Development of a board game in a 3D virtual environment

Alicia Muñoz López

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

July 3, 2023

Supervised by: Ignacio Miralles Tena

To all the people who have walked with me on this journey.

Acknowledgments

The journey I have traveled over these four years has been a path filled with great moments. When I began this career, I wasn't quite sure what I was getting myself into. I was moving to a city where I didn't know anyone, far from home, and I wasn't sure if I wanted to study this degree. Animation and 3D modeling had always intrigued me, but video games not so much. However, as time went on, I realized that I also enjoyed programming and that this was the career I wanted to pursue.

Throughout this journey, I have been accompanied by what are now my friends. They have always helped me whenever I needed it and we have shared many good times together. They are also a big part of the reason why I have come to enjoy this major, as they gradually instilled their passion for video games in me.

I especially want to thank María. I started my first day of university with her, and now we have reached the last day together. She has been very important during these four years. Whenever the path became dark, she would light it up for me, and because of that and because we have grown together over these years, she has become a very important person in my life.

I also want to thank Alberto, my partner. Although he started out only as my friend, he has gradually become my true support and my best friend. He has always been able to make me smile and helped me escape from my problems. And this year, more than ever, thank you. Thank you for believing in me even when I didn't, thank you for having so much patience with me during these past months, and thank you for giving me the strength that I often lacked to keep going.

As the years passed, our friendship grew stronger, and I know that part of the person I am today, which is completely different from who I was when I started this major, is thanks to the two of them.

Above all, I would like to thank my parents for helping me so much and supporting me unconditionally. And to my sister Nuria, who has always been and will always be my role model and the pillar of my life.

And lastly, I would like to thank my Final Degree Work supervisor, Ignacio Miralles Tena, for his effort to help me in all he can, and for showing interest during the entire process. And also thank Sergio Barrachina Mir and José Vte. Martí Avilés for their inspiring LaTeX template for writing the Final Degree Work report, which I have used as a starting point in writing this report.

Abstract

This document presents the technical report of the Final Degree Project "Implementation of mechanics and 3D environment in traditional board games". The main objective of this project is to focus on providing the audience with a fun and entertaining experience, whether it be with friends or family trough an island in which the players can choose their characters and embark on an adventurous journey.

In order to to meet these requirements, the development takes place in a three dimensional environment, and the mini games are developed in 2D to create contrast with the main game, and they are inspired by classic mini-games, but with my personal touch.

Contents

CHAPTER 1

INTRODUCTION

Contents

In this section the purpose is explained behind this video game and to share the motivations and ideas that inspired its development.

1.1 Work Motivation

From the very beginning, a clear idea of creating a 3D video game was present, because the art of modeling has always fascinated me and sparked my interest. Each advancement in my knowledge motivates me to continue exploring and improving my skills in 3D modeling. I feel excited about discovering new techniques and putting them to the test.

Specifically, this project follows the low-poly technique.This technique has always captured my attention because, although it may seem simple at first glance, it possesses great personality and a distinctive touch that greatly enriches the project and highlights the essence of the game. In addition the low poly technique enhances accessibility on low performance devices.

On the other hand,my video game is inspired by two iconic titles: Mario Party [28] and Wii Party [24]. These video games left an indelible mark on my childhood as I used to play them with my friends and family. They were always responsible for contagious laughter and moments of celebration. Even to this day, we occasionally gather to relive those moments and continue having fun.

Inspired by these cherished games, I finally decided to create my own virtual board game designed to be enjoyed with friends. I opted for a low poly 3D style that gives the environments and characters a charming and distinctive look. Additionally, I included a variety of 2D mini games to further enhance the fun and keep the players captivated. With this combination of elements, my intention is to provide a unique and enjoyable gaming experience.

1.2 Objectives

Based on the motivation of the work, there are the goals to achieve:

- Create a 3D environment.
- Program some 2D mini games.
- Design and model multiple low poly 3D objects.
- Create a captivating and entertaining experience that delights the player or players.
- Immerse the player or players in the virtual world, offering them an engaging game play experience.

1.3 Environment and Initial State

When I started brainstorming ideas for my final degree project, I had a clear intention in mind: to create a 3D video game. As I explored this idea, my mind opened up to new possibilities, such as transforming a 2D game into a three dimensional experience or bringing a tabletop game into the virtual 3D world.

After exploring all those initial ideas, I decided to choose the traditional game of Goose [35] (the traditional tabletop game) into the fascinating virtual world in three dimensions.

As the project progressed, I found myself drawing more inspiration from games like Mario Party [28] and Wii Party [24], two titles, as I mentioned before had a significant impact on my childhood. In that process, I realized that I was much more attracted to that style, so in the end, I decided to create a video game following that path, taking ideas from both of them. It was a choice that excited me deeply and allowed me to join the best of both worlds into my own game.

Chapter 2

Planning and resources evaluation

Contents

This chapter outlines how the project has been structured and planned, as well as the resources used to carry it out.

2.1 Planning

To ensure effective planning of the project, the decision was made to leverage several tools and programs that had been previously encountered in subjects like Software Engineering. During that course, knowledge was acquired about the use of organization and planning tools, such as Trello (see Figure 2.1) and the Gantt chart (see Figure 2.2), which proved to be valuable resources in managing the project.

Trello, an online project management platform with a highly intuitive interface, allowed to create interactive boards to organize and plan the different tasks of this project over time.

Additionally, the Gantt chart provided a comprehensive visualization of the project's timeline, enabling a clearer view of deadlines and facilitating better resource allocation. The different tasks depicted in this diagram are:

- **Create the environment:** This task began with the work of envisioning the shape of the board and then proceeding to model it. Once the board was modeled, the decoration work commenced, incorporating elements such as squares, trees, rocks, and more.

- **3D Models:** This task began with the decision to choose the low poly and cartoon style for the development of this project. Subsequently, it went through the creative process of designing and conceptualizing the appearance of the different characters. And this task concluded with the process of modeling each one of them.

- **3D Animations:** After completing the character modeling, I made the decision to personally create the animations. Since the ones I found online were too human for the characters. By creating them myself, I was able to give them a more personal and unique touch.

- **Program:** Initially, the main focus was on modeling rather than programming. However, as the project progressed, I started giving more importance to programming, dedicating even more hours to this task than to modeling itself.

- **2D Animation:** Continuing with the goal of providing a personalized touch to this project, I decided to take charge of the animations required for the mini games as well.

- **TFG memory:** I have also dedicated a significant amount of time to this section, describing in detail all the steps I have taken to carry out this project.

- **Bug fixed and optimization:** This step was crucial to achieve an optimal, efficient, and error free outcome.

- **Test the game:** I conducted tests with the aim of identifying potential errors (bugs) in order to fix them and also to improve aspects related to game play.

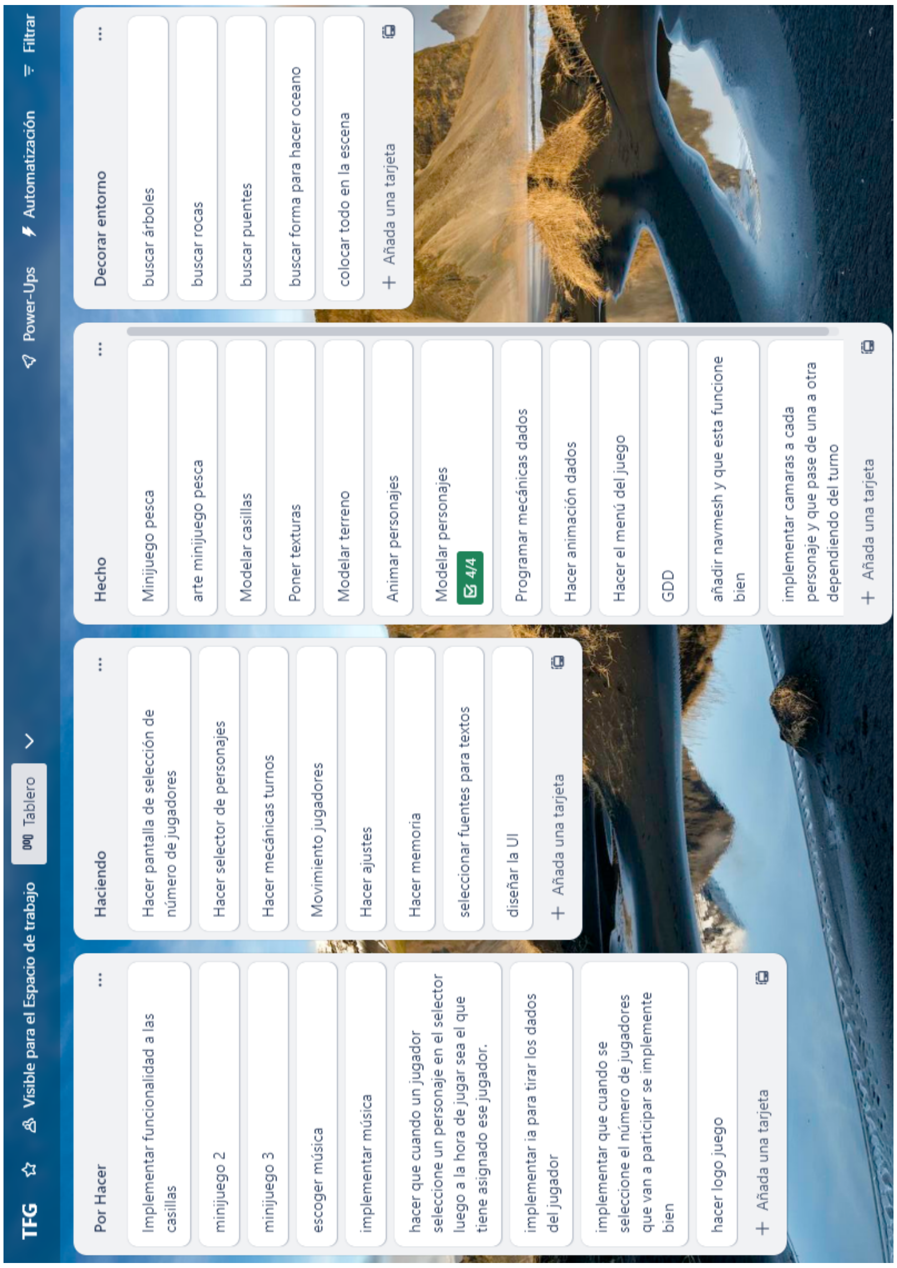
- **Mini games:** Mini games play a crucial role in this project, as they are the source of fun and challenge for the player. Therefore, I dedicated time to both the programming and artistic aspects of the mini-games, ensuring an enjoyable experience.

- **TFG presentation:** This task is also of great importance, as a good presentation is crucial to properly defend the project and explain it in a clear and concise manner.

In summary, the use of tools like Trello and the Gantt chart allowed to maintain greater control over the tasks in the project and achieve effective planning.

Also, at the beginning of the project, a table was created where the estimated time for completing the activities was documented 2.3. Now, another table has been generated to record the actual time spent on each of these activities. 2.4.

This comparison between the initial estimates and the actual times made it possible to assess the planning abilities and measure the progress throughout the project.



Por Hacer
- Implementar funcionalidad a las casillas
- minijuego 2
- minijuego 3
- escoger música
- implementar música
- hacer que cuando un jugador seleccione un personaje en el selector luego a la hora de jugar sea el que tiene asignado ese jugador.
- implementar ia para tirar los dados del jugador
- implementar que cuando se seleccione el número de jugadores que van a participar se implemente bien
- hacer logo juego
- + Añada una tarjeta

Haciendo
- Hacer pantalla de selección de número de jugadores
- Hacer selector de personajes
- Hacer mecánicas turnos
- Movimiento jugadores
- Hacer ajustes
- Hacer memoria
- seleccionar fuentes para textos
- diseñar la UI
- + Añada una tarjeta

Hecho
- Minijuego pesca
- arte minijuego pesca
- Modelar casillas
- Poner texturas
- Modelar terreno
- Animar personajes
- Modelar personajes 4/4
- Programar mecánicas dados
- Hacer animación dados
- Hacer el menú del juego
- GDD
- añadir navmesh y que esta funcione bien
- implementar camaras a cada personaje y que pase de una a otra dependiendo del turno
- + Añada una tarjeta

Decorar entorno
- buscar árboles
- buscar rocas
- buscar puentes
- buscar forma para hacer oceano
- colocar todo en la escena
- + Añada una tarjeta

Figure 2.1: Fragment of Trello document

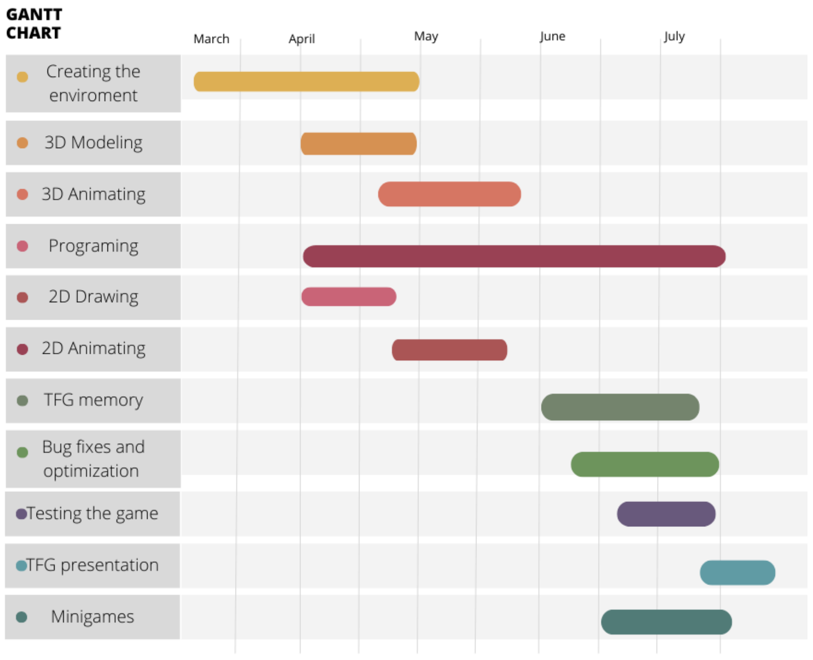
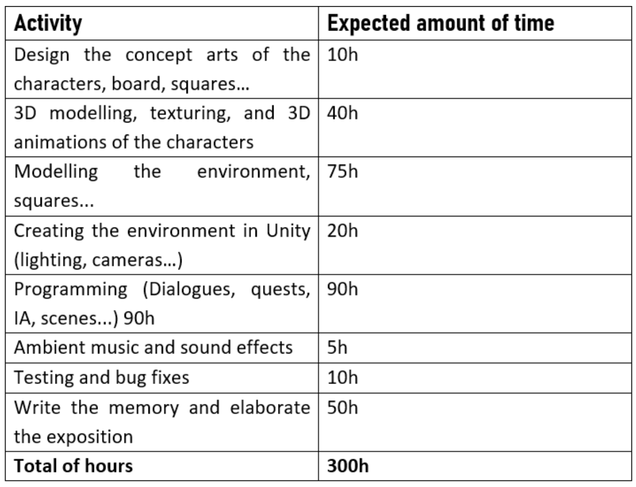
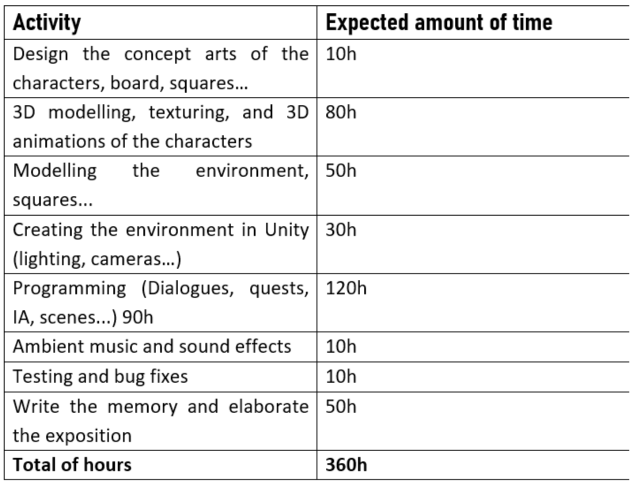


