Videogame design and development degree

Final degree project's Technical Report

Ion - Physics Puzzles and Exploration 3D Videogame

Author: *Héctor Ruiz Marco*
Tutor: *José Martinez Sotoca*

Summary

In this document is presented the whole related work of the project. In it is explained what type of project is as well as what are the phases that composes it. The project consist mainly in a 3D playable demo made with Unity. Its main characteristics are the use of default sphere physics for the puzzles and the use of the terrain and tree modeling tools included in Unity for all the modeling involved in the game world to enhance the exploration part of the game. The game is in first person and is only made for PC.

Keywords: *Unity, Videogame, 3D, First Person, Puzzle, Physics, Exploration*

Índex

Figures Index

Introduction

Nowadays many video game developers focus too much on wanting to achieve visual realism. In addition, they tends to complicate the game mechanics and overload users causing frustration and boredom, even end up leaving the game. The main objective of this project is to try to solve both problems. On the one hand you want to demonstrate that you do not need to have realistic graphics for a video game to look good. On the other hand is proposed a series of mechanics that are simple and entertaining, as well as, easy and fast to understand and have a curve of difficulty and learning well balanced.

Therefore, to achieve this, a demo of a videogame in Unity will be developed, which on the one hand will contain an explorable scenario that is pleasing to the eye and on the other hand a series of puzzles and small challenges based on physics of spheres, which will require the ability of the player to be solved during the game.

The graphic style will be very colorful, without practically any textures and with flat colors. In addition, small details like particles and sound effects will be added to increase the immersive effect.

Since the complete game design includes a wide variety of scenarios and puzzles, only a small demo will be developed that will try to show the greatest possible diversity of scenarios and puzzles without having to be part of the hypothetical final version of the game.

The objective of the player will be, to roll a series of orbs to a specific point of the stage through the interaction with the environment and the resolution of puzzles. The orbs can only be moved on a series of rails and other elements that are on the stage, like catapults. The demo will focus only on one of these orbs and their corresponding scenario.

Project Specifications

Story

We put ourselves in the shoes of Kodo, a man in charge of the protection and control of the city's lighthouses. The city is located on a central island, surrounded by six lighthouses. These lighthouses formerly served to protect the city, but wartime passed and that is why they are deactivated. However, Kodo receives a letter from the king in which he tells him that it is time to turn on the lighthouses again as a new danger approaches. To do this, Kodo should look for the six Ion orbs, which are the power source of each of the lighthouses. These orbs are each one located on their corresponding island, contained in the Gyro temples, which are responsible for keeping the orbs stabilized and charged all the time, because due to the large amount of energy these orbs possess, they need to be far of the lighthouses and the city and can not come into contact with any living being. The goal of Kodo is to find each of the Gyro temples to gather the orbs and bring them to their corresponding lighthouse. Since the orbs can not come into contact with any living being, the islands have mounted rail systems whereby the orbs can roll until they reach their lighthouse.

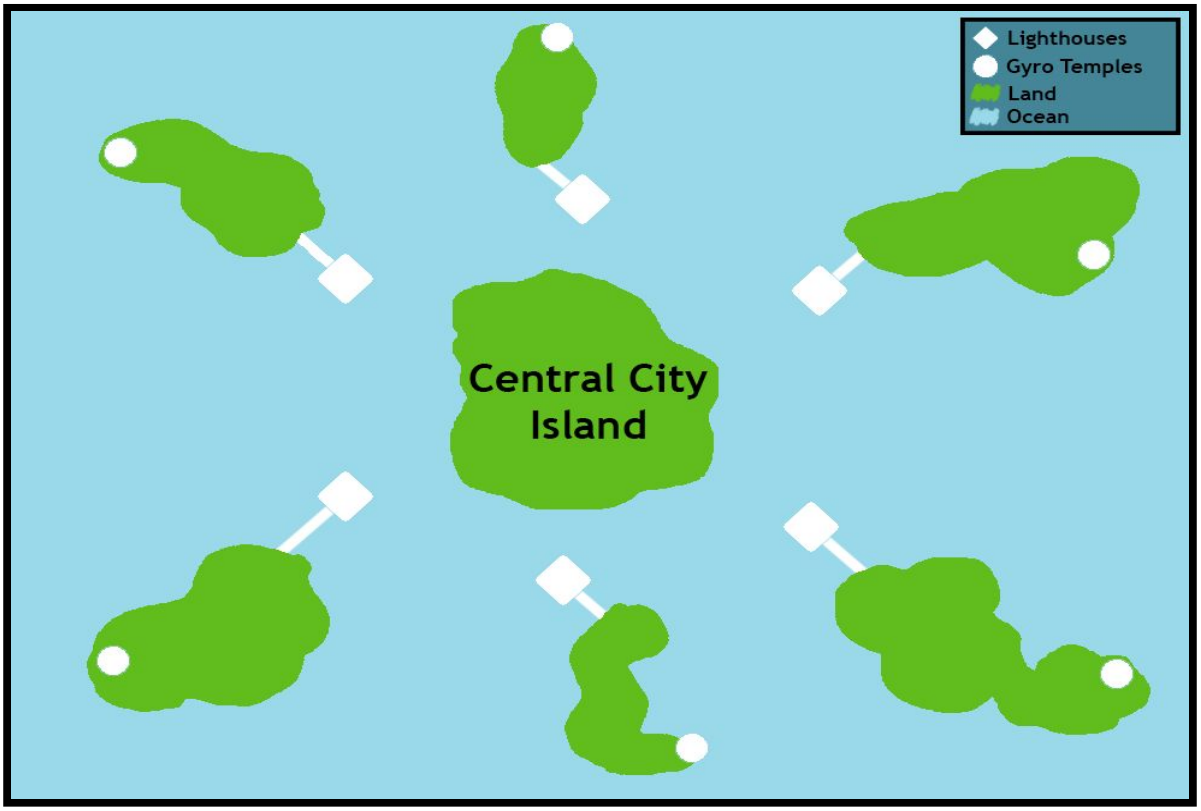

Figure 1: Example of the World Map

Mechanics

The complete designed game is divided into six islands, which can be explored and completed in the order you want by navigating between them freely. However, the demo developed in this project only focuses on one of these islands. Even so, the navigation system will be implemented anyway, to show it as part of the project.

The game flow on each island can be separated into two parts. On the one hand the exploration part, which corresponds to the route that goes from the lighthouse to the temple and on the other hand the part of puzzles, which corresponds to the way back that goes from the temple to the lighthouse.

The player will control Kodo in first person and must explore the six islands looking for the orbs. On each island he will find three keys, which will allow he to deactivate the Gyro temple of that island. Once deactivated, the orb is released and begins its journey along the rails. This route, as a security measure contains some stopping points in which the player must intervene solving puzzles, so that the orb can continue rolling.

The mechanics of these puzzles are based on the so-called Plexus or Superplexus [2], which consist of large transparent plastic spheres containing a three-dimensional mazes inside which runs a metallic ball or a marble. These are controlled by rotating the container sphere and the metal rings that hold it, as a gyroscope. The goal is to make the complete journey from start to finish.

Figure 2: Photo of a Superplexus

3D Design

The visual style presented has is clean and colorful with high saturation, with flat colors without barely any textures. The modeling is low-medium poly without becoming very detailed and recharged. As visual references have been used the videogames The Witness [3] and Rime [4]:

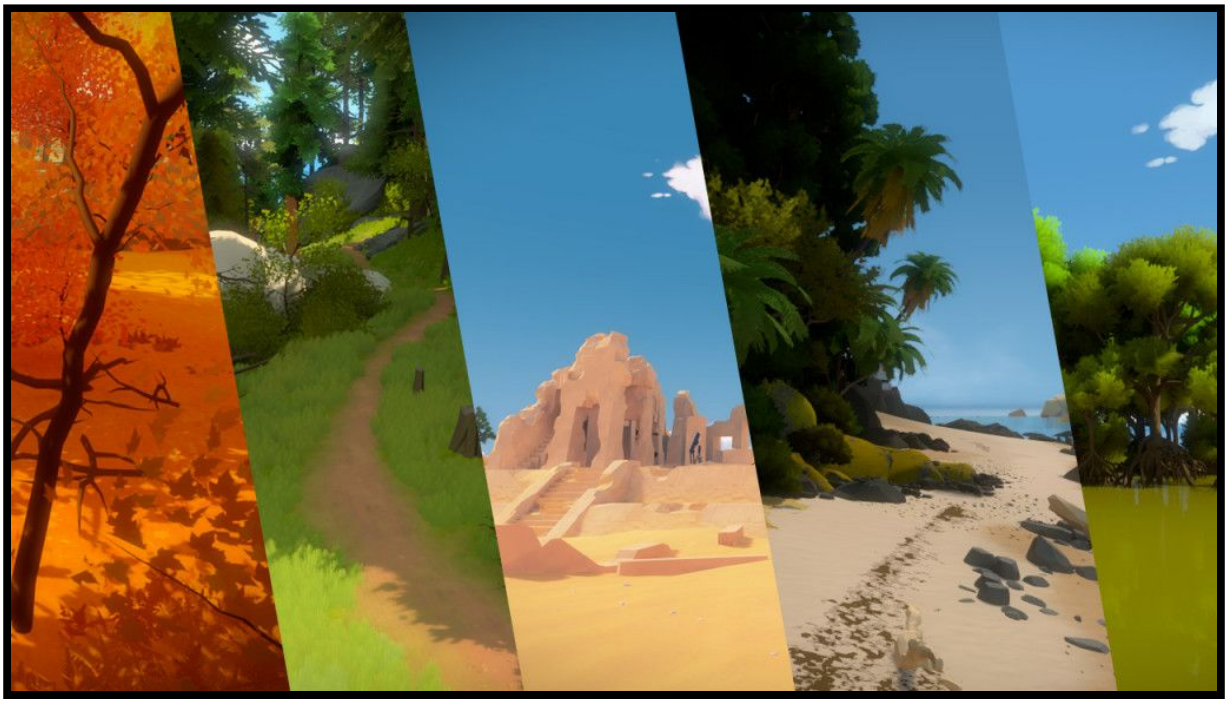

Figure 3: The Witness graphic style

Figure 4: Rime graphic style

Software

For the project development the next tools have been used:

· Unity 5.6.1f1: As the engine for the whole videogame, as well as other tools included in Unity:

  · Terrain tool: This tool offers the possibility to create terrain in the editor. In addition, it also allows you to paint with different textures and to place all type of grass and small plants.

  · Tree tool: For modeling trees and bushes in the editor. This is faster than using an external modeling tool.

  · Monodevelop: As the programming tool, in C# language.

· Photoshop CS6: As the 2D art tool for drawing elements like the interface, the skybox, etc.

· 3dsMax: As the 3D modeling tool for all the 3D elements including structures like arches and natural objects like rocks.

· MP3 Cutter Online: To cut sound files, for making them loopable or to remove empty spaces on the sounds.

· Google Docs: To write all the documents related to the project. This are the Technical Proposal and the final degree project's Technical Report.

Motivation

At the beginning of last year (2016) a video game called The Witness was published. This masterpiece contains everything I've always wanted in a video game and everything that the majority lack. The visual style of the scenes is very clean and colorful. The sound environment is very clear. The mechanics are based on a simple concept that is gradually adding elements, which allows the player to gradually learn the rules of that world. The curves of difficulty and learning are very well balanced. The scenario can be freely explored and completed in any desired order. All these elements make this video game a masterpiece.

It was this game that motivated me to make a kind of "The Witness 2", a game that contained all these wonderful aspects, but from my point of view, with my rules of game and with my own ideas.

In conjunction with this, at the end of the same year I was working on a video game called Hexascope, which consists of solving puzzles using the spheres physics as main mechanics. I really liked the result and I wanted to transfer part of what I learned and the game concept to this project.

Therefore this project consists of the combination of both ideas in the same video game.

On the other hand, this project is the second 3D video game I'm going to develop and I want to see what I'm capable of and experience the most with the options and tools offered by Unity 5 as a game engine. This includes optimization forms, the terrain modeling tool and the tree creation tool, among others.

Objectives

· **Obj 1:** Make a playable demo of a 3D videogame for PC in first person that will use the spheres physics as main mechanic and that will contain an free explorable scenario.

· **Obj 2:** The demo will focus exclusively on one of the six proposed islands, but trying to show a variety of environments in it, including the navigation system.

· **Obj 3:** The scenarios of the game will be pleasing to the eye using a colorful aesthetic without textures and the modeling will be low-medium poly without excessive detail.

· **Obj 4:** The terrain, grass and tree tools included in Unity will be used to create the scenario.

· **Obj 5:** It will be important that the game has a good sound environment.

· **Obj 6:** The interface will be as clean and clear as possible.

· **Obj 7:** The sound and interface should transmit good feedback to the player.

· **Obj 8:** The difficulty and learning curves will be well balanced, allowing the player to learn and understand the mechanics little by little without any kind of tutorial.

Justification

As I mentioned before, many video game developers now choose to focus on realistic graphics and on complicated or repetitive mechanics. In the videogame industry such games abound. However, I think that videogames are a way of abstracting us from the world around us and introducing us to a completely new world that is unknown to us at first and that we learn its characteristics little by little. If you use realistic graphics that are reminiscent of the real world, you already know what surrounds you and all that immersive experience is diluted. The same applies for the mechanics. If they are elaborated and complicated or very monotonous they end up boring the player and he ends up stop playing.

In this project I want to avoid this kind of thing and I want to offer the player an immersive experience in which appear situations that encourage him to continue playing and exploring the world around him and to continue learning about everything that the game proposes to him.

Initial Planning

The project is divided into different tasks established as follows with their time estimations:

T1. Write the technical proposal **[5h]**

T2. Tests on Unity of lighting, shaders, optimization, terrains and others. **[24h]**

T3. Design: It includes all aspects of the demo that will be performed, such as mechanics, scenarios, puzzles, models, interface, game flow, difficulty and learning curves, sound and visual feedback, story, etc. **[90h]**

T4. Modeling: It includes all the elements of the scene realized in 3dsmax, like the lighthouse, the Gyro temple, the rails, the catapults, the rocks, the puzzles, the keys, etc. **[50h]**

T5. Programming: It includes the movement of the First Person Controller and the camera, interaction with the stage, puzzles, normal sound and sound of physics, control of the boat, catapults and many other details. **[70h]**

T6. Look for resources like sounds, textures and assets. **[16h]**

T7. Montage: It includes all the assembly tasks of the scenario in the Unity editor that are terrain modeling, tree modeling, placing of textures and grass, animations, modeling of bridges and placement on the stage of all the elements such as puzzles, rails, temples, lighthouse, catapults and rocks, among others

T8. Write the Technical Report **[45h]**

T9. Prepare the final presentation **[10h]**

The following table shows in more detail all the tasks performed daily according to the initial planning that was decided. In this planning that goes from Tuesday to Saturday, every day is dedicated 4 hours to the indicated tasks, making a total of 300 hours:

SEMANAS	MARTES	MIÉRCOLES	JUEVES	VIERNES	SÁBADOS
13, febrero / 19, febrero	Pruebas de terrenos, cesped, arboles, etc	Pruebas de shaders y materiales	Pruebas de iluminación y efectos	Buscar Texturas, Sonidos y Música	Crear materiales. Pruebas partículas fuego
20, febrero / 26, febrero	Añadir FPC. Crear Orbes	Script PhysicalSound	Script Activator y PositionLerper	Script Interactuator + Cursor Sprites	Añadir Fog. Modelar escenario de pruebas
27, febrero / 5, marzo	Diseñar, modelar y configurar Gyro, y Estatua	Diseñar, modelar y configurar Puerta + Pared	Buscar shader Agua Añadir Océano al escenario	Diseñar, dibujar y añadir Skybox.	Testing Semanal Redactar parte de la memoria final
6, marzo / 12, marzo	Añadir Menús (versión básica)	Soporte de sonido para Activator y PositionLerper	Reconfigurar FPC (FirstPersonControlle r)	Script WalkableAreas	Testing Semanal Redactar parte de la memoria final
13, marzo / 19, marzo	Diseñar el faro (parte 1)	Diseñar el faro (parte 2)	Modelar el faro (parte 1)	Modelar el faro (parte 2)	Testing Semanal Redactar parte de la memoria final
20, marzo / 26, marzo	Añadir partículas y efectos (Shafts, Ambient Occlusion...)	Transiciones entre escenarios.	Pruebas de Optimización (Occlusion Culling)	Buscar fuente de texto y adaptar la interfaz	Testing Semanal Redactar parte de la memoria final
26, marzo / 2, abril	Diseñar habitación inicial del juego	Modelar habitación inicial (parte 1)	Modelar habitación inicial (parte 2) + Materiales	Modelar habitación inicial (parte 3) + Materiales	Testing Semanal Redactar parte de la memoria final
3, abril / 9, abril	Diseñar escenario (parte 1)	Diseñar escenario (parte 2)	Diseñar escenario (parte 3)	Diseñar escenario (parte 4)	Testing Semanal Redactar parte de la memoria final
10, abril / 16, abril	Diseñar puzzles (parte 1)	Diseñar puzzles (parte 2)	Crear terreno base del escenario (parte 1)	Pintar terreno del escenario	Testing Semanal Redactar parte de la memoria final
17, abril / 23, abril	Diseñar puzzles (parte 3)	Sonido de pasos del FPC según el suelo	Programar puzzles (parte 1)	Sonidos puzzles e interacciones	Testing Semanal Redactar parte de la memoria final
24, abril / 30, abril	Crear terreno base del escenario (parte 2)	Dibujar césped Poner césped en el terreno	Modelar árboles y rocas	Poner árboles y rocas en el escenario	Testing Semanal Redactar parte de la memoria final
1, mayo / 7, mayo	Diseñar puzzles (parte 4)	Programar puzzles (parte 2)	Diseñar Menús, interfaz y título	Dibujar el arte de la interfaz de los menús y el título del juego	Testing Semanal Redactar parte de la memoria final
8, mayo / 14, mayo	Programar puzzles (parte 3)	Programar puzzles (parte 4)	Buscar SFX	Añadir todos los sonido al juego	Testing Semanal Redactar parte de la memoria final
15, mayo / 21, mayo	Diseño y programación barca	Diseño y programación catapultas	Bordear con piedras los acantilados de las islas	Pruebas en diferentes ordenadores y pantallas (resolución)	Testing Semanal Redactar parte de la memoria final
22, mayo / 28, mayo	Añadir pequeños detalles	Testing de la demo final	Optimizar el juego (Occlusion Culling)	Testing Semanal Redactar parte de la memoria final	Testing del proyecto final con otras personas

In the section of Final Planning will explain what changes have been and what has been the actual planning that has been carried out.

Design

The design is a very important part involved in any development of a video game, since it includes from the artistic and sound section, to the mechanics and rules of game passing through history and architecture among others. In this project it has been necessary to design all this and much more.

One of the first things to do in order to start designing is to look for references and create a factory of ideas. For this, all types of images and folders have been compiled with references that include scenarios, artistic styles, different architectures, natural elements, concrete objects, etc. For this project, almost 2000 reference images have been collected. Throughout this section will be showing some of them that are more relevant.

On the other hand, once you have all the references it is time to start making different sketches of all the aspects involved in the project. In this case, sketches of the stage, the lighthouse, the temple, the puzzles and everything that is related to the initial idea.

Visual Style

At the beginning of the project, it was unknown what the mechanics of puzzles with spheres physics would consist of. The only thing that was known is that is wanted to create a large stage, pleasing to the eye and freely explorable with a camera in first person. For this, the first of all is the visual style, which as previously mentioned, is based on games like The Witness and Rime, which give much importance to the color and high saturation of the same, achieving very alive landscapes. They use flat colors or very subtle textures that allow to highlight the importance of shapes and reliefs with the use of lights and shadows. In addition they use modeling without much detail almost low-medium poly. In order to apply this visual style you must first study it. This is discussed in detail below.

Figure 5: Rocks' style comparison

The first thing to analyze are the textures. If you compare the rocks of Figure 5 you can see that the ones on the left have a texture that is used to highlight the shape of the rock while in the one on the right the texture is merely decorative and the shape of the rock Is highlighted by light and shadows. In this project is wanted to achieve the effect shown on the right, that is, the lighting is the one who handle the shaping of the elements on the stage. To help highlight the relief, the *Ambient Occlusion* effect will also be used.

Figure 6: Color contagion in The Witness

Figure 6 shows another of the effects to be achieved. It is the color contagion. On the ground you can see how it reflects slightly the purple color of the plants on the right or on the ceiling, on the left, you can see that it starts to take on a greenish tonality due to the vegetation that is on the other side of the wall. To achieve this effect is used what is called *Final Gather*. Through a series of bounces and calculations by the scene, this effect is painting and accumulating color where it bounces. To be able to apply this effect it will be necessary that the illumination of the scene is baked, that is to say, that it is not calculated in real time but that is applied with precalculated textures. This effect can be seen better in Figure 7.

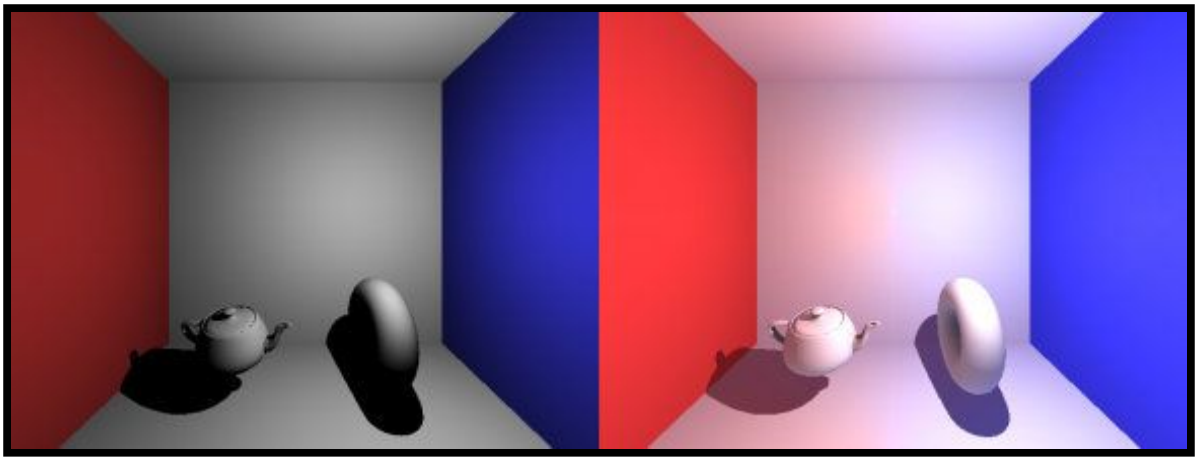

Figure 7: Without Final Gather (left) and with Final Gather (right) comparison

Another important aspect of this visual style is the vegetation. In The Witness, both the foliage of the trees, and the flowers and the grass, seem to have a glowing appearance almost as if they were emitting light. To achieve this effect it will be necessary to use the Emission property in the materials used for the leaves of the trees. This effect can be seen in Figure 8.

Figure 8: Emissive foliage effect in The Witness

As for the grass, because it will be placed using the tool that comes integrated in the tool of creation of land of Unity, it will be complicated to adapt it and to modify it so that it resembles to the visual style raised. To do this, special attention must be paid to the shape and color of the grass leaves. In the Figure 9 it is possible to appreciate better the style of the grass that is wanted to try to achieve.

Figure 9: Grass style from Zelda: Breath of the Wild

Mechanics

As for the mechanics, they started based on the so-called Marble Machines [1], which are small circuits by which a series of marbles roll. The idea was that the scenario of the game would be one of these circuits and that the player in first person could interact with different elements and mechanisms to help the orb to continue advancing correctly by the circuit until arriving at the lighthouse. However the mechanisms of the Marble Machines are very advanced and require many hours of design and modeling and there was no time for all the rest of the things they wanted to do, so is chosen to simplify the idea. It kept the idea of the circuit that carried the orb from its initial point to the lighthouse, but they were only a series of rails by which the ball fell and roll. During this tour, the orb encounters different obstacles and the player must solve them so that the orb continues to advance. These obstacles require the resolution of puzzles, which are puzzles based on spheres physics. As explained above, these puzzles are based, on the one hand, on a previous project called Hexascope and on the other hand, in the so-called Superplexus [2]. It is a combination of both concepts and the result is explained below.

Puzzles

Along the stage you will find a kind of pedestals or stone columns that have a spherical element of transparent glass in the upper part. This glass sphere is the one that works as a superplexus and the player can rotate it on both its x-axis and its y-axis. Inside the sphere there are one or more micro orbs that can be blue or red. In turn, the sphere has one or more ring shaped exits, which may also be red or blue. The initial concept is that the player must rotate the sphere to make the orbs fall through the exits of their same color. A small initial sketch is shown in Figure 10.

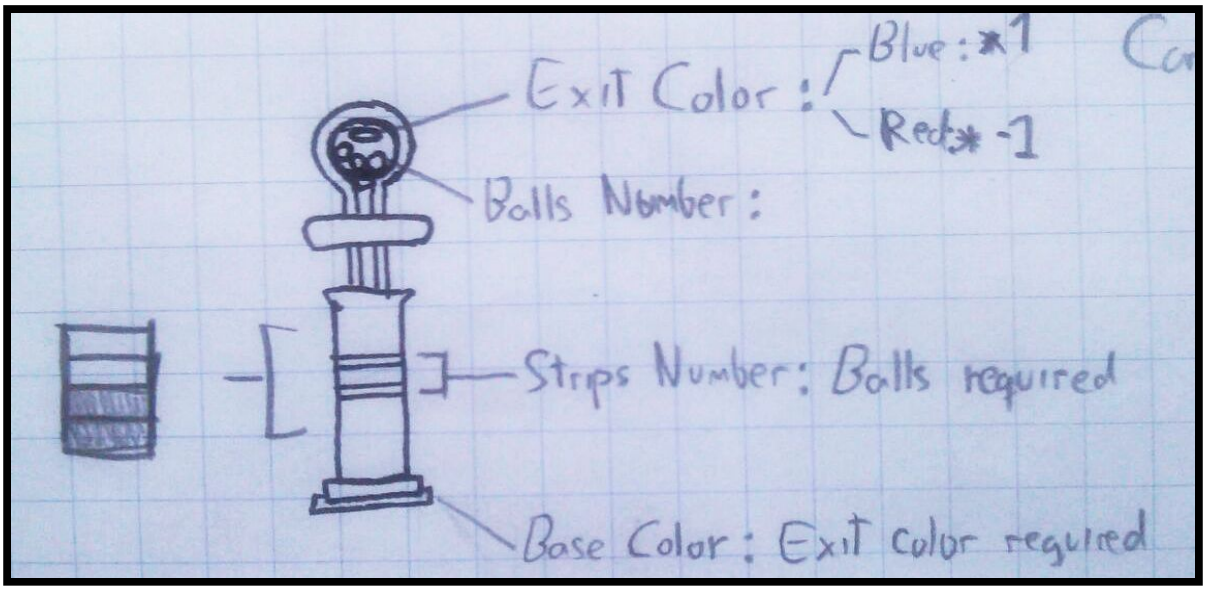

Figure 10: Puzzle sketch

However, the idea is that to this initial basic concept will be added small variants that will gradually appear as you progress in the game. For the demo, only the color variant of the base is added. The pedestal has a ring on the base that normally lights up in blue when the puzzle is available to be solved. However there will come a time when in some puzzles the base will be illuminated in red instead of blue. The blue color indicates that it is a normal puzzle and the red color indicates that it is an inverted puzzle. This means that if the base is red, instead of introduce the orbs through the exit of the same color, you have to enter the orbs through the exit of the opposite color. Therefore in inverted puzzles, blue orbs must be inserted into the red exit and the red orbs into the blue exit. This concept may seem strange or over complicated at first but thanks to the good design of the learning curve, the player will learn this easily without any tutorial. In The Witness is made a very good use of the learning curve, introducing the mechanics of the puzzles little by little and increasing the difficulty progressively. In Figure 11 you can see a sequence of puzzles in this game. Without needing to explain what these puzzles consist of, you can see that each panel, how much more complex it looks from left to right.

Figure 11: Black and white puzzles' sequence in The Witness

In this project is wanted to try to follow this model of learning and difficulty curves. In order to elaborate a learning curve it is necessary to try to follow the thread of thought that the player will carry. For this you have to make use of colors, shapes and even sounds that make you relate things in one way or another. Considering this thread of thinking, you can start to assemble the puzzles. In Figure 12 you can see a sketch that represents the sequence of puzzles that has been designed for the demo following this model of learning.

