A video game to simulate the managing of an hospital

Aleksey Vyachislavov Ovseychik

Final Degree Work

Bachelor's Degree in
Video Game Design and Development

Universitat Jaume I

July 1, 2021

Supervised by: Begoña Martínez Salvador

To all of my friends for helping me stress out and forget about the project
for a couple of hours.

*

To my parents for supporting during this four years.

*

And to you Ana,
for listening me talk for hours in some strange language
and giving me advises even if you did not even know what I was talking about,
for supporting me every single day during this project
and cheer me up when I was lost.
Thank you

Acknowledgments

First of all, I would like to thank my Final Degree Work supervisor, Begoña Martínez Salvador, for her patience and guidance through all this journey.

I also would like to thank Sergio Barrachina Mir and José Vte. Martí Avilés for their inspiring LaTeX template for writing the Final Degree Work report, which I have used as a starting point in writing this report.

Abstract

The aim of this project is to make the player understand how hard the medical stuff works and how difficult is to put together all the people, infrastructure and equipment in order to ensure our right to a free , universal and high quality medical attention.

Contents

CHAPTER 1

Introduction

Contents

This chapter must reflect what is going to be done during the development of the work. Although the fundamental point is to state the objectives of the presented work, it is also interesting to comment on the need, idea, etc., that motivates it, and the state from which it was started.

1.1 Work Motivation

What motivated me to do this project is to report the lack of resources that the public healthcare system faces.

Also, I wanted to report how the healthcare staff has to try to help any patient despite all. This total focus on the patient brings a lot of mental problems for not always being able to do it [2].

This project was chosen because in the last pandemics our healthcare system has been overcrowded. Because of this many people couldn't be treated as they deserve.

I want to report this situation and I want to make people feel it first hand.

The goal of the project is to teach every player that the public healthcare system is a basic right of every single citizen of our country.

To achieve universal attention we need more funds, more people, and the effort of all of us to overcome the actual situation.

This goal will be accomplished by the game's core mechanics. These mechanics were designed to make the player suffer the stress of building and maintaining a hospital. All of this disposing of a small number of resources.

Also, as in a real hospital, the game won't give a single moment to relax.
The patients will come non-stop. The player will have to be constantly attentive and making life-changing decisions. These decisions will make a huge impact on the patients. They will make the difference between life and death. They will also give the player total responsibility for the patients' lives.

The player may read every single one of the patients' clinical history. It is essential to understand the nature of their illnesses and treat them correctly.
The players won't be controlling a special character. They will manage the entire hospital. So in fact, they will be controlling every single worker of the medical staff.

This means that all the decisions and the prime responsibility lie on the player.

The implementation of this mechanic will make the players feel in the position of the healthcare staff. Also, it will enable them to understand the work and the effort that they put in favor of our health.

The aim of the mechanics is that the player feels mentally exhausted at the end of a play session. This is for making them feel as tired as the medical staff feels after a long work shift.
Regarding the art style, I will use low poly art [5] (see Figure 1.1) to make the visual aesthetic look simple.
The reason is that this game aims to make the player feel exhausted and understand the stress we put on the public healthcare workers.

I don't want the player to be distracted with the art or trying to build a beautiful and visually pleasant hospital. Instead, the player will need to focus on the effectiveness and to be able to treat correctly every single patient that crosses the door.

1.2 Objectives

The goal of the project is to teach every player that the public healthcare system is a basic right of every single citizen of our country.

FIGURE 1.1: Sample of low poly art

To achieve universal attention we need more funds, more people, and the effort of all of us to overcome the actual situation.

My objectives with this project are:

- Develop a game that makes the player think and raise awareness about the lack of resources in the public healthcare system.
- Implement mechanics that allow the player to be in the position of the healthcare staff.
- Implement mechanics that allow the player to manage and build their hospital.
- Deliver a unique game experience for every player.

1.3 Game Dynamics

In this section, I will explain the dynamic of the game and how the objectives are going to be reached. The mechanics and the system I will mention in this section will be described in detail in the following chapters.

The player will start with a fixed amount of money, he will need to invest that money in order to build the hospital and hire workers.

The player will have to buy rooms to build the hospital and hire workers to populate it. These rooms must be connected in order to be used. The hired workers will search for a place to work according to their role if they do not find it they will go to the resting room.

Each patient will be randomly generated with an illness, this illness will have a treatment. In order to treat a patient, the hospital will need to have the specific equipment and staff to do the procedure. For example, if a patient has a broken leg to treat him the hospital will need a radiology room and a radiologist.

Patients work in a similar way, they will search an empty place in the room they need to go to, if they do not find one they will go to the waiting room.

The patient first will go to the reception, after that to a consult, there the doctor will assign a treatment to the patient.

The player will be redirected to a series of doctors to be treated. If for some procedure a doctor or equipment is missing the patient will return to the waiting room.

In the waiting room, the patient's patience bar will start to decreases. If the patience bar reaches 0 at some point he will return to his home and the player will lose a certain quantity of money.

If the patient completes the treatment successfully the player will gain a certain quantity of money.

Every time more patients will come to the hospital. This will force the player to continue building the hospital and hiring workers in order to attend to all the new patients.

1.4 Environment and Initial State

This game will be developed using Unity Engine. The programming will be all done by myself excluding the usage of some libraries. The art will be partially done by me and for the other part, I will search for free assets on the Internet. This is due to the size of the project, the number of assets that are needed, and the time that I have to develop it.

All the art must resemble the aesthetic of the low poly art. This will improve the performance of the game and will help me dealing with the creation of the assets. This is due to the simplicity that characterizes this aesthetic.

1.5 The state of the healthcare system in Spain after the pandemics

One of the main objectives of this game is to raise awareness about the state of the healthcare system in Spain.

Our medical staff was the first line in the battle with COVID-19. and are the first to suffer the consequences of the pandemics.

According to this report one-half of the nurses are in danger of suffering from mental illnesses [1]. A 15 percent ensures that they had psychological help during the pandemics.

This situation is due to the overpopulation that our hospitals. This report marks that 8/10 nurses claim that the hospitals have a lack of medical staff [2].

This battle took the lives of more than 17,000 sanitarians all over the world that fought[3] this virus. They gave their life for us.

This game is my way to honor these people and to contribute even if it is in a slight way to fight against the COVID-19.

CHAPTER 2

Planning and resources evaluation

In this chapter, I will detail the planning that I will be following up during this project.

Contents

2.1 Planning

First of all, I planned the overall of the tasks on which I will work during this project(see Table 2.1).

This table served me as a guide to developing more precise planning of the project (see Figure 2.1). The planning was made using a Gantt chart, this planning will not be definitive. It will work more as a guide rather than strict planning.

In the reality, my work differed quite from the initial planning. After the finalization of the project I made another Gantt chart to visualize the job that I have done(see Figures 2.2, 2.3, 2.4).

For a comfortable visualization, this Gantt chart has been split into 3 parts. These parts are the sprints in which the project was split, in 4.1 I will explain in more detail the work methodology I followed in this project.

This chart differs from the first one due to few reasons:

- In the first chart I did not take into account the June exams.
- I implemented pretty all the functionality specified in Figure 2.1 but I changed the order of the implementations. This change was made to optimize the time and for being able to make small tests. First I planned to implement single mechanics and debug them individually and interacting with others. But it makes more sense to implement together a group of mechanics that are related and then test them all together.
- Also in the middle of the project I decided not to put so much effort into the art. Art is not my strong point and I prefer to deliver well-polished mechanics and interesting gameplay. Even if I had invested a lot of hours in modeling the result will be some mediocre characters and props.

Task	**Hours**
Creation or importation of assets	50
Hospital management mechanics	110
Patient management mechanics	80
Implement miscellaneous functionalities	70
Research about some diseases and the workflow of a hospital	20
Final memory	10
Presentation of the project	10
Total	**350**

