Development of an immersive narrative videogame with a mental health background

María Beatriz Villar López

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

July 3, 2023

Supervised by: Inmaculada Remolar Quintana

To my friends and family.

Acknowledgments

Four years ago I made one of the most crucial decisions of my life. Considered as an undecided person, thinking about my future was a really difficult time for me. However, with only eighteen years old I started university in a city hundreds of kilometres far from home and not knowing anybody. Maybe I could have chosen a more common degree to study or something closest to my hometown in Andalucía, or maybe I could have studied something not related with art and go for something more established. But looking back at that moment of my life, I do not have a single regret of my decision. It has been a difficult journey, especially at the beginning, but now I am not the same person I was four years ago. Nevertheless, I am proud of myself for not throwing in the towel at the toughest times and never giving up.

This journey would not have been the same without my friends. Some of them joined later, some of them left it along the way, but I have to give thanks to every single one of them for being such nice friends, people and of course, game developers. I do not want this to be a goodbye, because it is not. I would like to thank Alicia, the closest friend I have made here. Thank you for joining me in every moment of these years, because now I do not have a single memory that you are not in. I am going to miss not living with you anymore, but I hope our life paths do not separate now. Alberto, thank you for teaching me pacience. Because yes, I have had to be patient with you. But if there is something that is true is that you are one of the most loving and caring people I have ever known with your friends and family and I have felt super loved by you. I will miss you living rent free in my house and talking about cars and computers. Fer, thank you for beign my Erasmus partner. I will never forget that experience with you. I have learned to travel alone, be more independent and live abroad, but all of that is far easier when you have such a nice friend with you.

Nàdia, Lucía, Isa, Franc, Marc, Adriá... I will always carry you with me. You all have shown me that meeting our goals is far easier if we are together and help each other every time. This is not a competition but a cooperative race, and sharing our knowledge is the best that we can do. I hope we continue to meet at Sentits to have a beer and laugh remembering silly memories even if we do not have classes to skip anymore.

I want to thanks my coworkers at INIT too for helping me with this project and others. It has been such a nice experience to work with the team.

And last, but not less important, thank you Aure for showing me that I can do everything that I can think of. You are one of the most ambitious people I have ever met, and your constant support and desire to create new things have been fundamental to me. You have taught me many things, but what I will always remember is that at the end of the day, the most important thing is to be a good person, and I am sure you are.

I would also like to thank my Final Degree Work supervisor, Inmaculada Remolar, for her support during this work and my business practices. 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.

Finally, I would not be here without my family. Thanks to my parents and my brother for believing in me and always supporting my ideas even if they are a madness. If I have something clear in life is that you will always be there to me, and I can not be grateful enough for that. You always encouraged me to study what I really wanted and to be a better artist even if that was risky. You are the best family I could ever think of, and you will always be in my heart even if I end up living far from home. This work is dedicated to you and all the people that have had a significance part in this journey.

Abstract

Videogames are widely known for not only being an entertainment place but for also being capable of helping people in many different ways. This document presents the technical report of the Final Degree Project **"Development of an immersive narrative videogame with a mental health background"**. The work consists in the development of a **3D videogame in Unity** that immerses the players and help them avoid their circumstances, like anxiety or stress, focused on dealing with **mental issues** in a non-explicit way, while at the same time playing a videogame and telling an story. Furthermore, the game is not just a "therapeutic" game, but a fun game that anyone could play.

The idea is to create an original, curious, and friendly appearance to make the ambience and atmosphere of the game even more unique. In order to meet this requirements, firstly there will be an exhaustive investigation about mental issues, specially anxiety, depression and ADHD (Attention-deficit/hyperactivity disorder). On the other hand, the art style will be *toon-shaded* [1] by using *Unity shader* programming. This will help the immersion of the player in the game's world (making it similar to a tale), and for this purpose it is important to develop an interesting story. The game will be a third-person *sandbox* [42], kind of a little open world mainly set in an outside location (an island) where the main character has ended after a shipwreck. But after some time, it will be discovered that the island is not alone. The objectives will be to explore the place, find the islanders and talk with them to solve their problems, reunite elements... and explore the inside of the protagonist that must adapt to this new environment and confront his inner world before being ready to reach the end of the story. This character can have similar aspects with the player and the islanders, so the player can feel more identified.

Keywords: 3D game, third-person, mental health, cartoon style

Contents

List of Figures

Chapter 1

Introduction

Contents

This chapter reflects what has been done during the development of the project, the objectives and the initial motivation. The idea presented in this work is **the development of a video game with a narrative related to mental health.** This will be shown in a subtle way, making use of techniques such as *emergent narrative* (when the story is told by the elements of the environment or not explicitly told). Missions and dialogues will also show the psychological depth of the characters while the player has fun. The game will be a Windows PC *sandbox* called ***"Nebali: The lost island"***. A *sandbox* is a style of video game characterized by minimal limitations, allowing the player to roam and change a virtual world at will and to select tasks in a world to which the player has full access from start to finish [42]. The art style will be *3D toon shading* or *cel shading* [1].

1.1 Work Motivation

Nowadays it is widely accepted that videogames are more than just entertainment and they are considered for most people a new form of art. Nonetheless, there are different kind of games focused on various targets and inside the community it is common that some of them are considered less than others because of this, but nothing could be further from the truth. Even though videogames started only as a way of pure entertainment, it is impossible to affirm that today. The evolution, ramification and influences from other arts like cinema or photography has made them become what they are these years. Games such as *The Last of Us* [16], *Detroit: Become Human* [17] or *The Legend of Zelda* [32] are capable of telling stories that could perfectly be portrayed in a cinema while involving the player in a nice playing experience. And all of this with very different art styles and ambiences.

Regarding these last words, games are also capable of changing people's moods and influence their lives. There are games not so centered on competitiveness or adrenaline, but on creating an immersive world that the player can experiment, explore and build their own story. For example, *Minecraft* [27] is one of the most famous games of all times and the purpose is to explore an almost infinite world of cubes. The player can make whatever he or she wants, so each experience is different from the other.

The work motivation is to explore this sensation of feeling immersed in a game not because physical reasons (for example, virtual reality devices) but because world building and story telling reasons. Furthermore, these kind of experiences can help people with mental issues deal with them by avoiding real life stressful situations such as intrusive repetitive thoughts. Some of the inspirations are *Stardew Valley* [19] and *The Legend of Zelda: Breath of the Wild* [33], where the player has freedom of actions but at the same time there are unique mechanics and a story behind that make players feel inside the world the same way that a person can be immersed in a movie or a book.

In a more personal perspective, the motivation is to prove the artistic value of videogames the same way as other forms of art, besides exploring their capacity of immersiveness through story telling and world building to help with mental problems. The main inspiration is *A short hike* [41] (See Figure 1.1), a simple indie game that goes through the journey of Claire while she has to climb a mountain to know about his mother and at the same time explores and discovers the story of different habitants. This game also has a mental health background that will be explained later in detail.

1.2 Objectives

The main objective of this work is to achieve game psychological immersion by creating a well developed world environment with an interesting story told through the characters to make the players explore, build their own path and feel inside the game. It is important to say that it is not a therapy game, but it is a game that anyone can play and enjoy.

The psychological theme will not only be in the dialogues but also in the game mechanics, ambient, world building, music... All of these will be based on research and scientific papers such as *"The power of videogames"* (V.M. Ortiz, 2021) [35] and *"A short hike reflects my neurodivergence in a way that feels safe"* (T. Ricchardson, 2021) [40].

Figure 1.1: A short hike (Adamgryu, 2019).

- **Dialogues and narrative**. There will be some non-playable characters (NPCs) and each one will have a background story that the player can acknowledge. They will represent typical behaviours like fears, social integration problems or intrusive thoughts. The game must not centre on the suffering of a mental problem, but instead on accept that they will always be there and how to live and cope with them.

- **World building and art style**. A free-to-explore open world is ideal for the objective, but not too big because that would overstimulate the player. An easy-to-navigate environment with a toon shaded art style will make the game more appealing specially for ADHD (Attention-Deficit / Hyperactivity Disorder [6]) people.

- **Game objectives and quests**. One main objective but side optional quests with the NPCs that add lore to the story. This will make it easier to build their own story/path and focus on whatever the player wants at that time. The game must be nonlinear, so the player can decide if he or she prefers to go right to the main thing or to go through all of the secondary paths.

1.3 Environment and Initial State

For this purpose, a wide research about mental health and videogames has been made as an initial state. It is important not only to study the scientific background but also the current videogame scenario related to this theme.

The project will be oriented in general to mental health, but in particular it will be aimed at anxiety, depression and ADHD. According to a 2021 study by Victor M. Ortiz [35], audio-visual content can educate the audience on topics such as mental health in this case and in his article he studies *Celeste* [22] and *Hellblade* [51]. But what are exactly these mental illnesses? According to Dobson (2002) [13], anxiety is defined as an unpleasant physiological state prompted by fear and depression is a multifaceted state that occurs due to loss or a perception of loss. This can affect low self-esteem, personality traits, cognitive dissonance, and the self's perception of situational conflict. With this said, videogames are made to play but playing is not only for entertainment. According to Annetta (2008) [3] these activities include performance, multitasking, judgement, and simulation. And this is a potential for a learning environment or in this case a helping environment, because playing helps develop one's capacity to experiment with their own surroundings as a form of problem-solving.

Pollatos (2009) [38] conducted a study seeking to find out how depression and anxiety affect a person's perception. He discovered that there was a negative correlation between perception and depression, so in other words people with these problems can struggle with accuracy doing tasks. With this information it is known that in the project the world and environment cannot be a messy, difficult world to navigate. Pashtoon (2012) studied copying styles in a group of people with these problems. Some of them were active coping, instrumental support, planning, denial, self-blame, emotional support, humour, and positive re-framing, which can be modified by a psychiatrist.

The main references for the project are *A short hike* and *Celeste*. Ortiz talks about the second one in his paper and references an article by Grayson (2018) [23] that shows how a key developer of the game explains that they developed it to capture their personal experience with depression and anxiety. In the game, a woman named Madeleine climbs a mountain going through platform levels. Sometimes she exhibits symptoms and is up to the player to help her. She meets more characters and trough the dialogues and the player decisions the game writes the storytelling. As a result, the game sends the message to players that they should not do the same as he did, and claims that they learned to manage their mental health by realizing how awful they treated them-self. A lot of players wrote letters to the developers saying that after spending time digesting *Celeste*'s gameplay and story it helped them to cope with their mental health.

In an article written by Clark (2018) [12] she talks about her personal experience. There is a character that represents the manifested negative emotions of Madeleine called *Badeleine* that talked so bad to Madeleine, so she realised how sometimes she treated herself. Madeline slowly learned coping techniques and even made a friend but she never "defeated" her mental health problems, but that was a message about accepting yourself. It is also mentioned that while the game is certainly hard to complete, there are a lot of helpers to assist the player like wings, pass through walls or to execute jump techniques with fewer requirements. The idea is not to make it super easy, but to make it show the difficult of coping with mental health illnesses and struggling with it, but getting help to not frustrate the player (before, it was discussed that people with mental health problems can struggle with perception and other tasks).

***A short hike*, the main inspiration for the project,** is more related to ADHD. Trevor Richardson, who had both anxiety and ADHD, wrote an opinion article in 2021 [40] after playing it. The game has a very clear main objective, reach the top of the mountain to discover "something". But the path is not just that, there are a lot of distractions. So discovering what was on the climb was making him anxious to reach it, but the multiple NPCs that were on the world telling his stories, asking for favours or just waiting for someone to play with them was all that he wanted because of the ADHD. Then, the main goal is particularly short, not like AAA games but it was just a couple of hours to complete, but the dozens of routes that you can follow made him eagle to complete all the game, and that was more than just an hour.

On the other side, anxiety makes the brain overthink about certain scenarios that never happened but could happen. When the player reaches the top of the mountain, she receives a call and Richardson explains that though he spent time with the game anxious about what emotional payoff awaited at the end of the trail, at the end everything was okay and that reminded him of his real life patterns. *A Short Hike* allowed those anxious patterns to play out in a safe and rewarding experience. The most important thing here is that the game remembers the player that everything is okay in that sense, and while of course there are situations that can end bad, it does not make sense to be all in panic when that event did not even happen, and it can go well after all.

Another important thing is that all of these games have two things in common. First is that they are playable games. They are not games for therapist purposes or educational things, instead they are games that can perfectly work as a game itself even if the player does not have any mental issue. But if he or she has one, or has curiosity, it can also help. The second thing is that meanwhile other games tagged as "mental health" are centred on the suffering to recognize their own pain, these games are not. They want to reflect the experiences through dialogues, missions, storytelling and gameplay, accept them and make them play with joy too. Trevor ends his article saying that "*A Short Hike*, while not explicitly a depiction of various neurodivergences, managed to reflect mine in a way that did not center on suffering. Instead, I was able to spend a couple hours in which the wiring of my brain did not inhibit my enjoyment. I do not have access to therapy or medication for the time being, and most days I just have to accept the brain I have and do my best with it. *A Short Hike* seems to accept it just fine and invites it to take a walk for a while, no matter which direction in which it wants to wander."

Chapter 2

Planning and resources evaluation

Contents

This chapter explains the planning followed during the project and the resources needed to develop it.

2.1 Planning

Given the nature of the work, research and planning was a essential part of it. The original plan was made in February when the Game Design document was developed. Through the months it has been some changes, but the basics concepts stayed unaltered. The work was divided into tasks and *Trello* (See figure 2.1) was used for the planning. An initial planning was made using a table on the technical purpose, and then that was transferred to a Gantt chart (See figure 2.2 and table 2.1). The main tasks are the following:

- **Investigate and research**. Like it is mentioned before, the research is one of the pillars of this work. In this initial part, a series of scientific papers that talked about video games and mental health were investigated and compiled. From this, certain ideas and concepts were deduced that the game had to have in order for it to work.
- **Initial design work: GDD (Game Design Document).** In this part the development of the main idea of the game begins. Starting from the previous

research, the main idea for the story was developed as well as the game's characters, objectives, and graphic style. A basic map of the world and some artistic concepts for characters and stage elements were also made using Inkarnate and Krita.

- **Story, objectives and game design**. It consisted of developing the GDD concepts in depth. The main story and dialogues were written as well as all the character traits and designs. In this stage the main quest and the secondary quests were developed. The design of the scenario (world building) and the different parts of it were also made.

- **3D modelling**. Using Blender and the concepts for the main idea, most of the game objects were modelled. This part took a lot of the project time. Both the game characters (main and NPCs) and the elements of the environment (houses, trees, objects, etc) were modelled, textured and animated.

- **World building**. The Unity Terrain tool was very useful for this part. Some investigation about it was made and then a scenario for the game was created using the GDD map as a reference. In this part the main work was the lighting, ambience, terrain generation and textures, *skyboxes*, wind, grass, and the placement of all the 3D models and NPCs.

- **Shaders**. This may be one of the most technical stages of the project. From the beginning the art style was very clear, toon shading. To obtain this cartoon look, a series of shaders were developed that also allowed to control aspects such as the outline and the amount of tones it could capture. All of this was developed with ***Unity's Shader Graph tool*** and with the help of tutorials on Unity's own page. A lot of investigation was needed. The result was a colorful and friendly setting, similar to a fairy tale atmosphere.

- **Programming**. Along with the 3D modelling part, this was the other stage that took most of the time. Because the dialogues were going to be an important part of the project, it was decided to develop a tool to make it more clear and easier to implement and scale them. Rather than writing them using string arrays or something similar, a tool was developed using ***Unity Node Graph API*** [5] (Application Programming Interfaces) and some tutorials. The result was an incredible useful tool. It uses nodes that have information about the character name and the dialogue, and they are connected between each other. This allowed the dialogues to have ramifications and multiple choice options, and it was all packed into one asset file that can be loaded and saved from the tool. The camera, player movement, NPCs scripts, enemies, objectives, quests and dialogue management were the main programming work. It was all made in **Unity** using *C#* and *Visual Studio* [28].

- **Bug fixes and testing**. After all was implemented, some bug fixing and testing was made to ensure everything was okay.

- **Memory and presentation**. It includes the development of this report, the presentation and the preparation for the final assessment.

Activity	**Expected amount of time**
Investigate and research (Mental health, Unity shaders, Environmental design for immersion. . .). GDD.	20h
Design the NPCs, draw concept arts, write their character sheet with the personality traits, etc.	10h
Write the story, define the main objectives and goals, write the script and dialogues. Game design.	10h
3D modelling, texturing and 3D animations of the characters.	40h
Modelling the environment, props. . .	40h
Creating the environment in Unity (lighting, Unity terrain, wind, water. . .).	30h
Develop the main shader and other shaders using nodes.	15h
Programming (Dialogues, quests, IA, scenes. . .).	120h
Ambient music and sound effects.	5h
Testing and bug fixes.	15h
Write the memory and elaborate the exposition.	40h
Total	**325h**

Table 2.1: Time schedule.

2.2 Resource Evaluation

The idea of the project is to be as free as possible, so it was intentional that all of the software used is open source. The models are most of them made by the author, but the few models taken from the Internet are also open source and referenced at the end of this report.

It is a project made only by one person, so no human resources were needed. However, this is a demo of what could be an entire game, and for that maybe it could be necessary to include more people into it. With the project it is demonstrated that one person can develop a complete game by his own, but it is really difficult and time consuming for a bigger project.