Figure 2.2: Gantt chart



Activity	Expected amount of time
Design the concept arts of the characters, board, squares...	10h
3D modelling, texturing, and 3D animations of the characters	40h
Modelling the environment, squares...	75h
Creating the environment in Unity (lighting, cameras...)	20h
Programming (Dialogues, quests, IA, scenes...) 90h	90h
Ambient music and sound effects	5h
Testing and bug fixes	10h
Write the memory and elaborate the exposition	50h
Total of hours	**300h**

Figure 2.3: First Planning



Activity	Expected amount of time
Design the concept arts of the characters, board, squares…	10h
3D modelling, texturing, and 3D animations of the characters	80h
Modelling the environment, squares…	50h
Creating the environment in Unity (lighting, cameras…)	30h
Programming (Dialogues, quests, IA, scenes…) 90h	120h
Ambient music and sound effects	10h
Testing and bug fixes	10h
Write the memory and elaborate the exposition	50h
Total of hours	**360h**

Figure 2.4: Final Planning

2.2 Resource Evaluation

To attain success in the execution of this project, a diverse range of tools and software programs have been employed. These tools have played a fundamental role in enabling efficient management and coordination of the different tasks involved. Additionally, the use of high-quality physical elements has been necessary to ensure a smooth workflow and provide an optimal working environment. By effectively leveraging these tools and resources, adeptly confronting the challenges presented by the project and achieving the desired results has been possible.

- **Software:**
  - Blender: Used to model all 3D objects and to put some textures.
  - Unity 2021.3.17f1 : Used to do the game.
  - Unity Hub 3.4.1 : Used to open unity project.
  - GitHub Desktop: Used to share the project.
  - Krita: Used to make 2D drawings.
  - Visual Studio Code: Used to programming the game.
  - Overleaf: Used to do the memory of Final Degree Project.
  - Trello: Used to organize the different tasks.
- **Hardware:**
  - Laptop: ASUS TUF Gaming FX505DV FX505DV.
  - Mouse: Logitech G305.
  - Headphones: Edifier W82ONB.

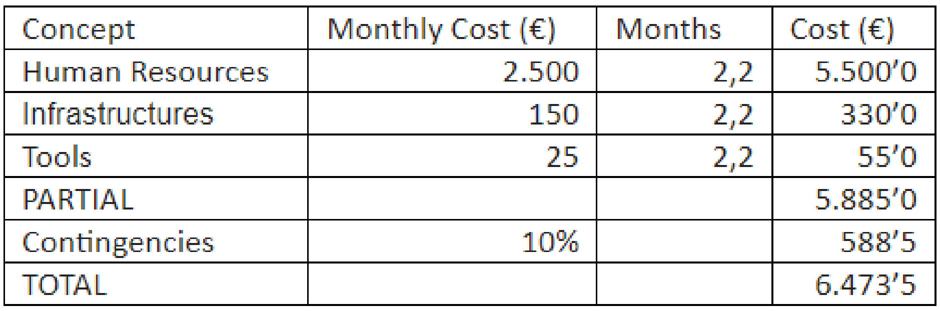
In the other hand in order to conduct a cost analysis for the development of a video game, three primary cost factors need to be taken into consideration: Human Resources, Infrastructures and Tools. Additionally, other potential secondary costs such as contingency measures, support, and internal training should be considered. When developing a TFG (Final Degree Project), an approximate calculation could be as follows:

Human Resources: Typically, this cost is measured in person/month. Assuming a total of 360 hours and a 37.5-hour workweek, we have 9.6 weeks. Considering that a month consists of approximately 4.4 weeks, the equivalent would be 2.2 person/month. If the average cost of a Junior developer is around €2,500 (including a salary of approximately €1,500), the cost of human resources would be 2.2 x €2,500 = €5,500.

Infrastructures: Adjusting the cost to the average price of a coworking space in the city where the TFG is being developed (Castellón), which is approximately €150 per month, the infrastructure cost would amount to approximately €330 (2.2 x €150). This cost includes expenses such as electricity, internet connection, and workspace.

Tools: The cost of tools should consider the depreciation of the development equipment over the course of 2.2 months. Assuming a useful life of approximately 5 years, the annual depreciation cost for a team with a value of €1,500 would be €300. Thus, the depreciation cost for 2.2 months would be approximately €55. It is worth noting that the cost of the software that is mentioned before licenses has been omitted, as they are free in their initial development versions.

Additionally, contingency costs have been considered, which are typically 10% of the total cost. Therefore, the following table provides an approximate breakdown of the cost analysis for the development of the game (See Figure:2.5).



Concept	Monthly Cost (€)	Months	Cost (€)
Human Resources	2.500	2,2	5.500’0
Infrastructures	150	2,2	330’0
Tools	25	2,2	55’0
PARTIAL			5.885’0
Contingencies	10%		588’5
TOTAL			6.473’5

Figure 2.5: Cost table

Chapter 3

System Analysis and Design

Contents

This chapter presents the requirements analysis, design and architecture of the proposed work, as well as, the interface design of the GUI.

3.1 Requirement Analysis

The video game created in this project falls into the category of "party game," and to enhance this categorization, the interface of this game is quite eye-catching and fun.

It starts with a main menu (Figure: 3.1) that displays the island where the game board is located as the background. There are three main buttons: start, options and exit. After clicking the start button, the game takes us to another screen where the number of players must be chosen, and then the game leads you to a character selector where each player can choose a character. Once all of this has been decided, the game begins.

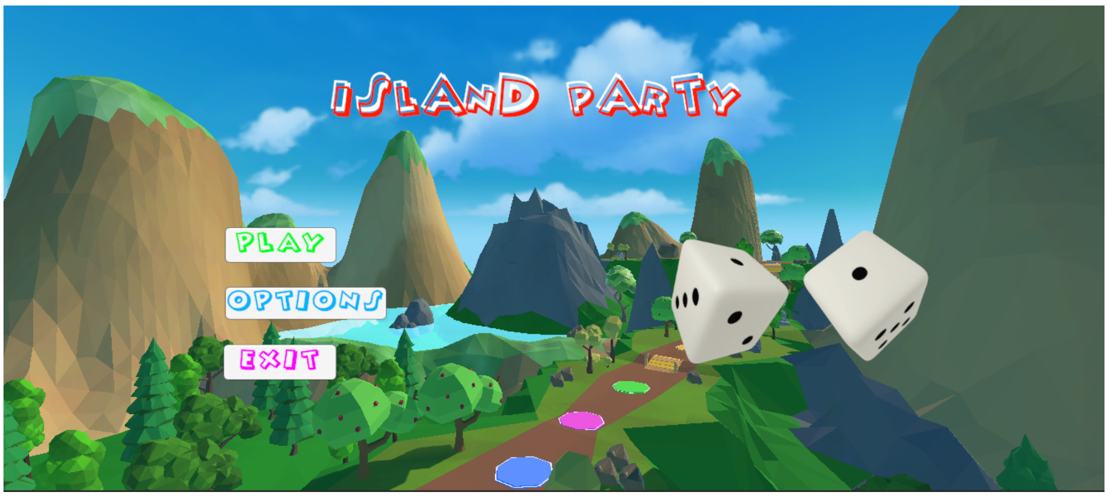


Figure 3.1: Main Menu

3.1.1 Functional Requirements

R.1. The player can start game pressing *Play* button.
R.2. The player can exit the game pressing *Exit* button.
R.3. The player can restart the game all the times that he wants.
R.4. The player can choose his character.
R.5. The player can adjust the volume of the game.
R.6. They can choose the number of participants who are going to play.
R.7. The player can enable full screen mode in the game.
R.8. The player can interact with the dice to roll it.
R.9. The player can play mini games.
R.10. The player can adjust the brightness of the game.

3.1.2 Non-functional Requirements

R.11. Mechanics should be easy to understand.
R.12. The game will be a 3D environment.
R.13. The mini games will be in 2D.
R.14. The controls should be accessible and easy to understand
R.15. The game will be available for PC.
R.16. The style will be cartoon and low poly fun and friendly.
R.17. The game will have a fun and friendly atmosphere.

3.2 System Design

This section present the logical and operational design of the system to be carried out. All the sequences of actions and decisions that the player has to do during the game (see Figure 3.2).

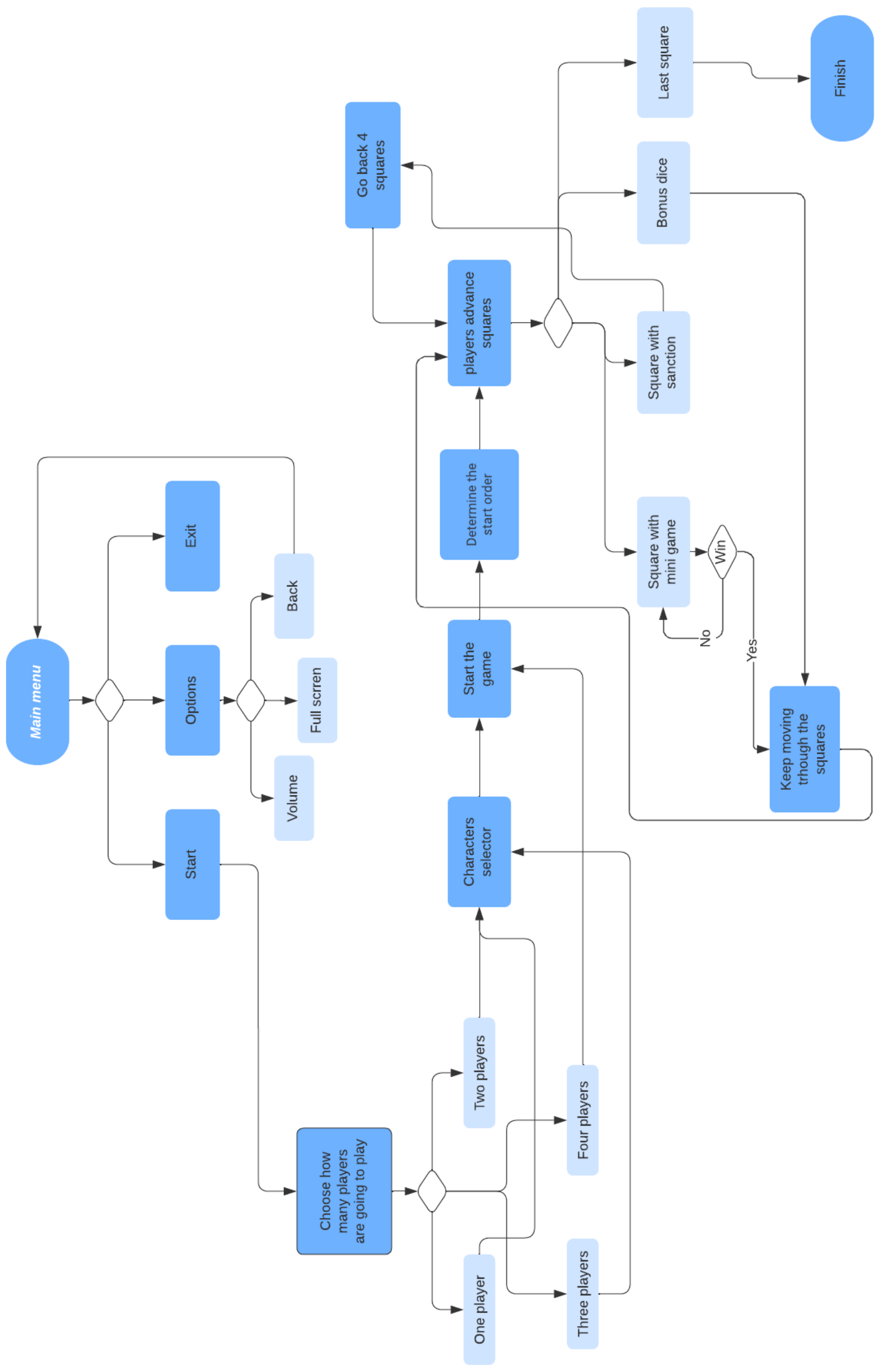


Figure 3.2: Flowchart diagram

3.3 System Architecture

After conducting several tests in order to identify the limitations for running the game, I have reached the conclusion that there are indispensable minimum requirements to guarantee a smooth and solid experience. Next, the minimum required architecture is detailed:

- **Operating System:** Window 7 64-bit or superior.
- **CPU:** Compatible with Dual-Core Intel or AMD equivalent.
- **RAM:** 4GB
- **GPU:** Nvidia GeForce GTX 550Ti
- **DirectX:** version 11

On the other hand, despite a computer with the aforementioned specifications being able to run this game, it is important to consider that there is a more recommended architecture that will ensure a much safer execution. For an optimal and seamless experience, it is suggested that the system architecture meets the following additional requirements:

- **Operating System:** Window 10 or Window 11 64-bit.
- **CPU:** Compatible with Quad-Core Intel or AMD equivalent.
- **RAM:** 4GB
- **GPU:** Nvidia GeForce GTX 750
- **DirectX:** version 11

3.4 Interface Design

The game user interface in this particular case has been designed to offer a simple, intuitive, and accessible experience to players of all ages. One of the highlights of this interface is its ability to connect different generations, allowing both grandparents and their grandchildren to enjoy the game together.

Special attention has been given to avoiding a cluttered and overwhelming interface, opting to display only the necessary elements for players to understand and follow the game's progress clearly and without confusion.

In the main menu, you will find the game title, along with three buttons: "Start," "Options," and "Exit." Additionally, there are two dice displayed, which start spinning when the mouse hover over them. In this screen, I have created everything myself, except for the dice image, which was sourced from the internet (Section A.1.1). Nonetheless, I have taken care of animating them.

On the other hand, I have personally crafted the menu design for selecting the number of players and the character selector. Additionally, I have created the character designs showcased in the selector. The only elements coming from other sources are in section A.1.1 and in section A.1.2).