TABLE 2.1: Resume of the planning

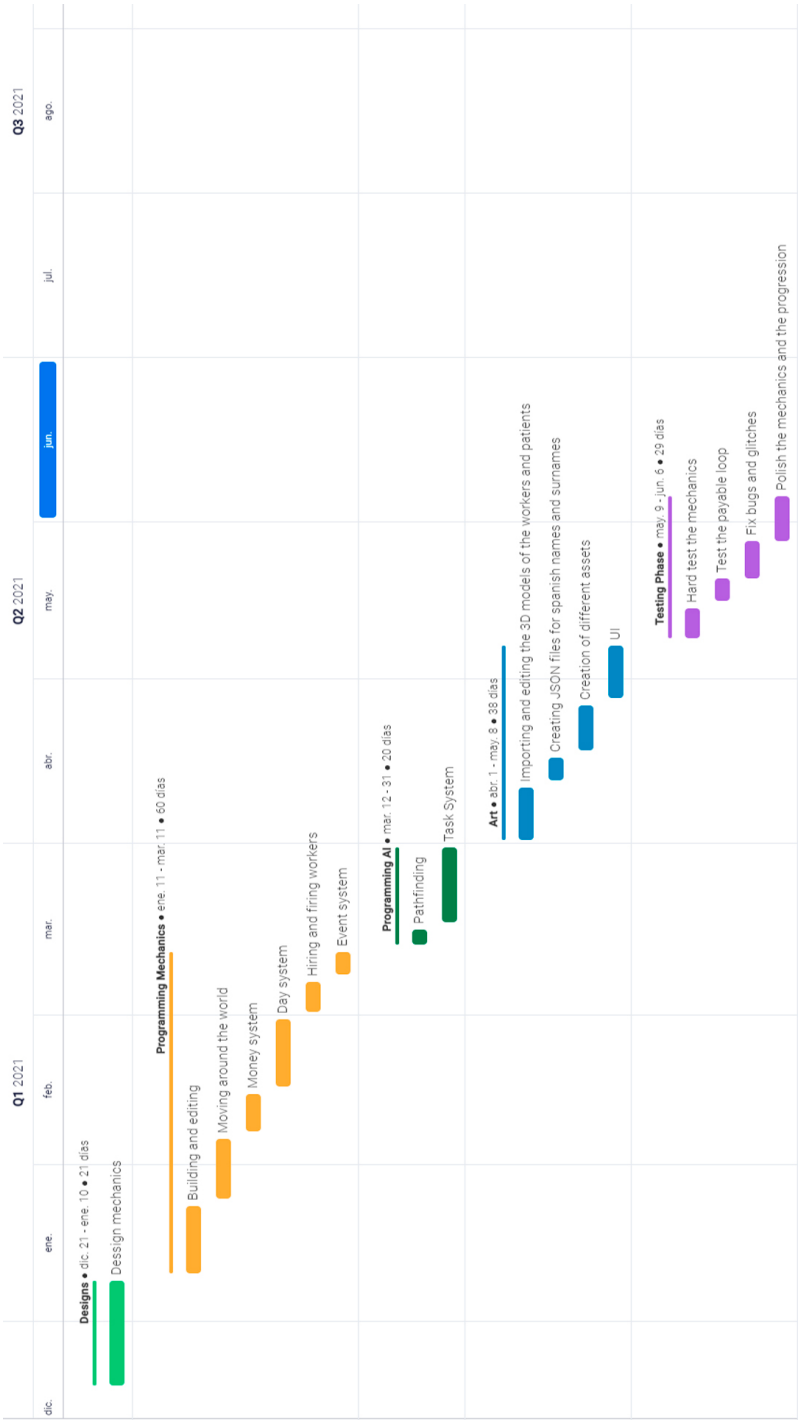

FIGURE 2.1: Gantt chart of the first planning

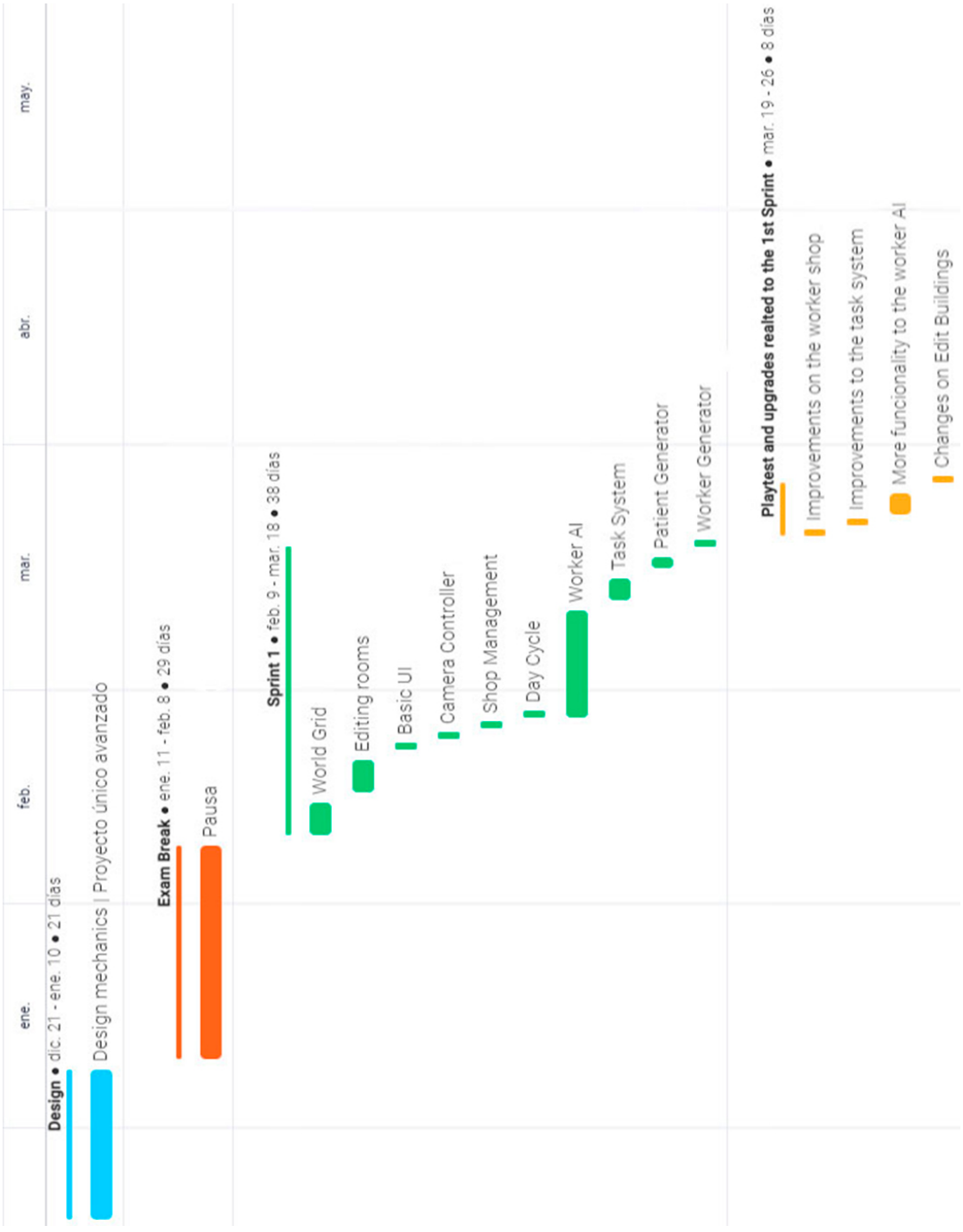

FIGURE 2.2: Gantt chart corresponding to the 1st sprint

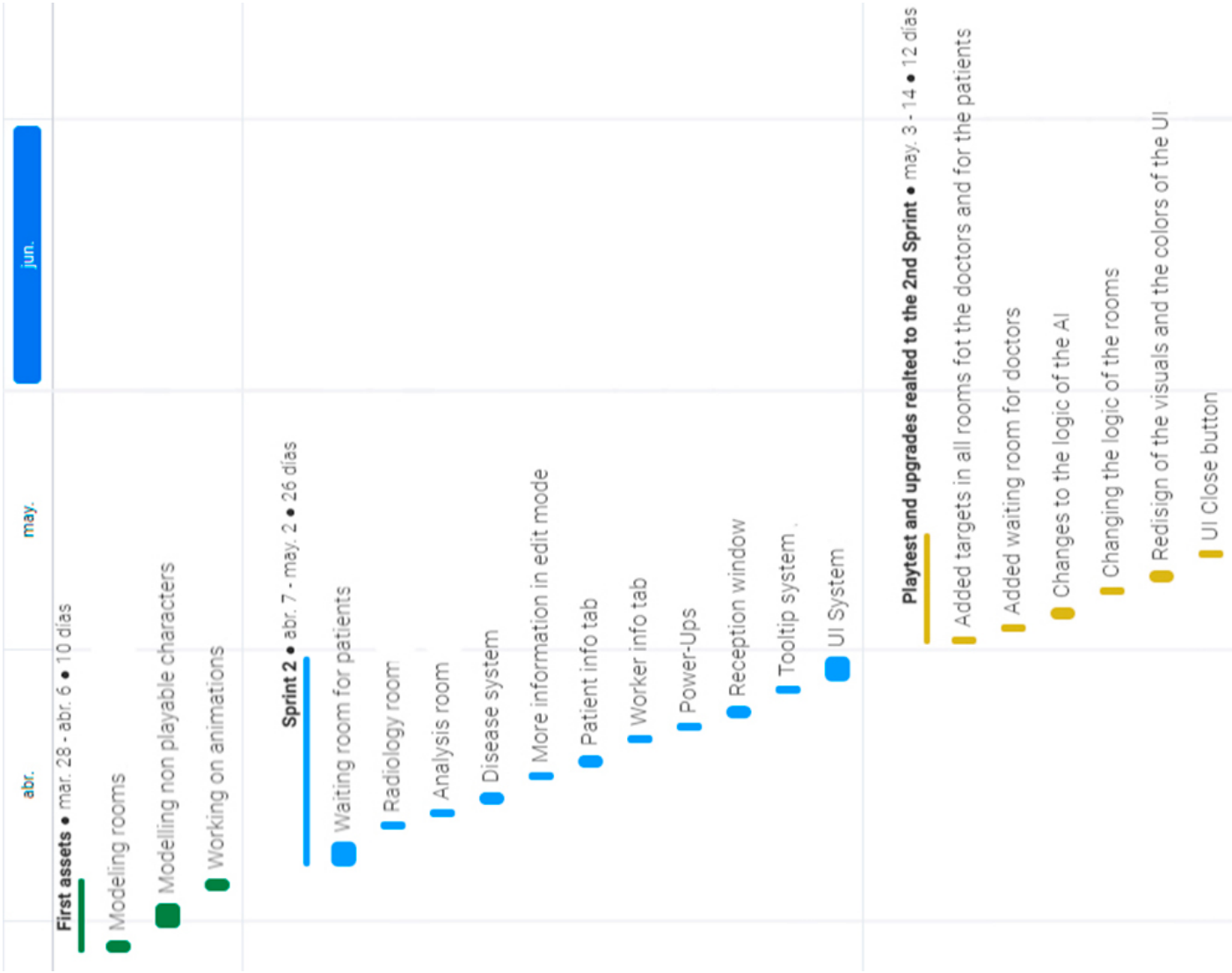

FIGURE 2.3: Gantt chart corresponding to the 2nd sprint

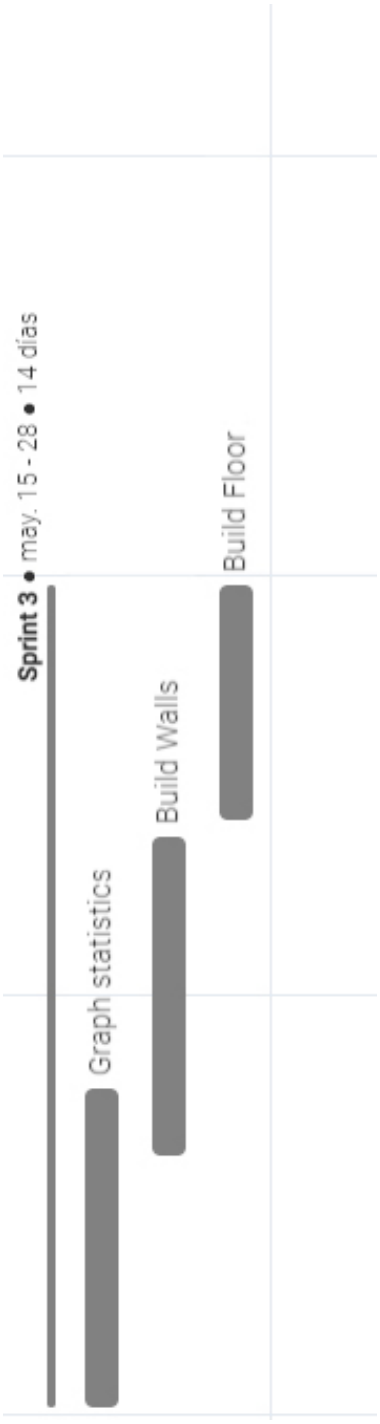

FIGURE 2.4: Gantt chart corresponding to the 3rd sprint

2.2 Resource Evaluation

The only human resource that I will need to develop this project is my time. I worked nearly 315 hours on this project. The average yearly salary of a junior game programmer is around 28.000€ [4]. If we divide it by 14 that is the number of payments that the average worker gets paid in Spain (12-month salaries + 2 extraordinary salaries) we get 2.000€ per month. In Spain, a regular worker works 40 hours a week, which is a month that means 160 hours of work. As mentioned above I worked 315 hours, which approximately means two months of work.

In conclusion, the time spent on this project on average had a value of approximately 4000€.

In regard of the equipment I will use:

- A laptop computer.
- An USB mouse.
- Headphones.
- A monitor.

I will also need the following software:

- Unity 3D (Student license)
- Visual Studio (Student license)
- GitHub (Free Version)
- Adobe Photoshop (Student license)
- Blender 2.91 (Free Software)
- OverLeaf (Free)
- GoogleDocs (Free)
- Monday (Free version)
- Lucid Charts (Free version)
- App Moqups (Free version)
- Visual Paradigm (Free version)

CHAPTER 3

System Analysis and Design

Contents

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

3.1 Requirement Analysis

To carry out a job, it is necessary to perform a preliminary analysis of its requirements. In this section, I will detail the functional and non-functional requirements of this work

3.1.1 Functional Requirements

A functional requirement is a feature or a function that the developers of a project must implement to allow users to accomplish their tasks.

This is the list of the functional requirements of this project:

1. The player will be able to start a new game.

2. The player will be able to move the camera in four directions using the arrow keys or the left mouse button.

3. The player will be able to rotate the camera using the right mouse button or with "Q" to rotate to the right and "E" to rotate to the left.

4. The player will be able to zoom in and out using the mouse wheel.

5. The player will be able to build a hospital placing the rooms where and how he wants.

6. The player will be able to choose which players hire if he has enough money.

7. The player will be able to check the statistics of the game.

8. The player will count on a specific amount of money and he will be able to manage the budget of the hospital.

9. The player will be able to earn money treating patients and then he will be able to waste it building rooms or hiring workers.

10. The player will be able to control the waiting times of the patients.

11. The player must maintain the hospital working while waves of patients are arriving.

12. The patients will arrive at the hospital and then lead to the reception. There they will be assigned to a consult if there is any free. If the patient can not find an empty consult he will go to the waiting room.

3.1.2 Non-functional Requirements

A non-functional requirement describes how the system must behave and establish the limits of its functionality.

This is the list of the functional requirements of this project:

1. A new player without experience will be able to learn to play the game in less than 15 minutes.

2. All the items present in the game will be low poly.

3. The UI will be simple and clean.

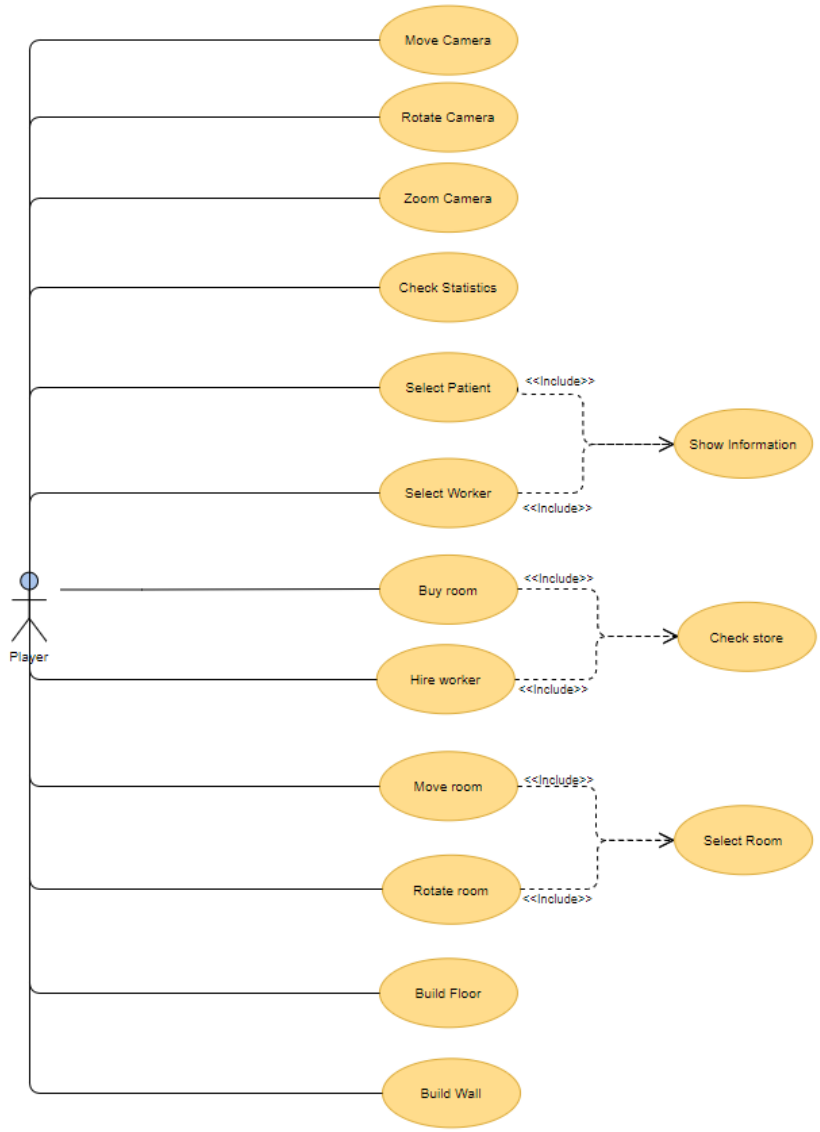

FIGURE 3.1: Use Case diagram

Use case ID: UC01
Name: Move the camera
Requirement: 1
Actors: Player
Description: The player moves the camera
Preconditions: 1. Not have any UI window open
Normal sequence steps: 1. Click and drag with the left mouse button 2. Press the arrow keys 3. The camera moves in the direction of the keys pressed or in the direction of the mouse
Alternative sequence steps: None

TABLE 3.1: Functional requirement «Use case 01. Move the camera»

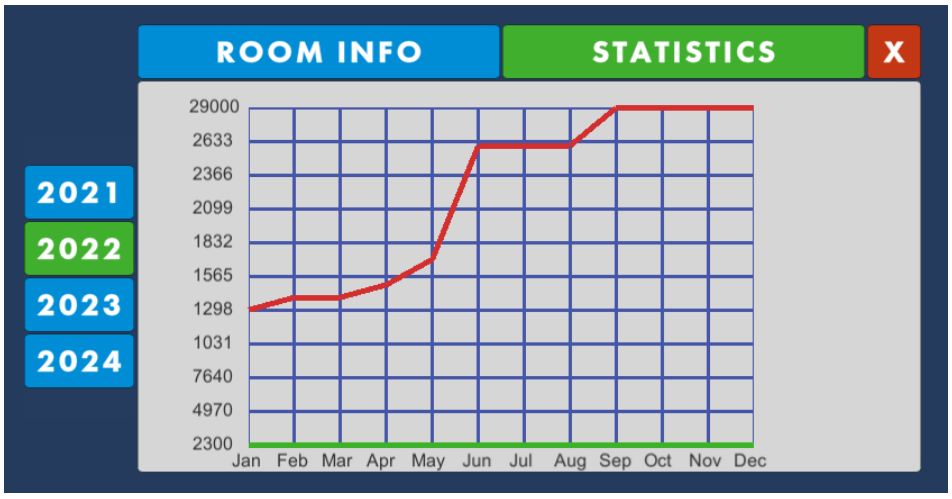

FIGURE 3.2: The statistics window in game

Use case ID: UC02
Name: Rotate the camera
Requirement: 2
Actors: Player
Description: The player rotates the camera
Preconditions: 1. Not have any UI window open
Normal sequence steps: 1. Click and drag with the right mouse button 2. Press “Q” or “E 3. The camera rotates to the left if the player drags to the left or presses "Q" or the right if the player drags to the right or presses "E"
Alternative sequence steps: None

TABLE 3.2: Functional requirement «Use case 02. Rotate the camera»

FIGURE 3.3: The tab displayed when a patient is selected

Use case ID: UC03
Name: Zoom
Requirement: 3
Actors: Player
Description: The player zooms the camera
Preconditions: 1. Not have any UI window open
Normal sequence steps: 1. Use the mouse wheel 2. If the player scrolls up the camera will zoom in, if he scrolls down the camera will zoom out.
Alternative sequence steps: None

TABLE 3.3: Functional requirement «Use case 03. Zoom »

FIGURE 3.4: The tab displayed when a worker is selected

FIGURE 3.5: The room store in game

FIGURE 3.6: The hiring tab in game

Use case ID: UC04
Name: Check statistics
Requirement: 12
Actors: Player
Description: The player opens a window where the statistics are displayed (see Figure 3.2)
Preconditions: 1. Not have any UI window open 2. Open the statistics window first
Normal sequence steps: 1. Left click on the statistics icon 2. The statistics window is opened and displayed in middle of the screen 3. The statistics icon changes it is color to green
Alternative sequence steps: None

TABLE 3.4: Functional requirement «Use case 04. Check statistics»

3.2 System Design

The next use case tables come from the following Use Case diagram (see Figure 3.1):

The following figures (see Figure 3.7) and (see Figure 3.8) represent the sequence of actions that follows a patient and the sequence of the actions described by the UC09 (see Table 3.9) respectively.

Use case ID: UC05
Name: Select patient
Requirement: 5
Actors: Player
Description: A patient is selected (see Figure 3.3)
Preconditions: 1. Not have any UI window open 2. Not have fixed another patient or worker info tab to screen
Normal sequence steps: 1. Left click directly on a patient 2. A small tab is opened int the down-right corner displaying the selected patient statistics 3. The camera is locked to the patient's position
Alternative sequence steps: None

TABLE 3.5: Functional requirement «Use case 05. Select patient»

Use case ID: UC06
Name: Select worker
Requirement: 6
Actors: Player
Description: A worker is selected (see Figure 3.4)
Preconditions: 1. Not have any UI window open 2. Not have fixed another patient or worker info tab to screen
Normal sequence steps: 1. Left click directly on a doctor 2. A small tab is opened int the down-right corner displaying the selected doctor statistics 3. The camera is locked to the doctor's position
Alternative sequence steps: None

TABLE 3.6: Functional requirement «Use case 06. Select worker»

Use case ID: UC07
Name: Open the information tab
Requirement: 7
Actors: Player
Description: Displays the information of the selected agent
Preconditions: 1. Not have any UI window open 2. Left click on a worker or on a patient 3. Not have fixed another patient or worker info tab to screen
Normal sequence steps: 1. Left click directly on a doctor or on a patient 2. The agent info tab will be displayed at the under-right corner
Alternative sequence steps: 1. If an information tab is fixed to the screen on click on a patient or on a worker it will still display the fixed tab

TABLE 3.7: Functional requirement «Use case 07. Open the information tab»

Use case ID: UC08
Name: Buy room
Requirement: 8
Actors: Player
Description: Purchase a room from the store (see Figure 3.5)
Preconditions: 1. Have opened the store 2. On the store have choose the buy rooms tab 3. Left click on the room you want to buy 4. Have enough money to buy the room
Normal sequence steps: 1. Open shop 2. The shop window will be displayed on the screen 3. Select room tab from the shop window 4. The room tab will change it is color to green and the room shop will be displayed 5. Purchase the room 6. The shop window will close 7. The room price will be subtracted from the player's money 8. The room will be ready to be placed on the map
Alternative sequence steps: None 1. If you do not have enough money to purchase the room an alert will be displayed and you will still be in the shop with the buy room tab opened.

TABLE 3.8: Functional requirement «Use case 08. Buy room»

Use case ID: UC09
Name: Hire worker
Requirement: 9
Actors: Player
Description: Hire a worker from the store (see Figure 3.6)
Preconditions: 1. Have opened the store 2. On the store have choosed the hire workers tab 3. Left click on the worker you want to hire 4. Have enough money to hire the worker
Normal sequence steps: 1. Open shop 2. The shop window will be displayed on the screen 3. Select the hiring tab from the shop window 4. The hire tab will change it is color to green and the worker shop will be displayed 5. Hire a worker 6. The shop window will close 7. The worker salary will be subtracted from the player's money 8. The worker will enter the hospital and start to work
Alternative sequence steps: None 1. If you do not have enough money to hire the worker an alert will be displayed and you will still be in the shop with the buy hire tab opened

TABLE 3.9: Functional requirement «Use case 09. Hire worker»

Use case ID: UC10
Name: Rotate the camera
Requirement: 10
Actors: Player
Description: Opens the shop window
Preconditions: 1. Not have any UI window open 2. Left click on the shop icon
Normal sequence steps: 1. Click on the shop icon 2. The shop window will be displayed 3. The shop button will change it is color to green
Alternative sequence steps: None

TABLE 3.10: Functional requirement «Use case 10. Rotate the camera»

Use case ID: UC11
Name: Move room
Requirement: 11
Actors: Player
Description: Move a room according to the mouse position
Preconditions: 1. Have edit mode active 2. Left click on a room
Normal sequence steps: 1. Left click on a room 2. The room will be displayed on green or in red depending on if it can be placed in the mouse position 3. Move the mouse to the desired position 4. The house will move according to that position position 5. Left click again to place the room on the mouse position 6. The room will be displayed with it is normal colors
Alternative sequence steps: None 1. If the room is colliding with another object the room will change its color to red and when you left-click to place it will not be placed and an alert will be triggered

TABLE 3.11: Functional requirement «Use case 11. Move room»

Use case ID: UC12
Name: Rotate room
Requirement: 12
Actors: Player
Description: Rotates a building +90 degrees on the Z-axis
Preconditions: 1. Have edit mode active 2. Left click on a room
Normal sequence steps: 1. Left click on a room 2. Right-click to rotate +90 degrees on the Z-axis 3. The room will be displayed on green or in red depending on if it can be placed in the mouse position 4. Left click again to place the room on the mouse position 5. The room will be displayed with it is normal colors
Alternative sequence steps: None 1. If the room is colliding with another object after a rotation the room will change its color to red and when you left-click to place it will not be placed and an alert will be triggered

TABLE 3.12: Functional requirement «Use case 12. Rotate room»

Use case ID: UC13
Name: Select room
Requirement: 13
Actors: Player
Description: Selects a room
Preconditions: 1. Have edit mode active 2. Left click on a room
Normal sequence steps: 1. Left click on a room to select it 2. The room will switch it is color to green
Alternative sequence steps: None

TABLE 3.13: Functional requirement «Use case 13. Select room»

Use case ID: UC14
Name: Build wall
Requirement: 14
Actors: Player
Description: Builds the walls of a room using the mouse input
Preconditions: 1. Activate the building mode 2. Select the building wall mode 3. Left click to select the start point 4. Left click to select the end point and build the wall from the start point to the end point
Normal sequence steps: 1. Left click to place the start point 2. A column will be instantiated in the start point 3. other columns will be instantiated in the path that the mouse follows 4. Left click again to place an endpoint and build a wall between these two points in a straight line 5. A wall will be built from the start to the endpoint
Alternative sequence steps: None 1. If the wall collides with another its color will change to red and if you left-click the second time to build the wall it will not be built and an alert will be triggered

TABLE 3.14: Functional requirement «Use case 14. Build wall»

Use case ID: UC15
Name: Build floor
Requirement: 15
Actors: Player
Description: Builds the floor of a room using the mouse input
Preconditions: 1. Activate the building mode 2. Select the building floor mode 3. Left click to select the start grid cell 4. Left click to select the end cell and build the floor in the area between the start cell and the end cell
Normal sequence steps: 1. Left click to place a start cell 2. A floor panel will be instantiated at the start cell 3. Floor panels will be instated filling the area between the start and the endpoint 4. Left click again to place an end cell and build the floor under the area between these two cells
Alternative sequence steps: None 1. If the floor collides with floor its color will change to red and if you left-click the second time to build the floor it will not be built and an alert will be triggered

TABLE 3.15: Functional requirement «Use case 15. Build floor»

The sequence of actions of a patient going to work described in (see Figure 3.7) can also be described using an activity diagram (see Figure 3.9).

