UNIVERSITAT JAUME·I

Universitat Jaume I
Video game Design and Development Degree

VJ1241: Final degree project

Memory

Title: Creation of an environment and automatic maze generation for multiplayer video game.

Author: Francesc Sánchez Rodrigo

Tutor: José Miguel Sanchiz Martí

CHAPTER 1: SUMMARY AND INDEXES

Summary

This document the technical proposal for the implementation of a multiplayer environment of a Shooter video game.

Will proceed with the creation of a database and implementing a server application that will communicates with the game client (made with Unity3d). In addition, the necessary infrastructure for the management and creation of user accounts will be created through a website. The video game will use the mechanics and other elements developed by other partners. The database will be created with PostgreSQL and the server with a C# console application.

In addition to the multiplayer system and their game modes, a maze generator will be created, that will take part on the design of one of the maps that will be used in the matches.

To evaluate the proper functioning of the game and the infrastructure, tests will be done with multiple, simultaneous, clients.

Key words

Video game programming, multiplayer, database, Unity 3D, postgreSQL, Shooter, maze generator, unet, networking, server.

Table of Contents

Index of tables

Index of Figures

Index of codes

CHAPTER 2: TECHNICAL PROPOSAL

1. Introduction

Every video game made in this grade was mainly thought, designed and developed as a single player games. In the second and third year, in the subjects "VJ1220 Bases de Datos" and "VJ1228 Redes y Sistemas Multijugador", were taught the basis of the operation of databases, and how to develop a simple server application. However, at any other moment of this degree, the knowledge acquired was never applied in a direct way in a video game development.

In this Final Degree Project is intended to apply the knowledges of those two subjects in a practical way on a video game, developing the required applications and libraries to create a multiplayer game.

The video game will have several online game modes, where players will participate in an active way and will be awarded by experience points and in-game items following a progression line. Both items and experience will be associated to each player account, so a good communication between the game client and the server that manages the database will be required.

The server application will be developed with C# in a console project, that allows create applications in a simple and fast way, although a visual interface is not provided. The database will be created with PostgreSQL by being a simple database management system, open source and free. For the development of the tools of creation and management of user accounts, a web platform will be developed with HTML, javascript and PHP.

In addition to the multiplayer system, a maze generator will be implemented using the Prim's algorithm. It will be the map used in-game. The main feature of this map is the randomness, so the algorithm must be able to create a different maze every time it is loaded. Furthermore, it should be able to create empty and wide rooms that will not overlap each other.

The video game demo will be developed with Unity3d engine, using the UNet service for the networking [1], by being and engine easy to use, and is the most known for me from the knowledge and experience acquired in the degree.

2. Context of the project

2.1. Multiplayer games

Currently, one of the game modes more demanded in the videogames industry is the multiplayer one. Through him, most players meet to play competitive and cooperative, and even creates communities on the network to share information, knowledge, help new players, and coordinate in more complex games, adding a world around a video game that probably be forgotten if it were not for the ability to interact with other players of the world in one way or another.

As a player, I've participated on these communities during the past year's, meeting new people and participating in different kind of events on several video games, even if being rewarded with satisfactory results as if not.

However, in these environments there are always players that only wants to create an atmosphere of tension with toxic behaviours, harming the quality of game to other players. This will be a factor to consider, so as will be explained below in section 2.4, in the infrastructure itself will have to create methods that penalize in some way these players.

2.2. How the Multiplayer works in most games

The most games with multiplayer includes a level and experience system, creating with that the progression of the game, so having high level means more progression.

Each certain number of levels, the game rewards the player with new or extra items that can be equipped to grant new and different functionalities to the avatar. These elements can be obtained for free and can be used immediately after leveling up, or are only unlocked to be available buying them with some in-game virtual currency, limiting the players and offering a sensation of more participation in the progression of the game, because it needs more effort to get the items.

The experience, and the virtual currency if the game has some, are usually obtained as a reward at the end of the matches depending of the performance in them. The matches may have different objectives to achieve the victory, and can be played on groups or individually in an all versus all mode. The more common game modes in the competitive games are "Deathmatch", in addition to other modes like "Capture the flag" and "Control points". These three game modes, with other ones will be explained in more detail in the 2.3.5 section.

2.3. My multiplayer implementation

In this section will be explained how each part of the project will be done and what will be included.

2.3.1. Database

The database will be designed as seen in the "VJ1220 Bases de Datos" subject, following the instructions of the theory book [2]. It will be started creating a conceptual design respecting the entity-relationship model, proceeding then with the relational logical design, and finally with the implementation of the SQL sentences needed for the creation, insert and update of the data and tables.

The database will contain all the information concerning user accounts and match making, as can be basic login data, experience and level, and match results.

Sensible user information, as passwords, will be stored encrypted with SHA1 in their corresponding table, so they will not be visible in case of data stealing or other security problems.

2.3.2. Login and accounts server

To ensure that the registered users can get logged in and receive the information of their accounts without allowing the game client handle any SQL query, a server application will be implemented. That application will process and respond to the client sending the needed data through the use of sockets as seen in the "VJ1228 Redes y Sistemas Multijugador" subject.

As indicated in the previous section, passwords will be encrypted for security. This encryption will be made on the game client, and the messages sent to the server will only contain the result of the encryption to prevent to send the password as plain text and getting obtained through some hacking technique to intercept packets.

2.3.3. Account management webpage

To log in into the game will be necessary some system to create accounts and manage them. Is here where the web page comes.

It will include a form to create new accounts, and some section to recover or change the password in case of forgetting it.

The webpage will use the database, so it will be necessary to manage queries using PHP and javascript and show the result in the browser.

After doing some important operation, an email will be sent to the provided direction to notify the user with the changes made.

2.3.4. Game Launcher app

Most online video games have a launcher app that has a last news section and checks for newer versions to download to maintain the game up to date. As a multiplayer game that can have multiple updates frequently, is necessary to have a game launcher that checks if there is a newer version and download it from a server to not have any advantage or disadvantage of playing with some version or other.

When updating the game it cannot be opened until all update operations are finished. After that, a "Play" button will be enabled and pressing it will open the most recent version of the game.

2.3.5. Planned in-game features

The game will have a progression system based on getting experience when finishing matches and level up.

For playing with other players, there will be a matchmaker, that will create matches and assign players to them.

To solve the problem of the toxic players, a report system will be implemented. After reporting someone, after a certain amount of reports of after being revised, the player can be banned from the game temporarily or permanently.

Regarding to the game modes that will be, the next ones will be implemented:

- **"Deathmatch"**: Mode where the player or team with the most number of kills wins.
- **Capture the flag**: Two teams of 5 players each one that will fight to capture the enemy flag. Each team will have its own flag placed on their base. The enemy flag can be picked up and should be carried to the allied base to be delivered and get a point. If the allied flag has been stolen, the enemy flag can't be delivered until the allied one is retrieved defeating the enemy carrier.
- **Points of control**: Two teams of 5 players each one will fight to capture some control points placed on a strategical points on the map. These control points can be captured being near, and the more allied players near, faster the capture will be. However, if two players (one of each team) are at the same time near the points, these are marked as "conflicted", stopping temporarily the capture. Getting control points and maintain them under control will reward the team with points every few seconds. The winner will be the team with more points after a lapse of time or being the first reaching a certain amount of points.
- **Virus**: All versus all mode where a random player will be selected as infected. The infected player will have a little advantage of walking speed and he may not be defeated, however shooting to him will slow down. The goal of this mode for

the infected players is infect other players to get points, and for the not infected, still healthy until the end. The match will end after a certain amount of time or when all players are infected.

- **Suitcase carrier**: A similar mode to capture the flag, but instead of playing by teams is an all versus all, and there is only one flag (the suitcase). The suitcase can be carried and doing it will give points to its carrier while not defeated. The winner will be the player with more points after a certain amount of time.
- **Automatic teleport**: An all versus all game mode, with exactly the same rules as the arena mode, but with the addition that the positions of all players will be exchanged between them.

2.4. Randomized maps

Maps from the most video games are always static, i.e., are always the same and the players ends knowing every spot with maximum detail, creating a disadvantage to new players that ever played that map. To solve this problem, i suggest to randomize some parts of the map or create an entire randomized one, a thing that can give much play. The randomized parts will be different every time they are loaded, removing that disadvantage by making all players not known how is the current map.

2.5. Maze generation (Depth-first search)

It will be based on a matrix of nodes, each one corresponding to a "square" that will form the maze, with his four sides (north, south, east and west) stored in them.

The maze will have rooms inside, creating a wide and open areas. These rooms will be created from a random selection of one of the nodes. When a node is selected, the cells that will be part of the corresponding room will be marked as visited and to be room with an extra variable, excluding them to be selected as part of the maze when the generation algorithm is running.

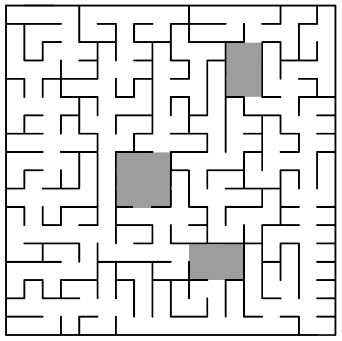
The rooms can't be overlapped or placed in the borders, so a check will be need to correct the positions of these special cases or to decide if the position is discarded and to generate other.

Once the rooms are created, it will proceed with the creation of the corridors of the maze, using a **randomized depth-first search** [3] following the next steps:

1. Starting from a matrix of nodes with walls, get randomly one of the nodes, that will be the current one.
2. Mark the current node as visited.
3. Get the unvisited neighbours nodes of the current node and select randomly one of them.

   a. If there aren't unvisited nodes around the current node, the last element put in the backtracking list is popped and will become the current node and the algorithm repeats this step.
4. Delete the wall that connects the current node with the neighbour node.
5. Mark the neighbour node as visited.
6. The current node is stored in a list to ease the backtracking in future iterations.
7. The current node is changed with the neighbour node and the algorithm continues until there aren't nodes in the list repeating steps 2 to 7.

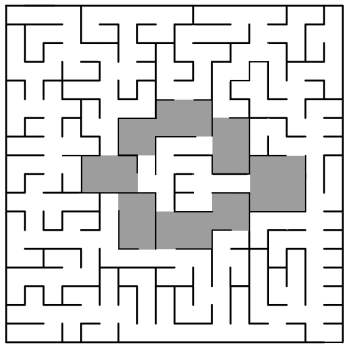
Figure 1: Example of the result of a maze of 18 x 18 with three rooms of random sizes between 2 and 3.



On this implementation with rooms can be special cases that isolate parts of the node matrix creating two groups of nodes.

To deal with these cases, a solution can be adding an extra variable as a counter that will have at any moment the counting of visited nodes (including the ones that are part of the rooms). For the example in the Figure 1, the counter will be initialized to 21 (3*2 + 2*3 + 3*3 = 21) and the algorithm to the creation of the maze will be executed as explained above but adding 1 each time a new node is visited. When the algorithm ends with the list empty, if the counter not corresponds to the maximum of possible visited cells (in the case of the Figure 1 it will be 18*18 = 324), then the algorithm is executed again from a randomly selected node from the unvisited remaining ones.

Figure 2: Example of a maze of 18 x 18 with two maze zones, separated with the positions of the 8 rooms.



3. Project objectives

As it indicates the teaching guide of the subject [4], the competences to be acquired by the end of the course are:

• *"Capacidad para realizar individualmente y presentar y defender ante un tribunal universitario un ejercicio original, consistente en un proyecto en el ámbito del diseño y desarrollo de videojuegos de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas."*

Also learning outcomes are the following:

• *"Planificar e implementar individualmente un proyecto original de naturaleza profesional en el ámbito de los videojuegos y en el que se sinteticen e integren las competencias adquiridas en las enseñanzas."*

• *"Redactar una memoria en inglés, presentar y defender oralmente ante un tribunal universitario un proyecto original de naturaleza profesional en el ámbito de los videojuegos y en el que se sinteticen e integren las competencias adquiridas en las enseñanzas."*

Therefore, one of the objectives of this project will be learn to perform in a professional way, presenting a realistic planning, adjusted to the time set for the project.

The main objective of this project, as to what regards the specific work to be performed, is the development and learning of how to do an implementation of a multiplayer mode in a videogame using existing tools (as extra Unity plugins or classes) and other of my own creation, achieving a stable and error free multiplayer game environment.

In a more specific way, the following list shows the objectives that are wished to fulfill with the realization of this project:

• **OBJ1: Develop a multiplayer environment similar to that in some important games of the industry.** This is the main objective of the project, creating a multiplayer environment and all the necessary tools, knowing from first hand which features were successful and which not on the various games I played.

• **OBJ2: Expand the knowledge of Unity3D to apply them in a professional project.** The other video games made with Unity have been mostly little or test projects for learning of some parts of the engine. Therefore, it's important to expand the knowledge and above all the experience when working with Unity3D to be able to apply them in a professional project.

• **OBJ3: Learn and experiment with new technology. Self learning.** Keep learning things by experimenting with them to create and implement features not seen in the degree, or not seen in depth. The self learning is a very important tool to progress in a world in which a new thing is outdated in one second.

• **OBJ4: Implement a database with good performance and table relations.** To provide good performance to the retrieval, update and new insertions of data to the database is important to create a clean design with the essential relationships between tables, without exceeding the necessary ones and not creating more tables and registers than needed.

• **OBJ5: Create a stable server communication framework.** The server communication should be fast to provide a good user experience when sending and receiving sensible data, like passwords and usernames.

• **OBJ6: Create a multiplayer game modes.** The game should have a matchmaker where players can join to matches and wait to their start. In addition, on the matches, the game modes should be easy to understand and play, and the objectives of each mode should have a scoring system that allows to determine the winner.

• **OBJ7: Create a functional registration and account management webpage.** The users should have a mean to create new accounts and manage them, in addition to download the game for playing.

• **OBJ8: Create a tool that helps to maintain the client always updated.** All users should have the same version of the game when playing on a multiplayer game. That is a safe practice to avoid having multiple versions at the same time that can have some unfixed bugs, or removed features creating version inconsistency.

4. Justification

This section will comment how the proposed project relates to the Video game Design and Development Degree and therefore see how the proposed project is suitable to be considered as final project.

The proposed project is related mainly to two subjects, "VJ1220 Bases de Datos" and "VJ1228 Redes y Sistemas Multijugador" as well with other in a lesser degree. In the remainder of this section is discussed the relationship with more detail.

4.1. VJ1220 Bases de Datos

In this second year mandatory subject, is taught how to create a database, from the basic design to an implementation using a database manager, as PostgreSQL, which was used in the practices of the subject and will also be used in this project.

In this subject the following skills are arised:

• IB04 - Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación en ingeniería.

From which the following results are obtained:

• IB04 - Diseñar una base de datos relacional a partir de la especificación de un problema.

• IB04 - Explicar los conceptos fundamentales de los sistemas de bases de datos relacionales, sus objetivos y arquitectura.

• IB04 - Formular consultas de recuperación y actualización de datos en bases de datos relacionales utilizando lenguajes estándar.

Since during the course of this project an implementation of a database is done, is necessary to know how to design a database and how to perform queries to it. It is also important to explain these concepts, in order to do a good job and a good writing for the final memory.

4.2. VJ1228 Redes y Sistemas Multijugador

In this third year mandatory subject, is taught how the network works and the different elements that form them. Both in theory and practice classes, at the end, were taught to implement and know the functioning of a web server. In this project the explanations and knowledge acquired during the realization if the web server will be used to create one that manages the database, performs the login operations and send data to the client.

In this subject the following skills are arised:

• E05 - Capacidad para desarrollar juegos en red para múltiples jugadores/as.

• IR05 - Conocimiento, administración y mantenimiento de sistemas, servicios y aplicaciones informáticas

en el ámbito de los videojuegos.

• *IR11 - Conocimiento y aplicación de las características, funcionalidades y estructura de los Sistemas Distribuidos, las Redes de Computadores e Internet y diseñar e implementar aplicaciones basadas en ellas.*

• *IR14 - Conocimiento y aplicación de los principios fundamentales y técnicas básicas de la programación paralela, concurrente, distribuida y de tiempo real.*

From which the following results are obtained:

• *E05, IR05, IR11 - Describir el funcionamiento de los servidores de juegos.*

• *E05, IR05, IR11 - Elaborar aplicaciones en red.*

• *E05, IR05, IR11 - Usar los protocolos básicos de la capa de aplicación en Internet.*

• *E05, IR14 - Describir los principios fundamentales y las técnicas básicas de la programación paralela, concurrente, distribuida y de tiempo real.*

• *E05, IR05, IR11 - Usar motores de juegos multijugador actuales.*

The first four results are related to the creation of the server and client interaction by using the basic protocols of the application layer (the sockets), while the last one is applied when programming the necessary elements of the game client, which will use Unity as engine.

4.3. Other related subjects

• "VJ1203 Programación I": The basics of programming are learned.

• "VJ1205 Inglés": Both the proposal and the memory must be in English.

• "VJ1208 Programación II": The basics of object oriented programming are learned.

• "VJ1215 Algoritmos y Estructuras de Datos": Problems are solved using advanced algorithms and data structures, which will be used in implementing the maze generator.

• "VJ1224 Ingeniería del Software": Tasks of project planning are performed, in addition to how to manage and organize projects with agile methodologies.

• "VJ1227 Motores de Juegos": In this subject is learnt how an engine works and to use Unity 3D, that will be used in the realization of this project to the programming of the client side.

5. Planning

In this section are the tasks to perform in this project, the duration and relation to the objectives is presented. A list of deliverables is also presented.

5.1 Tasks

Below, In the *Table 1,* are presented the tasks to do to develop this project.

Table 1: List of tasks and their associated subtasks.

T01 - Documents of the project
T01-S01 - Technical proposal drafting. Preparation and drafting of the technical proposal.
T01-S02 - Final memory drafting. Preparation and drafting of the final memory.
T02 - Server side
T02-S01 - Database. Design and implementation of a database with PostgreSQL.
T02-S02 - Design of SQL queries. Creation of the needed SQL queries to get all desired information.
T02-S03 - Server application (application layer). Implementation of the communication and messages sent and received to a basic client application using sockets.
T02-S04 - Server application (SQL queries). Integrations of the database in the server application using the queries created in the T03-S02 task.
T03 - Client side
T03-S01 - Login. Implementation of methods and visual interfaces needed to perform a login.
T03-S02 - Main Menu and User Profile. Implementation of methods and visual interfaces needed to perform the user profile information and previously played matches. In addition to the navigation through the main menu and in-game virtual purchases.
T03-S03 - Match list and Lobby. Implementation of the match list and lobby before starting a match.
T03-S04 - Game modes (multiplayer). Implementation of the game mode mechanics oriented to multiplayer matches.

T04 - Testing and debugging
T04-S01 - Debugging. Performance tests will be conducted throughout the development to fix any possible errors.
T05 - Other
T05-S01 - Maze generator. Implement the maze generator as explained in the section 2.2
T05-S02 - Account management webpage. Implementation of a simple webpage that offers to the users services as creation of new accounts and password recovery among others.
T05-S03 - Game Launcher. Implementation of the game launcher that will be used mainly to check and download new client updates from a server.

