Portfolio

Dear Sir/Madam,

My name is Tiago Fernandes, a Computer and Multimedia Engineer from Portugal.

This document contains an overview of the five projects that I find most relevant for this application (that also happen to be my personal favourites).

It's worth mentioning that all these projects were created and developed during my BSc in Computer and Multimedia Engineering.

Contents

1. GOCards, Final BSc Project

GOCards was developed by me and a colleague with a lot of inspiration from the game "Pokémon Go", with the initial intent of adding a better experience into visits of a Zoo. We thought that we could create a new and fun way to learn about the animals and make the visit more memorable. By collecting cards with information of each animal, the game adds a more entertaining and educational perspective to the visit that is easily accessible to any age group, with a particular focus on younger audiences. We then realized that this idea could also work as a more generalized app to be used in any point of interest, not just theme parks.

Here's a brief introduction from the project report: "Most of the interesting information available when visiting new places is, many times, unnoticed by what our eyes see, obfuscating facts equally important about a certain point of interest. Because of this, visits can become a little monotone, provided by the lack of information or even objectives. In this project, the application GOCards was developed, with the goal to complement visits to theme parks and other public places. Several features were included to help make visits more enriching. The application uses a map of the location, with the intention to guide and suggest points of interest, so the users can find those that are most appealing to them, in an easier way."

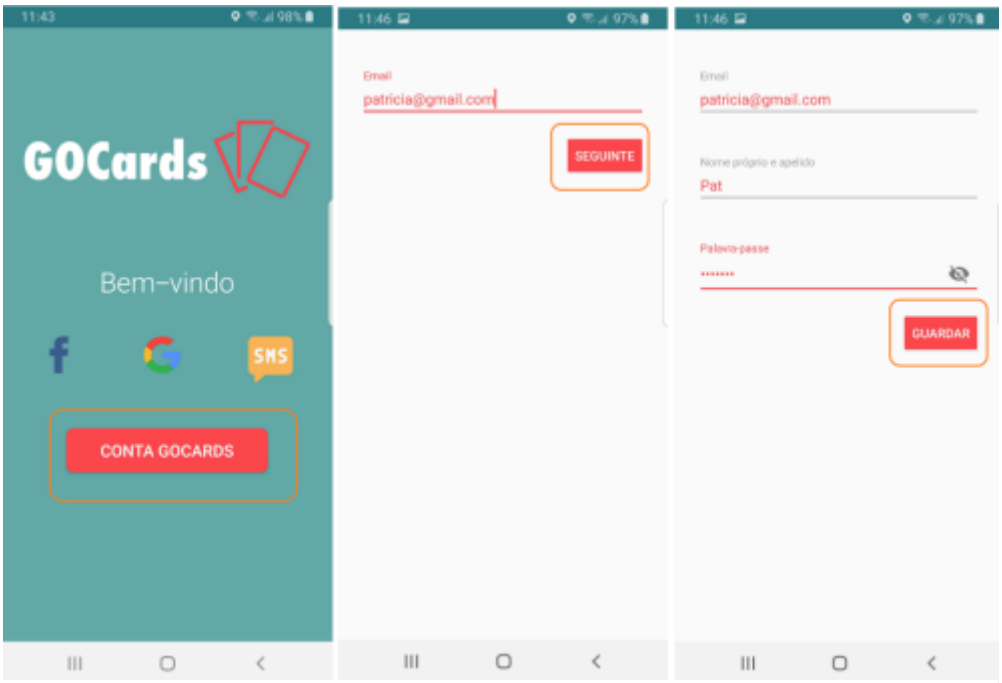

Figure 1 - Authentication and account creation

When opening the app, the user will be asked to log in (Figure 1). This can be done through the Firebase Auth service, using a Facebook or Google account, phone number or by creating an account with an email. Then, we will store the user's information in the database and an image if there is one linked to the user Facebook or Google account.

After the user complete the authentication process, the app leads into the main page with a map of the user's current location and nearby cards (Figure 2). In this main page there is also the QR code scanner, required to unlock some of the cards.

There are two different types of cards: visible and hidden.

- **Visible cards** (Figure 3) are obtained by proximity, and in some cases, it is required to scan a QR code placed somewhere near the point of interest. After the Card pop up appears the user need to correctly answer a question about the Card to unlock it (Figure 7).
- **Hidden cards** are also obtainable by proximity, but the user must solve a riddle to find its precise location. There is also a visual aid called Proximeter (Figure 4) which will increase as the user gets closer to the hidden card. Just like in the visible cards the user must answer a question correctly to unlock the card.

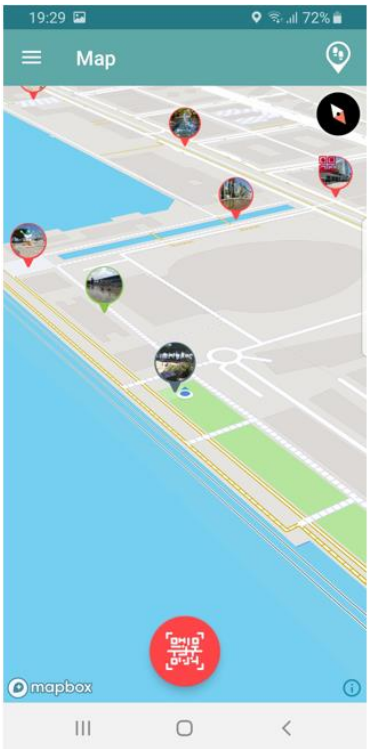

Figure 2 - GOCards Main Page

These cards also have a gamification element: for each question correctly answered, the user earns points. Points are awarded by answering the question correctly and points are lost each time the answer is wrong. The user will also have access to a leaderboard, which provides a more challenging and competitive experience (Figure 12).

The Button in the top right corner of Figure 2 is the suggestion button, it will highlight the closest unobtained card.