The user interface (see Figure 3.3, which is present throughout almost the entire game play, consists of different elements including:

- The dice, whose appearance is essential in determining the players' movements, as well as displaying the numerical sum of the dice.
- Another relevant element in the interface is the indicator of the current player's turn, which keeps all participants informed of when it is their turn to play.
- Furthermore, it has been considered important to visually indicate the type of square on which the player has landed so that they have immediate information about the circumstances they are in.
- And a button to proceed to the next turn or to the mini game, depending on which square the player has landed on.

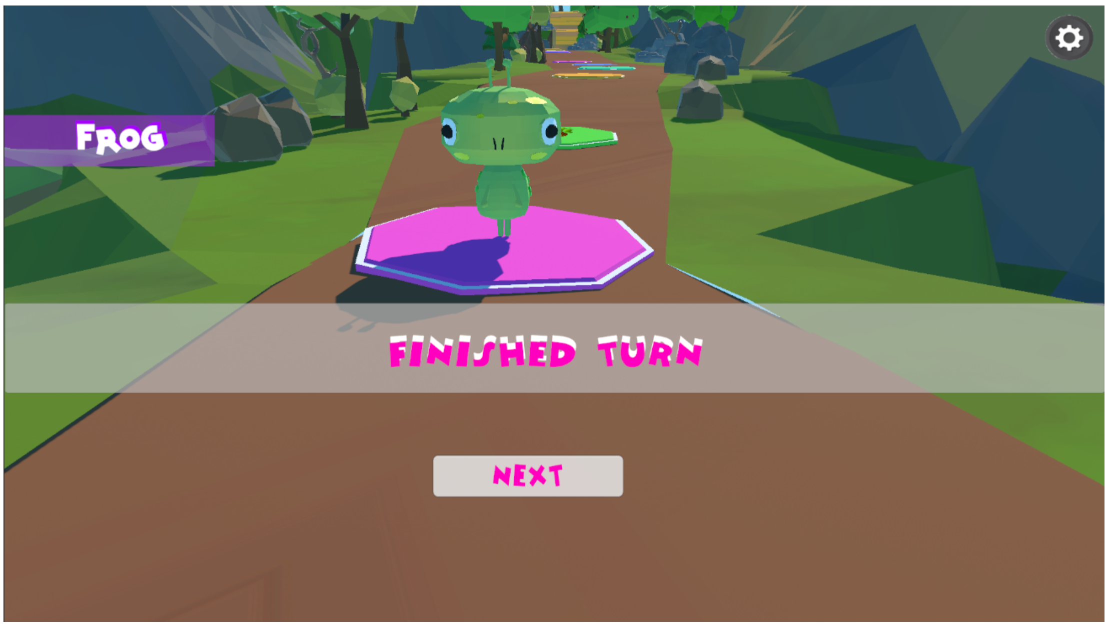


Figure 3.3: The Game User Interface (image taken in game)

Chapter 4

Game Design

Contents

This chapter aims to clarify various concepts related to this project, as well as provide additional information on fundamental aspects. These include the target audience, the artistic references behind this project, and the mechanics employed.

4.1 General Data

- **Age Rate:** 8-99 years old
- **Developed by:** Alicia Muñoz López
- **Platform:** Computer
- **Genre:** Party games

4.2 Game Description

This game is designed to provide players with a fun and entertaining afternoon with friends or family. It is inspired by iconic games such as Mario Party and Wii Party, where players must progress along a path or course towards the goal, overcoming obstacles and challenges.

The player will navigate a game board set on an island, aiming to reach the final square before the other players. They must be careful not to land on penalty squares and face various tests and obstacles along the way. These tests or obstacles will take the form of diverse mini-games or party games.

4.3 Game Play

This game consists of a total of 32 squares, some with an octagonal shape and others with a circular shape. Each type of square offers a different experience:

- Mini game squares require the player to overcome a challenge to avoid losing their turn in the next round.And there are three types:
    - Coconut mini game: The goal of the game is to dodge the falling coconuts for a period of 15 seconds, and the player has 3 lives to achieve success.
    - Shark mini game: In this challenge, you must cross the water without being caught by the sharks. The player has a total of 5 lives to accomplish this successfully.
    - Fishing mini game: This game is about catching the fish. There are a total of 4 fishing rods, and the player must blindly choose which one contains the fish. They have a total of 3 attempts to succeed.
- On the other hand, there are the "safe" squares. In these squares, the player does not face the possibility of losing a turn. These squares provide a moment of relief and tranquility.

In addition, this game offers different experiences depending on the chosen mode of play, thanks to the local multiplayer option. At the beginning, a screen is displayed where players can choose the number of participants, ranging from 1 to a maximum of 4. If fewer than 4 players are selected, the remaining characters will be controlled by artificial intelligence. Furthermore, players have the option to choose their preferred character before starting the game using a character selector.

To determine the turn order, players will roll the dice, and the one who gets the highest number will be responsible for starting the game.

4.4 Mechanics

From the very beginning, the creation of this video game has prioritized simplicity and clarity in its mechanics, aiming to provide an inclusive and enjoyable experience for people of all ages and levels of gaming expertise. Special attention has been given to avoiding frustration that may arise from complex or confusing mechanics.

It is important to highlight that simplicity in the mechanics does not mean a lack of depth or fun. A careful balance has been sought between accessibility and the satisfaction of feeling competent in game play.

During the game, interaction with the game board is limited to the use of the mouse, which allows players to roll the dice and determine the position to which their character should move. However, different mechanics are presented in the mini games. In the fishing mini game, the interaction is also exclusively done using the mouse, as players have to select the fishing rod they believe contains a fish. On the other hand, in the coconut and in the shark mini game, players must avoid various objects that appear, for which the keyboard is used (WASD or arrows).

4.5 References

As it was mentioned before, inspiration has been drawn from popular video games such as Mario Party and Wii Party to develop the mechanics and level design of this game. These references have played a crucial role in creating a fun and entertaining experience for the players.

Regarding the artistic aspect, an approach has been chosen that combines elements from the game Animal Crossing (Figure: 4.1), visual references found on Pinterest (Figure: 4.2), and the low-poly graphics technique. This artistic approach allowed to create characters with a charming and distinctive design while maintaining an attractive and friendly visual aesthetic.

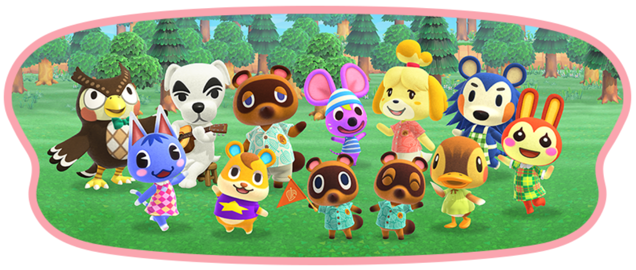


Figure 4.1: Animal Crossing characters

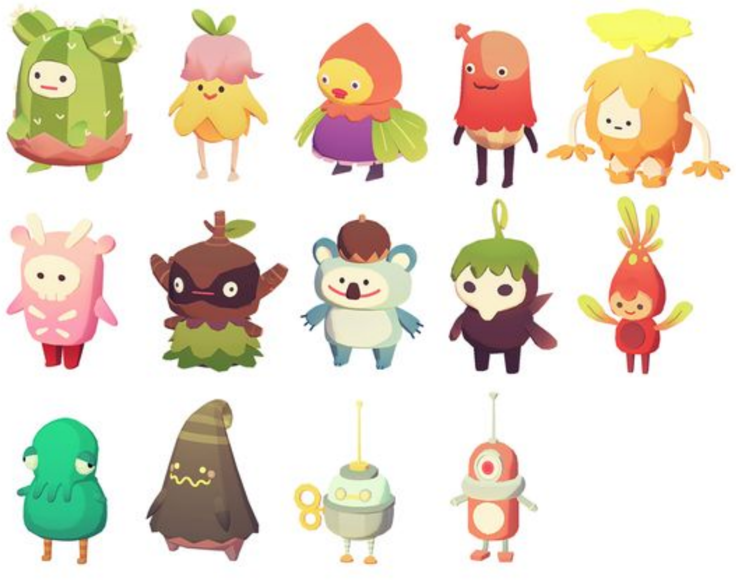


Figure 4.2: Pinterest references

The influence of Animal Crossing is reflected in how the game's characters have be designed and portrayed, aiming to convey that sense of warmth and familiarity that characterizes the Animal Crossing [12] universe. On the other hand, Pinterest[14] has been a valuable source of visual inspiration, enabling to explore different artistic styles and trends that have enriched the creation of the characters.

Lastly, the low poly graphics technique has been employed to achieve a more geometric and stylized visual style for the characters and environments of the game. This technique involves representing objects and characters with a reduced number of polygons, resulting in a simplified yet visually appealing and aesthetically pleasing appearance (Figure: 4.3).

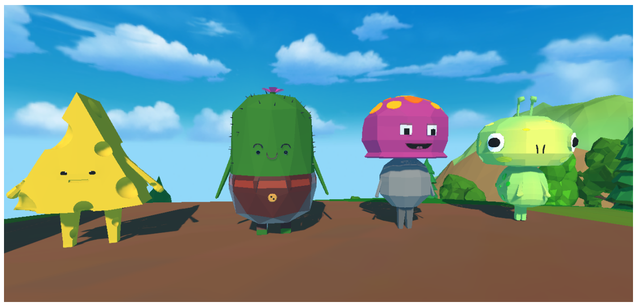


Figure 4.3: Characters of the project

Chapter 5

Work Development and Results

Contents

The present chapter aims to thoroughly and detailed cover each of the diverse steps followed in order to achieve the final result that have been obtained. It compiles and documents every single change that has occurred throughout the process.

5.1 3D Game Art

To start, it will be explained how the different characters and the terrain of this game were modeled and textured, as it was the first thing worked on.

5.1.1 The frog (Figure: 5.1)

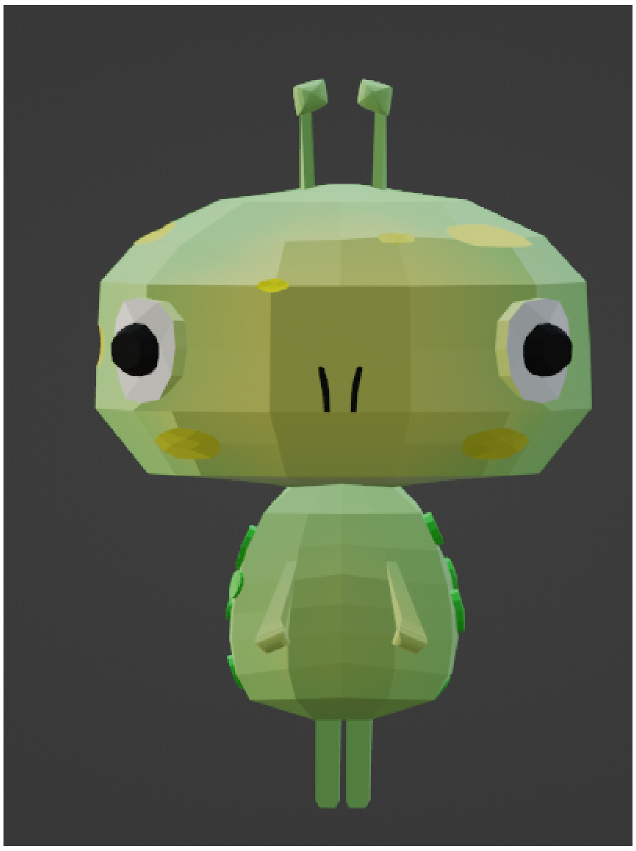


Figure 5.1: Frog model

When the frog character started the creation process, the main idea was to make it with a large head and bulging eyes, but the idea of adding colorful spots and the gradient on its body emerged later.

Specifically, to carry out the texturing of the frog's body (Figure: 5.2, the first step was creating it in Blender using two different materials to represent the desired color tones: green and yellow. Later, a *Mix Shader* was used to combine these two materials. However, upon doing so, the material colors simply blended together, resulting in a single color, whereas the objective was to achieve a smooth gradient between the two colors. To accomplish this, it was added a node called *Gradient Texture*, which allows to generate the necessary gradient. Additionally, was incorporated a *ColorRamp* to precisely configure the desired type of gradient. After conducting various tests, the *B-Spline* gradient type proved to be the most suitable for this case. Through the *ColorRamp*, it was adjusted the values ranging from the darkest to the brightest tones until it was obtained the desired result.

After carrying out all these steps in Blender, the problem arose that this material wouldn't export correctly to Unity. Therefore, a solution was searched to export the material to Unity, and during the research, it was discovered that Blender can convert the procedural texture into an image texture. To achieve this, it was necessary to perform an *Unwrap* of the model and add an *Image Texture* node. Then, a *Bake* was performed using the configuration shown in the image, and waited for the process to complete (Figure: 5.3).

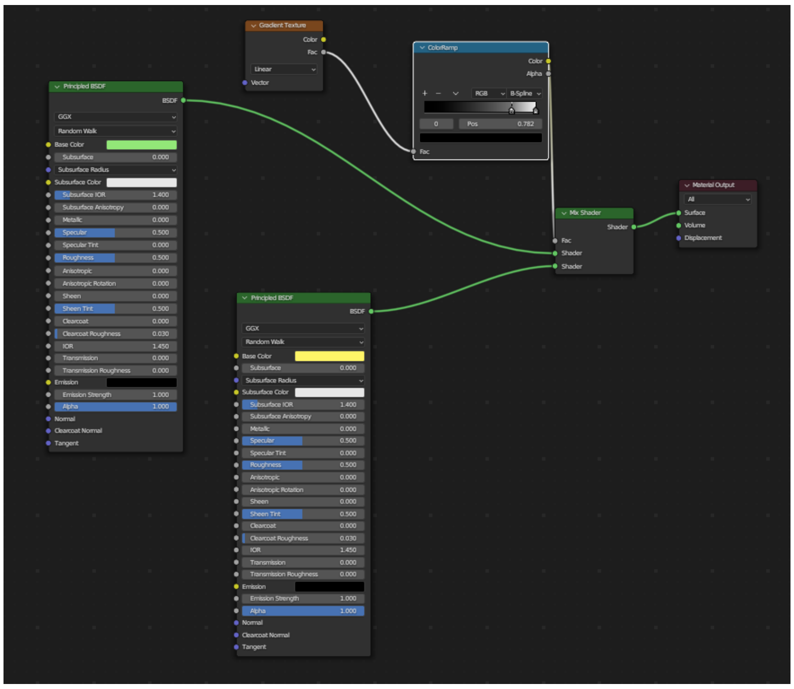
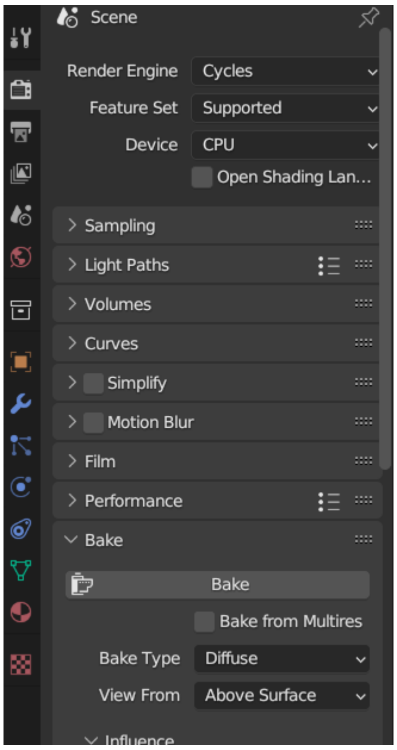
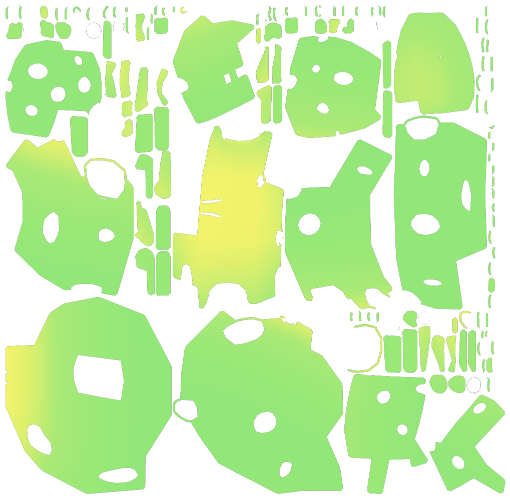