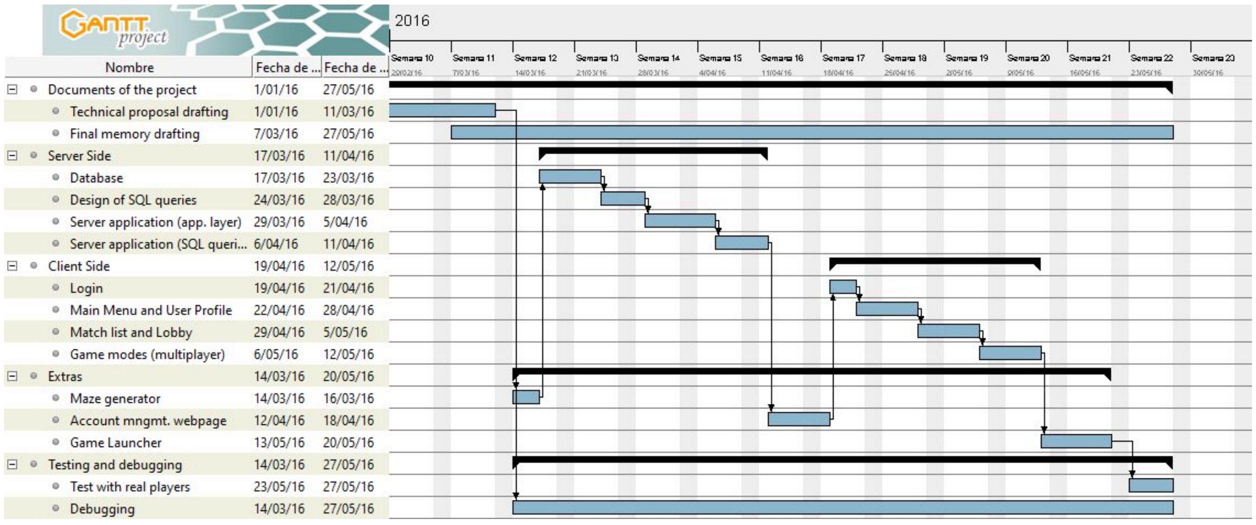
The Table 2 associates the objectives of the section 4 with the subtasks explained in the previous table.

Table 2: List of objectives and their associated subtasks.

Objectives	Tasks
OBJ1	T02-S03, T02-S04, T03-S01, T03-S02, T03-S03, T03-S04
OBJ2	T03-S01, T03-S02, T03-S03, T03-S04, T04-S01
OBJ3	T02-S03, T02-S04, T05-S02, T05-S03, T04-S01
OBJ4	T02-S01, T02-S02
OBJ5	T02-S03, T02-S04, T03-S01, T03-S02, T04-S01
OBJ6	T03-S03, T03-S04, T05-S01, T04-S01
OBJ7	T05-S02
OBJ8	T05-S03

The Figure 3, seen below, shows a Gantt diagram showing in a graphical way the planning and the estimated days for each task and subtask, and the order of proceeding with them.

Figure 3: Gantt diagram with tasks and subtasks for this project.



Nombre	Fecha de ...	Fecha de ...
Documents of the project	1/01/16	27/05/16
Technical proposal drafting	1/01/16	11/03/16
Final memory drafting	7/03/16	27/05/16
Server Side	17/03/16	11/04/16
Database	17/03/16	23/03/16
Design of SQL queries	24/03/16	28/03/16
Server application (app. layer)	29/03/16	5/04/16
Server application (SQL queri...	6/04/16	11/04/16
Client Side	19/04/16	12/05/16
Login	19/04/16	21/04/16
Main Menu and User Profile	22/04/16	28/04/16
Match list and Lobby	29/04/16	5/05/16
Game modes (multiplayer)	6/05/16	12/05/16
Extras	14/03/16	20/05/16
Maze generator	14/03/16	16/03/16
Account mngmt. webpage	12/04/16	18/04/16
Game Launcher	13/05/16	20/05/16
Testing and debugging	14/03/16	27/05/16
Test with real players	23/05/16	27/05/16
Debugging	14/03/16	27/05/16

The Technical Proposal is being made since January, having been completely ruled out an earlier version, that's why it was started long ago and took lot of hours and days.

5.2. Deliverables

Table 3 shows the deliverables, the objectives and tasks related to the deliverables, and the date expected as deadline.

Table 3: List of deliverables and their objectives and tasks.

Deliverables	Objectives	Tasks	Estimated time
D1: Technical Proposal	---	T01-S01	60 hours
D2: Database	OBJ1, OBJ5	T02-S01	20 hours
D3: Server app	OBJ1, OBJ6	T02-S03, T02-S04, T04-S01	55 hours
D4: Webpage	OBJ1, OBJ4, OBJ8	T04-S01,T05-S02	20 hours
D5: Game Launcher	OBJ1, OBJ4, OBJ9	T04-S01, T05-S03	15 hours
D6: Videogame demo	OBJ1, OBJ3, OBJ7	T03-S01, T03-S02, T03-S03, T03-S04, T04-S01, T04-S02, T05-S01	60 hours
D7: Memory	---	T01-S02	70 hours

In the list, as D1, this document, as the first deliverable needed to start the project.

As D2, is found the database. For that will be delivered the SQL code corresponding to the queries to create and manage the tables with some examples that will be used in the final implementation.

The server app, D3, will be delivered with a little example scene made with Unity with buttons and a text to see how the server responds and receives data from a client.

The next one, D4, is the webpage and the delivery will include all files that form it.

D5 is the game launcher application. This app will check on the database if there is a newer version of the game. If the result is true it will try to download the game. As the game probably is not finished at that point, the file downloaded will be the demo client from D3.

On D6 is found the videogame demo. It will include all deliverables from D2 prepared to work together without any placeholder data, since this deliverable should demonstrate the correct functioning of all systems, and the game.

And finally, D7, the memory of the project, that will be done periodically and revised at the end of the development before deliver it to be evaluated with the rest of the project.

6. Methodology

The project is developed individually, but it will take part of a bigger project after the deliver. As said, the project is made only by me, so everything that will be used or delivered will be developed or created by me.

I'm in charge to create the multiplayer environment by making the necessary tools.

For the code will be used a version control. The chosen one will be GitHub for the project, by being faster, and GitLab, by allowing to have more private repositories and organize them in groups of projects.

One of my objectives were maintain and follow a good programming naming standards by following the Microsoft Naming Guidelines [5]. One of the purposes of this objective is to ease the reading of the code to other people and being used to do it in that way. In addition to that, all methods will be commented following the Visual Studio templates that allows the explanations to be shown on the tooltips of IntelliSense.

Since the first task will be started until the end project, the hours each one will be counted and their will be presented on the memory.

7. Tools

The following table, Table 4, shows the tools that will be used to each deliverable explained in the section 5.2.

Table 4: List of tools that will be used in relation to deliverables.

Deliverables	Tools
D1: Technical Proposal	Documents by Google, GanttProject and Adobe Photoshop
D2: Database	PostgreSQL, Notepad++, Dia and Documents by Google.
D3: Server app	Visual Studio 2015 (C#)
D4: Webpage	Notepad++, EasyPHP, Google Chrome and Mozilla Firefox
D5: Game Launcher	Visual Studio 2015 (Visual C#)
D6: Videogame demo	Unity3D, Visual Studio 2015 (C#)
D7: Memory	Documents by Google, GanttProject, Dia and Adobe Photoshop

The videogame demo will be made in **Unity3D**, and will be used the built-in networking system for the multiplayer implementation for the matches.

The scripts for Unity, the server application and the game launcher, will be made with **Visual Studio 2015**, by being a powerful IDE which facilitates and saves time at programming.

For the edition and creation of the webpage and the SQL queries will be used **Notepad++**, by being a versatile tool when dealing with plain text files. PHP scripts of the webpage will be tested using a local web server provided by **EasyPHP**, that has everything prepared to be used and is easy to configure and use.

The documents that will be necessary to write will be done using **Documents by Google**, since they allow access from every computer or mobile phone with internet, also the possibility of easy sharing.

If some graphic is needed, **Adobe Photoshop** will be used to create it and export as image to be included lately in the corresponding documents.

The diagrams for the database and others will be done with **Dia**, a free and easy to use software for drawing UML diagrams. The gantt diagram of this proposal is made with **GanttProject**.

8. Risks and contingency plan

The realization of any project involves a number of risks, so you need to prepare a series of measures in case risks become reality.

R1: Experimenting with new technology. One of my objectives is learn to do new things. That is a risk, but I try persistently to achieve what I want to do. If some part is not understand at hundred percent, I will try to get help from grade teachers, tutor, specialized books or online documentation.

R2: Time constraints.

The time constraints is one of the risks that every project has. This project concretely will coexist with the external practices so not the entire day is available to work on it. If the project progresses too slow or can't be completed because that the solution will be to ask to the practices tutor and boss to get some extra time for the project.

CHAPTER 3: PROJECT DEVELOPMENT

1. Introduction

On this chapter the project development will be covered. As seen in the previous chapter, there are many deliverables and applications to develop. Each one will be explained deeply on individual sections, as do the planning, methodology, problems that have appeared and a conclusion.

2. Planning

On the section 5 of the previous chapter, there was a table with all tasks and subtasks. During the development of the project, some tasks were merged, and new ones appeared.

On the following table, Table 5, there are the tasks resulting of the modifications to the planning made during the development. With red yellow background, the tasks that are result of a merge of two tasks, and with green background the new ones.

Table 5: List of tasks and subtasks with the changes made during the development.

T01 - Documents of the project
T01-S01 - Technical proposal drafting. Preparation and drafting of the technical proposal.
T01-S02 - Final memory drafting. Preparation and drafting of the final memory.
T02 - Server side
T02-S01 - Database design and implementation Design and implementation of a database with PostgreSQL.
T02-S02 - Server application (application layer). Implementation of the communication and messages sent and received to a basic client application using sockets.
T02-S03 - Server application (SQL queries). Integrations of the database in the server application using the queries created in the T03-S02 task.
T03 - Client side
T03-S01 - Login. Implementation of methods and visual interfaces needed to perform a login.

T03-S02 - Main Menu, User Profile and MatchMaker. Implementation of methods and visual interfaces needed to create the user profile and a list of previously played matches. In addition to matchmaker and in-game virtual purchases.
T03-S03 - Game modes (multiplayer). Implementation of the game mode mechanics oriented to multiplayer matches.
T04 - Testing and debugging
T04-S01 - Debugging. Performance tests will be conducted throughout the development to fix any possible errors.
T05 - Other
T05-S01 - Maze generator. Implement the maze generator as explained in the section 2.2
T05-S02 - Account management webpage. Implementation of a simple webpage that offers to the users services as creation of new accounts and password recovery among others.
T05-S03 - Game Launcher. Implementation of the game launcher that will be used mainly to check and download new client updates from a server.
T05-S04 - Shared Library Creation of a shared library of classes between the client and server to have the same come on all of them and avoid the appearance of bugs only on some shared class by a result of a bad copy and paste.

The merged tasks have no big impact on the development of the project. The new **T02-S01** subtask, "**Database design and implementation**", now includes the design of the SQL queries necessary for **T02-S03**. On the technical proposal, this task only included the design and implementation of the tables and relations of the database, and has on a separated one the design of the SQL queries.

The other merged task, the new **T03-S02**, "**Main Menu, User Profile and MatchMaker**", was merged to unite the Main menu with the Matchmaker options. On this new task, the main menu is the matchmaker menu, showing the options on this screen without need to change between two or more menus.

And finally, the last task, **T05-S04** , "**Shared Library**", the extra task that appeared during the development, had a great impact on the development, simplifying some parts of the multiple projects. This task is now a deliverable too, so it will be discussed and explained more in his own subsection of the section 3.

On the following table, the Table 6, there is the new deliverable list. On it, there are the objectives and tasks associated to each one, and an approximation of hours dedicated to develop them.

The hours of the new deliverable are not taken in account on the total sum of the project because they were counted on the other deliverables after separating it to an individual project.

Table 6. The new deliverables table with the approximate hours dedicated to each one.

Deliverables	Objectives	Tasks	Approximate hous
D1: Technical Proposal	---	T01-S01	55 hours
D2: Database	OBJ1, OBJ5	T02-S01	10 hours
D3: Server app	OBJ1, OBJ6	T02-S02, T02-S03, T04-S01	100 hours
D4: Webpage	OBJ1, OBJ4, OBJ8	T04-S01,T05-S02	20 hours
D5: Game Launcher	OBJ1, OBJ4, OBJ9	T04-S01, T05-S03	15 hours
D6: Videogame demo	OBJ1, OBJ2, OBJ3, OBJ7	T03-S01, T03-S02, T03-S03, T04-S01, T04-S02, T05-S01	50 hours
D7: Shared Library	OBJ3, OBJ5	T02-02, T04-S01, T05-S04	*
D8: Memory	---	T01-S02	50 hours
			Total: 300 hours

3. Development of the project

After knowing the changes made to the planning, in this section will be explained in detail how the development were to each deliverable (excluding D1: Technical Proposal and D8: Memory). Each one will be presented with examples and explanations of what they do, and the reason to do what they do.

3.1. Database

The start of the project is the design and implementation of the database. The database was designed for PostgreSQL, following the design guidelines learned on the degree.

3.1.1. Database specification

First, we start to work from a database specification. The database specification is a document where the user explains what the database should store and how, in natural worlds, nothing technic. That document is commonly written by the clients when asking to someone to create a database, but in this case, I'm my own "client" so I write it myself to help me understand how it should work, and to identify better the entities and attributes.

Code 1: The specification written (in spanish), where the highlighted words indicates the entities and the underlined ones the attributes.

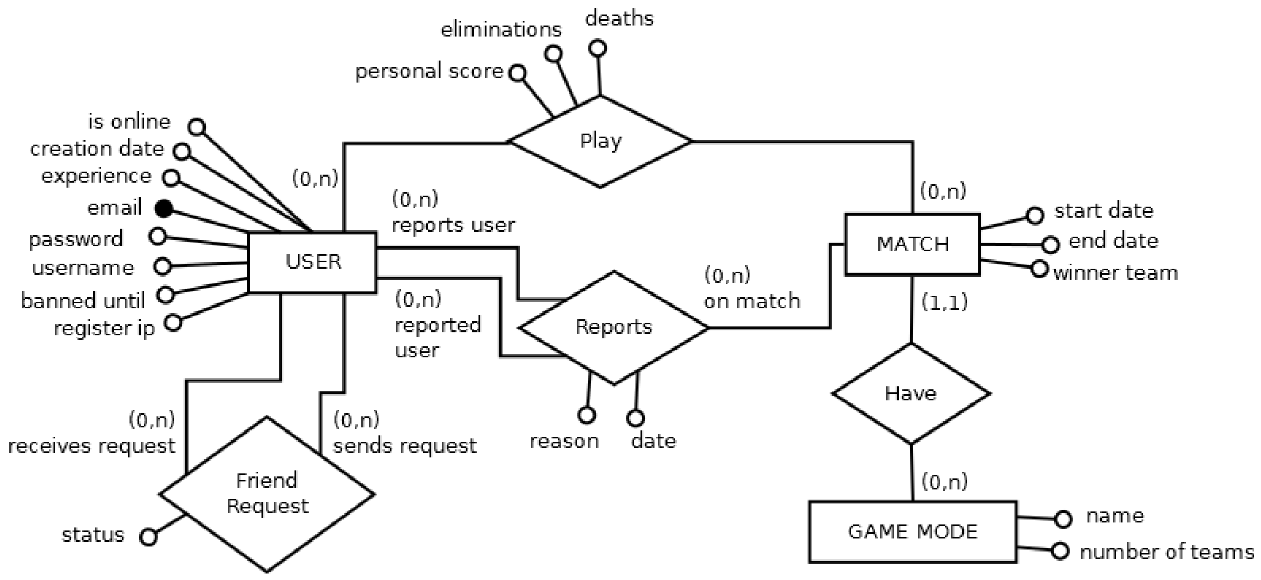
```
Necesitamos una base de datos que guarde las cuentas de los usuarios.
Estas cuentas necesitarán introducir de forma obligatoria un correo
electrónico, una contraseña y un nombre de usuario. Además, cada usuario
tendrá un nivel y un valor de experiencia que aumentará tras cada
partida. Interesa guardar por ciertos motivos la fecha de la creación de
la cuenta.

Los usuarios pueden hacerse amigos de otros. Para ello, enviarán una
solicitud de amistad al otro usuario, que podrá aceptar, cancelar o
mantener pendiente.

El juego tendrá partidas, y sus datos estarán almacenados en la base de
datos. Cada partida corresponderá a un modo de juego diferente, por lo
que según a qué modo corresponda los integrantes jugarán en equipo con
otros jugadores o lo harán de forma individual en un combate de todos
contra todos. En las partidas interesa conocer la fecha de inicio y la de
final, además de los resultados de los jugadores y del equipo. El
resultado del equipo se usará para indicar el equipo o jugador ganador de
la partida (por ejemplo, en una partida de todos contra todos, se puede
considerar que cada jugador forma parte de un equipo distinto). El
resultado de los jugadores, se usará siempre, y estará formado por un
valor de puntuación personal, número de eliminaciones y número de bajas
propias (muertes).
```

```
Los jugadores podrán realizar reportes de otros jugadores que se estén
comportando de una forma no adecuada en medio de una partida. Para ello
se pulsará un botón el el juego y se abrirá una ventana en la que
escribirá la razón del reporte. Además, se tendrá que guardar la fecha
del reporte para conocer el momento exacto en que se realizó.
```

3.1.2. Entity-relationship model

After being identified every entity and attribute, the next step is to make the conceptual design. This is a graphic with some geometric shapes and symbols that recreates the design of the database. These graphics and symbols are standard and represents the entity-relationship model.

Figure 4: Entity-relationship model of the database.



On this diagram, the relationships between tables are represented with a diamond, and on the lines is written the cardinality of each relation.

For example, the entity USER have two different relationships with MATCH, "Play" and "Reports". The relationship "Play" means that some user can play a match, in more specific way, means that a match can have zero or more users and an user can present be on zero or more matches. This is a relationship many to many with some data stored on it (eliminations, score, and deaths).

MATCH for example, has a relationship with GAME MODE. On this one, a MATCH can only "Have" one and only one GAME MODE, but a GAME MODE can be on zero or more MATCH.

3.1.3. Logical schema

After the entity-relationship model is completed, the next step is create the logical design. On this step the entity-relationship model is decomposed on something more approximate to the tables that the final design will have, obtaining a schema that use in the most efficient way the resources of the selected database management system. This schema will be used lately on the physical design.

The first version of the logical schema is probably not 100% correct, because can exist some compound attributes, so it's necessary to normalize it. In our case wasn't necessary because the schema (Code 2) was already on third normal form (the better one).

In addition, the Primary keys, the Foreign Keys and the Foreign Key Constraints are defined.

A key is a subset of columns in a table that allow a row to be uniquely identified. So, a key can be more than just one column. And, every row in the table will have a unique value for the key or a unique combination of values if the key consists of more than just one column.

- The Primary keys is the main identifier for each row in the table, it's unique and can't be null so each one will be different.
- The Foreign Key is a key that references from one table to another. The referenced column of the referenced table is its Primary key.
- The Foreign Key Constraints are an integrity rules that the database should follow when a value referenced by a foreign key (on the parent table) is modified or deleted. The actions that can perform the database when this happens are the following:
    - Restrict: The deletion of modification of the referenced row is restricted and not performed.
    - Cascade: The modification of the referenced row is propagated to that table changing the value to the new one. The deletion is propagated too, removing all the rows referencing that key.
    - Set NULL: Similar to propagate when the referenced is modified, but the change is not propagated, instead all rows referencing that one are set to null if they allow nulls. When deleting, the parent row is deleted and the rows referencing it are set to null if they allow it.
    - Set Default: The same as "Set NULL", but instead of setting to null, the rows referencing it are set to the default value.