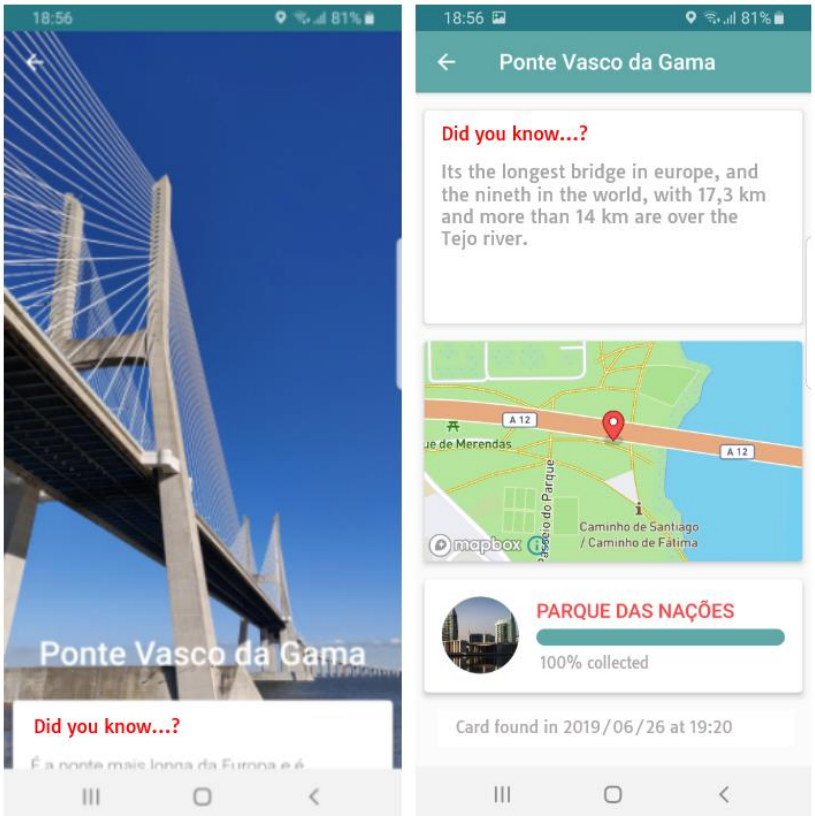

Figure 3 - Visible Card

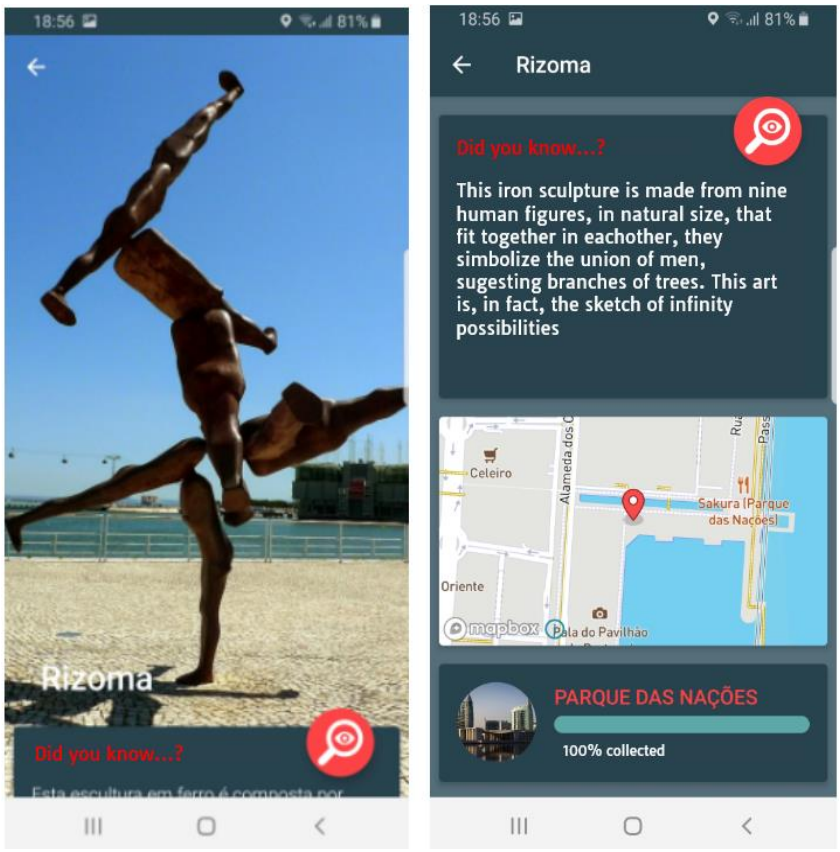

Figure 4 - Hidden Card

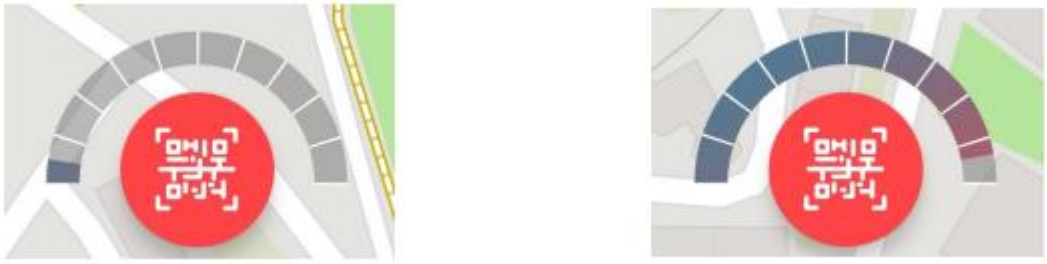

Figure 5 - Proximeter, left image will appear in the screen when the card is far and the right image will appear when the card is near

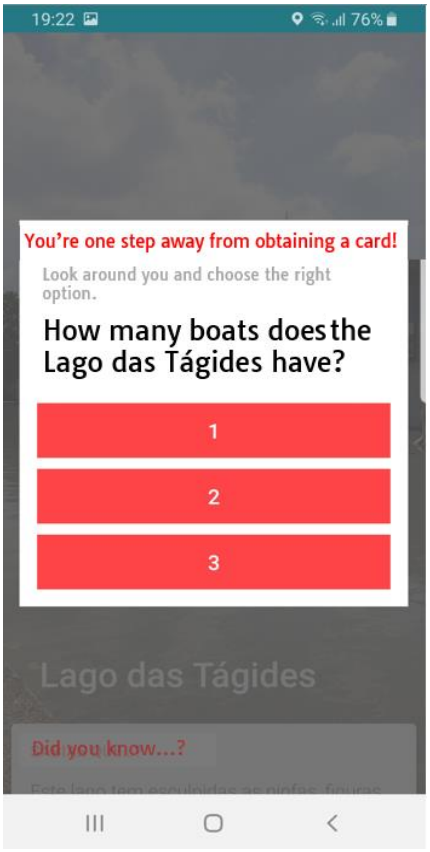

Figure 6 - Example of Question

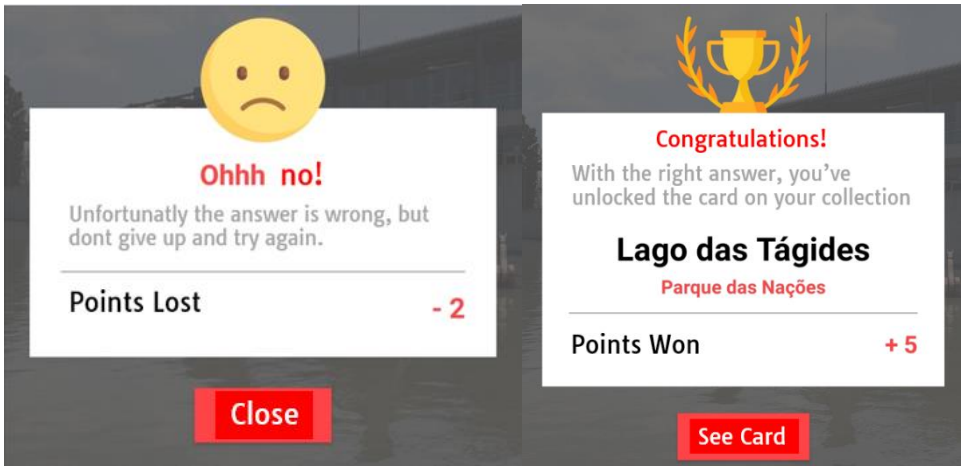

Figure 7 – Pop-up that will appear if the answer is wrong (left image) and if the answer is correct (right image)

