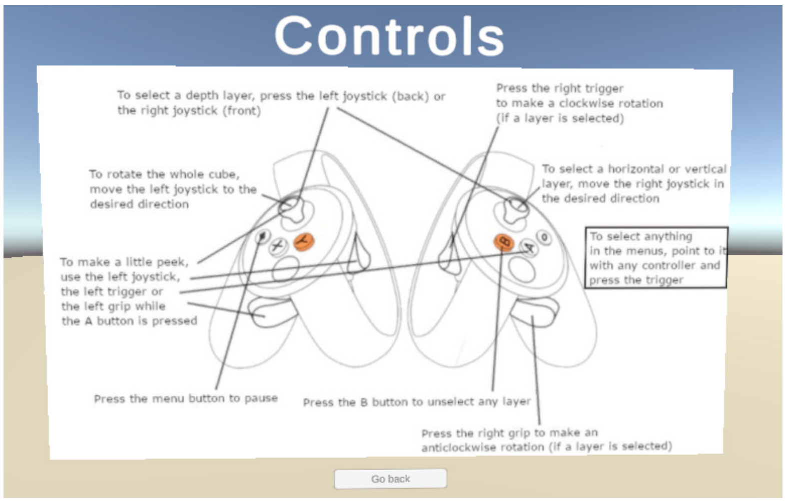


Figure 3.6: The *Play menu.*

If the *Controls* button is selected, a new menu is shown where the user can see a drawing of the controllers with some indications (see Figure 3.7).



To select a depth layer, press the left joystick (back) or the right joystick (front)

Press the right trigger to make a clockwise rotation (if a layer is selected)

To rotate the whole cube, move the left joystick to the desired direction

To select a horizontal or vertical layer, move the right joystick in the desired direction

To select anything in the menus, point to it with any controller and press the trigger

To make a little peek, use the left joystick, the left trigger or the left grip while the A button is pressed

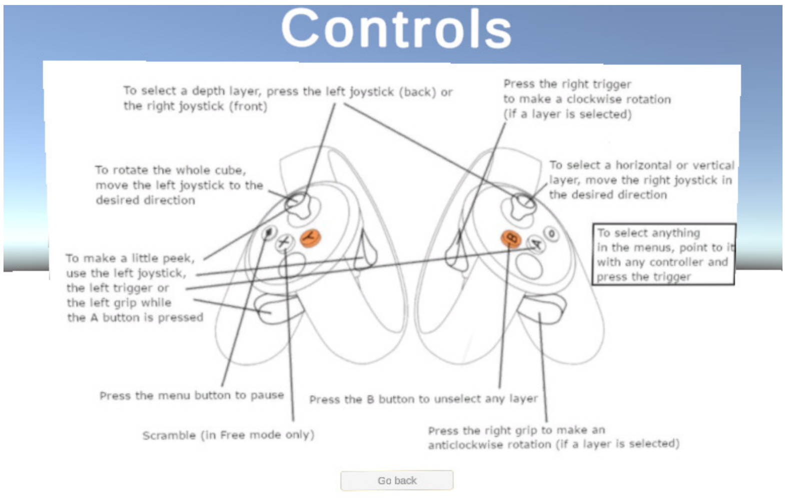
Press the menu button to pause

Press the B button to unselect any layer

Press the right grip to make an anticlockwise rotation (if a layer is selected)

Figure 3.7: The *Controls menu*.

It is needed to mention that the *Controls menu* of the pause in the Free mode (explained later) is different and shows also the indications of how to scramble the cube, which is not in the normal controls (see Figure 3.8).



To select a depth layer, press the left joystick (back) or the right joystick (front)

Press the right trigger to make a clockwise rotation (if a layer is selected)

To rotate the whole cube, move the left joystick to the desired direction

To select a horizontal or vertical layer, move the right joystick in the desired direction

To select anything in the menus, point to it with any controller and press the trigger

To make a little peek, use the left joystick, the left trigger or the left grip while the A button is pressed

Press the menu button to pause

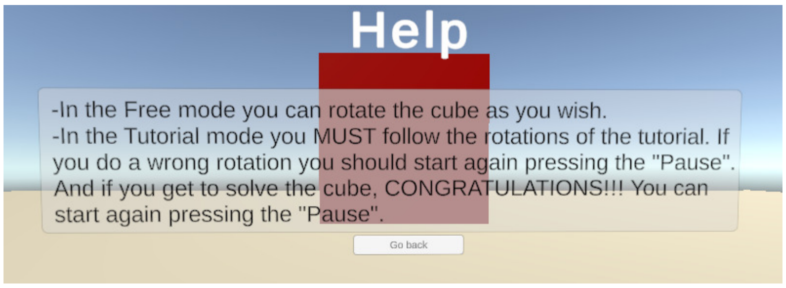
Press the B button to unselect any layer

Scramble (in Free mode only)

Press the right grip to make an anticlockwise rotation (if a layer is selected)

Figure 3.8: The *Controls menu* of the *Free* mode. The difference is below on the left, showing how to scramble with the X button.

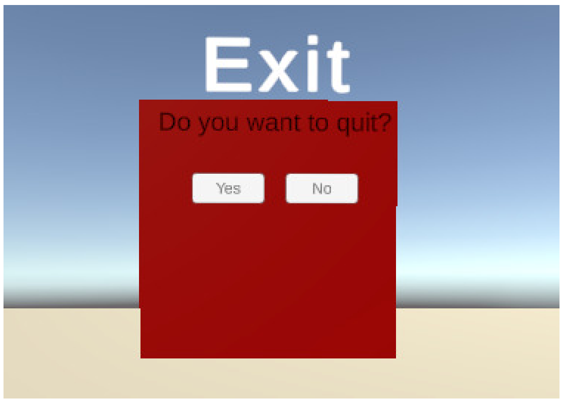
If the *Help* button is selected, a new screen appears, showing some help indications in case the user does a wrong rotation not fitting the indicated in the tutorial, and in case the user solves the cube (see Figure 3.9).



-In the Free mode you can rotate the cube as you wish.
-In the Tutorial mode you MUST follow the rotations of the tutorial. If you do a wrong rotation you should start again pressing the "Pause". And if you get to solve the cube, CONGRATULATIONS!!! You can start again pressing the "Pause".

Figure 3.9: The *Help menu*.

And finally the last button, *Exit*, asks the user to exit the game (see Figure 3.10).



Do you want to quit?

Figure 3.10: The *Exit menu*.

In addition, the *Pause menu* is similar to the *Main menu*, and is accessed in any of the two modes of the game. The difference is the first button, that is *Continue* instead of *Play* (see Figure 3.11).

3.7 Game Controls

The following are the controls of this game tutorial:

- To access the elements of the menu the user has to point with the pointer (a red laser) to the corresponding button and then press the trigger to click it.
- To select a layer of the cube, use the joysticks. Move the right joystick up or down to select a horizontal layer, or left or right to select a vertical layer. And click the left joystick to select a back layer and the right to select a frontal layer.

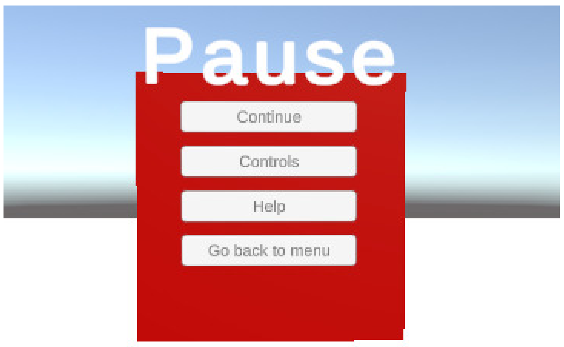


Figure 3.11: The *Pause menu* of the *Free* mode.

- Once selected a layer, to rotate with a clockwise direction, press the right trigger, and to rotate with an anticlockwise direction, the right grip.
- To unselect any layer, press the B button.
- To peek at the hidden face without rotating the cube, maintain the A button of the right controller while the left joystick, the left trigger or the left grip are used . Once released, the cube returns to its original orientation.
- To rotate the cube, use the left joystick, the left trigger and the left grip. This is different from the previous control because the cube changes its orientation.
- To access the pause menu, press the menu button of the left controller.
- In the *Free* mode, to scramble the cube press the X button.