Figure 5.2: Frog material



Render Engine: Cycles
Feature Set: Supported
Device: CPU
Open Shading Lan...
Bake from Multires
Bake Type: Diffuse
View From: Above Surface

(a) Bake configuration



(b) Frog texture

Figure 5.3: Bake process

5.1.2 The cactus (Figure: 5.4)

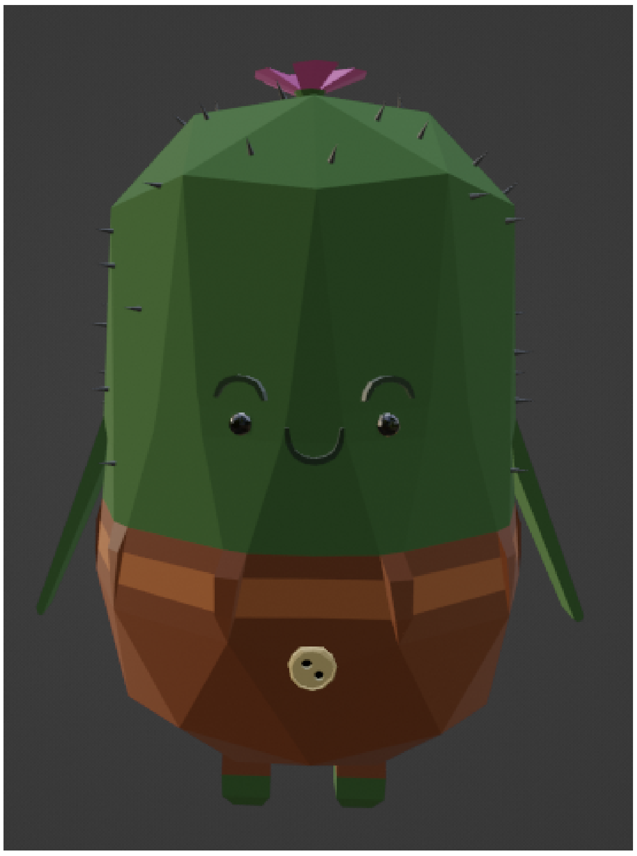
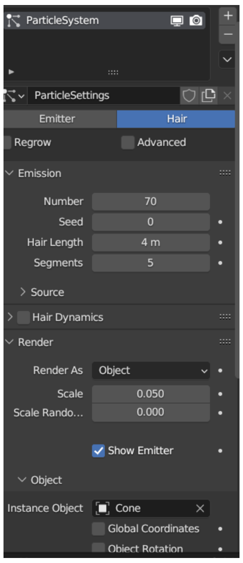


Figure 5.4: Cactus model

In the case of the cactus modeling, initially there was no intention of adding pants. However, later on, the idea of including pants arose. It is important to note that the pants were not created using the *Cloth* tool; instead, they are the same mesh with a different assigned material. This decision was made to simplify the animation process, as animating an object with the *Cloth* modifier applied can be more complex. Furthermore, this technique contributes to achieving a more cartoon style. To further emphasize the cactus character's features, in addition to giving the body a dark green color, it was decided to incorporate spikes.

To incorporate the spikes, a cylinder object was created in Blender and its dimensions were adjusted according to the desired size for each spike. Next, it was ensured that the faces of the cactus were properly oriented. Then, a particle system was added to the cactus, and the previously created cone object was assigned to that system. Further adjustments were made to the configuration until the desired result was achieved (Figure: 5.5). However, a problem arose where the spikes appeared all over the body, and it was not desirable for the character to have spikes on the face, legs, or arms. To address this, after applying the particle properties, the cones that were not correctly positioned are removed from *Edit Mode*.



ParticleSystem

ParticleSettings

Emitter | Hair

Regrow | Advanced

Emission

Number 70
Seed 0
Hair Length 4 m
Segments 5

Source

Hair Dynamics

Render

Render As Object
Scale 0.050
Scale Rando... 0.000
✓ Show Emitter

Object

Instance Object Cone
Global Coordinates
Object Rotation

Figure 5.5: Spike configuration

5.1.3 The cheese (Figure: 5.8)

Regarding the cheese character, there were doubts about how to represent the holes that give it its distinctive appearance and make it resemble a real cheese. Initially, the idea of applying a texture that would simulate the effect of holes on the character was considered 5.6. However, this effect looked unrealistic as the holes appeared flat and lacked depth, making it evident that they were not three-dimensional 5.7 .

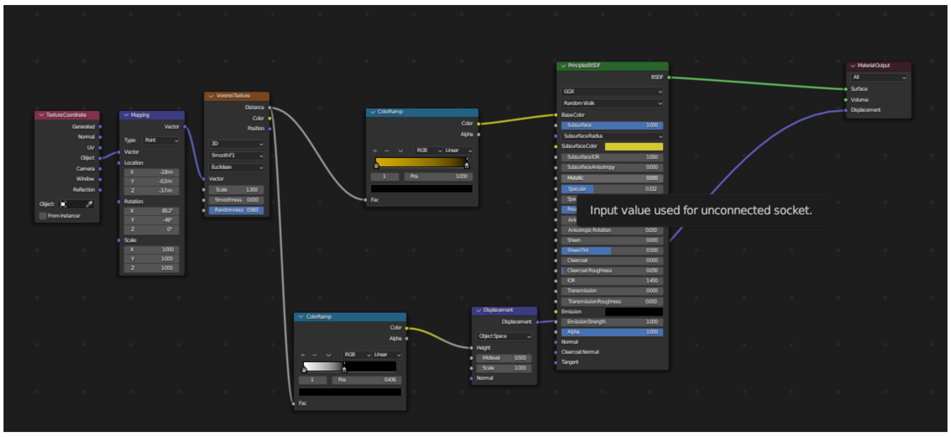


Figure 5.6: Cheese material

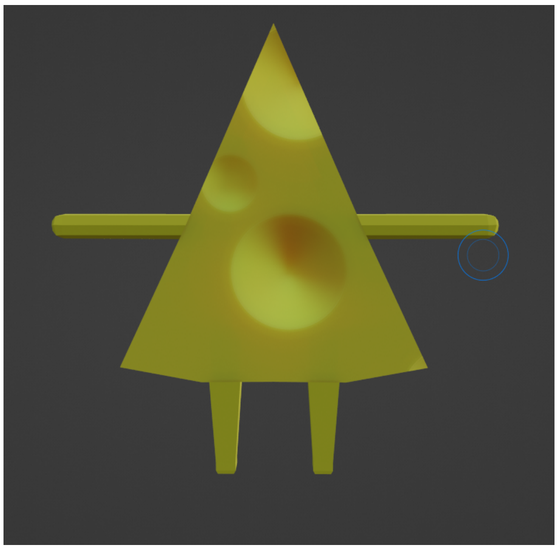


Figure 5.7: Cheese model with holes texture

Finally, the decision was made to create the holes manually. This involved selecting the corresponding vertices within the mesh of the character and applying the *Bevel* function. Then, the parameters were adjusted until a satisfactory circular shape was achieved. Subsequently, the faces of these circles were extruded inwards to create the desired holes.

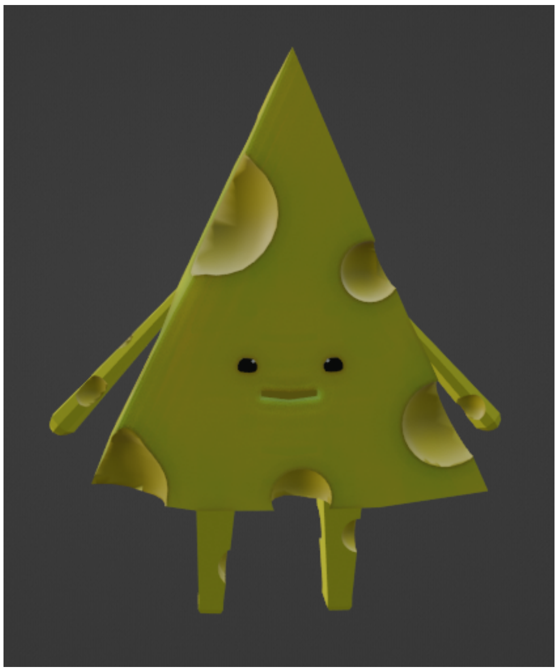


Figure 5.8: Cheese model

5.1.4 The mushroom (Figure: 5.9)

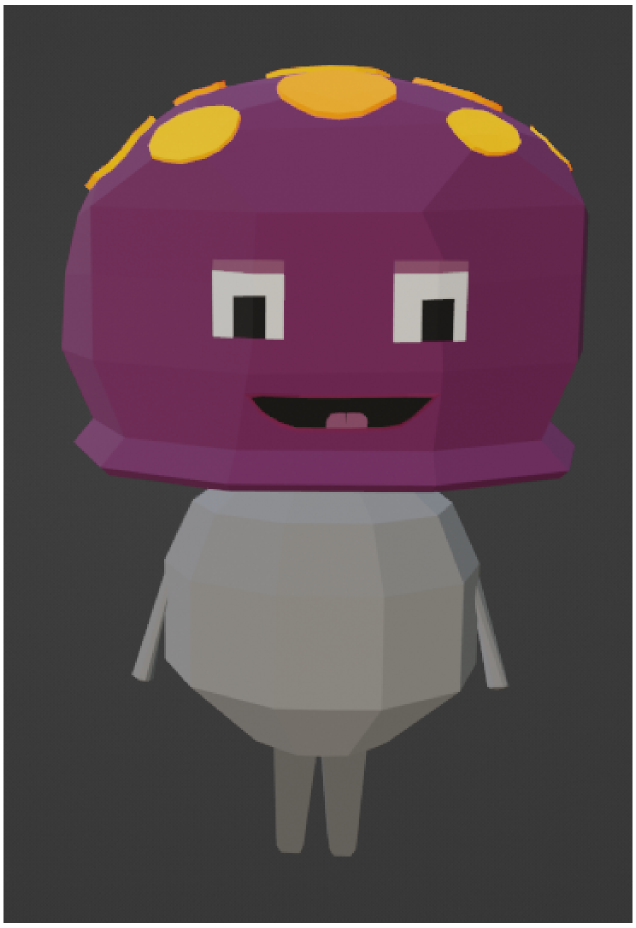


Figure 5.9: Mushroom model

The mushroom character was the first to be created, which made it the most time-consuming to complete. This was not due to the complexity of the modeling process itself, but rather because it involved making decisions about the style that would be used for the other characters as well. Accomplishing this task successfully required a significant amount of effort and dedication.

The modeling process progressed relatively quickly, as there was a general idea of the desired appearance for the character. However, when it came to the texturing process, things took a different turn. Initially, attempts were made to learn how to paint textures in Blender using the *Texture Paint* tool. Unfortunately, despite several efforts, the desired finish could not be achieved. After several days of experimentation, it was concluded that a more appropriate and consistent approach with the low-poly style would be to use flat colors instead of complex or hand-painted textures.

5.1.5 The terrain (Figure: 5.10)

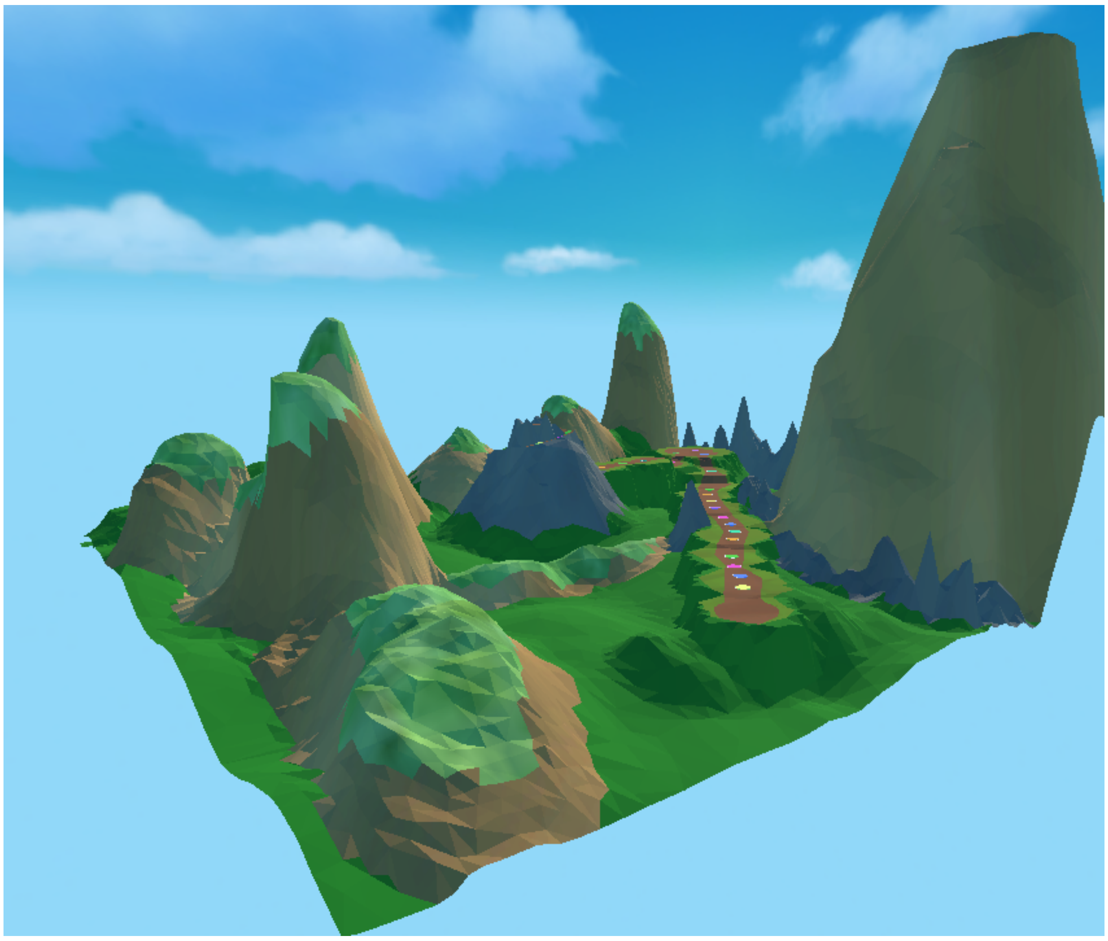


Figure 5.10: Terrain model

The terrain was a significant concern from the outset, with the goal of creating a realistic 3D environment. Initially, the idea of designing a unique environment for each square was considered but swiftly dismissed due to the overwhelming amount of work it would require. After careful consideration, a decision was made to follow the approach seen in games like Mario Party, where a single thematic environment is utilized. In this case, a path leading to the mountain was created, and the squares were adapted accordingly to fit within this environment.