The action described by the UC09 (see Table 3.9) and the sequence diagram (see Figure 3.8) can also be described using an activity diagram (see Figure 3.10).

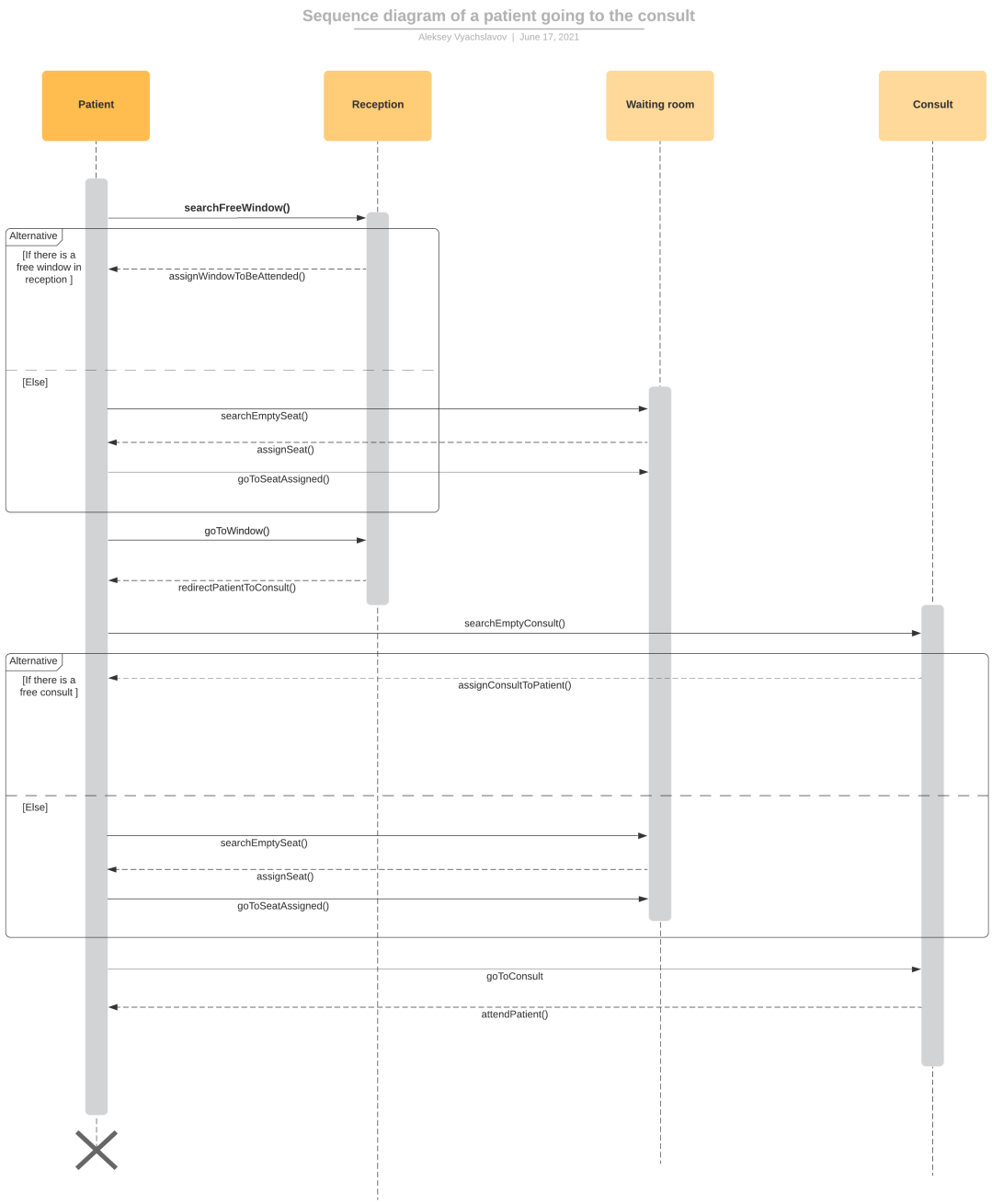

FIGURE 3.7: Sequence diagram of a patient going to the consult

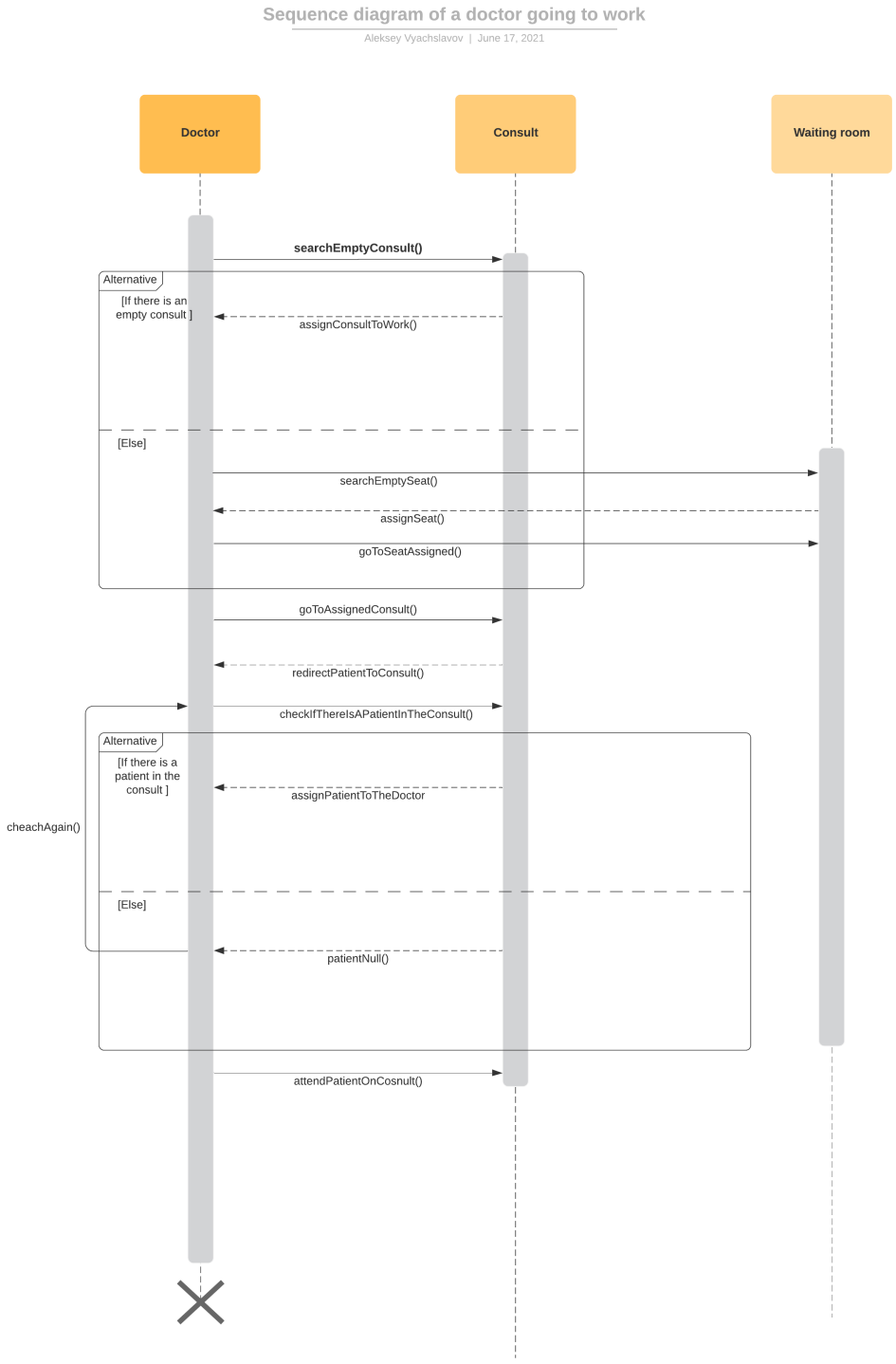

FIGURE 3.8: Sequence diagram of a doctor going to work

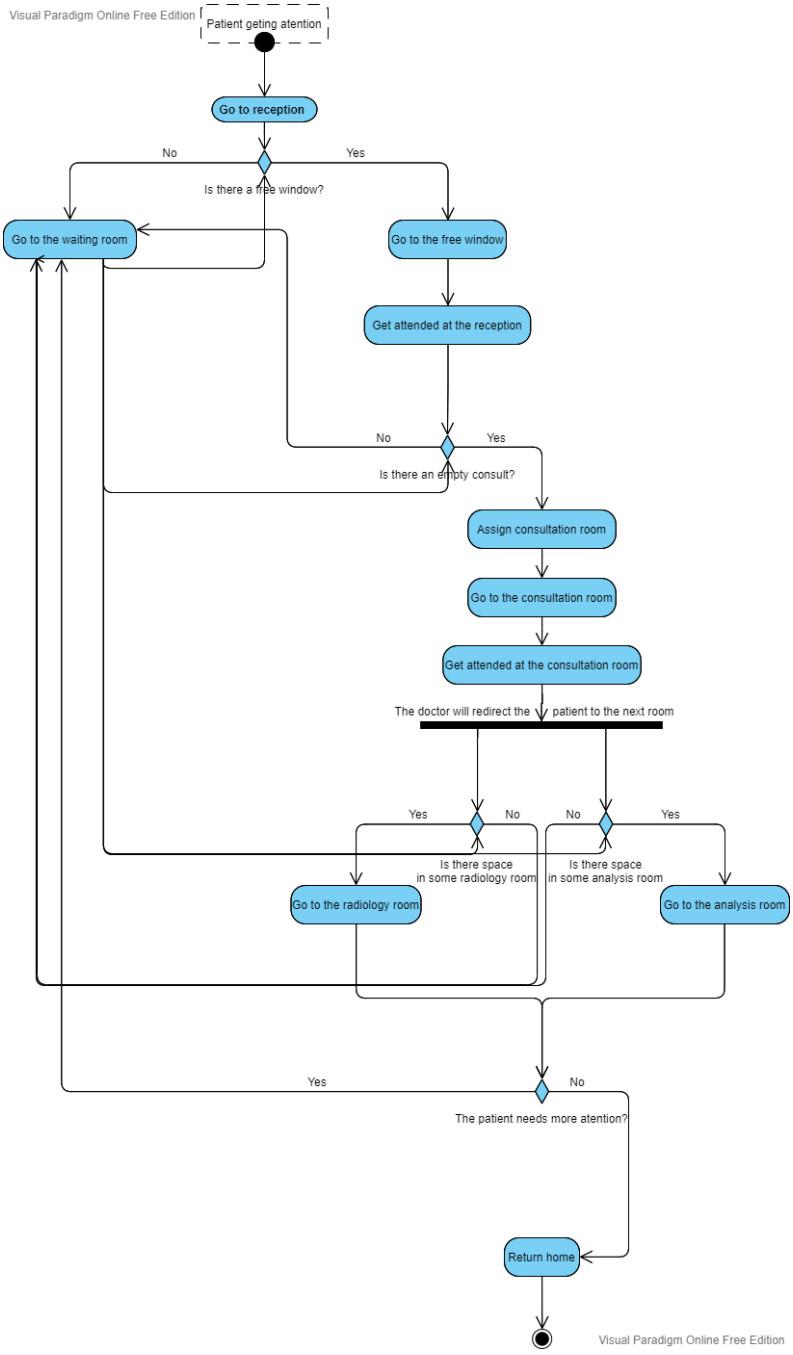

FIGURE 3.9: Activity diagram of a patient going to the consult

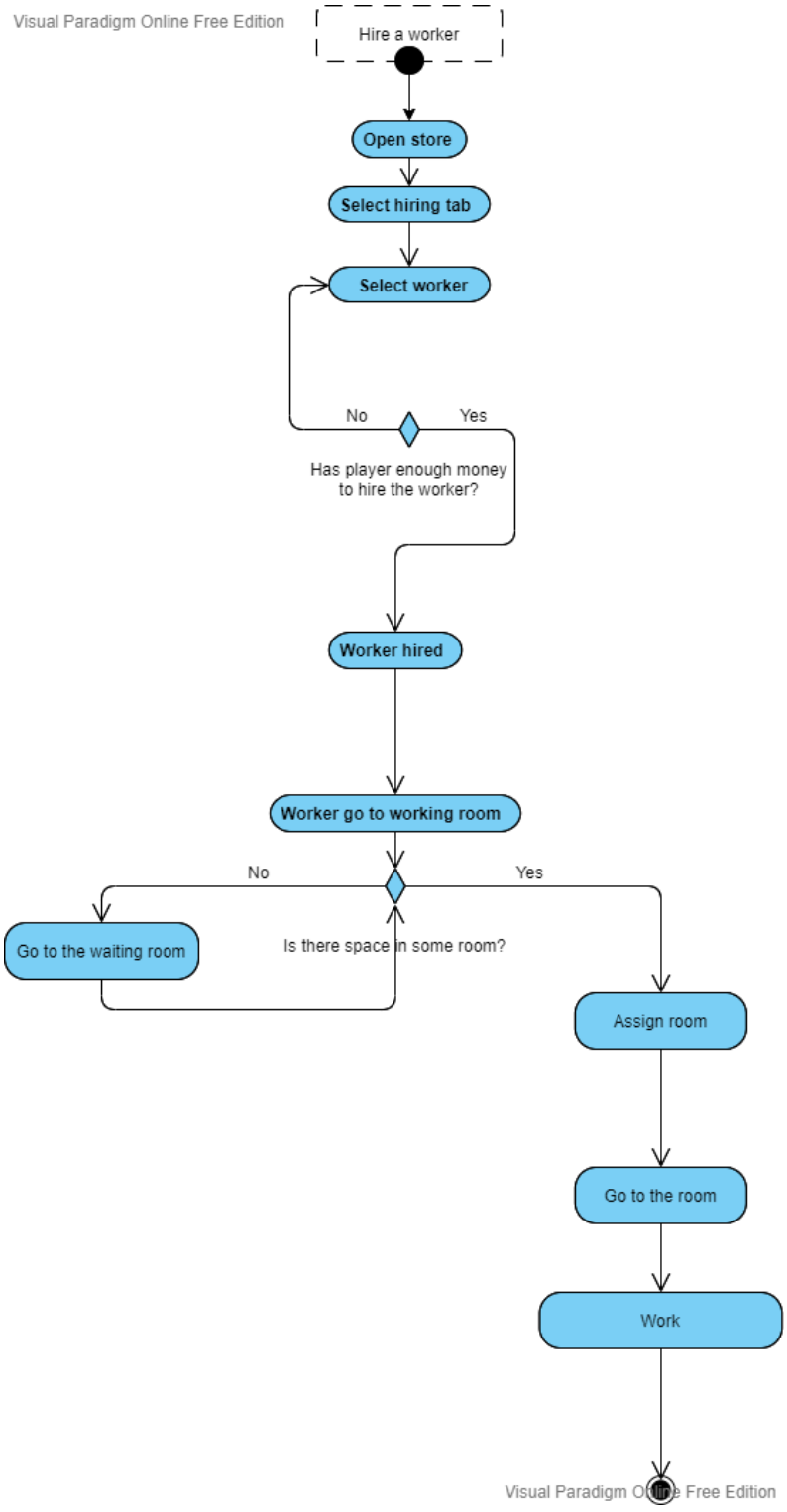

FIGURE 3.10: Activity diagram of a doctor going to work

3.3 System Architecture

The requirements to play this game will be very basic and are the following:

- CPU: Pentium 4 processor (3.0 GHz, or better)
- CPU SPEED: 3.0 GHz
- RAM: 1 GB
- OS: Windows 7/Vista/XP/ Windows 10
- VIDEO CARD: DirectX 9 level Graphics Card
- PIXEL SHADER: 2.0
- SOUND CARD: Yes
- FREE DISK SPACE: 1 GB
- DEDICATED VIDEO RAM: 1 GB

3.4 Interface Design

In this section, I will show the interface mock-ups that I developed during the project.

To keep consistency between all the components that integrate the interface I created a color palette to restrain the colors that the interface elements can have.