Chapter 4

Functional and Technical Specification

Contents

Here the elements of the Game Design Document are more detailed.

4.1 System Design

4.1.1 Use case diagram

This is the use case diagram (see Figure 4.1).

4.1.2 Functional Requirements

The work of the game tutorial has been previously explained so there is no need for an explanation to understand the functional requirements.

Functional Requirements

With all this said, the functional requirements are described below:

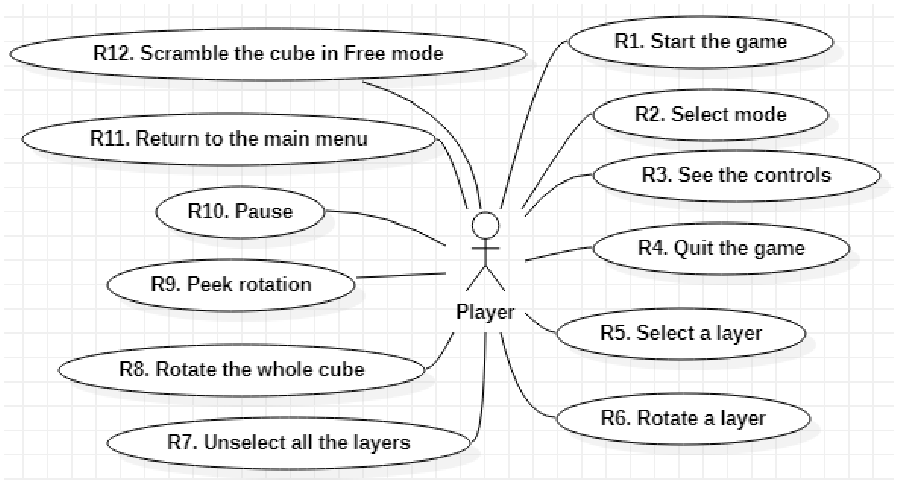


Figure 4.1: Use case diagram (made with *StarUML* [9]).

- **R1:** start the game.
- **R2:** select mode.
- **R3:** see the controls.
- **R4:** quit the game.
- **R5:** select a layer.
- **R6:** rotate a layer.
- **R7:** unselect all the layers.
- **R8:** rotate the whole cube.
- **R9:** peek rotation.
- **R10:** pause.
- **R11:** return to the *main menu*.
- **R12:** scramble the cube in *Free* mode

Non-Functional Requirements

And also, the non-functional requirements are some of the conditions on the game's design due to its characteristics. These are the following:

- **R13:** will be required virtual reality glasses.

- **R14:** the game is more a tutorial or simulator than a game.
- **R15:** it is a very artistically poor game.
- **R16:** the controls may be complicated.

Requirement:	R1
Actor:	Player
Description:	The player can start the game
Preconditions:	1. The player is in the *Main menu*
Normal sequence:	1. The player selects the button *Play*
Alternative sequence:	• The player selects the button *Controls* • The player selects the button *Options* • The player selects the button *Exit*

Table 4.1: Case of use «UC1. Start the game»

Requirement:	R2
Actor:	Player
Description:	The player can select either the *Free mode* or the *Tutorial mode*
Preconditions:	1. R1. The player is in the *Play menu*
Normal sequence:	1. The player selects one of the two available modes
Alternative sequence:	1. The player goes back to the *Main menu*

Table 4.2: Case of use «UC2. Select mode»

Requirement:	R3
Actor:	Player
Description:	The player can see the controls
Preconditions:	1. R1. The player is in the *Main menu* 2. R10. The player is in the pause
Normal sequence:	None
Alternative sequence:	1. The player presses the return button

Table 4.3: Case of use «UC3. See the controls»

Requirement:	R4
Actor:	Player
Description:	The player can quit the game
Preconditions:	1. R1. The player is in the *Main menu*
Normal sequence:	1. The player selects the button *Exit* 2. The player selects the option *Yes*
Alternative sequence:	1. The player selects the option *No*

Table 4.4: Case of use «UC4. Quit the game»

Requirement:	R5
Actor:	Player
Description:	The player can select a layer
Preconditions:	1. R2. The player is playing in one of the available modes
Normal sequence:	1. The player moves the right joystick or presses either the left or the right joystick to change the selected layer
Alternative sequence:	None

Table 4.5: Case of use «UC5. Select a layer»

Requirement:	R6
Actor:	Player
Description:	The player can rotate the selected layer
Preconditions:	1. R5. A layer is already selected 2. The cube must not be rotating
Normal sequence:	1. The player presses the trigger to rotate clockwise 2. The player presses the grip to rotate anticlockwise
Alternative sequence:	None

Table 4.6: Case of use «UC6. Rotate a layer»

Requirement:	R7
Actor:	Player
Description:	The player can unselect any selected layer
Preconditions:	1. R5. A layer is already selected
Normal sequence:	1. The player presses the B button of the right controller
Alternative sequence:	None

Table 4.7: Case of use «UC7. Unselect all the layers»

Requirement:	R8
Actor:	Player
Description:	The player can rotate the whole cube
Preconditions:	1. R2. The player is in any of the chosen modes 2. The cube must not be rotating
Normal sequence:	1. The player moves the left joystick
Alternative sequence:	None

Table 4.8: Case of use «UC8. Rotate the whole cube»

Requirement:	R9
Actor:	Player
Description:	The player can peek briefly a hidden face of the cube
Preconditions:	1. R2. The player is in any of the chosen modes 2. The cube must not be rotating
Normal sequence:	1. The player moves the left joystick or presses the left trigger or the left grip while maintaining the A button of the right controller
Alternative sequence:	None

Table 4.9: Case of use «UC9. Peek rotation»

Requirement:	R10
Actor:	Player
Description:	The player can pause the game in any moment
Preconditions:	1. R2. The player is in any of the chosen modes
Normal sequence:	1. The player presses the menu button of the left controller
Alternative sequence:	None

Table 4.10: Case of use «UC10. Pause»

Requirement:	R11
Actor:	Player
Description:	The player can return to the main menu in the pause
Preconditions:	1. R10. The player is in the pause
Normal sequence:	1. The player selects the *Main menu* option of the pause
Alternative sequence:	None

Table 4.11: Case of use «UC11. Return to the *Main menu*»

Requirement:	R12
Actor:	Player
Description:	The player can scramble the cube with the X button
Preconditions:	1. The player is in the *Free* mode
Normal sequence:	1. The cube scrambles randomly
Alternative sequence:	None

Table 4.12: Case of use «UC12. Scramble the cube in the *Free* mode»

4.2 Game Mechanics

The main working of the game has been previously explained, how to rotate the cube and its layers, and what the tutorial represents. The remaining to explain is the method shown in the tutorial to help to solve the cube. Basically it is done with an algorithm, it is, a sequence of movements that teaches how to solve the cube. This is coded and represented in the tutorial.

4.3 User Interface

4.3.1 Flowchart

The flowchart of the menu user interface is in Figure 4.2.

The flowchart of the pause user interface is in Figure 4.3.

4.3.2 Functional Requirements

- **Menu screen:** it is just a screen with a menu where the player can select the different options shown.

- **Free screen:** here the player can rotate the cube freely. Moreover, the user can also scramble the cube randomly.

- **Tutorial screen:** they are the same as *Free* except that the scramble is inactive and have some additions: the player can rotate the cube according to the tutorial. The tutorial will guide the player how to solve the Rubik's cube.