Code 2: The logical design of the database.

```
USERS(user_id, email, username, password, experience, creation_date,
register_ip, banned_until, isonline)
GAMEMODES(gamemode_id, gamemode_name, number_of_teams)
MATCHES(match_id, start_date, end_date, winner_team, gamemode_id)
PLAYS(user_id, match_id, personal_score, eliminations, deaths)
FRIEND_REQUESTS(from_user, to_user, status)
REPORTS(report_id, from_user, to_user, match_id, reason, repot_date)

MATCHES.gamemode_id is a foreign key to GAMEMODES.gamemode_id.
      Accepts NULL: no.
      On DELETE: Restrict.
      On UPDATE: Cascade.
PLAYS.match_id is a foreign key to MATCHES.match_id.
      [...]
PLAYS.user_id is a foreign key to USERS.user_id.
      Accepts NULL: no.
      On DELETE: Set Default.
      On UPDATE: Cascade.
FRIEND_REQUEST.from_user is a foreign key to USERS.user_id.
      Accepts NULL: no.
      On DELETE: Cascade.
      On UPDATE: Cascade.
FRIEND_REQUESTS.to_user is a foreign key to USERS.user_id.
      [...]
REPORTS.from_user is a foreign key to USERS.user_id.
      [...]
REPORTS.to_user is a foreign key to USERS.user_id.
      [...]
REPORTS.match_id is a foreign key to MATCHES.match_id.
      [...]
```

On the schema shown above, the Primary Key of each table is underlined, and the Foreign Key is specified in a written form. The Foreign Key Constraints are written below of the Foreign Key (only three are written for space reasons).

For example, the Primary Key of table PLAYS are two columns, `match_id` and `user_id`. One user can't be two times on the same match.

`MATCHES.gamemode_id` is a foreign key to `GAMEMODES.gamemode_id`, because `MATCHES.gamemode_id` is referencing the primary key of GAMEMODES, `GAMEMODES.gamemode_id`.

The constraints of this key, sets that the key can't be null under any circumstance. When a `GAMEMODES.gamemode_id` is deleted, the operation is restricted if that gamemode_id is present on any match. And if any `GAMEMODES.gamemode_id` is updated, the changes are propagated to the table MATCHES.

3.1.4. Physical design

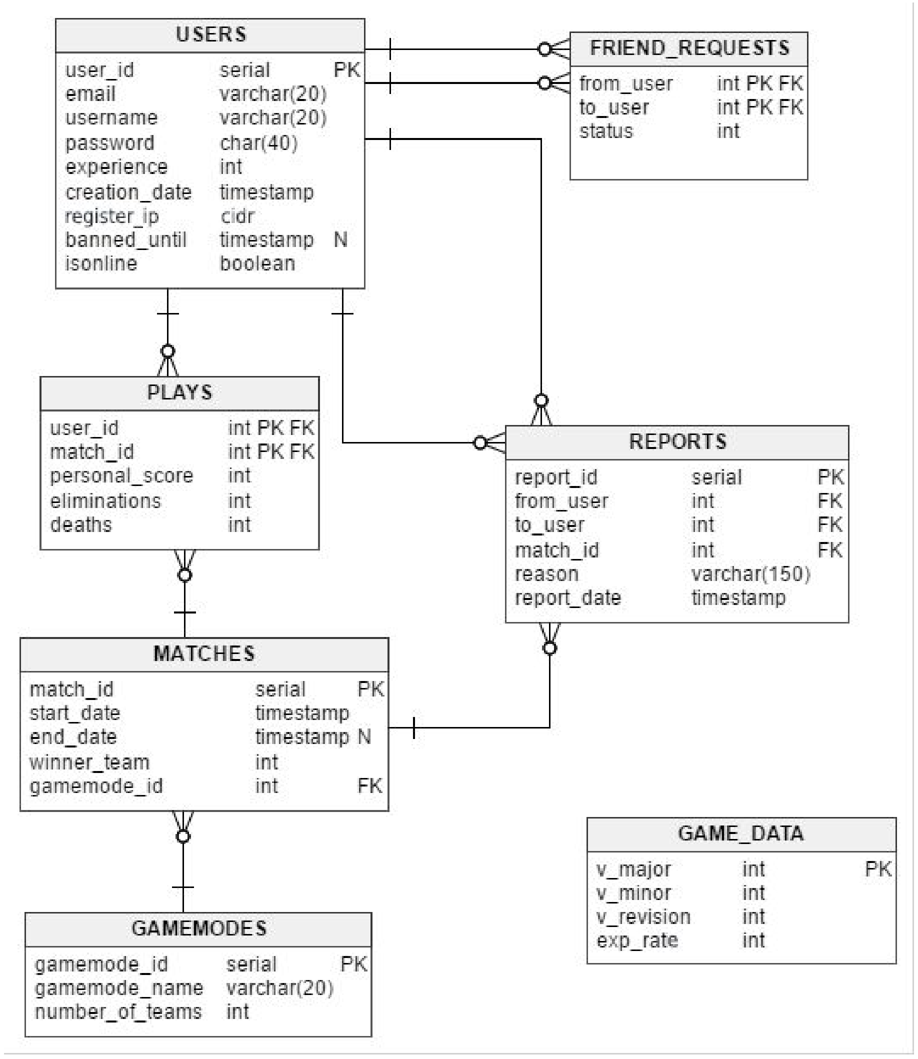
With all of these designs and schemes at this point, is time for the physical design. To create it, Vertabelo [6] was used, an online visual database designer. With some visual tools the tables and relationships of the final design were created, as the attributes with their corresponding data types.

After creating everything, the sql script was downloaded with the CREATE TABLE and ALTER TABLE instructions.

Code 3: Example of the creation of some tables.

```
-- Table: MATCHES
CREATE TABLE MATCHES (
    match_id serial  NOT NULL,
    start_date timestamp  NOT NULL,
    end_date timestamp  NULL,
    winner_team int  NULL,
    gamemode_id int  NOT NULL,
    CONSTRAINT MATCHES_pk PRIMARY KEY (match_id)
);

-- Table: USERS
CREATE TABLE USERS (
    user_id serial  NOT NULL,
    email varchar(20)  NOT NULL,
    username varchar(20)  NOT NULL,
    password char(40)  NOT NULL,
    experience int  NOT NULL DEFAULT 0,
    creation_date timestamp  NOT NULL,
    register_ip cidr  NULL,
    banned_until timestamp  NULL,
    isonline boolean  NOT NULL DEFAULT false,
    CONSTRAINT Email_ak_1 UNIQUE (email) NOT DEFERRABLE  INITIALLY IMMEDIATE,
    CONSTRAINT USERS_pk PRIMARY KEY (user_id)
);
```

Figure 5: Final database physical design.



In addition to all the tables mentioned and created on the previous steps, an extra table called GAME_DATA was included. This table, that has no relationship with any of the others, includes some technical data as the version numbers, and the experience ratio used for awarding the players with some experience bonus (for example, a temporal ratio of x2 to celebrate the launch of the game).

After that, some initial inserts were created to populate the database with some info and test it. The queries were created on Notepad++ just by writing them in plain text.

On the following code block, Code 4, there are some examples of these initial inserts.

Code 4: Some initial inserts for the database.

```sql
INSERT INTO GAMEMODES
(gamemode_name, number_of_teams) VALUES
('T_VS_T__DEATHMATCH',2),
('T_VS_T__CAPTURE_THE_', 2),
('T_VS_T__POINT_OF_CON', 2),
('A_VS_A__DEATHMATCH', 10),
('A_VS_A__VIRUS', 10),
('A_VS_A__SUITCASE', 10),
('A_VS_A__TELEPORT', 10);

INSERT INTO USERS
(email, username, password, experience, creation_date) VALUES
('', 'Unknown user', '', 0, current_timestamp);

INSERT INTO USERS
(email, username, password, experience, creation_date, banned_until)
VALUES
('email@test', 'Banned', '89b58e1ab7233dd84fe7da8b73cdacb48bed7f29', 0,
current_timestamp, current_timestamp),
('email@test2', 'Allowed', '98121f6eef6881e88d43640a7554498c68c69f7c', 0,
current_timestamp, null);
```

3.2. Shared library, TFGCommonLib.dll

3.2.1. Why I needed it?

The first thing I started to develop was the server application and a simple scene in Unity to simulate the login. During the first versions of the code, the shared parts were few. After adding some new functionalities I notices that the common parts were growing, having some classes and some code repeated on both projects. That was a problem, because that code was in development, and doing any change on on side, made the other side with an old version of the code.

Moving these parts to a shared library where to place the common code was an alternative, and a solution easy to do. With this change, I merely have to link the resulting library in .dll format to the other projects. For the Visual Studio solutions was needed to link it as a reference, and for the Unity project, just paste it inside the Assets folder and the engine automatically loads it.

Figure 6: The library, TFGCommonLib, is shared between projects.

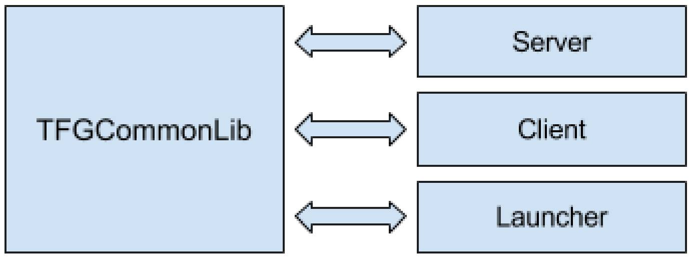
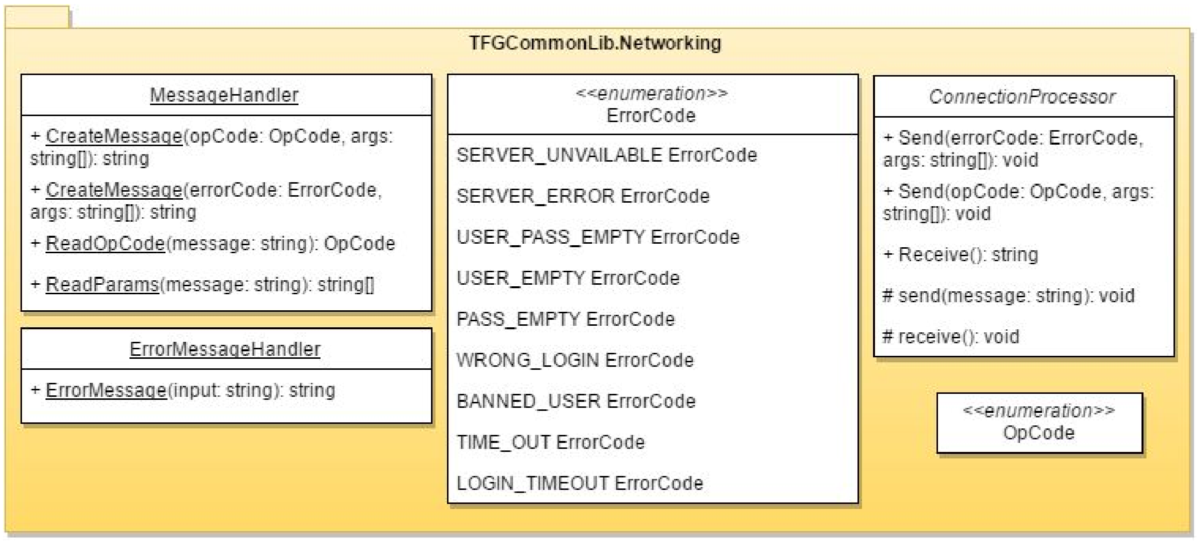
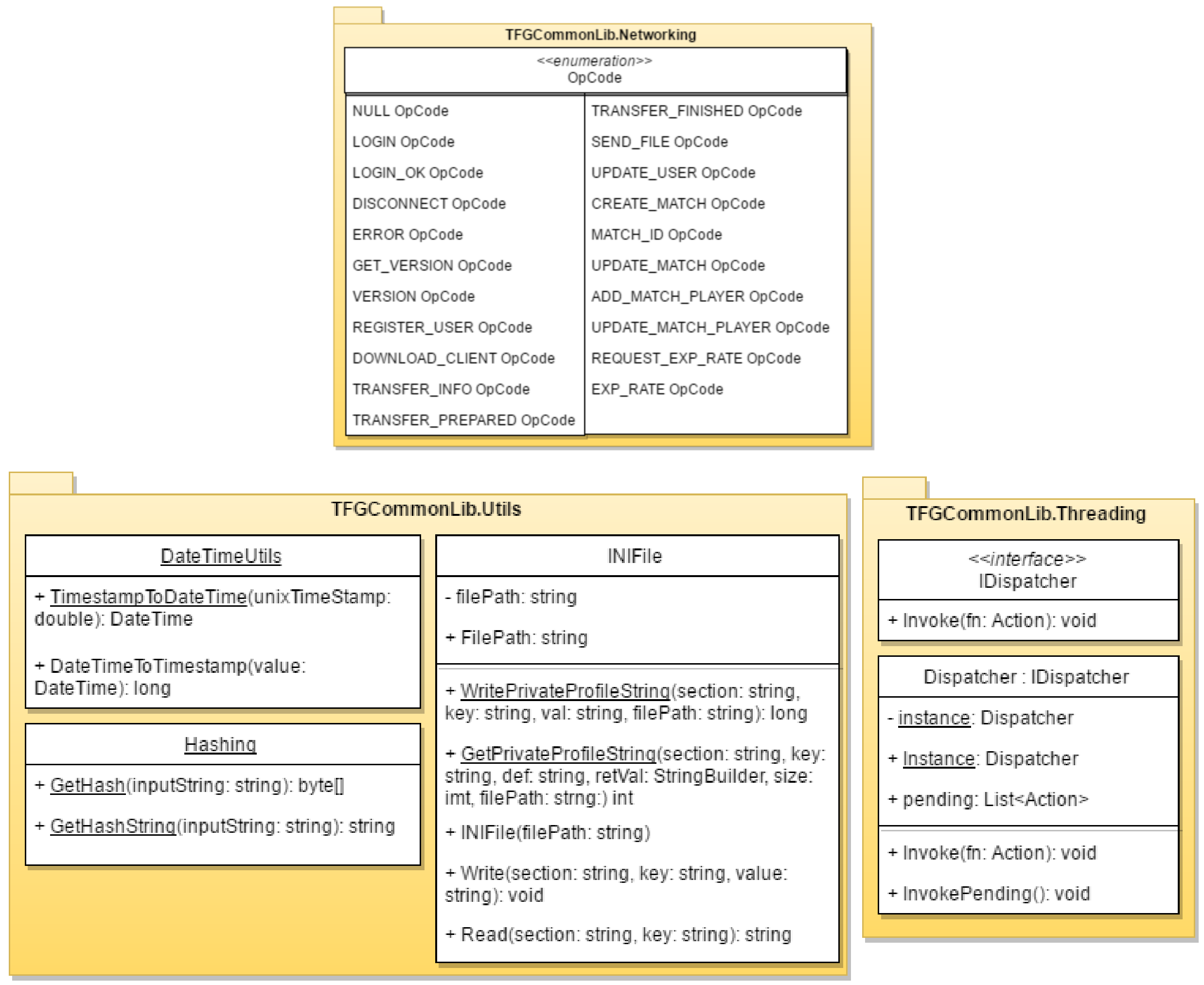


Figure 7. Class diagram of TFGCommonLib (continues on the next page).


TFGCommonLib.Networking
MessageHandler
+ CreateMessage(opCode: OpCode, args: string[]): string
+ CreateMessage(errorCode: ErrorCode, args: string[]): string
+ ReadOpCode(message: string): OpCode
+ ReadParams(message: string): string[]
ErrorMessageHandler
+ ErrorMessage(input: string): string
<<enumeration>>
ErrorCode
SERVER_UNVAILABLE ErrorCode
SERVER_ERROR ErrorCode
USER_PASS_EMPTY ErrorCode
USER_EMPTY ErrorCode
PASS_EMPTY ErrorCode
WRONG_LOGIN ErrorCode
BANNED_USER ErrorCode
TIME_OUT ErrorCode
LOGIN_TIMEOUT ErrorCode
ConnectionProcessor
+ Send(errorCode: ErrorCode, args: string[]): void
+ Send(opCode: OpCode, args: string[]): void
+ Receive(): string
# send(message: string): void
# receive(): void
<<enumeration>>
OpCode



3.2.2. Operation and error codes

Probably the most important part of this library are the items on the TFGCommonLib.Networking namespace.

One of these, are the operation and error codes (OpCode and ErrorCode). These codes are the basic identifiers for the messages sent between the server and the clients. They are two enums, with some special values that represents each code.

To make these codes work there is too a MessageHandler class. On this class the selected OpCodes and ErrorCodes (and their arguments) are transformed into the corresponding messages that will be sent using the sockets. This class is just a helper class for the other projects to translate between codes and messages, and vice versa.

Being placed on the library and not on each project individually, I can be sure that the messages sent are on every project the same, just by recompiling the library.

Code 5: Snippet of code of MessageHandler.CreateMessage(opCode: OpCode, args: string[]).

```
public static string CreateMessage(OpCode opCode, params string[] args) {
	switch (opCode) {
		default:
			return "";
		case OpCode.LOGIN:
			if (args.Length == 2)
				return "LOGIN " + args[0] + " " + args[1] + " \n";
			throw new Exception("ERROR: OpCode \"LOGIN\" needs two arguments (user, 
shaPass).");
//[...]
		case OpCode.EXP_RATE:
			if (args.Length == 1)
				return "EXP_RATE " + args[0] + " \n";
			throw new Exception("ERROR: OpCode \"EXP_RATE\" needs one argument 
(expRate).");
	}
}
```

On the block of code, Code 5, there is part of the method for create the messages for some OpCodes. If the OpCode is tried to used but the number of arguments is not the correct, and exception will be thrown. The list of OpCodes and their meaning will be explained in later sections where they are used.

On the MessageHandler there are too two methods for reading the OpCode of an incoming message, `MessageHandler.ReadOpCode(message: string)`, and the parameters of it, `MessageHandler.ReadParams(message: string)`.

For the ErrorCodes, the class ErrorMessageHandler is used to show some error messages on the screen. On the following table, Table 7, there is a list with all the error codes and their meaning.

Table 7: ErrorCodes list and description.

ErrorCode	Num	Description	Sender
SERVER_UNVAILABLE	-1	The server cannot be reached by the client.	Client to Client
SERVER_ERROR	0	Something happened on the server during the processing of some petition.	Server to Client
USER_PASS_EMPTY	1	The login fields for the username and password are empty.	Client to Client
USER_EMPTY	2	The login field for the username is empty.	Client to Client
PASS_EMPTY	3	The login field for the password is empty.	Client to Client

`WRONG_LOGIN`	4	The login is not correct and don't match with any register on the database.	Server to Client
`BANNED_USER`	5	The user that is trying to be logged in is banned.	Server to Client
`TIME_OUT`	6	The client didn't get any response to some petition.	Any to Client
`LOGIN_TIMEOUT`	7	The login was sent but the server didn't respond.	Client to Client

3.2.3. Other utilities and threading

In addition to the mentioned classes of the TFGCommonLib.Networking namespace, there is an additional class, the abstract ConnectionProcessor class, that defines the basic methods for the socket communication. This class is being derived on the other projects where the code for these methods will be.