- **Hardware**. The hardware in this case is the PC and its components with a total price of €1.020,40. These are the specifications:

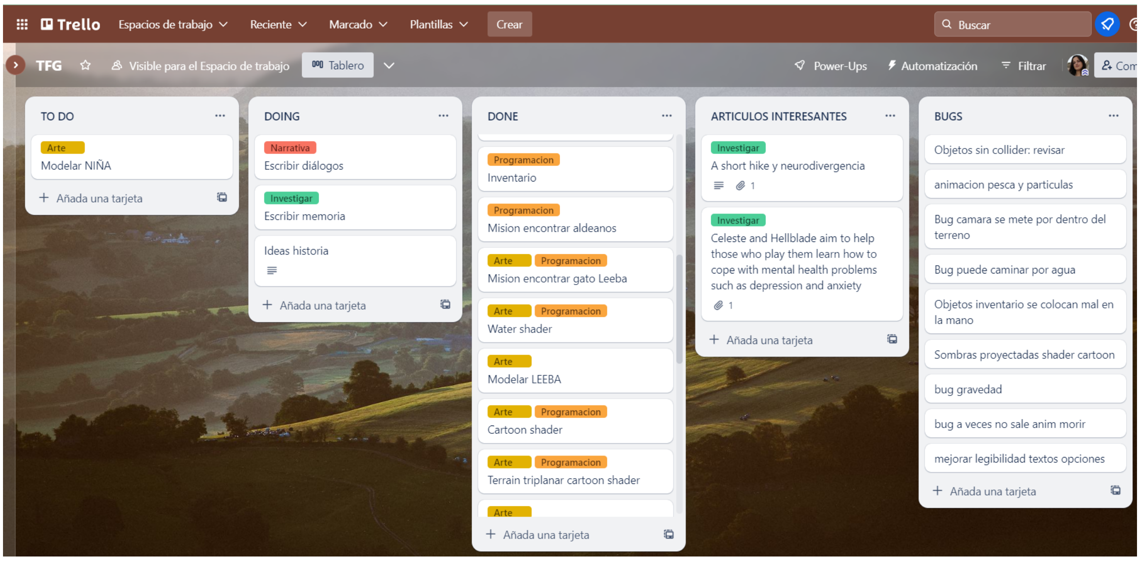
  - Laptop model: Asus TUF Gaming FX504 (€990).
  - CPU: Intel i7 8750H 12 CPUs
  - GPU: NVidia GeForce GTX 1060 4GB
  - RAM: 8GB
  - Memory: 1TB HDD and 256GB SSD
  - Optical mouse, Sailor Moon edition (€17,50).
  - Headphones Bluetooth Qilive (€12,90).

- **Software**. All of the software used is intended to be open source as long as possible.

  - **Windows 10 Pro**. Operating system. Not free but it is the standard. (€49,99)
  - **Unity Engine**. Free for personal use. It was chosen among other alternatives like Unreal engine because it is the one used during the degree and it is very flexible. Furthermore, the project was not intended to use a super realistic style, so Unity gave more options for toon shading development and was more than enough.
  - **Visual Studio 2020**. The main tool for programming. It is also free and it is the standard for Unity.
  - **Blender 3.0**. On the other hand, the main tool for 3D modelling. It is also free open source software and is incredible useful for modelling, texturing and animation.
  - **Clip Studio Paint**. Used for 2D illustration, like concept arts and sprites designs.
  - **Mixamo**. Made by Adobe, it is a free web based tool that contains thousands of animations and characters, and also allows the users to upload their own models and auto rig them. All of the animations in the game are from this website.
  - **Github**. Version control system used for storing the game stages.
  - **Trello**. Management of the workflow for the project.
  - **Overleaf**. Online tool useful for the memory development in Latex.
  - **Lucid.app**. Web used for diagrams like Gantt and cases of use.

- **Other resources**. If this project was developed by a company (or self-employed), the cost of human resources would be measured in person/month. If the project costs approximately 350 hours and the workweek is around 37,5 hours, the project will be completed in 9,33 weeks. If a month consists of approximately 4,35 weeks, this would be 2,14 months. The average salary of a Junior developer in Spain is €1494/month [24], that is a cost of around €2400. In conclusion, human resources would be 2,14 x €2.400 = **€5.136.** In Castellón, the rent of a co-working local is around €180 per month. (**€360** for 2 months). Electricity, water and internet connection would be included.

On the other hand, the software is mostly free. The hardware and software specified before costs around €1070. When calculating the expenses for equipment, there is a gradual loss of value. If the equipment has an estimated useful life of around 5 years and it has a value of €1,070, the yearly **depreciation** cost would amount to €200, so for the 2.14-month period it would be approximately **€53**. In conclusion, the total development cost would be €5,136 + €360 + €53 = €5549. If the **contingency** costs are taken into account, which are typically 10% of the total cost, it would be €5549 + €554,9 = **€6103.90**.



TO DO

Arte
Modelar NIÑA

+ Añada una tarjeta

DOING

Narrativa
Escribir diálogos

Investigar
Escribir memoria

Ideas historia

+ Añada una tarjeta

DONE

Programacion
Inventario

Programacion
Mision encontrar aldeanos

Arte Programacion
Mision encontrar gato Leeba

Arte Programacion
Water shader

Arte
Modelar LEEBA

Arte Programacion
Cartoon shader

Arte Programacion
Terrain triplanar cartoon shader

Arte

+ Añada una tarjeta

ARTICULOS INTERESANTES

Investigar
A short hike y neurodivergencia
1

Investigar
Celeste and Hellblade aim to help those who play them learn how to cope with mental health problems such as depression and anxiety
1

+ Añada una tarjeta

BUGS

Objetos sin collider: revisar

animacion pesca y particulas

Bug camara se mete por dentro del terreno

Bug puede caminar por agua

Objetos inventario se colocan mal en la mano

Sombras proyectadas shader cartoon

bug gravedad

bug a veces no sale anim morir

mejorar legibilidad textos opciones

+ Añada una tarjeta

Figure 2.1: Trello workspace.

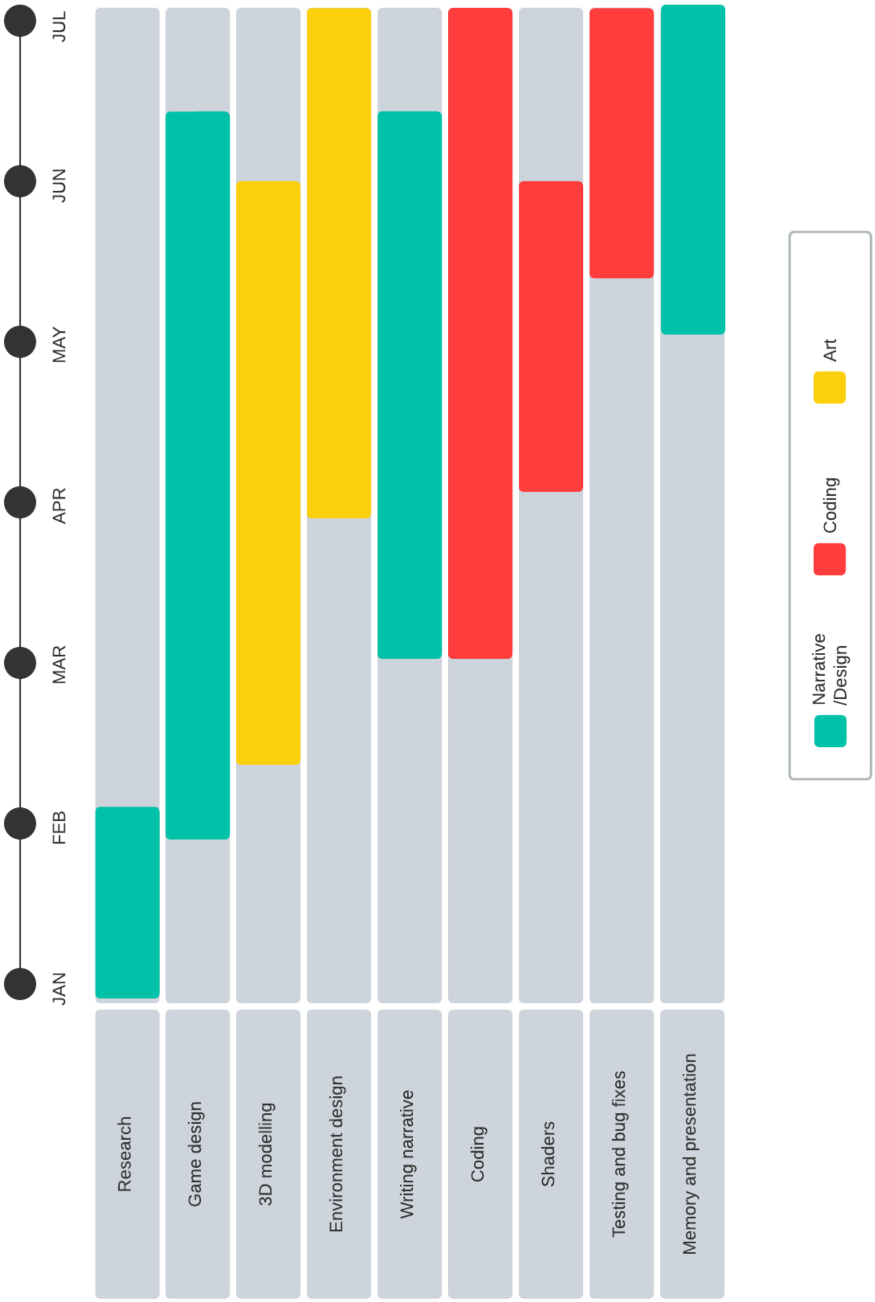


Figure 2.2: Gantt chart (made with Lucid.app)

Chapter 3

System Analysis and Design

Contents

3.1 Requirement Analysis

In this section it will be detailed the functional and non-functional requirements of the project as well as some diagrams.

3.1.1 Functional Requirements

R1. The player can start the game by clicking the “Play” button.
R2. The player can quit the game by clicking the “Quit” button.
R3. The player can learn the controls by clicking the “Controls” button.
R4. The player can learn about the development by clicking the "Credits" button.
R5. The player can move the character using WASD or arrow keys.
R6. The player can move the camera by moving around the mouse.
R7. The player can control the camera by moving the mouse.
R8. The player can run by pressing “Shift”.
R9. The player can interact with other characters pressing the “Space” key.
R10. The player can read dialogues and choose options by clicking on them in the screen.

R11. The player can recollect objects by passing through them.
R12. The player can open the inventory by clicking the "Q" button and can choose an object with the mouse.
R13. The player can use the object selected using the "E" key.
R14. In some moments, the player can attack enemies using the sword object.
R15. The player can open the pause menu during the game by pressing the “Escape” key.
R16. When the pause menu is open, the player can return to the game by clicking the “Resume” button.
R17. When the pause menu is open, the player can go to the main menu by clicking the “Main menu” button.
R18. When the pause menu is open, the player can go to the options by clicking the “Options” button.
R19. When the options menu is open, the player can change the music and effects volume using the "Music volume/SFX volume" options.
R20. The enemies can walk around the map (using Unity NavMesh).
R21. The enemies can detect the player when they are close enough and chase after them.
R22. The enemies can send the player to the initial state of the level when they chase them.
R23. The enemies return to their patrol path if the player escapes from them.

3.1.2 Non-functional Requirements

R24. The game is playable in Windows PC platform.
R25. The game is in 3D, simple not realistic style.
R26. The artistic style is cartoon shading.
R27. The UI is simple and clean, not obstructing the player’s vision and only showing the character’s hearts and messages for certain actions completed.
R28. The controls and mechanics are simple and easy to learn.
R29. The story should be learned by the dialogues and quests, and should have a mental health background.
R30. The music follows the aesthetics of the world and makes it more immersive.
R31. The entire ambience and atmosphere should be consistent and follow the style of the game.

3.2 System Design

The best way to present the logical design of the game is through a flow chart (see Figure 3.1).

3.3 System Architecture

This video game is made with the Unity engine, the version 2020.3.27f1 specifically. The minimum system requirements for running games with this engine are: At least Windows 7 operating system, 64-bit versions only; or macOS 10.13; CPU with SSE2 instruction set support and graphics card with DX10 capabilities.

All these requirements have been taken from the Unity documentation [55].

3.4 Interface Design

The interface is the same through all the game except for the conversation parts. There is always a heart counter at the top left screen that informs the player about how much life it has left. When the player approaches a character that has a available dialogue, a talk icon will appear on top of their head. It will have an animation when it appears, disappears and stays to make it more visually attractive.

When the player interacts with a character to start a dialogue, a panel appears on the bottom of the screen. There are texts for the character name and the dialogue content, and on the right there can be options to choose or just a Continue button if there are no options.

When the player interacts with an object (for example adding it to the inventory) or gets/completes a mission, a message will appear on the top-right part of the screen a few seconds to inform the player.

There is also an inventory menu, a diary and a pause menu. There is no much interface items because the world is very important and it is intended to not obstruct the player vision if it is possible. It is more detailed in section 4.3.6.

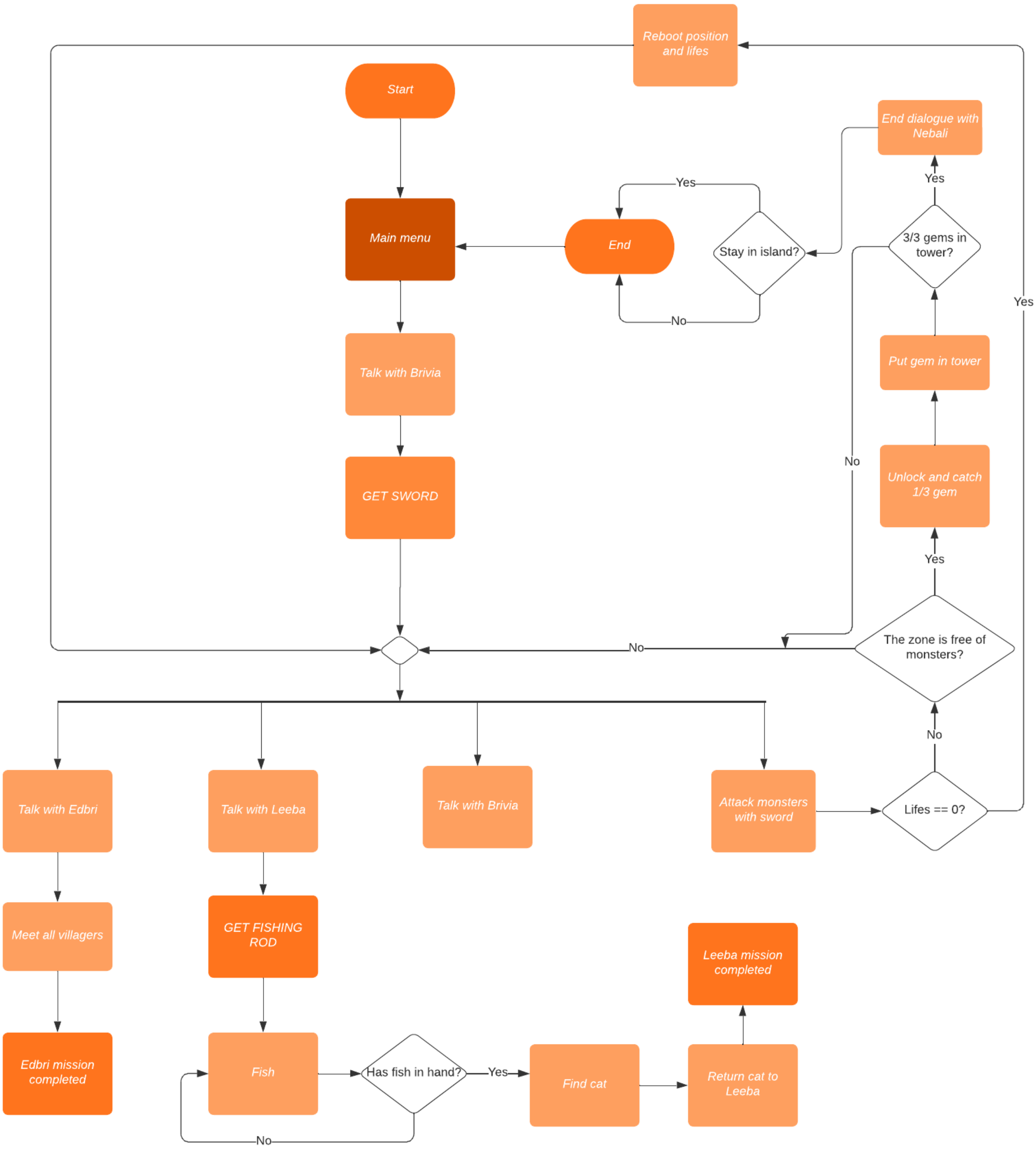


Figure 3.1: Game flow chart.

Chapter 4

Work Development and Results

Contents

4.1 Work Development

This section will explain the entire development process of the game. After all the research and planning, it was time for getting all of it together. The progress, the evolution, the methods used and the changes that have had to be made are going to be analyzed.

4.2 Game Design Document (GDD)

First of all, a GDD was developed to clarify the initial ideas. This was changed and modified during all the process according to the projects needs, but the basic concepts stayed unaltered.

4.2.1 Introduction

Title: Nebali: The lost island
Platform: Windows PC
Genre: Sandbox, third person, casual narrative game
Target audience: People with anxiety, depression or ADHD
Art style: 3D Toon or cell shading

The project is going to be a third person sandbox game set on an island where the main character was shipwrecked. At first, he will be alone but suddenly a small creature will attack him. Luckily, someone will appear and help him, and he will discover that he is not alone as little islanders are living close to the place. They will help him along his new path in this world, but he will need to deal with this completely new situation, adapt to the place and help the villagers to defend his village from these creatures. The narrative is about dealing with mental issues. New situations mixed with anxiety, depression or ADHD can entail feeling of loneliness, difficulties to adapt, isolation... because getting out of a comfort zone is always difficult. The little creatures represent intrusive thoughts that he will need to combat.