Once a clear idea was formed, the terrain creation process began in Unity using the *Terrain* tool provided by the software. However, as the completion neared, it became evident that the terrain appeared too realistic, both in terms of textures and polygon count. The mountains had a smooth appearance that did not align with the desired aesthetic. Consequently, a decision was made to transfer the terrain to Blender for further modifications in order to achieve a more low poly style. To accomplish this, the *Displace* modifier was applied, allowing adjustments to be made for the desired effect. A new texture, sourced online [10], was incorporated to enhance the overall appearance. Furthermore, the *Decimate* modifier was employed to reduce the polygon count by adjusting the ratio value. Additionally, the inclusion of flatter textures contributed to achieving the desired cartoon-style aesthetic.

5.1.6 The Dice (Figure: 5.11)

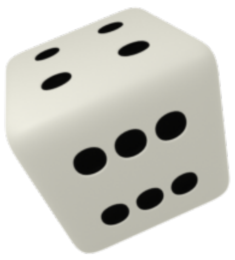
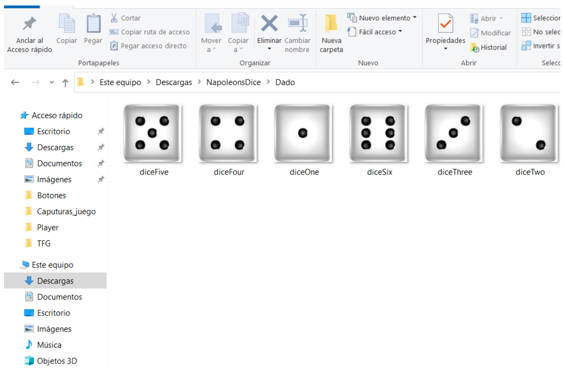
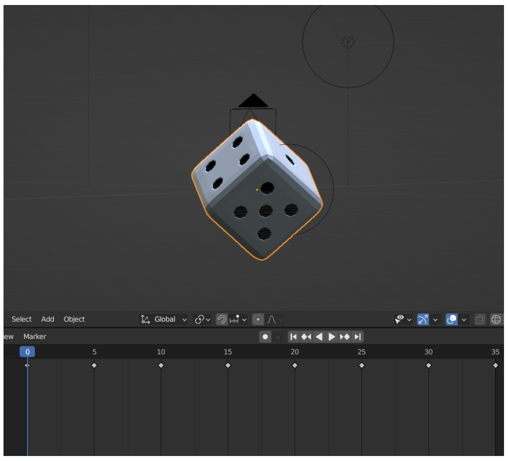


Figure 5.11: Dice

In the case of the dice animation, the task at hand was a laborious one, as the objective was to achieve a simulated 3D effect using 2D sprites. The process involved obtaining a texture from the internet (A.1.1) and applying it to a cube in Blender (Figure: 5.12), where the animations for both dice rolling and spinning were created. Upon completion, the animations were rendered frame by frame (Figure:5.13). Finally, the sequence of sprites was imported into Unity's animation component for further implementation.



(a) Sprites from Internet



(b) Dice in Blender

Figure 5.12: Process in Blender

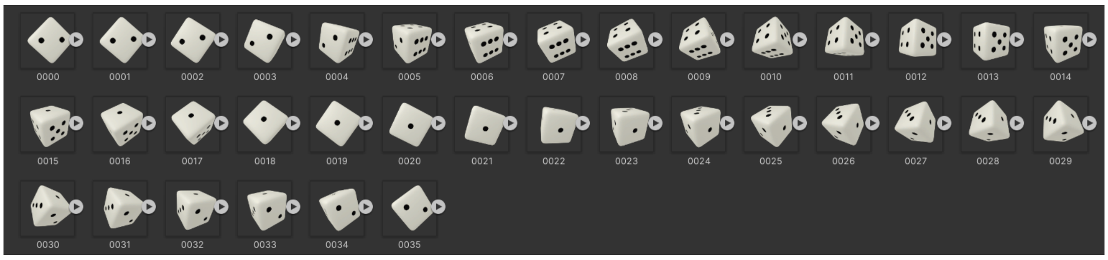


Figure 5.13: Rolling sprites

5.2 Programming

To finish the explanation of the functioning of the dice, once in Unity, its initial state is *rolling* and when the player clicked on the dice, or the AI waited for a few seconds, the dice was thrown (Figure: B.1). After that, by code too (Figure:B.2), one of the *Boolean* in the *Animator* (Figure: 5.14) was set to *true* so that once the dice was thrown, depending on the random number that came up, the animation would end showing one number or another.

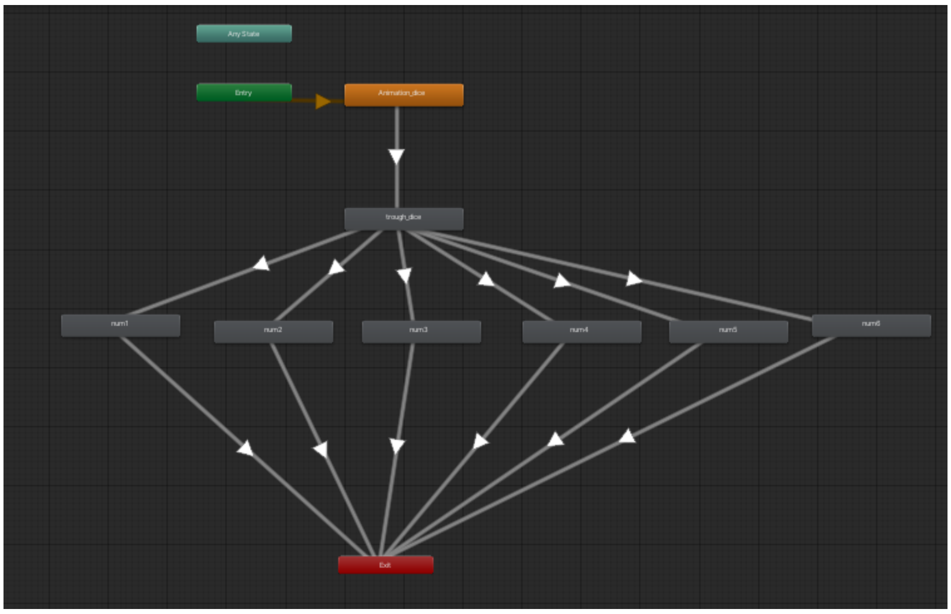


Figure 5.14: Dice animator

Before starting the game, the player will go through two scenes: one where they choose the number of participants and another to select the characters. In the character selection screen, there is a key script (Figure: B.9), that records the information of which player has chosen which character and, by process of elimination, which ones will be controlled by the artificial intelligence (AI). This information is stored in a list where the player number choosing at that moment (player number 1, player number 2, etc.) is added to the position assigned to the character. For example, the position assigned to the mushroom is 0 and the position assigned to the cactus is 4. Also takes into account the number of players who are going to play (information extracted from the player selection screen). And additionally, there is another background list that is responsible for removing the characters that have already been chosen and replacing them with 3D models that have a grayish material to indicate that they can no longer be selected (Figure:5.15).

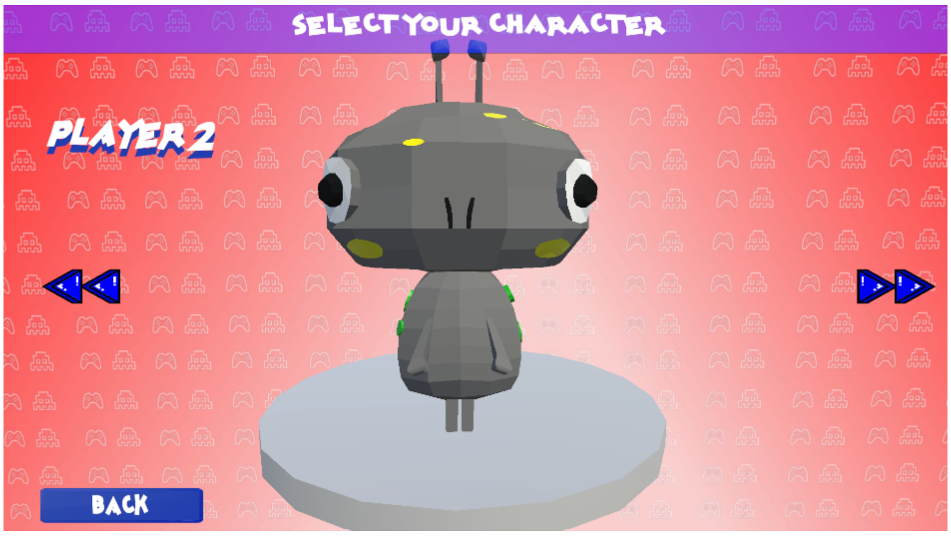


Figure 5.15: Character chosen

Once the player has chosen the number of participants and the selected character, they will proceed to determine the starting position. This is done using a script (Figure: B.10) that first checks whether it is the AI's turn or a player's turn and displays it on the screen. If it is the AI's turn, the character's name is displayed, and if it is a player's turn, *Player 1 or Player 2,etc.* is displayed based on which player's turn it is. The numbers rolled by the different characters are stored in a list, which is then sorted in descending order to obtain the starting position (See Figure: 5.16).

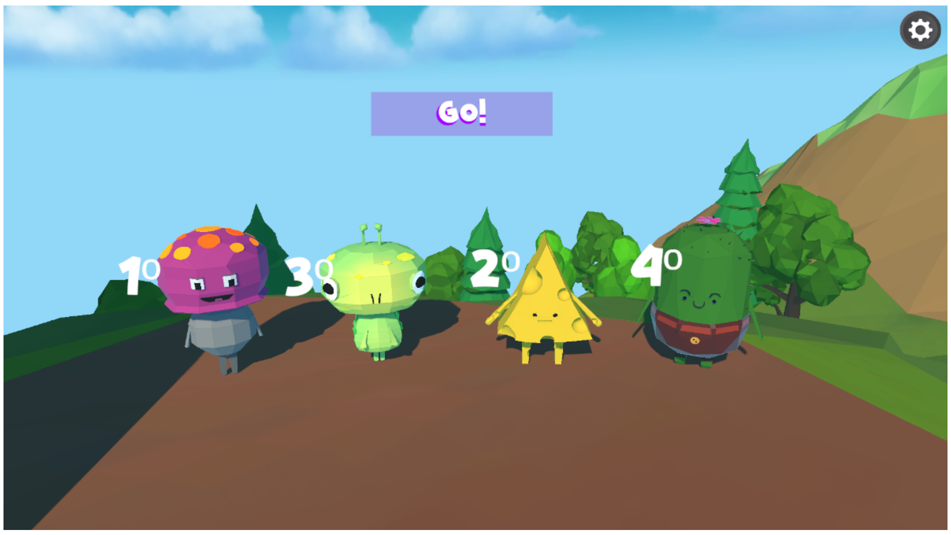


Figure 5.16: Choose starting positions

On the other hand, to facilitate the turn switch, a camera switching approach is used. For this purpose, the *Cinemachine* tool has been employed, assigning a different virtual camera to each character. To control the activation and switching of these virtual cameras, a script has been implemented (Figure: B.11), which includes a variable called *ActiveCamera* that represents the currently active virtual camera. Additionally, this script includes four methods:

- **IsActiveCamera:** This method checks if a specific camera is the currently active camera.
- **SwitchCamera:** This method allows switching the active camera to a specific camera. When performing this change, the priority of the new active camera is set to 10, while the priority of the other cameras is reduced to ensure that only one camera is active.
- **Register:** This method registers a new virtual camera in the list of available cameras.
- **Unregister:** This method removes a virtual camera from the list of available cameras.

To ensure effective coordination of turns and maintain organized player data, A shared script (Figure: B.12) was developed to store the information in a list of lists.. Each sub list includes a string representing the player and an integer indicating their ending position at the conclusion of their turn. This script also includes the *GetWaypoints* function, which is used to inform the player about the next way points they need to reach after rolling the dice. To achieve this, the function complete a list with the way points starting from the current position up to the current position plus the number rolled on the dice.

Finally, the player movement scripts. Although they are quite similar, each one is specific to a particular player. In these scripts, several actions are performed.

Firstly in this script, the *ThroughDice* function is called to through the dice, whether it is the artificial intelligence or the player. After this, the movement is carried out using the *navmesh*, which is a representation of the navigation surface in the game.

Once the character has reached the corresponding square, the character rotates, disassociate the camera from the character, so that it turns and faces the camera. Additionally, a message is displayed indicating whether the character has to play a mini-game or if it's the next player's turn.

5.3 Mini games

there were several ideas for mini games in consideration. However, as the project progressed, it became apparent that each mini game would require a significant amount of time and effort. After thorough deliberation, a decision was made to focus on developing three mini games that were feasible within the established period of time.

In order to identify which player is participating in the mini game, it was decided to create tokens with the faces of each character. This increase the diversity of the project by combining elements in 3D and 2D, resulting in a visually more varied game (See Figure: 5.17).

5.3.1 Fishing mini game

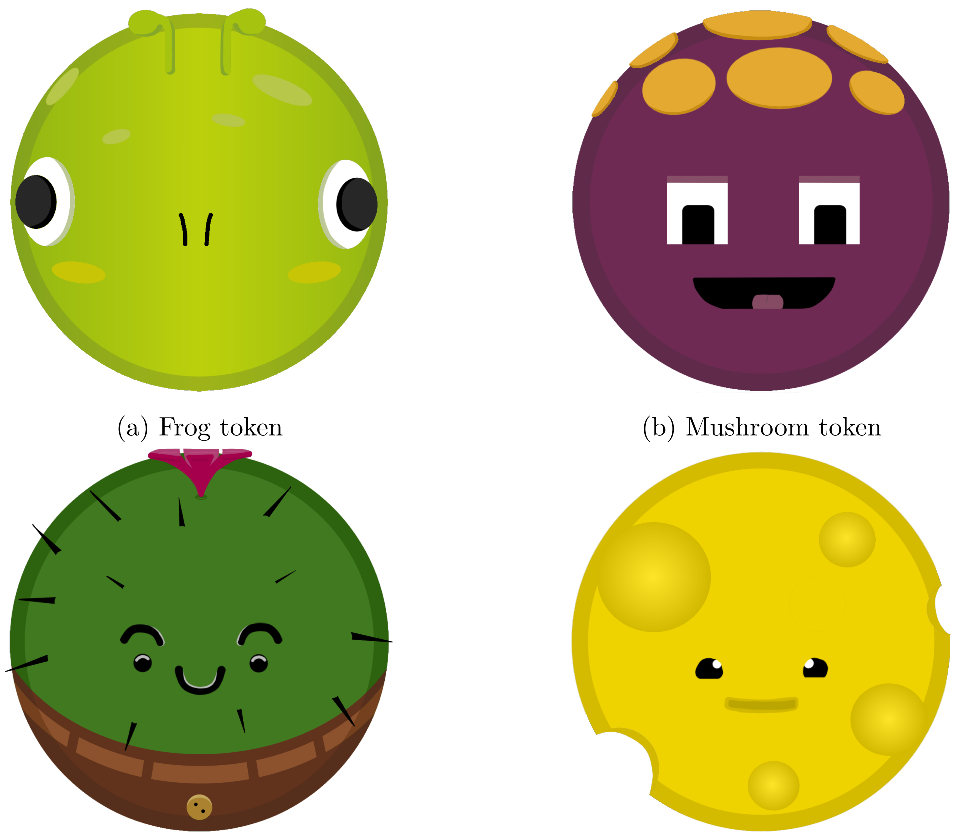
This mini game consists of 4 fishing rods and only one will have a fish. The player must randomly choose which one they think has the fish. They will have three chances.In case the player loses, they will not be able to play the next turn.