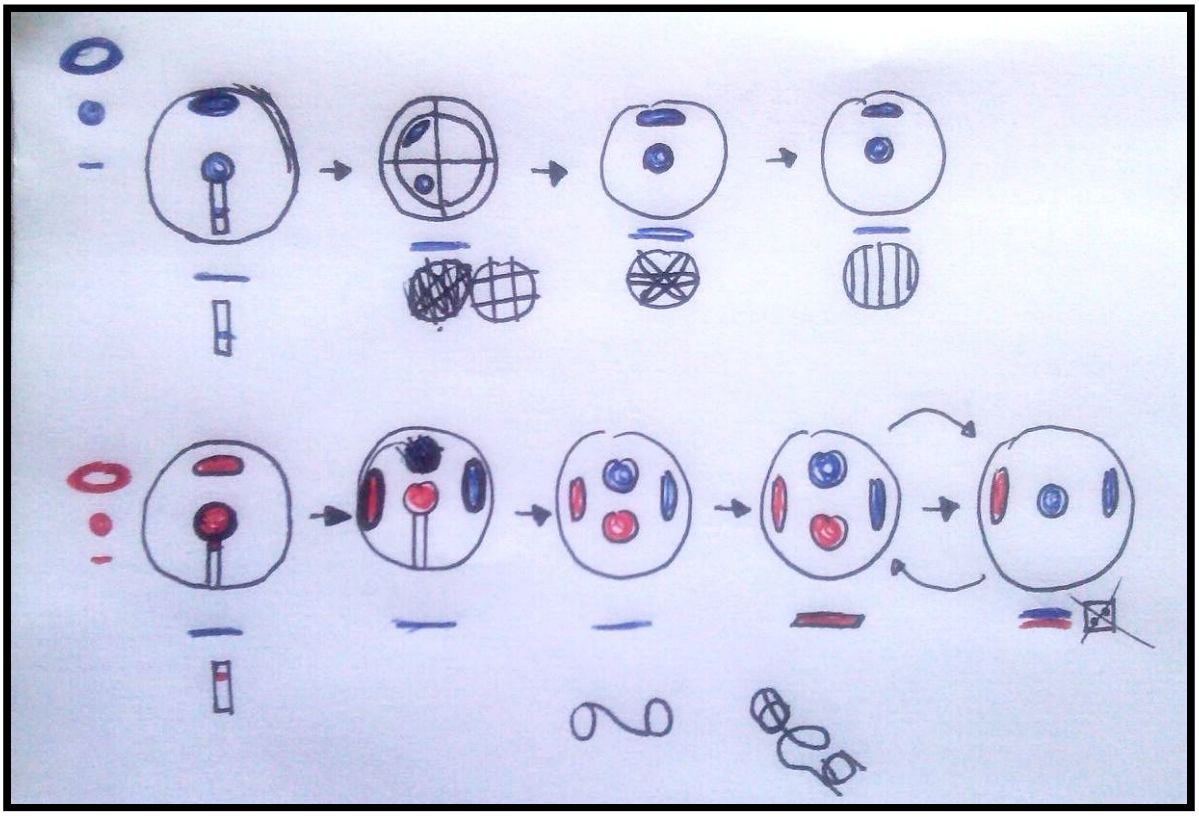

Figure 12: Puzzles' learning and difficulty curves design

The first puzzles are based only on the superplexus itself, without taking into account different colors or different exits, all being blue. In these, the player learns only the control of the sphere and the basic concept of getting the orb to fall through the exit, without paying special attention to the color.

In the following sequence of puzzles the red color and the red exit is introduced, something that will inevitably already catch the attention of the player. Later both colors are mixed and the player will next relate the colors and deduce that each orb must fall through the exit of its same color. If the player incorrectly completes a puzzle, it simply restarts, so if it is not able to deduce it, you can get it by try and fail.

Finally the bases of red color are introduced, that also will catch the attention of the player. Along with this, you will be forced to enter a red orb in a blue exit with a puzzle that only has these two elements. At first the player will surprise him, but will soon relate what has happened to the fact that the base is red. In this way, without having to explicitly explain it through a tutorial, the player has learned and deduced the mechanics of the puzzles, something that is much more satisfactory. Therefore, there is no tutorial as such, but the mechanics are explained on their own thanks to this learning curve.

Feedback

Since the puzzles are separated from each other and distributed on the stage, to be able to know at each moment what is next, where to go and if they are doing things right or wrong, different types of feedback are used.

On the one hand there is the visual feedback. When a puzzle is activated and ready to be solved, it is marked with a vertical beam of light that can be seen from a distance. Along with this, all the puzzles on the stage are interconnected sequentially with a cable. This cable lights up as the puzzles are completed, so following it is easy to know which way to go. Also, when the puzzle completes correctly it lights up the same color as the base.

On the other hand is the sound feedback that is an auditory support to visual feedback. When a puzzle is activated, it plays a 3D sound, which when being in stereo, can be perfectly deduced from where it comes from. On the other hand, when solving a puzzle, a sound of success or failure is reproduced according to whether it has been correctly completed or not. Also, if done wrong, the puzzle simply restarts.

Exploration

As already explained before, the mechanics of the game do not only consist of completing puzzles. An important part is the exploration of the environment. In order to perform this part, it is important to somehow get the player to have some interest in explore the island. A secondary mechanic has been designed for this purpose. This is the search of the three keys.

The first objective of the player as it is known, is to reach the Gyro temple where the orb is located. However getting there is too simple and monotonous. To improve the exploration experience, the player must find first three keys found in

different areas of the island. These keys are placed on pedestals under small domes, so it is easy to see them from far. In addition the player will be able to find a help map that will let you know where each key is located. Once the three keys are obtained, they can be placed in the Gyro temple to unlock the orb that is there. An initial sketch of the three keys is shown in Figure 13.

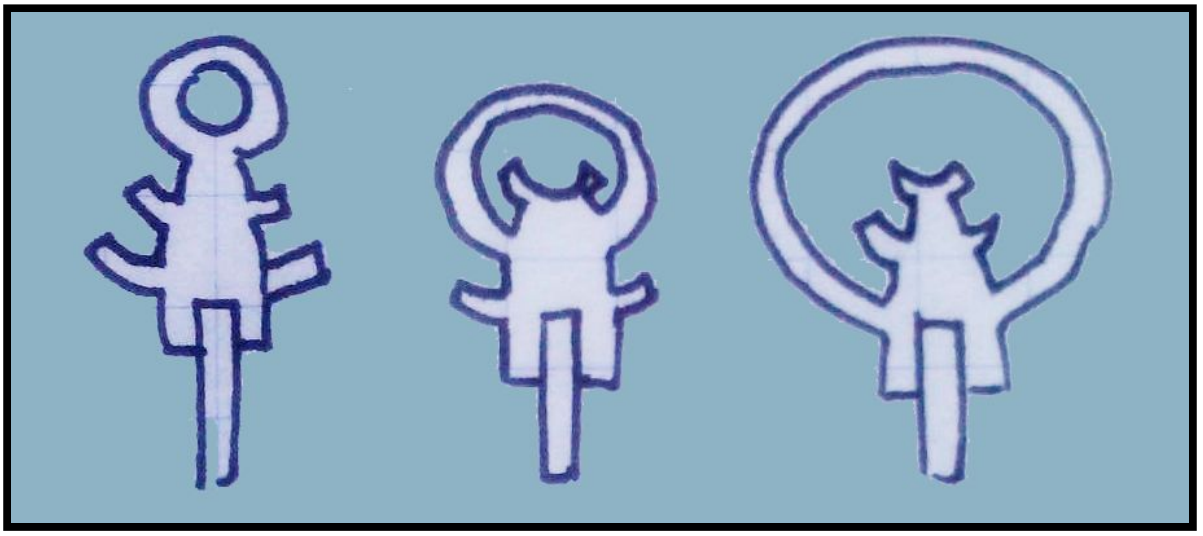

Figure 13: Keys' sketch

When a key is collected, no indicator appears on the interface, so the player must remember which keys has catched and which keys has not. This is intentional because you want to maintain a clean and clear interface to increase the immersion of the player in the environment. However, to help the player remember which keys they have, each one is identified with a color (blue, green and red) and this way you will also know which key goes in each Gyro temple slot, since these also have each one a corresponding color.

Environment

The scenario of the hypothetical final game would consist of 6 islands arranged along a circumference and a central island that would be the city, as shown in the map of Figure 1. Each island contains a lighthouse and a temple with an orb to get. The demo focuses only on one of these 6 islands.

It is chosen that the stage is formed by different islands for various reasons. The first one is that you want to create an environment and a different environment for each island. For example one island can be very leafy while another island can be quite arid. However, since for the demo only an island is going to be realized, different types of environments and plants are proposed in the same so that it becomes more diverse.

Another reason is that is wanted to implement a navigation system so it is necessary that the scenario is composed of different islands and areas interconnected solely by the sea.

And the last reason is the limitation. Making islands is a natural way of limiting the explorable space without the need for invisible walls or anything like that. However, it could be freely navigated by it, so it would still be necessary to limit the map in some way in the hypothetical final game. Anyway, the boat that is proposed for the demo is in a completely closed area so you can not go out to sea. Therefore this will not need to be implemented for the project.

Figure 14: Island reference model

Figure 14 shows one of the visual references that has been used to create the island and in Figure 15 an initial sketch of it can be seen with an aerial view.

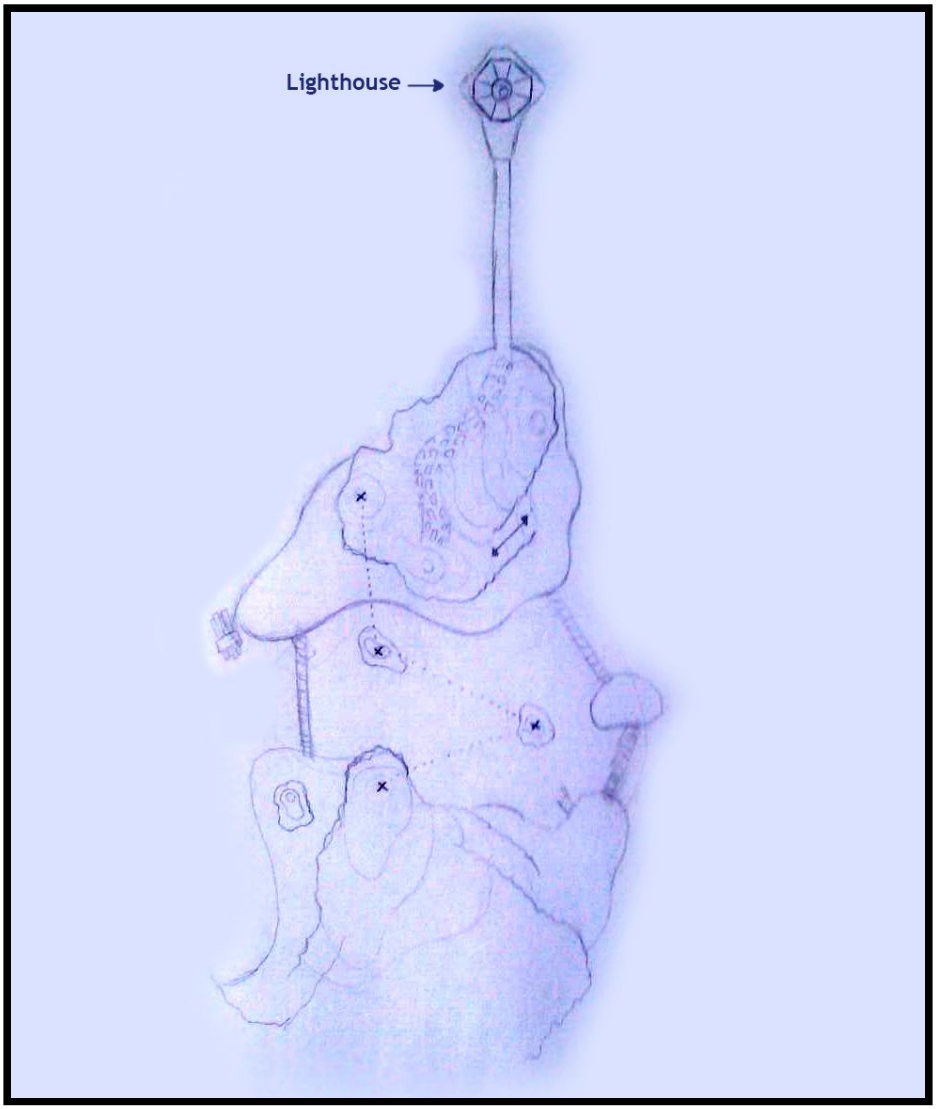

Figure 15: Island map sketch

Architecture

The project has some architectural elements such as Gyro lighthouses and temples. For both types of structure is wanted to use white colors as a base simulating chalk effect, dark gray for small details and blue for roofs. Figure 16 shows a reference image that reflects this palette of colors in real buildings.

Figure 16: Architecture colors reference

As for the type of architecture is wanted to differentiate the temples from the lighthouses. The temples would be rounded structures and of low height whereas the lighthouses would be structures with straight sharp edges and very tall.

In Figure 17 you can see a reference image for the architecture of the temples, together with a proper sketch.

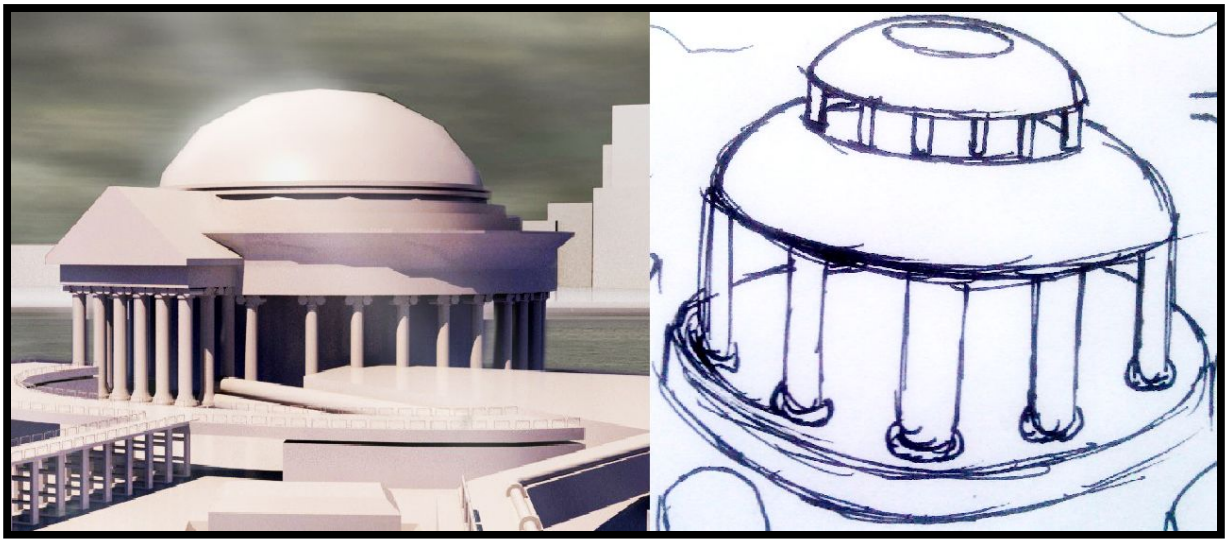

Figure 17: Temple's architecture reference (left) and temple's architecture sketch (right)

As for the lighthouse, it has been based on the oriental stone lanterns, since they are tall structures and with straight sides. In Figure 18 we can see both the reference image and the subsequent sketch that has been made.

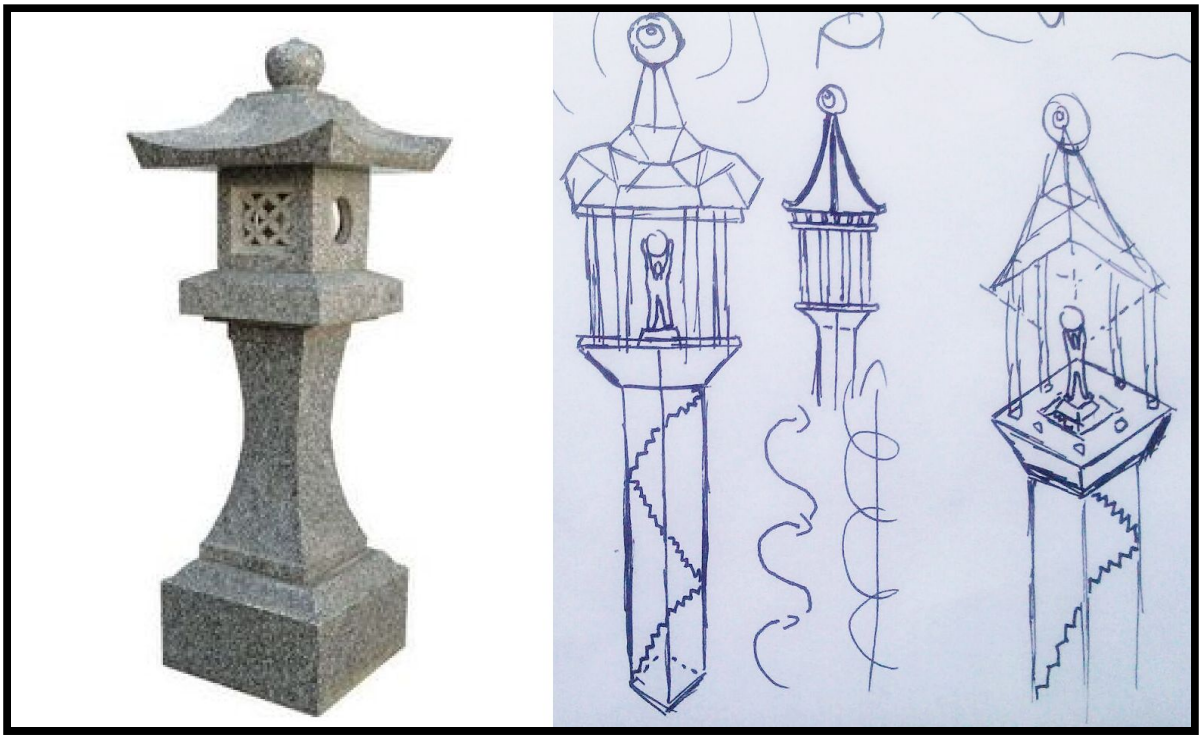

Figure 18: Lighthouse's architecture reference (left) and lighthouse's architecture sketch (right)

Another important structure is the large stone bridge that connects the lighthouse with the island. For this structure it has been decided to rely on the typical Roman stone aqueducts, as can be seen in Figure 19.

Figure 19: Romanic aqueduct

Development

Initial Tests

First of all before starting to develop anything of the game is needed to do all kinds of tests in Unity to see the different possibilities there are and know which ones to use, as well as look for some complementary and alternative assets. The following tests were carried out:

Terrain tool

The first tool to test is the terrain modeling tool. Is wanted to use this tool on one side to accelerate the process of creating the environment and on the other side to use the option of grass placement that comes integrated. The tests consist mainly of seeing how the tool works and how optimized the grass rendering is. Being a tool that is based on the operation of height maps to generate the terrain, you can not model structures such as stone arches or structures that are wider above than below. For each value of X and Z, there can only be one Y value of height. Therefore this is a disadvantage that has this tool, since if you wanted to do something more advanced would need to be molded separately with some external tool such as 3ds Max or Blender.

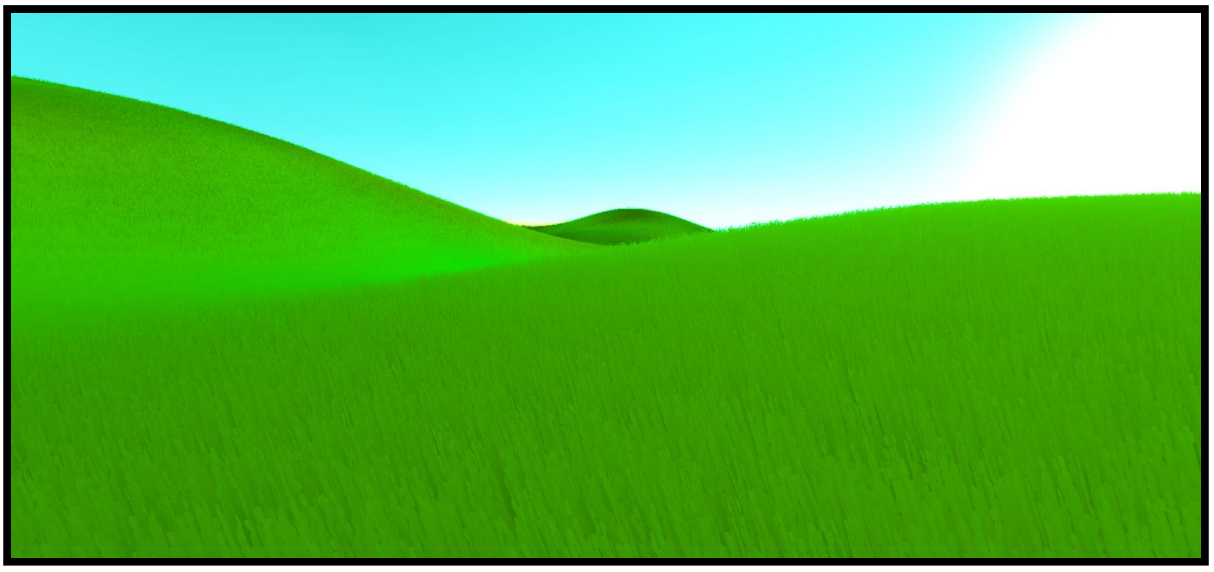

Figure 20: Terrain test full of grass

To test the optimization of the grass, a test map with a simple terrain filled with grass has been created as shown in Figure 20. In it a First Person Controller has been placed, the one that comes by default in the standard assets of Unity. Doing this and running the game in fullscreen, the fps were reduced to about 10, therefore did not seem very optimized. In order to try to improve it, we have studied the different options of this tool [5] and some of them have been modified.

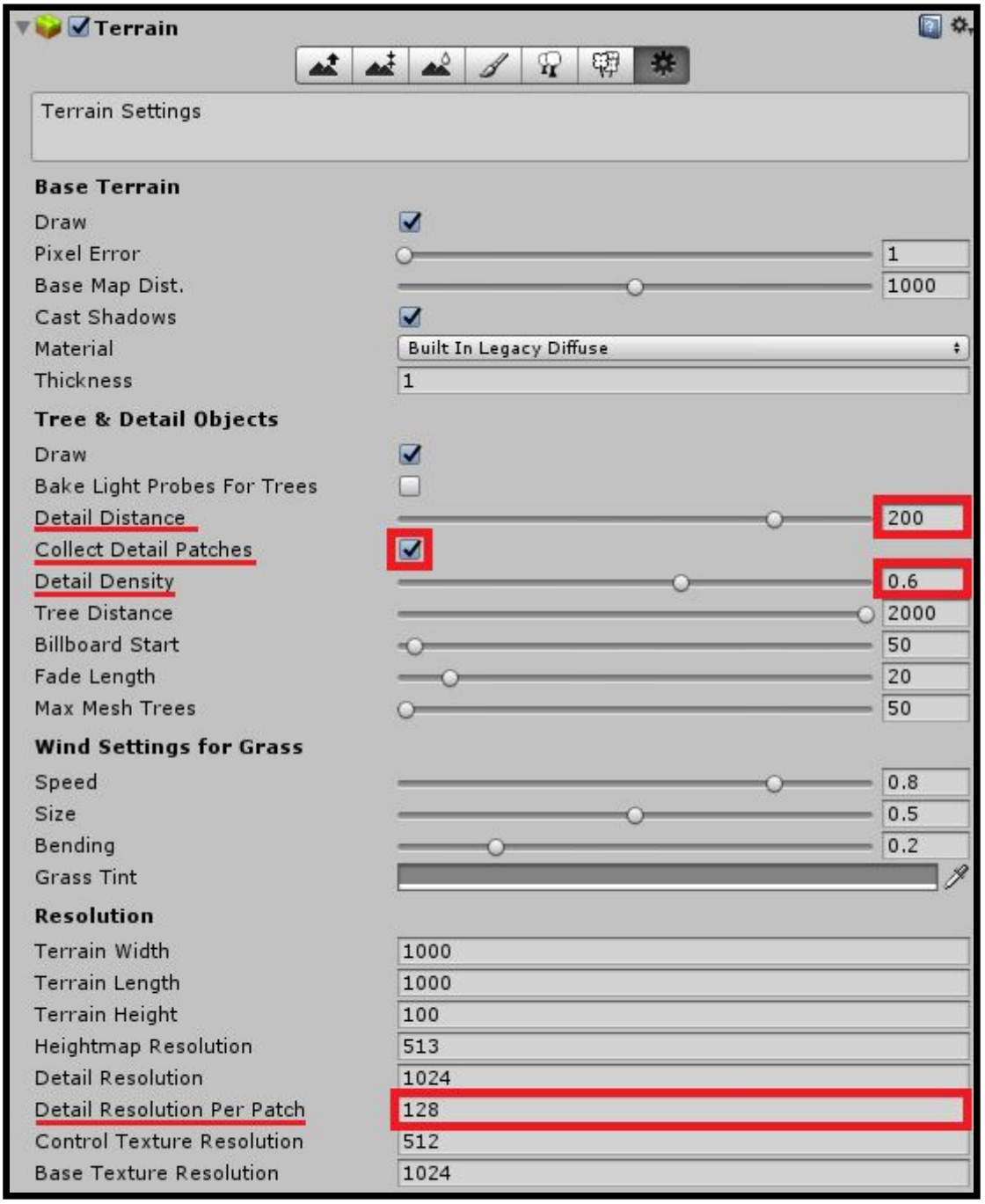

Figure 21: Terrain tool with optimized options

One of these options is *Detail Distance*, which is the maximum distance at which the grass is rendered. By default this is 250 which is the maximum but you can reduce it so that it does not render grass that is far away and that you really can not see. Another option is *Detail Density* that varies between 0 and 1 and the default is 1. This value allows to define the density of grass that is in each unit of area. The lower, the less grass there will be. Finally the most important option that has turned out to be a big difference in optimization is *Detail Resolution Per Patch*, which allows you to define the size of the square area that is rendered in a single draw call. By default this value is 8, which is the minimum. However, putting it to the maximum, which is 128, drastically reduces the number of draw calls needed to render. For this option to take effect, it is important to check the *Collect Detail Patches* checkbox. Thanks to all these changes, especially the last one, if you try again the scene, you get 60fps. The final options that have been chosen are shown in Figure 21.

Tree tool

On the other hand is the tree modeling tool. It has been tried to model some test trees and is quite easy and intuitive although it can be tedious if you want to make trees with many branches. To reduce the workload, this tool will be used, but only simple trees with few branches were modeled. This decision is also partly to maintain the low-medium poly style of the game.

Shaders

As previously mentioned, the visual style is based on games like The Witness and Rime, which give much importance to the color and high saturation of the same and highlight the importance of shapes and reliefs using lights and shades. To try to imitate this type of lighting has chosen to try different shaders of cel shading and toon shading. Toon shading shaders already come by default in the Unity standard assets. As for the shaders of cel shading, they have been obtained freely in different web pages [6]. However, both types of shaders have finally been discarded for not producing the desired effect and in return the standard shader will be used with vibrant and flat colors and with a slight *Emission* effect to increase the saturation. It is important that the *Emission Global Illumination* be marked as *None* so that it does not illuminate the scene, as shown in Figure 22.

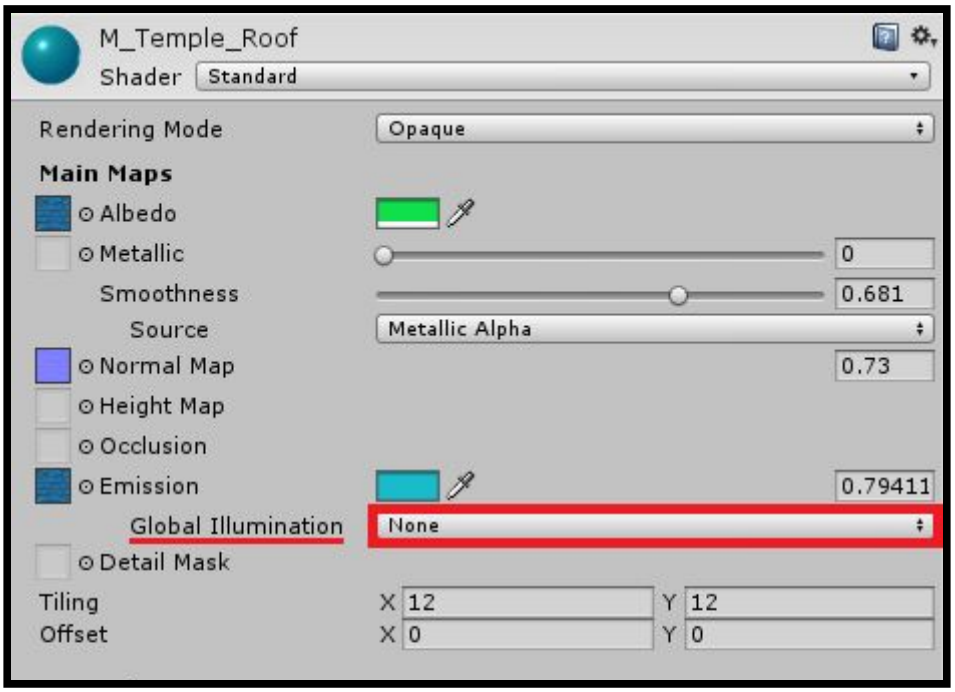

Figure 22: Material example without Global Illumination

Even so, some materials have been kept with the toon shader for elements such as rocks, trunks and some metals, to produce variations of saturation and interesting reflections, which simulate streaks in the rocks or glitterings in some metals. In addition, a own shader has been developed based on cel shading. This shader has been used for orbs and causes objects to have a colored kernel inside them. The shaders will be explained in more detail later.

Lighting

Lighting is an important aspect to take into account especially for the visual style that is wanted to achieve. However real-time lighting consumes too much and reduces fps drastically. This is mainly due to the amount of shadow casters in the scene and the complexity of the models. Therefore bake lighting is used. Different bake testing has been done with different options to know how it works and how to configure it when the final bake is needed to be done.

Camera Effects

In Unity's standard assets a series of effects that can be applied to the camera are included. These effects improve and enhance the visuals, but they consume too much and greatly decrease the fps. Therefore, despite wanting to use many of these effects, in the end has chosen to use only those strictly necessary such as *Antisliasing* and *Ambient Occlusion*. Although the *Ambient Occlusion* will already

be calculated when the bake is done, this *Ambient Occlusion* in real time achieves much more accurate results than the one that gets the bake, so that is why it has been decided to use it. However being in real time, the fps suffer enough especially when there is a lot of geometry.