Going now to the TFGCommonLib.Utils namespace, the INIFile class is found. This class uses some external methods of kernel32.dll to write and read from an ini file, in addition to an extra ones that facilitates the usage of them. The INIFile class is based on the explained on some article on CodeProject [7].

Other utility is the DateTimeUtils that transforms a DateTime object from C# to a unix timestamp, and vice versa. This is used to send dates on the messages sent by the sockets easily.

The last utility, the Hashing class, contains the code necessary to encrypt the password in SHA1 on the client before sending the message to the server to perform the login.

Code 6: Code of the methods of the class Hashing of the TFGCommonLib.

```
public static byte[] GetHash(string inputString) {
        HashAlgorithm algorithm = SHA1.Create();
        return algorithm.ComputeHash(Encoding.UTF8.GetBytes(inputString));
}

public static string GetHashString(string inputString) {
        StringBuilder sb = new StringBuilder();
        foreach (byte b in GetHash(inputString))
                sb.Append(b.ToString("X2"));
        return sb.ToString();
}
```

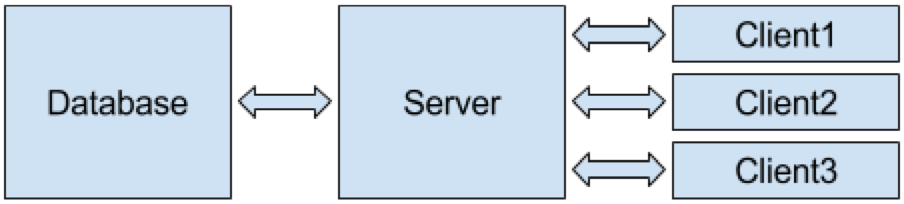
And finally, on the TFGCommonLib.Threading namespace, the Dispatcher is found. The Dispatcher is a class that can allows the execution on the main thread of some code triggered on any other thread. The implementation of the dispatcher is the same as explained in the article of Ashley Davis on the blog "What could possibly go wrong?" [8].

3.3. Server application

3.3.1. Introduction

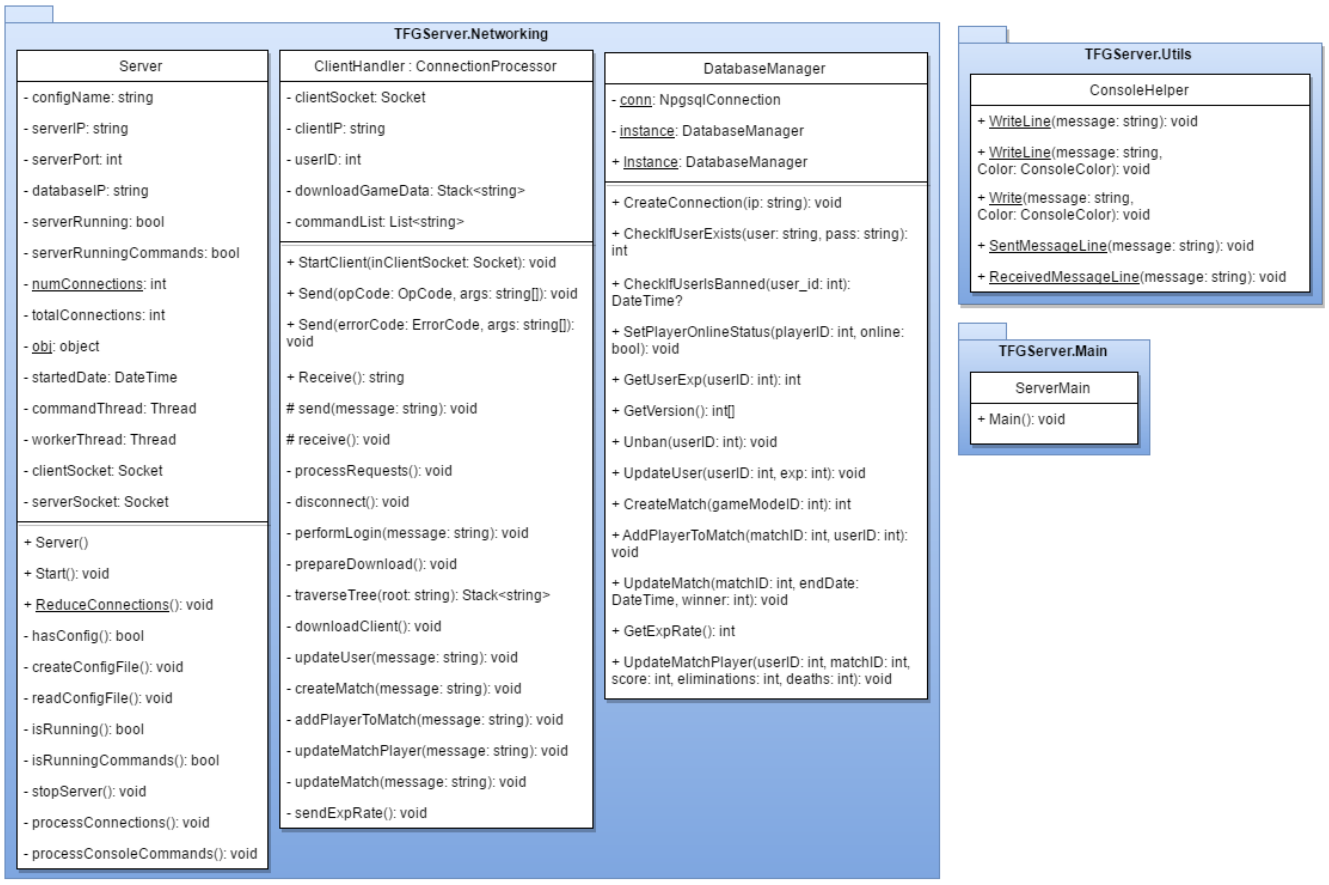
The server application is an application that works as a bridge between the clients and the database. This means that the clients don't handle with any SQL query, creating with this an additional security layer so any external client modification will not affect the database.

Figure 8: The server is always in the middle of the database and the clients.



Everything that the clients needs is asked to the server with sockets using the TCP protocol, sending some messages. When the server process the message, it executes the needed code to provide a response to the client, accessing to the database or doing other actions.

Figure 9: Class diagram of the server application.


TFGServer.Networking
Server
- configName: string
- serverIP: string
- serverPort: int
- databaseIP: string
- serverRunning: bool
- serverRunningCommands: bool
- numConnections: int
- totalConnections: int
- obj: object
- startedDate: DateTime
- commandThread: Thread
- workerThread: Thread
- clientSocket: Socket
- serverSocket: Socket
+ Server()
+ Start(): void
+ ReduceConnections(): void
- hasConfig(): bool
- createConfigFile(): void
- readConfigFile(): void
- isRunning(): bool
- isRunningCommands(): bool
- stopServer(): void
- processConnections(): void
- processConsoleCommands(): void
ClientHandler : ConnectionProcessor
- clientSocket: Socket
- clientIP: string
- userID: int
- downloadGameData: Stack<string>
- commandList: List<string>
+ StartClient(inClientSocket: Socket): void
+ Send(opCode: OpCode, args: string[]): void
+ Send(errorCode: ErrorCode, args: string[]): void
+ Receive(): string
# send(message: string): void
# receive(): void
- processRequests(): void
- disconnect(): void
- performLogin(message: string): void
- prepareDownload(): void
- traverseTree(root: string): Stack<string>
- downloadClient(): void
- updateUser(message: string): void
- createMatch(message: string): void
- addPlayerToMatch(message: string): void
- updateMatchPlayer(message: string): void
- updateMatch(message: string): void
- sendExpRate(): void
DatabaseManager
- conn: NpgsqlConnection
- instance: DatabaseManager
+ Instance: DatabaseManager
+ CreateConnection(ip: string): void
+ CheckIfUserExists(user: string, pass: string): int
+ CheckIfUserIsBanned(user_id: int): DateTime?
+ SetPlayerOnlineStatus(playerID: int, online: bool): void
+ GetUserExp(userID: int): int
+ GetVersion(): int[]
+ Unban(userID: int): void
+ UpdateUser(userID: int, exp: int): void
+ CreateMatch(gameModeID: int): int
+ AddPlayerToMatch(matchID: int, userID: int): void
+ UpdateMatch(matchID: int, endDate: DateTime, winner: int): void
+ GetExpRate(): int
+ UpdateMatchPlayer(userID: int, matchID: int, score: int, eliminations: int, deaths: int): void
TFGServer.Utils
ConsoleHelper
+ WriteLine(message: string): void
+ WriteLine(message: string, Color: ConsoleColor): void
+ Write(message: string, Color: ConsoleColor): void
+ SentMessageLine(message: string): void
+ ReceivedMessageLine(message: string): void
TFGServer.Main
ServerMain
+ Main(): void

3.3.2. Implementation

The server is implemented on a Visual C# console application. This means that the only visual interface that it has is the classical black console with text on it.

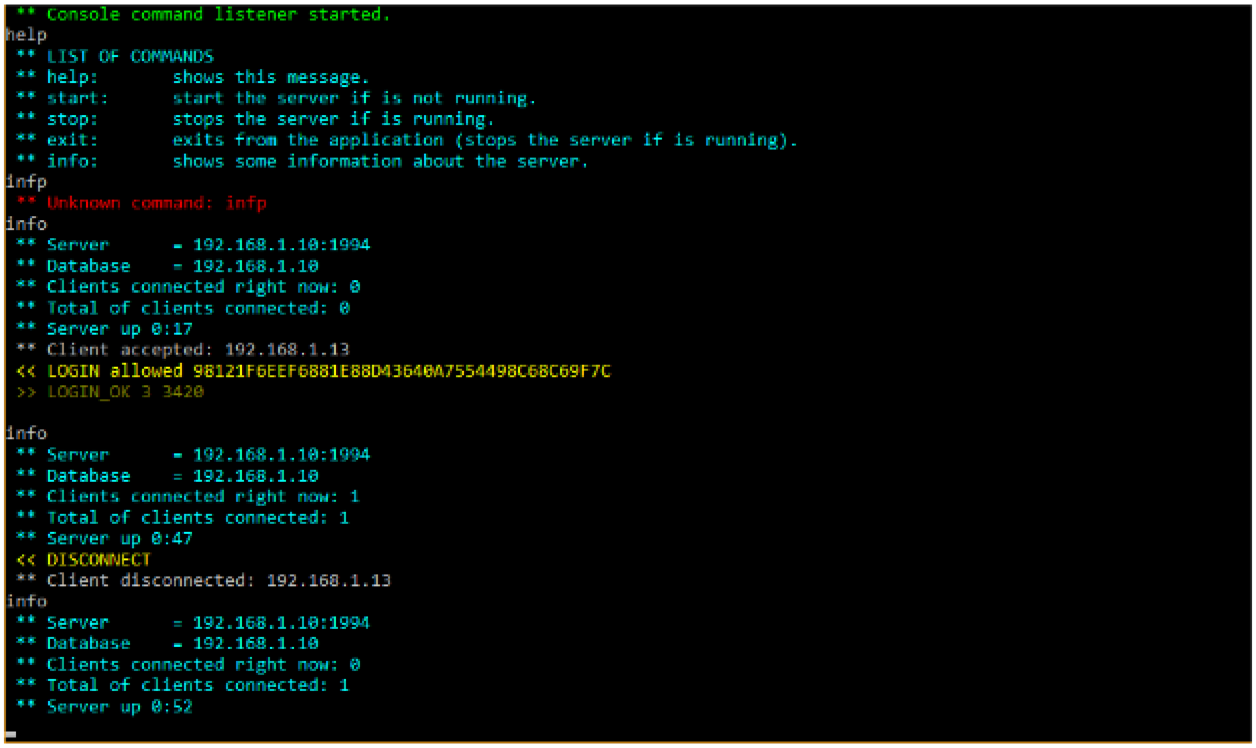
It consists of three namespaces TFGServer.Main, TFGServer.Networking and TFGServer.Utils. The first one, is provided by the template of the console application and is where the Main function is placed. On that function there are only two lines written, the first to create an initiate the Server, and the second to automatically start it.

The important part comes with the TFGServer.Networking, where the classes of the server are placed.

The "Server" class starts reading a config file (or creating it if it doesn't exists). This config file is a .ini file that contains the server IP and port, and the database IP. After that, the server start its processing threads.

There are two threads, the first is used to read and process the commands written in the console. These commands are used to get some info from the server when it's running, or to stop, start or restart it if needed (see figure 10).

Figure 10: The list of the console commands of the server and some info.



```
 ** Console command listener started.
help
 ** LIST OF COMMANDS
 ** help:       shows this message.
 ** start:      start the server if is not running.
 ** stop:       stops the server if is running.
 ** exit:       exits from the application (stops the server if is running).
 ** info:       shows some information about the server.
infp
 ** Unknown command: infp
info
 ** Server      = 192.168.1.10:1994
 ** Database    = 192.168.1.10
 ** Clients connected right now: 0
 ** Total of clients connected: 0
 ** Server up 0:17
 ** Client accepted: 192.168.1.13
 << LOGIN allowed 98121F6EEF6881E88D43640A7554498C68C69F7C
 >> LOGIN_OK 3 3420

info
 ** Server      = 192.168.1.10:1994
 ** Database    = 192.168.1.10
 ** Clients connected right now: 1
 ** Total of clients connected: 1
 ** Server up 0:47
 << DISCONNECT
 ** Client disconnected: 192.168.1.13
info
 ** Server      = 192.168.1.10:1994
 ** Database    = 192.168.1.10
 ** Clients connected right now: 0
 ** Total of clients connected: 1
 ** Server up 0:52
```

And the second thread processes the connections. When it starts, the serverSocket and clientSocket are initialized, and then a "infinite" while loop starts where the incoming connections are accepted. The work is passed to the ClientHandler class, where all operations queried by that client are handled.

Code 7: processConnections() method code.

```
private void processConnections() {
        ConsoleHelper.WriteLine("Server started!", ConsoleColor.Green);

        //The server and client sockets are initialized
        serverSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
        clientSocket = default(Socket);

        IPAddress ipAddress = IPAddress.Parse(serverIP);
        IPEndPoint remoteEP = new IPEndPoint(ipAddress, serverPort);

        //The server socket is bound to an end point and set to listen
        serverSocket.Bind(remoteEP);
        serverSocket.Listen(10);

        try {
                while (isRunning()) {
                        //The server socket accepts connections and creates a new
ClientHandler instance where the client petitions will be processed.
                        clientSocket = serverSocket.Accept();
                        ConsoleHelper.WriteLine("Client accepted: " +
((IPEndPoint)clientSocket.RemoteEndPoint).Address.ToString());
                        ClientHandler client = new ClientHandler();
                        client.StartClient(clientSocket);
                        lock (obj) {
                                numConnections++;
                        }
                }
        }
        catch (Exception e) {
                ConsoleHelper.WriteLine("Server socket closed.", ConsoleColor.Red);
        }

        ConsoleHelper.WriteLine("Server closed.", ConsoleColor.Yellow);
}
```

The ClientHandler class has a thread too. That thread will be only for the corresponding client connected to that ClientHandler, so each client have its own processor.

This class implements the ConnectionProcessor abstract class of TFGCommonLib.Networking, that provides it with the methods for receive and send commands using the client socket.

Code 8: Implementation of the send and receive methods.

```
public override void Send(OpCode opCode, params string[] args) {
        string message = MessageHandler.CreateMessage(opCode, args);
        ConsoleHelper.SentMessageLine(message);
        send(message);
}

protected override void send(string message) {
        byte[] sendBytes = Encoding.ASCII.GetBytes(message);
        if (clientSocket.Connected)
                clientSocket.Send(sendBytes, sendBytes.Length, SocketFlags.None);
}

protected override void receive() {
        byte[] bytesFrom = new byte[1024];
        if (clientSocket.Connected) {
                clientSocket.Receive(bytesFrom);

                string message = Encoding.ASCII.GetString(bytesFrom).TrimEnd((char)0);
                if (!message.Equals("")) {
                        string[] commands = message.Split('\n');

                        foreach (string command in commands) {
                                if (command != "") {
                                        commandList.Add(command);
                                        ConsoleHelper.ReceivedMessageLine(command);
                                }
                        }
                }
        }
}
```

The implementation of the public method `ClientHandler.Send(...)` has two variants, one for sending an OpCode, and other for sending an ErrorCode. The scode hown on the Code 8, creates a message for the provided OpCode, and passes the result to the protected method `ClientHandler.send(message: string)`. On this method, the message in string format is converted into bytes and sent to the client using the method `Socket.Send(buffer: byte[], size: int, socketFlags: SocketFlags)`.

The `ClientHandler.receive()` method it's the opposite of these ones. It uses the `Socket.Receive(buffer: byte[])`, transform that into a string message. That message is splitted in several ones separated by the character `'\n'`. That splitting operation is done because if the client sends lot of messages in a very short period of time, they are received all together in the same receive operation, so some of them can be ignored. Doing that, all messages are taken into account, and stored in a list of string that contains all the received messages.

The messages of that list are processed on the thread of the ClientHandler class. This thread only executes the method `ClientHandler.processRequests()` that receives the incoming messages, and process the list where they are stored.

Code 9: ClientHandler method to process client messages.

```
private void processRequests() {
       while (clientSocket.Connected) {
                try {
                      receive();
                      while (commandList.Count > 0) {
                              string message = commandList[0];
                              commandList.RemoveAt(0);

                              OpCode opcode = MessageHandler.ReadOpCode(message);
                              switch (opcode) {
                                     case OpCode.LOGIN:
                                             performLogin(message);
                                             break;

                                     //[Other switch cases omitted to space reasons]

                                     default:
                                     case OpCode.DISCONNECT:
                                             disconnect();
                                             break;
                              }
                      }
               }
               catch (Exception ex) {
                      Send(ErrorCode.SERVER_ERROR);
                      ConsoleHelper.WriteLine(ex.ToString(), ConsoleColor.Red);
                      disconnect();
               }
       }
}
```

3.3.3. Usage of OpCodes

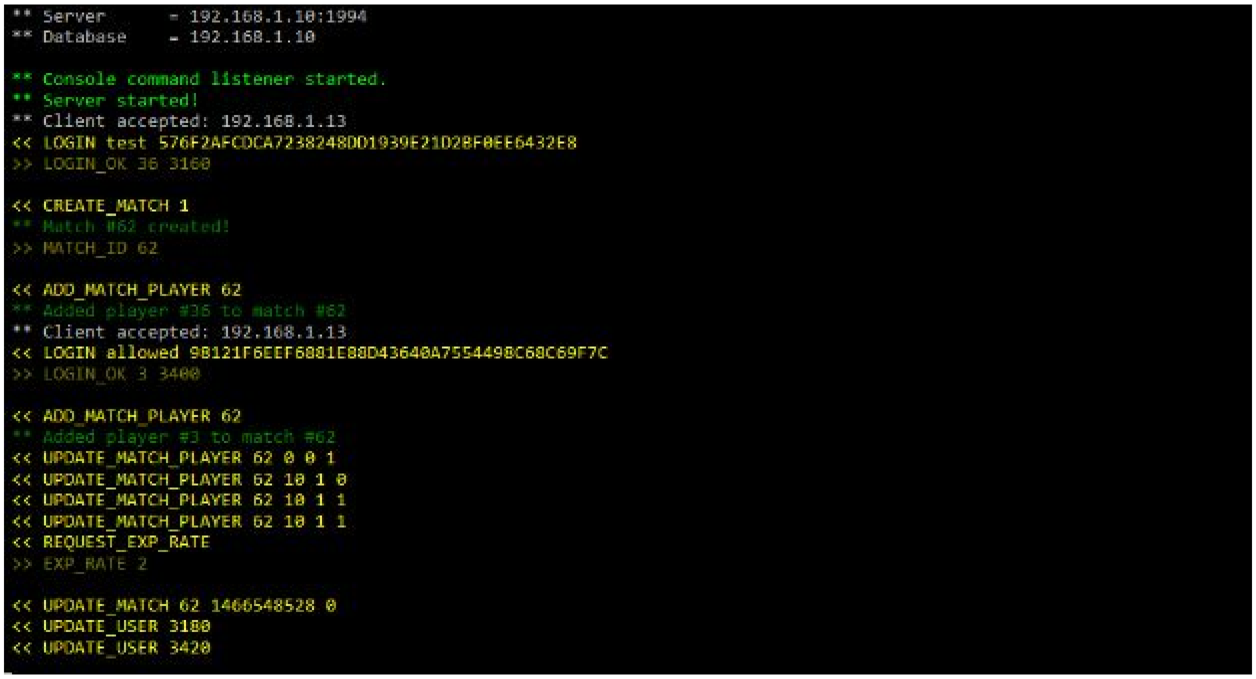
On the following table, the OpCodes used by the server and the client are listed with an explanation and some other data. The OpCodes used by the Launcher will be explained later in its own section.