With regards to the development of this mini game, on the artistic side, I was responsible for creating all the illustrations, both for the environment and the fishing rod (Figure: 5.18). Additionally, I performed frame by frame animation of the fishing rods using the animation tool provided by Krita (Figure: 5.19).

In terms of programming, this mini game requires three different scripts. The first one is responsible for randomly determining which fishing rod holds the fish each time the mini game is played. The second script is in charge of animating the fishing rods, making them move randomly to confuse the player and create the illusion that the fish could be in any of them (See Figure: B.3). Finally, the third script detects when the player clicks on a fishing rod, triggering the animation of pulling out the fishing rod (Figure: B.4).

5.3.2 Coconuts mini game

This mini game references the classic game where the player must dodge objects falling from the sky. In this specific case, the player must dodge coconuts during 15 seconds to



(a) Frog token (b) Mushroom token

(c) Cactus token (d) Cheese token

Figure 5.17: Tokens

win. They are given 3 lives to achieve this. If they lose all 3 lives, they will have lost and won't play in the next round (Figure: 5.20).

In terms of artistic aspect in this mini game,I have performed the tasks of searching for assets (A.1.1) and designing the GUI, both for the explanation (Figure:5.21) screen and for the actual game screen itself.

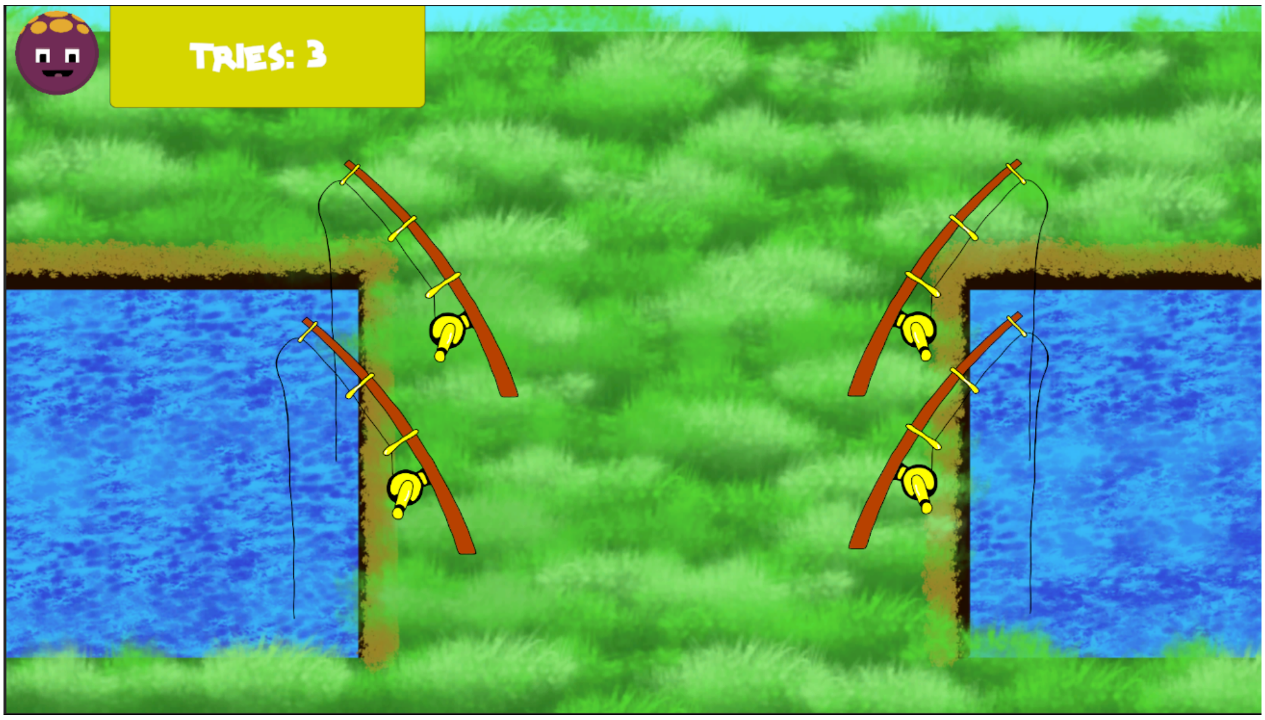


Figure 5.18: Fishing mini game

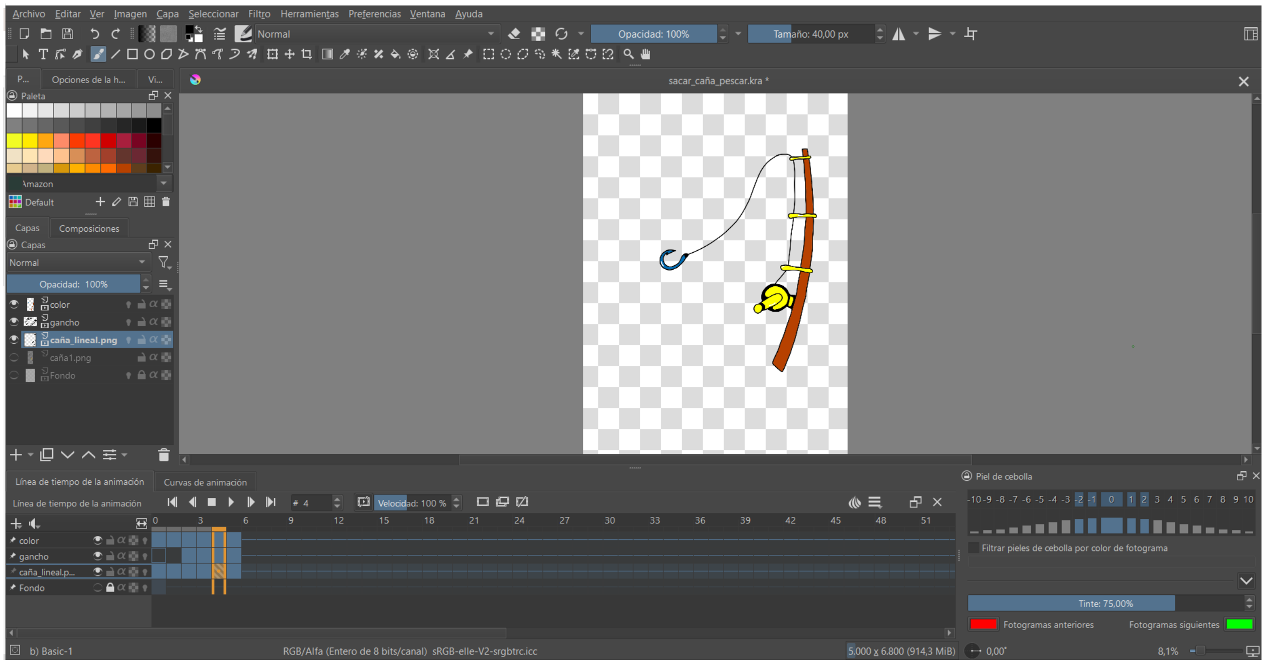

Archivo Editar Ver Imagen Capa Seleccionar Filtro Herramientas Preferencias Ventana Ayuda
Normal
Opacidad: 100%
Tamaño: 40,00 px
sacar_caña_pescar.kra *
Paleta
Amazon
Default
Capas Composiciones
Capas
Normal
Opacidad: 100%
color
gancho
caña_lineal.png
caña1.png
Fondo
Línea de tiempo de la animación Curvas de animación
Línea de tiempo de la animación
Velocidad: 100 %
Piel de cebolla
Filtrar pieles de cebolla por color de fotograma
Tinte: 75,00%
Fotogramas anteriores
Fotogramas siguientes
b) Basic-1
RGB/Alfa (Entero de 8 bits/canal) sRGB-elle-V2-srgbtrc.icc
5.000 x 6.800 (914,3 MiB)
0,00°
8,1%

Figure 5.19: Animating the fishing rod

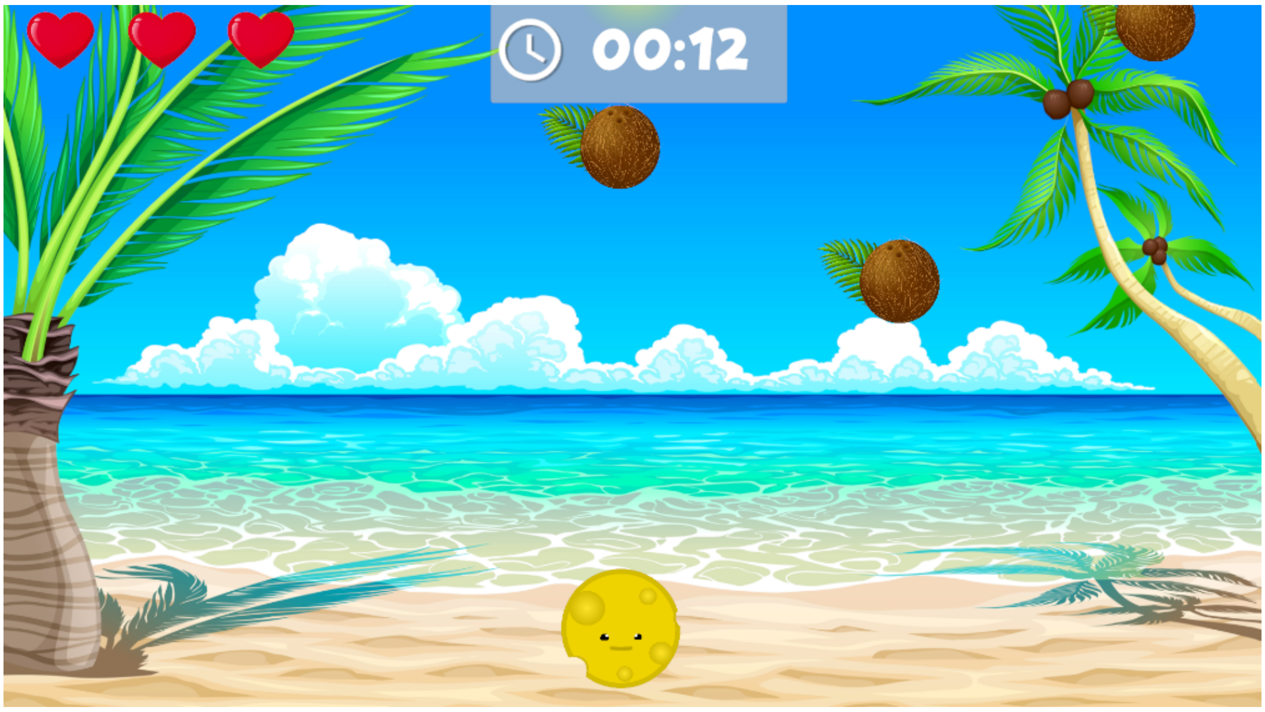


Figure 5.20: Coconut mini game (in-game image)

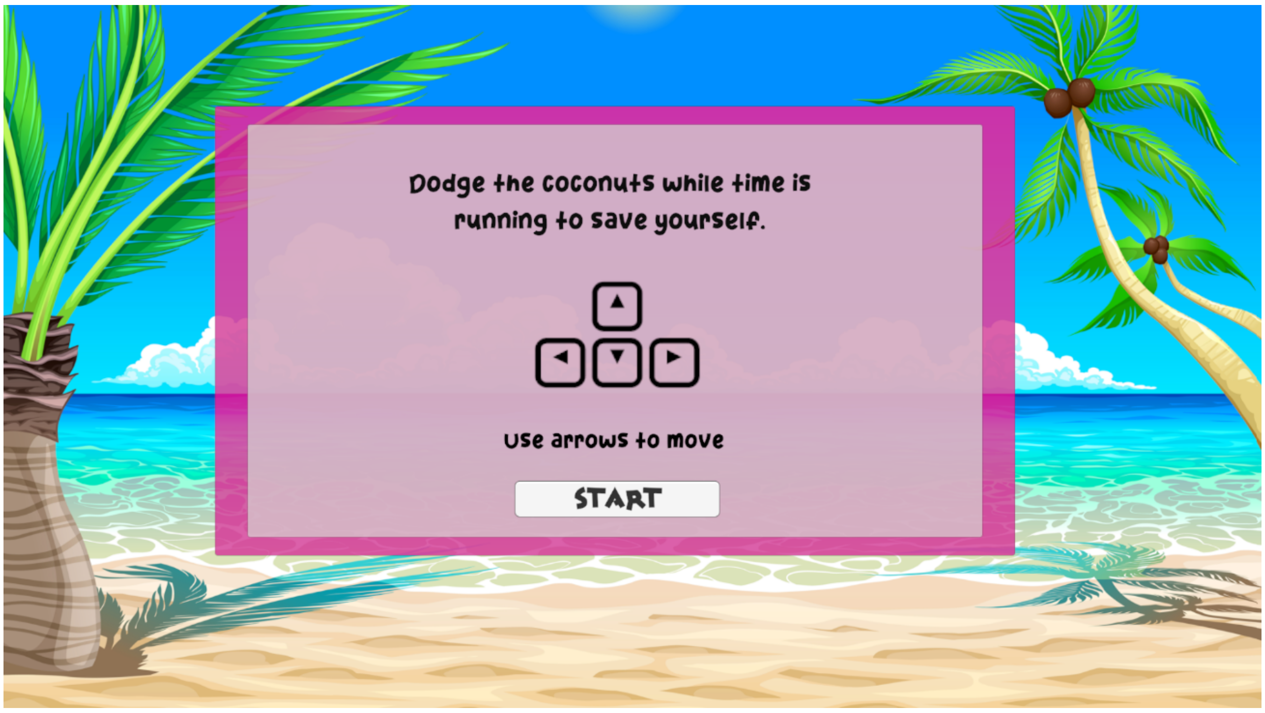


Figure 5.21: Mini game instructions

For the programming part, it began with the developing a coconut generator (Figure: B.5) that creates coconuts from the top of the canvas and removes them when they go beyond the bottom. This was achieved by obtaining the camera dimensions to generate coconuts within that area. There are two variables worth mentioning in this code: *tiempoEntreCocos* (time between coconuts) and *alturaGeneracion* (generation height). The speed of coconut generation depends on these two values, and consequently, the difficulty level of the game can be adjusted accordingly.