The interface of the game was developed with two main objectives in mind:

- Maintain the interface clear
- Provide the player access to a big amount of information on demand

All this extra information will be displayed in tabs and windows to keep the interface clean and do not overpopulate it with tons of information.

In the main state of the interface we can see that all the buttons are in the downside panel so the player screen is pretty clean. This panel provides the player a list of buttons that on interaction with will open extra windows. This will provide the player with information and will allow him to perform some actions. Some actions could be: opening the store (see Figure 3.11) or checking the game statistics (see Figure 3.12).

Moreover, some tabs can be opened on demand of the player like the patient state tab (see Figure 3.13) or the worker state tab (see Figure 3.14). These tabs will display

extra information to the player but still allowing him to keep playing.

All the icons used in the UI are open-license and have been downloaded from Streamline [6].

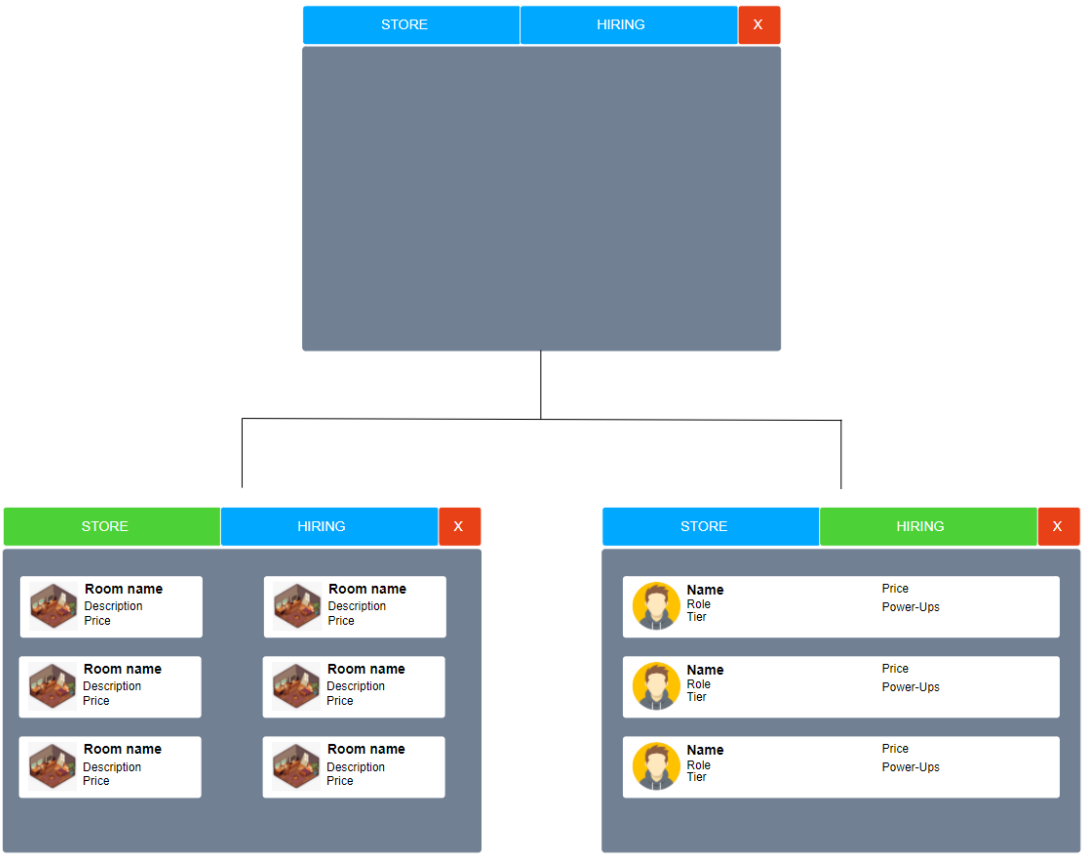

FIGURE 3.11: Mock-Up of the shop interface

FIGURE 3.12: Mock-Up of the statistics interface

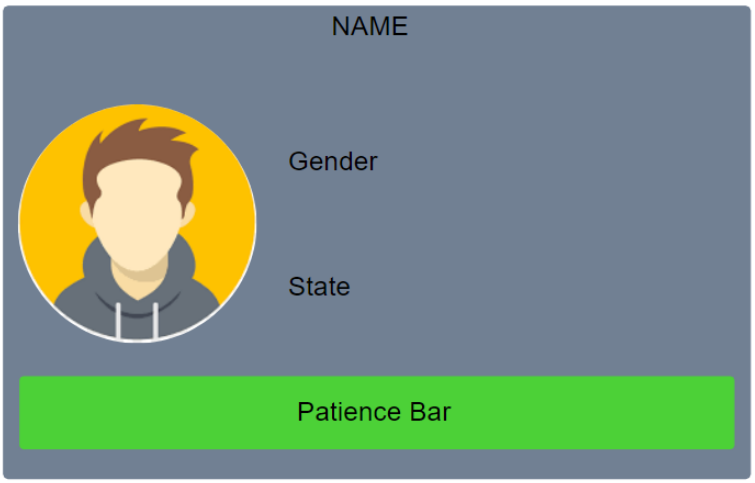

FIGURE 3.13: Mock-Up of the patient information tab

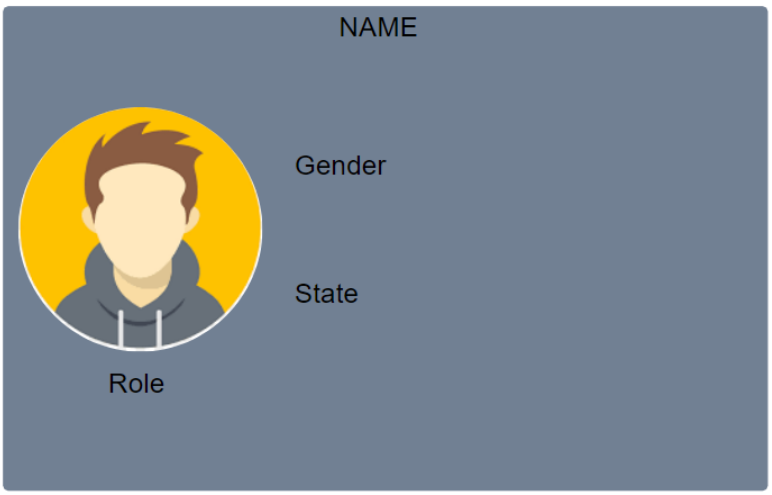

FIGURE 3.14: Mock-Up of the worker information tab

CHAPTER 4

Work Methodology, Work Development, and Results

Contents

In this chapter, I will resume the job done during this project, describe the workflow that I followed and explain with examples how works the most important mechanics of the game developed.

4.1 Work methodology

First of all, as mentioned in the section 2.1 I divided the work of this project into 3 sprints using an agile methodology [6].

The workflow followed in this methodology can be found in this figure (see Figure 4.1).

Before each sprint, I planned the mechanics that are going to be developed during the sprint. This planning was made using a tool called Milanote [7]. This tool provides the user an environment to organize the work using visual boards (see Figure 4.2).

After this planning, I started to work on the tasks that I planned for this sprint. The tasks were split into the important mechanics and add-on mechanics.

The important mechanics are the ones that their lack in the project will result in an unplayable game or an impoverished version of the game. The add-on mechanics are mechanics that do not affect drastically the gameplay. They improve some aspects of the gameplay but if they are missing the overall of the project will not be affected.

This distinction was made because every sprint has a deadline. First are implemented the important mechanics and then the add-ons. This is made to prioritize the important parts of the project. If after the deadline of a sprint all the important mechanics were completed the testing started even if the add-ons were not finished. If some important mechanics were not completed the deadline was delayed. This delay was big enough to ensure the completion of the unfinished mechanic or mechanics.

After the deadline had arrived and all the important mechanics of the sprint were completed the playtest started.

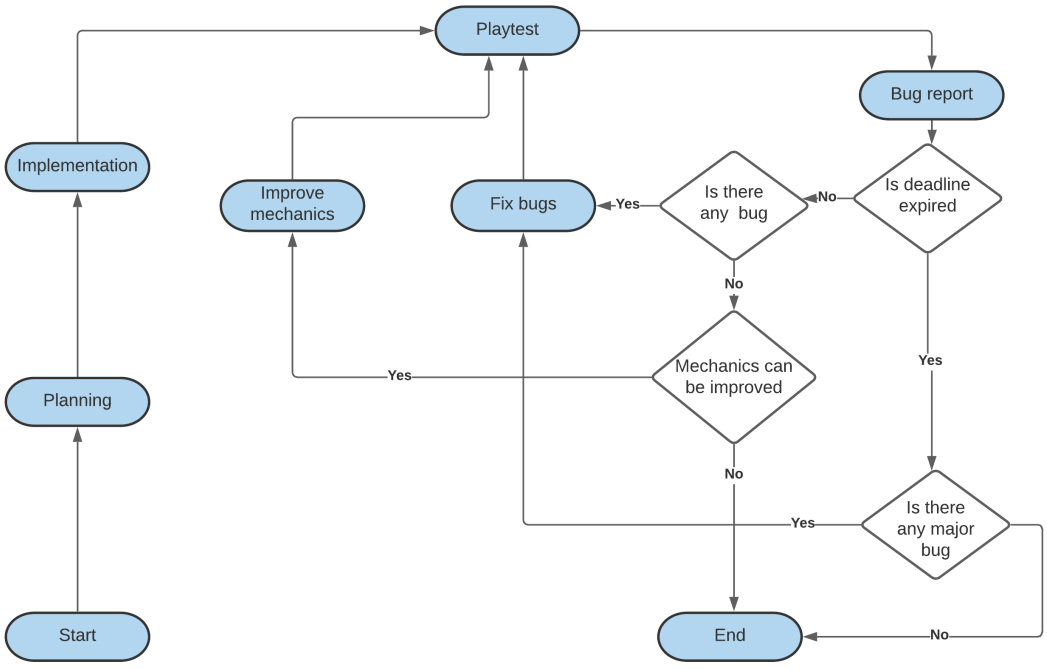

FIGURE 4.1: Diagram of the workflow of a Sprint

During the playtest I tested every mechanic implemented this sprint in interaction with the rest of the mechanics. If I found a bug or a glitch I wrote it down on a specific board in Milanote (see Figure 4.3).

When the playtests ended I tried to fix all the bugs that were found and improved the work of the mechanics. After the fixes and improvements, I playtested the game again and wrote down the bugs, this loop continued for a maximum of 2 weeks or until I fix all bugs. During the fixing, I focused on the major bugs. If after the 2 weeks there were bugs that caused major problems I delayed the beginning of the next sprint until the biggest bugs were fixed.

4.2 Work Development

This section will work as a resume in chronological order of the implementation of the most important mechanics of the game.

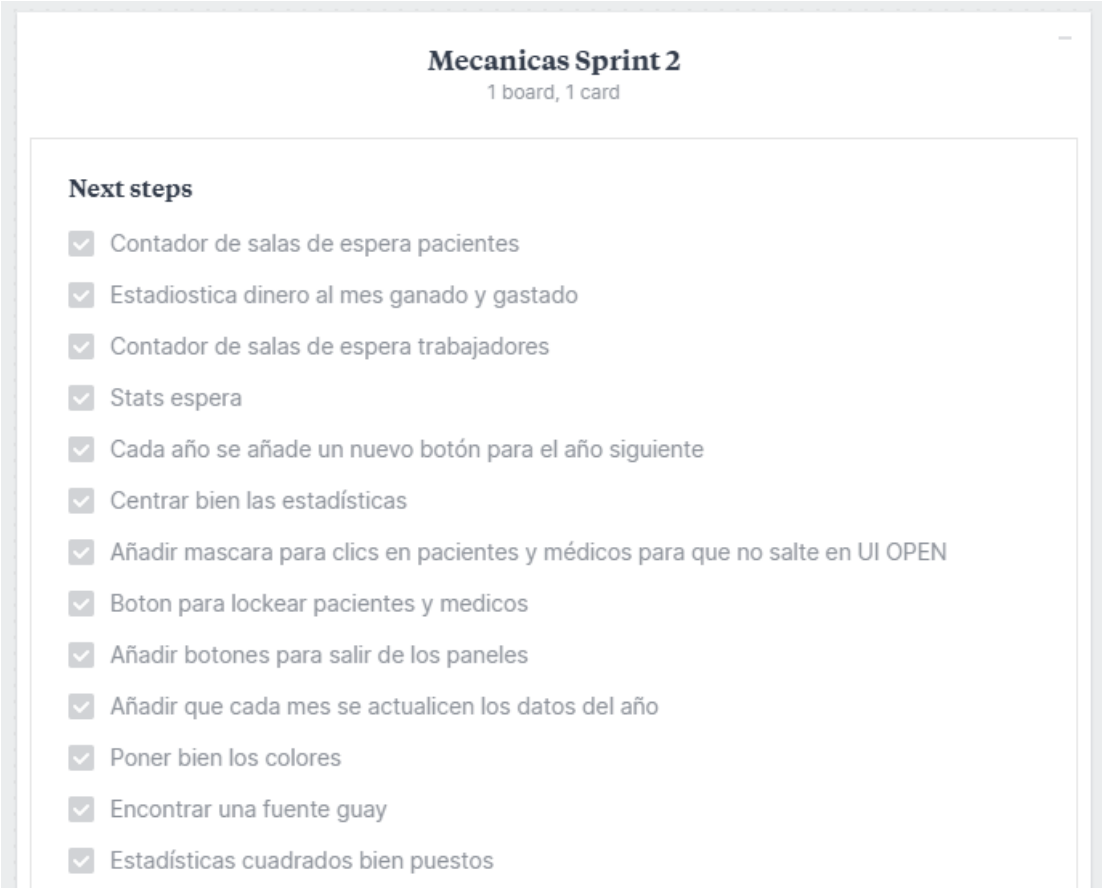

FIGURE 4.2: Sample of a board used for organize the tasks left to do

First of all, I created a placeholder space to simulate where the game is going to be played. Then I started the implementation of the first mechanic.

4.2.1 Building System

The core of a Tycoon game of building hospitals for sure is going to be the building mechanic. To develop this mechanic I first needed to implement some system to help the player place the rooms and manage the environment.

I decided that this system is going to be a grid system where space will be split into cells. In the grid system, every room will occupy a determined number of cells. The room's position will be snapped to a cell avoiding rooms be placed in a middle of a cell. This mechanic aims to help the player place the rooms without worrying on connect them exactly or overlap two rooms. Also to help the players place rooms the grid will be displayed on the floor (see Figure 4.4).

For the design of the building system, I aimed on designing a mechanic that is functional but more importantly easy to use and responsive. This is the design pattern that will follow over the whole project. In a game with so many mechanics, it is very important to make them intuitive. This prevents the player from having to learn how to interact with every single mechanic.

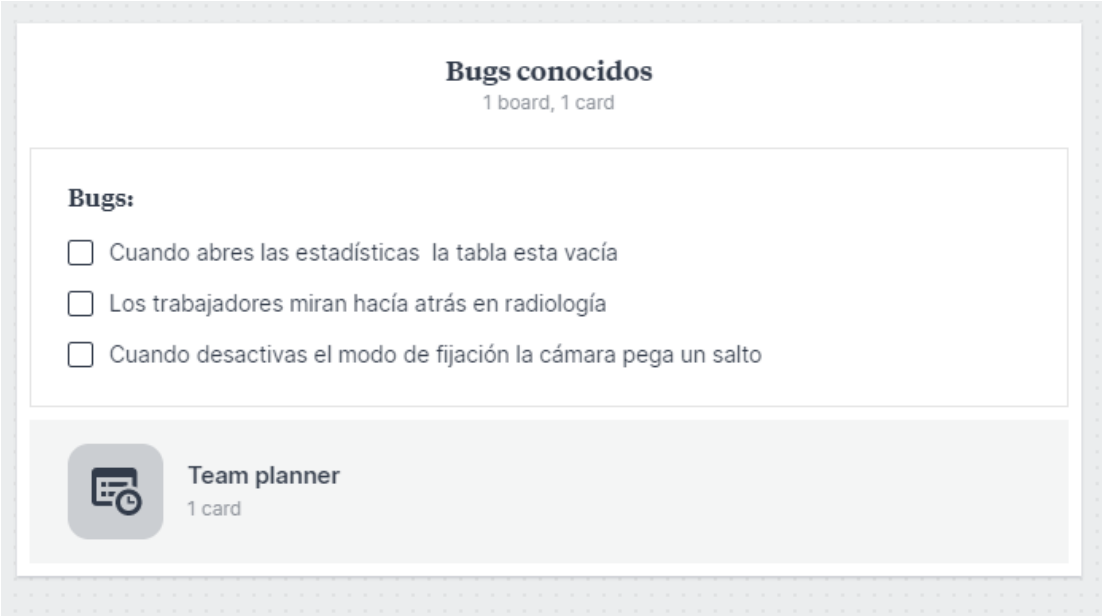

FIGURE 4.3: Sample of a board used for listing the known bugs

In the case of the building system, when the player buys a room it will appear on the map and will follow the player's mouse. When the player clicks if the room is placed correctly it will be built in the room's current position. To communicate to the player if he can build a room in the current position the room will be drawn in green (see Figure 4.5). If building is not allowed it will be drawn in red (see Figure 4.6). Also if the player tries to build in a not allowed place an error message will be shown in the console (see Figure 4.17). I will explain this mechanic further in this section.