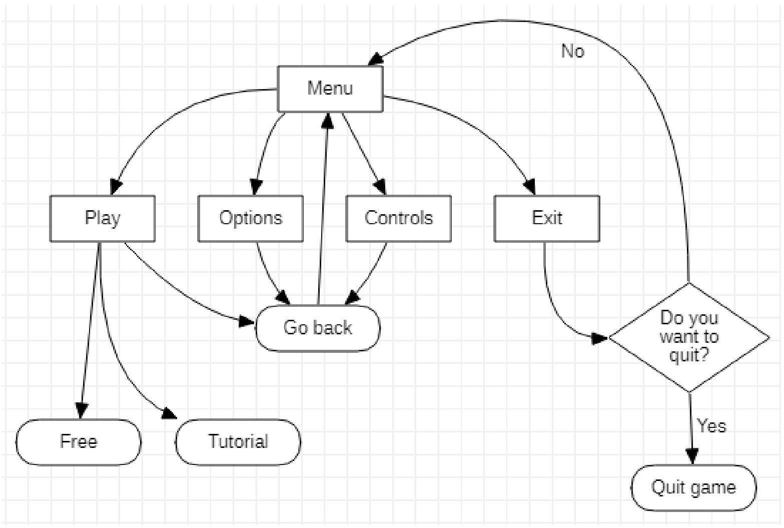


Figure 4.2: Menu flowchart (made with *StarUML* [9]).

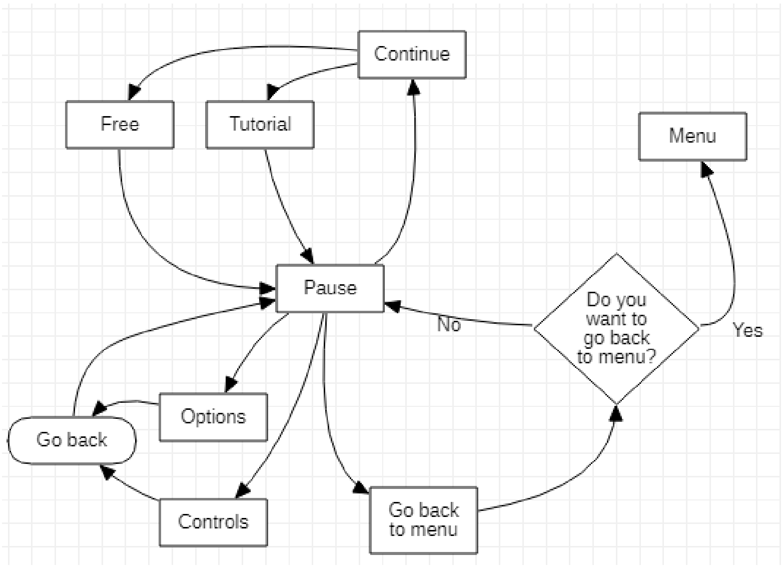


Figure 4.3: Pause flowchart (made with *StarUML* [9]).

Except the menu screen, all of the others have access to a pause window, with which the player can turn back to the menu.

4.3.3 Mockups

The following images are some mockups of the different screens and windows of the game:

4.3.4 Graphic User Interface Objects

The elements of the graphic user interface are the default buttons of Unity (see Figure 4.4).

Figure 4.4: Default button of Unity.

4.4 Art

This is the art of the game and its functionalities explained.

4.4.1 2D Art and Animation

The 2D art there is in this game are the arrows of the tutorial that shows the direction to rotate each layer of the cube. To avoid having 9 different arrows, only one curved arrow and one straight arrow were enough, modified in rotation and position to get the others.

4.4.2 3D Art and Animation

The 3D art of this game will be shown next:

There is a Rubik's cube model made of 27 cubelets done with separated squares in 3D space shaping a cube. This is all well ordered in the hierarchy, so inside each cubelet folder are the six faces of that cubelet, with its corresponding colours (see Figure 5.2).

To be able to rotate they are grouped by dynamic layers of 3 x 3 cubelets and therefore, rotate by their central axis. And to show a selected layer, the faces of the colours of that layer change their colours to highlight them.

The art of the tutorial is a netted cube obtained from the cube with a 3 x 3 square with 9 little squares by face of the net, in which the squarelets have the colours of the

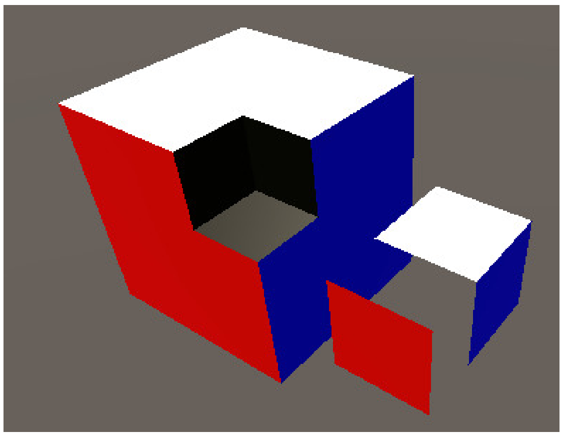


Figure 4.5: Model of the cube splitted to show the parts.

cube and its calculated sequences (see Figure 4.6).

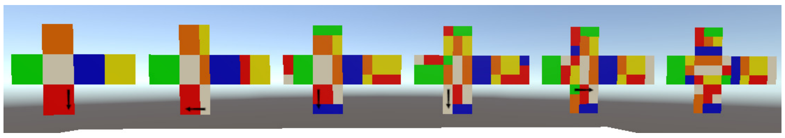


Figure 4.6: Sequence of the tutorial.

The remaining 3D art of the game are the hands of the user, that are 3D models of two hands and represent the position and orientation of the user's hands that hold the controllers.

The animations are controlled by code and are the rotations of each layer of cubelets around a central axis. Also the rotation of the whole cube.

4.5 Level Requirements

Here the diagram of the levels is shown in the Figure 4.7

In the Table 4.13 is the Asset Revelation Schedule is shown.

Menu	Cube	Tutorial	Menu list
Free	Cube	-	Pause
Tutorial	Cube	Tutorial	Pause

Table 4.13: Asset Revelation Schedule

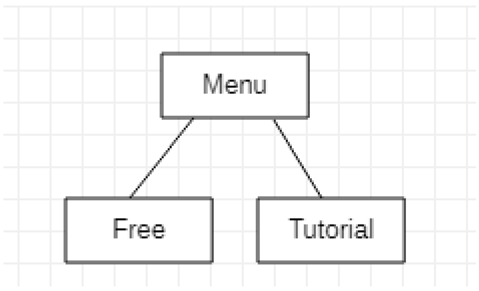


Figure 4.7: Diagram of the levels

In this section the level design is shown as follows:

- **Menu:** This is a scene with a menu. In the background there is a Rubik's cube.
- **Free:** This is a scene with only a Rubik's cube.
- **Tutorial:** This is the same with the addition of a tutorial.

CHAPTER 5

Results

Contents

In this chapter the work developed during these months is described. The changes on the planning in relation to the original planning are also presented, and the carried variations are also explained. Finally, the results of this Final Degree Project will be described in the end.

5.1 Work Development

As said in the introduction of this chapter, here only will be described the process of the work developed.

5.1.1 Creation of the Rubik's cube

At the beginning of this project, the first thing to develop was the Rubik's cube, both the model and the rotations. So the first to consider was how to make a model of a Rubik's cube, in which there are 27 cubes, all of them related and that can change its position and orientation all around the cube.

It started with an empty game object with 27 cubelets to at least have an idea of the shape, and then each cubelet was changed with another empty game object in which there were only the faces of each cubelet, with its corresponding colour (see the whole cube in Figure 5.1, and a sample of how it has been created in Figure 5.2). So the following was to evaluate how to make them rotate not only about one axis, but also about different axis because of the different positions every cubelets can attain.

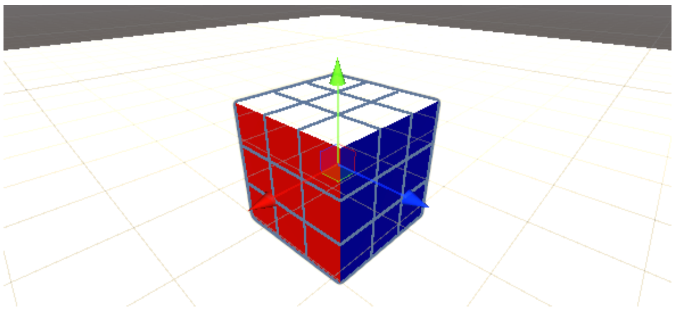


Figure 5.1: Initial Rubik's cube.