The player can talk with the islanders and do missions for them or just learn about their traits, personalities or the island story. They will tell him situations that the player can identify with and ask about advice, so the player can think about it. The **main objective** is to explore the island and destroy the enemies, but there are secondary paths with each character. There are 3 zones, and **when each one is free of enemies, a magical gem appears in the map. The player must collect all of them and put them on a tower to invoke a god (named Nebali). The god can grant a wish if the three gems are placed in the tower.**

After the main objective is completed, the game is still playable. It is important that the game is not only for people with mental problems, and it is not centred on suffering but on how to accept it and deal with it the best possible way. The immersive part of the project is essential, because the game intends to ponder on the topic while having a fun time and avoiding real life thoughts that can make people overthink in excess. Mental health issues are not mentioned explicitly, but is dealt with in plot, mechanics, or story.

4.2.2 Contributions

After analyzing other games, articles and papers (see section 1.3), these are the conclusions:

- The game should have an **"open world" scenario** but not super big to not overstimulate the player.
- The environment should not be messy, **easy to navigate.**

- The game must have **one main objective but side optional quests** with the NPCs (Non Playable Characters) that add lore to the story

- The NPCs should tell their micro fictions to the player related to situations that they may have and how to cope with them. (Example: In *A short hike* there is an habitant that appears in a forest. He is painting a landscape and the player tells him that it is really nice, but he is not satisfied because he is very perfectionist. He appears in different points of the game, and he is always stressed and anxious because he compares a lot with other painters and thinks that he will never be that good. The player tells him that it is really nice as it is, and that he will be improving with practice so there is no need to compare with others that are not in the same stage of learning as him.)

- The game must be nonlinear, so the player can decide if he or she prefers to go right to the main thing or to go trough all of the secondary paths.

- The game must not centre on the suffering of a mental problem, but instead on accept that they will always be there and how to live and cope with them.

4.2.3 The story

First part
The main character, controlled by the player, is shipwrecked on the island and finds himself alone in an unknown environment. He can only access a small part of the island. This will act as a tutorial where he will learn how to move, how to pick up items and add them to inventory and how to use them. Suddenly, some little bugs appear that are going to hurt him, but a character appears who charges it and enters. The small islander tells the character that there is a town with a few more of them in another part of the island and asks to follow him. From there he can go to the village, to discover that he is not alone.

During the game
The habitants are small humanoids that the character has never seen before. At first, it is difficult to communicate and they are a little reluctant, but they end teaching him basic things that helps him integrate. They give him a sword to fight the bugs and tell him that they are called the **Boo's**. They are infecting the island so he can help with it. The main objective of the game is to destroy all of them. But that is not the only thing, because the islanders have side optional quests that help to understand their problems, the lore, etc. The bugs are in 3 different areas. Each area can be accessed in any order the player likes.

The game is a metaphor of coming to a new place outside of a comfort zone and not knowing how to act or how to fit in. If they also have a mental health problem, they tend to shut themselves away and not face the situation. Therefore, the villagers will be the ones who help him cope and integrate both by teaching him basic skills and by teaching him their culture to make the game more interesting. In addition, the little bugs represent the intrusive thoughts that appear in certain areas of the game that he must discover. In the end, the main quest is completed when you destroy them and it represents having successfully dealt with anxiety and intrusive thoughts and being able to adapt and integrate into an unfamiliar place.

4.2.4 Mechanics

- Walk with WASD or the arrows.
- Recollect objects passing through, use objects with E, open inventory with Q.
- Talk with characters with Space.
- Choose dialogue options or pass dialogues with left click.

4.2.5 Characters

The Boo's

These are the bugs infecting the island. Depending on the zone, they have different abilities. They represent intrusive thoughts, both from the islanders and from the player himself. The player can end with them with a sword. Inspired by Studio Ghibli's *Susuwataris* that appear in *Spirited Away* (2001) and *My neighbour Totoro* (1988) by Hayao Miyazaki. (See Figure 4.1)

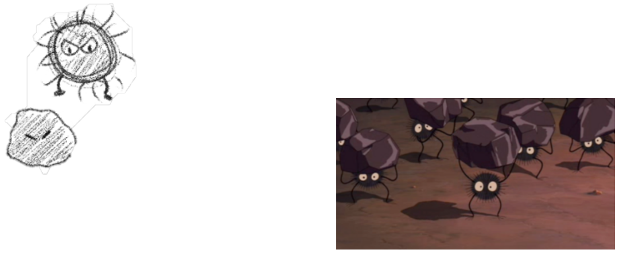


Figure 4.1: The Boo's (left) and *"Susuwataris"* (right).

Mateo

See figure 4.2. He is the main character. Mateo is 18 years old and comes from a small country town. He is a bit introvert and has never been especially comfortable with people that are outside his circle. However, he is really loyal and careful about his close relatives and friends. His family has a farm so since he was a kid he knows basic things about animals, agriculture and surviving but he does not want to continue being a farmer all his life. He does not know how to tell that to his traditional family so he just stays doing what they do.

Brivia, the fighter

See figure 4.3. She is known as the fighter of the island. She makes her own arms and sells them, and knows everything about the Boo's. Brivia can give advice to the player about the bugs, and she will give him the initial sword, like swords or shields. She lives in a house in the forest. If the player talks with her, she will talk more about herself.

Brivia is extrovert and very friendly, and she is very nervous. She appears to be strong and invincible, but in the end she is also scared and worried about things. She was all alone in the island when she found the others in the village, so she was the last in the group. Because she grew up alone, she knows perfectly how to survive without anyone else, but that makes her difficult sometimes to ask for help when she needs it. On the other hand, she learned social rules later than the others so sometimes she acts inadequate. She has so much energy that she gets frustrated sometimes because she does not know how to deal with that, so that is why she ended up finding her favourite hobby, to explore and to fight bugs.

Edbri, the mayor

See figure 4.3. He lives in the village. He is very charismatic and outgoing. If you talk with him, he will give you tips about the game. Sometimes he feels overwhelmed about the fact that he is the mayor, because he feels that he is not good enough to be the leader or to manage a whole town. The character will tell him that he is good enough because the others tell so, and he should care that much about others' opinions and compare that much.

Leeba, the old fisher

See figure 4.3. He is an old man that lives near to the lake. He spends his days fishing and sitting outside of his cabin. He does not care much about anything and usually ignores everyone. If the player talks with him, he will be annoying at first but eventually he will confess that sometimes he feels alone. The Boo's invaded his place to fish in front of his house.

For the secondary quest, if the player finds a cat that is lost and returns that to him, he will appreciate you a lot because the cat was a good friend of him when he went there to go fishing. He will give you a fishing rod and teach him how to use it. If the player catches a fish, finds the cat and holds a fish, it will follow the player, that way he can give the cat to Leeba.

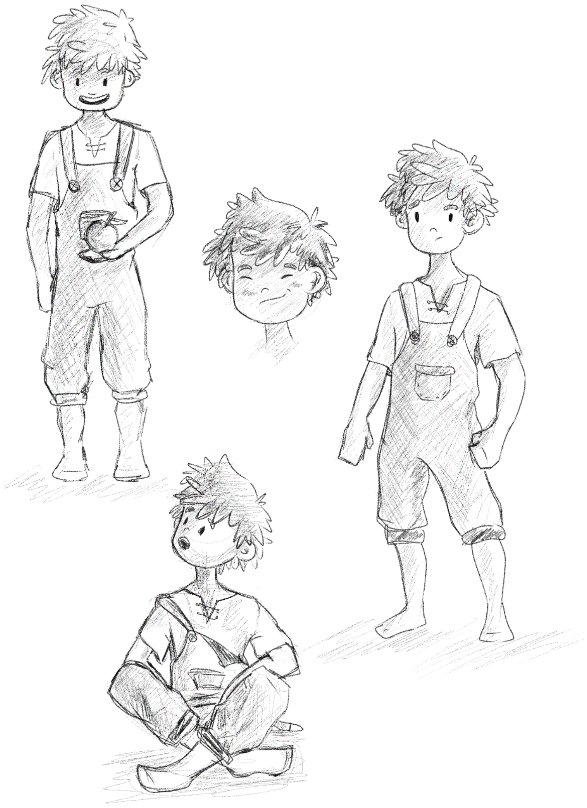


Figure 4.2: Mateo concept

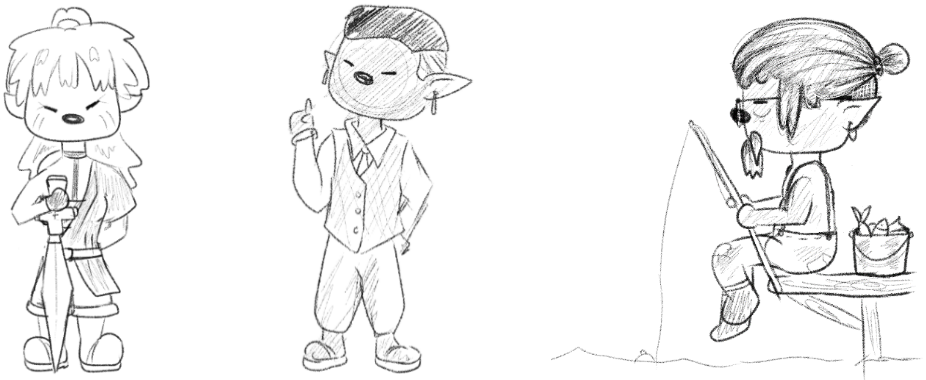


Figure 4.3: From left to right: Brivia, Edbri and Leeba concepts.

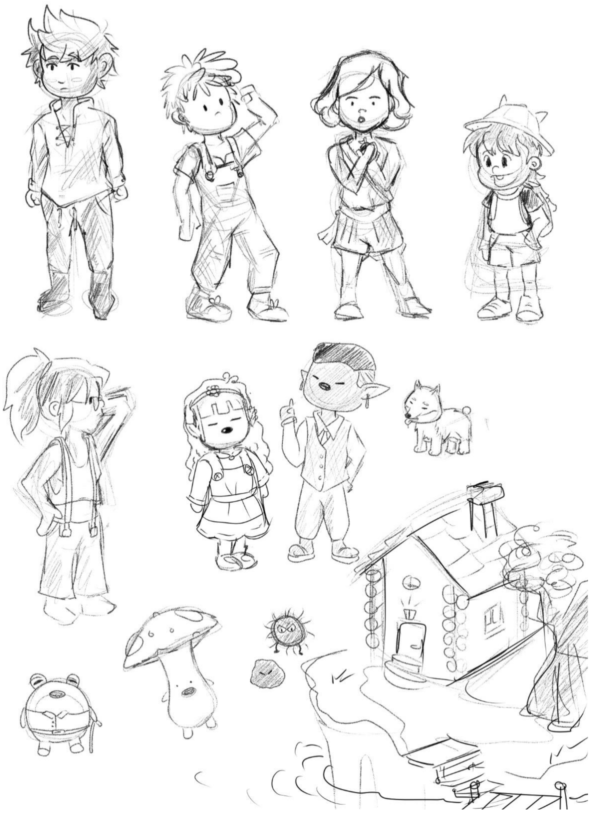


Figure 4.4: Some concept arts.

4.2.6 Levels and game progress

The “levels” are zones that the player needs to clean up (destroy all the Boo’s). They can be accessed in any order. If the player “dies”, Mateo passes out and appears at the beginning of the level/zone. A gem is unlocked for each liberated area. If the player places all 3 gems in the temple tower, the god Nebali will appear and grant Mateo’s final wish, which can be to stay on the island or return home.

The zones with enemies are the forest zone, the town, and the farm. On the other hand, each character has an optional secondary quest.

4.2.7 Graphics

Main graphic concepts

The graphic style is cartoon, using a toon shader with *Unity Shader Graph with URP* [5]. This way the game is intended to be relaxing and immersive, so the player feels involved into the story and the ambient and avoids outside stimulus.

The models will be developed in **Blender** [8] and the game and shaders in **Unity** [55]. The interface and UI/UX design will be similar to a sketchy style. Some of the games that will be used as a reference are *The Adventure Pals* (Massive Monster, 2018) [29], *Moonlighter* (Digital Sun, 2018) [50], *Life is Strange* (Dontnod Entertainment, 2015) [18] or *Miracle Merchant* (Arnold Rauers, 2017). [39]. See more graphic related images in Figures 4.4, and 4.5.

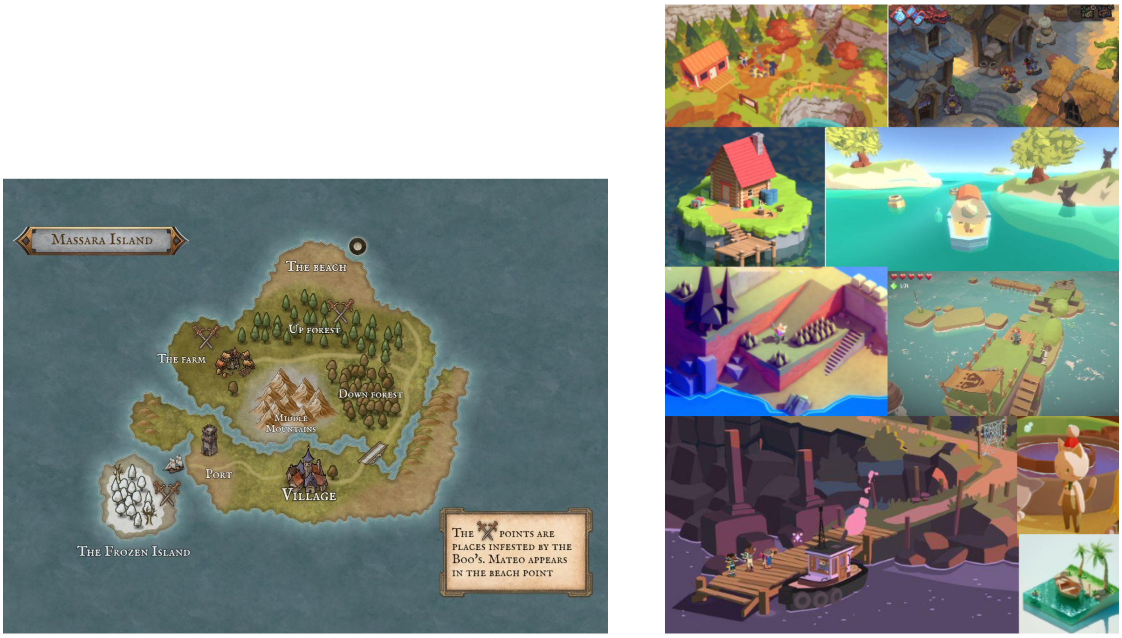


Figure 4.5: World map (left, made with Inkarnate) and moodboard (right).

4.2.8 Sound design

The music needs to serve for the immersive part of the game. Like the games used as a reference, it will be calming and relaxing but at the same time it should make the player interested in the game so he/she want to explore and continue playing.

The sound effects should be related with the game graphic style. It is cartoon, so they will not be very realistic, like the effects of games similar to *Animal Crossing*.

Examples of soundtracks could be *A Short Hike* [41], *Alba: A wildlife adventure* [56], *Stardew Valley* [19], *Animal Crossing New Leaf* [31] or *The Legend of Zelda: Skyward Sword* [34].

4.3 The art style: Unity shaders and low poly modelling

The first thing done to find the art style of the game was to grab a pencil and start drawing. After making the sketches appended in the GDD (section 4.2) it was decided that the style should be friendly and simple. The perfect style was cartoon shading. The 3D models are most of them low poly.

4.3.1 Modelling and animation

Using the concept art as the primary reference, the first thing modelled was the main character: Mateo (see Figure 4.6). He is the only one with "human" form. **Blender** was used to model all the objects of the game.

Following the art style, the models are all stylized and mainly low poly. This way, the entire atmosphere looks more "cartoony". Some of the techniques used for character modelling are the following:

- **Hair with curves.** This technique is perfect for stylized hair modelling, but if it is realistic it is better to use the hair particle system. With a *Bézier curve* the basic shape of the hair is defined, and with *Path* objects the hair stripes are positioned on the head. In this manner, each stripe has segments and they can be controlled individually (tilt, width...).
- **Skin modifier.** Useful for making the hands.
- **Shrinkwrap modifier.** This modifier wraps a mesh around another one. It is used, for example, for making details on the cloth or the eyebrows.
- **Cloth.** The *Cloth* modifier is used for simulating realistic clothes physics. First, the patterns were made on a plane using real patterns as a reference. With some subdivisions and topology work, all of them were connected using edges as seams. The character uses a *Collision* modifier and when the simulation starts, all the seams join the different pieces together along the body. With some try and failure work, this was the technique for most of the characters clothing.

The islanders all derive from the same basic shape (see Figure 4.7). They all have the same body, eyes and nose. The changes are on the clothes and the hair. Another technique used for the hair (for example, Brivia's hair) is using primitives like a cube with subdivisions and then apply a *Subdivision surface* modifier. This way is easier to control the shape of the stripe using the basic primitive, and for the final result the modifier makes it smooth.

For the animation, all of them are made on *Mixamo* [2]. It is a website owned by Adobe that allows the user to explore both models and animations, auto-rig their models and animate them, all of this for free. All of the project animations are downloaded from this web and modified in Blender for better results.

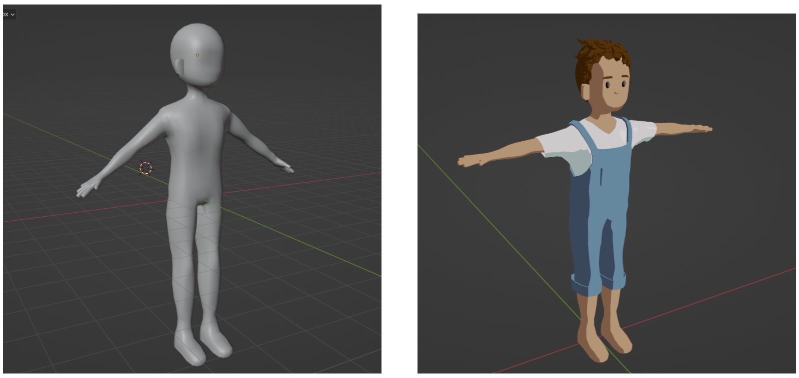


Figure 4.6: Process of Mateo modelling. Blender.