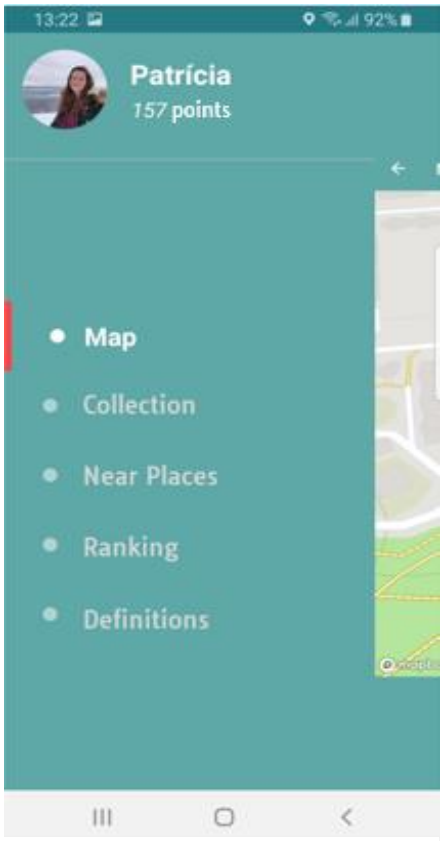

Figure 8 – Menu

There is a menu available (Figure 8), which can lead you to the map (main page) and also to the following options:

- **The User profile**, accessible by clicking the username or image, in which the user can see the number of unlocked cards, points, visited places, and the three latest cards obtained. There are also an edit profile button and a log out button. (Figure 9)

- **Collection**, in which the user can see all the places they visited along with the cards unlocked. When pressing one of the visited places, they can see general information about them and the location of where the visible cards are. (Figure 10)

- **Nearby Places** shows the currently available places to visit, their collection state, and the distance from the user's current location. These are only loaded in a 50km radius. (Figure 11)

- **Ranking**, where the user can see their current rank in red, and then everyone else on the leaderboard (Figure 12).

- **Definitions**, where the user can change the application settings, such as the radius to use in the nearby places search.