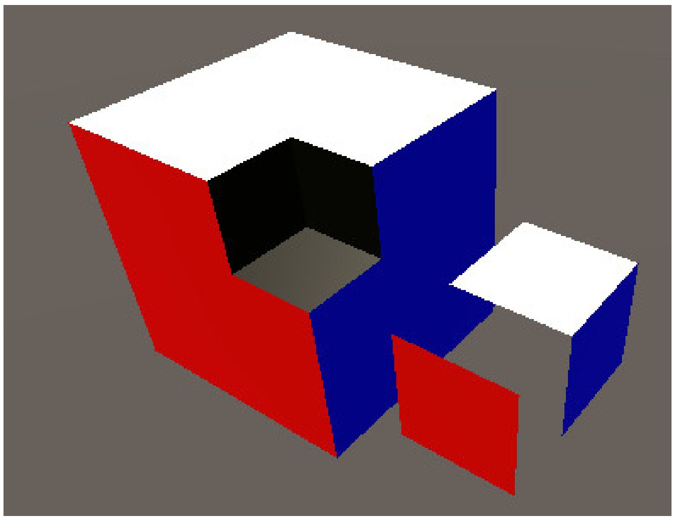


Figure 5.2: A sample of how the cube is done.

To make this the idea was to create by code nine blocks of 3 x 3 cubelets each one. Those nine blocks (which represent the layers) are three by depth, three by height and three by width. So the best was to centre all the cubelets axis components in -1, 0 or 1, and compare its x, y and z axis positions with those numbers. Thus, each block would have their corresponding nine cubelets (for example, the block of the front (seen in Figure 5.3) is the one whose cubelets' x axis component equals 1).

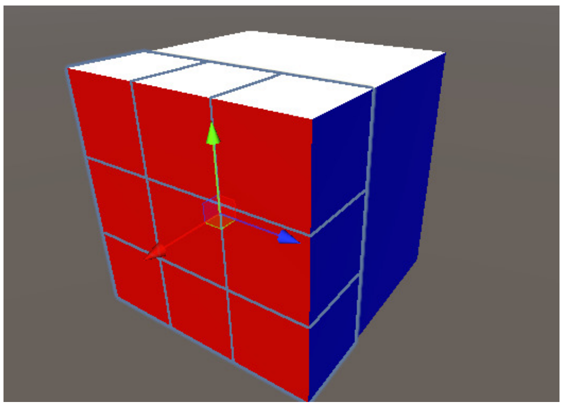


Figure 5.3: Example that shows the front block, whose each cubelets' x axis equals 1.

Only one thing remains, and that is that if a block rotates, it should change its cubelets. A fast way to implement this is just to empty the blocks and refill again with the position changed cubelets.

Once this was achieved, the following was to make each layer rotate around its centre, so to do that all the cubelets of a corresponding block had to be traversed and rotate them around the centre axis of that block. Then a complete rotation would have 90 degrees (see an example in Figure 5.4).

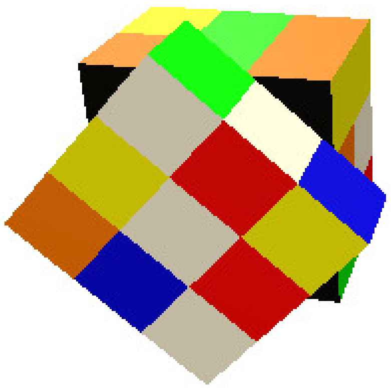


Figure 5.4: Example showing the rotation of the front block.

Finally, in order to interact with the cube in the computer was implemented an input of the numeric keyboard that represents the rotations, and the spacebar that represents the direction of rotation. To activate a rotation, when a key is pressed its corresponding rotation activates and the block rotates.

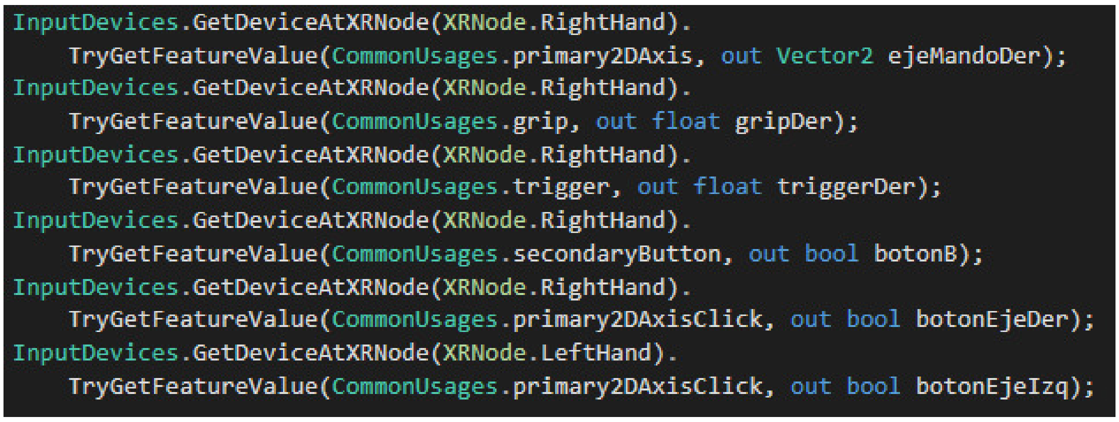
5.1.2 Learning about the virtual reality

At first, the material used to learn how to use Unity with virtual reality were some video tutorials for beginners [10] recommended by the professor. Although there were various assets for Unity to use virtual reality, it was enough with just XR Interaction Toolkit [11], which is a free Unity asset. Moreover, there is many documentation about XR Interaction Toolkit and many tutorials to solve anything, unlike other assets, many of them expensive.

So the next thing to do is adapt the project into the XR Interaction Toolkit with the virtual reality environment, and the only thing to change was needed to change the camera and have an XR Rig, where the head camera would be, and automatically the hands would also appear, that are controlled by the controllers. What left now was to adapt the interactions, it is, how to interact with the cube and rotate it.

5.1.3 Making the interaction with the cube

The part regarding the buttons was similar to the input of the computer keyboard, except the manner to make the inputs by code (see Figure 5.5), and despite the few buttons of the controllers, it was enough for the project.



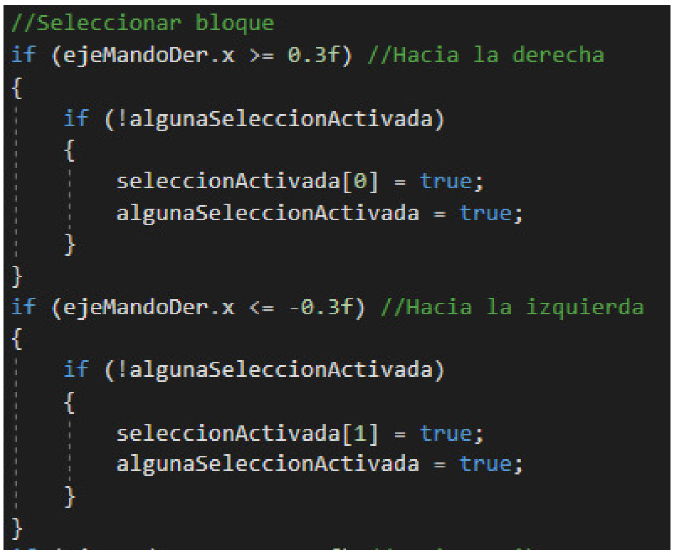
```
InputDevices.GetDeviceAtXRNode(XRNode.RightHand).
    TryGetFeatureValue(CommonUsages.primary2DAxis, out Vector2 ejeMandoDer);
InputDevices.GetDeviceAtXRNode(XRNode.RightHand).
    TryGetFeatureValue(CommonUsages.grip, out float gripDer);
InputDevices.GetDeviceAtXRNode(XRNode.RightHand).
    TryGetFeatureValue(CommonUsages.trigger, out float triggerDer);
InputDevices.GetDeviceAtXRNode(XRNode.RightHand).
    TryGetFeatureValue(CommonUsages.secondaryButton, out bool botonB);
InputDevices.GetDeviceAtXRNode(XRNode.RightHand).
    TryGetFeatureValue(CommonUsages.primary2DAxisClick, out bool botonEjeDer);
InputDevices.GetDeviceAtXRNode(XRNode.LeftHand).
    TryGetFeatureValue(CommonUsages.primary2DAxisClick, out bool botonEjeIzq);
```

Figure 5.5: Example of different button inputs.