Assets

In conjunction with the camera effects, it has been wanted to add another series of effects that allow for example to simulate neon effect or height fog effect. These effects have been obtained from the asset store or free form different web pages.

On one side there is the asset *MK Glow Free* [7] that allows to create materials that produce neon or incandescent effect. This requires the use of a camera effect to be able to render these materials correctly, which also causes the fps to drop. This effect is used as a feedback element in the different puzzles and in the wiring of the stage as well as the buttons. Lets you know the player when an item is on or off. On the way back, the player must follow the cables as they light up to know what route should follow, so you could say that the cables also work as a guide.

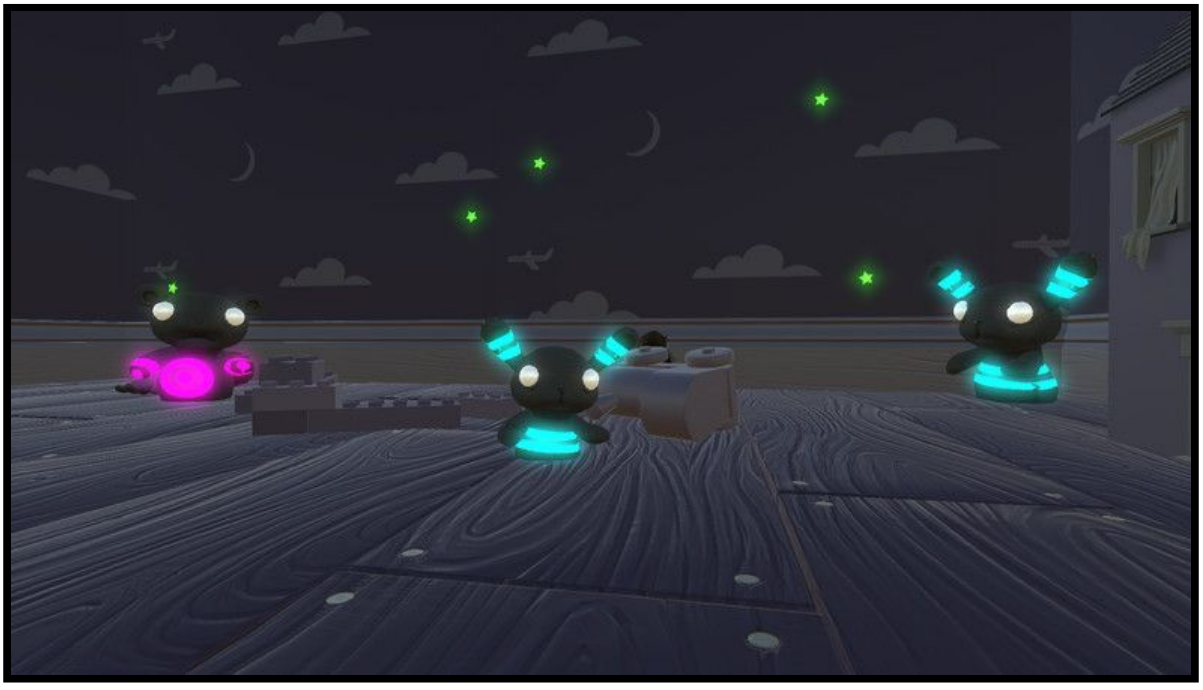

Figure 23: Neon effect in a test map

On the other hand there is the *Light Shafts* asset [8] which allows to create lightbeams as well as to simulate the effect of sun shafts and even of height fog. In the project will be used for all these aspects. On the one hand, like dense fog

inside the lighthouse to give a more immersive effect. On the other hand as a sun ray effect to improve the visuals. And finally as lightbeams like feedback illuminating certain zones and puzzles so that the player knows in each moment where he has to go and which way.

Figure 24: Sun shafts effect in a test map

Finally there is the *Volumetric Lights* asset [9] which allows to do things similar to the previous asset, but in this case, the use that is going to give is the area lighting. This will be applied to *Pointlights* and serves not only to illuminate the geometry with which it impacts, but also to illuminate the entire spherical area. This will be used for practically every spotlight in the scene, such as torches, door panes and even the orb itself. The only purpose of the use of this asset is to increase the immersive effect that is wanted to achieve in this project.

All these effects inevitably decreases the fps and being assets already prepared is not possible to optimize them, or at least not in a relatively simple or comfortable way.

Art Development

This section refers to any aspect of the artistic development that has been necessary done for the project. This includes sound and both 2D and 3D art. Inside 3D art is also included all the scenario modeling done in Unity's editor.

Game World

To make all the game world, the terrain tool from Unity has been used. Thanks to the initial tests carried out, this process has been able to be carried out more quickly. In general the modeling of the terrain has not been difficult but it is true that when trying to model ramps has been more complicated since this tool does not offer any simple way to do this, but you have to do it by pulse, little by little. One of these ramps can be seen in Figure 25.

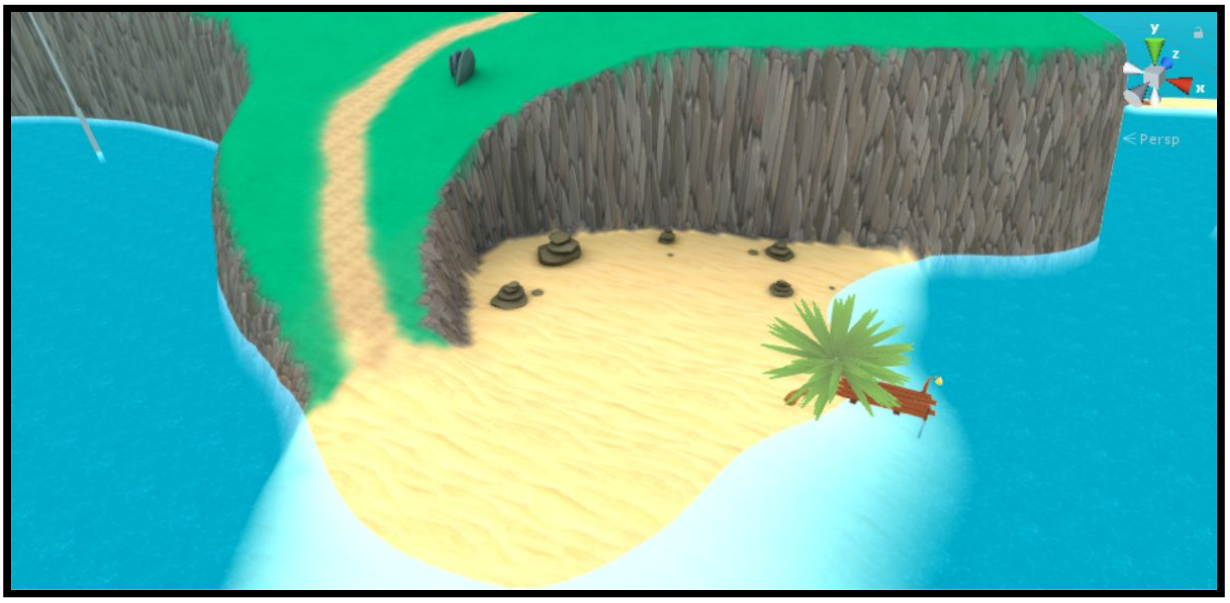

Figure 25: Natural ramp elevation in the terrain in the Unity Editor

Paint Height has been used on one hand to create large terrain shapes and elevations such as islands and beach areas. Thanks to this option you can model terrain at different heights. For example, for beach areas a base height of 22 has been used and for the island zones a base height of 70 has been used. This value can be defined in the Height property as shown in Figure 26.

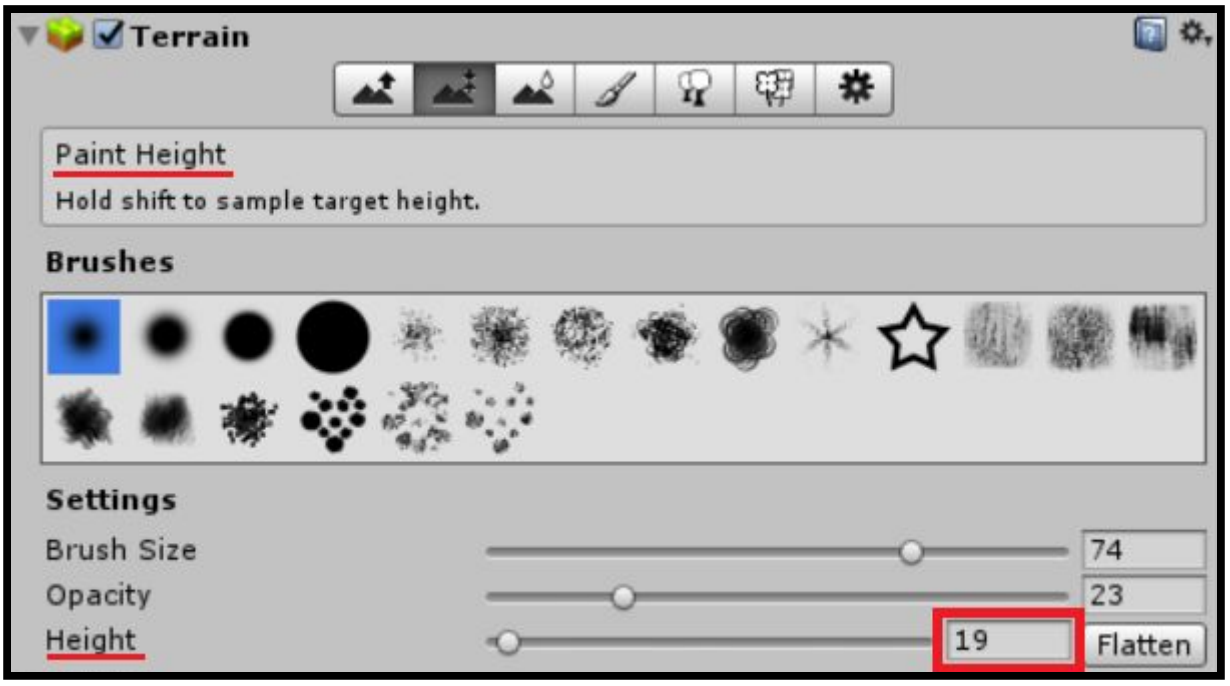

Figure 26: Paint Height tool from Terrain tool

Later the Raise / Lower Terrain option was used to model more in detail the different slopes and mounds of terrain as well as the ramps that join areas of different heights. This tool is used more by pulse, as if it were sculpting in the ground and it is easy to be mistaken so it is necessary to do it carefully.

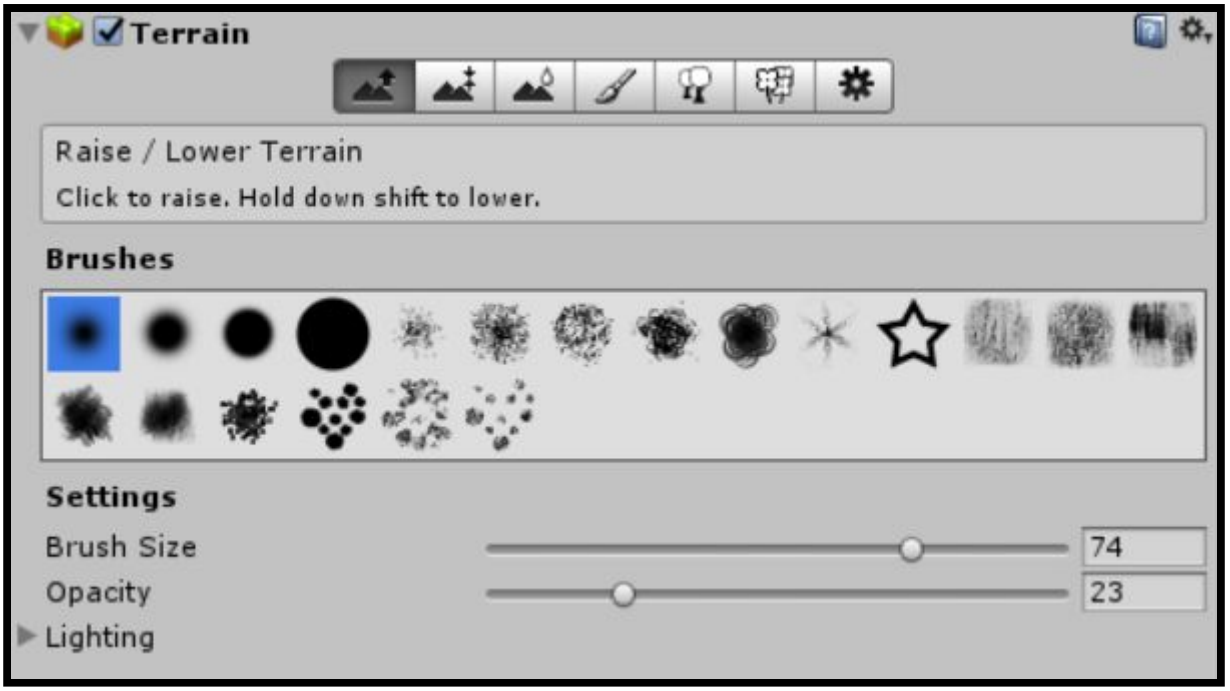

Figure 27: Raise/Lower Terrain tool from Terrain tool

Finally the Smooth Height option was used to smooth and round the terrain and make the transitions between heights more progressive.

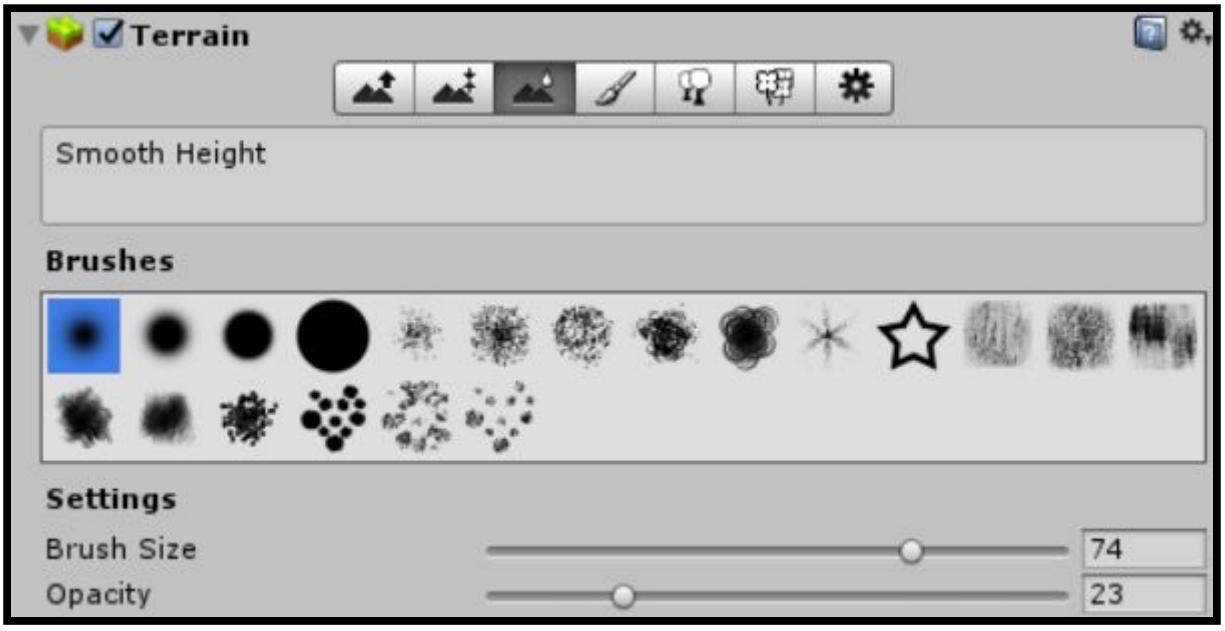

Figure 28: Smooth Height tool from Terrain tool

Once the whole terrain has been modeled, the Paint Texture option has been used to paint with different textures previously elaborated. You can see these textures and this option in Figure 29.

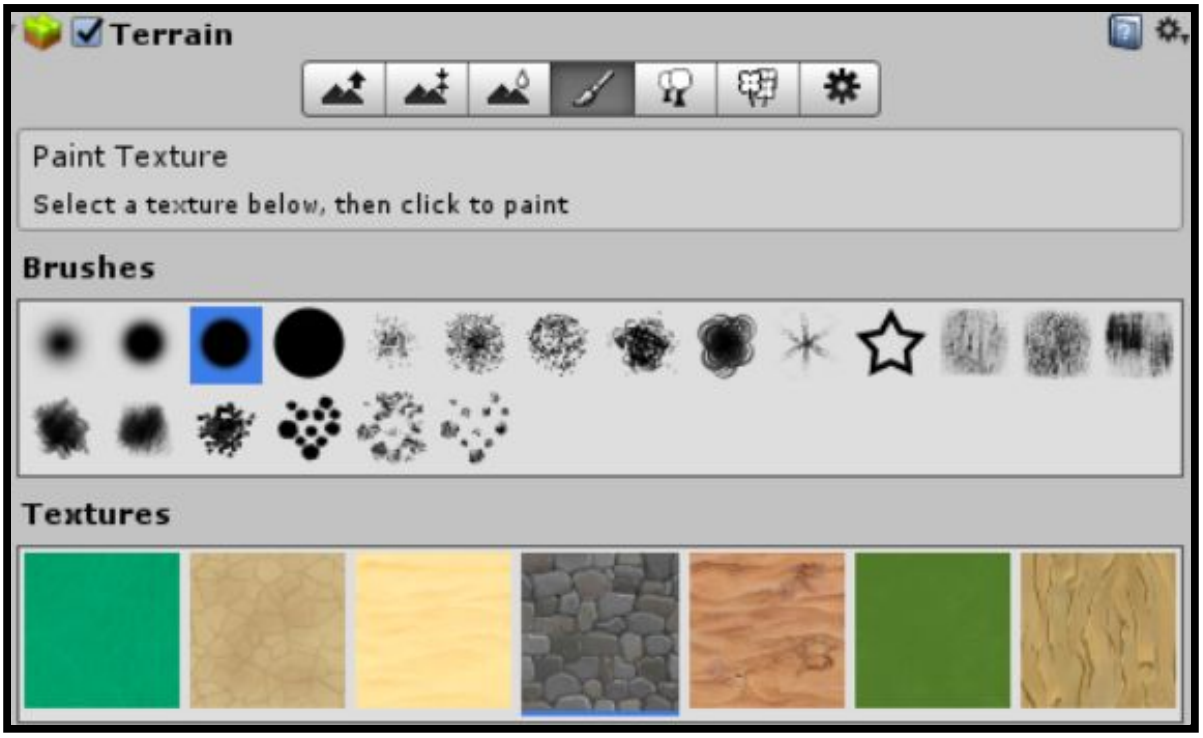

Figure 29: Paint Texture tool from Terrain tool

Modeling

To model all the elements and structures necessary for the project, the 3dsmax tool has been used. It is not necessary to go into detail in the use of this tool, in general it has been simple to use. As a curiosity, for the modeling of the lighthouse and the temple has also been necessary to model two types of colliders, one for the floors and another for the walls. These colliders are not rendered, they are simply used to delimit the areas where the First Person can move. In Figure 30 you can see the model of the lighthouse, along with the collider of the floor and the collider of the walls.

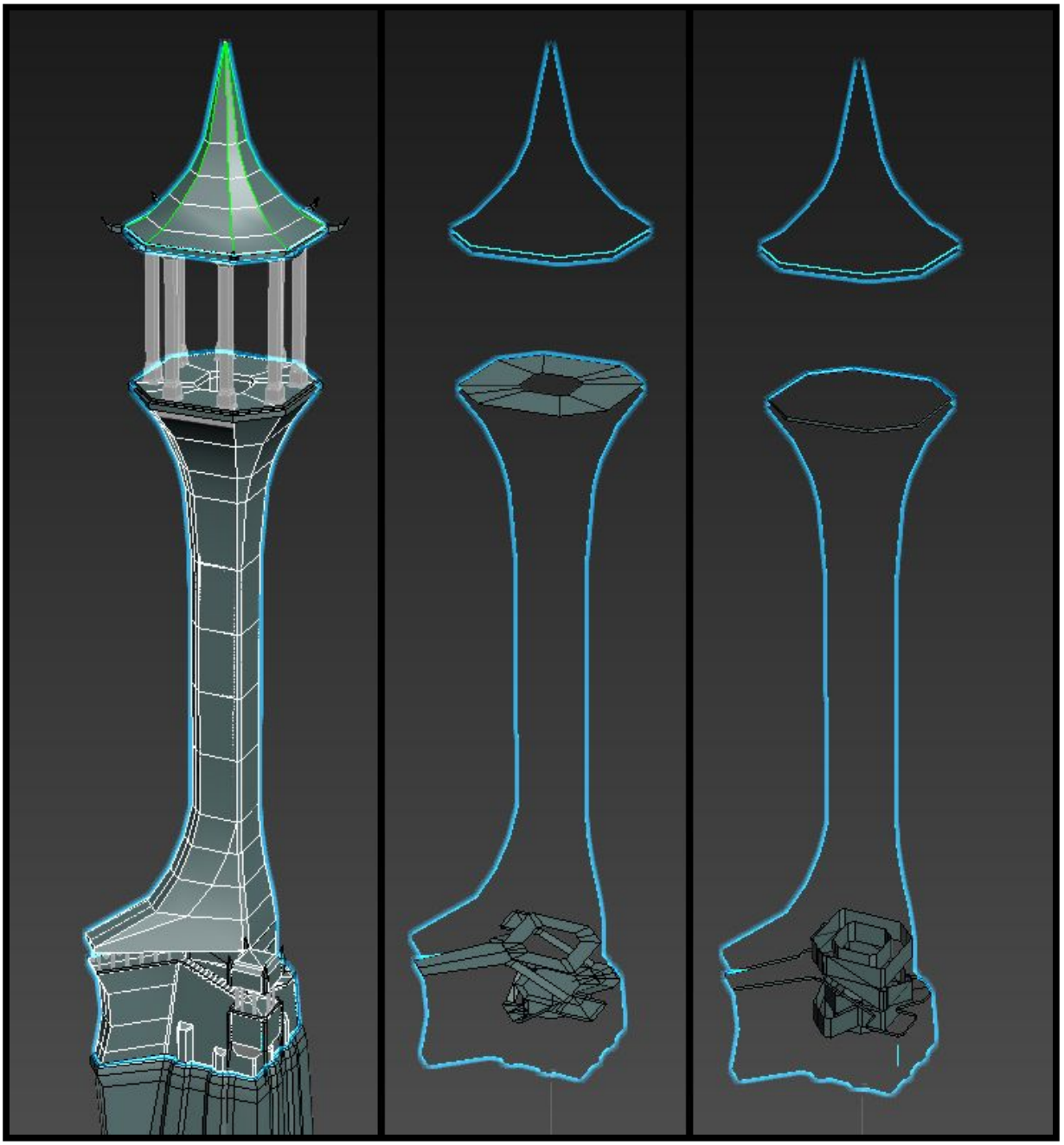

Figure 30: Lighthouse model (left), floor model (middle) and wall model (right) in 3dsmax

Figure 31 shows some more models that have been made in 3dsmax for this project.

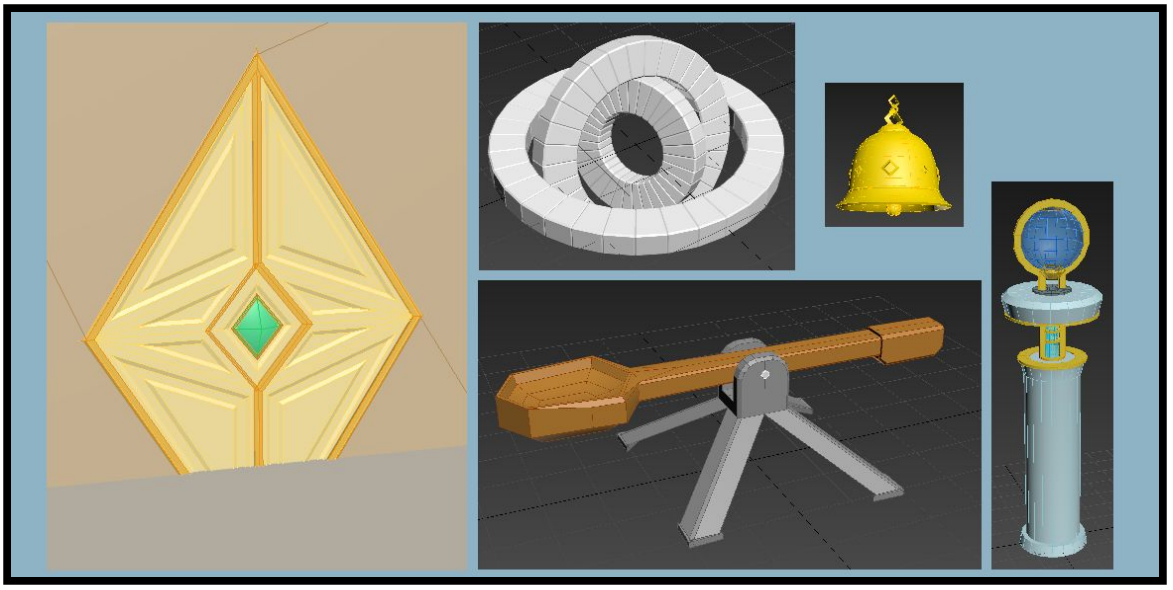

Figure 31: Example of different models made with 3dsmax

2D Art

For the elaboration of the different sprites and 2D images has been used Photoshop as the main tool and Paint as the complementary tool.

On the one hand, it has been used to make the textures of some materials and the stage terrain. These textures have not been made from scratch but they have been retouched and modified to increase saturation and brightness, to change some color or to be less marked so as to approach the visual style that is wanted to achieve. This effect can be seen in Figure 32.

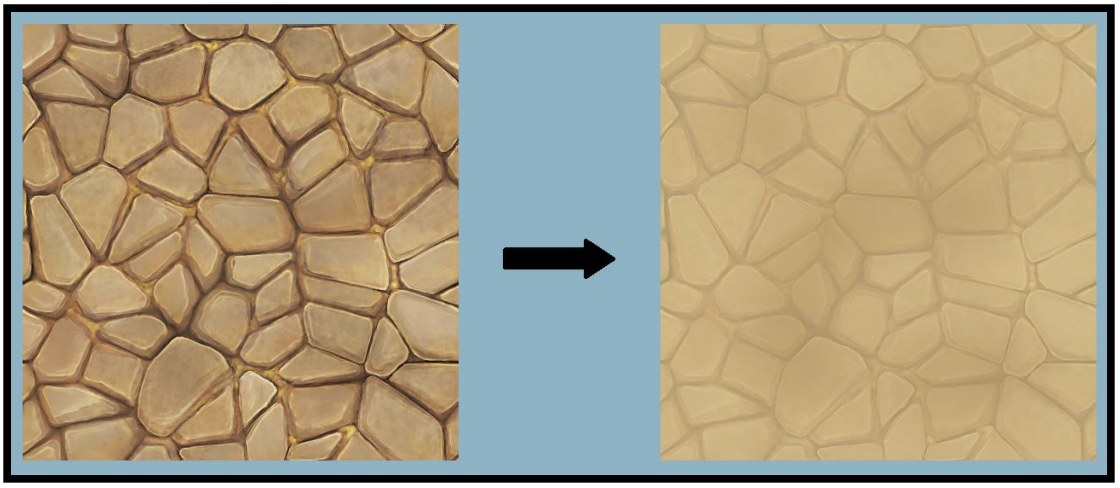

Figure 32: Original texture (left) and modificated texture (right)

On the other hand, it has been used to design and draw from scratch the skybox that is used for the game. For this skybox it has simply been necessary to make a blue gradient background, with a large sun peeping over the horizon and a couple of clouds. The clouds have not been drawn from scratch but they have been retouched to fit the skybox. At first it was necessary to make several versions with the sun of different sizes and to test as it remains in Unity, since it is hard to identify the correct proportions. Once the adequate size of the sun was achieved, the final version of the skybox shown in Figure 33 could already be obtained.

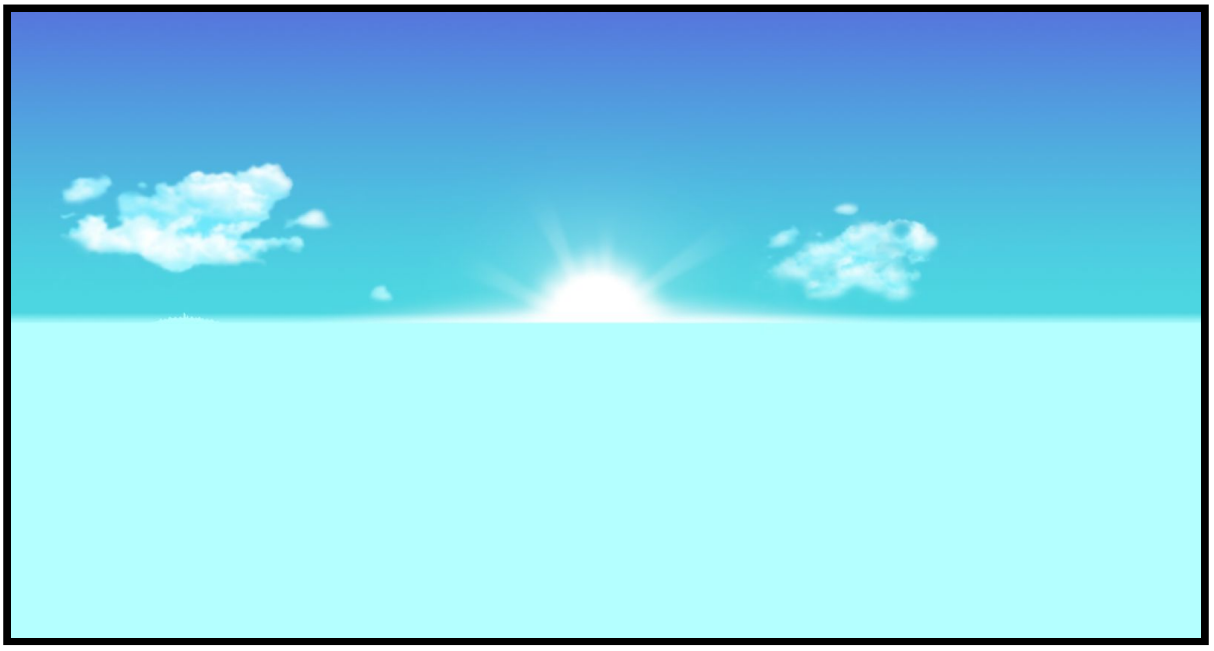

Figure 33: Skybox final version

It has also been used to create different types of grass, plants and flowers, retouching from other images. Some have even been used as is without retouching, such as the poppies. Since the Unity Terrain tool allows you to choose the color of the grass to be placed, some textures have been modified so that they are grayscale so that the color is better suited, as shown in Figure 34. Some have also been modified in shape and size to produce a better effect when placed on the ground.