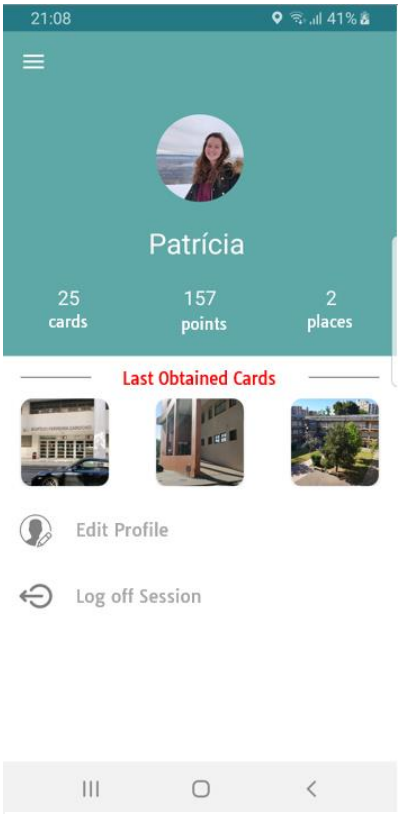

Figure 9 - User Profile

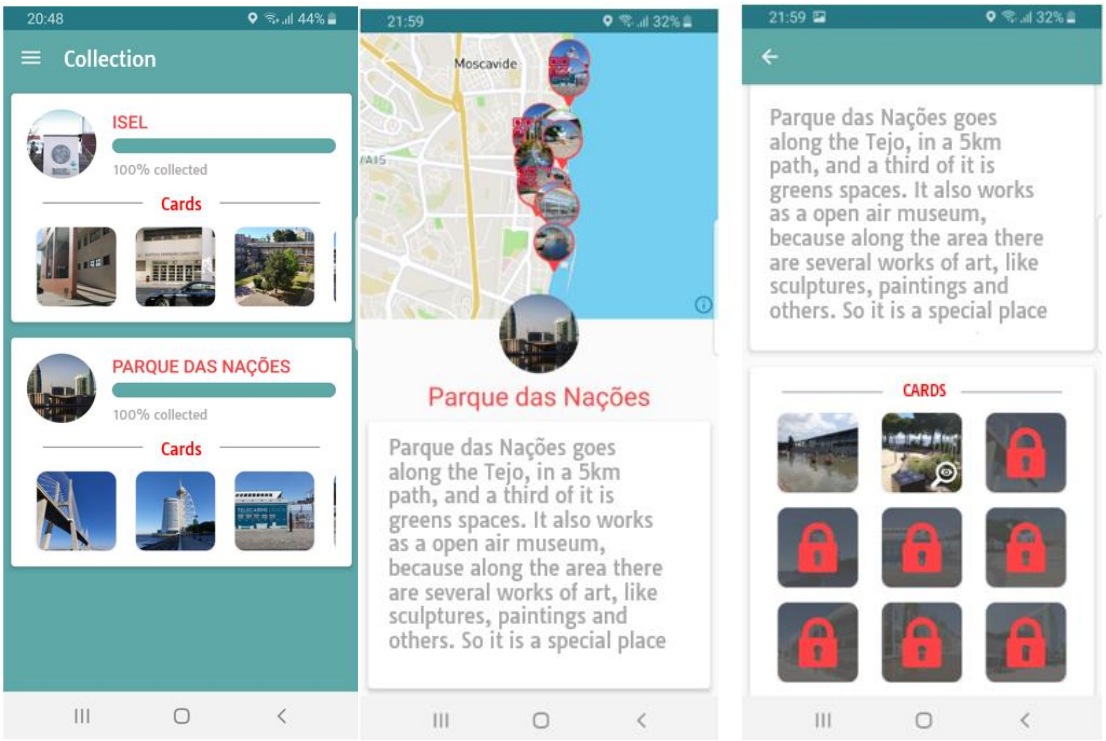

Figure 10 – Collection

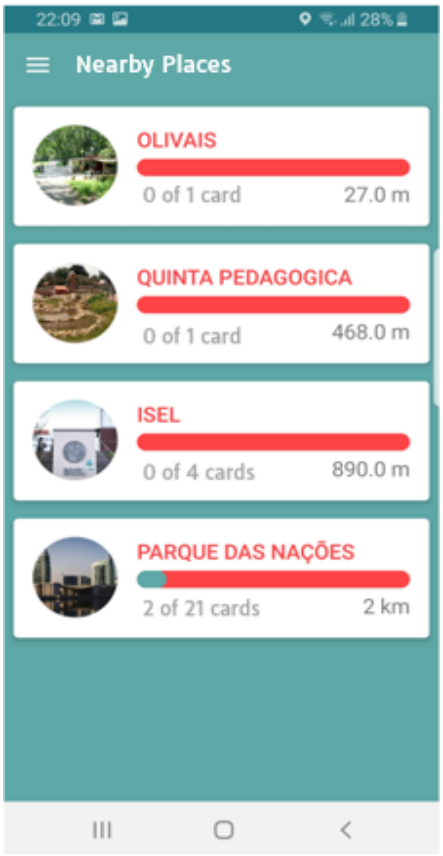

Figure 11 - Nearby Places

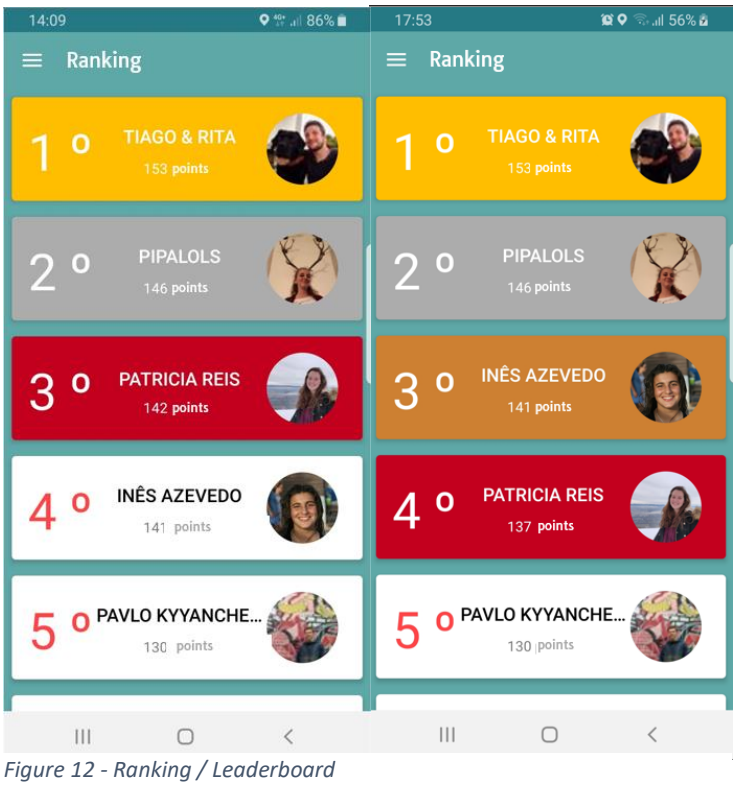

Figure 12 - Ranking / Leaderboard

Regarding the architecture of the project, we had a local database to save the user's data, such as information about close places to visit and cards already obtained, thus invalidating the need to make constant requests to the server.
We used Firebase Auth for the authentication process, ensuring an easier and safer implementation. Furthermore, the QR Scanner is a service from the ML Kit Library. The Map service was provided by MapBox, which uses vectorial maps updated by their own servers every time the user moves on the map.
The Back-End was implemented by us, using Node.js and Restify to create the APIs and the Database is in MySQL. It's important to note that our architecture is generalized to allow several users to interact with the back-end component simultaneously.

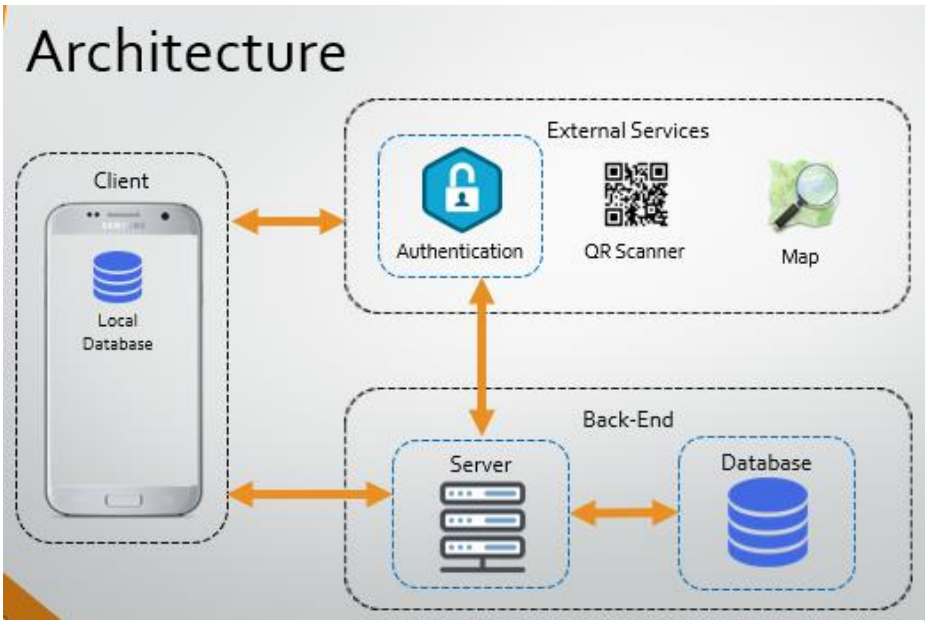

Figure 13 - Architecture of the app

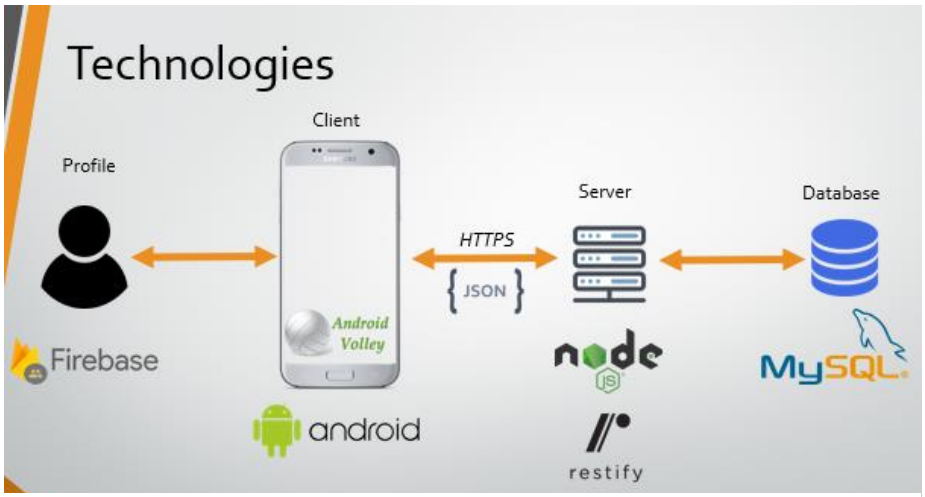

Figure 14 – Technologies used for the implementation

As the project was implemented by both me and my colleague, we both had different contributions to it. The most relevant contributions I have made to this project were the back-end components of the project, that being the Database and the Server.
Since we didn't have previous experience with Node.js or Restify, we had to learn while doing the project, with assistance from our project coordinator. Through Node.js we choose a REST architecture for communication with the user side. This allowed us to use the HTTP protocol, where we used the request-responses along with the URL to define what operation is executed.
The data was handled using JSON formatting. Along with Node.js, we used the Restify Library, which aided in creating an HTTP server designed to be used along with the JSON requests. Enabling us to build simple REST Web Services.