Table 8: OpCodes used by the client and the server.

OpCode	Num. Arg.	Description	Sender
`NULL`	0	Is not sent or received. It only exist for programming purposes when the OpCode needs to be nullified.	N/A
`LOGIN`	2	The message contains the user name and hashed password to perform the login.	Client
`LOGIN_OK`	2	The response of LOGIN. It sends to the client the userID and the amount of experience.	Server
`DISCONNECT`	0	Used to tell the other side that the connection is being closed.	Client Server
`UPDATE_USER`	1	Requests an update for the experience value for some user.	Client
`CREATE_MATCH`	1	A match is created on the database with this message with the gameModeID provided.	Client
`MATCH_ID`	1	Response by the server for the CREATE_MATCH where the matchID is sent to the client.	Server
`UPDATE_MATCH`	3	Updates the status of a match on the database. The message contains the matchID, end date and the winner team.	Client
`ADD_MATCH_PLAYER`	1	A player is added to match with the provided matchID.	Client
`UPDATE_MATCH_PLAYER`	4	Updates the personal score, eliminations and deaths of some player for the provided matchID.	Client
`REQUEST_EXP_RATE`	0	Requests the experience rate to the server to calculate the experience earned at the end of a match.	Client
`EXP_RATE`	1	Response to REQUEST_EXP_RATE with the value of the experience rate.	Server

On the following screenshot, Figure 11, the server window can be seen with some commands received (symbol "<<") and some sent (symbol ">>").

Figure 11: Screenshot of the server running.



```
** Server      - 192.168.1.10:1994
** Database    - 192.168.1.10

** Console command listener started.
** Server started!
** Client accepted: 192.168.1.13
<< LOGIN test 576F2AFCDCA7238248DD1939E21D2BF0EE6432E8
>> LOGIN_OK 36 3160

<< CREATE_MATCH 1
** Match #62 created!
>> MATCH_ID 62

<< ADD_MATCH_PLAYER 62
** Added player #36 to match #62
** Client accepted: 192.168.1.13
<< LOGIN allowed 98121F6EEF6881E88D43640A7554498C68C69F7C
>> LOGIN_OK 3 3400

<< ADD_MATCH_PLAYER 62
** Added player #3 to match #62
<< UPDATE_MATCH_PLAYER 62 0 0 1
<< UPDATE_MATCH_PLAYER 62 10 1 0
<< UPDATE_MATCH_PLAYER 62 10 1 1
<< UPDATE_MATCH_PLAYER 62 10 1 1
<< REQUEST_EXP_RATE
>> EXP_RATE 2

<< UPDATE_MATCH 62 1466548528 0
<< UPDATE_USER 3180
<< UPDATE_USER 3420
```

On the next code block, the commands that appear on the Figure 11 will be analyzed, making a trace of what happened during that execution of the server.

Code 10: Trace for the server commands of the Figure 11.

1. `LOGIN test 576F2AFCDCA7238248DD1939E21D2BF0EE6432E8`
2. `LOGIN_OK 36 3160`

Some player with the username "test" logged in correctly and the server sent a response to him with his userID (36) and the amount of experience he has (3160).

3. `CREATE_MATCH 1`
4. `MATCH_ID 62`

After that, he started to play, so a new match with the gamemode with id 1 has been created. The server responded to him with the matchID (62).

5. `ADD_MATCH_PLAYER 62`

With the match created, he joined to that match.

6. `LOGIN allowed 98121F6EEF6881E88D43640A7554498C68C69F7C`
7. `LOGIN_OK 3 3400`

Then, other player, this one with username "allowed", logged in.

8. `ADD_MATCH_PLAYER 62`

And he started to play on the same match of the previous user, "test".

9. `UPDATE_MATCH_PLAYER 62 0 0 1`
10. `UPDATE_MATCH_PLAYER 62 10 1 0`
11. `UPDATE_MATCH_PLAYER 62 10 1 1`
12. `UPDATE_MATCH_PLAYER 62 10 1 1`

The match progressed and they killed on time each one (the format of the arguments of the message is the following: matchID, points, kills, deaths).

The two first commands updated the scores on the first kill, from the player 1 to the player 2. And the second ones for the second kill, this time from the player 2 to the player 1.

13. `REQUEST_EXP_RATE`
14. `EXP_RATE 2`
15. `UPDATE_MATCH 62 1466548528 0`

The match ended and the client acting as host asked for the experience rate, and updated the status of the match with the end date and the winner team.

16. `UPDATE_USER 3180`
17. `UPDATE_USER 3420`

Finally, the experience earned on the match was awarded to each player. The first message is for "test" player (3160 + 10 * 2 exp rate = 3180) and the second for "allowed".

As seen on this trace, the userID is not provided on the messages even they are used to update some user related information. That's because the ClientHandler has it stored and on these commands is took from there to execute the database queries.

3.3.4. The database manager

The main reason of the server is to have the SQL queries and execute them instead of doing that on the clients. .Net and C# don't support natively the connection and management of PostgreSQL databases. To solve this issue, a NuGet [9] plugin called Npgsql [10] that adds this functionality.

The DatabaseManager class is a singleton with methods to update, insert and retrieve some information from the database.

Each method has a different query and gets or sets different data. The queries are not a simple commands, they are prepared statements to increase the security of the queries.

The prepared statements are parameterized queries with some special variables that are replaced with the real values during the execution. This makes them resilient to SQL injection because the parameters should coincide with the data type of the column where they will be set.

Code 11: Method to Update an user on the database. It uses a prepared statement.

```
public void UpdateUser(int userID, int exp) {
        NpgsqlCommand command = new NpgsqlCommand();
        command.CommandText = "UPDATE users SET experience = :exp WHERE user_id = :uID;";
        command.Parameters.AddWithValue(":uID", userID);
        command.Parameters.AddWithValue(":exp", exp);
        command.Connection = conn;

        conn.Open();
        command.ExecuteScalar();
        conn.Close();
}
```

3.4. Game Launcher.

3.4.1. The application

The Game Launcher is an application designed to maintain up to date the game client. It serves as an installer for the game for the first time, and for update it when a new version is available, making all clients consistent with their version.

It's created on a C# Windows Forms project. This kind of project allows the design of a visual interface and easy integration with the code. The interface consists only on a button and a progress bar. The button shows the option available at any moment (Play, Download, Update, etc), and the progress bar shows the progress made when a download is in progress.

Figure 12: Graphical design of the game launcher.

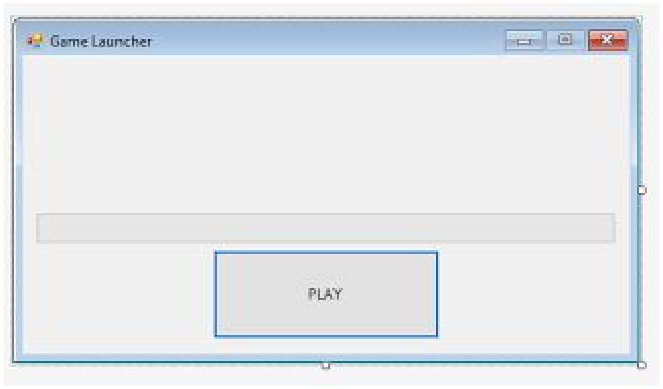
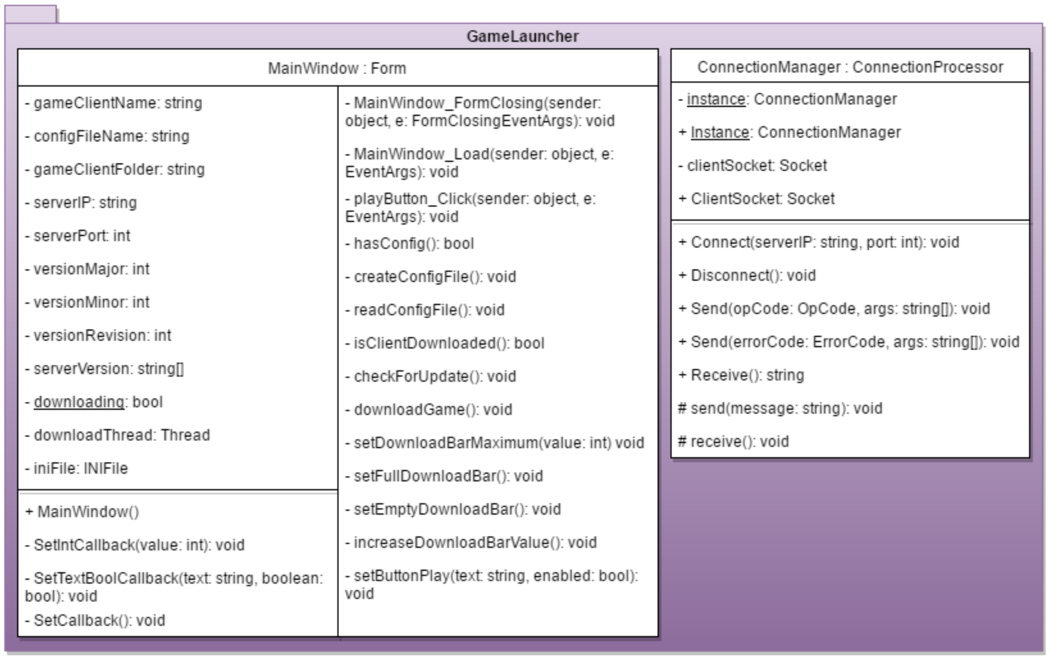


Figure 13: Class diagram of the Game launcher.


GameLauncher
MainWindow : Form
- gameClientName: string
- configFileName: string
- gameClientFolder: string
- serverIP: string
- serverPort: int
- versionMajor: int
- versionMinor: int
- versionRevision: int
- serverVersion: string[]
- downloading: bool
- downloadThread: Thread
- iniFile: INIFile
+ MainWindow()
- SetIntCallback(value: int): void
- SetTextBoolCallback(text: string, boolean: bool): void
- SetCallback(): void
- MainWindow_FormClosing(sender: object, e: FormClosingEventArgs): void
- MainWindow_Load(sender: object, e: EventArgs): void
- playButton_Click(sender: object, e: EventArgs): void
- hasConfig(): bool
- createConfigFile(): void
- readConfigFile(): void
- isClientDownloaded(): bool
- checkForUpdate(): void
- downloadGame(): void
- setDownloadBarMaximum(value: int) void
- setFullDownloadBar(): void
- setEmptyDownloadBar(): void
- increaseDownloadBarValue(): void
- setButtonPlay(text: string, enabled: bool): void
ConnectionManager : ConnectionProcessor
- instance: ConnectionManager
+ Instance: ConnectionManager
- clientSocket: Socket
+ ClientSocket: Socket
+ Connect(serverIP: string, port: int): void
+ Disconnect(): void
+ Send(opCode: OpCode, args: string[]): void
+ Send(errorCode: ErrorCode, args: string[]): void
+ Receive(): string
# send(message: string): void
# receive(): void

The application consists on two classes. The first the ConnectionManager, that derives from TFGCommonLib.Networking.ConnectionProcessor and has the methods for receive, send messages and manage the connection of the socket. And the second, the MainWindow class, initially provided by the project with conjunction of the visual interface, where is located the main logic of the application.

The application, when is launched, checks for the last version available with `checkForUpdate()` method. during the version check, it sends a message to the server with the OpCode GET_VERSION and the server responds to it with VERSION. During the version check, the button of the visual interface is disabled, and after the check, it becomes enabled again and its functionality changes depending of the result of the check:

- If the game is not downloaded.
  - Changes to "Download".
- If the game is downloaded but there is a new version.
  - Changes to "Update".
- If the current version is the most up to date.
  - Changes to "Play".

The button can have another functionality, "Check for updates", and only is available when the launcher is opened and the connection to the server can't be established.

The first two functionalities of the above list acts as the same, download the client, while the third launches the game client [11] and closes the launcher application.

Code 12: Snippet of code that launcher the game client.

```
Process.Start(AppDomain.CurrentDomain.BaseDirectory + "\\" + gameClientFolder + "\\" +
gameClientName);
Application.Exit();
```

To know which is the current version, the launcher have it stored on a .ini file with some other information about the connection.

As mentioned before, the application uses the shared library TFGCommonLib developed to share some functionality between projects. When the launcher is compiled the library files is copied into the output folder, generating two different files. As this executable (the launcher) is the one that will be on the webpage to download, makes no sense that the download contains two different files.

To solve this issue I found a NuGet plugin called Costura.Fody [12] that allows to embed any reference library on the final executable, so compiling the project with that results on a single executable file.

3.4.2. The download process

The download process have some special OpCodes designed specially for it, and are used in a specific way. On the following table, all OpCodes used by the Game Launcher are listed, notice that 5 of the 7 are for the download process.

Table 9: OpCodes used by the Game Launcher.

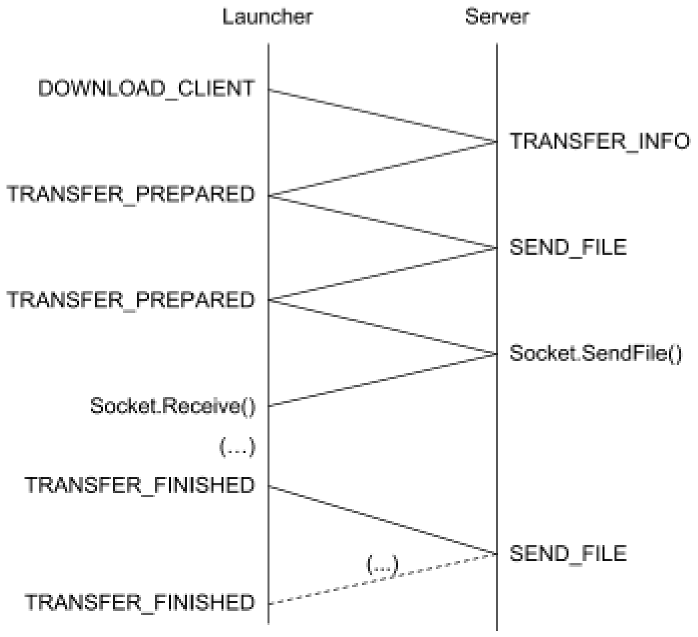
OpCode	**Num. Arg.**	**Explanation**	**Sender**
GET_VERSION	0	Message sent by as a request to obtain the version number of the last version of the game.	Launcher
VERSION	3	As a response for GET_VERSION, the server sends the major, minor and revision number of the last version of the game.	Server
DOWNLOAD_CLIENT	0	Requests a download for the game client.	Launcher
TRANSFER_INFO	1	As a response to DOWNLOAD_CLIENT, the server sends the number of files to be sent.	Server
TRANSFER_PREPARED	0	Tells to the server that the transfer of a file is prepared.	Launcher
TRANSFER_FINISHED	0	Tells to the server that the transfer of a file is finished.	Launcher

`SEND_FILE`	2	Sends the information about the file is going to be sent. This includes the file path and the size.	Server

Now that we know which OpCodes are used and their meaning, the process will be explained.

The first thing that the launcher does to start the download is to create a new thread that will manage the entire process in the background, without locking the main thread that manages the visual representation of the application.

Figure 14: Diagram of the download progress.



On the Figure 14 there is a diagram of how the download process works. The entire part of the download progress on the server side was not explained on the server section, and will be explained here to understand better what's happening at every moment, so it's recommended to check the Figure 9, the class diagram of the server.

The first thing the Launcher does is request the client download with the OpCode DOWNLOAD_CLIENT. When the server receives the message, it starts the preparation of the download with the `ClientHandler.prepareDownload()` method. Here, it creates a list of files to be sent and the directory tree by executing the method `ClientHandler.traverseTree(rootFolder: string)` [13]. With the list of files and folders created, the server sends to the Launcher the number of files to be transferred with the TRANSFER_INFO OpCode.

The Launcher reads the parameter of the message, stores this value and sets the progress bar maximum value. Changing values from the Windows Form items from a different thread than the main is not allowed, so a delegate callback functions with and Invoke were used as seen on the code block Code 13.

Code 13: Changing windows forms values form a thread.

```
delegate void SetIntCallback(int value);

private void setDownloadBarMaximum(int value) {
        if (downloadBar.InvokeRequired) {
                SetIntCallback d = new SetIntCallback(setDownloadBarMaximum);
                Invoke(d, value);
        }
        else {
                downloadBar.Maximum = value;
        }
}
```

After this, the Launcher creates the new directory for the new version, and moves the old one to another folder with the suffix ".old", and then sends a TRANSFER_PREPARED message to the server.

At this moment, the transfer process for all files starts. The server sends a message with the OpCode SEND_FILE with the path with the name and size of the file.

The launcher receives the message, and creates the directory tree needed to download the file on the same directory as the original.

Code 14: Creation of the directory tree for the current file.

```
string file = fileParams[0].Replace("\\", "/").Replace('*', ' ');
int size = int.Parse(fileParams[1]);

string[] folders = file.Split('/');
string path = "";
for (int i = 0; i < folders.Length - 1; i++) {
        path += folders[i] + "/";
        if (!Directory.Exists(path))
                Directory.CreateDirectory(path);
}
```

The server receives the message and starts the sending of the file using the method `Socket.SendFile(filePath)`. After the last message sent by the launcher, it starts to receive the data of the file, but for that it uses the regular `Socket.Receive()` method using a buffer.

The buffer has a size of 1024 bytes, but not always 1024 bytes are received on each `Socket.Receive()` call. After receiving the data, a new buffer is created with size of the bytes received, and they are written into the file, until the local size is the same as the sent by the server. This is done in that way to avoid writing empty bytes and create a corrupt file.