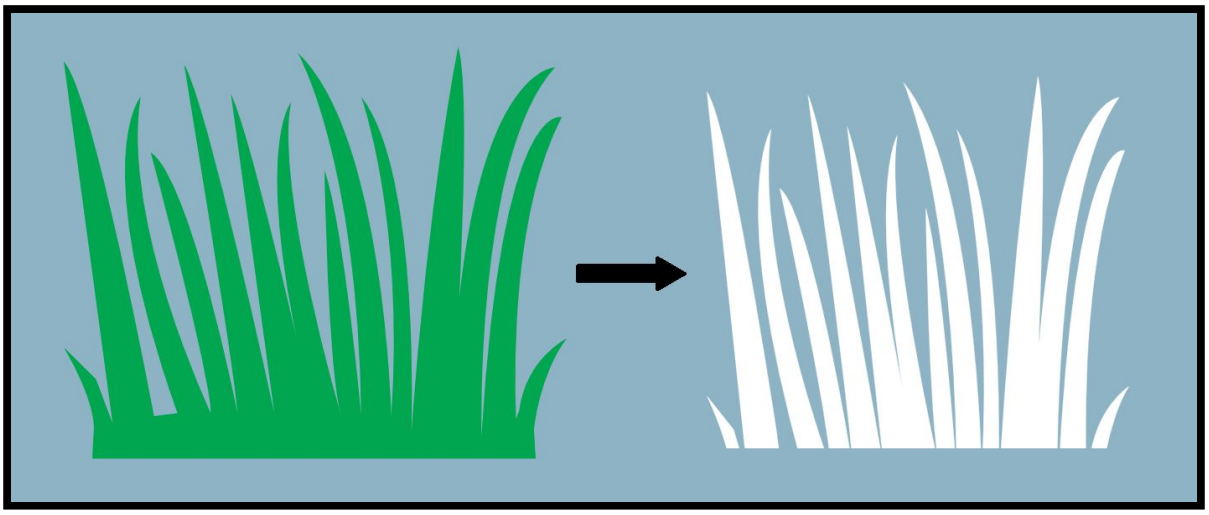

Figure 34: Original grass (left) and modificated grass (right)

Finally it has also been used to draw the different elements of the interface.

On the one hand the cursor and its different states as shown in Figure 35. The cursor, because it is a small element that does not take much space on the screen, has been decided to make small, so you can see the pixels.

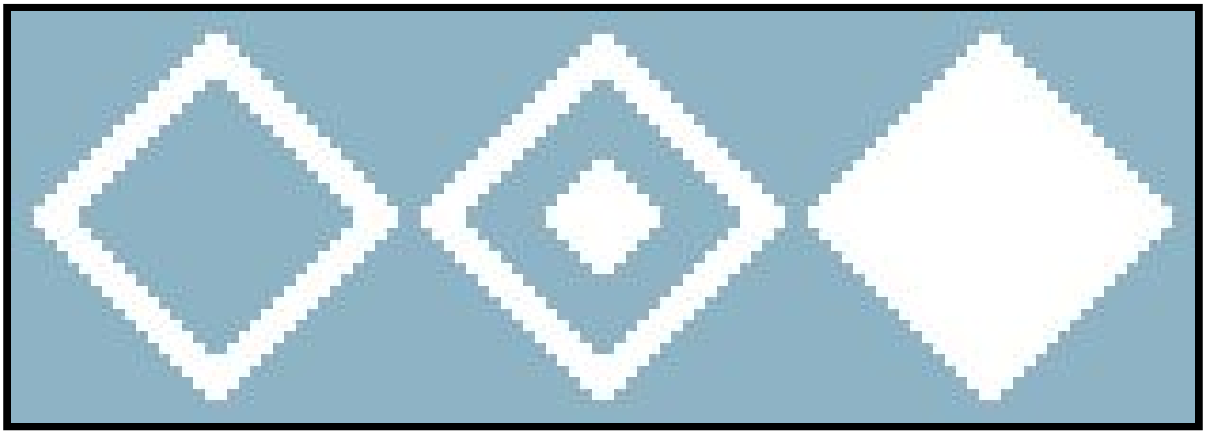

Figure 35: Image of the different cursor states

On the other hand for the map of the island, which can consult the player during the game. To elaborate this map has taken an orthogonal screenshot of the final island version seen from above and later has been edited in photoshop to give a drawing aspect. In addition, some indicators have been added to mark the location of the three keys on the map. You can see the final version of the map in Figure 36.

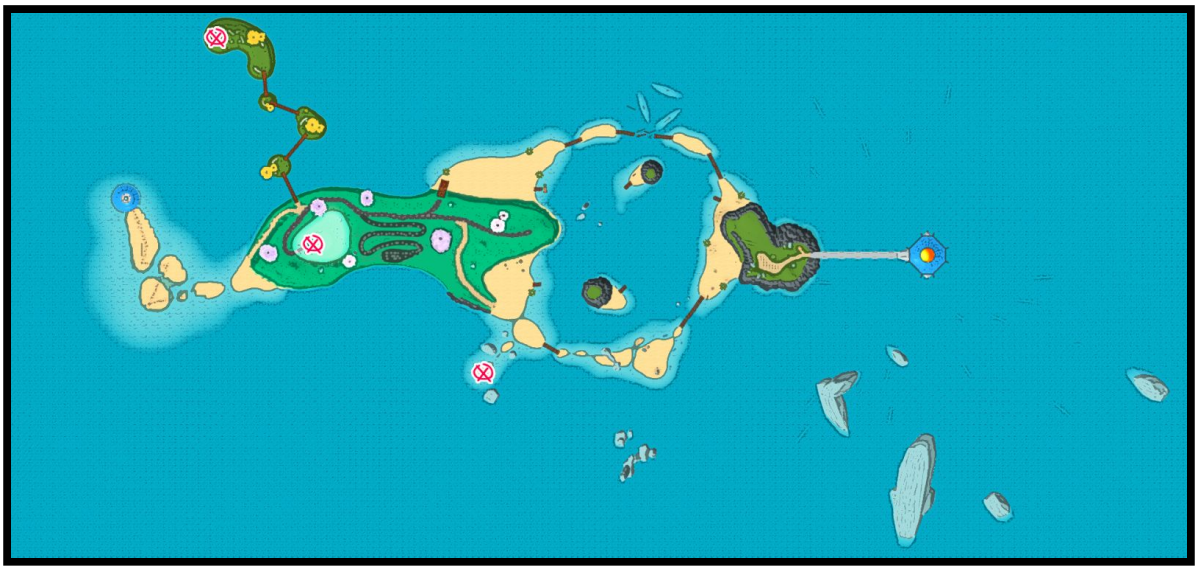

Figure 36: Final version of the island map

Finally, for the title of the game that can be seen on the cover of this document and also in Figure 37.

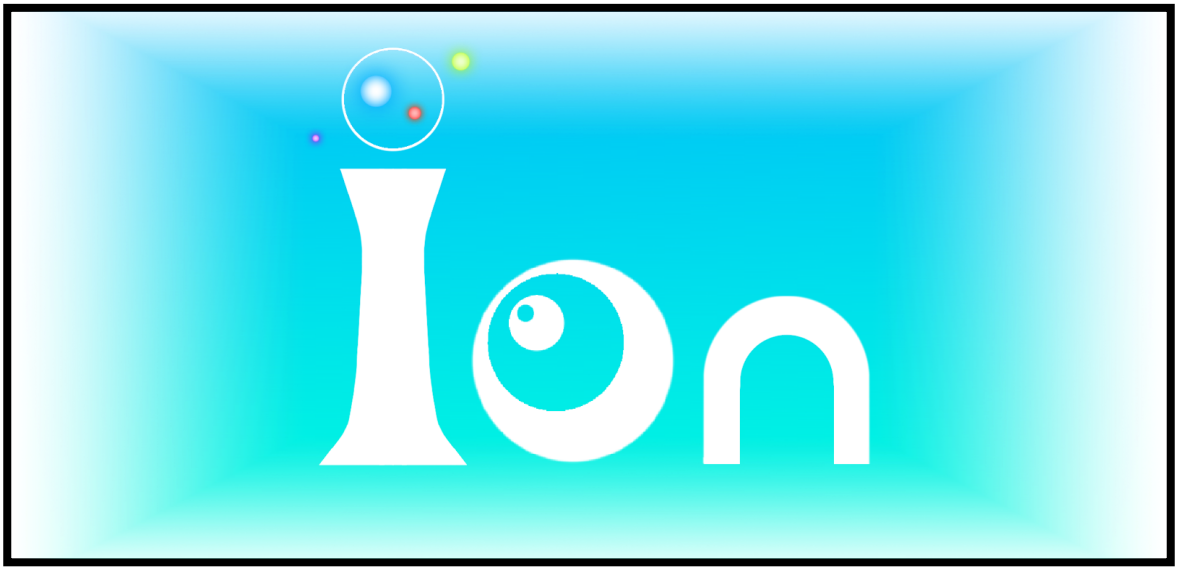

Figure 37: Final version of the game title

Sound

The sounds have not been developed from scratch but have been searched on different pages that offer music and sound for free with a Creative Commons license. Even so, it has been necessary to edit some sounds so that they can be played in a loop or to eliminate silent spaces. For this has been used an online tool called *MP3 Cutter Online* [11] very intuitive and easy to use that allows to do this kind of thing.

Technical Development

The elaborated demo consists of only two scenes. On the one hand, the *Main Menu* and on the other the *Game World*. Here is a brief explanation of the development of the *Main Menu*, as it is somewhat more secondary and has not required so much work. Subsequently, the rest of the *Technical Development* section will focus only on the *Game World*, which is the scene that contains the entire game and everything that has been developed.

Main Menu

In this scene you can choose to start playing or close the game with the buttons *New Adventure* and *Leave* respectively. For them it has been necessary to look for a suitable text font. Is also includes the game title previously developed. To place these elements it was necessary to create a canvas in Unity. This canvas has been configured so that both the title and the buttons adapt to the resolution of the screen, as this may vary depending on the device. To do this, it was necessary to add a *Canvas Scaler*, which has been configured as shown in Figure 38.

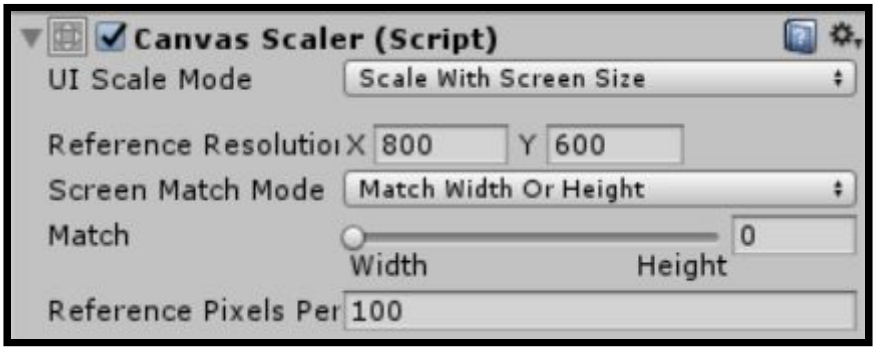

Figure 38: Canvas Scaler configuration

As for the menu background, it has been decided to place the lighthouse and a part of the island in the distance and the skybox with the sun shining behind. Also the boat has been placed floating in the water, since in the hypothetical final game, it is an important element that allows the player to sail between islands.

To do something more dynamic in the main menu has been added a series of effects. On the one hand, some particles have been added to the title and the screen itself for some movement, as shown in Figure 39.

Figure 39: Particles in the title

On the other hand has been programmed a script that causes the sun to shine more or less over time. This script can be applied to any *Light Shaft* of the corresponding asset previously discussed. In this case a *Light Shaft* has been used for the sun's rays and the effect remains as shown in Figure 40.

Figure 40: Light Shaft intensity comparison

Finally another script has been programmed that causes the tide to rise and low as time goes by. In addition, the elements that are in the water like the boat or the wooden boards, are also affected by this tide. It has simply been necessary to place these objects as children of the object that acts as sea, as shown in Figure 41.

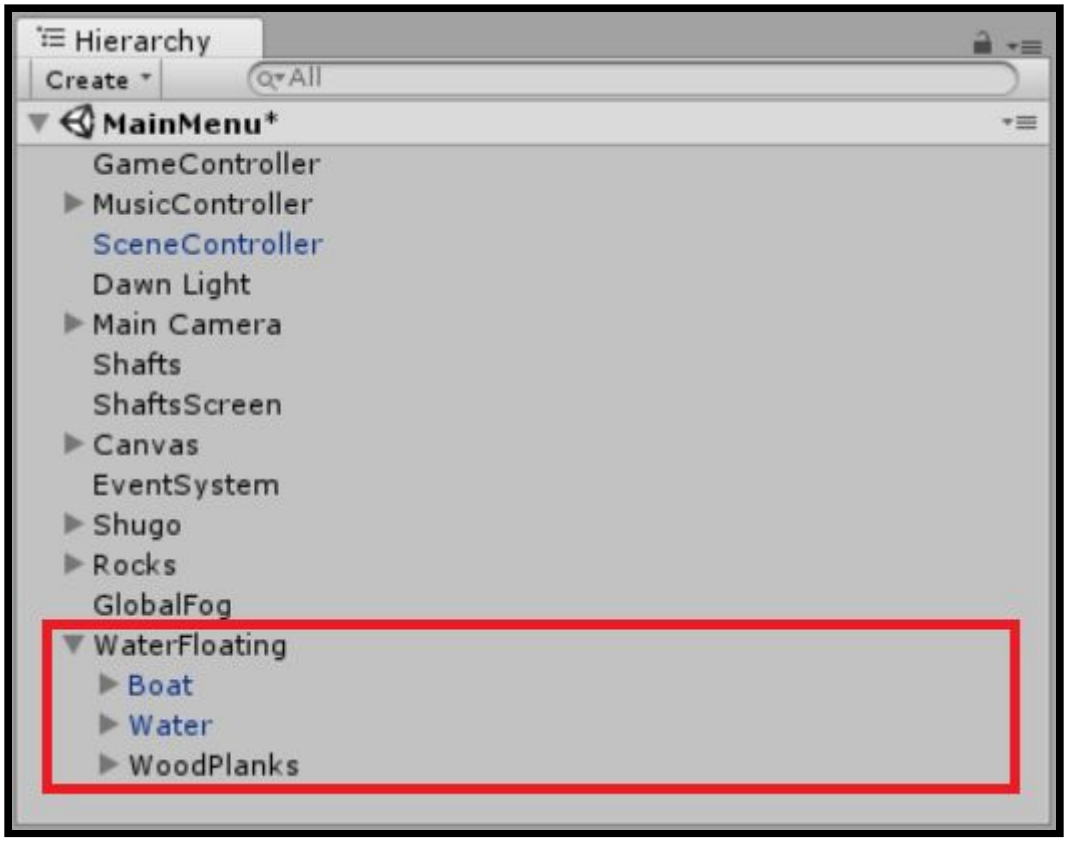

Figure 41: Objects as childs of the water plane in the Hierachy

Finally, to pass from one scene to another a script is programmed that allows to do both *Fade In* and *Fade Out* transitions. You can also use any image for the transition. For this particular project it has simply been decided to use a blank image as a transition.

Physics

Unity comes with its own functions and objects that are responsible for calculating different types of physics. For this project only the physics for spheres have been used, since they are the only physical elements of the game. This applies both to the orb of the Gyro Temple and to the micro orbs of the puzzles. To use these physics it was not necessary to program or modify anything other than to use the

default functions, so it has been relatively simple. However, a little trick has been used for the physics to work correctly in the puzzles, because when the superplexus were rotated, the micro orbs inside it behaved strangely even going through the small obstacles inside the puzzle. This is due to the speed with which the superplexus is rotated among other factors. To solve this problem what has been done is to place the micro orbs as children of the rotating element. Thanks to this small adjustment the physics function correctly.

Effects

To achieve a good ambience and a striking aesthetic, sound, camera and environment effects have been added.

Sound

The sound has been a very important aspect in the development of the game, since it has been wanted to take into account and to work it carefully and in detail. Both 2D sounds and 3D sounds have been created. 2D sounds sound homogeneously and always at the same volume. The sounds in 3D adapt their volume taking into account the distance to the object that has the *AudioListener*, which in the case of this project, is the First Person. The following explains the different types of sound that have been created and programmed.

Ambiental

The ambient sound refers to all those sounds of the environment that make the player more immersed in the *Game World*. Two types of ambient sound can be distinguished. On the one hand, there are those sounds of some element. In the project it has been used for example for the fire sound of the torches or the static sound of the neon lights found in the puzzles. These sounds sound low and you have to get closer to perceive them better. On the other hand, there are those sounds of the environment, which do not come from a particular sound source. In the project it has been used for the sound of the sea and the sound of the wind. Both sounds are always present throughout the game, but a script has been programmed that changes the volume of both sounds taking into account the Y coordinate of the player, more specifically, their distance from sea level. Therefore, when the player is closer to the sea, the sound of the wave is the one that sounds stronger and when it is farther away, such as at the top of the lighthouse, the sound of the wind is the one that sounds stronger.

Area Music

The game does not have background music that sounds throughout the entire game world, but there is some melody that sounds depending on the area in which the player is. For this a script has been programmed that allows to define areas with colliders as triggers and to define which melody sounds when it is inside that zone. In the project only this script has been used to define a particular melody when it is in the area of the lighthouse, so that place transmits a sense of mysticism. In addition, in this specific case it has been decided that the volume of the melody depends on the distance to the sea as well as the sounds of waves and wind, using the same script. In this way, the melody sounds louder the higher it is in the lighthouse.

Modifiers

To give a greater sound realism, a *Reverb* effect has been applied inside the lighthouse which makes the sounds resonate giving an echo effect. This effect does not need to be programmed since Unity allows to add this type of modifiers in spherical areas. Therefore, the only thing that has to be done is to place several of these areas along the lighthouse.

Feedback

One of the objectives of the project is to get the player to receive good feedback both visual and audible so that he can identify what happens. In the case of sound, this feedback has been applied on the one hand to the cursor and on the other hand to the puzzles. In the case of the cursor, when you point to an element with which you can interact, a slight sound is played, and when interacting, another is played a little more intense. Thanks to this the player can easily identify with what elements can interact. As for puzzles, each time a micro orb exits a sound is played and once the puzzle is completed, another sound is played that identifies whether the puzzle has been resolved correctly or not. If not, the puzzle restarts, playing a very loud negative sound.

Footsteps

The *First Person Controller* (FPC) that comes by default in Unity already comes with its own step sounds. However they are very basic and repetitive sounds so is opted to program a script called *PlayerWalkableArea* that allows you to improve this and add different types of sounds depending on the type of floor being stepped on. For this a vertical raycast has been used that leaves the base of the FPC and that detects the collider of the ground on which it is walking in each moment. Along with this, different tags have been defined to identify what

material each collider is. The tags that have been defined are stone, metal, wood, water and earth. Subsequently, these tags have been assigned correspondingly to the different elements of the scenario. In addition, the script also takes care of varying slightly the pitch of the sounds of random form. As a negative point to this method, since it works from colliders, there has been no way to define different sounds for ground and for grass, since both elements use the same collider of the terrain. One solution might be to try to define triggers for all grass areas but it would be a slow, tedious and not very effective process.

Physics

An important part of the project focuses on the sphere physics which are used in puzzles. To improve the sound experience has been decided to program a physical sound script called *PhysicalSound* that ensures that the orbs reproduce sounds when they collide and when they roll. The script is explained in detail in the *Scripts* section.

Camera

As previously mentioned, the *Post Processing Stack* asset has been used, which allows to use camera effects in a more comfortable and optimized way. The effects that have been used are as follows.

Antialiasing

This effect is used to improve the quality of the rendered image, causing the edges of the geometry not to be seen with sawtooth, but smoothed. To avoid consuming many fps, is chosen to select the default quality of *Antialiasing*, as shown in Figure 42.

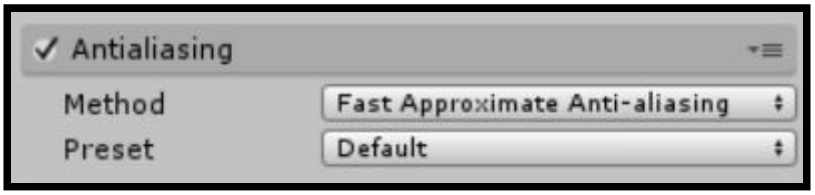

Figure 42: Antialiasing settings

Ambient Occlusion

This effect, like *Antialiasing*, serves to improve image quality. In this case, the *Ambient Occlusion* is responsible for obscuring those areas where the geometry is very close together. The options that have been chosen are shown in Figure 43.

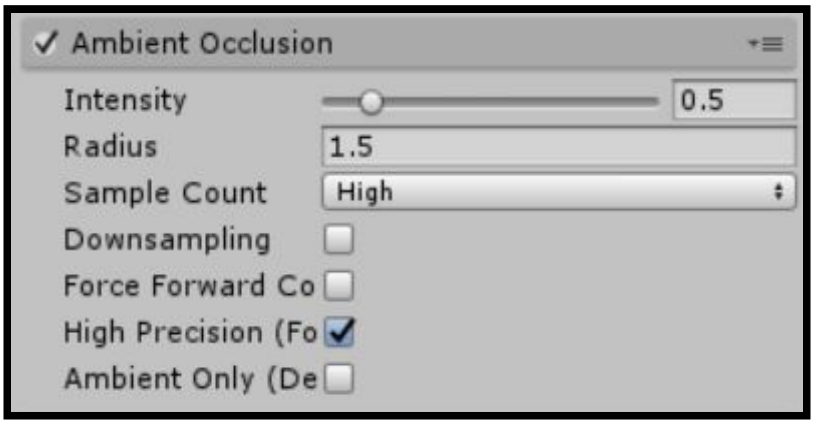

Figure 43: Ambient Occlusion settings

Bloom

The bloom effect has been used for two different purposes. On the one hand for the camera to render the stage with more lighting and saturation, something that is necessary to maintain the visual style presented. On the other hand, for the neon effect, using materials with *Emission* and HDR lighting. This is explained in more detail in the *Optimizations* section. Neon is used for wiring the stage and to illuminate the puzzles and buttons. The options chosen are shown in Figure 44.

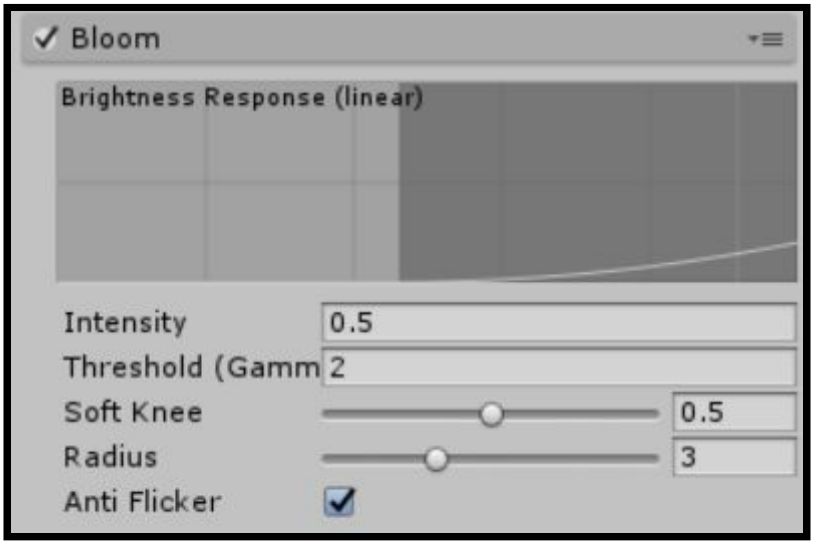

Figure 44: Bloom settings

Environment

For the environment of the game have wanted to add small details that make it look more careful with a greater immersion feeling.

Fog

Two types of fog have been used. On the one hand, the default type that is in Unity, which allows to add mist based on the distance from the camera. This

effect has been added very slightly, so only those elements that are really far away are affected by the fog. This effect also allows the horizon line to be blurred to soften the line that separates the sky from the sea.

On the other hand the asset *Light Shafts* has been used to create a height fog effect. This type of fog takes into account the height. It has been used for the interior of the lighthouse to give a feeling of dense air and humidity. This effect can be seen in Figure 45.

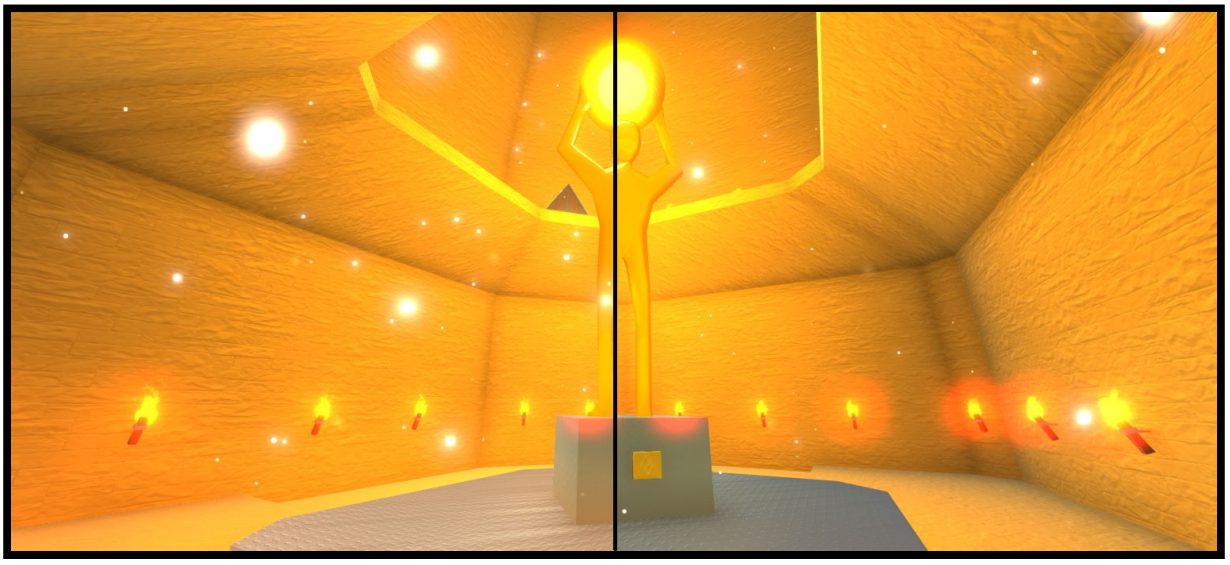

Figure 45: With and without Height Fog effect comparison

The *Light Shafts* asset is not mainly intended for this but has been configured in a certain way that can achieve such effect. Of course, you have to make sure that the collision mask is in *Nothing* so that no element blocks the fog effect.

Particles

A lot of particle effects have been used for different elements and specific areas. All these effects have been configured using the Particle System implemented in Unity and its use has been very similar for all the effects that have been elaborated. To explain this process will be used as an example the fire particles used in the fireplace and the torches.

The first task is to create the material of the particles as shown in Figure 46. For these particles a sprite of a white circle has been used and an *Additive* material has been chosen.