For the communication between server and database we used the promise-MySQL library, which deals with the data in an asynchronous form. This gave us access to use a connection interface with the MySQL server using Promises.

I can easily say that this was the hardest, but most rewarding project I've been a part of. We used all the knowledge we learned previously from our BSc programme and then went even further. We had to learn new technologies to adapt to our project needs, and also grew as a team and as individuals, by continuously sharing our progress and collaborating to ensure that we both knew what we were implementing. It was not an easy project as we started with a different set of technologies that were not adequate for our plans, and we had to find other ways to implement our project to work as we designed.

If you are interested in learning more, here is a link that leads to a repository with the project's PowerPoint presentation, final report, code, images and videos of the working product.

2. The Hole in Worlds, a Unity Project

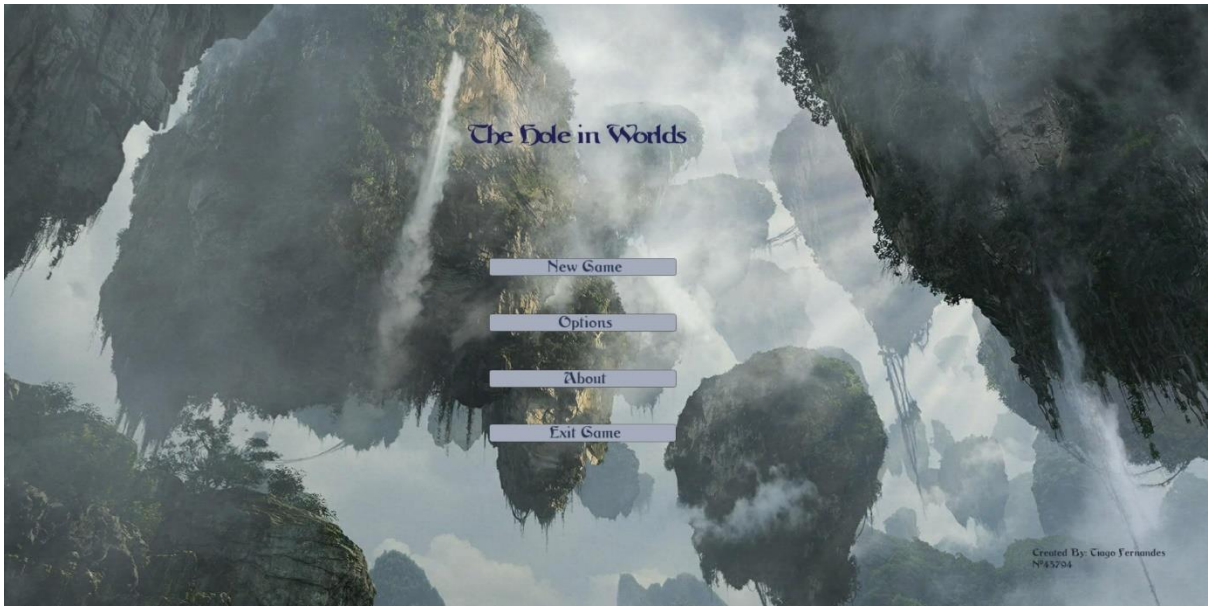

Figure 15 - Main Screen

"The Hole in Worlds" is a 3D Third Person Action Fantasy Game created during a class on Animation of Virtual Environments.
This was a solo project, where I used a collection of modules from the Unity asset store, Unity built-in assets and characters from Mixamo. As the class was an introduction to Unity, the project began from learning the basics of level design. While trying out the tools provided by Unity I had a lot of ideas for the project, but due to the scope of the class and time constraints the game was left on its initial state: a group of floating islands for the player to explore.

In "The Hole in Worlds", the player must search for relics, artifacts and defeat enemies wandering floating islands, set in a semi-open world. These floating islands are populated with beings and structures from different dimensions. The player's character can fight using their bow and flaming arrows to inflict damage to unsuspecting enemies or use stealth and try to go unnoticed while picking up the relics.

The game ends when all the relics are collected or if the character's life points reach zero.

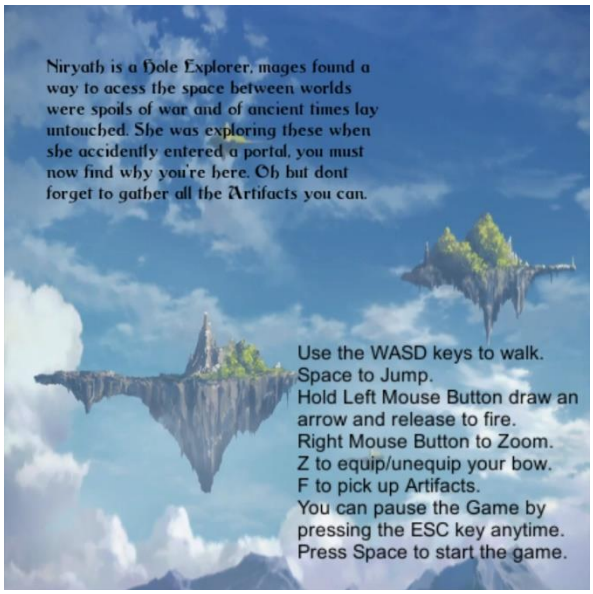

Figure 16 - Information displayed before the game starts

The game has 3 different islands to explore, the first island is the "spawn" where there are no enemies so the player can get accustomed to the game. The player can then choose to explore the "Rainy Island", which is the closest, where they will find monsters who are easy to target because of their size. In the largest island, "Castle Island", there is a castle filled with dark knights where stealth is key, if the player is seen they come in numbers that are very hard to beat.