Furthermore, a script for the character was also developed. This script captures horizontal movement through arrow keys or*AWSD* keys on the keyboard, collision detection with the coconuts, so when they come into contact, the coconut disappears, and the character's life is gradually reduced. This functionality is achieved in conjunction with another complementary script responsible for managing lives (Figure: B.6), using a *switch* control structure that removes hearts from the screen based on the remaining lives

And finally, for this mini game, a countdown timer was implemented that counts down from 15 to 0. When it reaches zero, it displays a message indicating that the player has won the mini game. If the player is eliminated before the time runs out, the timer stops. To achieve the desired format of the timer on the screen, it is used the function*string.Format("00:00:01:00")*.

5.3.3 Sharks mini game

The shark mini game is also a reference to another classic, Crossy Road [25]. In this game, the objective is to reach the other side of the water without being attacked by the sharks. The player has up to 5 chances (5 lives) to achieve this. Each time the player is attacked by a shark and still has remaining lives, they will return to the beginning and have to try again. However,if they lose all 5 hearts before reaching the goal, they will lose and won't be able to play in the next turn (See Figure: 5.22).

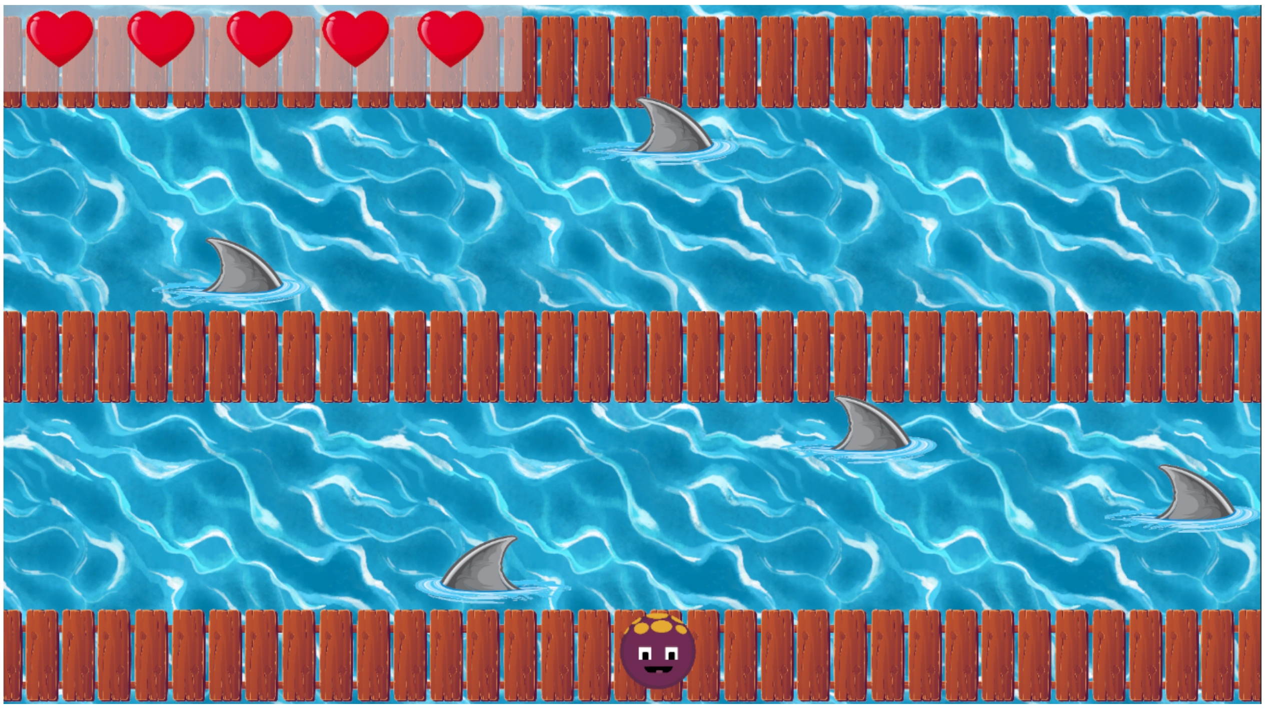


Figure 5.22: Sharks mini game

In this mini game, assets were searched in internet, just like in the coconut one A.1.1 and creating the graphical user interface (GUI). Additionally, I animated the water to make it appear more dynamic and placed each board one by one to build the bridges.

The water animation was done through code (Figure: B.7). In the *Update()* method, the scrolling of the water background is updated. To achieve this, the movement speed is multiplied by *Time.deltaTime* (the time elapsed since the last frame), which gives a value proportional to the movement speed in that frame. Finally, the offset of the main texture is updated in each frame by modifying the current offset. This is achieved by accessing the material and using *material.mainTextureOffset*. All of this creates the effect of movement.

At the programming level, there is also the patrol script (Figure: B.8), which controls the movement of the sharks. Just like in the coconut mini game, to detect when the sharks are going off-screen, the bottom right and left corners of the camera are checked. If any shark reaches the limits on the *X-axis*, *Mathf.Clamp()* is used to ensure that the X position value is within the limits, and the object's position is updated with the *clampedX* value to prevent it from going beyond the boundaries. After that, the *Flip()* method is called to reverse the movement direction and allow the shark to continue its patrol.

Also like in the coconut mini game, there are also two additional scripts. The player controller performs a similar function to the coconut one, except that this controller allows both horizontal and vertical movement. On the other hand, the live controller serves the same purpose as in the coconuts game.

5.4 Results

The main objectives of the project have been successfully achieved 1.2, resulting in the creation of a virtual and three-dimensional board game. The game features a fun and humorous style, with low-poly models, most of which have been created by myself in Blender. This distinctive style not only gives the game a unique look but also improves its performance by having fewer polygons, making it compatible and executable on most computers.

Furthermore, I have been responsible for all the animations in the game, as well as part of the 2D designs. These visual elements enhance the gaming experience by adding details and dynamism to the interaction.

The main goal of this game is to provide enjoyment to players, whether in the company of friends or family. Therefore, it is designed to be enjoyed by people of all ages who want to have a good time together, sharing laughter and entertainment in a fantasy environment.

The GitHub repository of the Unity project: https://github.com/aliciaaml/TFG.git

Chapter 6

Conclusions and Future Work

Contents

In this chapter, the conclusions of the work, as well as its future extensions are shown.

6.1 Conclusions

This project has undergone a series of changes that, although they have not affected the main objectives, have indeed influenced other secondary objectives. These changes have emerged as a response to the multiple difficulties I have faced when trying to materialize my original idea into something concrete and tangible. During these past months, I have experienced a noticeable shortage of imagination and a feeling of stagnation that generated an increasing level of stress as the months went by. Consequently, I have had to make quick and flexible decisions on the go, adapting and adjusting different aspects of my game to achieve its materialization in the most effective way possible.

However, despite these setbacks, I have enjoyed facing the experience of creating a video game on my own, as during my studies, I have worked on some projects but always in group. This experience is completely different, and although it is quite challenging to face the various problems that arise alone, it also allows for much greater learning. In my particular case, I have learned more in both the modeling and programming aspects. Therefore, even though it may be a scaled down version of what I would have desired, I am proud of this project.

6.2 Future work

As I mentioned earlier, I would have liked this project to have reached a more advanced stage of development, adding a greater amount of detail and expanding its mechanics. Therefore, the plan is to continue with the project in the future, with the aim of carrying it out based on its initial concept, or even exploring new ideas that will significantly enrich the gaming experience. The goal is to achieve a more comprehensive and engaging version of the project, surpassing its current state.

And once the improvements have been made, I would like to upload it to a platform so that people can try it out and officially have my first fully developed game visible and playable for everyone.

Furthermore, since this game features local multiplayer mode, I have also contemplated the possibility of expanding it to an online multiplayer mode. This option intrigues me, as it is an area in which I still have much to discover and learn. The idea of developing an online multiplayer component sparks my curiosity, as it presents an opportunity to acquire new knowledge and skills in this field. Not only that, but it also takes the gaming experience to a more interactive and socially connected level.

Finally, I would like to write an article about my game for CEV23 ("Congreso Español de Videojuegos"), a Spanish video game congress, that would provide an excellent opportunity to increase its visibility, reputation, and recognition.

Bibliography

[1] 3d platformer in unity health system and lives counter. https://www.youtube.com/watch?v=-tjoAUVwRjA&ab_channel=KozmobotGames. [Accessed 30-Jun-2023].

[2] Blick auf den strand vektor cartoon illustration, beach cartoon,summer backgrounds. https://www.pinterest.es/pin/557039047655015808/. [Accessed 30-Jun-2023].

[3] Computer mouse icon png and svg vector free download. https://uxwing.com/computer-mouse-icon. [Accessed 29-Jun-2023].

[4] Dice opengameart. https://opengameart.org/content/dice-1. [Accessed 29-Jun-2023].

[5] Five seamless tileable ground textures 2d floors unity asset store. https://assetstore.unity.com/packages/2d/textures-materials/floors/five-seamless-tileable-ground-textures-57060. [Accessed 30-Jun-2023].

[6] Free icon fast forward. https://www.freepik.com/free-icon/fast-forward_14565151.htm#page=5&query=fast%20forward%20icon%20cartoon&position=4&from_view=search&track=ais. [Accessed 29-Jun-2023].

[7] Free vector buttons for music player freepik. https://www.freepik.com/free-vector/buttons-music-player_957940.htm#query=botones%20reproducir%20musica&position=6&from_view=search&track=ais. [Accessed 29-Jun-2023].

[8] Free vector collection of colorful arrow stickers. https://www.freepik.com/free-vector/collection-colorful-arrow-stickers_1086584.htm#page=3&query=collection%20color%20full%20arrows&position=49&from_view=search&track=ais. [Accessed 30-Jun-2023].

[9] Game controller pad videogame svg png icon free download. https://www.vhv.rs/viewpic/hTxRoJw_game-controller-pad-videogame-svg-png-icon-free/. [Accessed 29-Jun-2023].

[10] How to create low poly terrain in unity blender. https://www.youtube.com/watch?v=6WeS8dLgGIk&t=103s&ab_channel=SamuelSalo. [Accessed 03-Jul-2023].

[11] Keyboard arrows icon free. https://thenounproject.com/icon/keyboard-arrows-1616634/. [Accessed 29-Jun-2023].

[12] La página oficial para animal crossing inicio. https://animal-crossing.com/es/. [Accessed 30-Jun-2023].

[13] Mickey. https://www.dafont.com/es/mickey.font. [Accessed 30-Jun-2023].

[14] Pinterest. https://www.pinterest.es/. [Accessed 30-Jun-2023].

[15] Sopa de aleta de tiburon, la aleta imagen png imagen transparente descarga gratuita. https://www.freepng.es/png-umf0ta/. [Accessed 30-Jun-2023].

[16] Wii party soundtrack. https://www.youtube.com/watch?v=Dowjr0YTbqw&list=LL&index=5&ab_channel=EYTPS. [Accessed 30-Jun-2023].

[17] Wii party soundtrack board game. https://www.youtube.com/watch?v=ilgQb_qun4s&list=LL&index=4&ab_channel=EYTPS. [Accessed 30-Jun-2023].

[18] Mickey kw. https://www.dafont.com/es/mickeykw.font, 11-01-2019. [Accessed 30-Jun-2023].

[19] Chromisu. Handpainted grass ground textures. https://assetstore.unity.com/packages/2d/textures-materials/nature/handpainted-grass-ground-textures-187634, 23-01-2023. [Accessed 30-Jun-2023].

[20] distillerystudio. Button by distillerystudio. https://freesound.org/people/distillerystudio/sounds/327728/. [Accessed 30-Jun-2023].

[21] Ebru Dogan. Low poly water particles effects. https://assetstore.unity.com/packages/tools/particles-effects/lowpoly-water-107563, 19-09-2018. [Accessed 30-Jun-2023].

[22] FAVPNG. Coconut milk clip art png download free. https://favpng.com/png_view/cocunt-coconut-milk-clip-art-png/KMdzwUxa. [Accessed 30-Jun-2023].

[23] Aditya Graphical. Low poly trees download free 3d model. https://sketchfab.com/3d-models/low-poly-trees-51cae4a194344e8bbfbd0a4cff205f76, 01-09-2021. [Accessed 30-Jun-2023].

[24] Laura Gómez. Análisis de wii party u. https://www.hobbyconsolas.com/reviews/analisis-wii-party-u-58485, 23-10-2013. [Accessed 29-Jun-2023].

[25] Rudy Hipster Whale. Crossy road. https://es.wikipedia.org/wiki/Crossy_Road. [Accessed 02-Jul-2023].

[26] Daniel Knaus. Low poly trees and shrubs download free 3d model. https://sketchfab.com/3d-models/low-poly-trees-and-shrubs-1f82922b6e674cc58a20cecc46122457, 21-03-2018. [Accessed 30-Jun-2023].

[27] Render Knight. Fantasy skybox free. `https://assetstore.unity.com/packages/2d/textures-materials/sky/fantasy-skybox-free-18353`, 07-03-2022. [Accessed 30-Jun-2023].

[28] Elisabeth López. Análisis de super mario party para nintendo switch. `https://www.hobbyconsolas.com/reviews/analisis-super-mario-party-nintendo-switch-310157`, 3-10-2018. [Accessed 29-Jun-2023].

[29] MMandali. Basic rock low poly download free 3d model. `https://sketchfab.com/3d-models/basic-rock-low-poly-8087b74858694be58f38ee8223ca810c`, 13-02-2019. [Accessed 30-Jun-2023].

[30] The Elliseran Modeller. Cartoon boardwalk ramp download free 3d model. `https://sketchfab.com/3d-models/cartoon-boardwalk-ramp-267cb51ed1004092bda44ef4e9928a02`, 06-07-2020. [Accessed 30-Jun-2023].

[31] purepoly. Low poly tree with twisting branches download free 3d model. `https://sketchfab.com/3d-models/low-poly-tree-with-twisting-branches-4e2589134f2442bcbdab51c1f306cd58`, 14/08/2020. [Accessed 30-Jun-2023].

[32] Bull studios. Stylized lowpoly rock download free 3d model. `https://sketchfab.com/3d-models/stylized-lowpoly-rock-6e476441b5614231bf4e8de194c418d9`, 05-05-2018. [Accessed 30-Jun-2023].

[33] Syafriza. Spicy style. `https://www.dafont.com/es/spicy-style.font`. [Accessed 30-Jun-2023].

[34] Victoria. Low poly shrub download free 3d model. `https://sketchfab.com/3d-models/low-poly-shrub-df2a9680dfe7431fa3a6403646737bc4`, 10-10-2017. [Accessed 30-Jun-2023].

[35] Wikipedia. Juego de la oca. https://es.wikipedia.org/wiki/Juego_de_la_oca. Accessed: 2023-06-14.