In addition to the building system, players can modify the hospital layout if edit

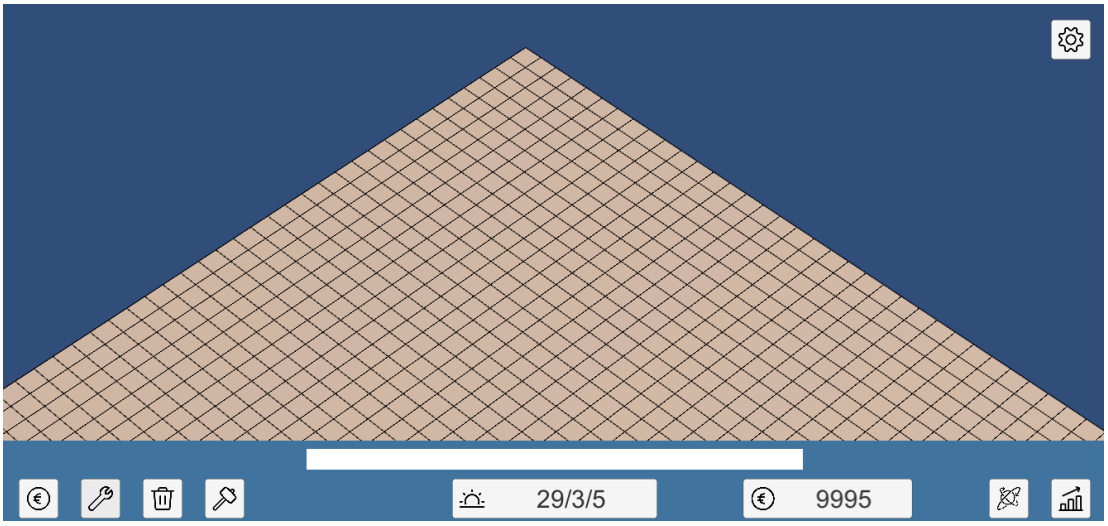

FIGURE 4.4: Image of the grid

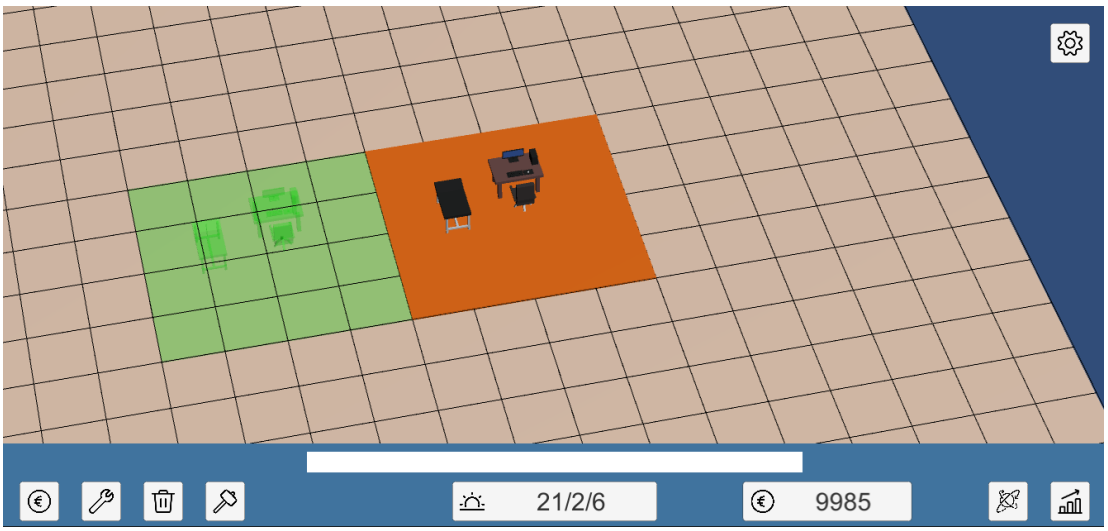

FIGURE 4.5: Image of a room that can be built

mode is activated. Edit mode allows the player to select a room and move it or/and rotate it. During the edition of a room color legend to communicate to the player if the room is placeable or not is the same.

4.2.2 Camera Controller

The next important functionality I added was the camera control script. The camera in a Tycoon game is a key aspect because, during the gameplay, the player will need to constantly navigate through the map. This navigation must be comfortable and easy. While controlling the camera the player will be able to move around, rotate and zoom. Speaking about the controls, every player has a different taste on how to control a game. Because of that, we allowed the player to control the camera both with the mouse and the keyboard.

At this point, I did not know the size of the playing area or the height of the rooms. I do not want to choose some values for the playing area and then be restricted to that values for the rest of the project. To avoid that I built a script that allowed me to tweak all that values. This permitted me to play with the values and get the best results (see Figure 4.7).

4.2.3 Character Generator

After the implementation of the camera, I decided that it was time to start the creation of the characters. I am not the best at modeling so I decided that for the sake of the

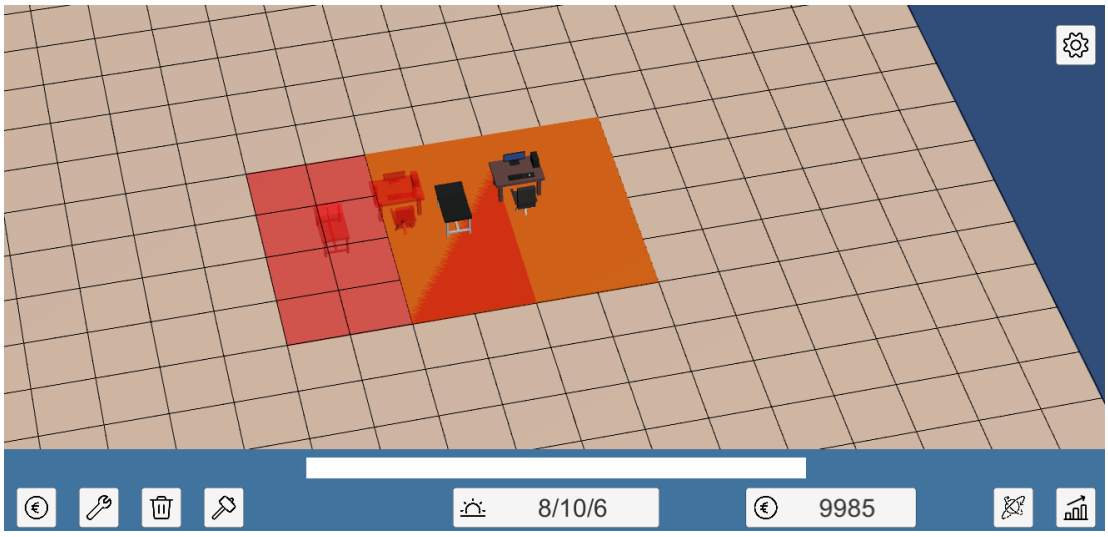

FIGURE 4.6: Image of a room that can not be built

project it was a better idea to find an asset pack instead of modeling the characters myself. I found this open license asset pack [8].

This pack comes with a lot of types of characters, but their format did not fit what I needed for my project. The reason was that the characters of the pack came all in one single mesh (see Figure 4.8).

As in this game, there will be a lot of patients it is mandatory to have a kind of system that generates them randomly. Otherwise, after a few hours, the player will notice that the same characters are appearing over and over again.

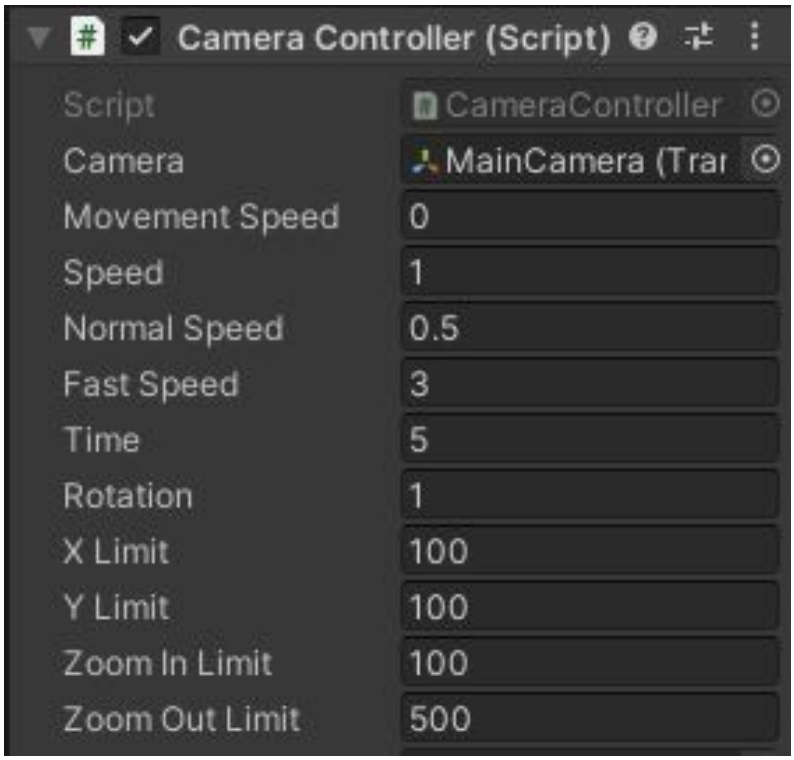

FIGURE 4.7: The camera options that can be modified

FIGURE 4.8: The original asset from the asset pack

To achieve this randomness I modified the assets of the pack. I separated all the parts of the mesh found on the characters(see Figure 4.9). The random character generator is inspired by the character creator systems that some games have (see Figure 4.10). In these systems the game has a pool of objects for every customizable part of the character and the player can choose every part and combine them. This results in a different character for every player. The players can also name their characters. In these systems, the variety comes from the size of the pools or the number of options.

In the game this character creation is done randomly, the character generator has a pool for:

- Hair models
- Hair color
- Eye color
- Skin color
- Upper-Clothes model
- Upper-Clothes color
- Down-Clothes model
- Down-Clothes color
- Names
- Surnames

Combining these parameters, every character generated will be different from the others (see Figure 4.11). Also, some combinations create special characters, there are more than 10 special characters. Find them all! (see Figure 4.12)
This character generator generates both workers and patients.

Following my attempt to populate a bit the world after the implementation of the character generator I started the creation of the rooms (see Figure 4.15). As mentioned before the assets will be low poly (see Figure 1.1). Low poly is both an art style and form of optimizing a game and gain performance.

For this game, I designed a color palette made of 64 colors (see Figure 4.14). The aim of having a palette is that all the assets of the game will have this palette as the only material.

The optimization comes when applying this unique material to a complex object. For example in a character, a material is created for each color used, brown for the hair,

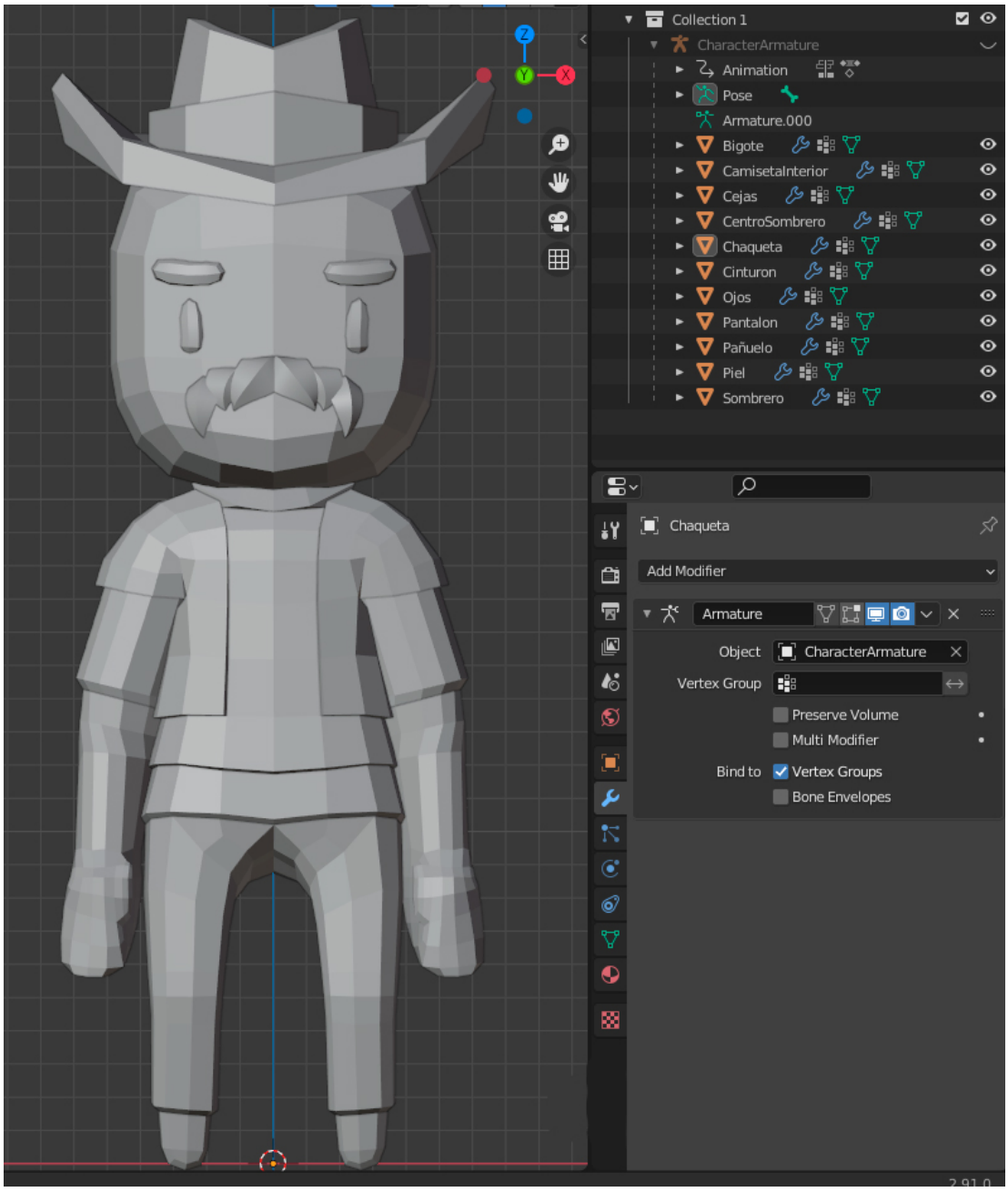

FIGURE 4.9: The modified asset from the asset pack

FIGURE 4.10: Character creator of the game Hytale

FIGURE 4.11: Two characters randomly generated

FIGURE 4.12: Special character

blue for the eyes, blue for the pants ...,. To draw this character the engine will have to search in memory every material and access it. In an object with a lot of materials, this can be very resource-consuming. Instead of that if an atlas material is used the machine only has to access one material. Using this atlas material the system will apply the color according to the coordinates of the texture. Here an explanation of how the texture coordinates work can be found (see Figure 4.13).

4.2.4 AI

Once the characters were modeled it was time to develop the IA. I split the IA into two types the basic IA and the task system.

The basic system is the one that all the medical staff will have. Workers can also have a specific role. For this project a developed a few, consultation doctors, radiologists, annalists, and receptionists. All of them follow the same basic rules. I will briefly explain them in the next paragraph but the full explanation can be found on this sequence diagram (see Figure 3.8) and in this activity diagram (see Figure 3.9).

A worker entering the hospital will search for a free space to work. In the case of the doctors an empty room of his role, in the case of the receptionist a free seat in the reception. If they do not find somewhere to work they will go to the resting room until there is a free space. If they find somewhere to work they will go there and wait until a patient comes. Once a patient came they will attend them and redirect to the next

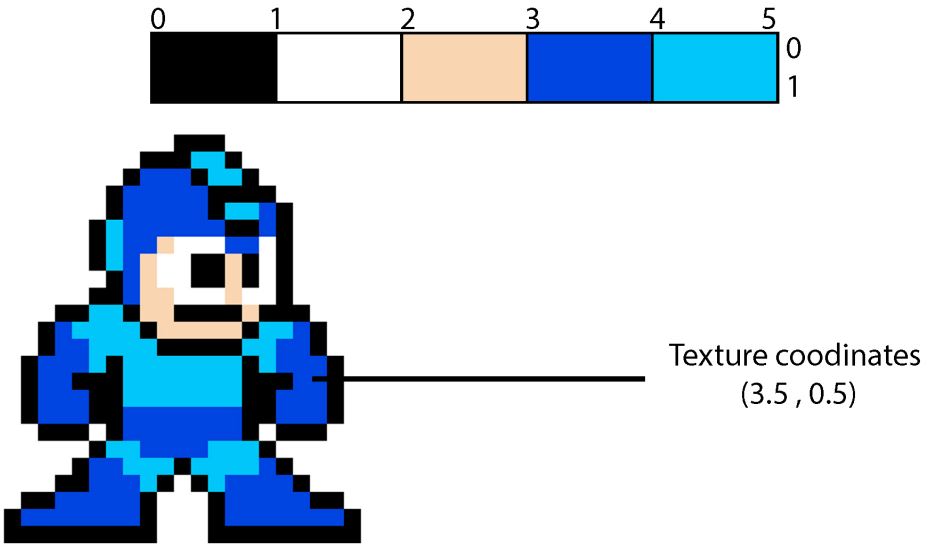

FIGURE 4.13: Explanation of how the texture coordinates work (Image of Mega-Man 8-bits)

FIGURE 4.14: This is the color palette used for all the assets in the game

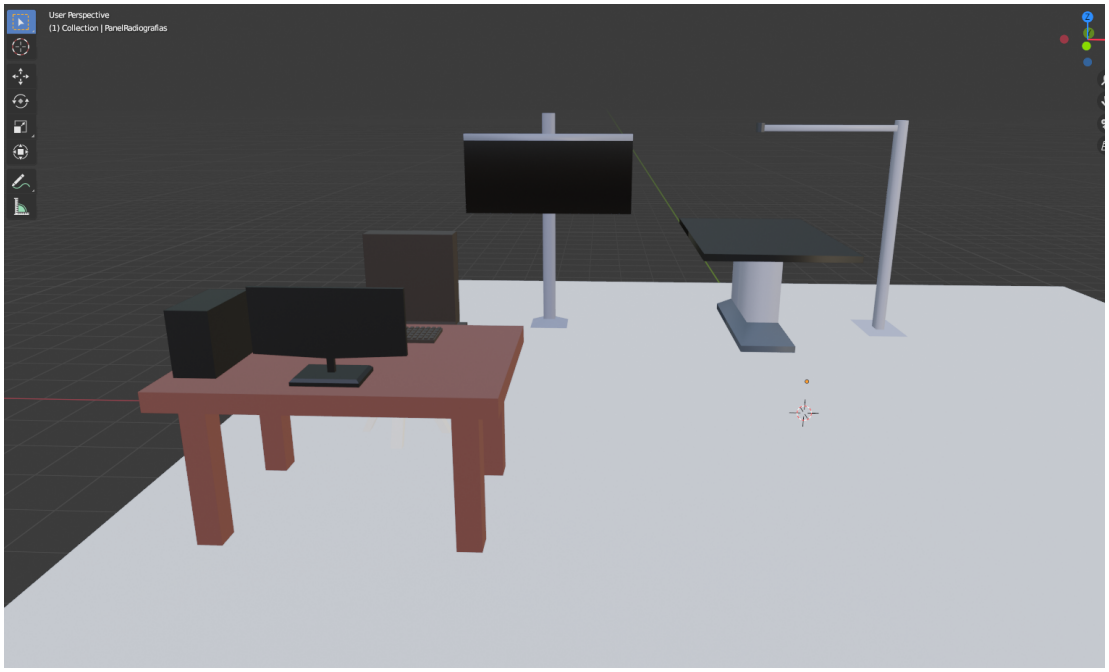

FIGURE 4.15: The modeling of the radiology room

procedure or home. If the patient has ended the treatment.