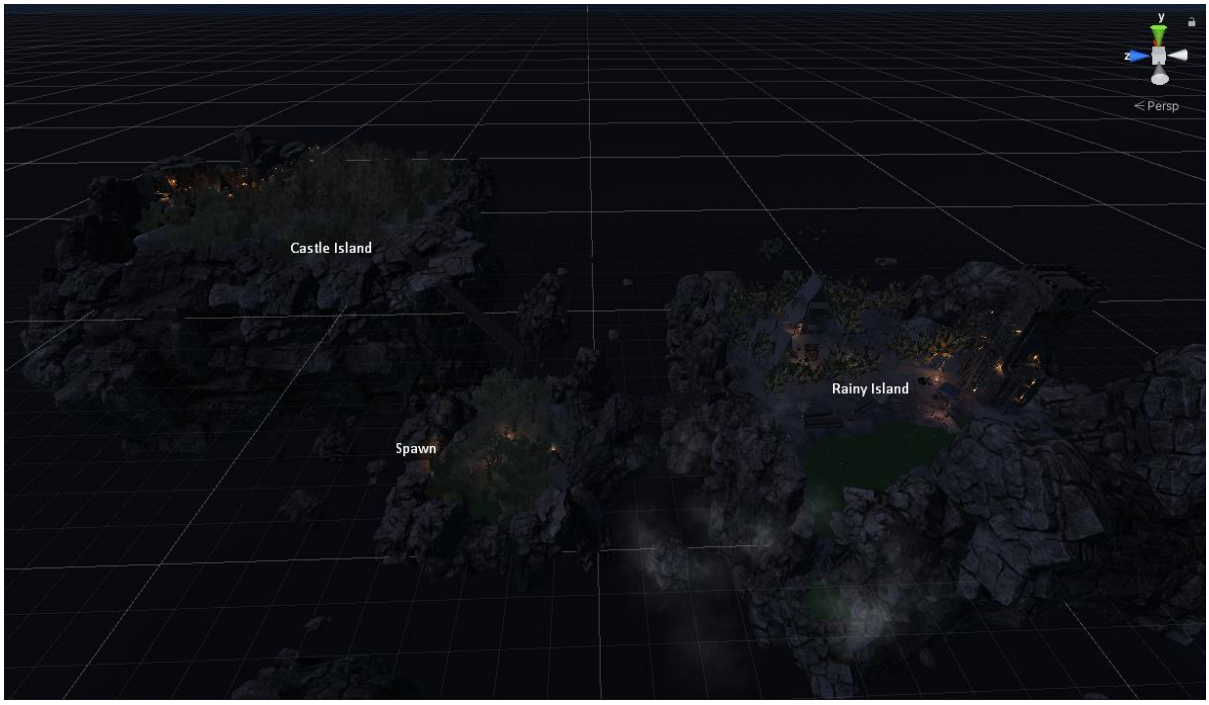

Figure 17 - Floating Islands

Along the way the player will find several relics and artifacts hidden away, often located "right under their noses". These hidden objects will have a ray of light shining on them and, when near, a prompt will appear and the player may pick them up by using the E key (Figure 23).

Figure 18 - Book Artifact as an Example.

There are two enemy types available for the player to defeat:

- **Monsters** (Figure 19) that roam the "Rainy Island".
- **Dark knights** (Figure 20) that protect the castle.

Both enemies use the same animation tree, sharing their idle, wander, patrol and chase states. Nevertheless, they each have their own animations and sounds.
The enemies have simple Artificial Intelligence that allows them to be vigilant, and when the player is within their field of view, they will chase the character and then attack them when close enough.
The enemies have health bars (Visible in Figure 22) that appear above them when hit.

Figure 19 – Monster

Figure 20 - Dark Knights

Figure 21 - Player Character

The player character is an archer, with animations for walking, jumping, rolling, drawing the bow, shooting arrows, picking up relics, unequipping and equipping the bow. The animation tree was made by myself with several key binds connected to each animation:

Key	Action
A, W, S, D	Movement
Space bar	Jump
Double W	Roll
Z	Unequip/Equip the bow
F	Pick up objects
Left Mouse Button	Draw the bowstring and shoot
Right Mouse Button	Zoom

Also, whenever the character takes a step, you can hear the sound throughout the walk animation.

The world itself is filled with assets that help enrich the ambient, the player can hear animals that they would expect to be in forest areas and see particles that resemble fireflies.

Figure 22 - Player fighting an enemy

Figure 23 - Player interacting with a relic

The basic character movement was implemented during the class. The player's aim was made by me following Unity tutorials. My ambition was to make a more unique approach where the player would have more control of where the bow was aiming, but wasn't quite able to complete it and the final implementation only let the player aim the bow by walking in the direction they wanted to shoot.

There is a character script for both the player and the monster which saved their common characteristics like movement speed and health points. The monsters would then have a script for their field of view, along with some triggers for the change between animation states.

The player has more scripts, one to handle camera collision, another for the camera itself, a script to insert the footstep sounds correctly, one for item pickup, and finally the player controller, that handles all the movement, aim and triggers for animations.

The User Interface was made using simple scripts for the characters health bar and also to handle all the buttons in the menus.

This was one of the most exciting classes during my BSc and I spent a lot of time working on developing and improving the project by including new features that would enrich the player's experience. One of the most challenging issues was the players aim control, which did not end up how I envisioned it. Overall, it was an amazing experience that further confirmed my intention to learn more about game development in the future. Creating the ambition to learn more about Unity and other game engines.

If you would like to try the game or learn more, here is a link to a repository where you can find the executable for the game, a video with a small demo and the images used in this pdf.

Figure 24 - Game Over Screen

Figure 25 - Inside the Castle

Figure 26 - Rainy Island

3. MineAway, Android Game

Figure 27 - Authentication Screen

Created during the class "Mobile Application Development", MineAway was a variation from the original Minesweeper. The goal of this class, and consequently the goal of this game, was to create android applications that were ready for different mobile phone resolutions. This was the final project of the class, and a solo one.

The goal of MineAway, much like Minesweeper, is quite simple: to collect gold nuggets while avoiding the dynamite in the caves.

When the user first opens the application, they need to pick a way to login. With the help from the Firebase library, the user can choose their Facebook or Google account, mobile phone or to create an account using an email (Figure 27).

After logging in, there are three buttons: New Game, About and Configuration (Figure 28). After the user completes one level, a button will appear to Continue, along with the leaderboard.

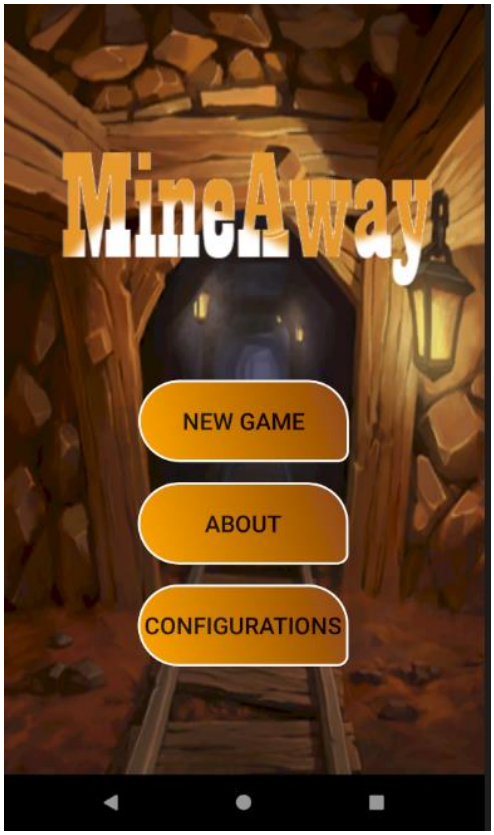

Figure 28 - Main Menu.

The New Game button will lead into a new instance of the game starting from the basic level, whereas if the user presses Continue, he would return to the last instance saved. The About button shows basic information about the app. In the Configurations page the user can control the app's vibration and sound.

The game screen (Figure 29) has the grid with the game itself and, on the top left corner, there's the restart button ("Restart") that allows restarting the grid, and a button to return into the menu ("Back"). On the top right corner, the username is displayed ("Name") along with the total gold collected so far ("Total Gold").

The grid grows larger as the user completes the levels, adding more dynamite and increasing the game's difficulty. The user's total gold marks their place in the leaderboard. Every time the user loses, the grid will reset and randomize the position of the dynamite. Gold is only collected after a successfully finished level.