Code 15: Downloading the file on the Launcher.

```
FileStream fs = File.Create(file, 1024);

byte[] buffer;
//size is the original file size and fs.Length is the current size of the file being written
while (fs.Length < size) {
       if (size - fs.Length >= 1024)
              buffer = new byte[1024];
       else
              buffer = new byte[size - fs.Length]; //If the remaining bytes to be received
are less than 1040, the buffer is adjusted to that size.

       //bytesRead is the number of bytes read on the current call of Receive. It may be
less than 1024.
       int bytesRead = ConnectionManager.Instance.ClientSocket.Receive(buffer,
buffer.Length, System.Net.Sockets.SocketFlags.None);

       //A new buffer is created with size of the number of received bytes.
       byte[] realBuffer = new byte[bytesRead];
       for (int i = 0; i < bytesRead; i++) {
              realBuffer[i] = buffer[i];
       }

       //The received data is written on the file.
       fs.Write(realBuffer, 0, bytesRead);
}

fs.Close();
```

When the receiving of the file is completed, the launcher sends to the server a message with the OpCode TRANSFER_FINISHED. The process is repeated starting from the SEND_FILE OpCode until all files are transferred.

In that moment, the launcher updates the version numbers stored on the ini file, and removes the old version of the client.

If the download is interrupted by half, the launcher restores deletes the progress of the download and restores the new version without changing the version number stored.

3.5. Management webpage

3.5.1. HTML and Javascript

The web page contains the basic tools for manage the account of the users, and create a new ones.The functionalities included on the webpage are the following:

- If you are not logged in:
  - Register: create a new accounts.
  - Login: login to the management.

    - Recover password: a new random generated password is created and sent to the email (the email should be registered).
- If you are logged in:
    - Account: some information about the account.
    - Download game: downloads the game launcher.
    - Change Password: changes the password to a new one.
    - Change Email: changes the email to a new one.
    - Log out: destroys the session and logs out.

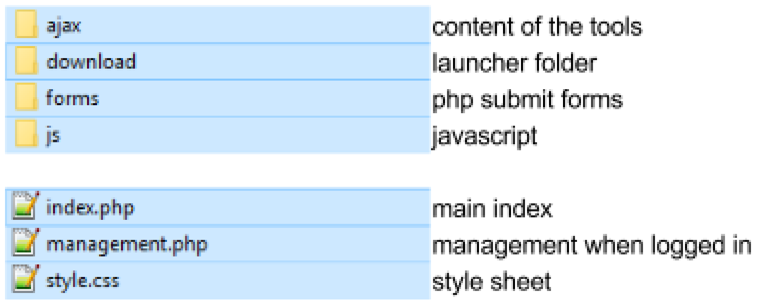
The visual part is formed by two simple HTML pages with some css styling applied to it to show it prettier than a plain text page.

The HTML pages are basically the same. They have some div that creates some tabs to select the tool, and other container div, where the content of the tools is loaded dynamically using a javascript library called jQuery. The Figure 16 shows this.

When the user is logged out, the main index is loaded by the browser, if is logged in, the page redirects to the management page.

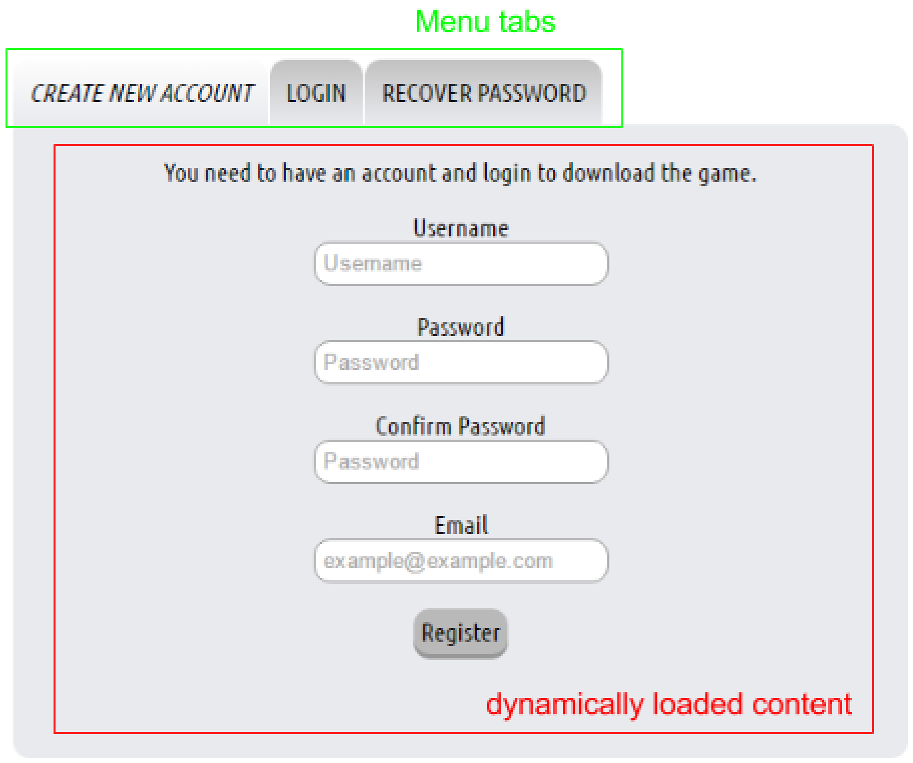
On the Figure 15, the files and folders of the webpage are shown. The main index is index.php and the management.php is the page where the user is redirected if is logged in.

Figure 15: Image of the files and folders of the root of the webpage.



ajax	content of the tools
download	launcher folder
forms	php submit forms
js	javascript
index.php	main index
management.php	management when logged in
style.css	style sheet

Figure 16: Image of the index.html with some content loaded.



When one of the tabs is pressed, the selected one changes to the pressed, and the content of that tool is loaded into the container without reloading the page using jQuery methods.

The ajax folder (see Figure 15) contains the files with the content. These files are the forms and information that is shown on the browser.

Code 16: Function that changes the selected tab and loads the content.

```
function SelectTab(clickedID) {
        $("#tabs").find(".selected").removeClass("selected");
        $("#" + clickedID).addClass("selected");

        if(clickedID == "registerTab") {
                $( "#dynamicContainer" ).load( "ajax/register.html" );
        }
        //[...]
        else if(clickedID == "logoutTab") {
                $( "#dynamicContainer" ).load( "ajax/logout.php" );
        }
}
```

The formularies, like the registration or log in, are sent without using the default behaviour of the forms. They don't reload the page to send the data, instead, a jQuery call using the post method sends it, obtaining the results in the same call.

Code 17: Post code for the change email form.

```
$.post(
       "forms/changeEmailForm.php", //post to this file
       { newEmail: params[0].value, //with the values newEmail and pass
       pass: params[2].value},
       function(response) { //the result of the execution is obtained here.
              if(response!="success")
                     $("#formErrors").html(response); //show errors
              else {
                     window.location.href = "/"; //in this case, if success returns to the
root
              }
       }
);
```

When some formulary is sent using jQuery post, uses a file of the folder "forms" (Figure 17). All the files of this folder are php files, and contains the logic executed on the server to perform actions like execute sql queries or create and destroy sessions..

Finally, a download section is present on the management page when logged in. On this section the Game Launcher can be downloaded using an extension of jQuery called jquery.fileDownload [14].

3.5.2. PHP and the database

PHP is only executed on the server hosting the webpage, so is safe to place sql queries, or other sensible code on it. If someone tries to download a file with php code, it will get a plain text file with the results of executing that code.

As explained on the last section, the formularies are sent using the post method with a php file of the "forms" folder.

The first thing done on these files is to check if the post variables exist, if then the necessary operations to perform the current actions are executed, like start the php session [15] or execute some queries on the database.

Code 18: PHP code of forms/changeEmailForm.php

```
<?
if(isset($_POST['newEmail']) && isset($_POST['pass'])) {

       include ("database.php");
       Database::Connect();

       //start php session
       session_start();
```

```
		//Check if the current password is correct
		if(Database::CheckPasswordMatch($_SESSION['user'], $_POST['pass'])) {

			//Send an email to the current direction to notify the change
			include ("mailer.php");

			Mailer::SendMail(/*[...]*/);

			//Update the email
			Database::UpdateEmail($_SESSION['user'], $_POST['newEmail']);

			//And send an email to the new direction to notify the change
			Mailer::SendMail(/*[...]*/);

			//the session is destroyed and the user is logged out (for safety reasons).
			session_destroy();

			//And returns a success
			echo "success";
		}
		else { //Errors found
			echo "The password is not correct.";
		}
}
else //the post variables are not set, return an ERROR message
		echo "ERROR";
?>
```

All the files of the "forms" folder have the same structure, seen on the Code X. To create a connection with the database and execute queries, a class Database was created on the file "database.php". Each time this class is used, it should be included to the current php code with "`include <filename>`".

That class has some static functions with the sql in prepared statements to perform the actions needed on the database.

Code 19: UpdateEmail function of the PHP Database class.

```
public static function UpdateEmail($username, $newEmail) {
		$sql = 'UPDATE users SET email = lower($2) WHERE lower(username) = lower($1)';
		$sqlName = 'updateEmail';
		if (!pg_prepare ($sqlName, $sql))
			die("Can't prepare '$sql': " . pg_last_error());
		pg_execute(self::$conn, $sqlName, array($username, $newEmail));
}
```

In addition to that class, another one was created, this time to send emails when a notification for the user is needed (for example, when changing the password an email is sent notifying the change). This class is the Mailer seen on the Code 19.

The Mailer class uses an external php plugin called PHPMailer [16] to send emails with a gmail account using php.

Code 20: Mailer class using PHPMailer.

```
include ("PHPMailer/class.phpmailer.php");
include ("PHPMailer/class.smtp.php");
class Mailer {
	private static $sender="email@gmail.com";
	private static $senderName="TFG Project";
	private static $password="password";

	public static function SendMail($receiver, $receiverName, $subject, $altBody,
$messageHTML) {
		$mail = new PHPMailer(true);
		$mail->IsSMTP();
		try {
			$mail->SMTPAuth   = true; //enables SMTP authentication
			$mail->SMTPSecure = "ssl"; //sets the SMTP security type
			$mail->Host       = "smtp.gmail.com"; //sets Gmail as SMTP server
			$mail->Port       = 465; //sets the port of the Gmail SMTP server
			$mail->Username   = self::$sender;   // gmail user
			$mail->Password   = self::$password; // gmail password

			//to which address can answered the mail
			$mail->AddReplyTo(self::$sender, self::$senderName);
			//send the email to this address
			$mail->AddAddress($receiver, $receiverName);
			//sender name
			$mail->SetFrom(self::$sender, self::$senderName);
			//mail subject
			$mail->Subject = $subject;
			//alternate message in case that the  receiver can't open HTML emails
			$mail->AltBody = $altBody;
			//the main message, with HTML
			$mail->MsgHTML($messageHTML);
			//send the email
			$mail->Send();
		} catch (phpmailerException $e) {
			echo $e->errorMessage();
		} catch (Exception $e) {
			echo $e->getMessage();
		}
	}
}
```

3.6. Game Client, Videogame demo

The video game demo, the game client, is developed in Unity. It consists of a multiplayer game where the players fight on an auto generated maze.

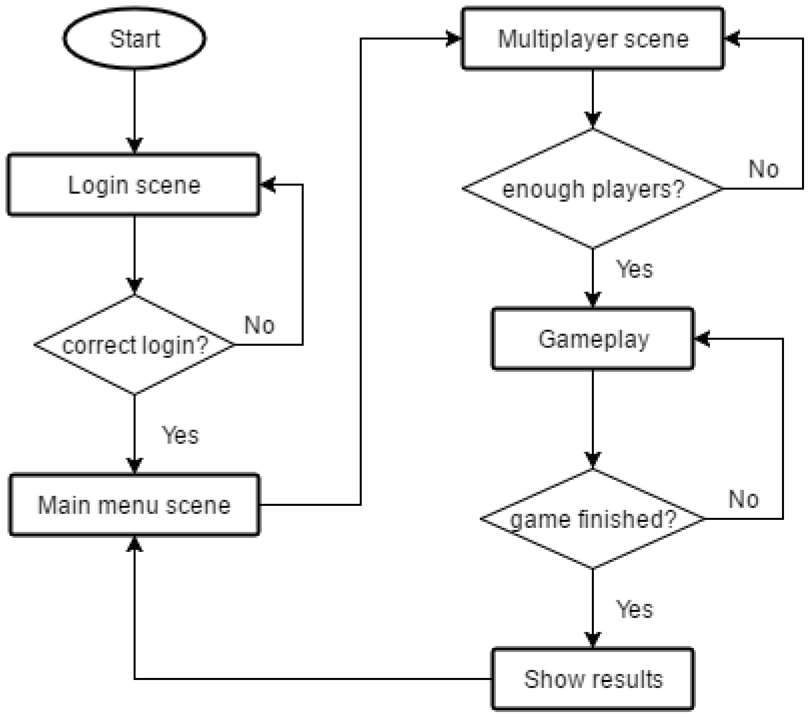
3.6.1. Unity scenes and game demo flow

The gage have three scenes. Two of them are for the menus and the third one for the gameplay:

- Login: scene where the player logs in using a registered account.
- MainMenu: scene where the player can see his profile (level and experience) in addition to the Matchmaker menu to start playing online.
- Multiplayer: scene where the gameplay occurs.

The details of each scene will be explained in the following sections, but now, we are focussing on the game flow and how the game progresses through the scenes.

Figure 17: Game flow.



The game starts on the Login scene, where the player should enter the credentials of his account to login. If the login is not correct or some error happens, an error message will appear.

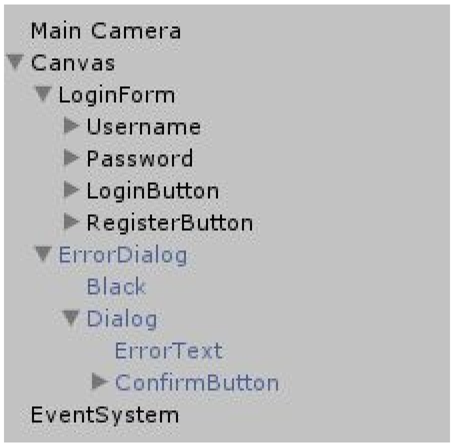
If the player logs in correctly, the Main menu scene is loaded. The player here can see the amount of experience he have and the level, in addition to select to start playing.

When the player decides to play, in the middle of the Main menu scene and the Multiplayer scene, the MatchMaker makes his appearance. The functionality of the MatchMaker will be explained better in the next section.

Then, the Multiplayer scene is loaded. If there are enough players to start a match, the gameplay starts, if not, it waits until that condition is met. After finishing the game, a results will appear on the screen and a button to return to the main menu.

3.6.2. Login scene

Figure 18: Login scene Hierarchy.

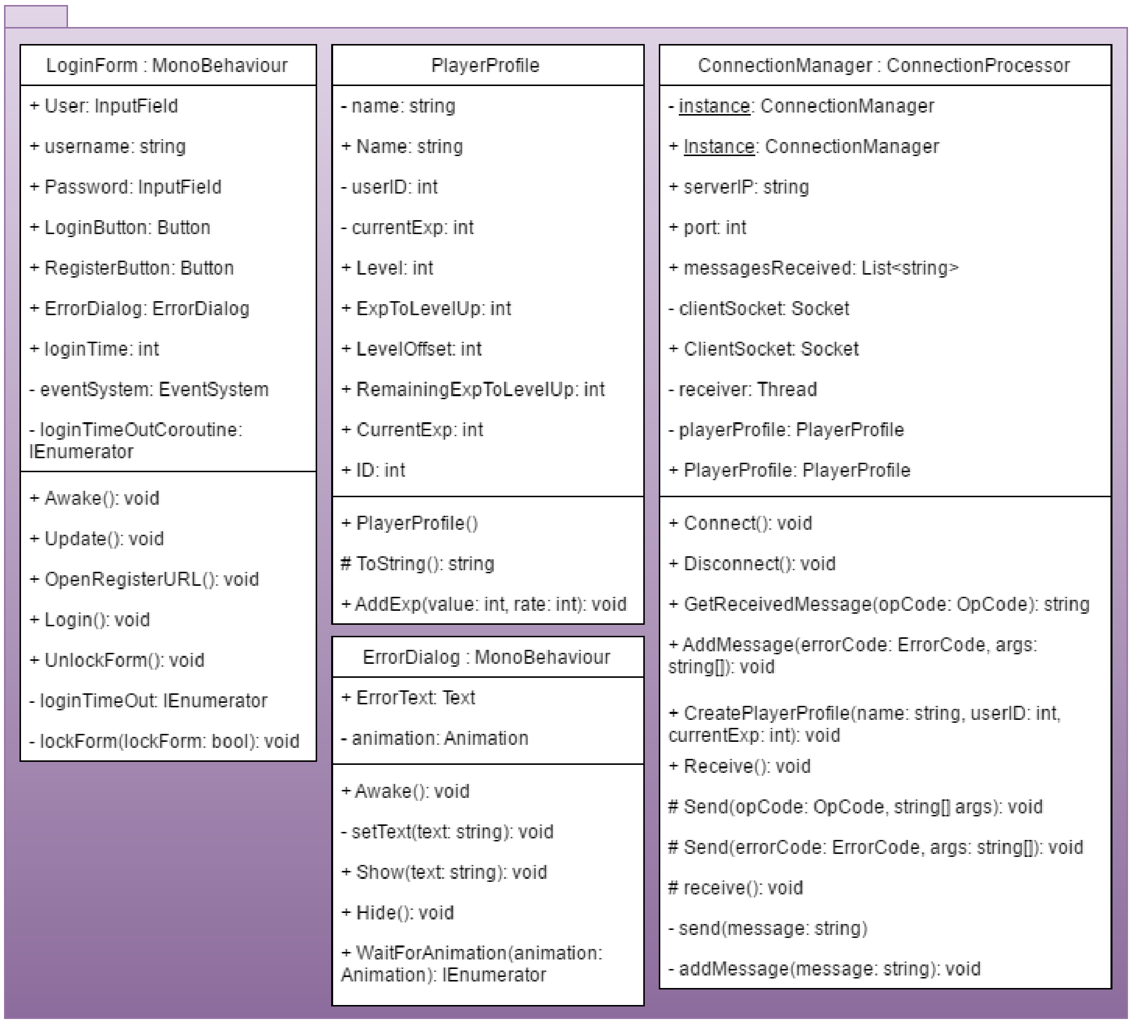


```
Main Camera
▼Canvas
  ▼LoginForm
    ▶Username
    ▶Password
    ▶LoginButton
    ▶RegisterButton
  ▼ErrorDialog
     Black
    ▼Dialog
       ErrorText
      ▶ConfirmButton
EventSystem
```

As seen on the last section, the game starts on the Login scene. Here the player should login with a registered account. The scene is composed mainly by a Unity Canvas object, that has all the fields of the login formulary, in addition to the Error Dialog that appears in case of any problem occurred during the login. On the Figure 18 the hierarchy of this scene can be seen.

Four different scripts take part on this scene. Their class diagrams can be seen on the Figure 19.

Figure 19: Login scene class diagram.