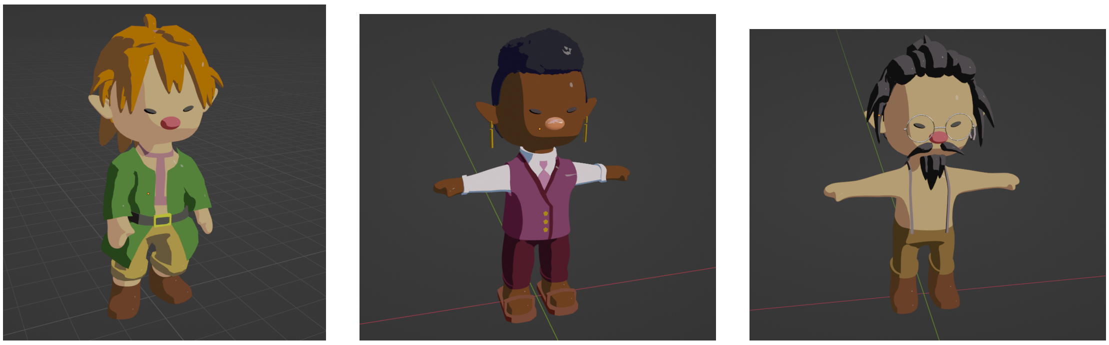


Figure 4.7: Brivia, Edbri and Leeba models in Blender

For the objects, most of them are also modelled for the project. The same techniques are used, for example the *Skin* modifier for making trees. Other modifiers used are *Array* for the house tiles, *Booleans* for windows and holes on the meshes, and *Vertex bevel* for making objects like wood and stones look more stylized. The stones use a procedural technique using *Displacement* and a *Voronoi* texture. Changing the values, this is also used for making Edbri's extra wavy hair. See the results in Figures 4.8 and 4.9.

4.3.2 Toon shader in Unity URP Shader Graph

Toon or *Cel shading* [1] has been widely used in videogame industry, but it is common to think that a videogame has better graphics if they are more realistic. However, this only had sense when the industry was in early development. Each new console was advertised as a new technological era with a lot more power and that considerably impacted people (for example the transition from 2D to 3D games). Nevertheless, last years the technological capacities of consoles and computers are not that different between generations. Graphic realism has reached his peak, and is not that impressive when showing a new game. That is why it is starting to appear the tendency of studying new art directions.

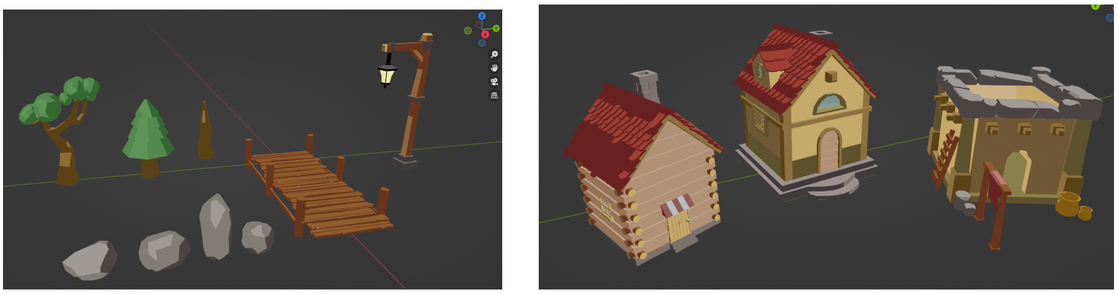


Figure 4.8: Some objects modeled in Blender.

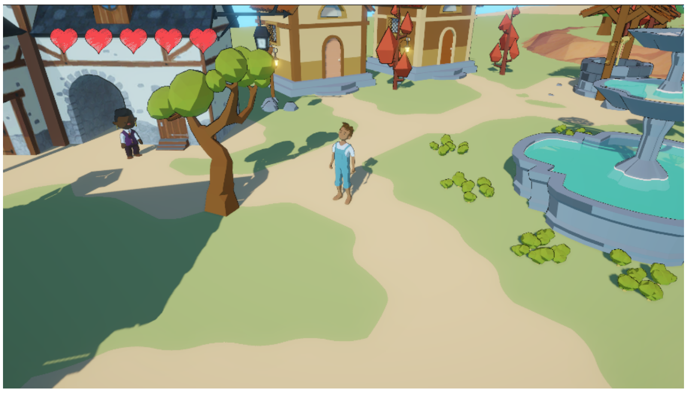


Figure 4.9: The town, initial prototype.

One example is *The Legend of Zelda: Breath of the Wild* (Nintendo, 2017) (See Figure 4.10).

Graphic realism does not equal better quality, but the concordance between an artistic style and a certain type of game. Besides, trying other art styles can lead a game to experiment more and be original, differentiate themselves from others. That is why this project decided on the toon style.

4.3.3 Cartoon shader with outline.

To get the wanted results, it was essential to investigate about Unity **URP Shader graph** [52]. It consists of a tool included in the engine (Universal Render Pipeline version) that allows the developer to create useful results for creating materials. It uses nodes connected between them and variables that can be set in the editor. After investigating about the tool, some tutorials about cartoon shaders where found on YouTube and blogs. One of the best resources was a video called "*How we built the toon shading | Open Projects Devlog*" from the Unity *YouTube* page [54]. They explained in full detail

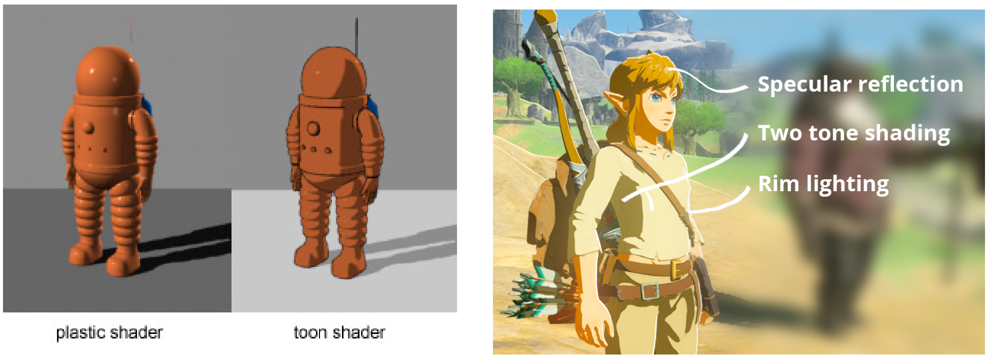


Figure 4.10: Example of standard vs. toon shading (left). The Legend of Zelda: BOTW toon shading (right).

how they did it for a game, and that was incredible useful for this project.

For the purpose of knowing in general aspects how the shader has been created, the main concepts will be explained so that they can be easily understood. Firstly, the shader gets the main light choosing the strongest directional light in the scene. Using a custom function, it gets all the data from this light (direction, color and shadow map). This is possible because URP has a function called *GetMainLight()*. This is the **ambient illumination.**

After that, it is necessary to get the normal vector of the surface and the light direction. If you get the dot product between these vectors and clamp them between 0 and 1 (normalize the vectors), we obtain the **diffuse illumination.** We also multiply it by the light color and the shadow attenuation to get it completed.

This way, the basic custom lighting model is built. To accept light from other sources (for example, a point light), the shader uses a custom function. URP has a function called *GetAdditionalLightsCount()* to loop through all of them and use the *GetAdditionalLight()* to get the information from each one, same as the main light. The shader now interacts with **additional lights.**

One of the most complex parts is getting the **real time baked shadows.** A technique called mixed lighting mode is used to get the shadows. Basically, it uses the shadow attenuation from the main light and multiplies it for a node called *Baked GI.*

To get the most interesting part, the toon shaded look, the shader uses a node called *Simple Gradient* that creates a custom color ramp by reading the main light data. This way, instead of getting thousands of values when the light hits an object, we restrict them to a fixed number of colors to achieve hard and clean divisions. The only necessary thing to do is to change the colors in the ramp (from clear color to dark color) to create different materials. The result is really good with this configuration. It was modified to also have a black outline because that way fitted better the game comic style. The results are in Figure 4.11.

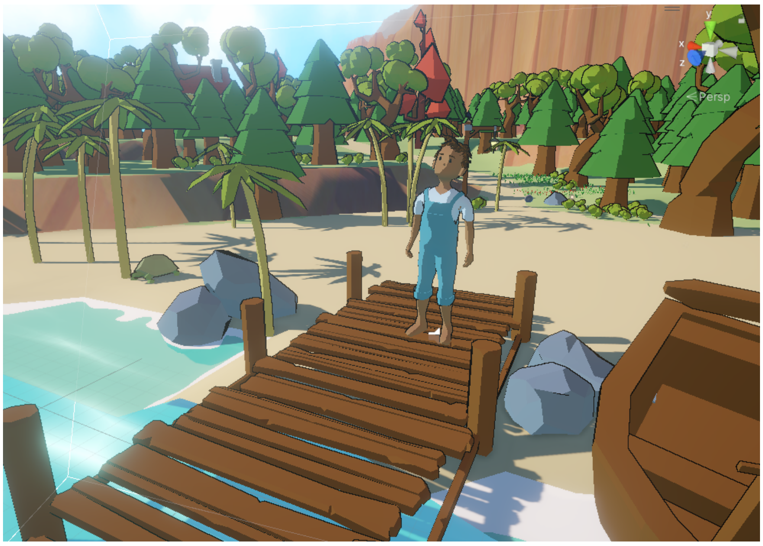


Figure 4.11: Main scene after toon shading.

4.3.4 Toon water shader.

Another shader made for the project was the **Cartoon water shader**. The game is set on an island, so it was necessary to make this. The idea for getting the cartoon look is to get something similar to the game *The Legend of Zelda: The Wind Waker* (Nintendo, 2002) (See Figure 4.12). It should have plain blue color, white waves in movement and plain white bubbles (foam) that interacted with objects that touched the water. This way, the water would only generate foam wrapping objects or at the end of the sea. To get this, the best tutorial was *"How to make a Water Shader with FOAM In Unity with URP"* by Ghost Studios [48]. The main concepts are the following:

First, the main idea is to use a *Voronoi texture* [9] for the *Fragment shader* (it is a node in URP). This procedural texture is similar to **water waves**, so it is really useful. Multiplying it by a *Time* node and a speed variable that can be set in the editor, when a base color is chosen it already looks like water. With some other variables like **ripple density** in the *Voronoi* node, **ripple slimness** in a *Power* node to control intensity and multiplying it by a ripple color, all of the typical water lines (ripples) are created. It is only necessary to use an *Add* node to add the lines to the base color. To create more movement, the shader uses a node called *Radial Shear* that creates a radial like movement. Adding this to the created before, the ripples are more dynamic.

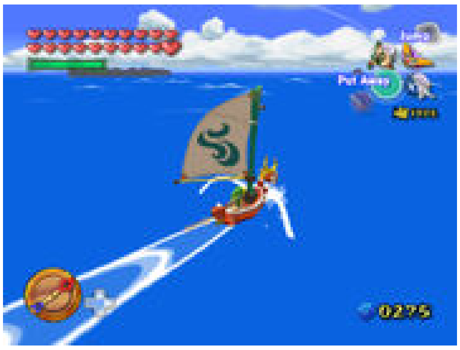


Figure 4.12: Cartoon water in The Legend of Zelda: The Wind Waker.

The problem is that when this shader is added to a plane, some lines are moving but the plane is standing still. The next step is to create **waves** using a Vertex shader. For this it uses a *Gradient noise* texture with a *Tiling and offset* node (multiplied by *Time*). If we multiply this by the normal vector of the plane and add it to the position *vertex shader* it looks like waves. It is also controlled by a variable in the editor.

To create the **foam**, it is necessary to use a node called *Depth Fade* that does not exist in this version of Unity. For this reason, it was necessary to create a *subgraph* and write a custom script to get this right. In simple words, it gets the depth of an object starting from the camera. Using this and a gradient noise again the foams look just like intended and interact with every object that touches the water. Results in Figure 4.13.

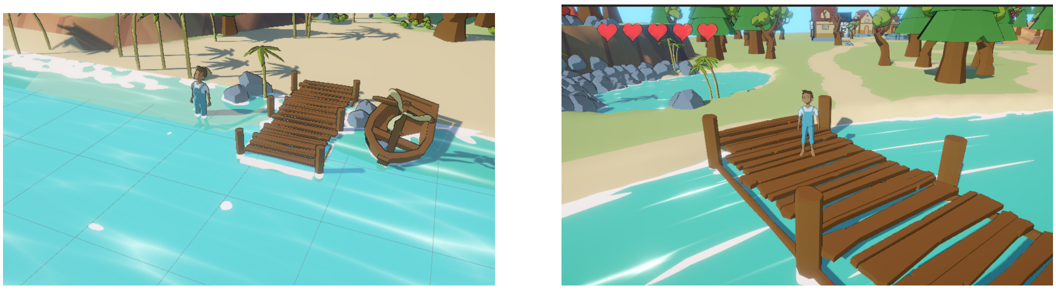


Figure 4.13: Water shader implemented in game.

4.3.5 The environment

The terrain has been one of the most complex things to think about in the project. Since it is set on an outside location, the environment was a priority. Unity terrain is a useful tool to make wide locations. It provides the user with height controls, texture painting, details placement, trees placement and optimization features like *LOD's* (Level Of Distance: depending of the distance of the camera it renders a more or less detailed mesh of an object. If it is far away from the player, it does not render).

It was the perfect solution for the game, however there was a problem. It is made for realistic projects, so the shading is not toon shaded. One solution was to modify the *Terrain Lit Shader* from Unity, but it was excessively complex and there was a lack of knowledge in that area. After considerable research, the best option was to active an option that changed the terrain textures transitions to be hard instead of smooth shaded. The limit is that it can only handle four different textures with this option. The textures used are green for the grass, the path, grey rocks and walls.

After that, the original map sketch was the reference (see Figure 4.5, left) and it was built following it. The next work was *worldbuilding*: place all the models, find a cartoon *skybox* (the sky of the game), compose the lighting and make all of it interesting.

The vast majority of the models where made by the developer, but some of the nature decorative models where downloaded to expedite the process like the flowers, little shrubbery or the red trees, as well as the farm models. One of the things that makes the environment more immersive are the little details, for example in the red forest area there are not only red trees but also red leaves on the grass and red mushrooms. This is only in that place, and the green shrubbery are only on the first forest to simulate different "*biomas*", see Figure 4.14.

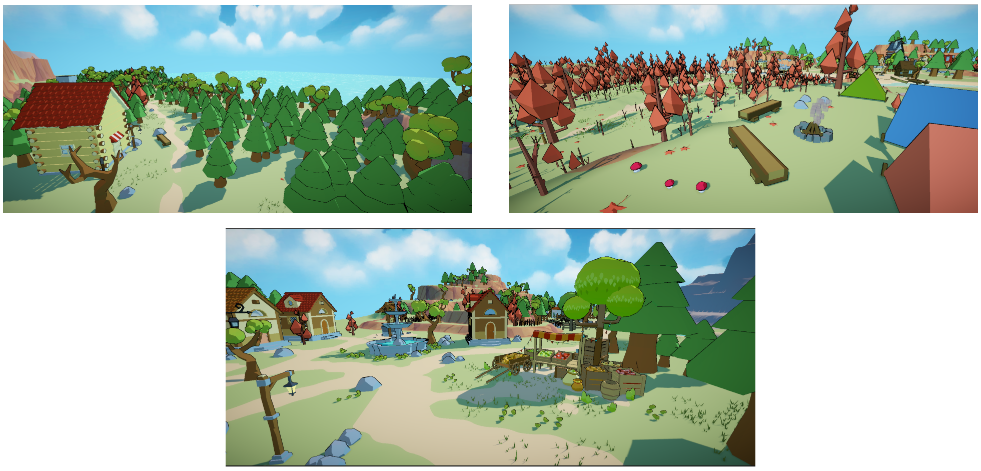


Figure 4.14: Green/red biomas, and the town.

4.3.6 UI and 2D art

Since the game has a cartoon style, the 2D art style is also similar to it. The decision was to make a hand made style. All of the art is made by the developer using *Clip Paint Studio* [11]. For the dialogues, the main reference is Miracle Merchant (Arnold Rauers,

2017) [39].

On top-left of the screen during the gameplay there is always an indicator of the player's remaining lives (red hearts). Then, when the player goes fishing a bar appears on the top-right part doing a ping pong effect to calculate the power to cast the rod (see Figure 4.15).

The idea of the game is that it looks like a tale, so the dialogues are like papyrus with a comic font style (see Figure 4.16). There are papyrus buttons for the menu options and the yellow ones for the choices. The logo is also made in the same program (see Figure 4.17). Same style is followed for the inventory and diary design (see Figure 4.18). The buttons and the icons for the inventory items are also hand drawn (see Figure 4.19).