Figure 29 - Game Page

MineAway's authentication process uses the Firebase Auth to simplify the process.

The game itself is a grid with a random number of dynamites and gold depending on the size (columns and rows increase by each successful clear). Each grid cell may have one of three different states: gold, dynamite and empty. If the grid cell is neither a gold nor dynamite, it identifies the number of adjacent dynamite (with the corresponding number, from 1 to 8). In the case there aren't dynamites surrounding the mentioned grid cell, it remains an empty space. If the grid cell is dynamite, an explosion animation is played, and the user fails the current level. When failing a level, all grid cells with dynamite are displayed, and the player would have to press the restart button to try again.

If you're interested in learning more, here is a link to a repository with the project's code.

4. Ecosystem Simulation/Game, Project in Java (Processing)

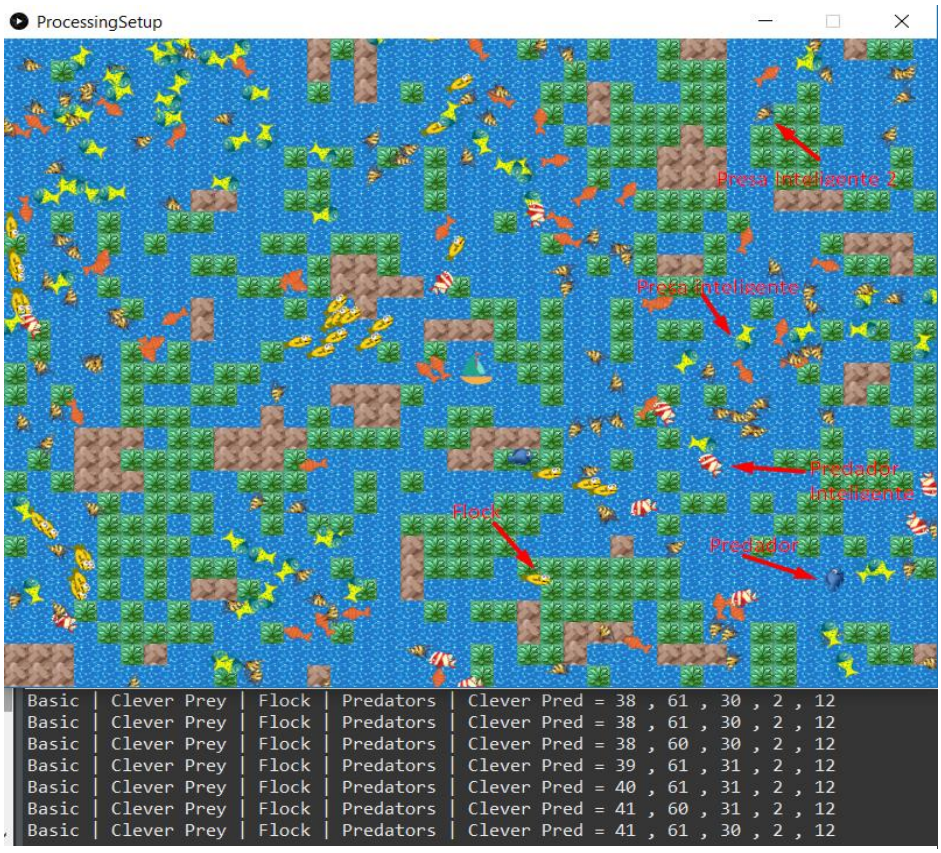

Figure 30 – Simulation

This project was implemented for the "Modelling and Simulation of Natural Systems" class, where the main objective was to improve our knowledge of object oriented programming and simulate a natural system. The theme of the project was renewable resources and how these could be explored for an unlimited amount of time, while considering ecosystem stabilization and the regeneration ability of each resource.

This was a collaborative project, developed by me and a colleague, and as our renewable resource we choose fish.

Accordingly, we created a simulation/game of the ocean, consisting of prey and predators. The terrain is a 2D grid that may have three different states:

- **Water**: represents the empty state of a grid cell.

- **Seaweed**: represents food for the prey, and randomly generates on a water tile.

- **Rock**: an obstacle on which fish cannot survive and loses energy while standing on it.

The terrain is randomly generated, but a rule is applied to make sure it maintains the correct percentage of states for the ecosystem to exist.

There are two different types of prey:

- **Simple prey** that roams in flocks through the ocean and solely avoid obstacles.
- **Smart prey** that also avoids predators.

Each of these types of prey have a level of energy which they consume to stay alive and must pass through seaweed in order to regenerate. The prey dies when its life reaches zero or if it's eaten by a predator. The prey reproduces when their energy reaches a certain point, this energy amount is divided by both the progenitor and offspring.

Predators also have two different types:

- **Simple predator** that roams and chases prey when they are within its field of vision.
- **Smart predator** that avoids obstacles and chases the closest prey.

Regarding energy, reproduction and death, the predators function in the same way as the prey.

Lastly, there is a **boat** that the player can use to change the balance of the ecosystem and catch any nearby fish, interacting either positively or negatively with the ecosystem.

The simulation was implemented using Java and the Processing library, allowing us to create a canvas and draw these "vectors" hidden as little fish (boids). The first step was programming the grid and its cells, followed by the world with its population and terrain. Then, using mathematic calculations, the boids (vectors) were assigned movement patterns. A general animal class was also implemented, from which every prey and predator are children, making this a good example of polymorphism in object-oriented programming.

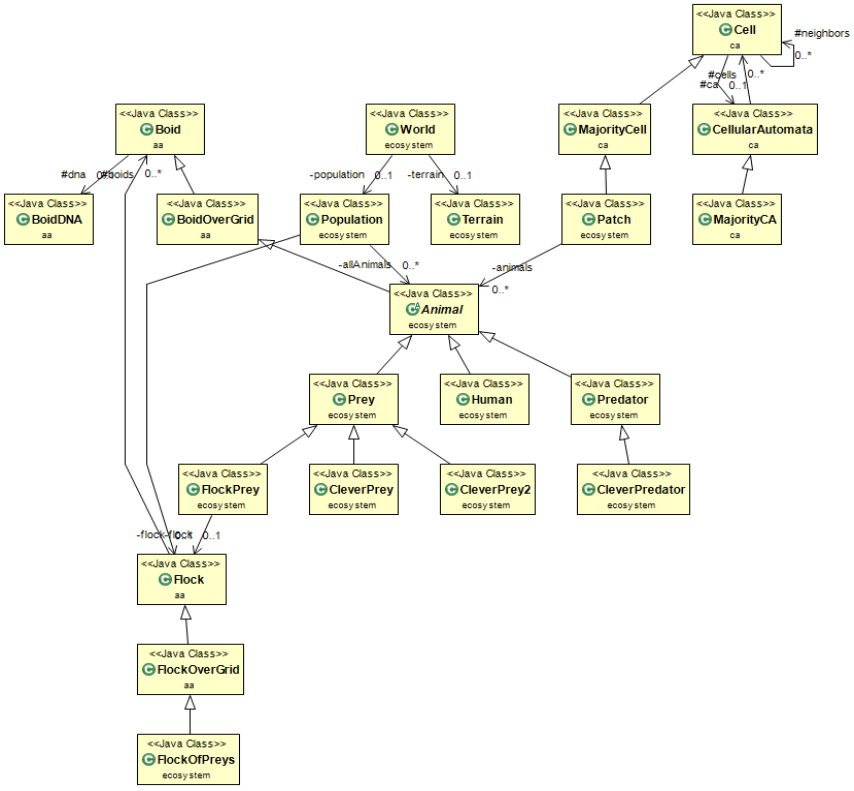

Figure 31 - UML of the classes

If you would like to know more, the code used in this project is present in this link, along with a small demo video of the simulation.