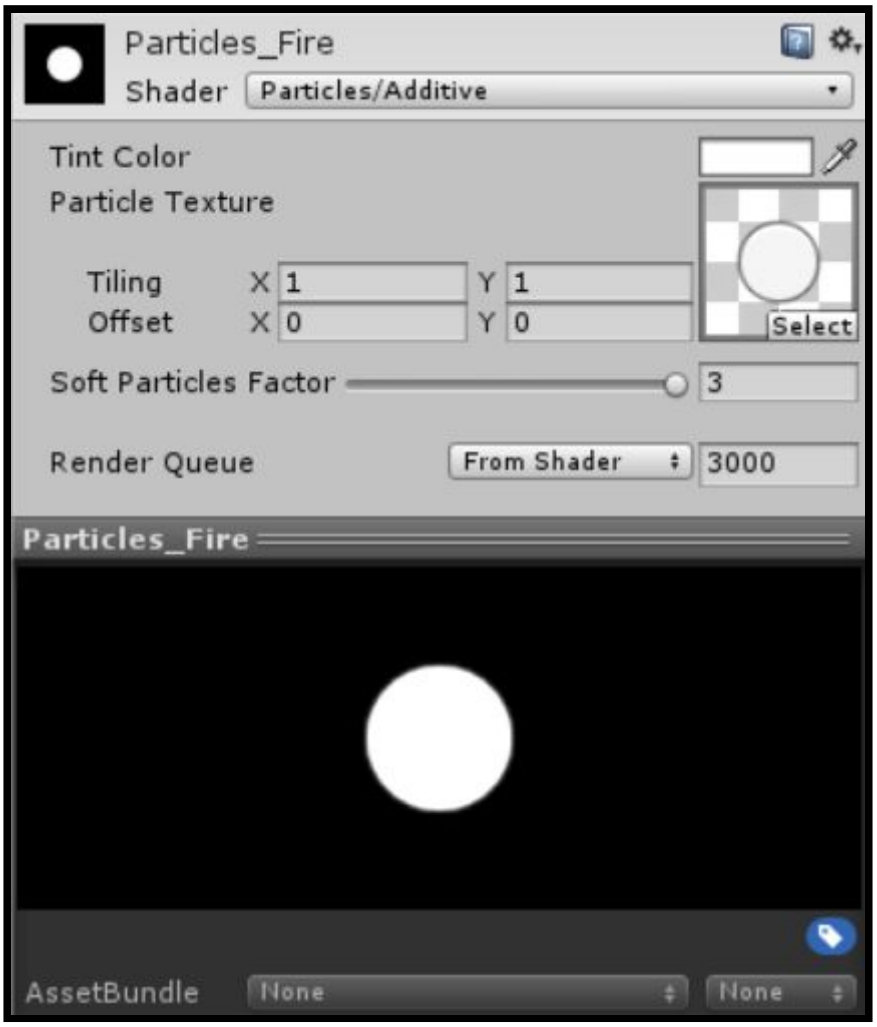

Figure 46: Fire particles material

Later an empty object has been created with the *Particle System* and the material created has been placed in the *Renderer* tab as shown in Figure 47.

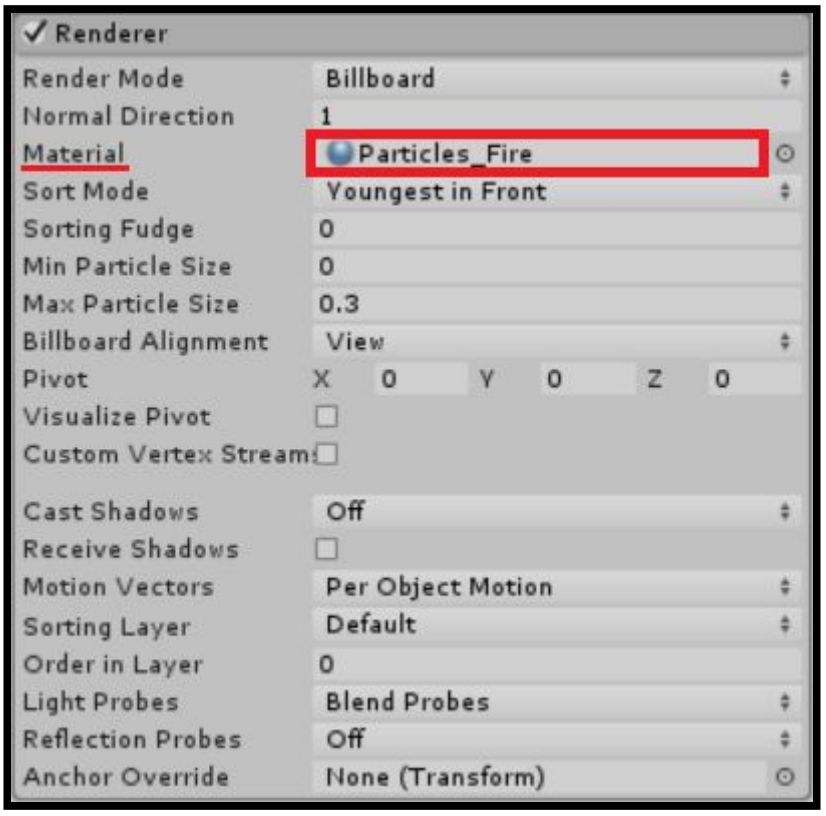

Figure 47: Fire particle system renderer

Finally the different options such as colors, lifetime, shape, etc. have been configured to achieve the desired fire effect, as shown in Figure 48.

Figure 48: Final fire particles effect

Below are other types of particles that have been created, such as foam when walking on the water or when sailing with the boat (Figure 49) and the floating white particles of the small temples where the keys are found (Figure 50).

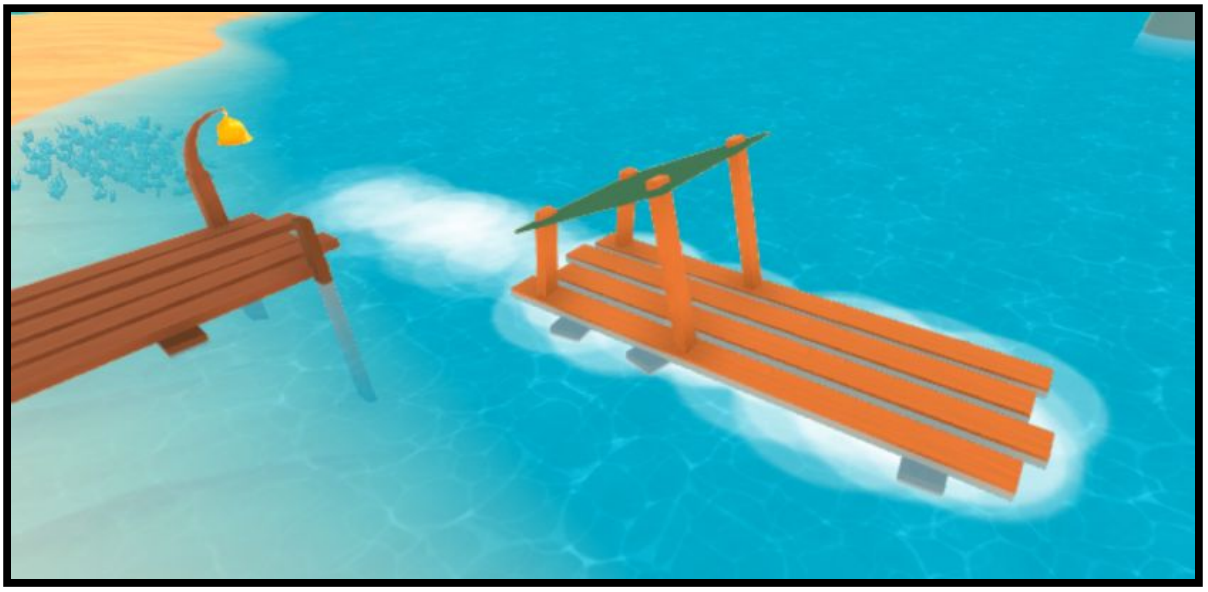

Figure 49: Water foam effect

Figure 50: Shiny floating dots effect

Light Shafts

Aside from using the *Light Shafts* asset to reproduce the *Height Fog* effect, it has also been used for other things. On the one hand for the effect of sunbeams. A large *Light Shaft* has been created that covers the sun of the skybox and points in the same direction as the directional light that is used as the sun. Then, a script is programmed called *SynchronizedPosition*, which causes the Light Shaft to follow

the movement of the *First Person Controller*, so it is always present without creating a huge one that covers the entire scenario.

In addition to emphasize this effect has been programmed a script called *ShaftAdapter* that is in charge of increasing the intensity of the sunbeams when it is in the zone of the lighthouse, of this form the rays pass between the columns giving a very striking effect as it can be seen in Figure 51.

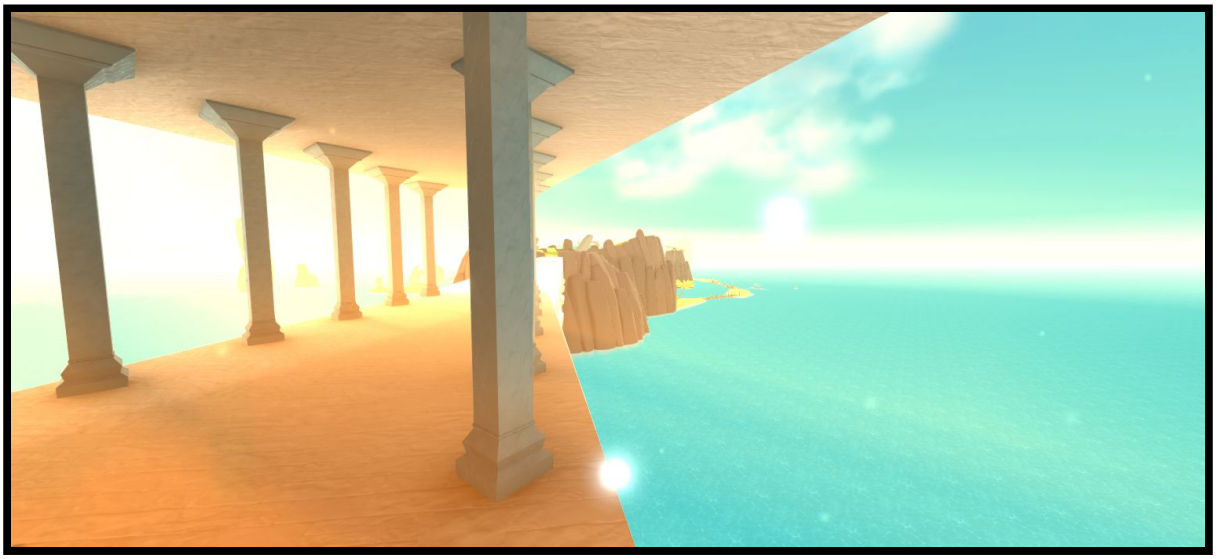

Figure 51: Sun shafts effect

Volumetric Lights

The Volumetric Lights asset has been used to give volume to pointlight type light sources such as the orb. This effect illuminates the entire spherical area around the light. To use this asset you must, on the one hand, add the *Volumetric Light* script to a light source and on the other hand you have to place the *Volumetric Light Render* script on the camera, so that this effect can be rendered. Figure 52 shows the effect of door crystal.

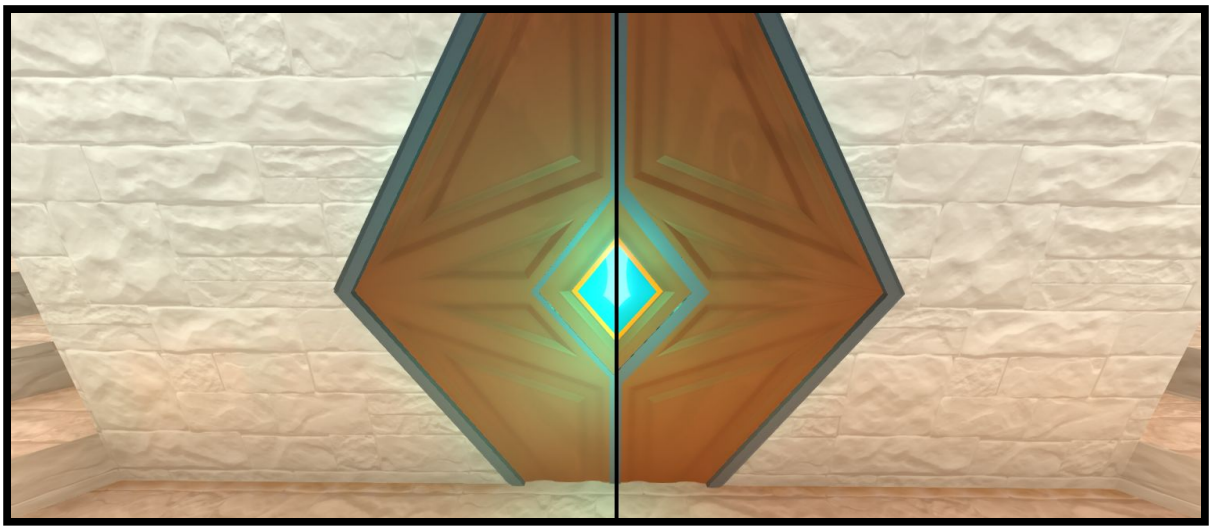

Figure 52: With and without Volumetric Light effect comparison

Shaders

Although the standard shader has finally been used for most materials, other secondary materials have also been used.

Water

The water shader that comes in the Unity standard assets has been used to create the plane material that has been used as the ocean. However the result that was obtained was not the desired one so a second plane has been created with a transparent material that highlight the coastline. In Figure 53 you can see the comparison and the big visual difference that this supposes.

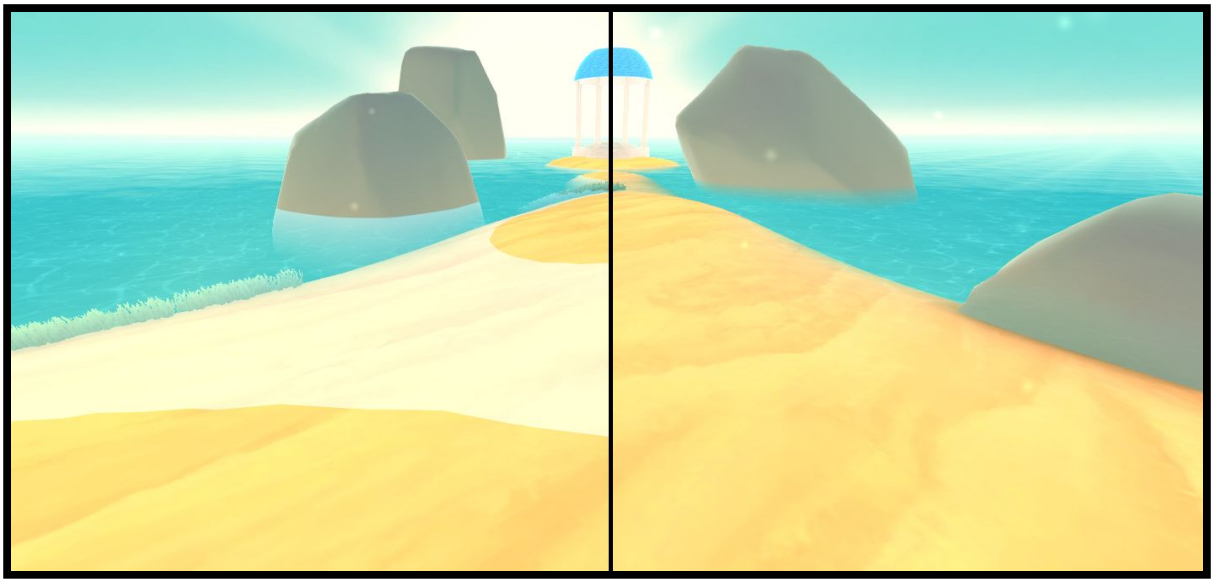

Figure 53: Water shader with and without transparent plane comparison

Toon

This shader is included in Unity and produces an effect similar to Cel shading. It has been tested to use materials with this shader but the resulting aspect did not resemble the raised visual style, so it has been discarded. In addition in the baked lighting, these types of materials looked bad with shadows and stains throughout the object.

Cel Kernel

An own invented shader has been programed using one of *Cel Shading* as a base. The cel shading is characterized by interpreting lighting in a segmented way. Instead of interpolating the light from the most illuminated point to the least illuminated point, what it does is to stay with intermediate values, without interpolating. These segments are called cuts. The comparison of a standard shader and a cel shading shader can be seen in Figure 54.

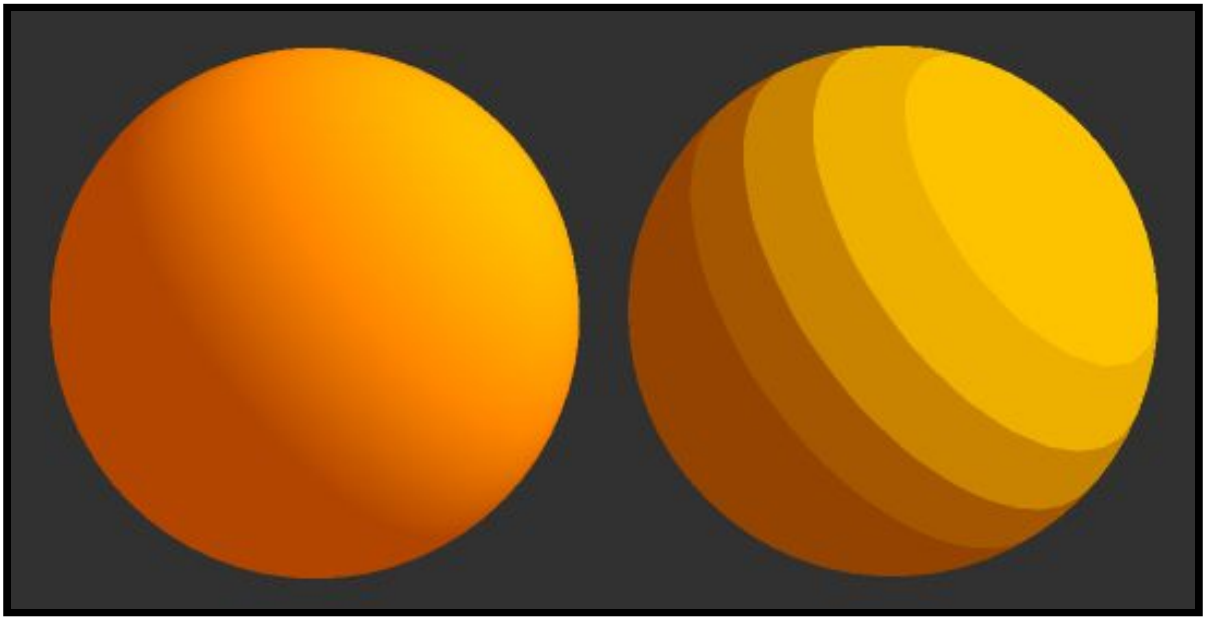

Figure 54: Standard shader (left) and Cel Shading shader (right) comparison

The main modifications that have been made to create the *Cel Kernel* focus on the type of diffuse and specular lighting, and to add properties that allow you to customize colors and different values like number of cuts or the proximity of them.

On the one hand, three types of diffused lighting have been programmed, each offering a different result. You can choose which one to use with the *DiffuseType* property that has been created for this purpose. The first type is the basic one that came by default in the shader of *Cel Shading*. Basic diffuse illumination is

calculated as the vector product of the normal surface by the direction of light. The other two types are small modifications of the previous one, changing some of the parameters that make the calculation. One of them is calculated as the vector product of the normal surface by the direction of view (the FPC camera) producing a frontal illumination effect. The other one calculates the vector product of the two previous ones obtaining a very curious mixed effect, that is the one that has been used mainly for the orbs. The comparison of the three types of diffuse illumination can be seen in Figure 55.

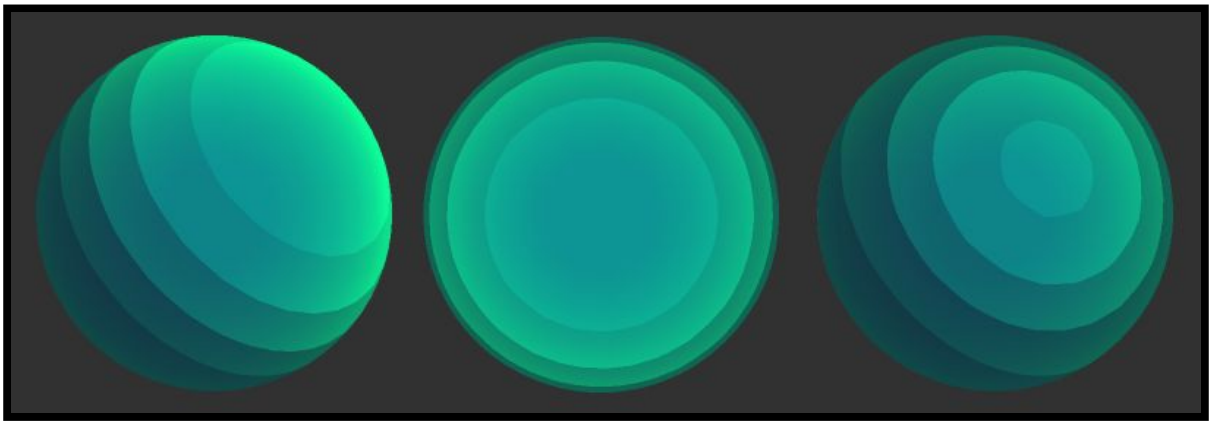

Figure 55: Diffuse Types, from left to right, basic, frontal and mixed

On the other hand has been added specular lighting, which did not come in the shader. But it has not been added with the aim of producing specular illumination effect, but has been used for another purpose. This lighting has been used to create the shader kernel effect, which is the main feature of this shader. The effect that was wanted to be that any object with this shader will show a kind of colored kernel in its interior. For this purpose the specular illumination has been calculated in the same way as the type of diffuse frontal illumination. This calculation interprets that the camera is the source of light, therefore, you look from where you look, you will see the specular illumination in the center of the object. Different properties have been added to customize the size and color of the kernel, as well as the colors and intensity of the lights and shadows among others. In Figure 56 you can see several examples of materials that can be achieved with this shader.

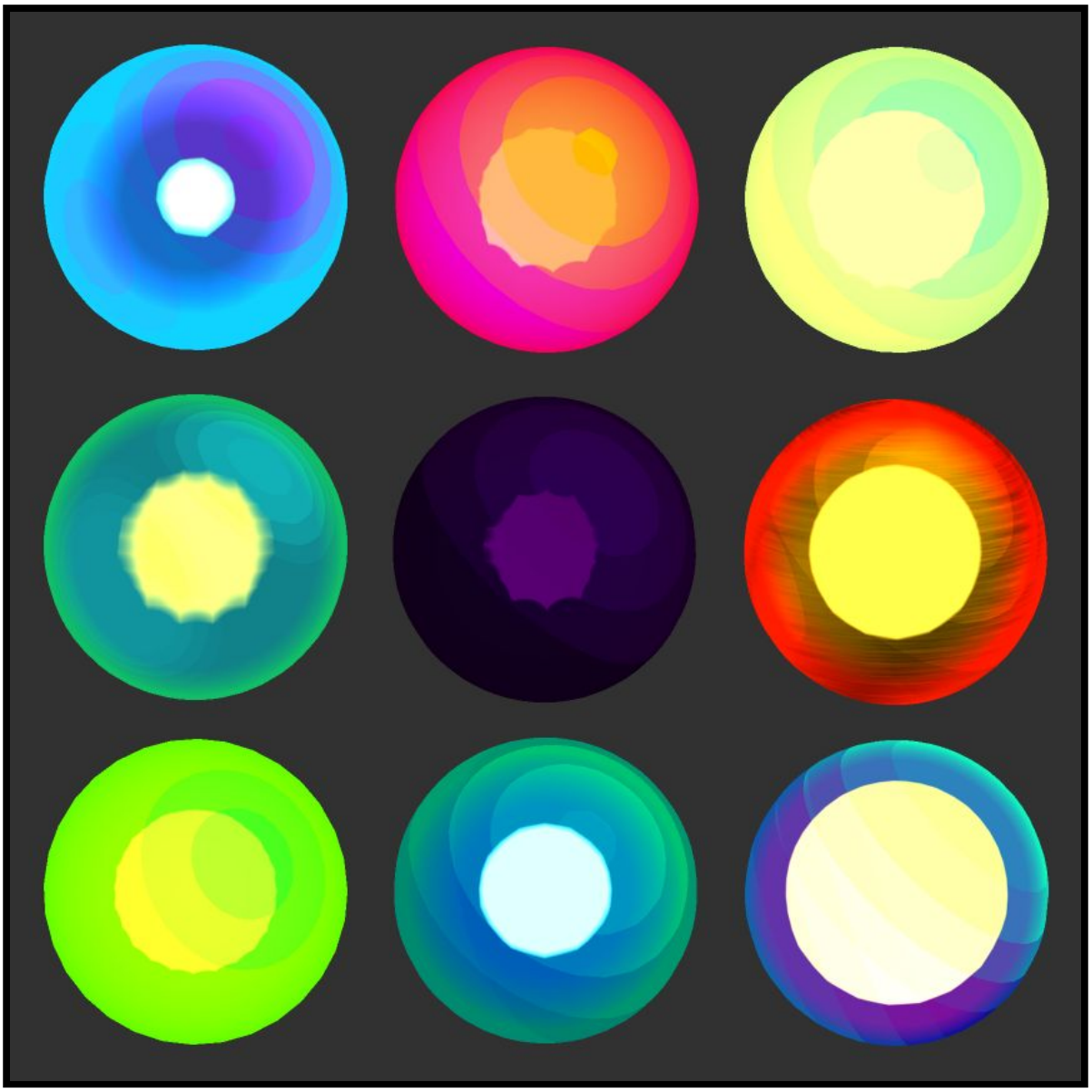

Figure 56: Cel Kernel shader material examples for the orbs

Animations

To make some elements have predefined movements, animations have been made using the Unity *Animator* tool. This has been used only to animate catapults and elevators. For the catapults it has been necessary a single animation, which causes the catapult to produce a launching movement. However for the elevators it has been necessary to create two animations, one of rise and one of descent. In order to activate these animations some triggers and booleans have been created that are called from the scripts of these objects themselves (Catapult script and Elevator script).

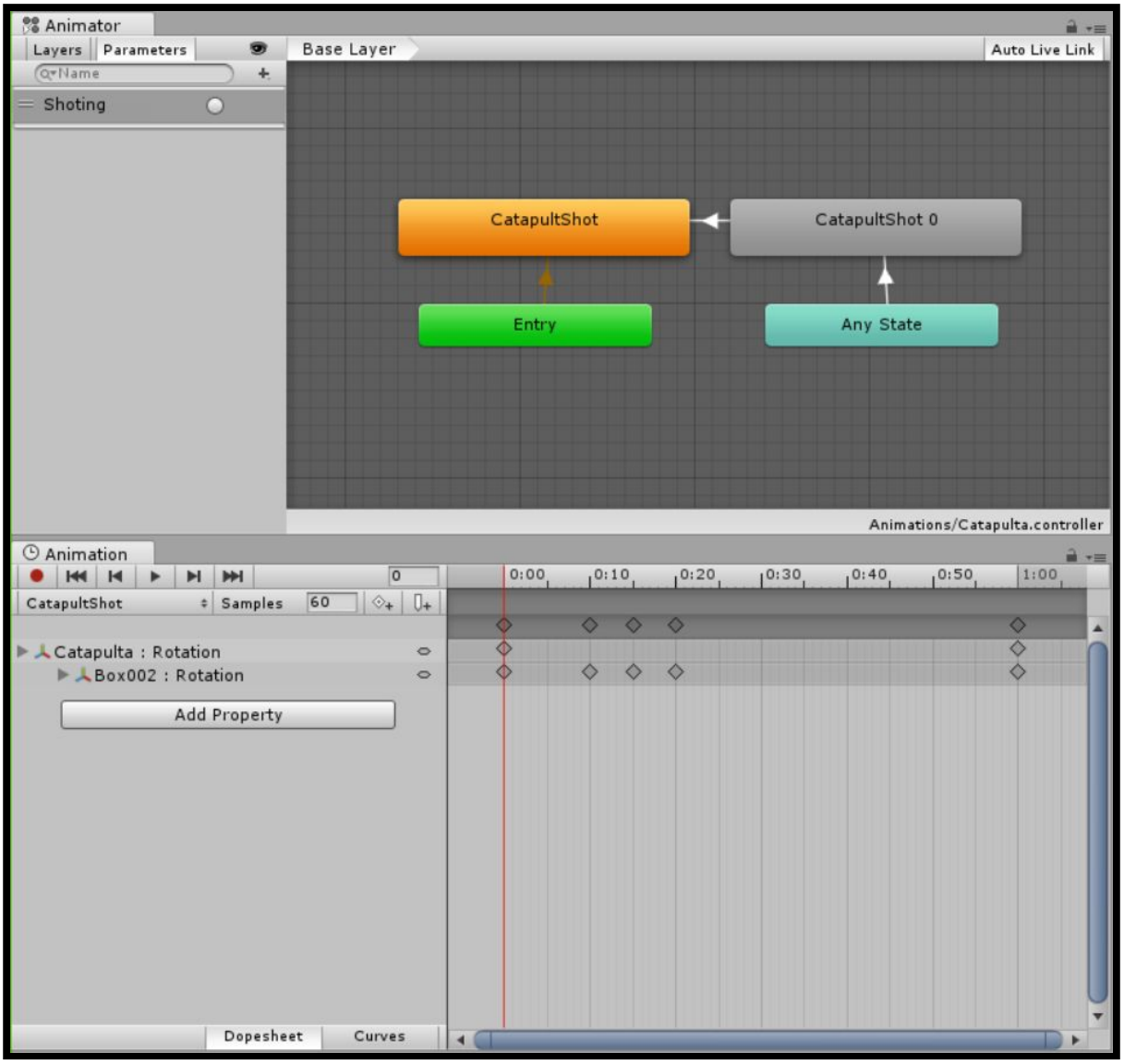

Figure 57: Catapult animation graph (above) and catapult animation timeline (below)

Scripts

For the demo behaviour it has been necessary to program very diverse scripts of all kinds. The most relevant are grouped according to their main functionalities below.

Player Scripts

As mentioned before the player will move around the game world using a *First Person Controller*, a camera in first person. The FPC has been created using the basic FPC that comes in the standard assets of Unity as a base, since it lacked some functions necessary to achieve the desired result. The controls chosen for the FPC are WASD for the movement, *Shift* for the sprint and the mouse to move

the camera. The following explains in detail how the FPC has been modified and that other scripts are directly related to it.