Patients follow this same method with the difference that if they do not find a space in the place they want to go they will go to the waiting room. More explanation about this can be found on this sequence diagram (see Figure 3.7) and in this activity diagram (see Figure 3.9).

The task system is used by special workers like cleaners. While there is nothing to do they will be in the resting room. The tasks are put on a queue to be completed in order of entry, once a task is on the queue a free worker will be assigned to that task. The worker will see what consists the task and will try to complete it.

All the workers will need pathfinding to be able to traverse the hospital. Implementing pathfinding was a difficult part of the project where I invested quite a time thinking about the better way to do it.

To implement pathfinding I thought of two ways. One way was using the grid system and assigning to each cell a node and then simply use the A* algorithm [9] to find the shortest path between two nodes [11]. The other using the Unity navigable mesh system [12].

Using navigable mesh the mesh must be baked in the editor mode to after be used during the gameplay. Using pathfinding at the beginning of the script a graph made of nodes must be built[10]. The problem of this is that they require baking the mesh or creating the graph both of these operations have a heavy impact on the performance. But there is another problem, the machine can not know how every hospital will look. This means the navigable mesh or the graph should be recalculated during gameplay. This can cause a big slowdown in the player's computer what it is inadmissible for a good playing experience.

The way I resolved this is using Unity navigable mesh and the NavMesh Component repository [12] shared on GitHub [13].

This repository provides a script that generates a volume of the desired size. Inside this volume, the navigable mesh can be modified on run-time. Instead of rebuilding the entire navigable mesh, this script updates only the nodes that have been affected by the modification (see Figure 4.16).

4.2.5 UI Design and Implementation

After the implementation of the AI, I focused on the design of the UI. The UI aims to be simple but still allow the player to open tabs and windows to display more information when needed. Tycoon games are hard to manage and involve a lot of mechanics. This makes that sometimes it is hard to know exactly what is going on, to overcome this I

implemented the console. The console is a text box where important information will be displayed. This is made to inform the player when something has happened. It is placed in the middle of the HUD to catch the player's focus. Also, when the message to display is an error message the text box will flash in red to ensure the player watch it (see Figure 4.17).

4.2.6 Statistics

Another mechanic related to the UI worth mentioning here is the statistics graph. Statistics are very important in the Tycoon genre. It is very important to allow the player access to as much data as it is possible to help him manage the hospital and make important decisions.

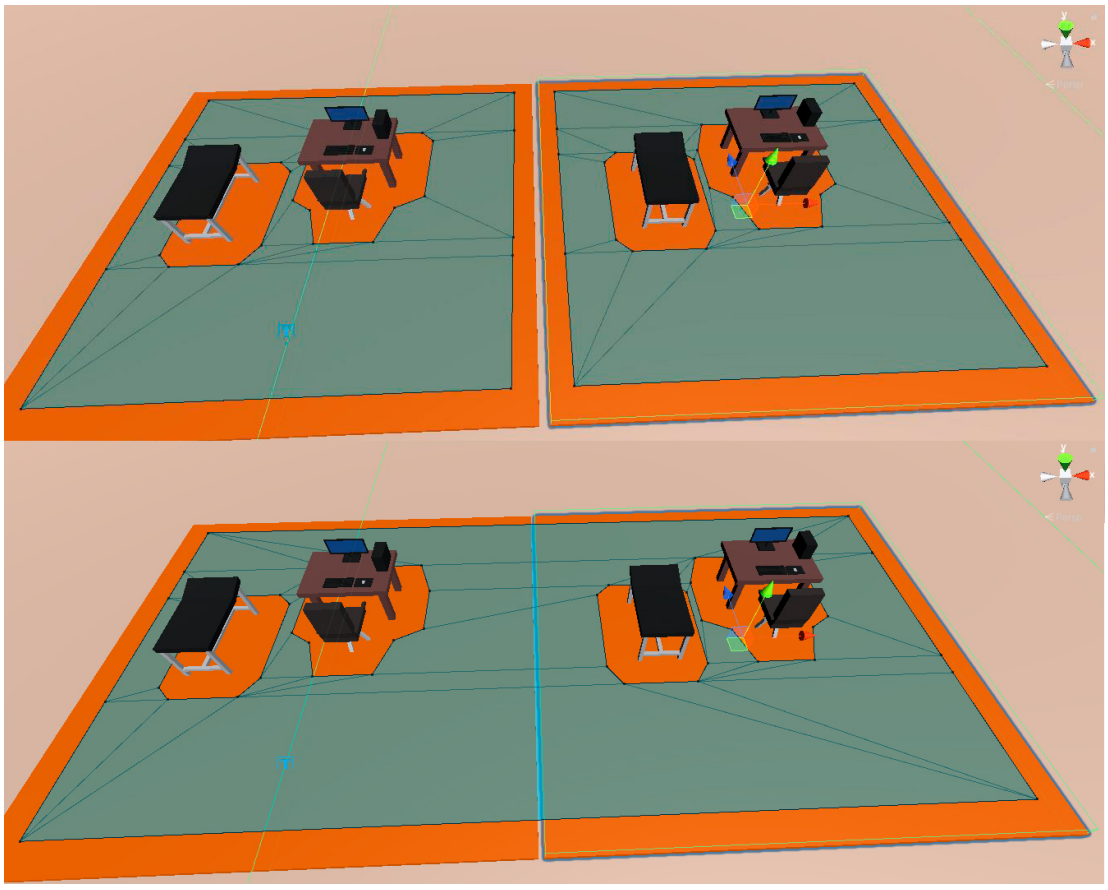

FIGURE 4.16: Two navigable meshes generated on run time and merged together

4.2.7 The edit mode

The statistics graph has two variables, the number of incomes (green) and the number of expenses (red). Every graph represents the statistics of a year and every step on the graph represents a month. The player will have access to all statistics starting from the year the game started and can switch between them to display them. The most complex part of this implementation was showing the information correctly. Values between months and between years differ a lot. To solve this I developed a system that ensures that the displaying information is scaled properly. The values will go always from the minimum value to the maximum and the steps in the Y-axis will be adjusted based on that. Due to this, the player can switch between all years without problem because the graph will be recalculated and displayed correctly on the screen.

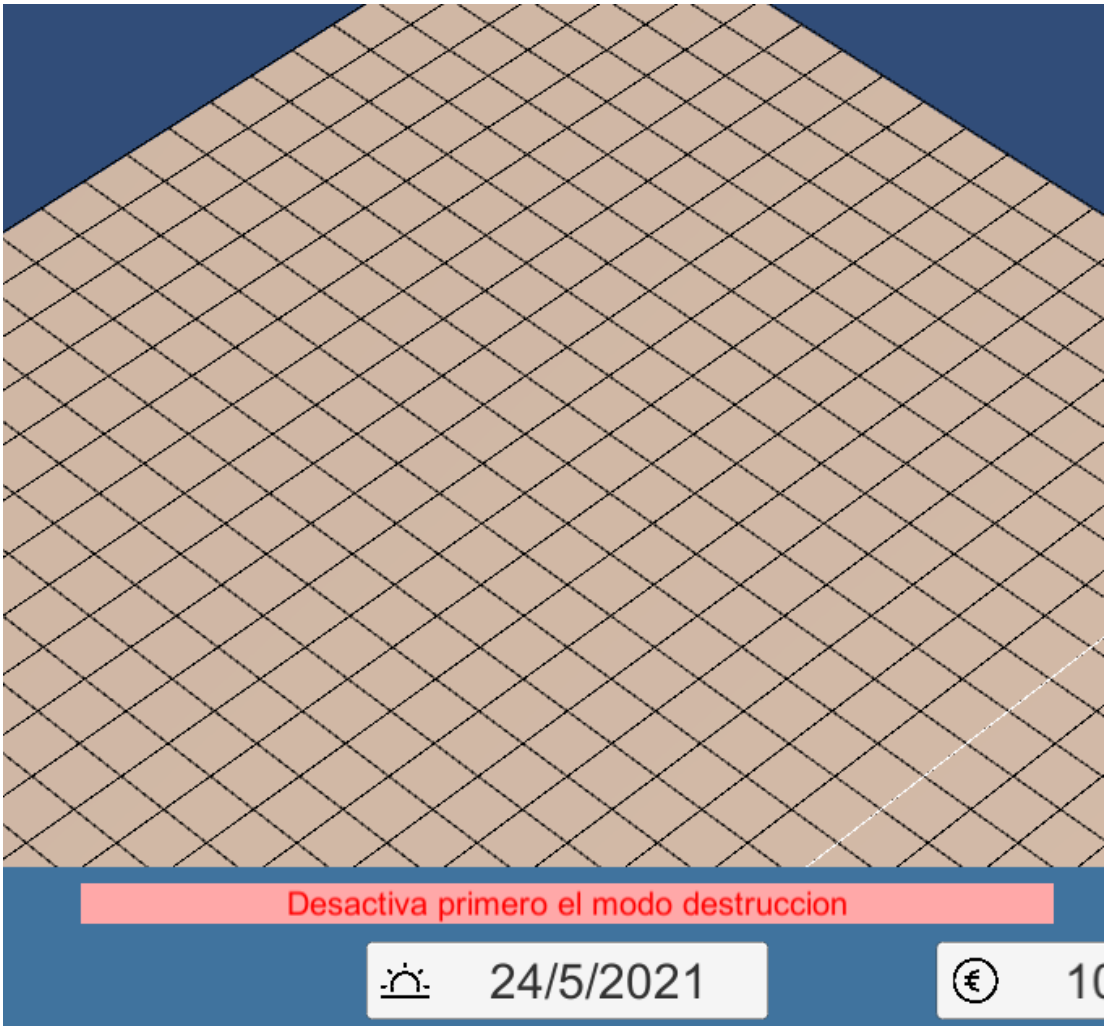

FIGURE 4.17: Image of an error message

The last important mechanic to the UI is the Edit Mode, the edit mode has two functionalities. On one hand, it enables the selection of the rooms to then move or/and rotate them (see Table 3.11) and (see Table 3.12). On the other, it shows the information about all the rooms. It informs visually the type and the state of the room so the player is always informed on what is going on. It warns the player which rooms are not usable marking them on red and displaying a text explaining why are they unusable (see Figure 4.17).

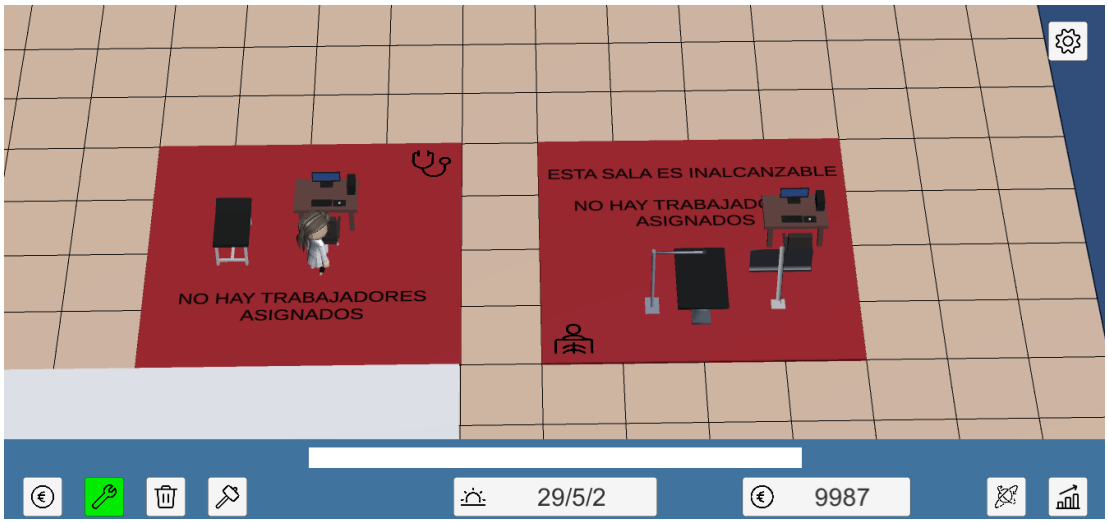

FIGURE 4.18: Left: Usable consultation room
Right: Unusable radiology room

4.3 Results

Speaking about the results, at this point the game is playable. It is not enjoyable and it has not the content needed to be a game that can be found in the market but it can be played. It presents perfectly the main mechanics of the game that will result if I had more time to end this project.

All the chore mechanics are implemented. The game is in a state where if a player could try the game for a few minutes he will be able to understand how the game loop works and what is the direction that the project has.

On one hand, this game is not ready to be commercialized. It is normal because tycoons games require a lot of mechanics, systems interconnected, and a lot of assets and options to keep the player engaged in the game.

But on the other hand, the game is at a state where it could be published or uploaded as early access. This means that the game is playable but it is still under development. The early-access works as a form to show what your game has to offer to the world and to start building a community.

In conclusion the game is not finished but I think the objectives of this project are reached. Tycoon games are huge in content and mechanics. I have accomplished the main aim of the project that was to make a playable version of a Tycoon game and I am proud of the results.

CHAPTER 5

Conclusions and Future Work

Contents

5.1 Conclusions

This project was a good challenge to put to test all the knowledge that I acquired during this degree. I have used the skills I acquired in almost every subject I have taken in this degree, I have:

- Modeled 3D assets (VJ1216 - 3D DESIGN)
- Used a game engine to develop my game (VJ1227 - GAME ENGINES)
- Programmed on C Sharp, an object-based programming language (VJ1203 - PROGRAMMING I and VJ1208 - PROGRAMMING II)
- Rendered custom meshes using GLSL (VJ1221 - COMPUTER GRAPHICS)
- Used some data structures to improve performance (VJ1215 - ALGORITHMS AND DATA STRUCTURES)
- Implemented a basic AI to control agents in the game (VJ1231 - ARTIFICIAL INTELLIGENCE)
- Used diagrams and other types of documentation to organize my work (VJ1224 - SOFTWARE ENGINEERING)

These are only a few examples of all the knowledge and skills acquired in this degree that I put into practice to develop this project.

To conclude I think that developing this project I have learned a lot because is the first project I faced alone and it is the biggest project that I took part in and handling it has put me in a real challenge and gave me a really good experience in this type of projects.

5.2 Future work

I am very proud of this project and I think that I prepared the ground for a big project that is powerfull and I will work on it until the due of the project and after that, I will continue to work on it. My plan with this game is to develop something I will be proud of publishing and publish it as my first serious game.

APPENDIX A

Other considerations

A.1 First section

During the development of this project I watched a series of videos to help me get ideas or concepts to develop the different mechanics.