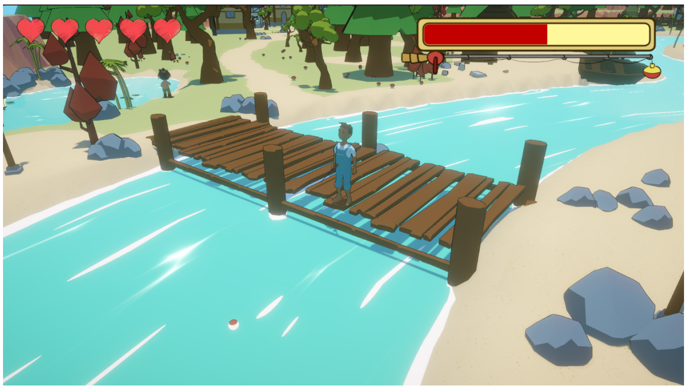
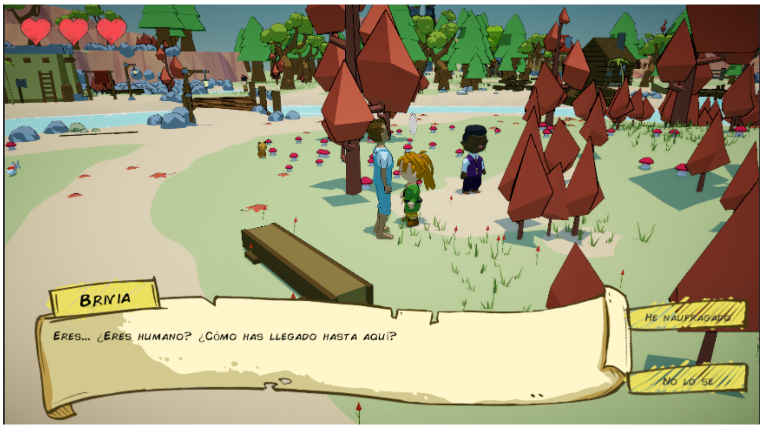


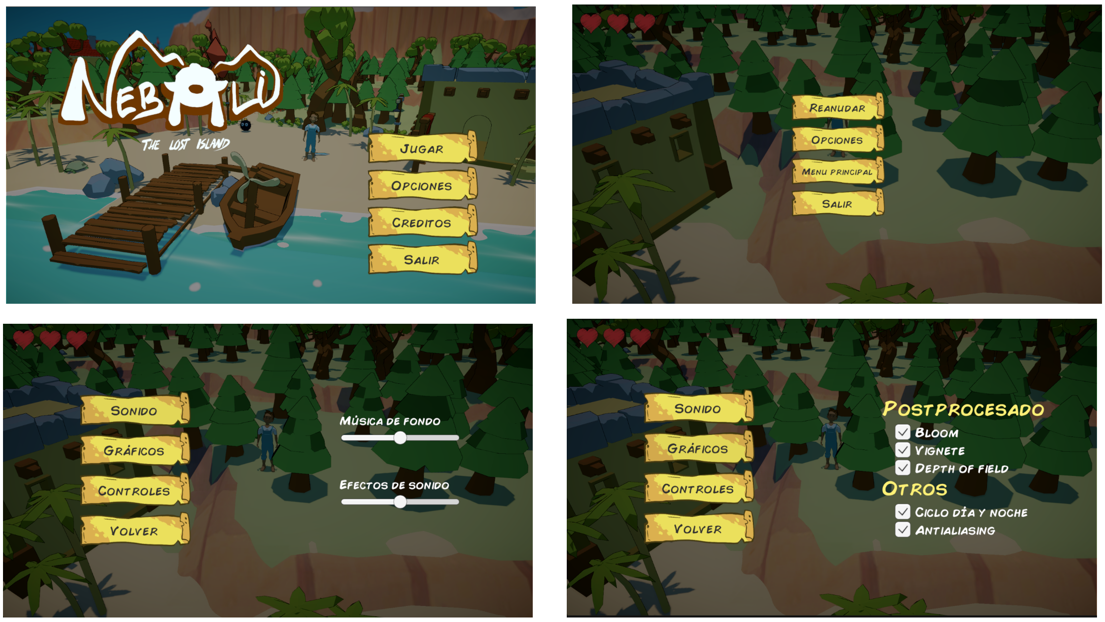
Figure 4.15: Fishing minigame.



BRIVIA

ERES... ¿ERES HUMANO? ¿CÓMO HAS LLEGADO HASTA AQUÍ?

Figure 4.16: An example of the dialogue UI.



Postprocesado

☑ Bloom

☑ Vignete

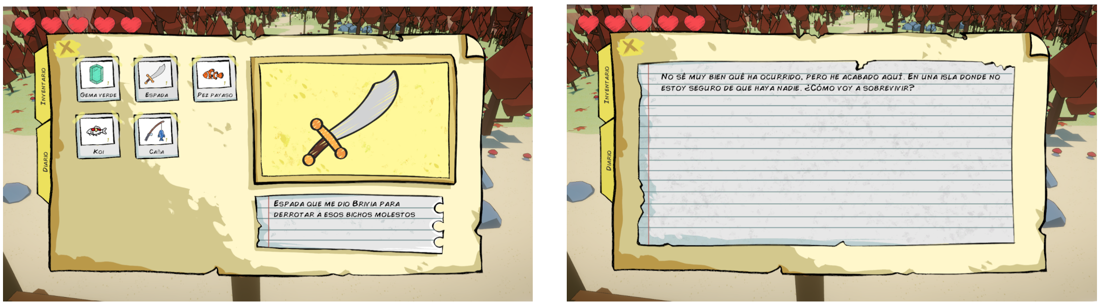
☑ Depth of field

Otros

☑ Ciclo día y noche

☑ Antialiasing

Figure 4.17: Different menus from the game.



Espada que me dio Brivia para derrotar a esos bichos molestos

No sé muy bien qué ha ocurrido, pero he acabado aquí. En una isla donde no estoy seguro de que haya nadie. ¿Cómo voy a sobrevivir?

Figure 4.18: Inventory (left) and diary (right).

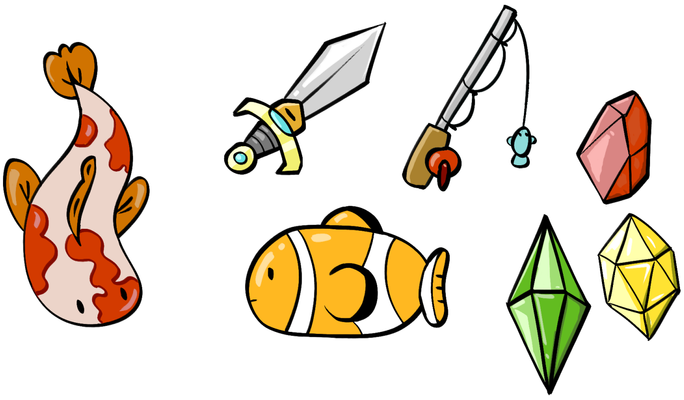
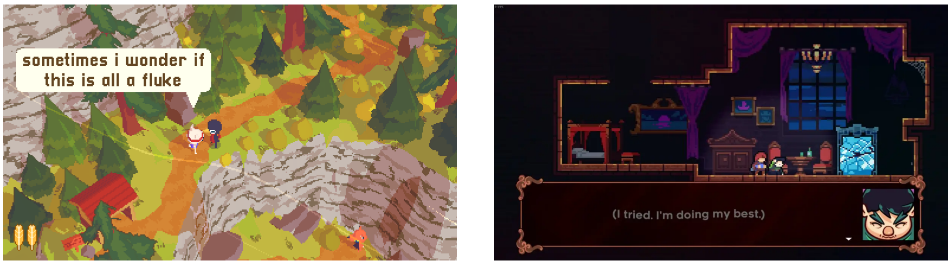


Figure 4.19: Icons for the inventory items.

4.4 Narrative design: how to make it related to mental health

The purpose of the game is to create a fun, immersive and entertaining experience and at the same time explore an story with a mental health background. After a wide research about both anxiety, depression and ADHD studies and videogames of this theme, the narrative design could start its development. The main references are *Celeste* [22] and *A Short Hike* [41]. See Figure 4.20.



sometimes i wonder if this is all a fluke

(I tried. I'm doing my best.)

Figure 4.20: A short hike (left) and Celeste (right) dialogues.

The game is an **open world** to make the player able to explore whatever he/she wants. The main objective is to end with the little enemies (a reference to intrusive thoughts), but the most interesting thing are the **dialogues**. Talking with the villagers that can be found around the map, the player can understand their problems and even identify in some way. They represent common human behaviours like the feeling of loneliness, the need of fitting in, going out of a comfort zone, feeling overwhelmed, impostor syndrome... Besides, each one has secondary quests that the player can do while it explores the world.

A well written narrative is essential to have good immersion. If the story is convincing, the world building is attractive, the music contributes to the general atmosphere and the gameplay is fun, the player will feel immersed into the game. In this scenario, immersion does not mean psychical immersion (like VR games for example), but more like **psychological immersion**. The idea is that the player feels "into" the game, avoiding his possible real life stress and at the same time having fun.

The first thing to do is write the characters. Originally, the idea was to have 4-5 secondary characters (in fact, their personalities were all developed yet) but due to the time limit the game finally has 3 characters.

Brivia is the first one that the player will meet. She is the fighter and knows everything about the Boo's. Brivia will show in her dialogues that she is very extrovert, nervous and friendly, but at the same time distrustful. If the player talks with her, she will tell that she lives alone in the woods and even she appears to be super strong and invincible, , she has also fears like anyone else. Brivia will tell the player that she was all alone in the island until she found the others, so on one hand she knows perfectly how to survive on her own but on the other hand she lacks social rules and sometimes is inadequate. She struggles asking for help and the player will learn that if he/she is frustrated with something that does not know how to deal with, it is okay to ask for help. She kind of represents ADHD, but not to be confused with the idea that every character represents a mental issue or disorder.

After that, the player will get to know **Edbri**, the major. Outgoing, charismatic and with outstanding leadership, he is perfect for his job. However, he will end up telling the player that sometimes he feels that he is not enough. He feels overwhelmed and thinks that he is not capable of being in such a responsible work. Mateo will talk with him and convince him that he is in fact more than enough, but he should care less about others opinions. Sometimes it is inevitable, but comparing with everybody does not lead anywhere. This is a common behaviour in human lives, to feel that they are not enough for something, so maybe the player can feel identified and empathize with Edbri.

Lastly, **Leeba** is an old fisher near to the lake. He spends his days fishing and sitting outside of his cabin and he does not care much about anything. He usually ignores everyone but if the player talks with him, he will eventually confess that sometimes he feels alone. It is common to feel that way in different situations, not only being an old man. For example, is normal that Mateo feels alone because he is in a new situation out of his comfort zone and he has to deal with it almost alone. Because of that, they will have things in common to talk.

Everybody faces situations like Mateo. They may not be exactly like that (a shipwreck) but they can be very similar. It is very usual that in a moment of our lives we need to leave our home, family and friends behind and start a new life. Getting out of the comfort zone can be very hard, but **with good mental health it is possible to overcome**.

4.5 Gameplay and level design: the coding

Since the beginning of the project it was well known that designing, modelling, writing and coding everything by only one person was maybe too difficult for this project. Even so, programming is one of the most important skills learned in this degree, so a Final Degree Work without interesting coding features did not felt enough. Furthermore, it was intended to demonstrate that it is possible to like **both** the artistic and the coding part of a game development. That is why it was decided to make a project that combined both of them.

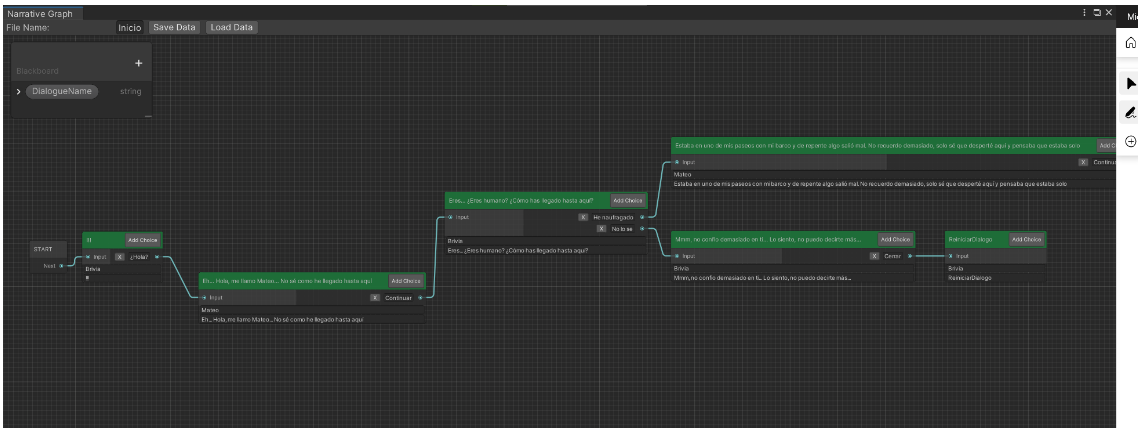
The programming part of this game can be divided into the dialogue system and management, the inventory system, the player controller, the camera, the NPC's quests and the enemies. The whole project is based on how this scripts communicate between each other, mostly using **Events**.

Some parts of the code will be shown to explain better the concepts, **but the full source code should be accessed through the GitHub repository.** [58]

4.5.1 Dialogue system based on Node Graph

This game was going to be mainly a narrative game based on dialogues. If the dialogue scaled a lot, it could be really difficult to manage. For this reason, it was decided to dedicate a decent amount of time to think about the best way possible to make the bases. After some research, a really interesting option was using ***Unity Node Graph API*** [**5**] to make a node based dialogue system.

Unity offers an scripting API called *Unity Editor*. This is useful for programming tools intended to be used in the editor. One of them is the *GraphView* class. With this amazing framework, it is possible to program node based windows to use in your projects. Using a tutorial as a reference, a node based dialogue system was coded for the project. It consists of a window that allows the user to create nodes with a character name and a dialogue text. Each node can have one or more choices (multiple choice ramified dialogues), and they can be linked to other nodes. This is all stored in a *.asset* file. The files can be saved and loaded into the tool window. The saved files can be accessed from any script. The tool also has a Blackboard, which is another window with variables that can be used in the dialogue nodes (for example, a character name) and can also be accessed from other scripts. See Figure 4.21.



!!!
Brivia
!!!

Eh... Hola, me llamo Mateo... No sé como he llegado hasta aquí
Mateo
Eh... Hola, me llamo Mateo... No sé como he llegado hasta aquí

Eres... ¿Eres humano? ¿Cómo has llegado hasta aquí?
Brivia
Eres... ¿Eres humano? ¿Cómo has llegado hasta aquí?

Estaba en uno de mis paseos con mi barco y de repente algo salió mal. No recuerdo demasiado, solo sé que desperté aquí y pensaba que estaba solo
Mateo
Estaba en uno de mis paseos con mi barco y de repente algo salió mal. No recuerdo demasiado, solo sé que desperté aquí y pensaba que estaba solo

Mmm, no confio demasiado en ti... Lo siento, no puedo decirte más...
Brivia
Mmm, no confio demasiado en ti... Lo siento, no puedo decirte más...

ReiniciarDialogo
Brivia
ReiniciarDialogo

Figure 4.21: Dialogue graph window editor in Unity.

Mert Kirimgeri tutorials [26] were essential for this purpose. Using his code as a base, it was modified to store more data like the dialogue name or the character name, as well as the creation of a dialogue parser to show the dialogue on screen. Without getting into too much technical detail, these are the main parts of the code.

4.5.2 Dialogue System Editor

To start, there is a class for the dialogue nodes called *Dialogue Node* that contains a name, text, an ID and a entry point. Only the start node has this boolean true.

```
public class DialogueNode : Node
{
    public string Name;
    public string DialogueText;
    public string GUID;
    public bool EntyPoint = false;
}
```

The next class is called *StoryGraphView.* It derives from *GraphView* (not *Monobehaviour*) and uses *UnityEditor.Experimental.GraphView* namespace and Unity UI elements.

First, there is a constructor that has a *StoryGraph* as a parameter (explained later). It adds a grid background, a search window and some manipulators like content dragger, selection dragger, rectangle selector and freehand selector.

- **CreateCommentBlock().** To create the comment block data.
- **AddPropertyToBlackboard().** This function loads the exposed properties (name and value) and adds them to a Blackboard instance.
- **GetCompatiblePorts().** Starting from a port, for each port that is in the graph it analyzes if it is compatible and if it is, it adds the port to a compatible ports list.
- **CreateNode().** Creates a new node. The parameters that needs are the node name, the character name and the position. It creates a new instance of a *DialogueNode*, adds input ports when needed (the choices), registers the changes on the text fields with *RegisterValueChangedCallback* event and adds the text fields to the node container. It also adds buttons for the choice ports.

The next class is called *StoryGraph.* It uses the *StoryGraphView* and the *DialogueContainer* to create **the editor window and load/save the work** on a file.

- **CreateGraphViewWindow().** It creates a new window called *Narrative Graph.* Using *[MenuItem]* the user can create the window from a menu on the top toolbar in the editor.
- **ConstructGraphView().** This function creates the graph view.
- **GenerateToolbar().** To create a toolbar on top of the window. It has buttons for loading and saving the graph.
- **GenerateBlackboard().** To manage the variable creation (exposed properties).

There is also a class called *NodeSearchWindow* that allows the user to right click on the grid and search for the types of nodes available. For now, there are only the dialogue node and the comment block.

The *GraphSaveUtility* class is used for **storing, loading and saving the graphs**. To save the graph, it creates a scriptable object of a *DialogueContainer* instance. It reads the exposed properties and comments first to store that, and then it checks if there is a file with that name in the *Assets/Resources* folder (this is the default path). If there is, it just updates the changes. If there is not, it creates a new asset with the scriptable object created before. The asset will create or update all of the variables that a *DialogueContainer* has (*NodeLinks, DialogueNodeData, ExposedProperties and CommentBlockData*). For the asset creation purpose it uses the *AssetDatabase* Unity Editor class. To save the nodes it has to check for each node in the graph, and this is done in a *SaveNode* function.

To load the graph, the user writes a file name on a text field. If it exists, the *LoadNarrative()* function clears the graph, generate the dialogue nodes reading the *DialogueNodeData*, connects the nodes using the target node GUID of each node and add the exposed properties and the comments.

There could be much more to explain but this is the basic functionality of the tool.