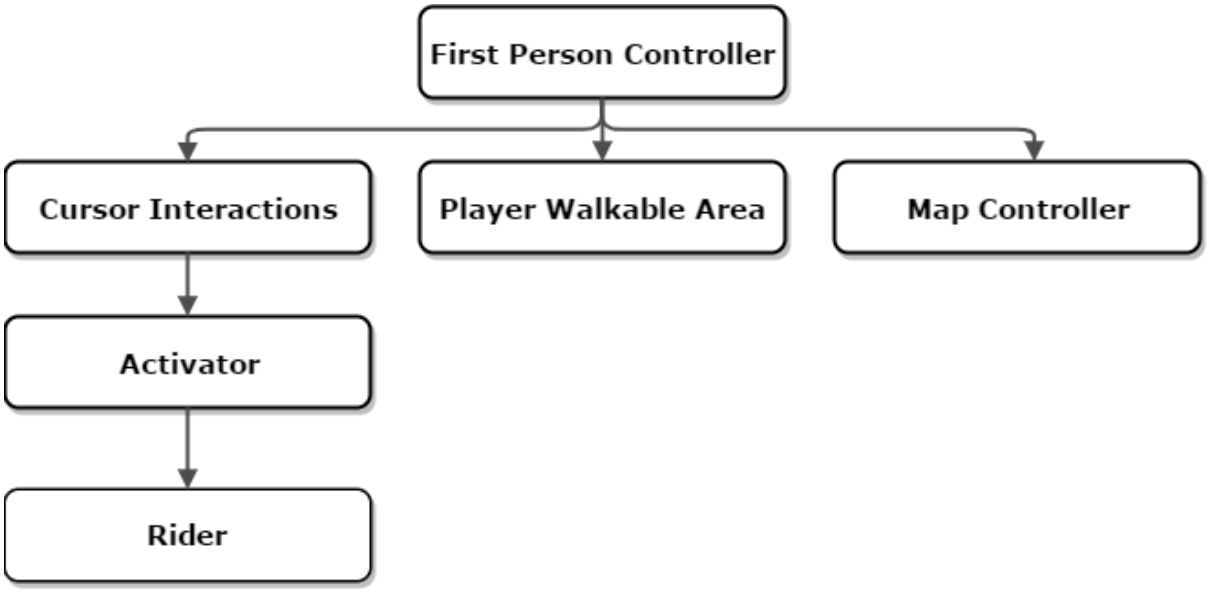

Figure 58: Player Scripts flow chart

First Person Controller

The movement of the basic FPC was too abrupt so an acceleration value has been added so that it does not go from speed 0 to maximum speed instantly but it does it progressively. In addition this included some functions that were not necessary for this project, so they have been eliminated. These are, on the one hand the head bobbing effect of the camera, which tries to simulate the head swing of the character and on the other hand the option to jump. The first option has been eliminated because it was too dizzy and the second option simply because in this game is not contemplated that you can jump.

In addition to this have been added some public functions that can be called from other scripts. On the one hand, the *Freeze()* and *UnFreeze()* functions that block the character's movement and hide the cursor (which is controlled by the *CursorInteractions* script). However the camera movement can still be used freely. These functions are called from the *PuzzleController* script for when solving a puzzle and from the *MapController* script when you are browsing the map. On the other hand *Ride()* and *UnRide()* functions that work similar to *Freeze()* and *UnFreeze()* but these are called by the *Rider* script when the boat is being controlled. Finally the function *SetFootstepSound()* that allows to define the sound clips that correspond to the steps including also the base pitch and volume. This function is called from the *PlayerWalkableArea* script every time the FPC steps on

a new terrain type. Apart from these changes, the only thing that has been done is to configure different options and values that already come by default in the FPC. This includes defining the values of speed of movement and sprint as well as values of movement and sensitivity of the camera among others.

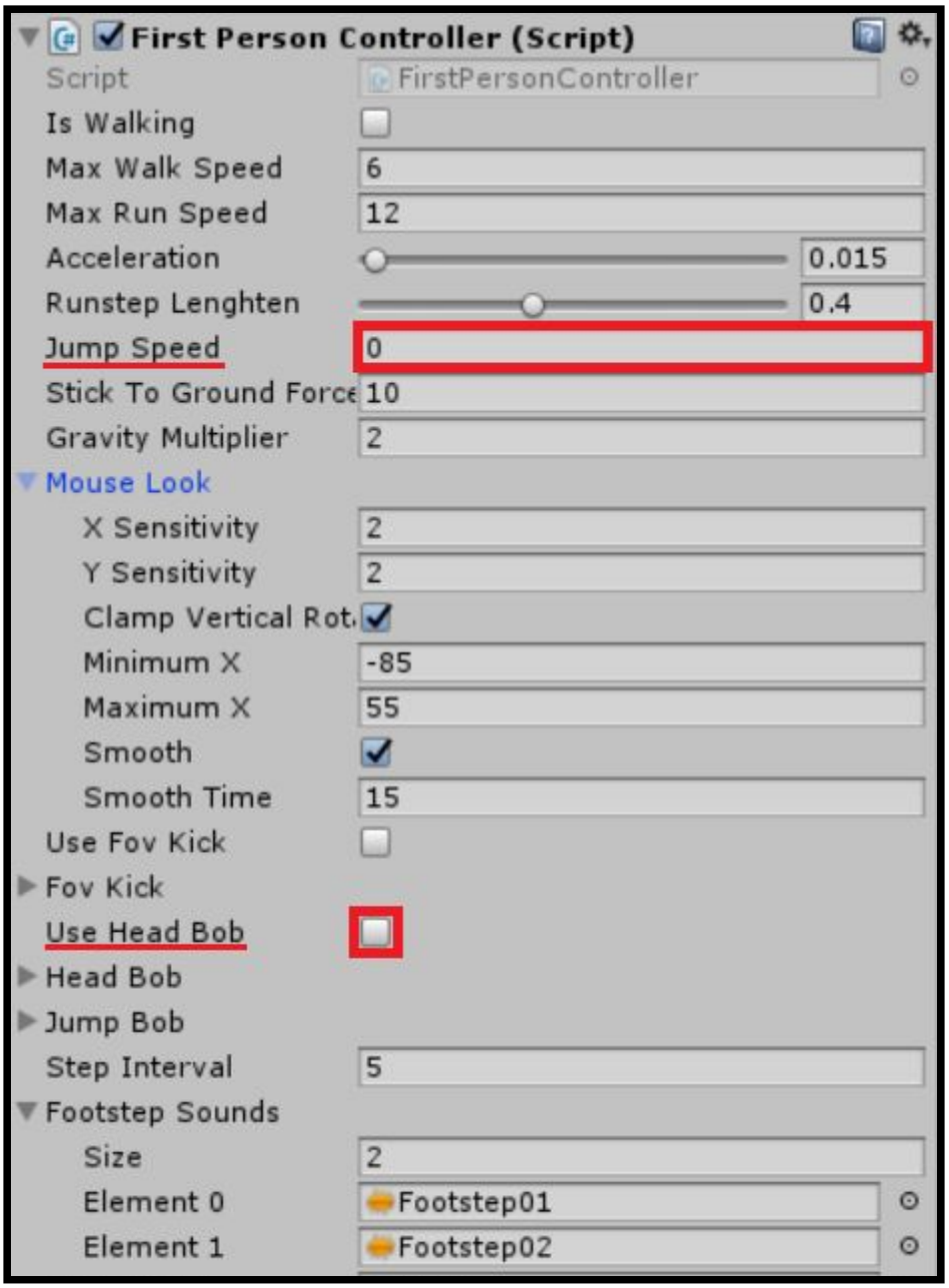

Figure 59: FPC final values configuration with Head Bobbing and Jump options deactivated

Cursor Interactions

This script is responsible for controlling the different states of the cursor, which are *Normal*, *Targeting* and *Pressed*. These states have been defined using an enumerator and each state has been assigned a sprite. To show a state or another

of the cursor, on the one hand a front raycast is used that leaves the camera of the FPC and on the other hand is controlled which buttons of the mouse are pressed. The front raycast, if it is detecting an object with the *Interactable* tag, the cursor goes to *Targeting* state and if it does not detect any, it changes to *Normal*. You can set the distance of the raycast in the inspector. If the left mouse button is pressed, the cursor changes to the *Pressed* state. However, the *Pressed* state performs no function unless it happens just after the *Targeting* state. If there is nothing with which to interact, no matter if the mouse is pressed or not.

Apart from this, a higher state called *Hide* has been prioritized, which hides the cursor if the FPC is in the *Freeze()* state, as previously explained. If any mouse button is pressed while in the *Freeze()* state, it automatically exits from this state by calling the *UnFreeze()* function. This has been done willfully so that the player can decide when to stop interacting with a puzzle.

Player Walkable Area

Script in which are defined all the sound clips of steps for each type of terrain and that is responsible for checking what type of terrain is stepping on the FPC at any time through a vertical raycast. Each type of terrain has been defined with a different tag so that raycast can detect it and know how to identify each terrain.

In addition, this script has another important function, which is to define the zones where the FPC can be moved. For this a mask has been used which has been called *Walkable*. This mask has been put to all those elements on which the player can walk. This includes the lighthouse and the lands of the islands among others. If the player tries to walk on something that does not have the corresponding mask, he simply can not advance. On the other hand, to prevent the FPC from getting into the sea, you could not use the mask since the land collider above the sea is the same collider as the land under the sea. Therefore the player's movement on the Y axis has been limited. That is, the player can not move from a certain height downwards. In this case, this height is a little below the sea level, which is at 22 in the Y coordinate. Therefore, the player can walk into water only in those areas where he does not just cover but can not enter more. As an extra, when you step on the water, other than sounding the corresponding sound, particles that produce a foam effect are showed.

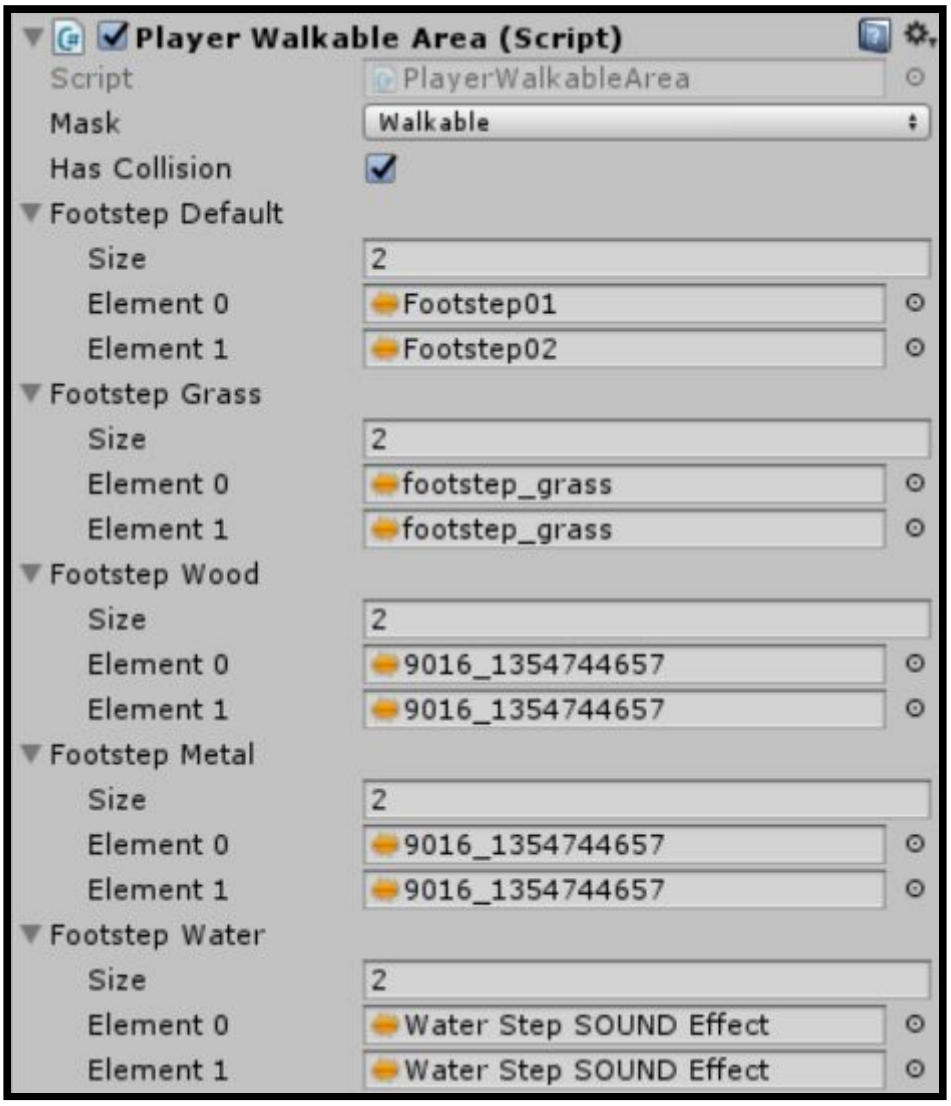

Figure 60: Defined footsteps sounds and Walkable mask from Player Walkable Area script

Rider

This script on the one hand is responsible for controlling when the player is mounted or dismounted from the boat. To do this, the boolean *IsRiding* is used to know if the player is already mounted or not and with that information the Ride() or *UnRide()* function of the FPC is called. This script is activated when the player interacts with the bells that are on the docks. If the boat is in that dock, then the player is automatically mounted. To verify this, the distance between the bell and the boat is calculated, and if it is at a distance of 15 units or less, it is valid. The same rule applies when trying to moor. To do this you have to navigate to the dock where you want to tie up and interact with the bell.

Figure 61: Dock with bell and boat

On the other hand, it is also responsible for allowing the player to operate the boat with WASD. For the W and S keys a positive or negative *AddForce* is used to give speed forward or backward. On the contrary for the keys A and D a positive or negative *AddTorque* is used to give a turning force to the boat in one direction or another.

Map Controller

It is responsible for showing and hiding the map when the M key is pressed. This only works if the player has picked up the map before, which is located near the fireplace as shown in Figure 62. When the map is displayed or hidden, The *Freeze()* and *UnFreeze()* functions of the FPC are called respectively.

Figure 62: Folded map in a stone next to the fireplace

Activator

It is a very important script that is used in all those elements with which you can interact. It acts as an intermediary and is responsible for calling other functions of other scripts. For example when interacting with a button on a catapult, the

Activator calls the *Shot()* function of the *Catapult* script. This is configured from the inspector, indicating which function or functions of which scripts are the ones triggered by the *Activator*. This has been decided to do so for convenience, since in this way when interacting with an object, the script that is activated is always the same, the *Activator* that is in that object. Then the *Activator* is already in charge of communicating with the rest of scripts and indicated functions.

Puzzles Scripts

An important part of the project are physical puzzles. For this have been necessary several scripts explained below.

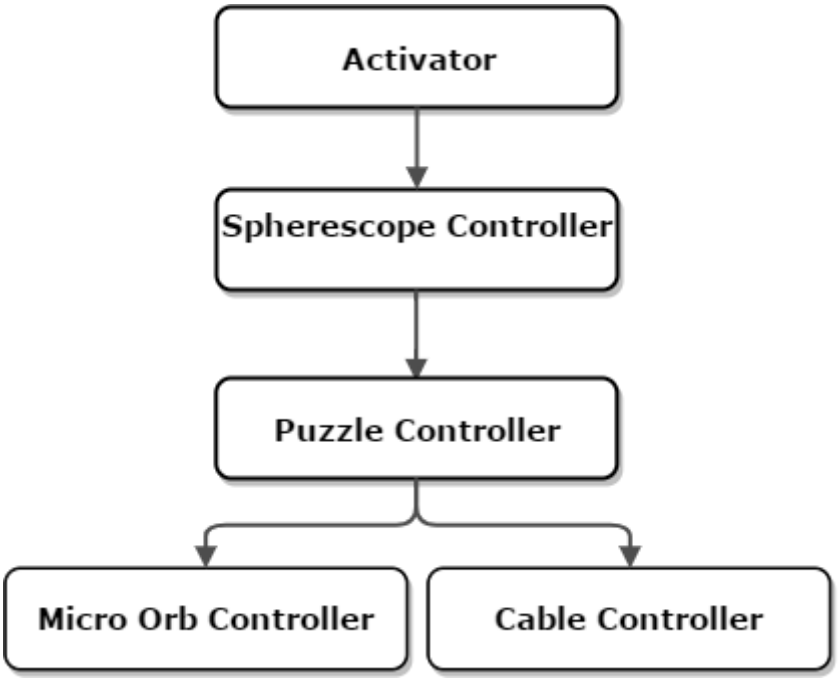

Figure 63: Puzzle Scripts flow chart

Puzzle Controller

This script is responsible for controlling each of the puzzles in the game. Since it has many functions and variables, it is going to be explained without going into much detail. The script is programmed so that it can be as customizable as possible through all the public variables it has. You can configure all sorts of options like what kind of puzzle it is (normal or inverted), how many orbs there are and what they are, what is the next puzzle, what elements are illuminated, what state it is in (available, solved, deactivated), etc. It has functions that allow to check in which state the puzzle is, if different requirements are met to change from one state to another, if it is necessary to reset the puzzle, etc. Apart from this it is important to mention some details to understand how the puzzles work.

The puzzles are connected sequentially so that when one is solved, the next becomes available. These sequences are also defined with a cable that runs through all the puzzles connecting them and that is illuminating. This cable is controlled by the *CableController* script. On the other hand, each puzzle has a series of micro orbs inside and you have to get them rotating the superplexus. These orbs are controlled with another script, called *MicroOrbController*. Finally, the rotating element is controlled with a last script, called *SpherescopeController*. All these scripts are called and consulted by the PuzzleController to be able to manage correctly everything that happens in each puzzle. Each of them is explained below.

Spherescope Controller

It has been decided to call spherescope to those elements with which the player can interact and rotate them freely in the axes X and Y. This is used mainly to handle the spheres of the puzzles. This script takes care of it. The W and S keys rotate the object on the X axis, and the A and D keys rotate it on the Y axis. It also has many public variables that can be configured in the inspector and allow you to customize the type of rotation. For example, you can limit the rotation to the Y axis only, with a maximum of 90º (both ways) and with a speed multiplier of x2.

Cable Controller

The only purpose of this script is to check the wiring status of the map and know when it should turn on. To do this you have two public functions *Activate()* and *Deactivate()* that are called from the *PuzzleController* and are responsible for turning the cables on and off respectively. What it does is to change the material of the cable to a material with neon effect to give the feeling that it ignites. For each section of cable a *CableController* is necessary and all the cables are children of a same *Gameobject* for comfort.

Micro Orb Controller

This script has all the micro orbs found in each of the puzzles. It serves to configure the material and the value of the micro orb, as well as to control when it leaves the puzzle and with which exit does it (red or blue). Micro orbs can be blue or red. The blues have a value of 1 and the red ones have a value of -1. The same applies for the two types of exits. When an orb crosses an exit ring, both color values are multiplied, that is, the value of the orb by the value of the exit. This means that if for example a blue orb (value 1) exits a red ring (value -1), the result is 1 x -1 = -1. In normal puzzles, the results should always be positive, and in

inverted puzzles, they should always be negative. These values are read by the *PuzzleController* to be able to identify if the puzzle is solved correctly or not.

Audio Scripts

Another very important part is the sound as previously explained. Here are some scripts that have been programmed to manage the audio.

Background Music Controller

This script lets you define what music plays in the background inside an area that is defined by a collider that acts as a trigger. When the FPC is inside the collider, the defined music plays. When you leave the area, the music stops.

High Volume

This script controls the volume of the *AudioSource* that is in the same *GameObject*. For this, the FPC position in the Y coordinate is taken into account. Depending on this value, the sound sounds at a higher or lower volume. You can define the minimum and maximum value of Y for which the sound plays at the minimum and the maximum volume respectively. It also allows you to reverse this calculation with a public boolean called *IsInverted*. This script is used for ambient sounds of waves and wind, as well as for the slight music that sounds in the lighthouse.

Physical Sound

This script is responsible for simulating two types of physical sounds for orbs.

On the one hand the sound of bounces and collisions. For this it has been necessary to take into account when a collision happens and at what speed it do it. Having this information, one sound can be reproduced for each collision and can be reproduced at different volumes and different pitches depending on the speed with which the crash occurred. The faster the hit, the higher and pitcher it sounds.

On the other hand the sound of sliding or rolling. For this it has been necessary to take into account the speed and angular velocity of the orb, as well as if it is in contact with some surface or not. Taking this data into account, a clinking sound has been produced when the orb rotates on a surface. This sound is repeated to a greater or lesser frequency and pitch taking into account the speed of the orb. The faster it spins, the more frequent and pitcher the clinking is.

Debug Scripts

Some debug scripts have been used to help with the development of the game. These scripts have not been developed from scratch but some changes have been made.

Take Screenshot

This script saves a screenshot in the project's root folder when the T key is pressed. A small modification has been programmed to be able to save them under different names. This is to add at the end of the base name the current date and time, so you can take as many captures as you want and will be saved in order by date. This script has been very useful for taking out captures of the project as it was progressing and in this way have a folder full of images of progress.

FPS Display

This script shows in the upper left corner of the camera the fps that the game is running. Although in the editor of Unity this functionality already exists, if an executable is mounted, the fps can not be shower, so this script has been used. In addition a small modification has been made so that if the fps are less than 30, the text is shown in red, and if it is greater than or equal to 30, it is shown in green. Thanks to this is quicker to identify if the game is running at more than 30 fps or not.

Menús Scripts

For the management of the menu and the different scenes it has also been necessary to program some scripts. Below are the highlight ones.

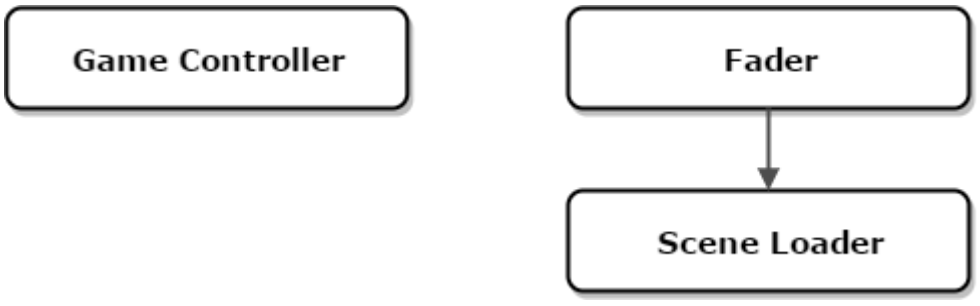

Figure 64: Menu Scripts flow chart

Fader

This script is responsible for displaying a transition on the screen that can be any given image. It has two functions, *FadeIn()* and *FadeOut()*. Both show the transition but one is input and one output. When the transition completes it calls the *SceneLoader* script.

Scene Loader

This script has a public function *LoadScene*() that given the index of the scene loads that scene. This has been used to move from the *Main Menu* to the *GameWorld*. This script is called by the *Fader* script, after completing a transition.

Game Control

Although the function to save and load game at the end has not been implemented in the demo, if it has been programmed a script that takes care of it. For this it is necessary to create as many variables as you want to save all the necessary information of the progress of a game. In general this is something simple but long and tedious so it has not finally been done. However something that is not so simple is to save and load the position and rotation of the FPC and the FPC camera. Therefore since this had more interest if it was done.

The first part was to store and load the position X, Y and Z coordinates. This part has been relatively simple since it only consists on replacing the position of the FPC with the position vector that has been saved.

On the other hand had to save and load the rotation of the camera so that it looks exactly at the same point, something that has been quite more complicated, especially because you have to work with rotations. For this, it has been necessary to investigate more in depth how the camera movement of an FPC works. An FPC is composed of two main elements. On the one hand a CharacterController that allows to move the object (FPC) with the corresponding keys of movement (WASD in this case). On the other hand a Camera, which can be rotated with the movement of the mouse. However this rotation is based on the movement of two axes. On the one hand the rotation on the X axis, which is applied to the FPC and on the other hand the rotation in the Y axis that applies only to the camera. Therefore, the X rotation of the FPC and the Y rotation of the camera must be saved. The *Save()* function is now complete and writes the data to a file that is saved externally. However the *Load()* function of the rotation has required a series of transformations and more advanced calculations with Quaternions and

eulerAngles that are not necessary to explain not to enter such level of detail. The *Load()* function is responsible for reading the saved data from the external file and loading it into the corresponding variables.

As a result, for the demo this script simply allows you to save and load the player's position and rotation, but not the progress of the game.

Effects Scripts

As already mentioned, a series of effects have been made and for this it has been necessary to program some scripts.

Position Lerper

It is a generic script that allows an object to move between two given positions. It has the *Activate()* and *Deactivate()* functions that allow you to move the object from its start point to its end point or vice versa. This has been used for example for the tidal effect, causing the sea plane to rise and fall continuously. To indicate that the effect has to occur indefinitely in a loop, the boolean *Loop is used*.

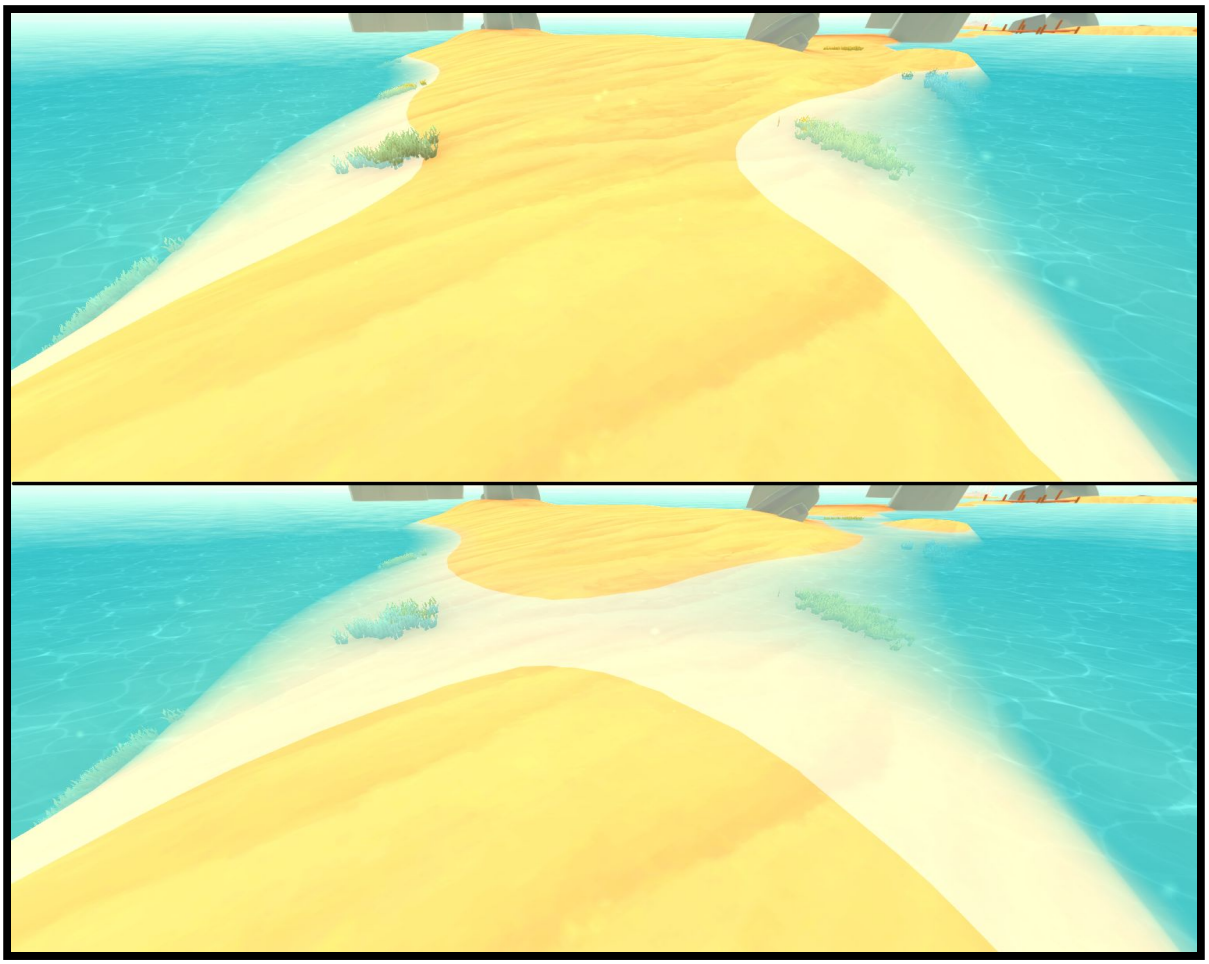

Figure 65: Low tide (above) and high tide (below) comparison

This script has also been used for many other movements such as the doors. With the *Activate()* function the doors are opened, and with the *Deactivate()* function, they close.