Appendix A

Other considerations

A.1 Bibliography

In this section, all the references to the external sounds, 3D models, sprites and fonts used for the realization of the work will be referenced.

A.1.1 A.1.1 Sprites

Sprite of the dice.[4]
Sprite icon for the settings. [7]
Sprite arrow icon.[6]
Sprite arrows keyboard icon.[11]
Sprite mouse icon.[3]
Sprite icon for the background of the scene where the character is chosen.[9]
Sprite arrow icon for selected character.[8]
Sprite background coconut mini game.[2]
Sprite coconut icon.[22]
Sprite of the sharks.[15]
Sprite of the lives icon.[1]

A.1.2 A.1.2 3D Models

Bridge model.[30]
Tree model one.[31]
Tree model two.[26]
Tree model four.[34]

Tree model five.[23]
Rock model one.[29]
Rock model two.[32]
Water low poly.[21]

A.1.3 A.1.3 Textures

Water 2D.[5]
Sky box.[27]
Textures for the terrain.[19]

A.1.4 A.1.4 Fonts

Font used in the main menu.[13]
Font used for explanations.[33]
Font used for buttons.[18]

A.1.5 A.1.5 Sound

Soundtrack used in the main menu.[16]
Soundtrack used in game.[17]
Sound effect for the buttons.[20]

Appendix B

Source code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TroughDice : MonoBehaviour
{
    public Animator animator;
    public static bool dados_tirados = false;
    public static int num = 1;
    public static bool mouse = false;
    public static float aux = 0;
    public static bool wait_t = true;

    void Start(){
        animator = transform.parent.GetComponent<Animator>();
    }
    void Update (){
        if(ElegirPosiciones.turno_terminado){
            animator.SetBool("lanzado",false);
            mouse = false;
            dados_tirados = false;
            animator.SetBool("repetir",false);
        }
    }

    public void OnMouseDown(){
        if(ElegirPosiciones.colliderDado){
            mouse = true;
            animator.SetBool("lanzado",true);
            dados_tirados = true;
            animator.SetBool("repetir",false);
        }
    }
    public void IADown(){
        Wait();
        if(wait_t == false){
            ElegirPosiciones.unica = false;
            MovementPlayer1.una_vez = false;
            wait_t = true;
            aux = 0;
            animator.SetBool("lanzado",true);
            dados_tirados = true;
            animator.SetBool("repetir",false);
        }
    }
    public static void Wait(){
        aux += 1*Time.deltaTime;
        if(aux >= 2f) wait_t = false;
    }
}
```

Figure B.1: Throwing dice

```

    if(ThroughDice.dados_tirados == true){

        Wait_through();

        if(wait_t == false){
            animator.SetBool("lanzado",false);
            range = Random.Range(1,7);
            num_sacado = true;
            aux = 0f;
            wait_t = true;
            ThroughDice.dados_tirados = false;
            animator.SetBool("repetir",false);
        }

        if(range == 1){
            animator.SetBool("numero1",true);
            animator.SetBool("numero2",false);
            animator.SetBool("numero3",false);
            animator.SetBool("numero4",false);
            animator.SetBool("numero5",false);
            animator.SetBool("numero6",false);
        }
    }
```

Figure B.2: Get the number of the dice

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MovimientoAleatorioCana : MonoBehaviour
{
    float currentTime = 0f;
    float startingTime = 0f;
    float aux =0f;
    private Animator animator;
    bool wait = true;

    // Start is called before the first frame update
    void Start()
    {   animator = GetComponent<Animator>();
        startingTime = Random.Range(1f,5f);
        currentTime = startingTime;
    }

    // Update is called once per frame
    void Update()
    {
        currentTime -= 1 * Time.deltaTime;
        if(currentTime <= 0){
            animator.SetBool("mover",true);
            Wait();
            if(wait == false){
                animator.SetBool("mover",false);
                startingTime = Random.Range(1f,5f);
                currentTime = startingTime;
                wait = true;
                aux = 0f;
            }
        }
    }

    void Wait(){
        aux += 1*Time.deltaTime;
        if(aux >= 0.5f) wait = false;
    }
}
```

Figure B.3: Random movement of the fishing rod

```
    void Update (){
        if(gana){
            Wait_pesca();
            if(wait_pesca == false){
                tries.SetActive(false);
                canas.SetActive(false);
                win.SetActive(true);
            }
        }
        else if(pierde){
            Wait_pesca();
            if(wait_pesca == false){
                canas.SetActive(false);
                lose.SetActive(true);
                tries.SetActive(false);
            }
        }
        if(mouse_fuera){
            Wait_pesca2();
            if(wait_pesca2 == false){
                animator.SetBool("pez",false);
                animator.SetBool("no_pez",false);
                wait_pesca2 = true;
                aux_pesca2 = 0f;
            }
        }
    }
    void OnMouseDown(){
        mouse_fuera = false;
        if(ElegirQuienPez.range == 0){
            animator.SetBool("pez",true);
            gana = true;
        }
        else if(contador>1){
            animator.SetBool("pez",false);
            animator.SetBool("no_pez",true);
            contador-=1;
        }
        else{
            pierde = true;
            contador = 0;
            animator.SetBool("pez",false);
            animator.SetBool("no_pez",true);
        }
    }
    void OnMouseUp(){
        mouse_fuera = true;
    }

    public static void Wait_pesca(){
        aux_pesca += 1*Time.deltaTime;
        if(aux_pesca >= 1.3f) wait_pesca = false;
    }

    public static void Wait_pesca2(){
        aux_pesca2 += 1*Time.deltaTime;
        if(aux_pesca2 >= 0.2f) wait_pesca2 = false;
    }
}
```

Figure B.4: Fishing rod detects the click of the mouse

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CocoSpawn : MonoBehaviour
{
    public GameObject cocoPrefab;
    public float tiempoEntreCocos = 1f;
    public float alturaGeneracion = 10f;

    private float temporizador = 0f;
    private Camera mainCamera;

    private void Start(){
        mainCamera = Camera.main;
    }

    private void Update(){
        temporizador += Time.deltaTime;
        if (temporizador>=tiempoEntreCocos){
            GenerarCocos();
            temporizador = 0f;
        }
        DestruirCocos();
    }

    private void GenerarCocos(){
        //obtiene las medidas de la camara para generar cocos dentro de ella
        float camHeight = 2f * mainCamera.orthographicSize;
        float camWidth = camHeight * mainCamera.aspect;

        float xMin = mainCamera.transform.position.x - camWidth / 2f;
        float xMax = mainCamera.transform.position.x + camWidth / 2f;

        //Genera los cocos dentro de las medidas de la camara
        Vector3 posicionGeneracion = new Vector3(Random.Range(xMin, xMax), alturaGeneracion, 0f);
        GameObject coco = Instantiate(cocoPrefab, posicionGeneracion, Quaternion.identity);

    }

    private void DestruirCocos(){
        GameObject[] cocos = GameObject.FindGameObjectsWithTag("coco");
        foreach (GameObject coco in cocos){
            if(coco.transform.position.y <
            mainCamera.transform.position.y-mainCamera.orthographicSize){
                Destroy(coco);
            }
        }
    }
}
```

Figure B.5: Coconuts generator

```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Lives : MonoBehaviour
{
    public GameObject heart1,heart2,heart3;

    // Update is called once per frame
    void Update(){
        switch(PlayerCoco.contador){
            case 1:
                heart3.gameObject.SetActive(false);
                break;
            case 2:
                heart2.gameObject.SetActive(false);
                break;
            case 3:
                heart1.gameObject.SetActive(false);
                break;
        }
    }
}
```

Figure B.6: Lives controller

```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FondoAgua : MonoBehaviour
{
    [SerializeField] private Vector2 velocidadMovimiento;
    private Vector2 offset;
    private Material material;

    private void Awake()
    {
        material = GetComponent<SpriteRenderer>().material;
    }
    private void Update()
    {
        offset = velocidadMovimiento*Time.deltaTime;
        material.mainTextureOffset += offset;
    }
}
```

Figure B.7: Water movement

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Patrulla : MonoBehaviour
{
    public float velocidad = 5f;
    public int direccion = -1; //-1 izquierda y 1 derecha
    private float originalX;
    private float maxX;
    private float minX;

    private void Awake(){
        originalX = transform.position.x;
        float camDistance = Vector3.Distance(transform.position,
        Camera.main.transform.position);//Calculamos distancia objeto y camara principal
        Vector2 bottomCorner = Camera.main.ViewportToWorldPoint
        (new Vector3(0, 0, camDistance)); //Aqui guardamos la esquina inferior izquierda
        Vector2 topCorner = Camera.main.ViewportToWorldPoint
        (new Vector3(1, 0, camDistance)); //Aqui guardamos la esquina inferior derecha
        //Aqui almacenamos el parametro x de las coordenadas anteriores
        maxX = topCorner.x;
        minX = bottomCorner.x;
    }

    private void Update()
    {
        transform.Translate(Vector2.right *
        velocidad * direccion * Time.deltaTime); //movimiento horizontal
        if(transform.position.x >= maxX+1.8 || transform.position.x <= minX-1.8){
            float clampedX = Mathf.Clamp(transform.position.x, minX, maxX);
            transform.position = new Vector3(clampedX, transform.position.y, transform.position.z);
            Flip();
        }
    }

    private void Flip(){
        transform.Rotate(0f, 180f, 0f);
    }
}
```

Figure B.8: Sharks patrol

```

public void Confirm(){
        if(EscogerJugador.one_player && character_choosed.Count>0){
            character_choosed[selectedCharacter] = contador;
            SceneManager.LoadScene("Terreno");
            juegoComenzar = true;
        }
        else{
            if(EscogerJugador.two_player && character_choosed.Count>0 ){
                if(contador<2){
                    character_choosed[selectedCharacter] = contador;
                    characters[selectedCharacter].SetActive(false);
                    charactersList = new List<GameObject>(characters);
                    charactersList.RemoveAt(selectedCharacter);
                    charactersList.Insert(selectedCharacter,
                    characters_deselection[selectedCharacter]);
                    characters = charactersList.ToArray();
                    contador += 1;
                    jugador.text = "Player␣" + contador.ToString();
                    jugador2.text = "Player␣" + contador.ToString();
                    selectedCharacter = 0;
                    characters[selectedCharacter].SetActive(true);
                }
                else if(contador ==2){
                    character_choosed[selectedCharacter] = contador;
                    SceneManager.LoadScene("Terreno");
                    juegoComenzar = true;
                }
            }
```

Figure B.9: List of which character has been chosen by which player

```

        if(!ElegirTurnoTerminado){
            if(DialogControler.dialog_terminado){
                if(!EscogerJugador.four_player){
                    if(flechas_characters[f] && turno_terminado== false){
                        if(EscogerPersonaje.character_choosed[f] == 0 && unica){        //IA
                            through.GetComponent<ThroughhDice>().IADown();
                            colliderDado = false;
                            if(f==0){
                                turno_jugador.text = "Mushroom";
                                turno_jugador_b.text = "Mushroom";
                            }
                            else if(f == 1){
                                turno_jugador.text = "Frog";
                                turno_jugador_b.text = "Frog";
                            }
                            else if(f== 2){
                                turno_jugador.text = "Cheese";
                                turno_jugador_b.text = "Cheese";
                            }
                            else if (f==3){
                                turno_jugador.text = "Cactus";
                                turno_jugador_b.text = "Cactus";
                            }
                        }
                        if(EscogerPersonaje.character_choosed[f] != 0){       //JUGADOR
                            colliderDado = true;
                            turno_jugador.text = "Player␣" +
                            EscogerPersonaje.character_choosed[f].ToString();
                            turno_jugador_b.text = "Player␣" +
                            EscogerPersonaje.character_choosed[f].ToString();
                        }
                        ...
                        if(solo_una){
                                solo_una = false;
                                 // Ordena la lista de mayor a menor
                                numeroDado.Sort((a, b) => b[0].CompareTo(a[0]));;
                                for (int a = 0; a < numeroDado.Count; a++){
                                    int f_personaje = numeroDado[a][1];
                                    TextMeshProUGUI texto_posSalida =
                                    num_pos_salida[f_personaje].GetComponent<TextMeshProUGUI>();
                                    texto_posSalida.text = (a+1).ToString();
                                    if(f_personaje == 0){
                                        ComunPlayers.OrdenInicioPlayers.Add("player1");
                                    }
                                    else if (f_personaje == 1){
                                        ComunPlayers.OrdenInicioPlayers.Add("player2");
                                    }
                                    else if (f_personaje == 2){
                                        ComunPlayers.OrdenInicioPlayers.Add("player3");
                                    }
                                    else if (f_personaje == 3){
                                        ComunPlayers.OrdenInicioPlayers.Add("player4");
                                    }
                                    ComunPlayers.OrdenInicioPlayers.Count);
                                }
```

Figure B.10: Choose initial position

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Cinemachine;

public static class CameraSwitcher
{
    static List<CinemachineVirtualCamera> cameras = new List<CinemachineVirtualCamera>();
    public static CinemachineVirtualCamera ActiveCamera = null;
    public static bool IsActiveCamera(CinemachineVirtualCamera camera){
        return camera == ActiveCamera;
    }
    public static void SwitchCamera(CinemachineVirtualCamera camera){
        camera.Priority = 10;
        ActiveCamera = camera;
        foreach(CinemachineVirtualCamera c in cameras){
            if(c != camera && c.Priority !=0){
                c.Priority = 0;
            }
        }
    }
    public static void Register(CinemachineVirtualCamera camera){
        cameras.Add(camera);
    }
    public static void Unregister(CinemachineVirtualCamera camera ){
        cameras.Remove(camera);

    }
}
```

Figure B.11: Camera Switcher

```

  if(primeraRonda && siguiente){
            primeraRonda=false;
            List<ElementoLista> sublista1 = new List<ElementoLista>();
            sublista1.Add(new ElementoLista(OrdenInicioPlayers[0], colisionPlayer.actual));
            PosicionActualPlayers.Add(sublista1);;

            for(int i = 1; i<OrdenInicioPlayers.Count; i++ ){
                colisionPlayer.actual = 0;
                List<ElementoLista> sublista2 = new List<ElementoLista>();
                // Agregar elementos a la sublista
                sublista2.Add(new ElementoLista(OrdenInicioPlayers[i], colisionPlayer.actual));
                PosicionActualPlayers.Add(sublista2);
            }
            dic_lleno = true;
    }

    public List<Transform> GetWaypoints()
    {
        List<Transform> recorrer = new List<Transform>();
        if (casilla_destino > 0 && casilla_destino < waypoints_todos.Count &&
        NumDado.resultado_dado_obtenido){
            if(Inicio){
                casilla_antes_tirar = colisionPlayer.actual;
                for (int i = 0; i < casilla_destino; i++){
                    recorrer.Add(waypoints_todos[i]);
                }
            }
            else{
                casilla_antes_tirar = colisionPlayer.actual;
                for (int i = colisionPlayer.actual; i < colisionPlayer.actual +
casilla_destino; i++){
                    recorrer.Add(waypoints_todos[i]);
                }
            }
        }
        return recorrer;
    }
```

Figure B.12: Common script for all players