Therefore, next was to be able to choose the adequate layer to rotate:

In order to make that, a system to choose handly the layers was done using the joysticks, and instead of having a button for each layer, previously the user would select the layer to rotate and then press either the trigger or the grip to do a clockwise or anticlockwise rotation. So if the user moves the right joystick up or down, the selected layers would be from the vertical axis; moving it to the right or to the left, the selected layers would be from the horizontal axis; and for the depth axis, pressing the left joystick selects the back layers and with the right, the front.

The functionality is explained next:
There is a function called *seleccionarYRotarBloques()* where there are the inputs of the buttons. Then, it checks if the button is correctly pressed and activates the adequate boolean (0, 1, 2, 3, 6 and 7 are to select the layer, and 4 and 5 to make the rotation), like in the Figure 5.6.



```
//Seleccionar bloque
if (ejeMandoDer.x >= 0.3f) //Hacia la derecha
{
    if (!algunaSeleccionActivada)
    {
        seleccionActivada[0] = true;
        algunaSeleccionActivada = true;
    }
}
if (ejeMandoDer.x <= -0.3f) //Hacia la izquierda
{
    if (!algunaSeleccionActivada)
    {
        seleccionActivada[1] = true;
        algunaSeleccionActivada = true;
    }
}
```

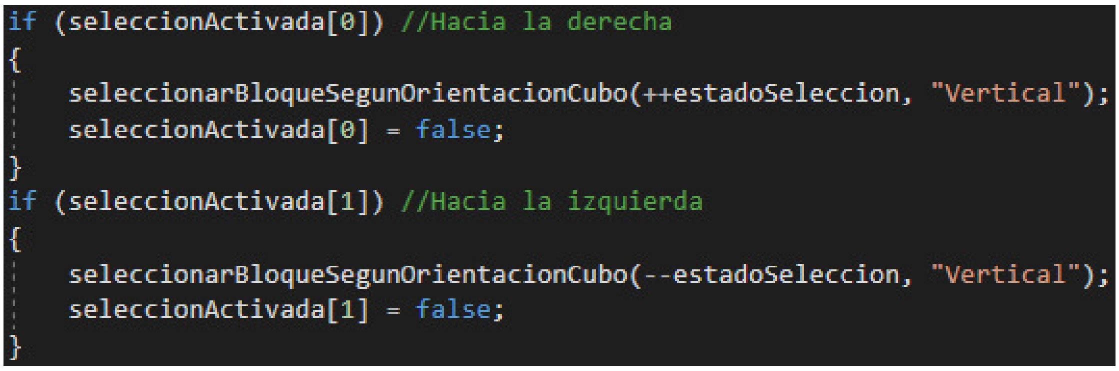
Figure 5.6: Extraction of part of the code to select the layers.

According to the activated selection, if it is one of the selections', a new function *seleccionarBloquesSegunOrientacionCubo(int estado, string seleccion)* is called with some parameters (see Figure 5.7). That function will return the selected block according to a state (-1, 0 or 1) and a string that can be "Horizontal", "Vertical" or "Profundidad".

But if the selection is one that activates the rotation, there is a code that activates the adequate rotation and with a determined direction. The rotations have been previously explained, and there is an example in Figure 5.4.

With this, the function *seleccionarBloqueSegunOrientacioncubo(int estado, string seleccion)* will be explained:

It starts checking the state, which must be between 1 and -1, and then the different options are to be coded.



```
if (seleccionActivada[0]) //Hacia la derecha
{
    seleccionarBloqueSegunOrientacionCubo(++estadoSeleccion, "Vertical");
    seleccionActivada[0] = false;
}
if (seleccionActivada[1]) //Hacia la izquierda
{
    seleccionarBloqueSegunOrientacionCubo(--estadoSeleccion, "Vertical");
    seleccionActivada[1] = false;
}
```

Figure 5.7: Extraction of part of the code to call the function.

Furthermore, to highlight the selected layer the colours of each of the facelets are changed to lighter ones (except the white colour, that darkens a little bit) to differentiate them (shown in Figure 5.8). First it returns the original colour of any selected layer (first *foreach* of any *estado* in Figure 5.8), and then, it traverses in the second foreach only the cubelets of the selected block to change its colour (also seen in the same Figure 5.8). It was just to access the mesh component of each facelet and assign a new colour previously chosen according to the original colour of the facelet.

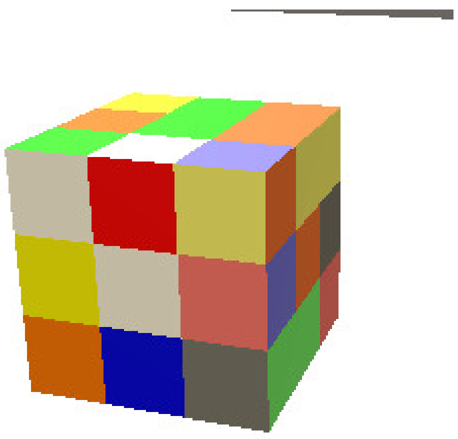
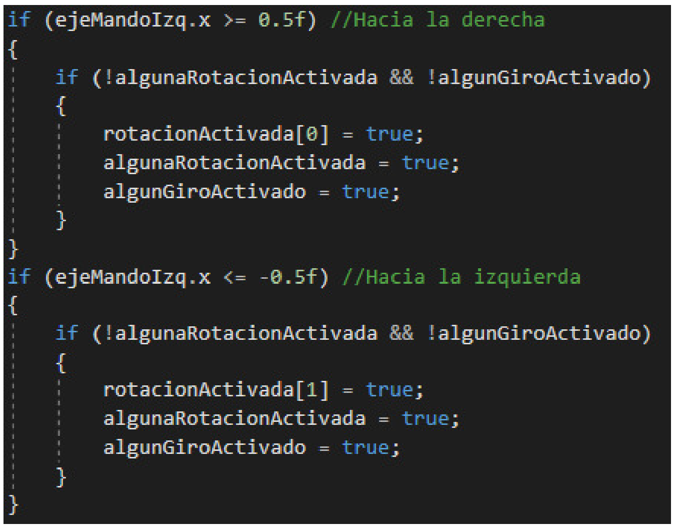


Figure 5.8: Example of the cube with the right layer selected.

Once a layer is selected, to rotate it the right trigger is to make a clockwise rotation and the right grip to the anticlockwise rotation.

Finally, to unselect any layer it was the same but changing from the modified colour to the original, and this is done with the B button.

On the other hand, it was useful not only to rotate a layer but also the cube or just make a little peek of another face. To do that the buttons and the joystick of the left controller in combination with the A button of the right controller were used. So to rotate the cube and change its orientation, the right joystick was needed. This is done in the function *rotarCubo()*, and is similar to the rotation of the layers. First there is the activation of the code of the desired rotation (from 0 to 6, that is towards right, left, up, back, anticlockwise and clockwise), as seen in Figure 5.9, and then the rotations themselves (Figure 5.10).