4.5.3 Dialogue Parser

This class is used for displaying the dialogues in game. The dialogues are stored in a *Scriptable object* called *DialogueContainer*. An scheme was made in Figure 4.22 to understand better the concepts. The scriptable object wraps the following data:

- **NodeLinkData.** This is the info of the graph nodes. The base node GUID is the ID of the node that is reading, the port name is the name of the port and the target node GUID is the ID of the node (or nodes) that is connected to.
- **DialogueNodeData.** It stores the dialogue data itself. The Node GUID is the ID of the node, the *characterName* is the name of the character that is talking in that node, the dialogue text is the dialogue words and the position is where it is situated on the graph.

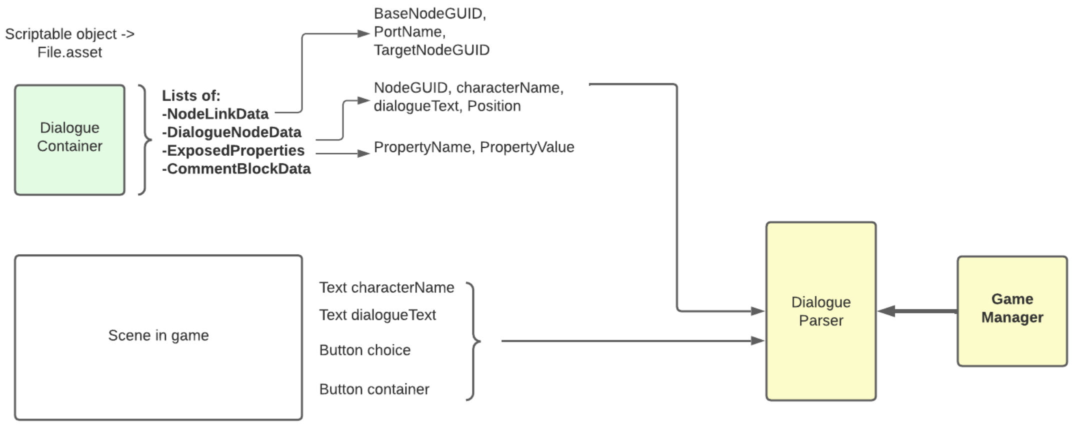


Figure 4.22: Dialogue Parser diagram.

- **Exposed properties.** This corresponds to Blackboard properties, variables that can be used in the nodes (for example a name that is often mentioned) or accessed by other scripts (for example, a name for that dialogue). Is has a property name and its value. Example: *DialogueName*, *StartDialogue*.
- **CommentBlockData.** It stores the comments that can be added in the graph.

Each *Dialogue Container* is saved into a *.asset* file. The *Dialogue Parser* class reads that file and scene variables, like the text for the dialogue, the character name or the buttons for the choices. Then, when it starts it reads the first node through its entry node and starts reading each node in order. To do this, it uses a function called *ProceedToNarrative()*.

```
        public delegate void OnNextLineEvent();
        public static event OnNextLineEvent onNextLineEvent;

        [...]

        private void ProceedToNarrative(string narrativeDataGUID)
        {
            var name = dialogue.DialogueNodeData.Find(x => x.NodeGUID ==
            narrativeDataGUID).characterName;
            var text = dialogue.DialogueNodeData.Find(x => x.NodeGUID ==
            narrativeDataGUID).DialogueText;
            var choices = dialogue.NodeLinks.Where(x => x.BaseNodeGUID ==
            narrativeDataGUID);

            characterNameText.text = ProcessProperties(name);
            dialogueText.text = ProcessProperties(text);
            var buttons = buttonContainer.GetComponentsInChildren<Button>();
            for (int i = 0; i < buttons.Length; i++)
            {
                Destroy(buttons[i].gameObject);
            }

            foreach (var choice in choices)
            {
                var button = Instantiate(choicePrefab, buttonContainer);
                button.GetComponentInChildren<Text>().text =
                ProcessProperties(choice.PortName);
                button.onClick.AddListener(() =>
                ProceedToNarrative(choice.TargetNodeGUID));
            }
            if (onNextLineEvent != null) onNextLineEvent();
        }
```

Essentially, it finds the name and dialogue text in the *DialogueNodeData* from the *DialogueContainer* dialogue file, and the choices in the *NodeLinks*. Then it assigns it to the scene text objects (*ProcessProperties* is a function that just replace the variables in *ExposedVariables* if there are any in the dialogue text). Finally, it destroy all buttons on the scene button container and instantiate the new ones depending on the choices. These buttons have a listener added so when the player clicks, it calls again this function to find the new dialogue lines.

The event *onNextLineEvent* is invoked every time a line of dialogue ends. This is used in another class to manage when the choices should appear. With this done, the next part is to write all the dialogues and develop a way to manage them. The game should know what dialogue comes when the player actives a dialogue, depending of the previous dialogues, the character that they are talking to and the game situation (quests completed, etc).

4.5.4 Dialogue UI and appearance.

When a character is talking, the letters appears like a typewriter and the options only appears when the text ends. If the player clicks while the player is still talking, all of the dialogue is shown. Following the art style and the immersion and cohesion with all the game, the characters talking sounds should be friendly and fun. The villagers are strangers, humanoids from other species, so it was interesting that they had a not understandable "language" like *Animal Crossing* or *The Sims*. [31]

For making this, a "beep" sound was used. When each letter appears on the screen, there is a random pitch between some defined values playing. Then, some talking presets were made for each character with minimum and maximum pitch values so each one have a different tone but keeping the randomization.

The way that this is done is using a **Coroutine** [15]. In the script that manages the dialogues (*ManagerDialogos.cs*), if the player clicks the coroutine is stopped, the text is changed to show the full line and the choices are activated.

The "*displayLineCoroutine*" variable corresponds to ***IEnumerator DisplayLine(string line)***. If the dialogue is being executed, the coroutine starts. It saves the full dialogue into a variable (textoDialogoEntero), empties the text in the scene and executes a loop. For each letter in the full text, it adds one letter to the scene text, plays a beep sound and waits.

```
private void DisplayLineFunction()
{
    if (dialogo.activeInHierarchy && textoDialogo.text != "CerrarDialogo"
    && textoDialogo.text != "ReiniciarDialogo")
    {
        if(displayLineCoroutine != null)
        {
            StopCoroutine(displayLineCoroutine);
        }
        displayLineCoroutine = StartCoroutine(DisplayLine(textoDialogo.text));
    }
}

private IEnumerator DisplayLine(string line)
{
    textoDialogoEntero = textoDialogo.text;
    textoDialogo.text = "";
    choicesContainer.SetActive(false);
    puedeContinuarSiguienteLinea = false;

    foreach (var letter in line.ToCharArray())
    {
        textoDialogo.text += letter;
        PlayDialogueSound(textoDialogo.text.ToCharArray().Length);
        yield return new WaitForSeconds(velocidadDialogo);
    }

    puedeContinuarSiguienteLinea = true;
    choicesContainer.SetActive(true);
}
```

PlayDialogueSound is the function that makes each letter sound with a different pitch randomly to give the illusion of speaking.

```
    private void PlayDialogueSound(int numCharactersDisplayed)
    {
        if(numCharactersDisplayed % frequencyLevel == 0)
        {
            audioSource.pitch = Random.Range(minPitch, maxPitch);
            audioSource.PlayOneShot(typingSoundClip);
        }
    }
```

4.5.5 Dialogue Manager

The Game Manager object has a script called ***ManagerDialogos***, responsible for handling the scene dialogues. These are the functions:

- **The Update method.** It checks the last node of the dialogue. There are 2 special nodes, one for ending the dialogue and closing it and one for rebooting the dialogue.
- **CambiarDialogo.** The most important method. It has a reference to the global *DialogueParser*, so when a *DialogueContainer* object is passed as a parameter, it changes the dialogue in *DialogueParser* and updates 2 global variables: **"personaje"** and **"dialogoActual"** that contains the current character name and the current dialogue. This is useful because other classes can read that information any time.
- **MostrarDialogo.** When a dialogue is opened, this method fixes the camera, unlocks the cursor (for making choices), stops the player and activates the dialogue object in the scene. It also updates the *personaje* and *dialogoActual* variables.

```
    public void MostrarDialogo()
    {
        //Congelar al personaje, fijar la camara y desbloquear el cursor
        cinemachineSwitcher.FijarCamara();
        Cursor.lockState = CursorLockMode.None;
        mateo.GetComponent<PlayerMovement>().canMove = false;

        //Activar el dialogo y mostrarlo línea a línea
        dialogo.SetActive(true);
        displayLineCoroutine = StartCoroutine(DisplayLine(textoDialogo.text));

        //Ver qué dialogo es, de qué personaje y según eso actualizar la info
        personaje = dialogueParser.dialogue.DialogueNodeData[0].characterName;
        dialogoActual = dialogueParser.dialogue;
    }
```

- **CerrarDialogo / ReiniciarDialogo.** They do the necessary to close the dialogue. **"onDialogueEvent"** is an event invoked when a dialogue is closed. It is used later for each character to know which dialogue has ended.

```
    public void CerrarDialogo()
    {
        cinemachineSwitcher.ReiniciarMovimientoCamara();
        Cursor.lockState = CursorLockMode.Locked;
        dialogo.SetActive(false);
        mateo.GetComponent<PlayerMovement>().canMove = true;
        if (onDialogueEvent != null) onDialogueEvent();
    }
```

4.5.6 Character dialogue management

Each character with whom the player can speak has the following scripts:

- **ActivarDialogo.** See Figure 4.23. When the player approaches a character, it checks if there is a dialogue available and if it is, a talk icon appears.

  If the player talks (press Space), this script looks for the character that is talking. Each character has a global variable called "*miProximoDialogo*" that stores their next dialogue. This script reads that variable and changes the dialogue that must show (using *ManagerDialogos.CambiarDialogo*). This is an example with Brivia:

```
if (ActivarIconoConversacion())
        {
            if (Input.GetKeyDown(gameManager.hablar) && !dialogo.activeInHierarchy)
            {
                //Que personaje es?
                switch (gameObject.name)
                {
                    case "Brivia":
                        managerDialogos.CambiarDialogo
                        (gameObject.GetComponent<Brivia>().miProximoDialogo);
                        break;
                    case "Edbri":
                        [...]
                }
                managerDialogos.MostrarDialogo();
            }
        }
```

- **Character script.** Each character has an own character script and each one has a list of *DialogueContainer* objects in order of appearance.

  It has been previously stated that when a dialogue was closed, an event called "**onDialogueEvent**" was invoked. The characters use that event to execute a function that, **depending on the last dialog displayed, their next dialog will be a specific one and will be updated in "miProximoDialogo"**. They read the "personaje" and "dialogoActual" variables explained before to know this information. Depending of that, it updates the list with the next dialogue that should appear if the player interacts with that character again. This way all the game timeline is programmed. For example: when the dialogue where Brivia says that she will give Mateo a sword finishes, a sword should appear in front of her and Edbri will unlock another dialogue.

```
//EXAMPLE WITH BRIVIA.CS. EventoDialogoBrivia():
[...]
        //Se ha terminado un dialogo con Edbri
        if (managerDialogos.personaje == "Edbri")
        {
            if (managerDialogos.dialogoActual.ExposedProperties[0].PropertyName ==
            "DialogueName")
            {
                switch (managerDialogos.dialogoActual.ExposedProperties[0].PropertyValue)
                {
                    case "Edbri1":
                        miProximoDialogo = dialogos[3]; //Brivia4
                        break;
                    case [...]
                }
            }
        }
```

  These scripts (*Brivia.cs, Edbri.cs, Leeba.cs, Nebali.cs*) are really long and they refer both to the dialogues and the missions, so they can be full seen in the GitHub repository. *(Assets/Scripts/Personajes).*

4.5.7 Game Manager

The Game Manager script is used to manage the global functions and variables of the game, as well as UI button functions.

- **The enums.** This class has two "enums". One is called **"Zones"** and contains the zone of the game where the player is to have a reference to the *respawn* if it dies (the forest, the beach, the town, the farm...). The other is called **"Missions"** and

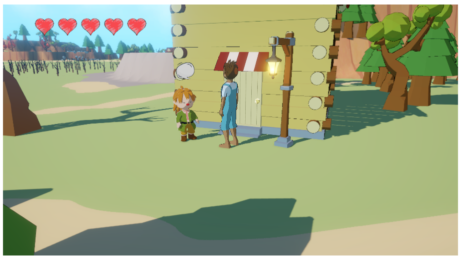
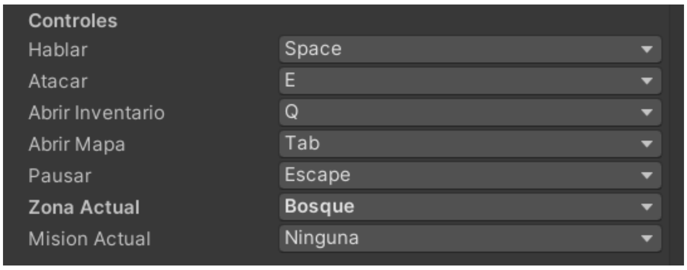


Figure 4.23: Talking icon.

it includes all of the secondary missions, this way the game knows every moment which mission the player is currently doing. Depending on the zones, a different song is played.

- **Controls.** There is a public reference to the controls using *KeyCodes*. This way they can be changed any time without changing other scripts. See Figure 4.24.



Controles

Hablar	Space
Atacar	E
Abrir Inventario	Q
Abrir Mapa	Tab
Pausar	Escape
Zona Actual	**Bosque**
Mision Actual	Ninguna

Figure 4.24: Controls in Game Manager.

- **UI**. This class also has functions for opening and closing the map, inventory, main menu and pause menu. When a menu is opened, the player is stopped from moving using the *isMoving* variable from the *PlayerMovement* class. The camera is fixed with a reference to the *Cinemachine Switcher* class (explained later) and the cursor is locked. When certain actions occur, there is a function to show a message to the player for a number of seconds (for example when an object is added to the inventory or a mission is completed).

  There are functions for every button. The Pause menu, for example, has buttons for resuming, the options, main menu and exit. The options menu has options for the sound, graphics and controls. See Figure 4.17.

  The sound is made using ***Audio Mixer*** [14], a very useful Unity tool for adding sound effects and separating different channels using *Audio Mixer Groups*. There is a group for the sound effects and one for the background music so the player can easily control the different volumes. The volume is not linear so a logarithm with base 10 is used.

- **Inventory.** One of the most useful resources used in this project are the **Events**. When the player opens the inventory, an event is launched so the inventory scripts draw the UI.

```
         public delegate void OnInventoryOpenedEvent();
         public static event OnInventoryOpenedEvent onInventoryOpenedEvent;

         [...]

         if (Input.GetKeyDown(abrirInventario) && !inventario.activeInHierarchy)
        {
            AbrirInventario();
        }

         public void AbrirInventario()
        {
            inventario.SetActive(true);
            cinemachineSwitcher.FijarCamara();
            Cursor.lockState = CursorLockMode.None;
            playerMov.canMove = false;
            if (onInventoryOpenedEvent != null) onInventoryOpenedEvent();
         }
```

There is also a global variable that stores the object that the player is holding so other scripts can work with that.

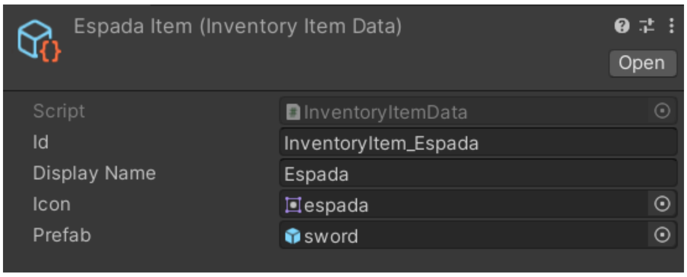
4.5.8 Inventory system

In some parts of the game, the player needs to save objects that will be used later on. The inventory can be accessed at any moment during the gameplay and consists of a grid with an image of the object, the name and a number of stack. The main scripts used are the following:

- **InventoryItemData.** This is a scriptable object that can be created from the Asset menu. It is used for saving the properties of a type of object. It stores the id, name to be displayed, icon and prefab of the object. See Figure 4.25.

```
        [CreateAssetMenu(menuName = "Inventory␣Item")]
        public class InventoryItemData : ScriptableObject
        {
            public string id;
            public string displayName;
            public Sprite icon;
            public GameObject prefab;

        }
```



Espada Item (Inventory Item Data)

Script InventoryItemData
Id InventoryItem_Espada
Display Name Espada
Icon espada
Prefab sword

Figure 4.25: An example of the sword InventoryItemData.

- **InventoryItem.** The difference with the previous class is that *InventoryItem* stores an item or an slot **in the inventory**, not a type of object, so it needs to have an *InventoryItemData* (the object that is storing) and the stack size (the number of objects of that type that the player has). It has functions to control the stack.
- **InventorySystem.** The current inventory, stored in a List of *InventoryItem* objects. There is also a dictionary with an *InventoryItem* data as the key and an *InventoryItem* as the value. That way is easy to know for each type of object how many of them are in the inventory, if there is one. The class has functions to get, add and remove items from the inventory using both classes described above.

```
public class InventorySystem : MonoBehaviour
{
   [...]
    public InventoryItem Get(InventoryItemData referenceData)
    {
        if (itemDictionary.TryGetValue(referenceData, out InventoryItem value))
        {
            return value;
        }
        return null;
    }
    public void Add(InventoryItemData referenceData)
    {
        if(itemDictionary.TryGetValue(referenceData, out InventoryItem value))
        {
            value.AddToStack();
        }
        else
        {
            InventoryItem newItem = new InventoryItem(referenceData);
            inventory.Add(newItem);
            itemDictionary.Add(referenceData, newItem);
        }
    }
```

```
        public void Remove(InventoryItemData referenceData)
        {
            if (itemDictionary.TryGetValue(referenceData, out InventoryItem value))
            {
                value.RemoveFromStack();

                if(value.stackSize == 0)
                {
                    inventory.Remove(value);
                    itemDictionary.Remove(referenceData);
                }
            }
        }
    }
```

- **ItemObject.** This script is attached to the objects that can be collected in the scene so when the player collides with them, *OnHandlePickupItem*() is executed and they are added to the inventory and destroyed from the scene. It is necessary to assign a *InventoryItemData*.