LoginForm : MonoBehaviour
+ User: InputField
+ username: string
+ Password: InputField
+ LoginButton: Button
+ RegisterButton: Button
+ ErrorDialog: ErrorDialog
+ loginTime: int
- eventSystem: EventSystem
- loginTimeOutCoroutine: IEnumerator
+ Awake(): void
+ Update(): void
+ OpenRegisterURL(): void
+ Login(): void
+ UnlockForm(): void
- loginTimeOut: IEnumerator
- lockForm(lockForm: bool): void
PlayerProfile
- name: string
+ Name: string
- userID: int
- currentExp: int
+ Level: int
+ ExpToLevelUp: int
+ LevelOffset: int
+ RemainingExpToLevelUp: int
+ CurrentExp: int
+ ID: int
+ PlayerProfile()
# ToString(): string
+ AddExp(value: int, rate: int): void
ErrorDialog : MonoBehaviour
+ ErrorText: Text
- animation: Animation
+ Awake(): void
- setText(text: string): void
+ Show(text: string): void
+ Hide(): void
+ WaitForAnimation(animation: Animation): IEnumerator
ConnectionManager : ConnectionProcessor
- instance: ConnectionManager
+ Instance: ConnectionManager
+ serverIP: string
+ port: int
+ messagesReceived: List<string>
- clientSocket: Socket
+ ClientSocket: Socket
- receiver: Thread
- playerProfile: PlayerProfile
+ PlayerProfile: PlayerProfile
+ Connect(): void
+ Disconnect(): void
+ GetReceivedMessage(opCode: OpCode): string
+ AddMessage(errorCode: ErrorCode, args: string[]): void
+ CreatePlayerProfile(name: string, userID: int, currentExp: int): void
+ Receive(): void
# Send(opCode: OpCode, string[] args): void
# Send(errorCode: ErrorCode, args: string[]): void
# receive(): void
- send(message: string)
- addMessage(message: string): void

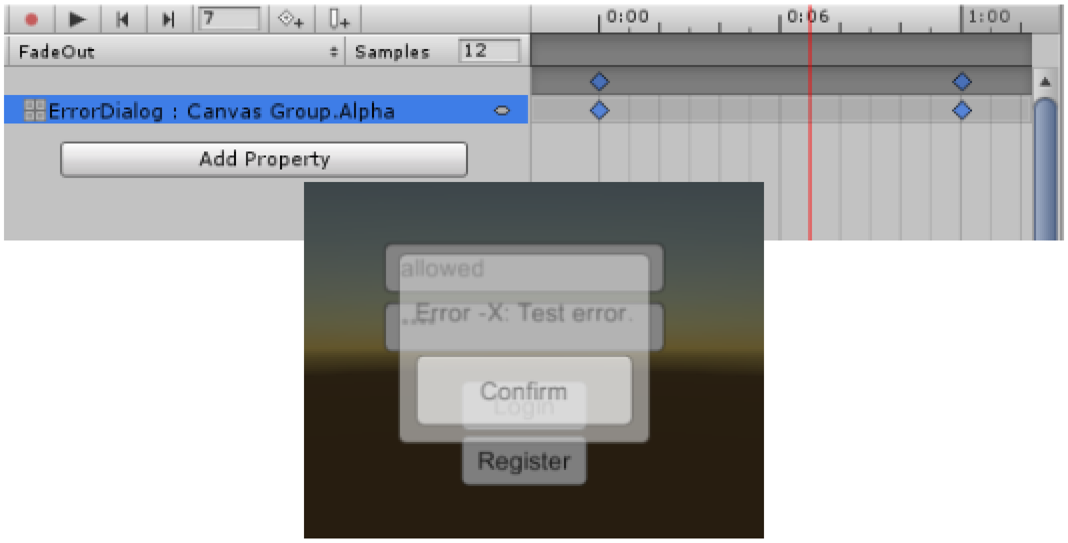
The most important classes here, are the ConnectionManager and the LoginForm. The ConnectionManager as the name indicates, manages the connection between the server and the client. It implements the abstract class ConnectionProcessor of TFGCommonLib to obtain the necessary methods to receive and send data.

This class operates in conjunction with the LoginForm. When the login button is pressed, LoginForm uses the ConnectionManager to send the login information from the text fields. If some error is produced, like wrong login credentials, or no connection with the server, an ErrorCode is auto sent by the ConnectionManager to himself and the corresponding error message appears on a ErrorDialog on the screen.

The login formulary has a timeout of 10 seconds to receive the response from the server after trying to log in, if in that time no response is received, it sends a LOGIN_TIME_OUT and the message appears on the screen.

The LoginForm is attached to the GameObject with name "LoginForm". The mentioned ErrorDialog, one of the four scripts used on the login, is attached to the GameObject with name "ErrorDialog" and appears and disappear on the screen with a fade animation.

Figure 20: FadeOut animation for the ErrorDialog.



The user can move between the fields of the formulary with the same behaviour of the formulary of the web pages by pressing the Tabulator key and send it by pressing the Enter key. This is a normal behaviour that a user can expect, but Unity doesn't provides it by default.

Code 21: Behaviour of the formulary with Tab and Enter keys.

```
if (Input.GetKeyDown(KeyCode.Tab)) {
        Selectable next =
eventSystem.currentSelectedGameObject.GetComponent<Selectable>().FindSelectableOnDown();
        if (next != null) {
                InputField inputfield = next.GetComponent<InputField>();

                if (inputfield != null)
                        inputfield.OnPointerClick(new PointerEventData(eventSystem));

                eventSystem.SetSelectedGameObject(next.gameObject, new
BaseEventData(eventSystem));
        }
}
else if (Input.GetKeyDown(KeyCode.KeypadEnter) || Input.GetKeyDown(KeyCode.Return)) {
        LoginButton.Select();
        Login();
}
```

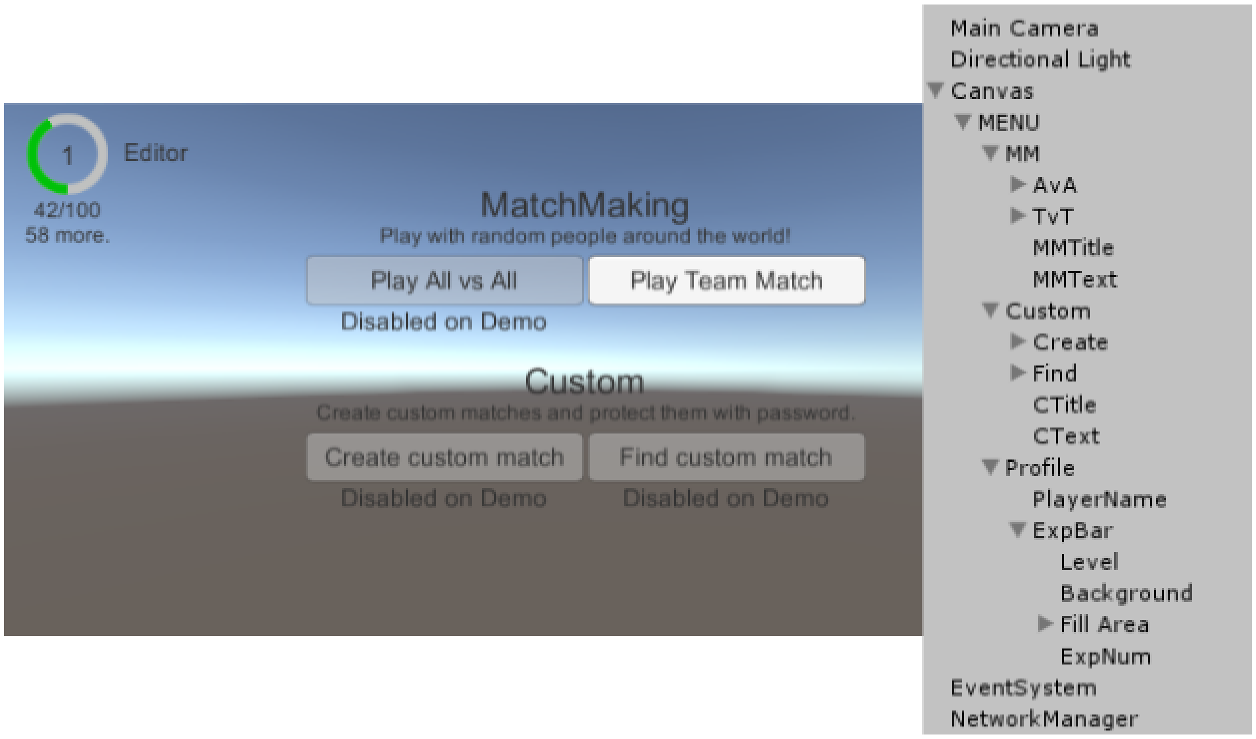
Finally, the PlayerProfile class, manages the information of the player profile, like the amount of experience, level and user name. The calculation of the level is based on a "linearly rising level gap" formula [17]:

Code 22. Experience formula.

```
Level         = (sqrt(100*(2*experience+25))+50)/100
Experience    = (level^2+level)/2*100-(level*100)

User with 120 experience points:
Level = (sqrt(100*(2*120+25))+50)/100 = 2.12 ~> Level 2

Level 1:   0 -  99   //Needs 100 points to level up
Level 2: 100 - 299   //Needs 200 points to level up
Level 3: 300 - 599   //Needs 300 points to level up
```

3.6.3. Main menu scene and MatchMaker

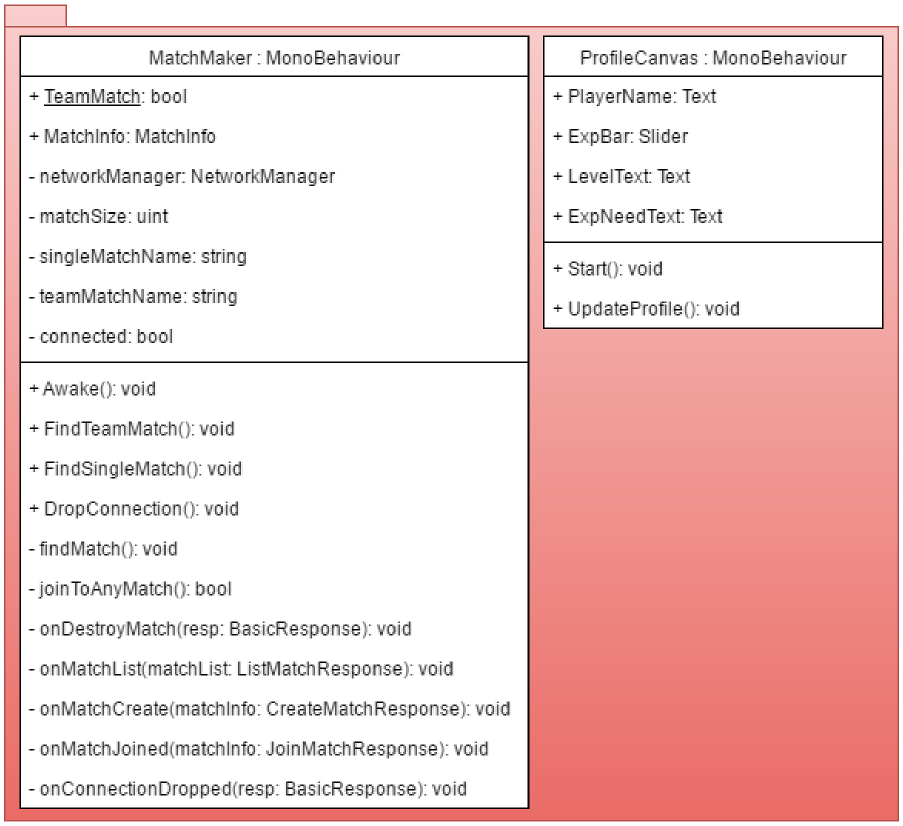
Figure 21: Main menu scene hierarchy and scene.



As explained on the 3.6.1 section, the main menu is the scene where the player can see his experience, level, and start to play on multiplayer matches.

The scene is mainly composed of a canvas with all of the elements on the screen. The player profile (at the top left of the screen), is managed by the class ProfileCanvas, that takes the data from the PlayerProfile explained on the last section and shows it on the screen in a graphical way.

Figure 22: MainMenu scene class diagram.



MatchMaker : MonoBehaviour

+ TeamMatch: bool
+ MatchInfo: MatchInfo
- networkManager: NetworkManager
- matchSize: uint
- singleMatchName: string
- teamMatchName: string
- connected: bool

+ Awake(): void
+ FindTeamMatch(): void
+ FindSingleMatch(): void
+ DropConnection(): void
- findMatch(): void
- joinToAnyMatch(): bool
- onDestroyMatch(resp: BasicResponse): void
- onMatchList(matchList: ListMatchResponse): void
- onMatchCreate(matchInfo: CreateMatchResponse): void
- onMatchJoined(matchInfo: JoinMatchResponse): void
- onConnectionDropped(resp: BasicResponse): void

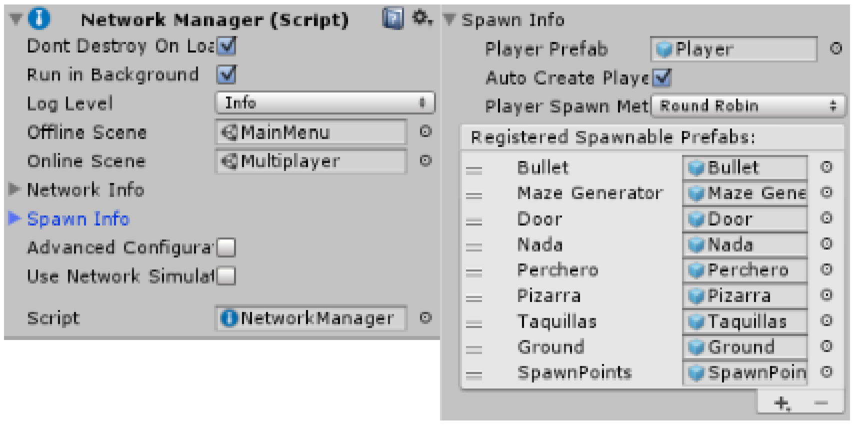
ProfileCanvas : MonoBehaviour

+ PlayerName: Text
+ ExpBar: Slider
+ LevelText: Text
+ ExpNeedText: Text

+ Start(): void
+ UpdateProfile(): void

One of the most important things on this scene, is the "NetworkManager" GameObject (the last one of the hierarchy on the Figure 21). This GameObject has the component with the same name, NetworkManager [18], and manager everything the necessary on the online games.

Figure 23: NetworkManager component.



Network Manager (Script)
Dont Destroy On Loa ✓
Run in Background ✓
Log Level: Info
Offline Scene: MainMenu
Online Scene: Multiplayer
Network Info
Spawn Info
Advanced Configura ☐
Use Network Simulat ☐
Script: NetworkManager

Spawn Info
Player Prefab: Player
Auto Create Playe ✓
Player Spawn Met: Round Robin
Registered Spawnable Prefabs:

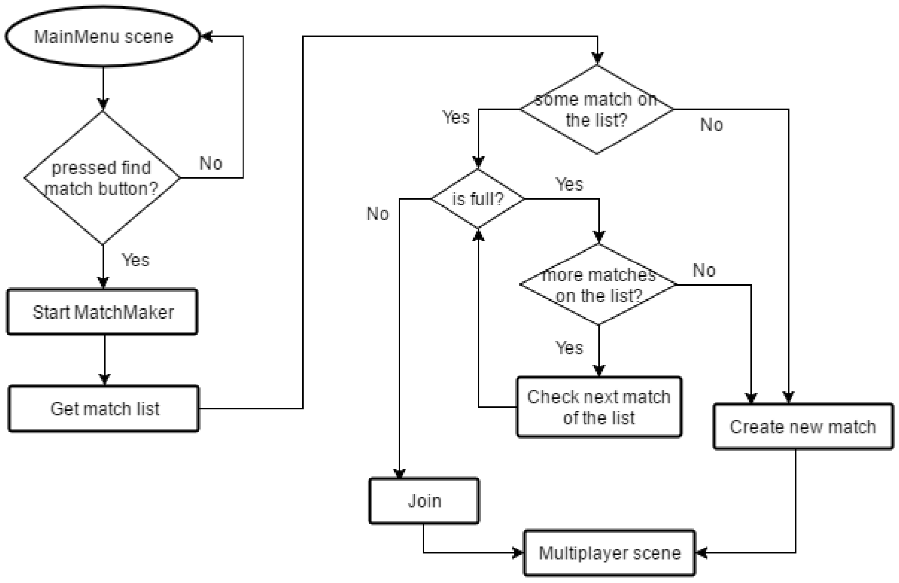
Bullet	Bullet
Maze Generator	Maze Gene
Door	Door
Nada	Nada
Perchero	Perchero
Pizarra	Pizarra
Taquillas	Taquillas
Ground	Ground
SpawnPoints	SpawnPoin

The NetworkManager allows you to select the scenes where the player will be redirected when the game starts or when they lose the connection. In this case (see Figure 23 above) these scenes are "MainMenu" for the offline scene, and "Multiplayer" for the online scene, where the gameplay will be. It has too a list of networked prefabs (they should need a component called NetworkIdentity on them. This will be explained on the next section) registered that will be spawned on some moment of the gameplay on a dynamic way from the code. For example, the player prefab can be registered here and the own NetworkManager will spawn it at the start automatically if it found some NetworkStartPosition [19] objects on the online scene.

In addition to these functionalities, the NetworkManager also offers access to the MatchMaker [20][21][22]. The MatchMaker allows the creation of multiplayer matches using the Unity servers without the need to use a public IP.

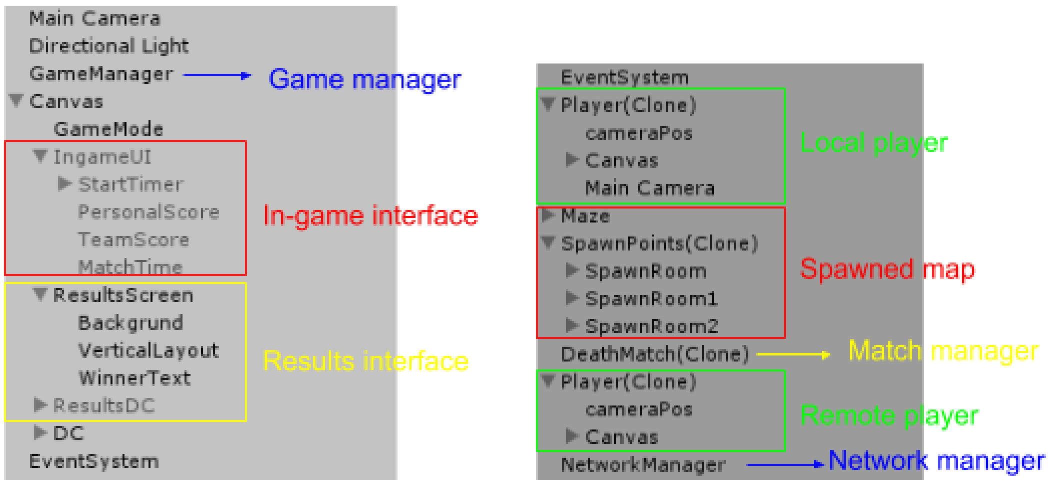
To make the matchmaker create and join to matches automatically like the most games, I created a MatchMaker script that uses the NetworkManager MatchMaker's methods to do it. The workflow of this script is described on the Figure 24.

Figure 24: Workflow of the MatchMaker script.



3.6.4. Multiplayer scene

Figure 25: On the left, the original hierarchy of the scene. On the right, the extra objects of the hierarchy when a match is running.