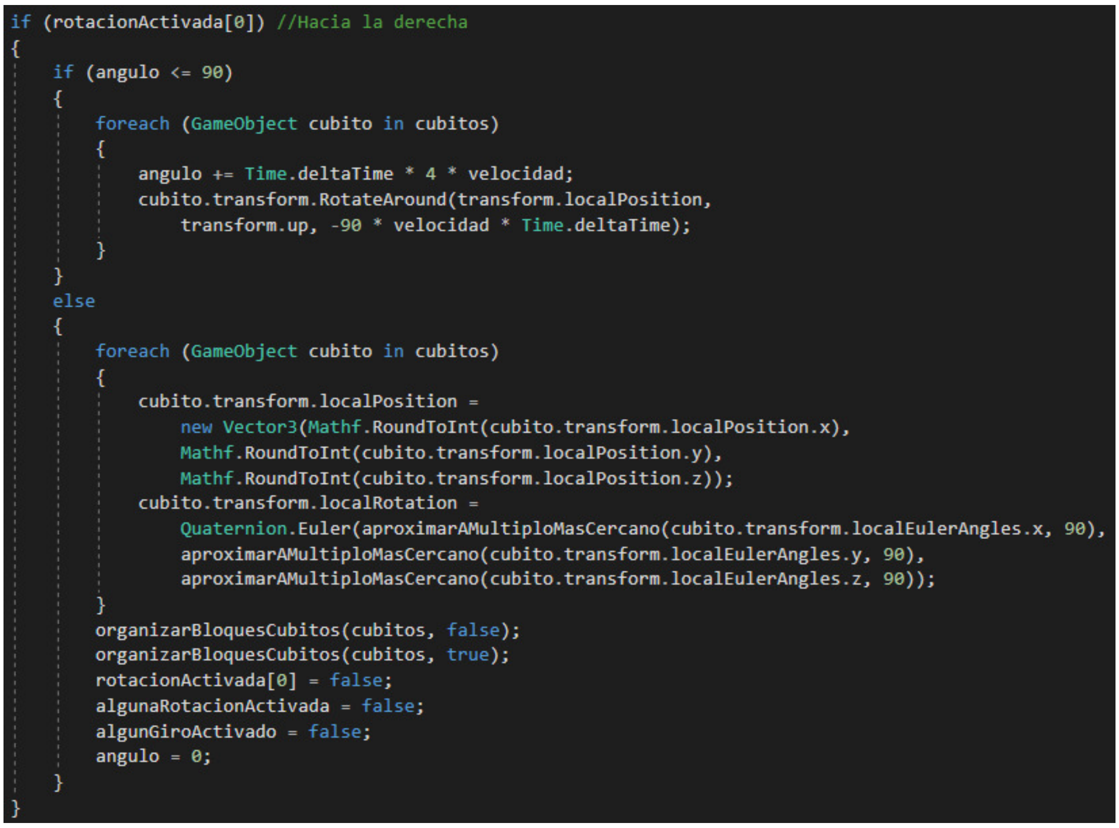


```
if (ejeMandoIzq.x >= 0.5f) //Hacia la derecha
{
    if (!algunaRotacionActivada && !algunGiroActivado)
    {
        rotacionActivada[0] = true;
        algunaRotacionActivada = true;
        algunGiroActivado = true;
    }
}
if (ejeMandoIzq.x <= -0.5f) //Hacia la izquierda
{
    if (!algunaRotacionActivada && !algunGiroActivado)
    {
        rotacionActivada[1] = true;
        algunaRotacionActivada = true;
        algunGiroActivado = true;
    }
}
```

Figure 5.9: Example of code showing how to activate the rotation of the whole cube towards the right and the left.

And to make a little peek, it was the same but while maintaining the A button pressed and moving the joystick or pressing the left trigger or the left grip.

This code is very short, as it is just a simple transform of the angles (see Figure 5.11).



```
if (rotacionActivada[0]) //Hacia la derecha
{
    if (angulo <= 90)
    {
        foreach (GameObject cubito in cubitos)
        {
            angulo += Time.deltaTime * 4 * velocidad;
            cubito.transform.RotateAround(transform.localPosition,
                transform.up, -90 * velocidad * Time.deltaTime);
        }
    }
    else
    {
        foreach (GameObject cubito in cubitos)
        {
            cubito.transform.localPosition =
                new Vector3(Mathf.RoundToInt(cubito.transform.localPosition.x),
                Mathf.RoundToInt(cubito.transform.localPosition.y),
                Mathf.RoundToInt(cubito.transform.localPosition.z));
            cubito.transform.localRotation =
                Quaternion.Euler(aproximarAMultiploMasCercano(cubito.transform.localEulerAngles.x, 90),
                aproximarAMultiploMasCercano(cubito.transform.localEulerAngles.y, 90),
                aproximarAMultiploMasCercano(cubito.transform.localEulerAngles.z, 90));
        }
        organizarBloquesCubitos(cubitos, false);
        organizarBloquesCubitos(cubitos, true);
        rotacionActivada[0] = false;
        algunaRotacionActivada = false;
        algunGiroActivado = false;
        angulo = 0;
    }
}
```

Figure 5.10: Example of code that shows the rotation of the cube towards the right.

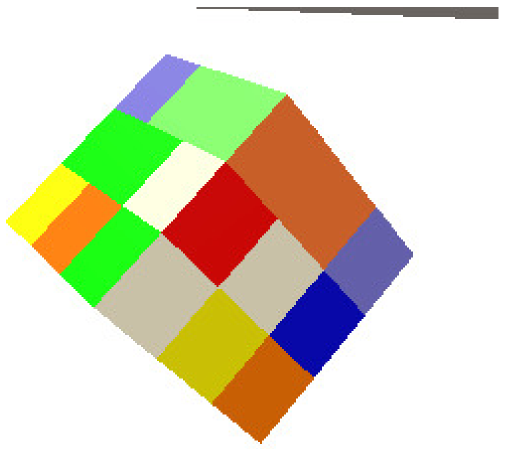
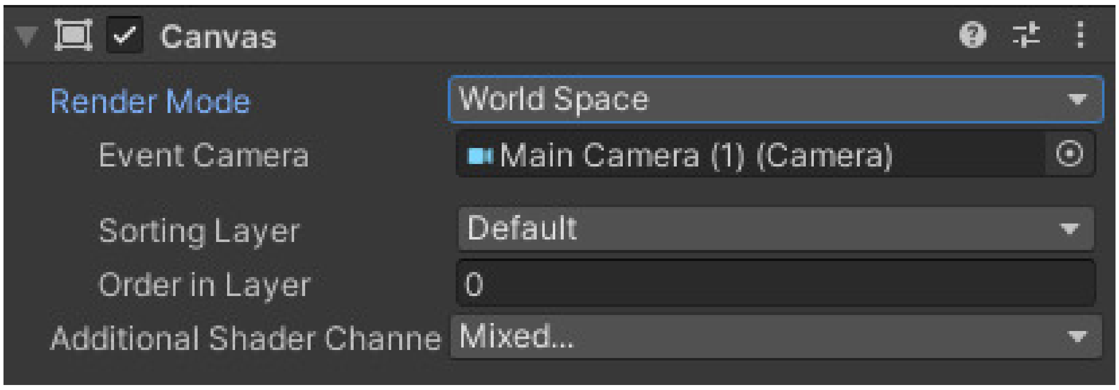


Figure 5.11: Peek rotation of the cube.

To end, in the case of the *Free* mode a button to scramble the cube was created. To activate it the user has to press the X button of the left controller and there will be a random list of movements that will activate the necessary rotations.

5.1.4 Creation of the menus

The creation of the menus is a canvas prepared for the virtual environment. The difference with the usual canvas is on the camera, because the canvas should appear in 3D and not in a fixed screen, so it is needed to render in *World Space* instead of the default *Screen Space* (see Figure 5.12).



Canvas

Render Mode: World Space
Event Camera: Main Camera (1) (Camera)
Sorting Layer: Default
Order in Layer: 0
Additional Shader Channe: Mixed...

Figure 5.12: World Space render mode of the canvas.

All the menus are made with the Unity user interface components and are very similar. Furthermore, because this part does not require technical explanations and the system of the menus have been already explained in the GDD, this section finishes here to pass to explain the creation of the tutorial.