  The next two classes handle how the inventory is shown on screen.

- **UIInventoryItemSlot.** This stores the data that it is needed to be shown in each inventory slot. It uses the image to be shown in the slot, the name, the current prefab of the object and the stack number. The data can be modified with the *Set* function.

- **InventoryUI.** This class manages the whole UI of the inventory. It is attached to a *Canvas* with a *Panel* and an *Horizontal Layout Group* and *Content Size Fitter* component. This way each time an object is added or deleted it automatically snaps into place. The script uses the event that the *GameManager* launches when the player opens the inventory (See section 4.5.7, inventory). On the other hand, and item prefab is made on the editor to adjust how the image, name and number is seen. This prefab is a *Unity Button*.

  When the inventory is opened, the script cleans the previous one and draw the new, searching each item in the *InventorySystem* list. For each one, it instantiates an item prefab like the child of the container (the UI *Panel*). Then the UI data is filled using the *UIInventoryItemSlot* and a listener is added to the button.

  The button calls a function so when the player clicks on a slot, that object appears on the right hand. The player object container references the hand "slot", so first it is necessary to check if it is empty. If it is not, it deletes wherever is on the hand. Then the function instantiates the object on the hand (deleting the *ItemObject* component to avoid errors) and closes the inventory.

The results can be seen in the UI section, see Figure 4.18.

4.5.9 The player, enemies and the camera

The player is controlled by to main scripts: ***PlayerMovement*** and ***PlayerController***.

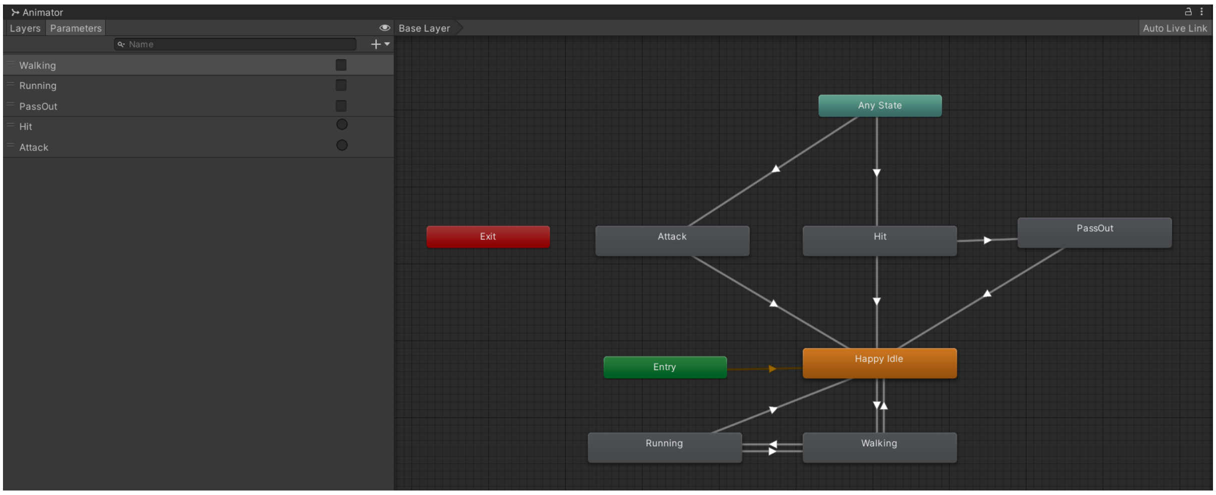


Figure 4.26: Animator diagram for Mateo.

- **PlayerMovement.** This script is used exclusively for the movement calculations, depending on a ***Character Controller*** Unity component. It has variables for the normal and running speed, the turn smooth speed, the gravity... and also a public boolean called *isMoving* to control when the player should not move (for example on the menus).
- **PlayerController.** This manages the animations (see Figure 4.26), enemies and lives. It also has a function so when the player is near to a campfire zone, he can sit by pressing space. To attack the enemies, it is done using **Raycast.** The player draws a ray and if it collides with an enemy object, it executes the *TakeDamage* function in the enemy.

```
    private void Atacar()
    {
        GameObject espada = gameManager.objetoEnMano;

        // Lanza un rayo desde la posición y la dirección del jugador
        Ray ray = new Ray(transform.position, transform.forward);
        RaycastHit hit;

        Debug.DrawRay(ray.origin, ray.direction * rangoAtaque, Color.red, 0.5f);

        // Comprueba si el rayo colisiona con algún objeto en la capa de los enemigos
        if (Physics.Raycast(ray, out hit, rangoAtaque, enemyLayer,
        QueryTriggerInteraction.Collide))
        {
            // Obtiene el componente del enemigo
            EnemigoController enemyHealth =
            hit.collider.GetComponentInParent<EnemigoController>();

            // Verifica si el enemigo tiene el componente EnemyHealth
            if (enemyHealth != null)
            {
                // Reduce la vida del enemigo
                enemyHealth.TakeDamage(dañoAtaque);
                espada.transform.GetChild(espada.transform.childCount-1)
                .gameObject.GetComponent<ParticleSystem>().Play();
            }
        }

        animator.SetTrigger("Attack");
        //Ejecutar vfx
        espada.transform.GetChild(Random.Range(0, espada.transform.childCount-1))
        .gameObject.GetComponent<ParticleSystem>().Play();
    }
```

If the player collides with an enemy, one life is subtracted, one heart is deleted and the damage animation is executed. If he dies, the position is reset according to the zone he is in.

The enemies are controlled by one script. They have an AI with a machine state. There are 3 states: **patrol, alert and attack.** When they take damage by the player, the following coroutine is executed:

```
IEnumerator Daño(int dañoRecibido)
   {
       vida -= dañoRecibido;
       mrenderer.material.SetColor("_MainColor", colorAtacado);
       rb.AddForce(-transform.forward * fuerzaImpulsoRecibeDaño,
       ForceMode.Impulse);
       yield return new WaitForSeconds(tiempoDaño);
       mrenderer.material.SetColor("_MainColor", colorOriginal);
   }
```

They patrol between specific waypoints randomly. If the player is near, they switch to alert state, and if a few seconds pass, they switch to attack state and start chasing the player. They have 2 attacks that execute randomly in this state, one is a sprint to the player and the other is a jump. It is done using coroutines and the *AddForce* function from *Rigidbody*. When they die, some VFX is shown to make it more appealing and the number of enemies in each zone is updated in the Game Manager. See Figure 4.27.

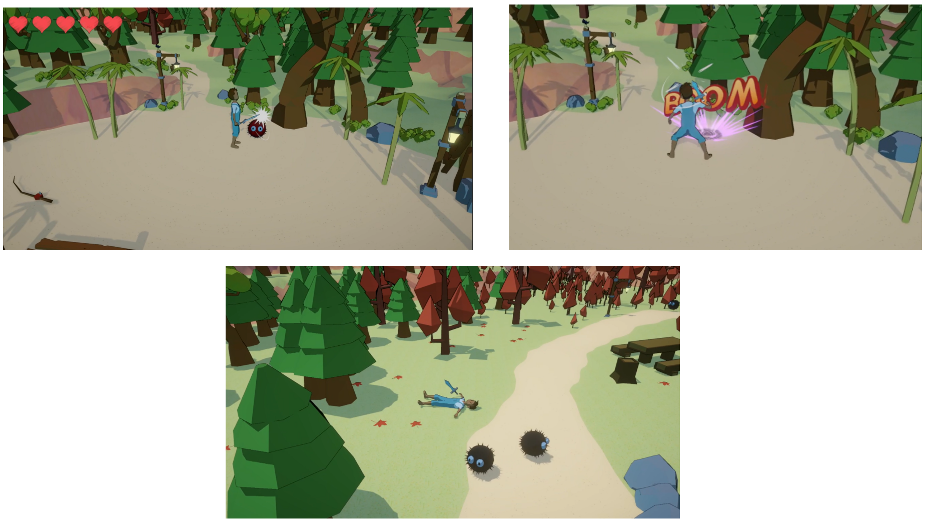
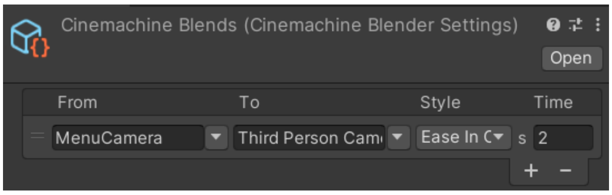


Figure 4.27: Attacking enemies and player passing out.



Cinemachine Blends (Cinemachine Blender Settings)

From	To	Style	Time
MenuCamera	Third Person Cam	Ease In C	s 2

Figure 4.28: Cinemachine blends asset.

The cameras use the **Cinemachine** plugin available on the *Unity Package Manager* [53]. There is a *Free Look camera* that always follows the player from a distance. It has X and Y speed values so the player can move the camera with the mouse and restrict values so the player can not move the camera in strange angles (for example underground).

Furthermore, there is a camera for the main menu. It focuses on the starting point and when the player clicks the play button, there is a transition to the player camera. This transition is made using **Cinemachine Custom Blends** (see Figure 4.28). The transition values are stored in an asset. To control them, a custom script called *CinemachineSwitcher* was made. This script switches the priority of the cameras using an state variable. If the current state is the menu, it has to change to 3rd person camera so the priority is changed, and vice versa. The script also has functions to fix the camera.

On the other hand, a map was implemented to prevent the player from getting lost. It consists of an orthogonal camera from above. All of the important objects (the player, the gems...) have a canvas on top of them with an icon (a 2D sprite). This is in a special layer that the main camera does not render on the *culling mask*. This way, all of the icons are rendered only on the map camera, for example a flashing red point where the player is at each moment. See Figure 4.29.

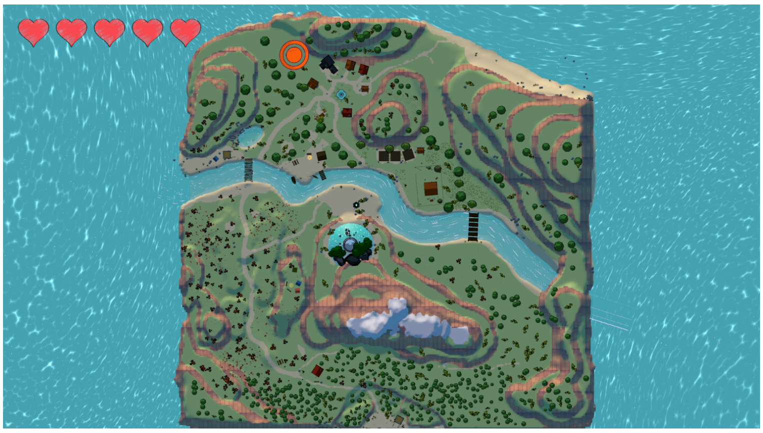


Figure 4.29: In game map.

4.5.10 NPC's and quests

One of the main tasks for the player to do is to talk with the villagers. There are 3 of them, but in the future there could be more.

The first villager that the player will meet is **Brivia**. She is at the forest house. When the first dialogue ends, she will accompany Mateo to town to tell him to talk to the mayor. This is made using **Waypoints** and a **Navmesh agent. Edbri, the mayor**, will explain the basic introduction. The first mission is to meet all the villagers (script *EncontrarAldeanos.cs*).

When it is done, if the player talks again with him he will encourage him to talk with Leeba. **Leeba** has the **cat mission**. The player has to find his cat but he will only follow Mateo if he has a fish in his hands. The player needs to take Leeba's fishing rod, catch a fish and give it to the cat so he will follow Mateo. Then, the player will give the cat to Leeba and the mission is done. **All of this is handled using Events.** There are 3 scripts: *Pescar.cs* for fishing, *Anzuelo.cs* instantiated with the hook and *Gato.cs* for the cat.

```
//MAIN METHODS IN PESCAR.CS

    private void CalcularSliderImpulso()
    {
        tiempoImpulso += Time.deltaTime;
        valorImpulso = Mathf.PingPong(tiempoImpulso, 1);
        sliderImpulso.value = valorImpulso;
        if (Input.GetKeyDown(gameManager.hablar) && !elegirImpulso && !lanzado
        && anzueloLanzado == null && clicks == 2)
        {
            Debug.Log("Lanzado");
            LanzarCanya();
        }
    }

    private void LanzarCanya()
    {
        lanzado = true;
        Vector3 dir = transform.forward;
        GameObject anzuelo = Instantiate(anzueloPrefab, inicioLanzamiento.transform.position,
        transform.rotation);
        Rigidbody rb = anzuelo.AddComponent<Rigidbody>();
        anzuelo.AddComponent<Anzuelo>();
        rb.AddForce(maxImpulso * valorImpulso * dir);
        anzueloLanzado = anzuelo;
    }

    private void RecogerCanya()
    {
        lanzado = false;
        sliderImpulso.gameObject.SetActive(false);
        anzueloLanzado.GetComponent<Collider>().isTrigger = true;
        Rigidbody rb = anzueloLanzado.GetComponent<Rigidbody>();
        rb.AddForce((inicioLanzamiento.transform.position - anzueloLanzado.transform.position)
        * 400, ForceMode.Impulse);
        rb.mass = 1;
        rb.angularDrag = 0.05f;
        rb.drag = 0;
        rb.useGravity = false;
        elegirImpulso = true;
        clicks = 0;
        SacarPezAleatorio();
    }

    private void SacarPezAleatorio()
    {
        int randomPez = UnityEngine.Random.Range(0, peces.Count);
        if(peces[randomPez] != null && anzueloLanzado != null)
        {
            Vector3 pos = new Vector3(anzueloLanzado.transform.localPosition.x,
            anzueloLanzado.transform.localPosition.y + 1f,
            anzueloLanzado.transform.localPosition.z);
            GameObject pez = Instantiate(peces[randomPez], pos,
            anzueloLanzado.transform.localRotation);
            Rigidbody rb = pez.AddComponent<Rigidbody>();
            rb.mass = 1;
            rb.angularDrag = 0.05f;
            rb.drag = 0;
            rb.AddForce((inicioLanzamiento.transform.position
            - anzueloLanzado.transform.position) * 2, ForceMode.Impulse);
        }
        else
        {
            Debug.Log("Error␣pesca:␣no␣existe␣pez␣o␣anzuelo");
        }
    }
```

The **main quest** is to kill all the enemies in each of the 3 zones: forest, village and farm. Each one will unlock a **gem** (*Cinemachine Blends* is also used to focus the camera on the gems locations). When the 3 gems are collected, the player needs to put them in a tower and the temple will be activated. **The god Nebali** will talk with him and grant him a wish, either to stay on the island or to return home. This is the end of the game. See Figure 4.30.

```
    public delegate void FinalEvent();
    public static event FinalEvent onFinalEvent;
    [...]
    void ColocarGema()
    {
        if (Vector3.Distance(gameManager.player.transform.position,
        transform.position) <= radio)
        {
            //Si tiene un objeto
            if (gameManager.objetoEnMano != null && Input.GetKeyDown(gameManager.atacar))
            {
                GameObject gemaColocada;
                switch (gameManager.objetoEnMano.tag)
                {
                    case "GemaVerde":
                        gemaColocada = containerVerde.transform.GetChild(0).gameObject;
                        gemaColocada.SetActive(true);
                        Destroy(gameManager.objetoEnMano);
                        if (gemaColocada.TryGetComponent<ItemObject>(out ItemObject item))
                        {
                            gameManager.GetComponent<InventorySystem>().Remove(item.referenceItem);
                            gameManager.MostrarMensaje("Gema␣verde␣colocada␣con␣éxito.");
                            verde = true;
                        }
                        break;
                    case "GemaRoja":
                        [...]
                }
            }
        }
    }

    IEnumerator Final()
    {
        gameManager.GetComponent<CinemachineSwitcher>().SwitchPriority("VerFinal");
        yield return new WaitForSeconds(2);
        llama.Play();
    }
}
```

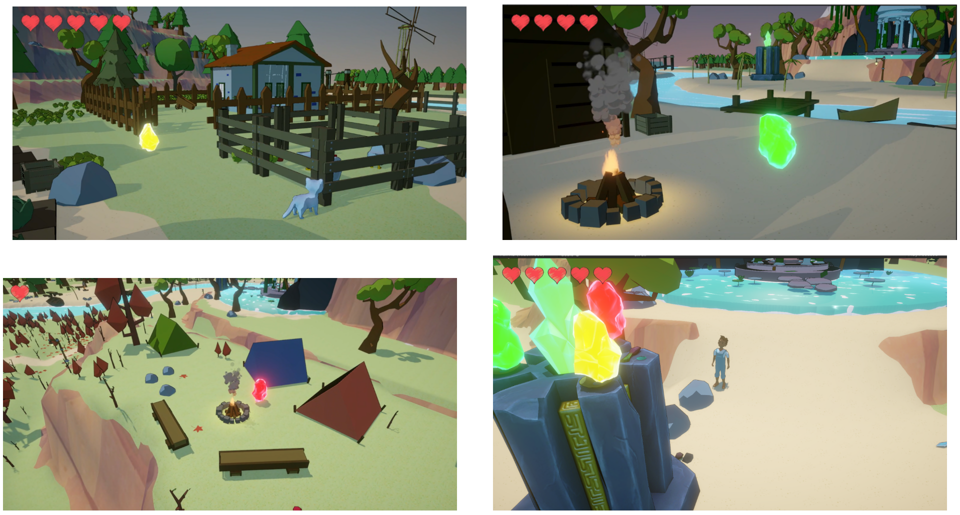


Figure 4.30: The different gems and the tower.

4.5.11 Optimization

The game uses a wide terrain with hundreds of models, so this was an essential part. The **Unity Terrain** already has optimization features like **details distance**, so they are only rendered when the player camera is looking at it. It had to be adjusted to avoid "*popping*", when an object suddenly appears in front of the player.