Here is the reproduction list of all of these videos:

https://www.youtube.com/playlist?list=PLfnR6EzV3q3LWpeyeQSHjaMTRjB7Bxc-P

Bibliography

[1] Covid-19: 50 percent of the nurses are at risk of suffering from mental disorders,
https://www.redaccionmedica.com/secciones/enfermeria/covid-19-50-enfermeras-tiene-riesgo-sufrir-trastorno-mental-1767

[2] Eight out of 10 nurses report the lack of medical staff during the pandemic,
https://www.actasanitaria.com/enfermeras-denuncian-falta-personal-pandemia/

[3] We have not cared for those who care for us: more than 17,000 health workers died from Covid,
https://www.redaccionmedica.com/secciones/enfermeria/covid-19-50-enfermeras-tiene-riesgo-sufrir-trastorno-mental-1767

[4] Sueldos para Game Programmer,
https://www.glassdoor.es/Sueldos/espa~na-game-programmer-sueldo-SRCH_IL.0,6_IN219_KO7,22.htm

[5] Streamline,
https://app.streamlinehq.com/icons

[6] Building an Agile Process Flow: A Comprehensive Guide,
https://kanbanize.com/agile/project-management/workflow

[7] Milanote,
https://milanote.com/

[8] Ultimate Animated Character Pack,
https://quaternius.com/packs/ultimatedanimatedcharacter.html

[9] A* search algorithm,
https://www.geeksforgeeks.org/a-search-algorithm/

[10] Graph Data Structure,
https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

[11] Node,
https://en.wikipedia.org/wiki/Node_(computer_science)

[12] Unity Navigable Mesh,
https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html

[13] Git Hub,
https://github.com/

[14] Unity-Technologies/NavMeshComponents,
https://github.com/Unity-Technologies/NavMeshComponents

APPENDIX B

Source code

In the following pages you can find fragments of my code, the length of the full code it is very long to be write here so I only wrote the most important functions of my project, the full source code can be found in this repository:

https://github.com/al375729/Hospital-Tycoon

Grid Display

Listing B.1: Grid Display

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

public class GridDisplay : MonoBehaviour
{
    // Start is called before the first frame update

    public Test test;
    private Grid grid;
    private int[,] cuadricula;
    private TextMesh[,] gridTextMesh;
    private int filas;
    private int columnas;
    public Material material;

    private Vector3 Origin;
    private Vector3 Diference;
    void Start()
    {
        grid = test.getGrid();
        cuadricula = test.getCuadricula();
        gridTextMesh = test.getTextMesh();
        filas = test.getFilas();

        columnas = test.getColumnas();
    }

    private bool IsMouseOverUI()
    {
        return EventSystem.current.IsPointerOverGameObject();
    }

    // Update is called once per frame
    void Update()
    {

    }

```

```
41      private void OnPostRender()
42      {
43          for (int i = 0; i < cuadricula.GetLength(0); i++)
44          {
45
46              for (int j = 0; j < cuadricula.GetLength(1); j++)
47              {
48
49                  DrawLine(grid.GetWorldPosition(i, j) -
50                  new Vector3(filas * 2.5f, 0, columnas
51                  * 2.5f) , grid.GetWorldPosition(i, j + 1) -
52                  new Vector3(filas * 2.5f, 0,
53                  columnas * 2.5f));
54
55                  DrawLine(grid.GetWorldPosition(i, j)
56                  - new Vector3
57                  (filas * 2.5f, 0, columnas
58                  * 2.5f),
59                  grid.GetWorldPosition(i + 1, j) -
60                  new Vector3(filas * 2.5f, 0,
61                  columnas * 2.5f));
62
63              }
64          }
65          DrawLine(grid.GetWorldPosition(0, columnas) -
66          new Vector3(filas * 2.5f, 0, columnas *
67          2.5f), grid.GetWorldPosition(filas, columnas)
68          - new Vector3(filas * 2.5f, 0, columnas
69          * 2.5f));
70
71          DrawLine(grid.GetWorldPosition(filas, 0)
72          - new Vector3(filas * 2.5f, 0, columnas *
73          2.5f),
74          grid.GetWorldPosition(filas, columnas)
75          - new Vector3(filas * 2.5f, 0, columnas
76          * 2.5f));
77      }
78
79
80      void DrawLine(Vector3 inicio, Vector3 fin)
81      {
82
83          GL.Begin(GL.LINES);
```

```
84          material.SetPass(0);
85          GL.Color(Color.black);
86          GL.Vertex(inicio);
87          GL.Vertex(fin);
88
89          GL.End();
90
91      }
92
93      void LateUpdate()
94      {
95          if (Input.GetMouseButtonDown(0))
96          {
97              Origin = MousePos();
98          }
99          if (Input.GetMouseButton(0))
100         {
101             Diference = MousePos() - transform.position;
102             transform.position = Origin - Diference;
103         }
104
105     }
106     Vector3 MousePos()
107     {
108         return Camera.main.ScreenToWorldPoint(Input.mousePosition);
109     }
110
111 }
```

Drag Buildings

Listing B.2: Drag Buildings

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

public class DragBuildings : MonoBehaviour
{
    public bool placed = false;
    private float zCoord;

    public GameObject prefab;

    private Grid grid;

    public bool isSelected = true;
    bool isColliding = false;

    public Material originalMaterial;

    public Material[] materiales;

    private Quaternion objectToRotate;

    public static bool globalSelection = false;
    private Vector3 position;
    private Quaternion rotation;

    private bool lastFrameWasEditMode;

    private bool addedReferences = false;

    ConsultController consultController;
    RadiologyController radiologyController;
    AnalisisController analisisController;
    private enum State
    {
        WaitingForTask,
        DoingTask,
```

```
41          DoingTaskClean,
42      }
43      private void Start()
44      {
45          consultController = ConsultController.Instance;
46          radiologyController = RadiologyController.Instance;
47          analisisController = AnalisisController.Instance;
48
49          transform.GetChild(transform.childCount -
50          2).gameObject.GetComponent<MeshRenderer>().enabled = false;
51
52          this.gameObject.transform.GetChild(gameObject.transform.childCount -
53          2).GetComponent<RoomStatus>().workers = "NO␣HAY␣TRABAJADORES␣ASIGNADOS␣" +
54          "\n" + "\n";
55      }
56
57      void OnMouseDown()
58      {
59          //PatientInfo.disablePanel();
60
61          if (!IsMouseOverUI())
62          {
63              if (!isSelected && !globalSelection && GlobalVariables.EDIT_MODE)
64              {
65                  isSelected = true;
66                  globalSelection = true;
67                  position = transform.position;
68                  rotation = transform.rotation;
69
70                  if(addedReferences)
71                  {
72                      deleteReferences();
73                      addedReferences = false;
74                  }
75
76
77              }
78              else if (!isSelected && !globalSelection &&
79              GlobalVariables.DELETE_MODE)
80              {
81                  if (addedReferences)
82                  {
83                      deleteReferences();
```

```
84                      addedReferences = false;
85                  }
86                  Destroy(this.gameObject);
87
88              }
89              else
90              {
91                  if (!isColliding && isSelected)
92                  {
93                      if(this.gameObject.GetComponent<RoomComprobations>().
94                      isReachable())
95                      {
96                          addReferences();
97                          addedReferences = true;
98
99                          this.gameObject.transform.GetChild
100                         (this.gameObject.transform.childCount -
101                         2).GetComponent<RoomStatus>().reachable = "";
102
103                         this.gameObject.transform.GetChild
104                         (this.gameObject.transform.childCount -
105                         2).GetComponent<RoomStatus>().updateText();
106                     }
107                     else
108                     {
109                         this.gameObject.transform.GetChild
110                         (this.gameObject.transform.childCount -
111                         2).GetComponent<RoomStatus>().reachable =
112                         "ESTA␣SALA␣ES␣INALCANZABLE" + "\n" + "\n";
113                         this.gameObject.transform.GetChild
114                         (this.gameObject.transform.childCount -
115                         2).GetComponent<RoomStatus>().updateText();
116                     }
117
118                     changeMaterialOfChildren(0);
119                     position = transform.position;
120                     rotation = transform.rotation;
121
122                     int x, z;
123                     GetGridPos(GetMouseWorldPos(), out x, out z);
124
125                     Vector3 posicion;
126                     posicion = GetWorldPosition(x, z);
```

```
127
128                     Vector2 vec = GridController.gridToMatrix(x, z);
129                     x = (int)vec.x;
130                     z = (int)vec.y;
131                     GridController.setPrefabRoom(x, z, this.gameObject);
132                     Debug.Log(x + "␣,␣" + z);
133
134                     isSelected = false;
135                     globalSelection = false;
136                 }
137             }
138         }
139
140     }
141
142     private bool IsMouseOverUI()
143     {
144         return EventSystem.current.IsPointerOverGameObject();
145     }
146     private void Update()
147     {
148         if (GlobalVariables.UI_OPEN)
149         {
150             changeMaterialOfChildren(0);
151             isSelected = false;
152             globalSelection = false;
153         }
154         if (Input.GetMouseButtonDown(1))
155         {
156             if (isSelected)
157             {
158                 objectToRotate = this.transform.rotation * Quaternion.Euler
159                 (0, -90, 0);
160             }
161
162         }
163
164         if (isColliding && isSelected)
165         {
166             changeMaterialOfChildren(2);
167         }
168
169         else if (!isColliding && isSelected) changeMaterialOfChildren(1);
```

```
170
171         if (isSelected == true)
172         {
173             zCoord = Camera.main.WorldToScreenPoint(
174
175             gameObject.transform.position).z;
176
177
178             int x, z;
179             GetGridPos(GetMouseWorldPos(), out x, out z);
180             Debug.Log(GetMouseWorldPos());
181             Vector3 posicion;
182             posicion = GetWorldPosition(x, z);
183
184             transform.position = new Vector3(posicion.x, 0, posicion.z);
185
186         }
187
188
189
190         if (GlobalVariables.EDIT_MODE && !isSelected && !isColliding)
191         {
192             transform.GetChild(0).gameObject.GetComponent<ObjectsOnRoom>()
193             .changeMaterial(0);
194
195             this.gameObject.transform.GetChild(gameObject.transform.childCount -
196             2).GetComponent<RoomStatus>().updateText();
197
198             transform.GetChild(transform.childCount -
199             2).gameObject.GetComponent<MeshRenderer>().enabled = true;
200
201             transform.GetChild(transform.childCount -
202             1).gameObject.GetComponent<SpriteRenderer>().enabled = true;
203
204
205         }
206         else if (GlobalVariables.EDIT_MODE && isSelected)
207         {
208             transform.GetChild(transform.childCount -
209             2).gameObject.GetComponent<MeshRenderer>().enabled = false;
210
211             transform.GetChild(transform.childCount -
212             1).gameObject.GetComponent<SpriteRenderer>().enabled = false;
```

```
213
214         }
215         else if (!GlobalVariables.EDIT_MODE && !isSelected && !isColliding)
216         {
217             transform.GetChild(0).gameObject.GetComponent<ObjectsOnRoom>()
218             .changeMaterial(3);
219
220             transform.GetChild(transform.childCount -
221             2).gameObject.GetComponent<MeshRenderer>().enabled = false;
222
223             transform.GetChild(transform.childCount -
224             1).gameObject.GetComponent<SpriteRenderer>().enabled = false;
225         }
226
227         lastFrameWasEditMode = GlobalVariables.EDIT_MODE;
228
229     }
230
231     private void changeMaterialOfChildren(int index)
232     {
233         //transform.GetComponent<MeshRenderer>().material = material;
234         for (int i = 0; i < transform.childCount - 2; i++)
235         {
236             if (transform.GetChild(i).GetComponent
237             <ObjectsOnRoom>() != null)
238             {
239                 transform.GetChild(i).GetComponent
240                 <ObjectsOnRoom>()
241                 .changeMaterial(index);
242             }
243             else
244             {
245                 for (int j = 0; j < transform.GetChild(i).childCount; j++)
246                 {
247                     if (transform.GetChild(i).GetChild(j).GetComponent
248                     <ObjectsOnRoom>() != null)
249
250                     transform.GetChild(i).GetChild(j).GetComponent
251                     <ObjectsOnRoom>().changeMaterial(index);
252                 }
253
254
255             }
```

```
256
257         }
258     }
259
260     private void LateUpdate()
261     {
262         if (!IsQuaternionInvalid(transform.rotation) &&
263         !IsQuaternionInvalid(objectToRotate))
264         {
265             transform.rotation = Quaternion.Lerp(transform.rotation,
266             objectToRotate, 70f * Time.deltaTime);
267         }
268     }
269
270     private bool IsQuaternionInvalid(Quaternion q)
271     {
272         bool check = q.x == 0f;
273         check &= q.y == 0;
274         check &= q.z == 0;
275         check &= q.w == 0;
276
277         return check;
278     }
279     private Vector3 GetMouseWorldPos()
280     {
281         //(x,y)
282         Vector3 mousePoint = Input.mousePosition;
283
284         //z
285         mousePoint.z = zCoord;
286
287         return Camera.main.ScreenToWorldPoint(mousePoint);
288     }
289
290
291     public Vector3 GetWorldPosition(int x, int z)
292     {
293         return new Vector3(x, 0, z) * 5;
294     }
295
296     public void GetGridPos(Vector3 posicion, out int x, out int z)
297     {
298         x = Mathf.FloorToInt(posicion.x / 5);
```

```
299         z = Mathf.FloorToInt(posicion.z / 5);
300     }
301 
302 
303     void OnCollisionStay(Collision col)
304     {
305         if ((col.gameObject.CompareTag("Building")
306         && isSelected))
307         {
308             isColliding = true;
309         }
310     }
311 
312     void OnCollisionExit(Collision other)
313     {
314         if ((other.gameObject.CompareTag("Building")
315         && isSelected))
316         {
317             isColliding = false;
318         }
319     }
320 
321 
322     public void addReferences()
323     {
324         for (int i = 0; i < transform.childCount - 2; i++)
325         {
326             if (transform.GetChild(i).GetComponent
327             <ObjectsOnRoom>() != null)
328             {
329                 ObjectsOnRoom obj = transform.GetChild(i).GetComponent
330                 <ObjectsOnRoom>();
331 
332                 int index;
333 
334                 switch (obj.objectType)
335                 {
336                     case ObjectsOnRoom.type.ConsultDoctor:
337                         index = consultController.addDoctor
338                         (transform.GetChild(i).transform);
339                         obj.indexInList = index;
340 
341                         break;
```

```
342
343                     case ObjectsOnRoom.type.ConsultPatient:
344                         index = consultController.addPatient
345                         (transform.GetChild(i).transform);
346
347                         obj.indexInList = index;
348                         break;
349
350                     case ObjectsOnRoom.type.None:
351                         break;
352
353                     case ObjectsOnRoom.type.RadiologyDoctor:
354                         index = radiologyController.addDoctor
355                         (transform.GetChild(i).transform);
356                         obj.indexInList = index;
357
358                         break;
359
360                     case ObjectsOnRoom.type.RadiologyPatient:
361                         index = radiologyController.addPatient
362                         (transform.GetChild(i).transform);
363
364                         obj.indexInList = index;
365                         break;
366
367                     case ObjectsOnRoom.type.AnalysisDoctor:
368                         index = analisisController.addDoctor
369                         (transform.GetChild(i).transform);
370
371                         obj.indexInList = index;
372                         break;
373
374                     case ObjectsOnRoom.type.AnalysisPatient:
375                         index = analisisController.addPatient
376                         (transform.GetChild(i).transform);
377
378                         obj.indexInList = index;
379                         break;
380
381                 }
382             }
383         }
384     }
```

```
385
386     public void deleteReferences()
387     {
388         for (int i = 0; i < transform.childCount - 2; i++)
389         {
390             if (transform.GetChild(i).GetComponent
391             <ObjectsOnRoom>() != null)
392             {
393                 ObjectsOnRoom obj = transform.GetChild(i).GetComponent
394                 <ObjectsOnRoom>();
395
396                 switch (obj.objectType)
397                 {
398                     case ObjectsOnRoom.type.ConsultDoctor:
399                         consultController.updateIndexOfDoctors
400                         (obj.indexInList);
401                         break;
402
403                     case ObjectsOnRoom.type.ConsultPatient:
404                         consultController.updateIndexOfPatients
405                         (obj.indexInList);
406                         break;
407
408                     case ObjectsOnRoom.type.None:
409                         break;
410                 }
411             }
412         }
413     }
414 }
```

Camera Controller

Listing B.3: Grid Display

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

public class CameraController : MonoBehaviour
{
    public static CameraController instance;
    public static Transform objectToFollow;

    public float movementSpeed;
    public float speed;
    public float normalSpeed;
    public float fastSpeed;
    public float time;
    public float rotation;
    public float xLimit = 100;
    public float yLimit = 100;
    public int zoomInLimit = 100;
    public int zoomOutLimit = 500;

    public Image img;
    public static Image button;
    public Vector3 zoom;



    public Vector3 newPosition;
    public Quaternion newRotation;
    public Vector3 newZoom;

    public Vector3 dragStartPos;
    public Vector3 dragCurrentPos;

    public Vector3 rotateStartPos;
    public Vector3 rotateCurrenttPos;

    void Start()
```

```
41      {
42          instance = this;
43          newPosition = transform.position;
44          newRotation = transform.rotation;
45          newZoom = camera.localPosition;
46          button = img;
47      }
48
49      // Update is called once per frame
50      void Update()
51      {
52          if(objectToFollow != null)
53          {
54              transform.position = objectToFollow.position;
55
56          }
57          else
58          {
59              HandlePlayerKeyboardInput();
60              HandlePlayerMouseInput();
61          }
62
63          if(Input.GetKeyDown(KeyCode.Escape))
64          {
65              button.color = Color.white;
66              objectToFollow = null;
67          }
68
69      }
70
71      private void HandlePlayerMouseInput()
72      {
73          if (!IsMouseOverUI() && !GlobalVariables.UI_OPEN)
74          {
75              if(Input.GetMouseButtonDown(0) && !DragBuildings.globalSelection &&
76              !GlobalVariables.UI_OPEN)
77              {
78                  Plane plane = new Plane(Vector3.up, Vector3.zero);
79
80                  Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
81
82                  float hitPoint;
83
```

```
84                  if(plane.Raycast(ray , out hitPoint))
85                  {
86                      dragStartPos = ray.GetPoint(hitPoint);
87                  }
88              }
89
90              if (Input.GetMouseButton(0) && !DragBuildings.globalSelection &&
91              !GlobalVariables.UI_OPEN)
92              {
93                  Plane plane = new Plane(Vector3.up, Vector3.zero);
94
95                  Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
96
97                  float hitPoint;
98
99                  if (plane.Raycast(ray, out hitPoint))
100                 {
101                     dragCurrentPos = ray.GetPoint(hitPoint);
102
103                     newPosition = transform.position + dragStartPos - dragCurrentPos;
104                 }
105             }
106
107             if (Input.mouseScrollDelta.y != 0 && !GlobalVariables.UI_OPEN)
108             {
109                 newZoom += Input.mouseScrollDelta.y * zoom;
110             }
111
112             if (Input.GetMouseButtonDown(1) && !DragBuildings.globalSelection &&
113             !GlobalVariables.UI_OPEN)
114             {
115                 rotateStartPos = Input.mousePosition;
116             }
117
118             if (Input.GetMouseButton(1) && !DragBuildings.globalSelection &&
119             !GlobalVariables.UI_OPEN)
120             {
121                 rotateCurrenttPos = Input.mousePosition;
122
123                 Vector3 rotation = rotateStartPos - rotateCurrenttPos;
124
125                 rotateStartPos = rotateCurrenttPos;
126
```