5.1.5 Creation of the tutorial

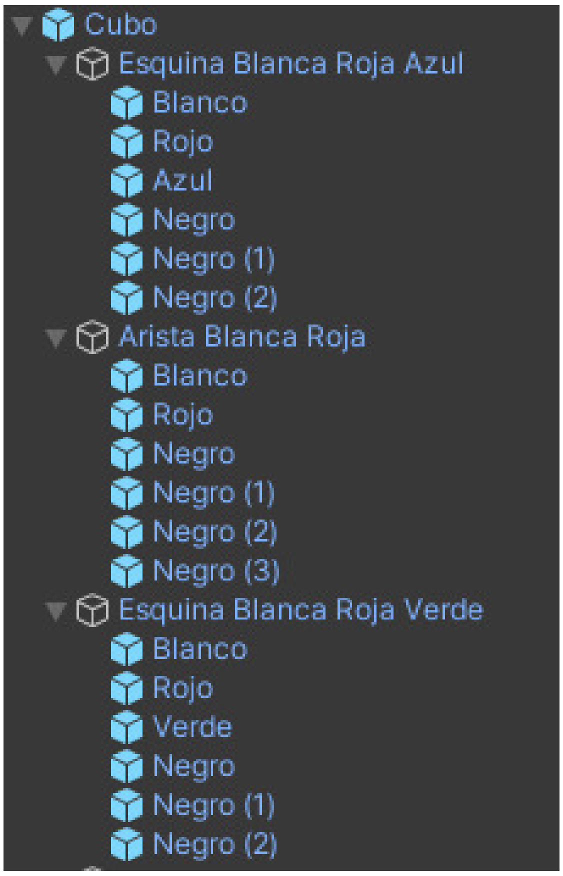
Once this is all achieved, the next and one of the most important parts of the game is the tutorial.

The main idea is to graphically show the steps of the algorithms with arrow indications, and this is based on the tutorial of the book "El Cubo Mágico" [1], because it is a very graphical method to learn from. But in the game, instead of only showing the main face the net of the whole cube is shown in each step, so it is easier to understand what happens in the entire cube.

To start creating the tutorial, the first was to create a new empty game object at the top of the screen with nine squarelets, altogether shaping a 3 x 3 bigger and whole square. They were initially coloured in black.

After that, they had to represent the colour of each facelet, in this case of the frontal face, and for that, there was needed a way to access the facelets of the cube. To do that a map of all the facelets of the cube was done, not only to represent the faces in the tutorial but also to make the calculi of the next movements of the tutorial.

Unity traverses the game objects in order in the hierarchy (see Figure 5.13), so can be taken advantage of it to get a map from the order of the facelets of the cube.



```
Cubo
  Esquina Blanca Roja Azul
    Blanco
    Rojo
    Azul
    Negro
    Negro (1)
    Negro (2)
  Arista Blanca Roja
    Blanco
    Rojo
    Negro
    Negro (1)
    Negro (2)
    Negro (3)
  Esquina Blanca Roja Verde
    Blanco
    Rojo
    Verde
    Negro
    Negro (1)
    Negro (2)
```

Figure 5.13: A sample of the order of the facelets in the hierarchy.

Thus, the order would be written on a piece of paper to facilitate the calculi of each step. With all that, the mapping is initially done in the *Start()* part of the code from the name of each facelet and then added to a list.

And also, six little arrays that represent each face would be created to indicate the numbers of the list. This is to later make the calculi and to represent it in the tutorial.

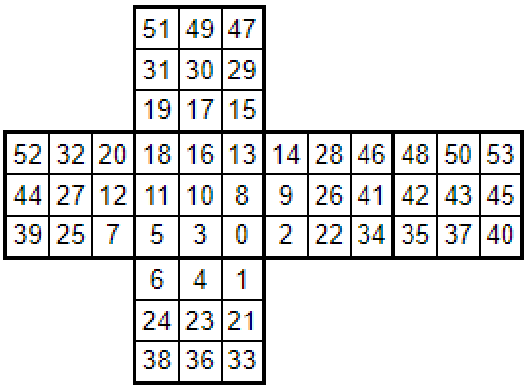
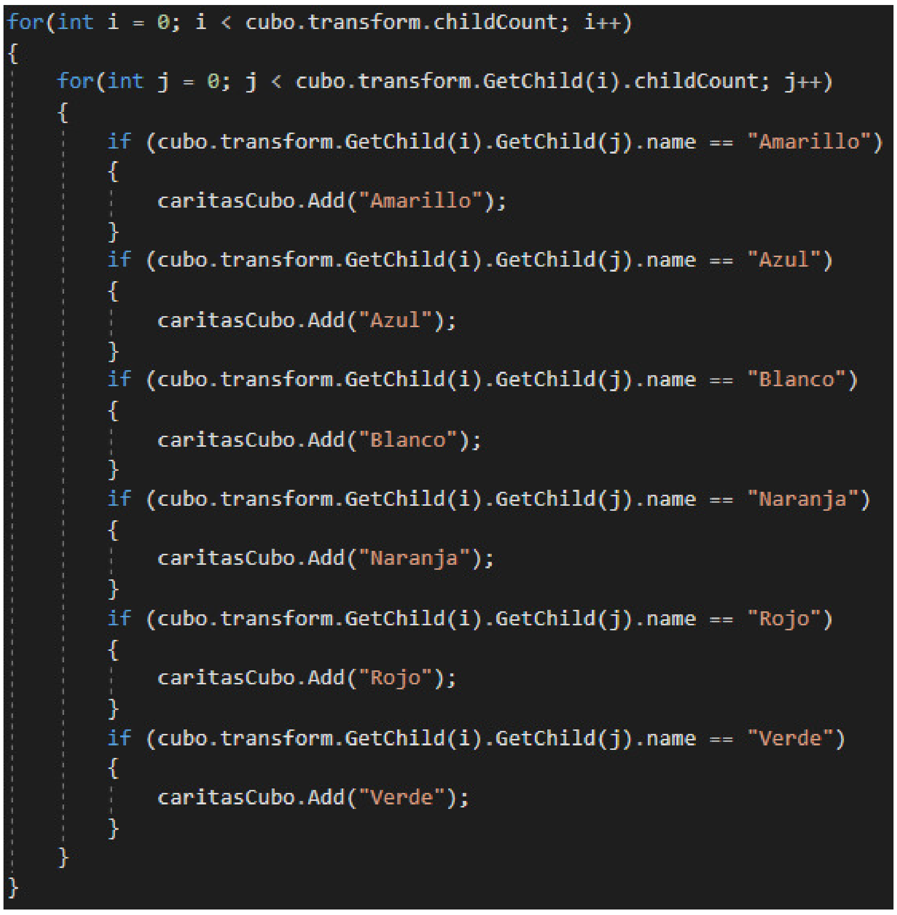


Figure 5.14: The order shown in a netted cube .



```
for(int i = 0; i < cubo.transform.childCount; i++)
{
    for(int j = 0; j < cubo.transform.GetChild(i).childCount; j++)
    {
        if (cubo.transform.GetChild(i).GetChild(j).name == "Amarillo")
        {
            caritasCubo.Add("Amarillo");
        }
        if (cubo.transform.GetChild(i).GetChild(j).name == "Azul")
        {
            caritasCubo.Add("Azul");
        }
        if (cubo.transform.GetChild(i).GetChild(j).name == "Blanco")
        {
            caritasCubo.Add("Blanco");
        }
        if (cubo.transform.GetChild(i).GetChild(j).name == "Naranja")
        {
            caritasCubo.Add("Naranja");
        }
        if (cubo.transform.GetChild(i).GetChild(j).name == "Rojo")
        {
            caritasCubo.Add("Rojo");
        }
        if (cubo.transform.GetChild(i).GetChild(j).name == "Verde")
        {
            caritasCubo.Add("Verde");
        }
    }
}
```

Figure 5.15: Code to get the facelets in the order of the hierarchy in the inspector.

Once done this, the next thing to do was to colour the squarelets of the tutorial according to the name of the colours already obtained in the previous list (see Figure 5.16.

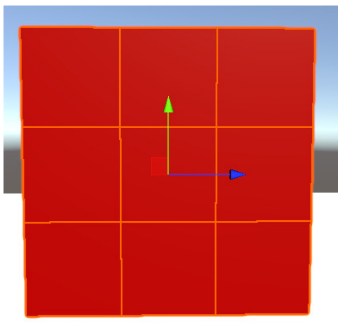


Figure 5.16: A sample of the front face represented in the tutorial.

Now a single facelet is shown in the tutorial, and the next was to get multiple copies of it. This was made making as many copies of the face as needed and translated horizontally to the right separated by a little gap.

Next, each copy should show step by step the theoretical movements to make. To do this there is another function, *giroTutorial(int giro, int sentido)*, that calculates the next position according to a given rotation and direction. It is just done having calculated the next position a facelet would achieve.