Most of the terrain objects are part of the Paint trees/details brushes for them to use that optimization features. However, some of them are not part of the terrain like the houses, characters, water, bridges and other models. For that purpose, they all have **LOD** components with a culling percentage to disappear when they are not on the players vision. The most important thing is the **Occlusion culling** (Figure 4.31). This technique hide all objects not seen by the camera (because they are static) to reduce computer costs.

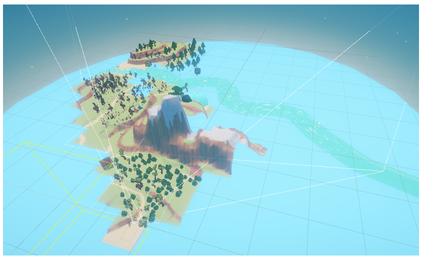


Figure 4.31: Occlusion culling.

4.5.12 Others

To favor immersion, an interesting idea was to implement a day and night cycle. For this purpose, these were the steps:

- **Time percent.** The main script for the cycle is called *Lighting Manager* and does all of the calculations to divide the time in 24 parts. Depending on that and on the speed value set by the developer, some changes are made using a lighting preset. The night is accelerated a bit, this way there are more day hours.
- **Directional light rotation.** The directional light (sun) rotates as the day goes on so the shadows and global illumination change dynamically.
- **Shadows and ambient light color change.** An scriptable object was created to save Lighting presets for these values. The ambient light and shadows color change when the day continues using a gradient defined in that preset.
- **Skybox.** An special skybox is used for the night and day cycle. It has a texture with night, dawn, noon and sunset. The idea is to increase the UV offset value so the texture changes dynamically. On the other hand, the skybox also rotates to give the impression of the real rotation of the earth.

The result is an environment much more immersive and interesting (see Figure 4.32). Here is an example of a part of the full script.

```
public class LightingManager : MonoBehaviour
{
    [...]

    private void UpdateLighting(float timePercent)
    {
        //Set ambient and fog
        RenderSettings.ambientLight = Preset.AmbientColor.Evaluate(timePercent);
        RenderSettings.fogColor = Preset.FogColor.Evaluate(timePercent);

        if (DirectionalLight != null)
        {
            DirectionalLight.color = Preset.DirectionalColor.Evaluate(timePercent);

            DirectionalLight.transform.localRotation = Quaternion.Euler
            (new Vector3((timePercent * 360f) - 90f, 170f, 0));
            sky.transform.localRotation =
            Quaternion.Euler(new Vector3(0, (timePercent * 360f) - 90f, 0));

            skyboxMaterial.mainTextureOffset = new Vector2((TimeOfDay / 24f) + 0.5f, 0);
        }
    }
}
```

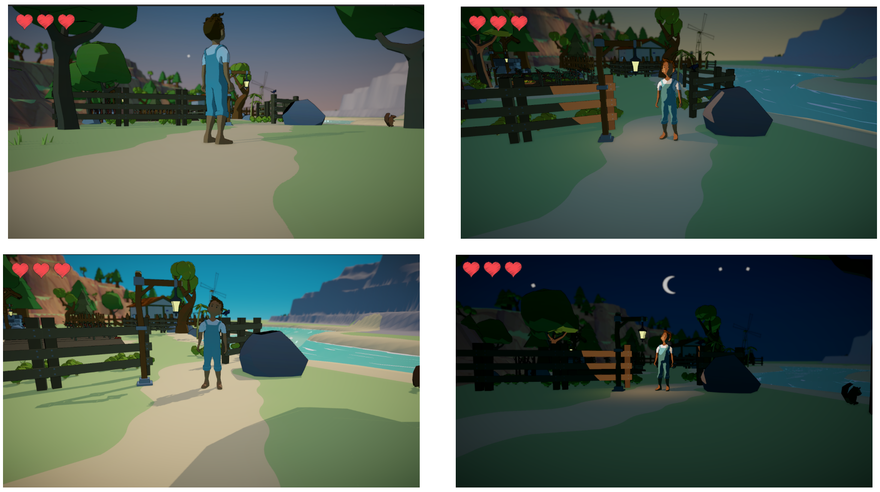


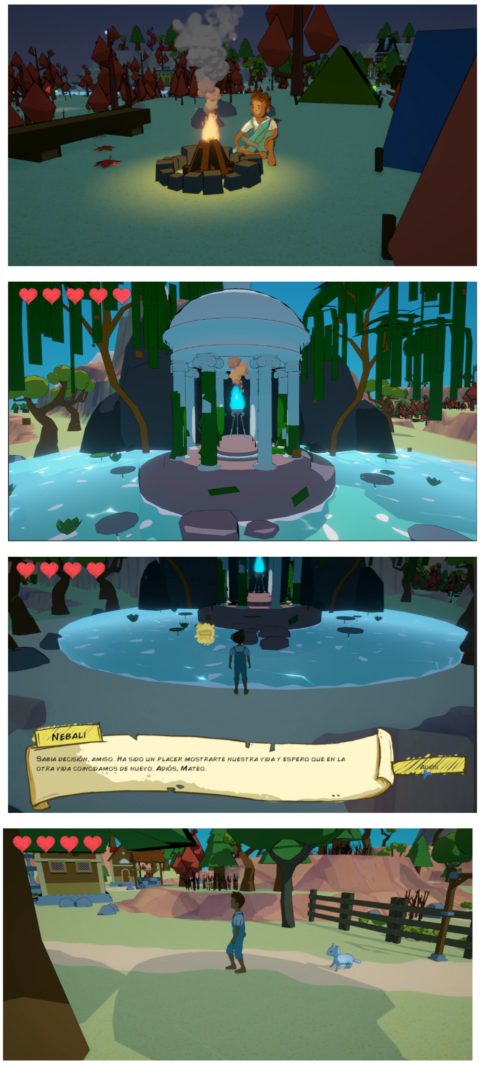
Figure 4.32: Dusk, dawn, day and night.

4.6 Results

After a hard work, the result is quite satisfactory. The game **meets most of the initial requirements**, has a beginning and an end, has some interesting mechanics, side quests and the implementation of enemies. The environment is friendly and the art looks cohesive with each other. There is an interesting story and the characters tell a narrative related to mental health. And most importantly, it is playable and entertaining. It has menus, a lot of options, a diary, inventory, a map, different characters, enemies, different shaders... and even the dialogue tool for making easy ramified dialogues with nodes.

The idea was very ambitious and during development a lot of things had to be cut back in order to be on time, but the result is **an interesting demo of what could be a bigger game.** Adding more characters, stories, quests and areas can result in an interesting game that is not only entertaining to play but can also help people with mental disorders feel represented. See results on Figure 4.33.

The game has an own web page in **itch.io** [57] where it can be downloaded. The project repository is published on **GitHub** [58].



Nebali

Sabia decisión, amigo. Ha sido un placer mostrarte nuestra vida y espero que en la otra vida coincidamos de nuevo. Adiós, Mateo.

Figure 4.33: Some screenshots of the final game.

Chapter 5

Conclusions and Future Work

Contents

5.1 Conclusions

After finishing this project, I consider that the idea was perhaps too complex for one person in the given time. However, I have personally learned many things that I would not have achieved if it were not for this project. It has been many hours of work and very frustrating moments where it seemed that nothing was going to go well and that I did not arrive on time, but I think I have managed to have a game that I am proud of and is close to the initial idea I had. I have applied both modeling and art concepts as well as design and programming, and it has helped me to consolidate what I have learned during the degree. Making a complete game being only one person is difficult but it is possible.

I would have liked to add more things that maybe do not look exactly the way I wanted, but I still like the final result. At the end of the day, the project has its pros and cons but it **represents my personal project and my passage through the degree**. I think it is a good presentation for my portfolio and a good closure for what I have learned during these 4 years.

5.2 Future work

I would really like to continue the project. As I mentioned earlier, I consider the project to be a demo of what could be a full game. In the future I would like to revise it and fix some things. Also, I would like to add more characters, more story and more quests.

One of the initial ideas that I had and that due to time I could not realize is that there would be several islands and the player could **use a ship and navigate between them**. If I continue the game, this would be something I would totally add. At first I also thought it would be a good idea that, being a game about a castaway, there would be **survival mechanics**. It would be very interesting to add necessities (hunger, sleep, hygiene...) and that the player would have to hunt for food, make a house with materials, etc. Using the current implementation of the day and night cycle, you can also add **meteorology** so that it rains, snows or is sunny, and with this you can even make seasons. I think there are endless possibilities to continue the game and make it much more complete. If I could get to that point, **I would love to upload it to Steam and my portfolio.**

In the near future, I would also like to write an article about this project to present it to the **CEV23 (*"Congreso Español de Videojuegos"*)** [36], a spanish video game congress that will take place in 2023. I recently joined an university fellowship to work in INIT (*Instituto de Nuevas Tecnologías de la Imagen*) and my coworkers, who write articles and scientific papers, told me about this congress. They convinced me to present my final degree project to the congress, because many other spanish students from these field did it in previous years and it is a memorable experience. On the other hand, the CEV collaborates with the **Guerrilla Game Festival** by sharing its conference program. Moreover, this event will take place concurrently with the **Madrid in Game Summit**, a significant gathering dedicated to promote the video game industry.

Bibliography

[1] Adobe. Guide to cel shading animation. https://www.adobe.com/uk/creativecloud/animation/discover/cel shading.html: :text=Cel%20shading%20is%20a%20computer,falls%20across%20a%203D%20model.

[2] Adobe. Mixamo. https://mixamo.com.

[3] L. Annetta. Video games in education: Why they should be used and how they are being used. 2008.

[4] aphom000. Button ui sound effect. https://freesound.org/people/aphom000/sounds/623175/, 2019.

[5] Unity Documentation. Scripting API. Graph view. https://docs.unity3d.com/ScriptReference/Experimental.GraphView.GraphView.html.

[6] American Psychiatric Association. What is adhd? https://www.psychiatry.org/patients-families/adhd/what-is-adhd.

[7] Blednaya. Fruit store model. sketchfab. https://sketchfab.com/3d-models/little-fruit-store-0cc1a3a35737494380e7cbc31ac52e53, 2020.

[8] Blender. blender.org. https://www.blender.org/features/.

[9] Blender. Voronoi texture node. https://docs.blender.org/manual/en/2.80/render/shader$_n$*odes/textures/v*
text = The%20Voronoi%20Texture%20node%20adds, by%20distances%20to%20other%20points.

[10] Acorn Bringer. Simplistic low poly nature. unity asset store. https://assetstore.unity.com/packages/3d/environments/simplistic-low-poly-nature-93894, 2018.

[11] Inc. Celsys. Clip paint studio. https://www.clipstudio.net/es/.

[12] N. Clark. My biggest revelations of 2018 came from an indie video game. https://www.vice.com/en/article/pa5937/my-biggestrevelations-of-2018-came-from-an-indie-video-game, 2018. Accessed: 2023-03-28.

[13] K. Dobson. The relationship between anxiety and depression. 2002.

[14] Unity Documentation. Audio mixer. https://docs.unity3d.com/Manual/AudioMixer.html.

[15] Unity Documentation. Coroutine. https://docs.unity3d.com/Manual/Coroutines.html.

[16] Naughty Dog. The last of us. https://es.wikipedia.org/wiki/The_Last_of_Us. Accessed: 2023-04-02.

[17] Quantic Dream. Detroit: Become human. https://es.wikipedia.org/wiki/Detroit:_Become_Human. Accessed: 2023-04-02.

[18] Dontnod Entertainment. Life is strange. https://es.wikipedia.org/wiki/Life_Is_Strange. Accessed: 2023-04-02.

[19] "Concerned Ape" Eric Barone. Stardew valley. https://es.wikipedia.org/wiki/Stardew_Valley. Accessed: 2023-04-02.

[20] Free Dimension Forge. Forest models. unity asset store. https://assetstore.unity.com/packages/3d/environments/landscapes/low-poly-forest-environment-package-173273, 2020.

[21] FreddyAbson. Low poly sword model. sketchfab. https://sketchfab.com/3d-models/low-poly-sword-50fcadf2918a459c97373c5325287c89, 2019.

[22] Extremely OK Games. Celeste. https://es.wikipedia.org/wiki/Celeste_(videojuego). Accessed: 2023-04-02.

[23] N. Grayson. Celeste taught fans and its own creator to take better care of themselves. 2018.

[24] Indeed.es. España, cuál es el sueldo al mes de un programador junior. https://es.indeed.com/career/programador-junior/salaries.

[25] JustCreate. Farm models. unity asset store. https://assetstore.unity.com/packages/3d/environments/in
poly-farm-pack-lite-188100, 2023.

[26] Mert Kirimgeri. Unity dialogue graph tutorials. https://www.youtube.com/c/MertKirimgeriGameDev.

[27] "Notch" Markus Persson. Minecraft. https://es.wikipedia.org/wiki/Minecraft. Accessed: 2023-04-02.

[28] Microsoft. Guía de c. https://learn.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/.

[29] Massive Monster. The adventure pals. https://store.steampowered.com/app/396710/The_Adventure_
Accessed: 2023-04-02.

[30] Jean Moreno. Cartoon vfx remaster free. https://assetstore.unity.com/packages/vfx/particles/cartoon-fx-remaster-free-109565, 2023.

[31] Nintendo. Animal crossing: New leaf. https://en.wikipedia.org/wiki/Animal_Crossing:_New_Leaf.

[32] Nintendo. The legend of zelda. https://es.wikipedia.org/wiki/The_Legend_of_Zelda. Accessed: 2023-04-02.

[33] Nintendo. The legend of zelda: Breath of the wild. https://es.wikipedia.org/wiki/The_Legend_of_Zelda:_Breath_of_the_Wild. Accessed: 2023-04-02.

[34] Nintendo. The legend of zelda: Skyward sword. https://es.wikipedia.org/wiki/The_Legend_of_Zelda:_Skyward_Sword.

[35] V. M. Ortiz. The power of video games: How celeste and hellblade address mental health. *The Faculty of the Communication Studies Department California Polytechnic State University, San Luis Obispo.*, 2021.

[36] Sociedad Española para las Ciencias del Videojuego. Cev 2023. https://secivi.org/cev/.

[37] Peluche. Town music jam. https://tangzie.itch.io/town-music-jam-2022, 2022.

[38] O. et al. Pollatos. Differential effects of anxiety and depression on interoceptive accuracy. 2009.

[39] Arnold Rauers. Miracle merchant. http://www.miracle-merchant.com/. Accessed: 2023-04-02.

[40] T. Richardson. 'a short hike' reflects my neurodivergence in a way that feels safe. aipt. https://aiptcomics.com/2021/01/26/a-short-hike-indie-games-anxiety-adhd/, 2021. Accessed: 2023-03-28.

[41] Adam Robinson-yu. A short hike. https://ashorthike.com/. Accessed: 2023-04-02.

[42] Margaret Rouse. What does sandbox mean? https://www.techopedia.com/definition/3952/sandbox-gaming.

[43] RunemarkStudio. Koi fish model. sketchfab. https://sketchfab.com/3d-models/koi-fish-8ffded4f28514e439ea0a26d28c1852a, 2017.

[44] Pixel Sagas. Comic book font. https://www.dafont.com/es/comic-book.font.

[45] Everyday Cinematic Sounds. Footsteps walking on concrete. https://www.youtube.com/watch?v=rAOkC6UAKrgab$_c hannel = EverydayCinematicSounds$.

[46] Meshtint Studio. Low poly chicken animations. unity asset store. https://assetstore.unity.com/packages/3d/characters/animals/meshtint-free-chicken-mega-toon-series-151842, 2019.

[47] Omabuarts Studio. Stylized cat model with animations. unity asset store. https://assetstore.unity.com/packages/3d/characters/animals/lowpoly-toon-cat-lite-66083, 2016.

[48] Ghost Studios. How to make a water shader with foam in unity with urp. https://youtu.be/yJrc_ywpTJw.

[49] Synty Studios. Simply sky cartoon assets. https://assetstore.unity.com/packages/3d/simple-sky-cartoon-assets-42373, 2023.

[50] Digital Sun. Moonlighter. https://store.steampowered.com/app/606150/Moonlighter. Accessed: 2023-04-02.

[51] Ninja Theory. Hellblade. https://es.wikipedia.org/wiki/Hellblade:_Senua%27s_Sacrifice. Accessed: 2023-04-02.

[52] Unity. About shader graph. https://docs.unity3d.com/Packages/com.unity.shadergraph@16.0/manual/

[53] Unity. Cinemachine. https://unity.com/es/unity/features/editor/art-and-design/cinemachine.

[54] Unity. How we built the toon shading, open projects devlog. https://youtu.be/GGTTHOpUQDE.

[55] Unity. Unity 2020.3.27 system requirements. https://unity.com/releases/editor/whats-new/2020.3.27.

[56] ustwo games. Alba: A wildlife adventure. https://store.epicgames.com/es-ES/p/alba-a-wildlife-adventure-93736a. Accessed: 2023-04-02.

[57] María B. Villar. Itch.io game web page. https://mariavllr.itch.io/nebali-the-lost-island.

[58] María B. Villar. Nebali, the lost island - repository. https://github.com/mariavllr/Nebali-TheLostIsland.

Appendix A

Other considerations

A.1 Bibliography of external elements

All the references to the important links and external elements downloaded from different resources used for the project will be referenced in this part.

A.1.1 Referenced links

Github repository for the project [58].
Itch.io game web page [57].

A.1.2 3D Models

Koi fish [43].
Little fruit store [7].
Farm models [25].
Stylized cat with animations [47].
Low poly chickens with animations [46].
Nature environment [10].
Forest low poly model [20].
Low poly sword [21].

A.1.3 Sound

Background music [37].
Button UI sound effect [4].

Walking sound effect [45].

A.1.4 Others

Cartoon VFX effects [30].
Simple skybox [49].
Comic book font [44].