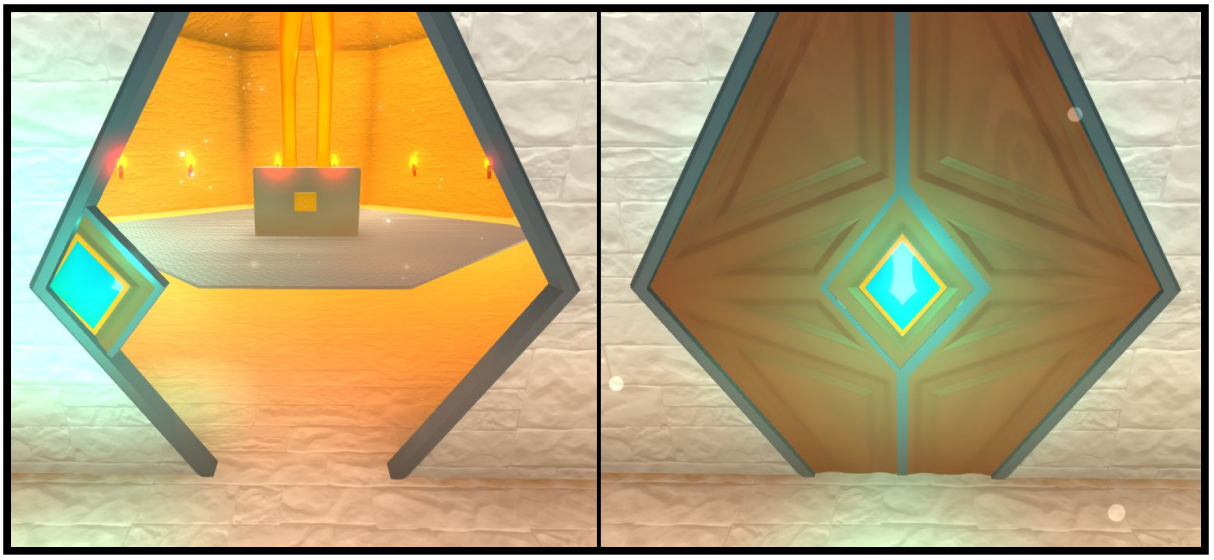

Figure 66: Opened door (left) and closed door (right)

Synchronized Position

Placing this script in a *GameObject* causes that object to move synchronously with another object defined in the inspector. In addition you can indicate which axes should be synchronized and which ones should not. This is used for example for the seaplane, synchronized with the FPC in X and Z but not in Y. In this way however much the player moves, the sea will remain, giving the impression that it is infinite. Since the shader applied to the water uses textures that are applied in world coordinates, however much the plane changes position the textures are maintained in their global position so it is not noticed that the sea is moving in X and Z .A part of this has also been used for the foam particles that are produced when the FPC steps on water. These are synchronized in X and Z with the FPC and in Y with the seaplane.

Effects

This script was originally intended to contain different functions that serve to cause different effects, hence the generic name of *Effects*. However, it finally only includes one effect. This effect is used in the main menu and causes the *LightShaft* that acts as sunbeams to change his intensity between two values given in the inspector.

Shaft Adapter

This script has a similar functionality to the *Effects* script, but in this case the intensity of the *LightShaft* does not change with time, but it changes according to the zone where the FPC is located. To do this a collider is used as a trigger that defines a specific area. This is used for the area of the lighthouse, in which it has been decided that the rays of sun are more intense for aesthetic reasons. When leaving the zone of the lighthouse the intensity of the *LightShaft* low gradually until reaching its base value, defined in the own inspector.

Rotator

With this script you can define that an element rotate in one or several specific axes and at the established speed and direction. It has a public function called *Stop()* that when called, causes the object to slowly slow down until it stops in the same rotation that it was initially in. This is used for example for the three rotating rings of the Gyro Temple, which stop when the three keys are entered. Each key calls the *Stop()* function of each of the rings. The *Activator* script explained above is used as the intermediary.

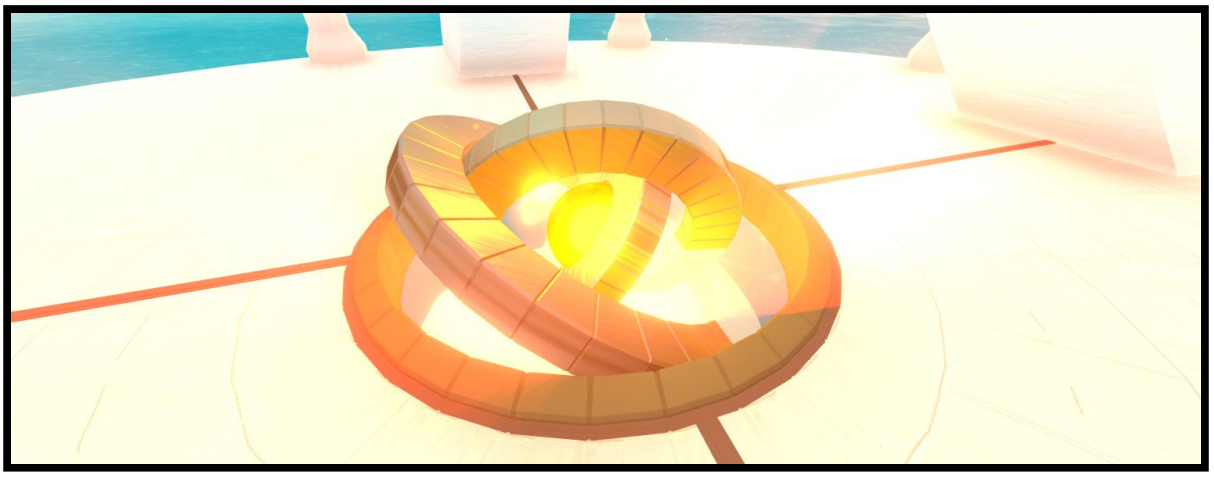

Figure 67: Rotating rings from the Gyro Temple

Mesh Fader

This script allows an object that has the *Mesh Renderer* element (a visible geometry with an applied material), to gradually fade with the pass of time. It also changes size as it fades. You can configure all these aspects from the inspector, such as fade speed, scale multiplier for resizing and the object material. In order to be able to produce the effect of fading it is necessary that the material used is of *Fade* type, since these materials allow to change its opacity. This decreases the opacity until the object becomes completely invisible. When this occurs it simply is destroyed. This effect appears when collecting the keys of the pedestals in which they are located. As geometry simply a sphere has been used, but any other geometric models can be used.

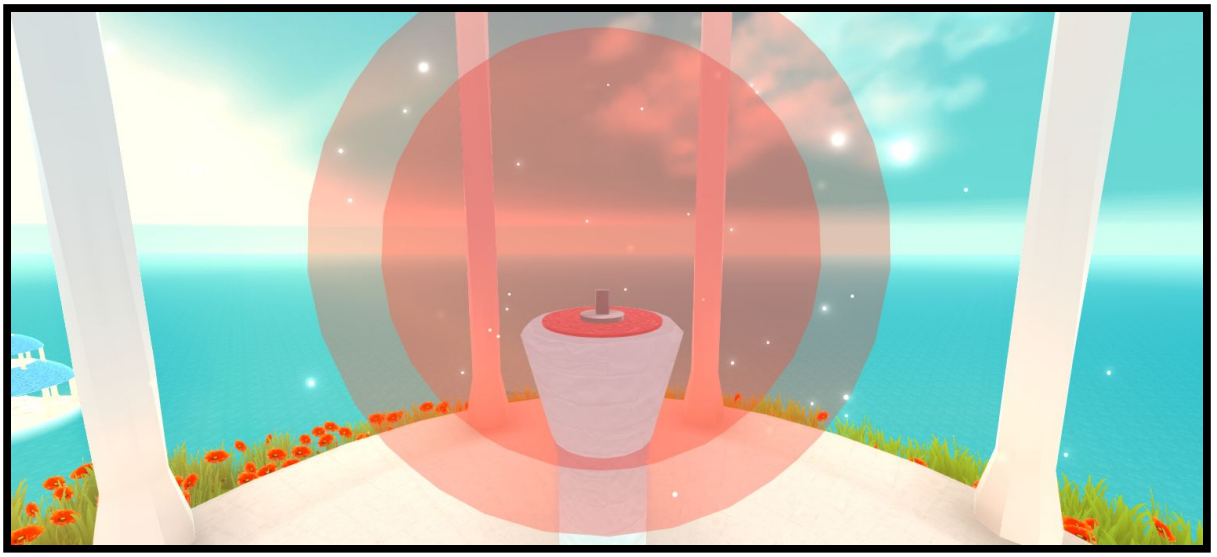

Figure 68: Mesh Fader effect when taking the red key

Optimizations

In order to make the demo performed smooth and playable comfortably, many optimization processes have had to be carried out. Some of them have already been commented a bit in previous sections, but then they are all collected and some more.

Baking

Real time illumination is a fps killer because of the many shadow casters a scene can have. Moreover, in this game there is no need for real time lighting so baking the scene is a good option.

It has been tested to make bakes with different options that affect the speed of the bake and the quality of the textures that are calculated. Even so, with options almost to the minimum, if the scene contains many objects, the bake can take hours, especially if they are very big models like the lighthouse. The options that have been decided to use for the final bake are shown in Figure 69. It is a balance between quality and time.

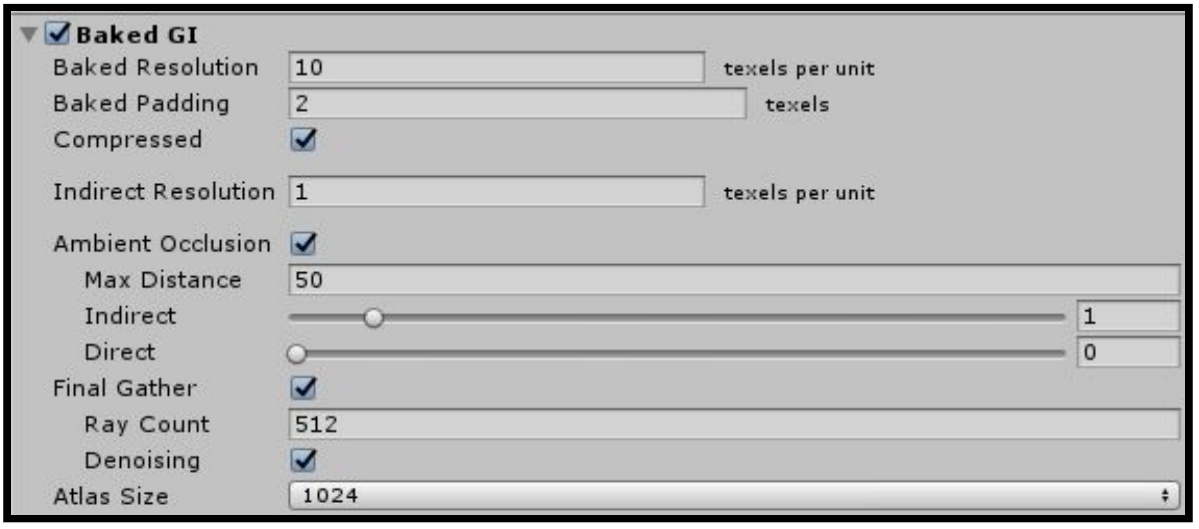

Figure 69: Baked GI options

To increase the final bake quality, the *Ambient Occlusion* and *Final Gather* boxes have been marked. On the one hand, there is the *Ambient Occlusion*, which calculates which areas are difficult to access (by proximity of geometry) and darken them, taking into account the value indicated in the Indirect option. On the other hand is the *Final Gather*, which is responsible for calculating bounces of light pigmenting the scene. This gives a more colorful look to the scene and makes it more immersive.

In addition to this, for an object to be included in the bake calculation it is important to check a couple of options. On one side is the *Static* box. From the object inspector, on the top right there is an arrow next to the word Static, as shown in Figure 70. Clicking on it opens a drop-down in which the *Lightmap Static* option appears. This option must be selected so that the object is included in the bake.

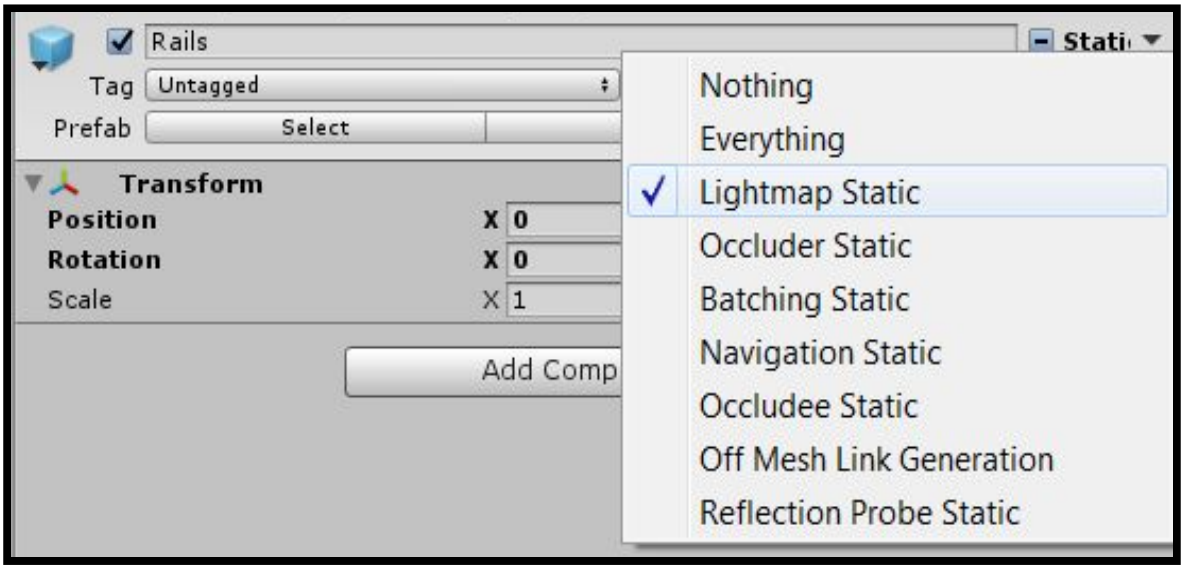

Figure 70: Lightmap Static option

On the other hand there is another option that is chosen from the maya's own geometry. This option is *Generate Lightmap UVs*. Checking this option gives permission to generate UVs of illumination for all those elements that have this model.

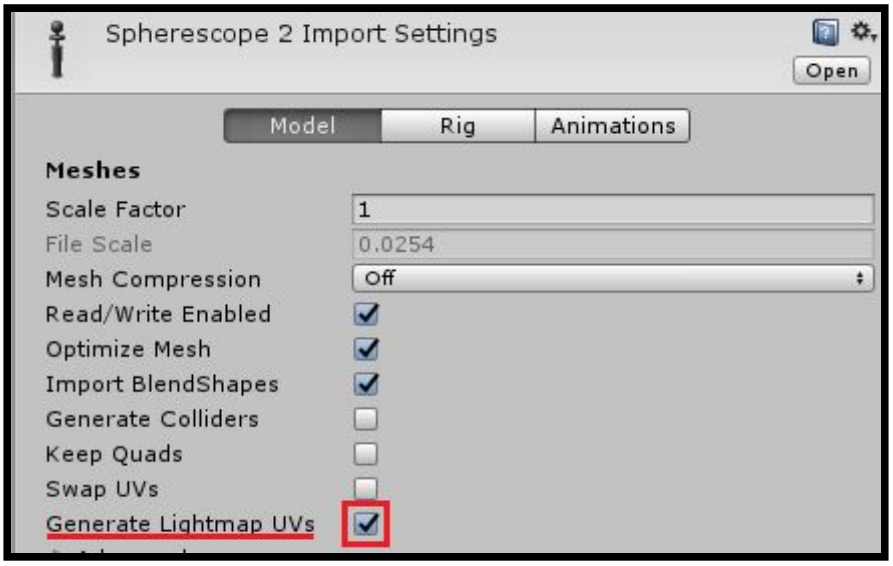

Figure 71: Generate Lightmap UVs option

Occlusion Culling

Another method used to optimize the game is the *Occlusion Culling*. This method is responsible for rendering only what is visible at each moment. Therefore, if for example the player is inside the lighthouse, nothing that is outside of the lighthouse is rendered even though it is inside the *Frustum* of the camera. To carry out this method simply mark the *Occluder Static* and *Occludee Static* options on those static objects that will not be moved, as shown in Figure 72.

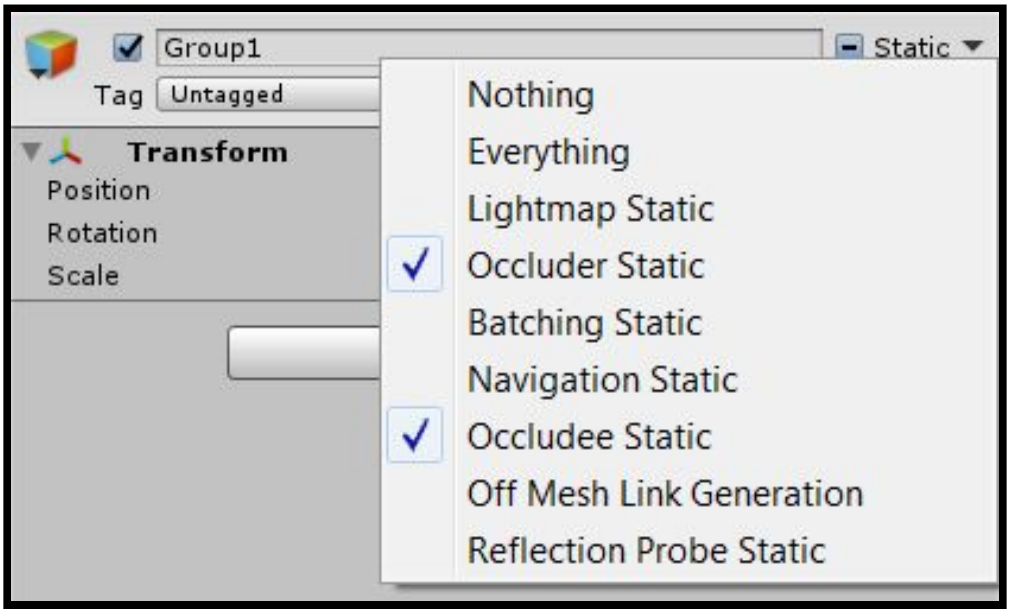

Figure 72: Occluder and Occludee Static options

For more accuracy, only the corresponding occlusion option can be marked for each object. The *Occluder Static* option is marked on large objects that potentially cover other smaller objects. The *Occludee Static* option is marked on those small objects that are likely to be covered by larger ones. However, this project has not been so accurate. Marking both options on both objects ensures that an object can act as both Occluder and Occludee, but slows down the occlusion bake process. Figure 73 shows a screenshot of the final scene occlusion bake.

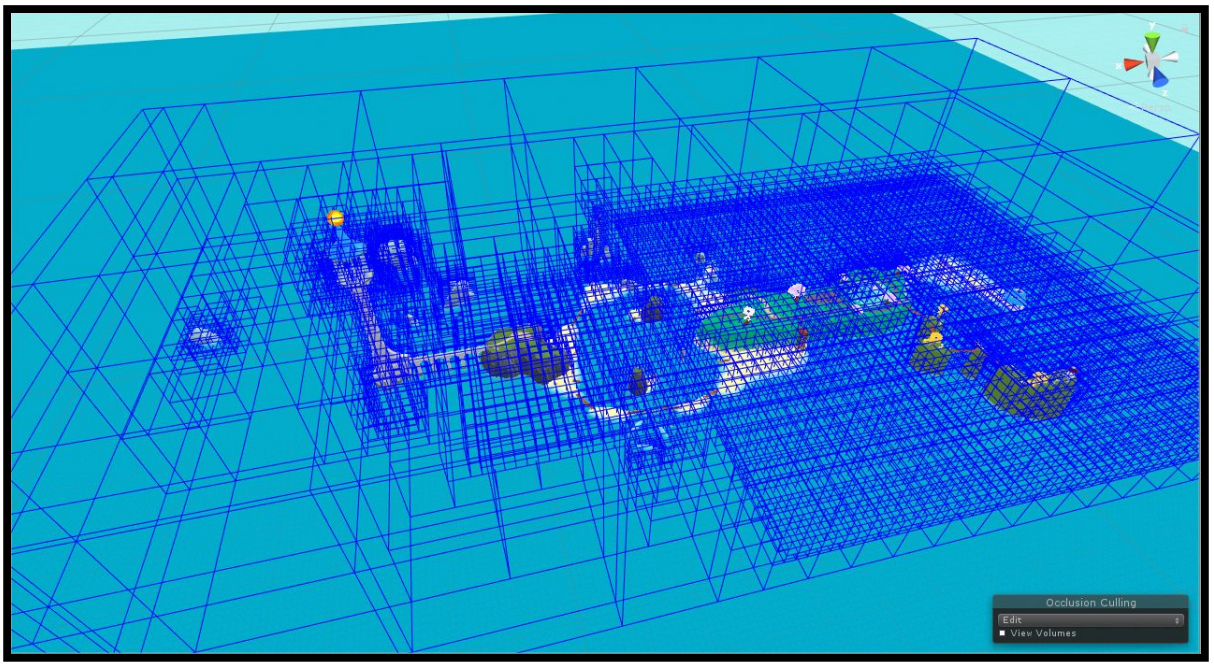

Figure 73: Baking Occlusion process

Post Processing Stack

Due to the use of different assets the final fps obtained are around 15 and it would be appropriate to obtain at least 30, so there had to find some alternatives. Looking for the asset store has found an asset called *Post Processing Stack* [10], which allows to use the effects of camera in a much more optimized way. These effects include *Antialiasing* and *Ambient Occlusion*, which are the two main effects that have been chosen to use to make the scenario look good. By being optimized you can also use more of these effects without reducing the fps, as is the case of *Bloom*. Thanks to *Bloom* you can achieve the same neon effect that is achieved with the asset *MK Glow Free*, but in a more optimized way, so that the *MK Glow Free* can already be discarded. To make the effect work correctly, you must activate the HDR rendering option in the camera itself and make sure that the elements that have to have a neon effect have materials with the Emission property activated with a value greater than 1 in HDR. Shown in Figure 74.

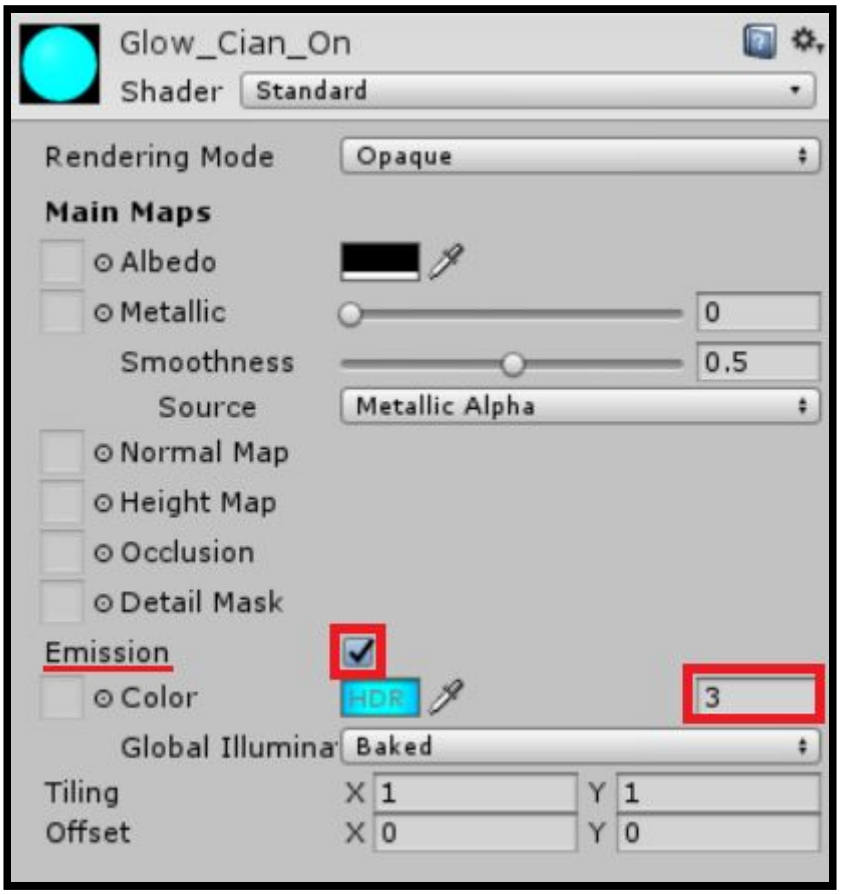

Figure 74: Material example with HDR emission of 3

Results

The final result is the desired one, a playable demo that contains on the one hand a three-dimensional scenario freely explorable and pleasing to the eye and on the other hand a series of puzzles based on the physical of spheres. A series of captures of the final project will be shown below. This will explain the game flow.

The player begins the adventure in a bedroom room, inside the lighthouse. In this room lives the protagonist Kodo, who is in charge of the care of this lighthouse. In Figure 75 you can see the room with all its furniture. This room has been made octagonal by aesthetics since it is an out of the ordinary.

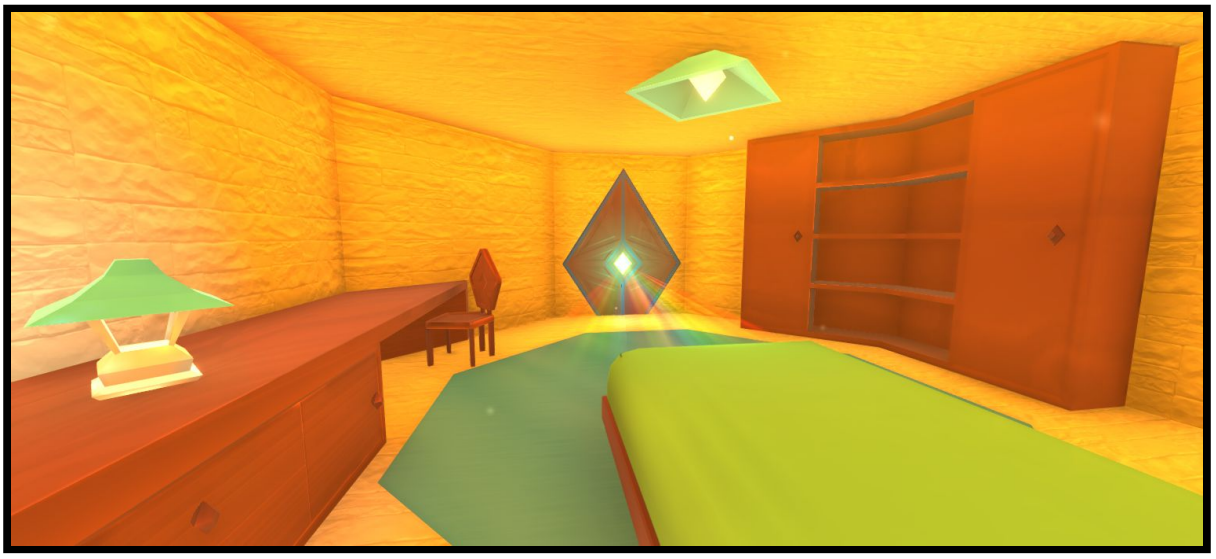

Figure 75: Bedroom

After leaving the room you reach the main room of the lighthouse, which is lit by a series of torches on the wall, impregnating the entire room with an orange light. In the center is a golden statue that works as an elevator as shown in Figure 76. At the base of the statue there is a button that drives the elevator and allows the player to climb to the top of the lighthouse. In the hands of the statue is where you have to get to place the orb that is in the Gyro Temple, although the demo does not reach this point. However, the final part is programmed when the orb is already in the statute.

Figure 76: Statue elevator

After leaving the lighthouse you can observe how the sun dazzles and sneaks between the columns. At the bottom there is a large, fairly long bridge that communicates with the island, as shown in Figure 77.

Figure 77: Lighthouse outside and bridge

When arriving at the island the first thing that catches the attention is an area to the right with a fireplace, in which is the map of the island. After picking it up, you can start searching for the three keys. This is only a passing area of the island, to communicate the lighthouse with the rest of areas. In Figure 78 you can see what this zone looks like.

Figure 78: First zone of the island

Continuing it is reached a descent that leads to a zone of coast, with sand and some palm trees as well as big rocks as shown in Figure 79. In addition this zone is communicated by different wooden bridges.

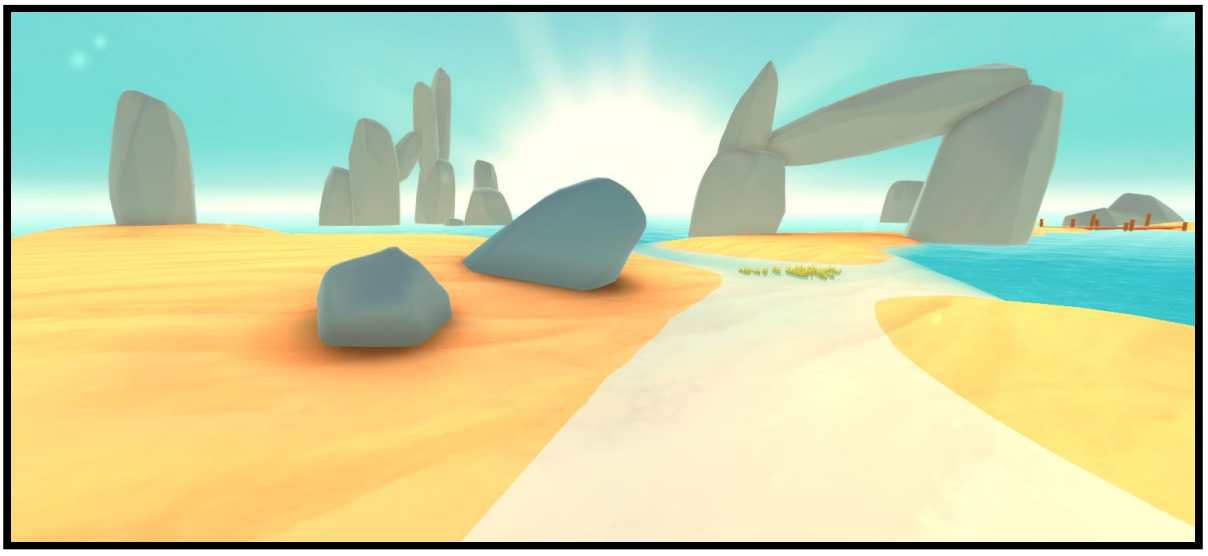

Figure 79: Beach zone

In this area is found the first key, the blue one, which can be easily seen and inevitably catches the player's attention, so if the map is not consulted, it is also very easy to find. This key just like the others is located on a pedestal under a small dome as shown in Figure 80.