```
127                 newRotation *= Quaternion.Euler(Vector3.up * (rotation.x / 5f));
128             }
129 
130         }
131     }
132     void HandlePlayerKeyboardInput()
133     {
134         if (!IsMouseOverUI() && !GlobalVariables.UI_OPEN)
135         {
136             if (Input.GetKey(KeyCode.LeftShift))
137             {
138                 speed = fastSpeed;
139             }
140             else
141             {
142                 speed = normalSpeed;
143             }
144 
145             if(Input.GetKey(KeyCode.UpArrow)  || Input.GetKey(KeyCode.W))
146             {
147                 newPosition += transform.forward * speed;
148             }
149 
150             if (Input.GetKey(KeyCode.DownArrow) || Input.GetKey(KeyCode.S))
151             {
152                 newPosition += transform.forward * -speed;
153             }
154 
155             if (Input.GetKey(KeyCode.RightArrow) || Input.GetKey(KeyCode.D))
156             {
157                 newPosition += transform.right * speed;
158             }
159 
160             if (Input.GetKey(KeyCode.LeftArrow) || Input.GetKey(KeyCode.A))
161             {
162                 newPosition += transform.right * -speed;
163             }
164 
165             if (Input.GetKey(KeyCode.Q) )
166             {
167                 newRotation *= Quaternion.Euler(Vector3.up * -rotation);
168             }
169 
```

```
170             if (Input.GetKey(KeyCode.E))
171             {
172                 newRotation *= Quaternion.Euler(Vector3.up * rotation);
173             }
174 
175             if (Input.GetKey(KeyCode.R))
176             {
177                 newZoom += zoom;
178             }
179 
180             if (Input.GetKey(KeyCode.T))
181             {
182                 newZoom -= zoom;
183             }
184 
185             newPosition.x = Mathf.Clamp(newPosition.x, -xLimit, xLimit);
186             newPosition.z = Mathf.Clamp(newPosition.z, -yLimit, yLimit);
187 
188             newZoom.y = Mathf.Clamp(newZoom.y, zoomInLimit, zoomOutLimit);
189             newZoom.z = Mathf.Clamp(newZoom.z, -zoomOutLimit, -zoomInLimit);
190 
191             transform.position = Vector3.Lerp(transform.position,
192             newPosition, time * Time.deltaTime);
193 
194             transform.rotation = Quaternion.Lerp(transform.rotation,
195             newRotation, time * Time.deltaTime);
196 
197             camera.localPosition =
198             Vector3.Lerp(camera.transform.localPosition, newZoom, time
199             * Time.deltaTime);
200         }
201     }
202 
203     private bool IsMouseOverUI()
204     {
205         return EventSystem.current.IsPointerOverGameObject();
206     }
207 
208     public static void setObjectToFollow(GameObject gameObject)
209     {
210         Debug.Log(gameObject.name);
211         objectToFollow = gameObject.transform;
212         button.color = Color.green;
```

```
213         Debug.Log(button.name);
214 
215     }
216 
217     public static void deleteObjectToFollow(GameObject gameObject)
218     {
219         if(objectToFollow == gameObject) objectToFollow = null;
220     }
221 }
```

Character Generator

Listing B.4: Character Generator

```
1
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5
6  public class CharacterGenerator : MonoBehaviour
7  {
8      public GameObject prefab;
9
10     public Material[] materialesPelo;
11     public Material[] materialesPiel;
12     public Material[] camsieta;
13     public Material[] pantalon;
14     public Material[] ojos;
15
16     public GameObject[] pelosHombre;
17     public GameObject[] pelosMujer;
18     public GameObject[] peloFacial;
19
20     private List<GameObject> genertaedCharactersList;
21
22     public PopulateWorkerShop workerShop;
23
24     public Material bata;
25
26     private int generatingCount = 18;
27
28     public GameObject parent;
29
30     void Start()
31     {
32         genertaedCharactersList = new List<GameObject>(20);
33
34
35
36         for (int i = 0; i < generatingCount; i++)
37         {
38             int genero = Random.Range(0, 2);// 0 --> M || 1 --> F
39
40
```

```
41              int ranType = Random.Range(0, 4);
42
43              int colorDePelo = Random.Range(0,materialesPelo.Length);
44
45              GameObject instance = Instantiate(prefab, this.transform.position + new Vector3((
46              instance.transform.SetParent(parent.transform);
47              genertaedCharactersList.Add(instance);
48
49
50
51              switch (ranType)
52              {
53                  case 0:
54                      instance.GetComponent<Worker>().setType("Receptionnist");
55                      instance.GetComponent<Worker>().role = "Receptionnist";
56                      instance.AddComponent<Recepcionsit>();
57                      break;
58
59                  case 1:
60                      instance.GetComponent<Worker>().setType("Consult");
61                      instance.GetComponent<Worker>().role = "Consult";
62                      instance.AddComponent<Consult>();
63                      break;
64
65                  case 2:
66                      instance.GetComponent<Worker>().setType("Radiologist");
67                      instance.GetComponent<Worker>().role = "Radiologist";
68                      instance.AddComponent<Radiologist>();
69                      break;
70
71                  case 3:
72                      instance.GetComponent<Worker>().setType("Analist");
73                      instance.GetComponent<Worker>().role = "Analist";
74                      instance.AddComponent<Analist>();
75                      break;
76              }
77
78              int ranBonuses = Random.Range(0, 10);
79
80              switch (ranBonuses)
81              {
82
83                  case 0:
```

```
84
85                  instance.GetComponent<Worker>().walkingSpeedBonus = 3;
86                  break;
87
88              case 1:
89
90                  instance.GetComponent<Worker>().treatingSpeedBonus = 9;
91                  break;
92
93              case 2:
94
95                  instance.GetComponent<Worker>().moneyBonus = 15;
96                  break;
97          }
98
99          if (genero == 0)
100         {
101             instance.GetComponent<Worker>().gender = "Male";
102             int randomPelo = Random.Range(0, pelosHombre.Length);
103
104             string name = Names.getNameMale();
105             instance.GetComponent<Worker>().name = name;
106             instance.name = name;
107
108             if (randomPelo != materialesPelo.Length)
109             {
110                 GameObject pelo = Instantiate(pelosHombre[randomPelo],
111                 genertaedCharactersList[i].transform, false);
112
113                 pelo.name = "Pelo";
114                 pelo.transform.rotation = Quaternion.Euler(-90f, 0, 0);
115                 pelo.transform.localScale = new Vector3(1f, 1f, 1f);
116                 pelo.transform.localPosition = new Vector3(0f, 0f, 0f);
117             }
118
119             int randomBarba = Random.Range(0, 11);
120             if (randomBarba == 0 || randomBarba == 1)
121             {
122                 GameObject barba = Instantiate(peloFacial[randomBarba],
123                 genertaedCharactersList[i].transform, false);
124
125                 barba.name = "PeloFacial";
126                 barba.transform.rotation = Quaternion.Euler(-90f, 0, 0);
```

```
127                     barba.transform.localScale = new Vector3(1f, 1f, 1f);
128                     barba.transform.localPosition = new Vector3(0f, 0f, 0f);
129                 }
130             }
131 
132             else
133             {
134                 int randomPelo = Random.Range(0, pelosMujer.Length);
135 
136                 string name = Names.getNameFemale();
137                 instance.GetComponent<Worker>().name = name;
138                 instance.name = name;
139                 instance.GetComponent<Worker>().gender = "Female";
140                 if (randomPelo != materialesPelo.Length)
141                 {
142                     GameObject pelo = Instantiate(pelosMujer[randomPelo],
143                     genertaedCharactersList[i].transform, false);
144 
145                     pelo.name = "Pelo";
146                     pelo.transform.rotation = Quaternion.Euler(-90f, 0, 0);
147                     pelo.transform.localScale = new Vector3(1f, 1f, 1f);
148                     pelo.transform.localPosition = new Vector3(0f, 0f, 0f);
149                 }
150             }
151 
152 
153 
154 
155             int children = genertaedCharactersList[i].
156             transform.childCount;
157 
158             for (int j = 0; j < children; ++j)
159             {
160                 int ran = Random.Range(0, materialesPiel.Length);
161 
162                 if(genertaedCharactersList[i].transform.GetChild(j).
163                 GetComponent<SkinnedMeshRenderer>() != null)
164                 {
165                     if (genertaedCharactersList[i].transform.GetChild(j).
166                     name == "Cejas")
167                     {
168                         genertaedCharactersList[i].transform.GetChild(j).
169                         GetComponent<SkinnedMeshRenderer>().material =
```

```
170                         materialesPelo[colorDePelo];
171
172                     }
173                     else if (genertaedCharactersList[i].transform.GetChild(j)
174                     .name == "Piel")
175                     {
176                         genertaedCharactersList[i].transform.GetChild(j).
177                         GetComponent<SkinnedMeshRenderer>().material =
178                         materialesPiel[ran];
179
180                     }
181                     else if (genertaedCharactersList[i].transform.
182                     GetChild(j).name == "Bata")
183                     {
184                         genertaedCharactersList[i].transform.GetChild(j).
185                         GetComponent<SkinnedMeshRenderer>().material = bata;
186                     }
187                     else if (genertaedCharactersList[i].transform.GetChild(j).
188                     name == "Camiseta")
189                     {
190                         int randomCamiseta = Random.Range(0, camsieta.Length);
191                         genertaedCharactersList[i].transform.GetChild(j).
192                         GetComponent<SkinnedMeshRenderer>().material =
193                         camsieta[randomCamiseta];
194                     }
195                     else if (genertaedCharactersList[i].transform.GetChild(j).
196                     name == "Pantalones")
197                     {
198                         int randomPantalon = Random.Range(0, camsieta.Length);
199                         genertaedCharactersList[i].transform.GetChild(j).
200                         GetComponent<SkinnedMeshRenderer>().material =
201                         pantalon[randomPantalon];
202                     }
203                     else if (genertaedCharactersList[i].transform.GetChild(j)
204                     .name == "Ojos")
205                     {
206                         int ojosRandom = Random.Range(0, ojos.Length);
207                         genertaedCharactersList[i].transform.GetChild(j).
208                         GetComponent<SkinnedMeshRenderer>().material =
209                         ojos[ojosRandom];
210                     }
211                 }
212                 else if (genertaedCharactersList[i].transform.GetChild(j).
```

```
213                 GetComponent<MeshRenderer>() != null)
214                 {
215                     genertaedCharactersList[i].transform.GetChild(j)
216                     .GetComponent<MeshRenderer>().material =
217                     materialesPelo[colorDePelo];
218                 }
219
220
221
222             }
223         }
224
225         workerShop
226             .setUI(genertaedCharactersList);
227     }
228
229
230 }
```

WorkerAI

Listing B.5: WorkerAI

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

public class WorkerAI : MonoBehaviour
{
    private State state = State.WaitingForTask;
    private CurrentTask currentTask = CurrentTask.nullTask;

    private float maxWaitingTime = 1f;
    private float waitingTime = 1f;

    //[SerializeField]
    private TaskManagement taskManagement;
    private TaskManagement.TaskClean task;

    private Vector3 target;

    Renderer rend;

    public Color c;

    public bool working = false;

    private bool sub_task1 = false;
    private bool sub_task2 = false;
    private bool sub_task3 = false;

    private bool runing = false;

    private TaskManagement.TaskClean taskClean;
    private TaskManagement.TaskCleanStain taskCleanStain;

    private NavMeshAgent agent;

    public GameObject mancha;

    private GameObject Stain;
```

```
41
42      private Vector3 comprobacion = new Vector3(123f, 321f, 456f);
43
44      NavMeshAgent navMeshAgent;
45      private enum CurrentTask
46      {
47          task1,
48          task2,
49          task3,
50          nullTask,
51      }
52      private enum State
53      {
54          WaitingForTask,
55          DoingTask,
56          DoingTaskClean,
57      }
58
59      private void Start()
60      {
61          navMeshAgent = this.GetComponent<NavMeshAgent>();
62          taskManagement = TaskManagement.Instance;
63          state = State.WaitingForTask;
64          currentTask = CurrentTask.nullTask;
65
66          agent = this.GetComponent<NavMeshAgent>();
67      }
68
69      private void Update()
70      {
71          if (target != null && target != comprobacion &&
72          agent.remainingDistance >= 1.5f)
73          {
74              //target = comprobacion;
75              //Vector3 rotation = Quaternion.LookRotation(target).eulerAngles;
76              //rotation.y = 0f;
77              //rotation.z = 0f;
78
79              transform.LookAt(target);
80          }
81
82          if (state == State.WaitingForTask && working &&
83          gameObject.GetComponent<NavMeshAgent>()!= null)
```

```
84          {
85              waitingTime -= Time.deltaTime;
86
87              if (waitingTime <= 0)
88              {
89                  waitingTime = maxWaitingTime;
90                  RequestTask();
91                  RequestTaskClean();
92              }
93          }
94
95          if (state == State.DoingTask && working)
96          {
97              ManageTaskClean(taskClean);
98
99          }
100         else if (state == State.DoingTaskClean && working)
101         {
102             Stain = taskCleanStain.trash;
103             ManageTaskCleanStain(taskCleanStain);
104
105         }
106
107     }
108
109
110
111     private void ManageTaskClean(TaskManagement.TaskClean taskClean)
112     {
113         if (sub_task1 == false && !runing)
114         {
115             target = taskClean.position;
116             currentTask = CurrentTask.task1;
117             callCoroutine();
118         }
119
120         else if (sub_task1 == true && !sub_task2
121         && !runing)
122         {
123             currentTask = CurrentTask.task2;
124             callCoroutine();
125         }
126
```

```
127
128         else if (sub_task1 == true && sub_task2 &&
129         !sub_task3 && !runing)
130         {
131             Debug.Log("r2");
132             target = taskClean.position2;
133             currentTask = CurrentTask.task3;
134             callCoroutine();
135         }
136         else if (sub_task1 == true && sub_task2
137         && sub_task3)
138         {
139             Debug.Log("He acabado todo");
140
141
142
143             StopAllCoroutines();
144             RestartValues();
145             target = Vector3.zero;
146
147         }
148     }
149
150     private void ManageTaskCleanStain(TaskManagement.
151     TaskCleanStain taskClean)
152     {
153         if (sub_task1 == false && !runing)
154         {
155             target = taskClean.position;
156             currentTask = CurrentTask.task1;
157             callCoroutine();
158         }
159
160         else if (sub_task1 == true && !sub_task2
161         && !runing)
162         {
163             currentTask = CurrentTask.task2;
164             callCoroutine();
165         }
166
167         else if (sub_task1 == true && sub_task2)
168         {
169             Debug.Log("He acabado todo");
```

```
170             Destroy(Stain.gameObject);
171             Stain = null;
172 
173             StopAllCoroutines();
174             RestartValues();
175             navMeshAgent.isStopped = true; ;
176 
177         }
178     }
179 
180     private void RestartValues()
181     {
182         //agent.isStopped = true;
183         taskClean = null;
184 
185         state = State.WaitingForTask;
186 
187         sub_task1 = false;
188         sub_task2  = false;
189         sub_task3 = false;
190 
191         runing = false;
192     }
193 
194     public void callCoroutine()
195     {
196         runing = true;
197         if (currentTask == CurrentTask.task2
198         && state == State.DoingTaskClean)
199         {
200             target = comprobacion;
201             StartCoroutine(FadeOut());
202         }
203         else if (currentTask == CurrentTask.task2
204         && state == State.DoingTask)
205         {
206             sub_task2 = true;
207             runing = false;
208         }
209         else
210         {
211             StartCoroutine(ExampleFunction());
212         }
```

```
213
214     }
215
216     public void RequestTask()
217     {
218         taskClean = taskManagement.RequestTask();
219         if (taskClean != null)
220         {
221             state = State.DoingTask;
222         }
223     }
224
225     public void RequestTaskClean()
226     {
227         taskCleanStain = taskManagement.
228         RequestTaskClean();
229         if (taskCleanStain != null)
230         {
231             state = State.DoingTaskClean;
232         }
233     }
234
235
236
237     IEnumerator ExampleFunction()
238     {
239         bool end = false;
240         agent.destination = target;
241         while (!end)
242         {
243
244             if (agent.remainingDistance <= 0.1f
245             && agent.pathPending == false)
246             {
247                 end = true;
248             }
249
250             if (end)
251             {
252                 //state = State.WaitingForTask;
253
254                 if (currentTask == CurrentTask.task1)
255                 {
```

```
256                     //Debug.Log("Fin de la tarea 1");
257                     sub_task1 = true;
258                     runing = false;
259                     yield break;
260                 }
261                 else if (currentTask == CurrentTask.task3)
262                 {
263                     //Debug.Log("Fin de la tarea 2");
264                     sub_task3 = true;
265                     runing = false;
266                     yield break;
267 
268                 }
269                 yield break;
270             }
271 
272             yield return null;
273         }
274     }
275 
276 
277     IEnumerator FadeOut()
278     {
279         LeanTween.alpha(Stain, 0f, 2f).setDelay(0f);
280         yield return new WaitForSeconds(2);
281         sub_task2 = true;
282         runing = false;
283     }
284 
285 }
```