Once done the calculi, the calculated faces would be added in order to the tutorial (as seen in Figure 5.17).

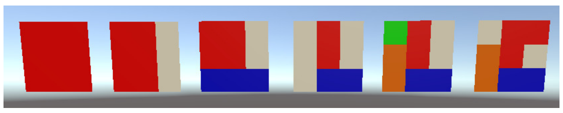


Figure 5.17: A sequence of the tutorial showing the movements of the front face step by step.

With this all done, the process to make the net of the cube was the same, but instead of only the front face, all the faces shaping the net (see Figure 5.18).

However, to make the tutorial useful not only the changes should be made, but also some type of indications to the user: the arrows. They were created with Inkscape [5], and with geometric transforms done in the function *Flecha(int giro, int sentido)* I added them to the tutorial the same way I did with the faces, but in a previous position. That is because the arrows indicate the action to do to be shown in the next movement (see

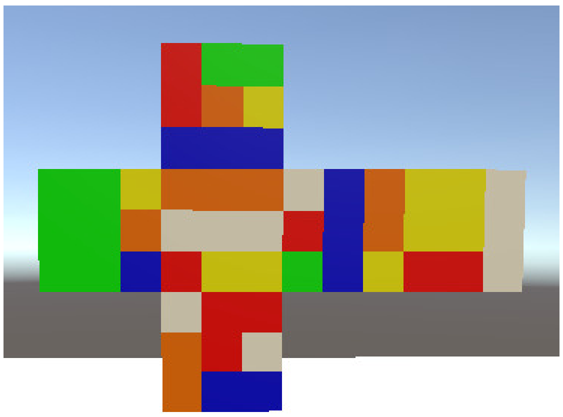


Figure 5.18: An example of a netted cube of the tutorial representing one movement.

Figure 5.19).

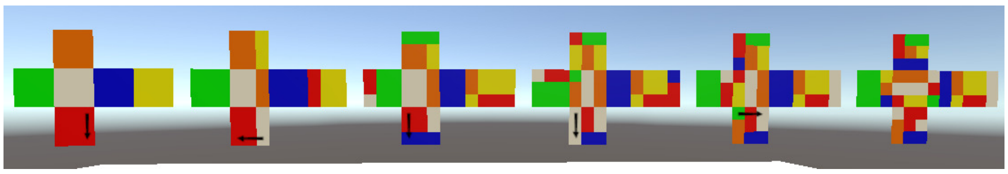


Figure 5.19: The same sequence of the Figure 5.17 but showing all the faces with netted cubes.

And finally, what lasts is to check if the movement done by the user to the cube corresponds to the indicated from the tutorial.

5.1.6 Implementation of the algorithm to solve the cube

The final step is to make the algorithms to solve the cube, and for that the official notation [12] will be used.

This is a set of letters that indicate either the rotation of the layers and the rotation of the whole cube:

- F (front), R (right), U (up), L (left), B (back) and D (down) are for the rotation of the faces.
- M (middle, same direction as an L rotation), E (equatorial, same direction as a D rotation) and S (standing, same direction as an F rotation) are for the rotation of the middle layers.

- x (same direction as an R rotation), y (same direction as a U rotation) and z (same direction as an F rotation) are for the rotation of the whole cube along the axis.
- The ' after a letter means that the rotation is anticlockwise.
- The 2 after a letter means that the rotation is double.

It is important to mention that the clockwise and anticlockwise direction of the notation for the left, back and down is different from the ones for the rotations programmed in the game. That is because in the game, all the layers that move along the same axis will rotate to the same direction, and in the case of the notation, it is seen from the point of view of the face, so the anticlockwise direction for the notation of the left is the clockwise direction for the rotation.

Once this is all explained, to make the algorithms in the tutorial it is needed to pass a list of strings in which each string is the algorithm written in the official notation. And then, a conversion into the already coded rotations and directions is needed. Each algorithm of the list will be shown in the tutorial and, when it is completed, the next one continues.

The result is that now it is possible to write any algorithm only with strings with the official notation, and in the case of this project, the list of algorithms used to solve the cube is {"ULB'U'BU'", "R'UB'D'", "R'FLF'L'", "FDF'R'L"}.

With all of this, the work development has been finally explained.

5.2 Planning Control

The planning achieved for this project is the following:

- **Learning:**

  - Task 1 (3 hours): Learn about basic virtual reality programming for a game engine.

- **Interaction:**

  - Task 2 (25 hours): Make the selection of the faces and rotate them with hand controllers.

  - Task 3 (10 hours): Adapt the cube to virtual reality.

  - Task 4 (25 hours): Make the selection of the layers.

  - Task 5 (35 hours): Rotate the selected layer.

  - Task 6 (40 hours): Rotate the cube.

- **Interface:**

    - Task 7 (20 hours): Make the menus.

    - Task 8 (45 hours): Make a tutorial with indications at the top.

    - Task 9 (5 hours): Make the algorithm to solve the initial previously scrambled cube.

- **Control of the game:**

    - Task 10 (2 hours): Create a model of the virtual Rubik's cube in a game engine.

    - Task 11 (25 hours): Create the movements of the cube.

    - Task 12 (5 hours): Create the logic for when the cube is completed or a bad movement is done.

    - Task 13 (10 hours): Complete the remainder of the game.

- **Memory and presentation:**

    Task 14 (40 hours): Final memory.

    Task 15 (10 hours): Final presentation.

The total hours is 300.

5.3 Changes, Difficulties and Solutions

In this section will be explained the changes from the initial planning to the final results.

As the project was achieving the total hours and the main objective of the project was to create a tutorial to solve the Rubik's cube, the part of the AI was discarded and instead, a sample of how an algorithm would solve the cube from an initial scrambled cube. The AI required the study of all the cases of the cube and construct the algorithms in base of each case.

5.4 Results

The final result of the project is a game tutorial to solve the Rubik's cube, in which there is a mode to freely rotate the cube from an initially solved cube; and another mode, that starts from an initially scrambled cube, that has a tutorial that shows step by step how to solve the cube.

Chapter 6

Conclusions

Contents

6.1 Objectives Achieved

Through all this time most of the objectives of the project have been finally achieved, which are the following:

- Analyse how to program a Rubik's cube from zero and create a virtual cube that was able to rotate its faces independently of the position and rotation in the computer.
- Learn how virtual reality works in a game engine and get that cube to work in such an environment.
- Make the menus.
- Create a way to make the controls of the game for some controllers with very few buttons.
- Analyse how to create a tutorial for the cube, based on the book "El Cubo Mágico" [1] and implement it.
- Make an algorithm that solves the cube given the initial scrambled cube.

6.2 Future work

An initial version of the main idea is already done, that is a tutorial to solve the Rubik's cube in a virtual reality environment, controlled by the controllers, and given an initial scrambled state. But due to the short period of time allowed, an AI to be able to make algorithms for any initial state of the cube was not possible to do, so in the future it is planned to complete it. Moreover, it is scheduled to adapt the project to a hand-tracking virtual reality to track the user's hands and, if it is possible, also a real Rubik's cube, so the tutorial could be used to solve a real cube, and not only a virtual one.

In the end, with the advances in technology, a more advanced variant is interesting, in which the virtual reality glasses also scans the colours of the faces and shows a tutorial to solve it, just like some robots do nowadays.

Chapter 7

Bibliography

[1] El Cubo Mágico, Tom Werneck (1981), Ediciones DS. ISBN: 84-7464-111-X

[2] https://unity.com/es

[3] https://www.blender.org/

[4] https://visualstudio.microsoft.com/es/vs/older-downloads/

[5] https://inkscape.org/es/

[6] http://www.gimp.org.es/

[7] https://pixabay.com/es/music/synthwave-synthwave-vintage-future-synth-80s-retro-game-futuristic-music-16535/

[8] https://pixabay.com/es/music/ambiente-cosmic-glow-6703/

[9] https://staruml.io/

[10] https://www.youtube.com/playlist?list=PLmc6GPFDyfw-LG5NUdrJcUeAU21TrrTWT

[11] https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@1.0/manual/index.html

[12] https://ruwix.com/the-rubiks-cube/notation/advanced/