Figure 80: First key zone

After picking up the first key and continuing along the coast you reach an ascent that leads to another area, in which the blue and purple colors take on importance, giving off a mystical air. In this area there is a small pond in the center surrounded by trees and some rock formations. Also in this area grow flowers that expand in circles of different sizes. In the center of the pond is the second key, the green one, as shown in Figure 81.

Figure 81: Second key zone

Finally you reach the area where the last key is located. This area is formed by small elevations of land that are interconnected by hanging wooden bridges. In this area grow yellow trees and shrubs and, above all, many red poppies. In addition there are vestiges of some type of old construction, since there are columns and

great blocks of stone by all the zone, although not in very good condition. This zone can be seen in Figure 82.

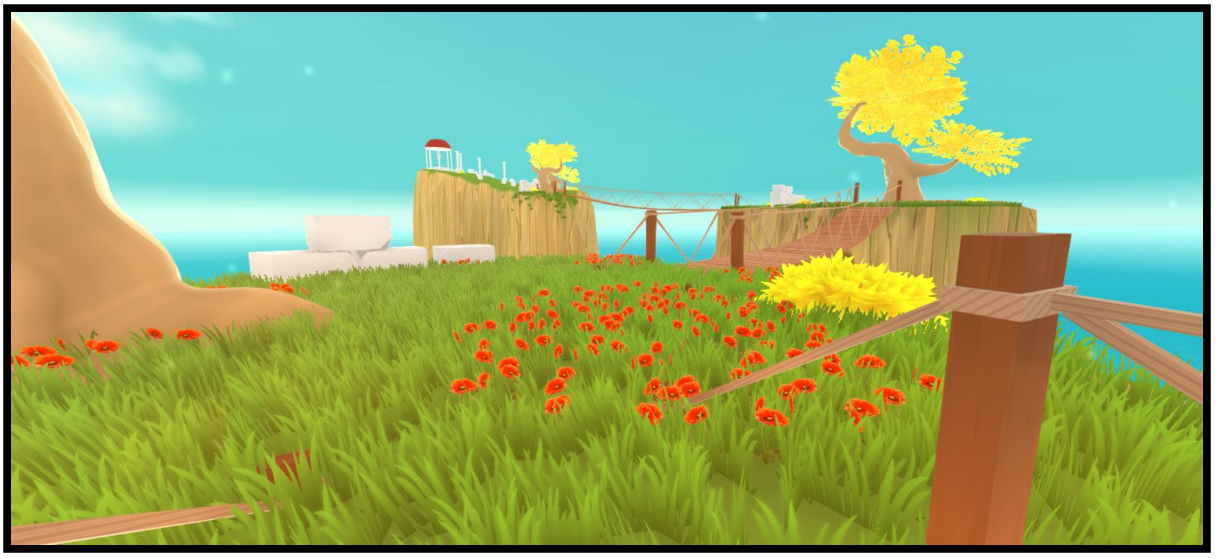

Figure 82: Hanging bridges zone

As you can see in Figure 83, at the end of this area, after climbing a hill full of poppies and columns on both sides, is the dome with the last key to be obtained, the red one.

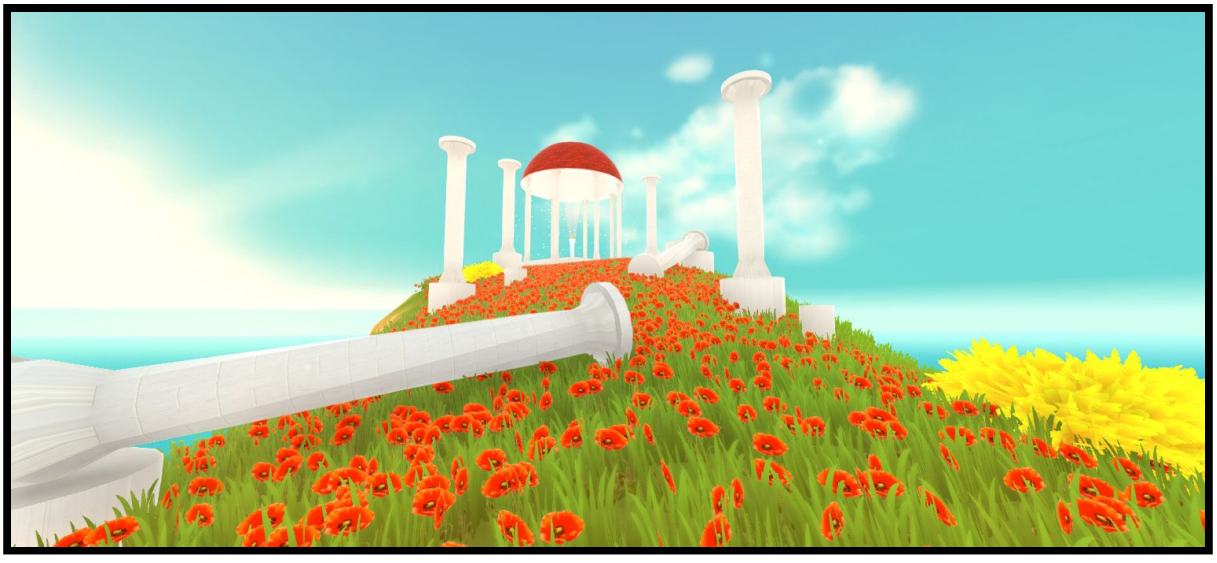

Figure 83: Third key zone

Finally, after having obtained the three keys, head to the Gyro Temple, which is also down to the coast. In the temple can be seen in the center three metal rings spinning at high speed and producing a deafening sound. In the center of the rings is the orb. On the sides of this element are three raised areas in which you have to

place the three keys that have been collected. The color of the light beam indicates which key goes on each pedestal. In this case, this is irrelevant since you have the three keys so inevitably will go one key on each pedestal. The Gyro Temple is shown in Figure 84.

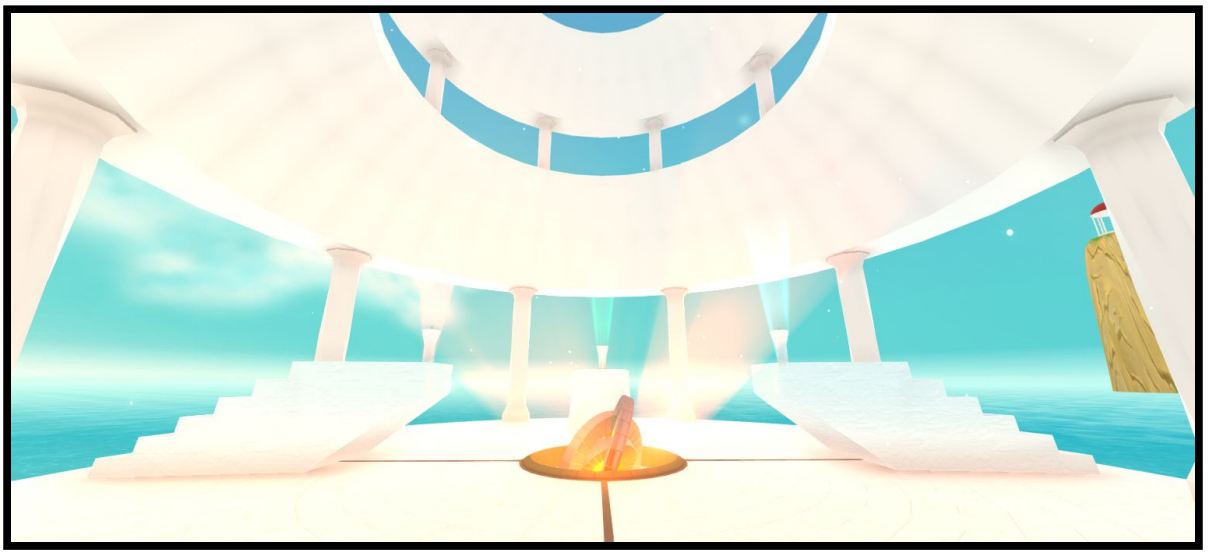

Figure 84: Gyro Temple

When interacting with the pedestals, the corresponding key is automatically placed in each one. Each key that is placed turns off one of the rotating rings. When the three rings stop, the orb is released by shooting upwards, where it reaches a rail by which it will begin to roll.

Here begins the second part of the demo, in which you have to solve the different puzzles to help the orb to advance. The demo only reaches the second zone of puzzles. The first zone of puzzles is located right next to the temple, in the same zone of coast, as can be seen in Figure 85.

Figure 85: First puzzles sequence

This first sequence of puzzles is formed by four puzzles, that when completed they allow the use of a catapult, in which the orb is after having rolled by the first rail. To activate the catapult simply press the button next to a pedestal. When activated, the orb is thrown into a kind of funnel, which directs the second and last rail of the demo. Figure 86 shows the catapult with the orb, as well as both rails and the glass funnel.

Figure 86: Orb in catapult

Following the way back and the path of the orb you get to the second and final sequence of puzzles made for the demo. This zone is activated after having used the catapult. In this sequence of puzzles the red micro orb is introduced as well as

its corresponding output. In addition also inverted puzzles are introduced, which are illuminated with a red light beam instead of blue, as can be seen in Figure 87.

Figure 87: Inverted puzzle waiting to be solved

After completing these puzzles could be terminated the demo, although there are more things than have been made for the way back to the lighthouse. After completing this zone would activate a wooden elevator, which allows to return to the lighthouse by a different road, in which supposedly would find more sequences of puzzles that would guide the orb. This elevator can be seen in Figure 88. This elevator is programmed and works perfectly.

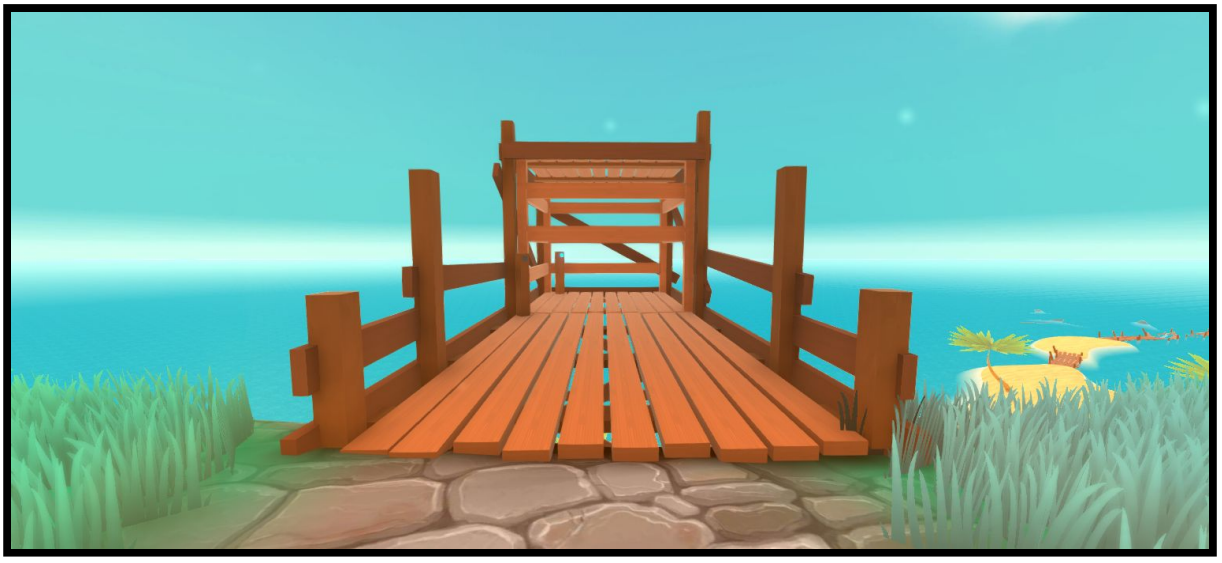

Figure 88: Outdoor wood elevator

Using the elevator you can go down to another zone of coast that communicates with the first with a wooden bridge. However this bridge is broken so that to return is made use of a boat. It is here that shows the small navigation system that has been implemented. You can see the broken bridge in Figure 89. As a curiosity have placed in that area a series of rocks in the sea that if seen from afar they look like three sharks swimming in circles. This tries to simulate that the bridge has been broken possibly by some shark.

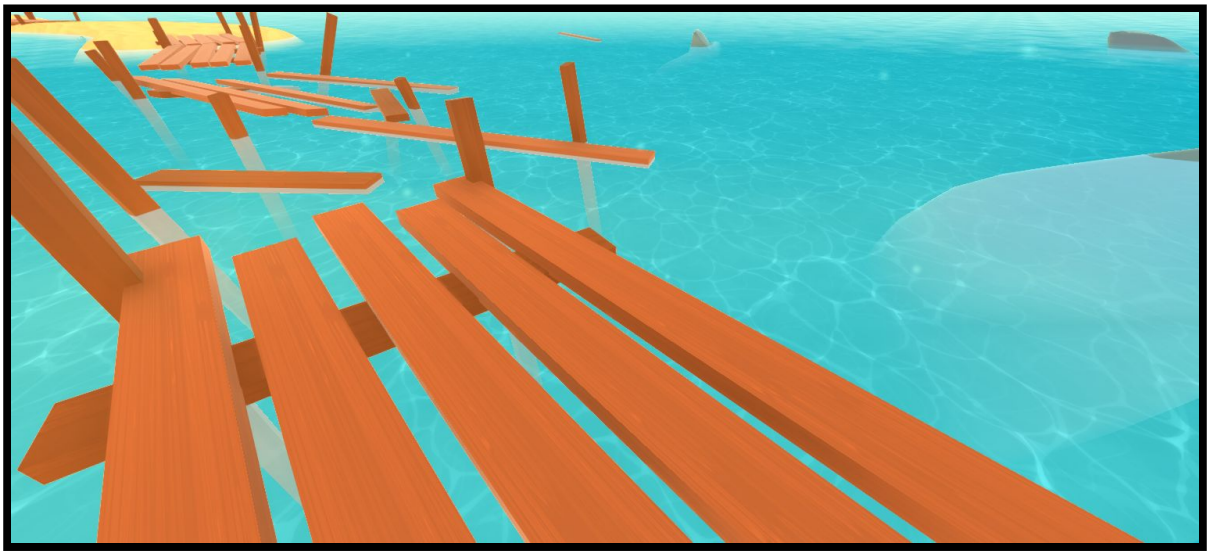

Figure 89: Broken wood bridge and shark rocks

On the way back you can see a nice view of the lighthouse and the bridge shown in Figure 90.

Figure 90: Lighthouse from the distance

Finally the final sequence is programmed in which the orb is placed in the statue and the elevator rises to the top of the lighthouse. When it arrives at the top a ray of light illuminates the orb and it begins to glow and diffract in different colored rays of light. Slowly the screen turns white and the game ends and closes. You can see this final scene in Figure 91.

Figure 91: Final scene with lighthouse being illuminated

Final Planning

Although all the initially planned tasks have been carried out correctly, the time estimates are far from reality. The project should be designed in a way that lasts approximately 300 hours. However, despite having planned it this way, it has finally had to spend much more time. Almost all tasks have taken twice the estimated time to complete satisfactorily. It has been wanted to do a varied project that requires tasks of all kinds that cover the greatest possible number of different subjects, but this has turned out to be negative because it has taken too much time to develop.

The initial planning was planned so that they were dedicated 4 hours a day, from Tuesday to Saturday. However it has ended up dedicating 5 or more hours, from Monday to Sunday. It was also planned that Saturdays were dedicated exclusively to advance the final memory, something that has been impossible to accomplish. The report was made once the project was finished at the end of May. Therefore, the entire month of June has been dedicated to writing the final report.

Conclusions

Objectives

The project has been successfully completed fulfilling all the stated objectives. Each one will be explained in detail below.

· **Obj 1:** Make a playable demo of a 3D videogame for PC in first person that will use the spheres physics as main mechanic and that will contain an free explorable scenario.

Despite considering this goal accomplished, it is true that a section of the demo was decided to skip because it consisted of repeating again the same process already performed (create more rails, more puzzles, more catapults, etc.) and time had to be spent on other more important tasks in the project.

· **Obj 2:** The demo will focus exclusively on one of the six proposed islands, but trying to show a variety of environments in it, including the navigation system.

In fact, only an island has been designed, modeled and assembled with its lighthouse and its corresponding temple. In addition, the island has been designed so that it is necessary to use the navigation system to complete it. This way, this part of the game can be displayed correctly as part of the demo.

· **Obj 3:** The scenarios of the game will be pleasing to the eye using a colorful aesthetic without textures and the modeling will be low-medium poly without excessive detail.

The final visual style obtained corresponds to the established, being colorful, with few textures and pleasing to the eye. As for the modeling perhaps some elements like the lighthouse could be considered something more than low-medium poly, but equally it has been well and does not disagree with the esthetics.

· **Obj 4:** The terrain, grass and tree tools included in Unity will be used to create the scenario.

All these tools have been used although in some of them it is not to be deepened as much as it is the case of the tool of creation of trees. As for the terrain tool and grass has explored practically all the options, even optimizing the rendering of the grass.

· **Obj 5:** It will be important that the game has a good sound environment.

Although the sound may sometimes go unnoticed, much attention has been paid to this and it has been elaborated in detail. It is for example the case of ambient waves and wind sounds that vary according to the height, or the case of the different sounds of steps that reproduce when player steps on different types of terrain like water, earth or even wood.

· **Obj 6:** The interface will be as clean and clear as possible.

The only element that appears as part of the interface is the cursor that allows the player to know how to interpret what type of element is looking at.

· **Obj 7:** The sound and interface should transmit good feedback to the player.

Indeed the interface has been designed so that the player receives different type of information. The interface is formed only by the cursor, which is responsible for visually and auditory indicating when you can interact with an element and when not, as well as knowing if it is interacting with something. The three cursor states take care of this function perfectly.

As for the sound, other than playing when interacting or looking at an interacting element, it is also found in other elements. In puzzles they can identify if they are completed correctly or not, and in the orbs, let you know by a clinking sound when the orb is spinning and if it goes fast or slow.

· **Obj 8:** The difficulty and learning curves will be well balanced, allowing the player to learn and understand the mechanics little by little without any kind of tutorial.

Both types of curve have been taken into account and have been elaborated carefully based even on curves used in other video games.

Problems and Solutions

Throughout the development of the project have been appearing all kinds of problems. Here are some of the most important ones.

Final Bake

Despite having previously prepared the bake and have done tests, for the final bake it has been necessary to change some options because due to the large size of the scene and the number of elements in it, the bake could not be performed. When trying to make the bake, not only took a lot of time, but was unable to finish and errors appeared as "Out of memory". To solve this, some options have had to be changed that lower the quality, as shown in Figure 92.

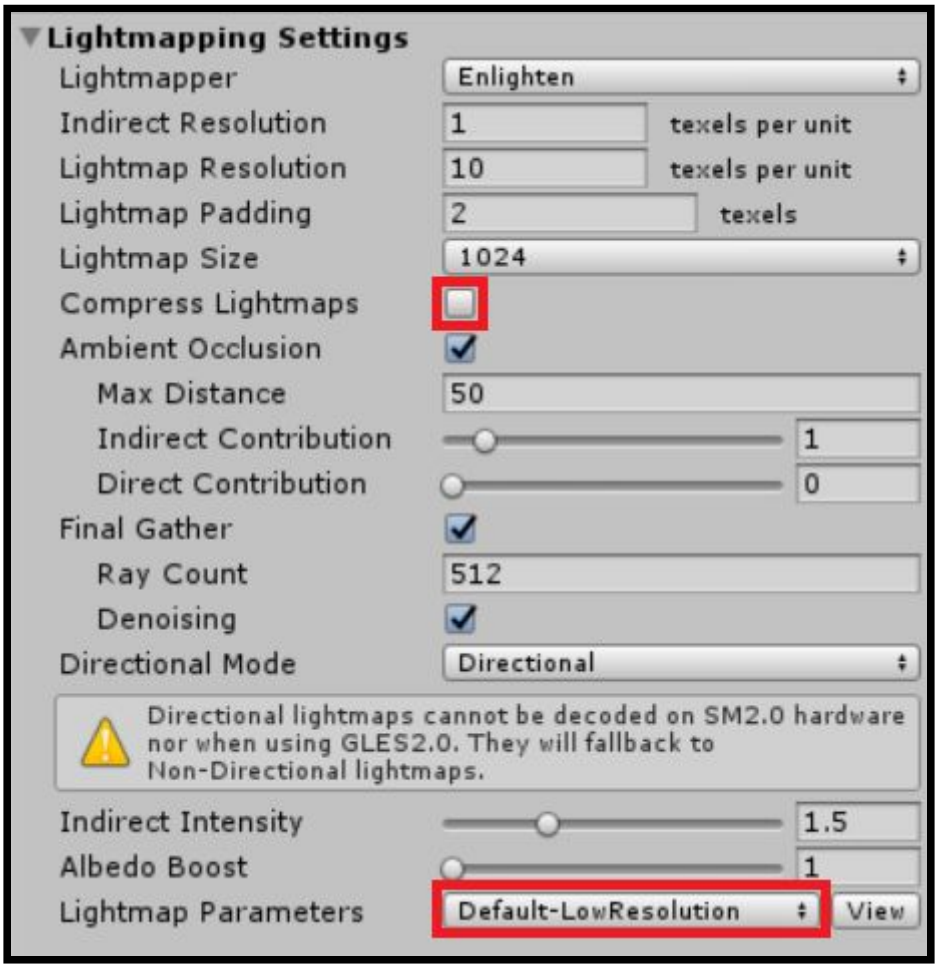

Figure 92: New bake settings

After modifying these bake options the bake has been able to realize with success, obtaining a good result. Even so, some small graphical glitches have appeared in

some textures as can be seen in Figure 93. This may be due perhaps to some vertex or badly closed edge in the geometry that has let the light through. It may also be simply some miscalculation, since the lighthouse is a fairly large model.

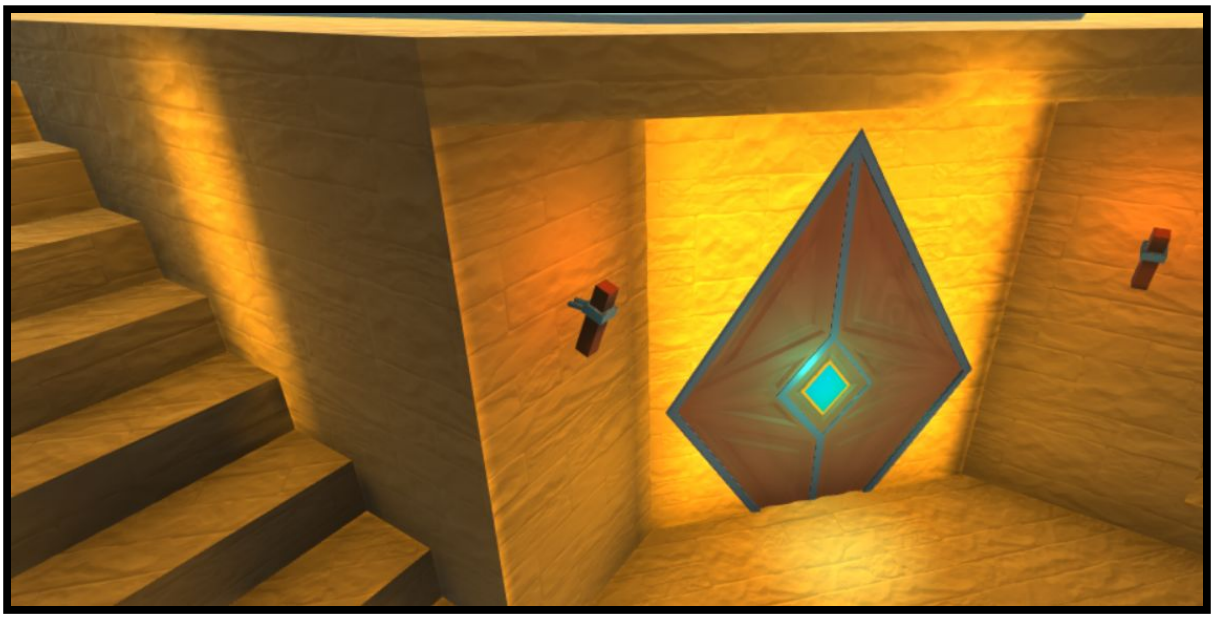

Figure 93: Lighthouse bake artifacts near the door and near the stairs

Anyway the result looks great and the final scene works at 45 fps on average.

Terrain

One of the biggest problems when modeling has been the proportions. Because the terrain has been modeled in the Unity editor itself, it was difficult to know the size and size of elements such as bridges. The ideal would be to model these elements in 3dsmax, however to avoid all these problems of proportions have been looked for an alternative. This consists of building these elements directly on the ground in the Unity editor, using basic geometric shapes such as scaled cubes and cylinders. This is a rather tedious process but the result has been very good. This method has been used for elements such as bridges, docks, the external elevator and all the wiring of the stage.

Another problem that also arises from the fact of using the terrain tool of Unity, is the limitations of this. As previously discussed, this tool uses heightmaps to model terrain. Therefore, structures like the one shown in Figure 94 can not be generated.

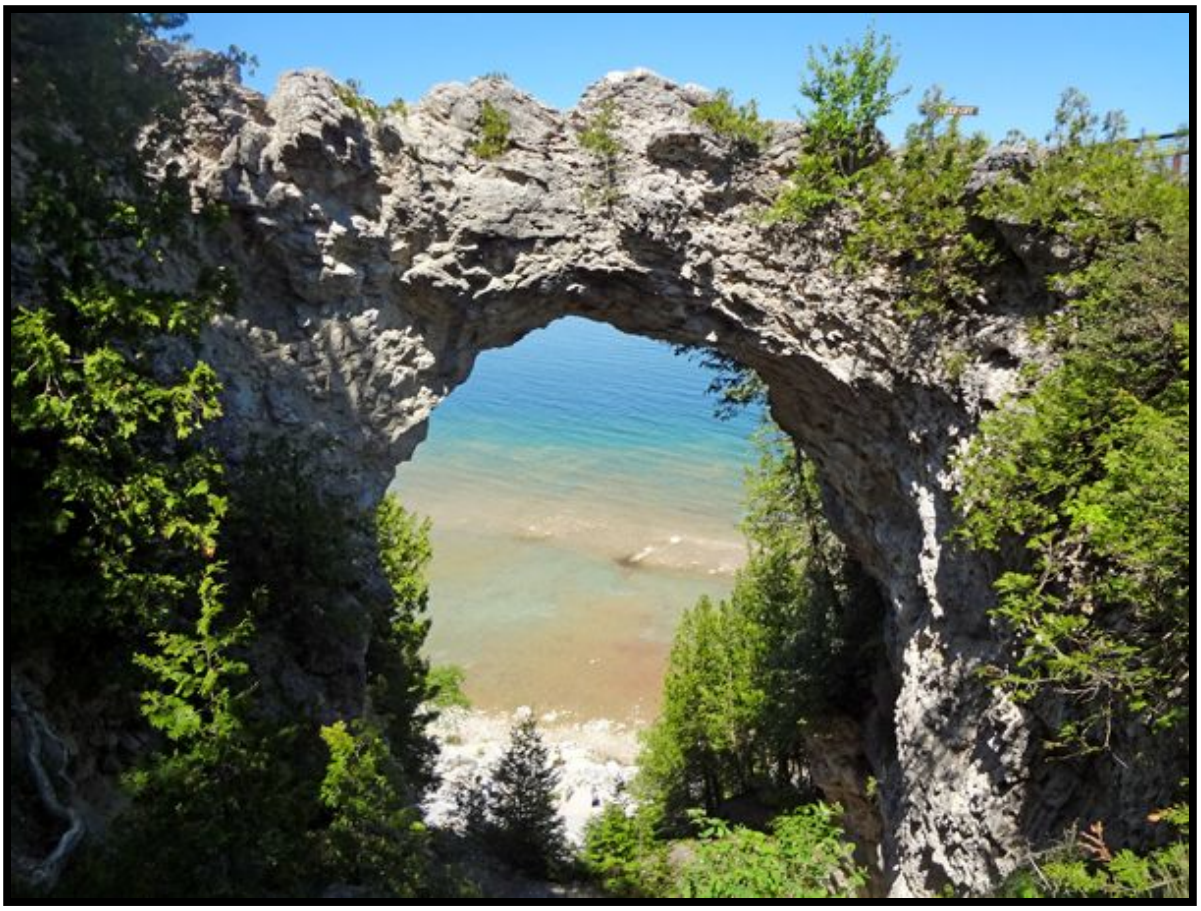

Figure 94: Rock arch reference

In a zone of the game it was proposed to create arches of stone to connect different islands, but with this tool can not do this. So what was finally done was to connect these islands with hanging wooden bridges.

Bibliography

References

[1] Marble Machines:
https://www.youtube.com/watch?v=3NJ7Fr6VrPU

[2] Superplexus
https://www.youtube.com/watch?v=9wohFfpLU7s

[3] The Witness
https://www.youtube.com/watch?v=SPMMKFX78x0

[4] Rime
https://youtu.be/PLszT6nHeAo

Documentation

[5] Terrain Settings
https://docs.unity3d.com/Manual/terrain-OtherSettings.html

Assets

[6] Cel Shading:
http://www.zehngames.com/developers/next-gen-cel-shading-con-unity-5/

[7] MK Glow Free:
https://www.assetstore.unity3d.com/en/#!/content/28044

[8] Light Shafts:
https://github.com/robertcupisz/LightShafts

[9] Volumetric Lights:
https://github.com/SlightlyMad/VolumetricLights/

[10] Post Processing Stack:
https://www.assetstore.unity3d.com/en/#!/content/83912

Tools

[11] MP3 Cutter Online:
http://mp3cut.net/es/