5. 2048 Trine, Game made in Python

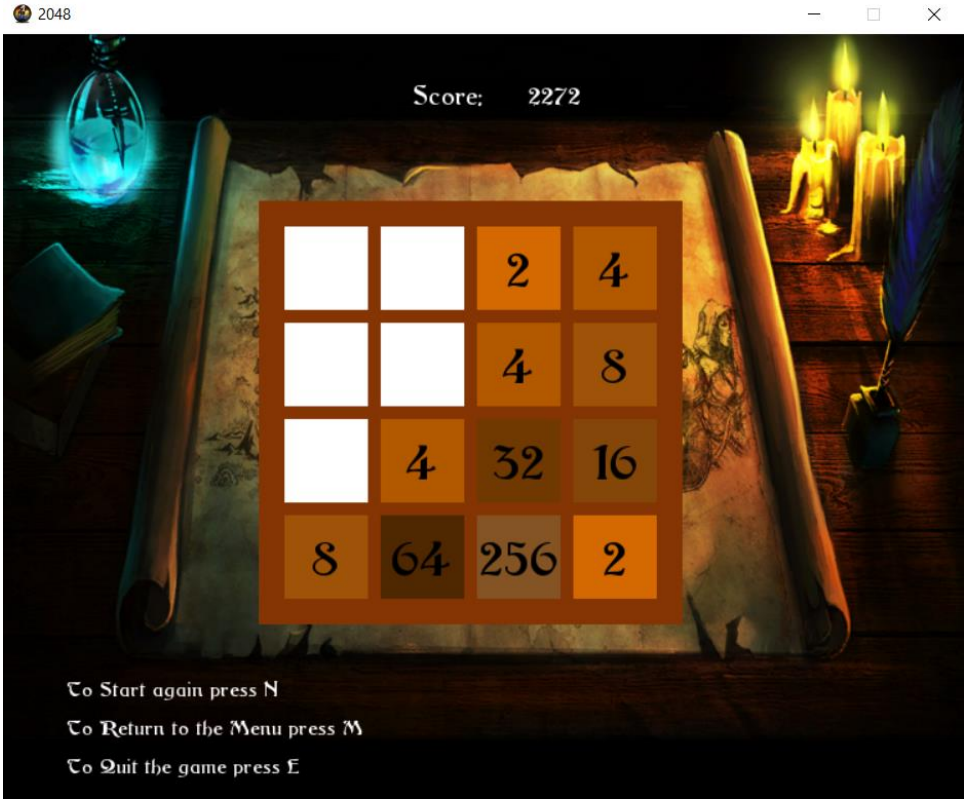

Figure 32 - Game Screen

Trine was the first game I developed. It was a project for the "Discrete Mathematics and Programming" class, where we were assigned the task of designing the game "2048" with all the twists we wanted. I decided it would be fun to add storytelling and a medieval component and, inspired by the game "Trine", I wrote a small story to give the player motivation to finish the game (that is, reach 2048).

2048 is a single-player sliding block puzzle game. Its objective is to slide numbered tiles on a grid to combine them and eventually reach the number 2048. The player can continue the game even after they reach that number. Every time they slide the grid, a new number 2 or 4 appears on an empty grid cell. The player loses when there are no possible moves. The game's base logic was created during each lesson to help us understand programming basics.

In addition to the 2048 game, this implementation has a leaderboard (Provided by the professor). This leaderboard was a means to connect all games developed in class, and for the students to share their best scores.

Figure 33 - Main Menu

The main menu (Figure 33) has four available buttons: the player can watch a short introduction movie by clicking the "I" button, go directly to the game by pressing "N", explore the instructions page (Figure 34) by clicking "H", or close the game by pressing "E".

The game was implemented using Python, and the library Pygame was used for the User Interface. The leaderboard was updated by sending a txt file to a server whenever a user lost a game.

Albeit not taught during the "Discrete Mathematics and Programming" class, I learned how to implement an introductory section by using a list of images and sound files that played and changed through time, basically it was a script that resembled a short movie, based on the original game "Trine".

If you would like to learn more, here is a link to a repository with the code for this game.

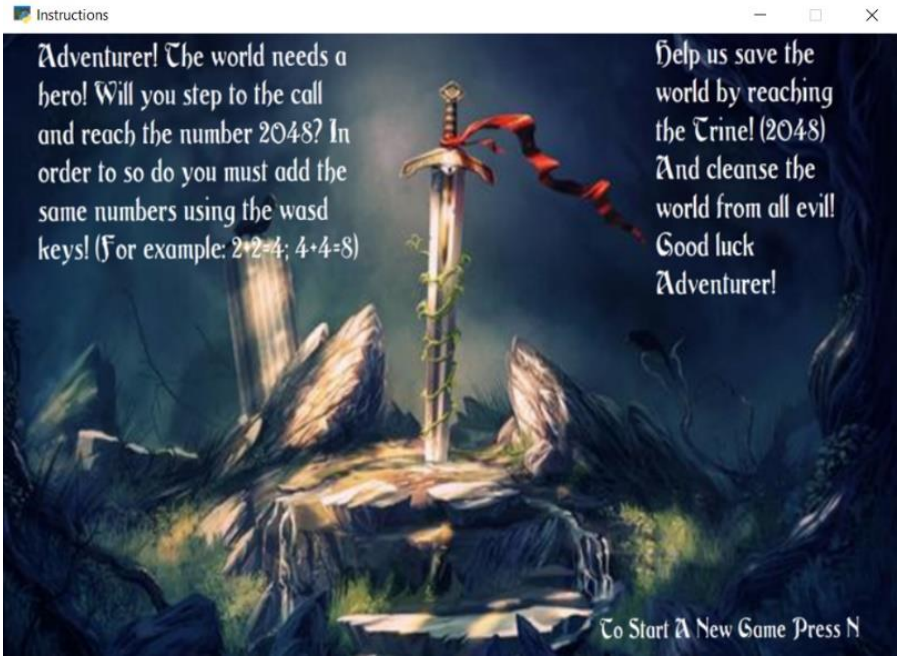

Figure 34 - Instructions displayed after the introduction

Thank you for reading my portfolio. If there is any question about any of these projects, feel free to contact me in the ITU application portal. Here is a link to my website https://wyrekore.github.io where you'll be able to find more information, such as my CV, LinkedIn, and other projects.

Best Regards,
Tiago Fernandes