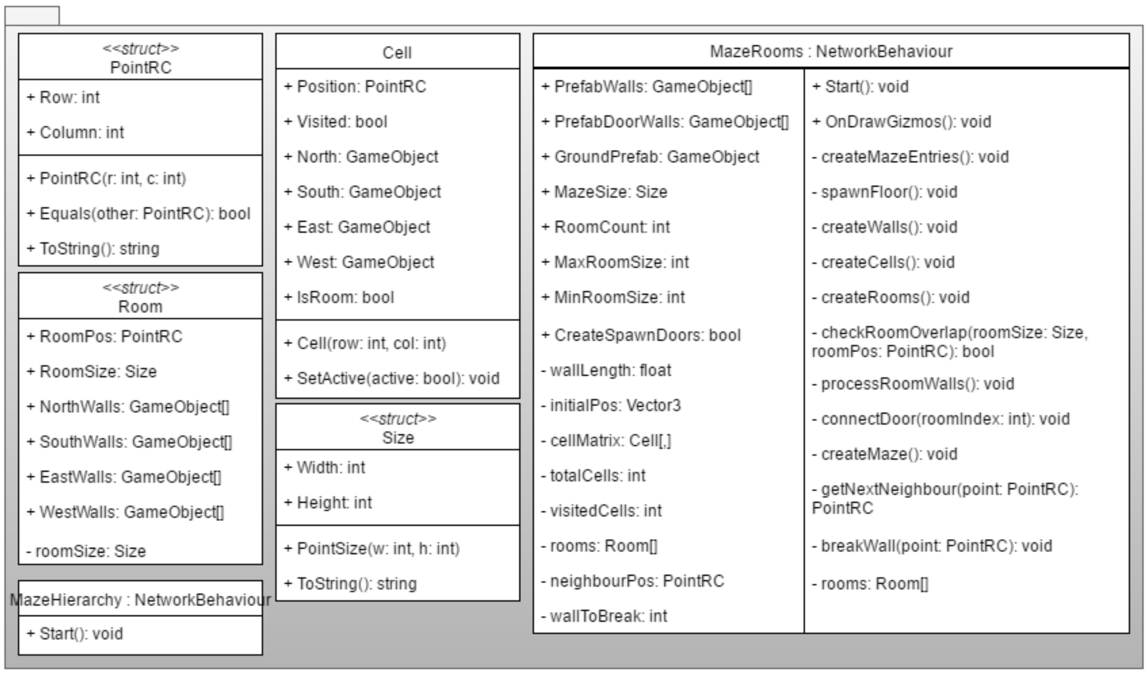


This last scene, the Multiplayer scene, is where the gameplay occurs. The scene, when the game is not execute, is almost empty of GameObjects as seen on the Figure 25. It only contains the interfaces necessaries and the GameManager. There is no map, no players, and no objectives to do.

This is because everything is created dynamically from code when a networked match is present.

If we see, the Figure 25 again, on the right side there are the objects created when a match is running, and the inherited NetworkManager from the previous scene (exactly the same object that was explained on the last section). We will start explaining the map and how is created.

Figure 26: Maze Generator class diagram.



The map is spawned on the GameManager. It is designed to change depending on the GameMode that the match is, but in the demo developed it only contains the Deathmatch mode, so the map is always the same, concretely, a maze generator and some spawn rooms outside the map.

We need to know before continue that when a networked match is played, some of the players is acting as a host and client at the same time, and the others only as client. That mean that some code can be executed only on the host, and send to the clients. This is how the map is spawned.

The map is spawned on the host when the match is created. Every part of the map is created locally using the Instantiate method of Unity, but after that a network spawn should be done. This network spawn sends the object to the other clients, creating a copy of it on their scenes. On this case, the map is the maze generator implemented as explained on the section 2.5 of the Chapter 2, but is only executed (partially) on the host. The methods that script has are restricted to be executed only on the host with the attribute "`[Server]`".

Code 23: Part of a method that is executed only on the host to create the walls of the maze.

```
[Server]
private void CreateWalls() {
        //[...]
        for (int i = 0; i < mazeSize.row; i++) {
                for (int j = 0; j <= mazeSize.col; j++) {

                //[...]

                wallWorldPosition = new Vector3(initialPos.x + (j * wallLength) - wallLength
/ 2, initialPos.y, initialPos.z + (i * wallLength) - wallLength / 2);
                tempWall = Instantiate(prefabWalls[rand], wallWorldPosition,
Quaternion.identity) as GameObject;
                tempWall.transform.parent = transform;
                NetworkServer.Spawn(tempWall);
                }
        }
         //[...]
}
```

This networked spawn can't be done on any GameObject. They should have the component NetworkIdentity [23] on them. This component makes the object a network identity and makes the network system aware of it.

The NetworkIdentity component allows to set the authority of the object. If the authority is set as Server only, that object will be spawned only on the server, and not on the clients, while if is set to local authority it can be controlled by the client that owns it.

The objects spawned with a network spawn are placed on the origin of the scene, something probably not desired. To solve this, Unity have a component called NetworkTransform [24] that allows to synchronize with the other clients its transform (position, rotation and scale), RigidBody or character controller. For the map, the NetworkTransform component is set to synchronize only the transform.

Figure 27: The map spawned seen from the Unity editor.

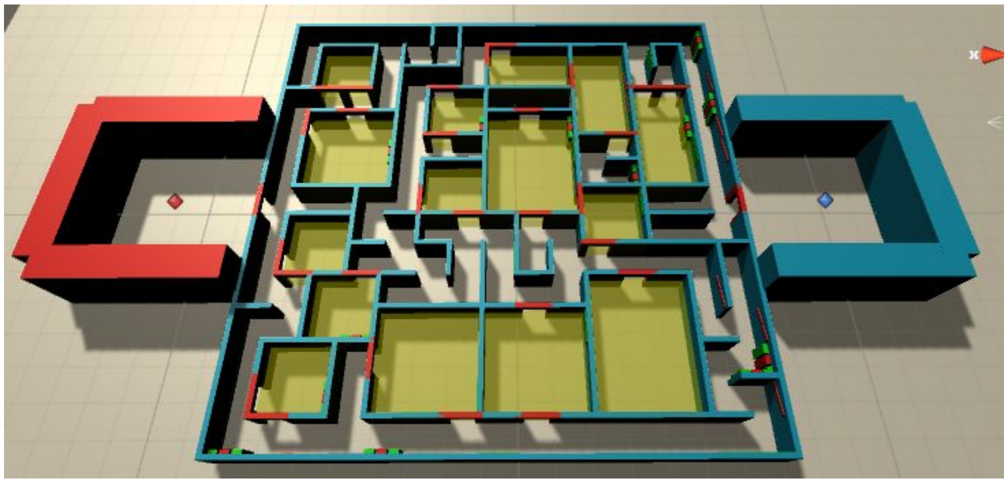
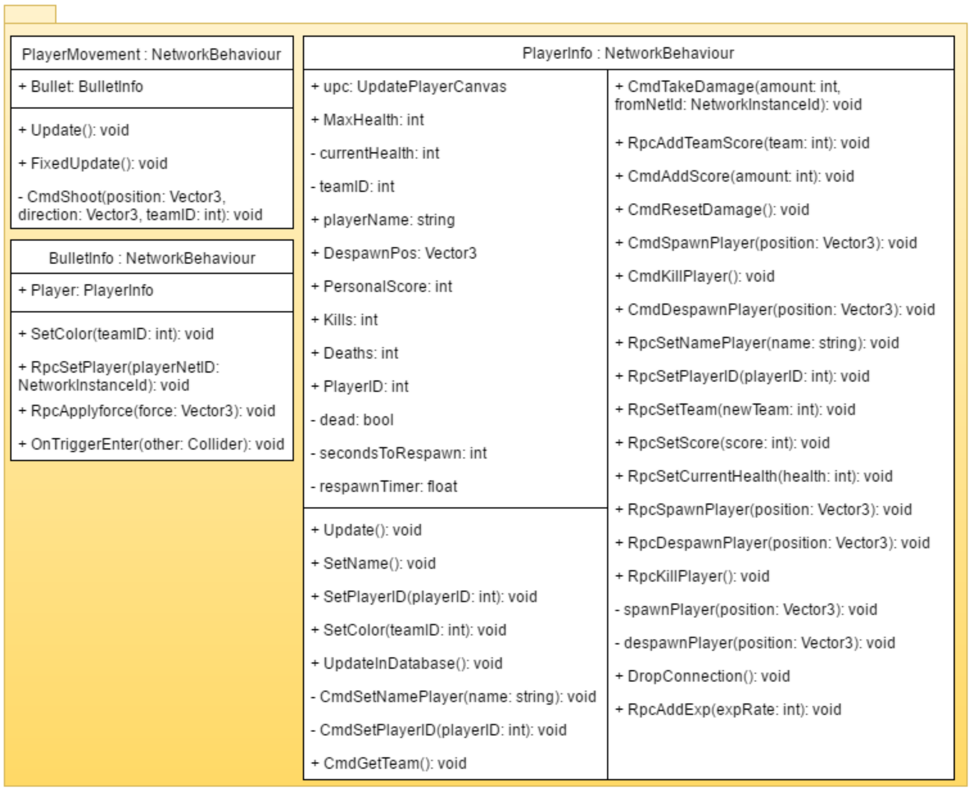


Figure 28: Player class diagram.


PlayerMovement : NetworkBehaviour
+ Bullet: BulletInfo
+ Update(): void
+ FixedUpdate(): void
- CmdShoot(position: Vector3, direction: Vector3, teamID: int): void
BulletInfo : NetworkBehaviour
+ Player: PlayerInfo
+ SetColor(teamID: int): void
+ RpcSetPlayer(playerNetID: NetworkInstanceId): void
+ RpcApplyforce(force: Vector3): void
+ OnTriggerEnter(other: Collider): void
PlayerInfo : NetworkBehaviour
+ upc: UpdatePlayerCanvas
+ MaxHealth: int
- currentHealth: int
- teamID: int
+ playerName: string
+ DespawnPos: Vector3
+ PersonalScore: int
+ Kills: int
+ Deaths: int
+ PlayerID: int
- dead: bool
- secondsToRespawn: int
- respawnTimer: float
+ Update(): void
+ SetName(): void
+ SetPlayerID(playerID: int): void
+ SetColor(teamID: int): void
+ UpdateInDatabase(): void
- CmdSetNamePlayer(name: string): void
- CmdSetPlayerID(playerID: int): void
+ CmdGetTeam(): void
+ CmdTakeDamage(amount: int, fromNetId: NetworkInstanceId): void
+ RpcAddTeamScore(team: int): void
+ CmdAddScore(amount: int): void
+ CmdResetDamage(): void
+ CmdSpawnPlayer(position: Vector3): void
+ CmdKillPlayer(): void
+ CmdDespawnPlayer(position: Vector3): void
+ RpcSetNamePlayer(name: string): void
+ RpcSetPlayerID(playerID: int): void
+ RpcSetTeam(newTeam: int): void
+ RpcSetScore(score: int): void
+ RpcSetCurrentHealth(health: int): void
+ RpcSpawnPlayer(position: Vector3): void
+ RpcDespawnPlayer(position: Vector3): void
+ RpcKillPlayer(): void
- spawnPlayer(position: Vector3): void
- despawnPlayer(position: Vector3): void
+ DropConnection(): void
+ RpcAddExp(expRate: int): void

The Player uses the mentioned components for the movement managed by the PlayerMovement script, but that's not enough if the movement is not controlled to be only on the local player. For that, exists an attribute of the NetworkBehaviour that allows to know which of all spawned players is the one that corresponds to the current client. That attribute is "`isLocalPlayer`". Adding that attribute as condition restricts the execution of the movement to be only on the local player. Without that, all spawned players can be moved by any client, something not desirable for a multiplayer game.

Code 24: Fixed update of PlayerMovement

```
void FixedUpdate() {
        if (isLocalPlayer) {
                transform.Translate(Vector3.forward * Input.GetAxis("Vertical") *
Time.fixedDeltaTime);
                transform.Rotate(new Vector3(0, Input.GetAxis("Horizontal") * 2, 0));
        }
}
```

The shooting is on the Update method of the PlayerMovement script. But it can't be done in the regular way, instantiating a bullet and shoot it, because the bullet will be spawned only on the client that made the shoot, so the other clients will not see the action.

Unity have something called Remote Actions [25] that allows to execute code on the host from a call of some client and vice versa. The remote actions are Remote Procedure Calls, and there are two types on the Unity network system:

- Commands: are called from the client and run on the host. Command methods have a "Cmd" prefix on them.
- ClientRpc: are called on the host and run on all clients, including the one acting as host. ClientRpc methods have a "Rpc" prefix on them.

Some actions or information that the gameplay handles needs to be synchronized everywhere, so some actions, like the shooting mentioned before, need to use these special methods.

For example, for shooting a Command is used, and it spawns the bullet on the host and then uses a networked spawn to sent it to the clients.

Code 25: Command called on the clients and executed on the host for shooting.

```
[Command]
private void CmdShoot(Vector3 position, Vector3 direction, NetworkInstanceId playerNetID) {

       BulletInfo bulletInfo = Instantiate(Bullet, position,
Quaternion.LookRotation(direction)) as BulletInfo;
       GameObject player = ClientScene.FindLocalObject(playerNetID);
       NetworkServer.SpawnWithClientAuthority(bulletInfo.gameObject, player);
       bulletInfo.RpcSetPlayer(playerNetID);
       bulletInfo.RpcApplyforce(direction * 800);
}
```

On the Code 25, we can see the code of the Command used for shooting. Fater spawning the GameObject on the clients two ClientRpc of the BulletInfo class are called.

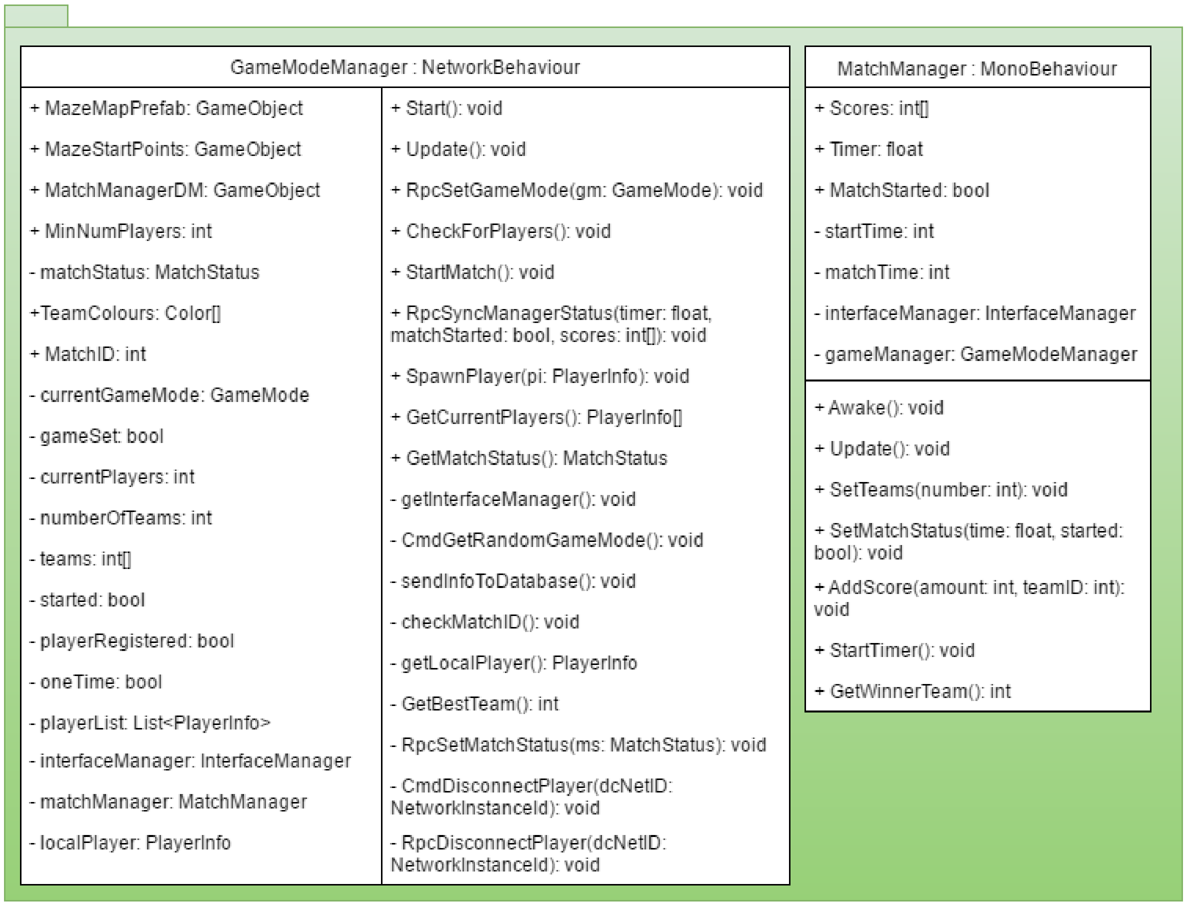
The Bullet has the component BulletInfo on it, and when is spawned on the clients that component is not synched even the values are set on the server. That's why after spawning it, two different ClientRpc are executed. The first to tell the bullet who is the player that shooted it, and the second to apply the force for the movement.

The player GameObject has a PlayerInfo component attached to it, like the bullet, and it manages all variables that the player has, like the amount of health, name and scores. As seen on the Figure 28, the class diagram, it has lot of Commands and ClientRpcs to sync all of these data between all clients. Most Commands are just a simple operations like updating some variable, and after that a ClientRpc call to send the new value to the clients.

Code 26: Example of one Command and ClientRpc from PlayerInfo.

```
[Command]
private void CmdAddScore(int amount) {
	PersonalScore += amount; //Personal score is updated
	RpcSetScore(PersonalScore); //and sent to all clients
}
[ClientRpc]
private void RpcSetScore(int score) {
	PersonalScore = score;
	Kills++;
	if (isLocalPlayer)
		UpdateInDatabase();
}
```

Figure 29. GameManager class diagram.


GameModeManager : NetworkBehaviour

+ MazeMapPrefab: GameObject
+ MazeStartPoints: GameObject
+ MatchManagerDM: GameObject
+ MinNumPlayers: int
- matchStatus: MatchStatus
+TeamColours: Color[]
+ MatchID: int
- currentGameMode: GameMode
- gameSet: bool
- currentPlayers: int
- numberOfTeams: int
- teams: int[]
- started: bool
- playerRegistered: bool
- oneTime: bool
- playerList: List<PlayerInfo>
- interfaceManager: InterfaceManager
- matchManager: MatchManager
- localPlayer: PlayerInfo

+ Start(): void
+ Update(): void
+ RpcSetGameMode(gm: GameMode): void
+ CheckForPlayers(): void
+ StartMatch(): void
+ RpcSyncManagerStatus(timer: float, matchStarted: bool, scores: int[]): void
+ SpawnPlayer(pi: PlayerInfo): void
+ GetCurrentPlayers(): PlayerInfo[]
+ GetMatchStatus(): MatchStatus
- getInterfaceManager(): void
- CmdGetRandomGameMode(): void
- sendInfoToDatabase(): void
- checkMatchID(): void
- getLocalPlayer(): PlayerInfo
- GetBestTeam(): int
- RpcSetMatchStatus(ms: MatchStatus): void
- CmdDisconnectPlayer(dcNetID: NetworkInstanceId): void
- RpcDisconnectPlayer(dcNetID: NetworkInstanceId): void

MatchManager : MonoBehaviour

+ Scores: int[]
+ Timer: float
+ MatchStarted: bool
- startTime: int
- matchTime: int
- interfaceManager: InterfaceManager
- gameManager: GameModeManager

+ Awake(): void
+ Update(): void
+ SetTeams(number: int): void
+ SetMatchStatus(time: float, started: bool): void
+ AddScore(amount: int, teamID: int): void
+ StartTimer(): void
+ GetWinnerTeam(): int

For the management of the match progress there is a GameObject called "GameManager" with the script GameModeManager attached to it.

This script is what decides which gamemode is and when to start the match. The script starts choosing gamemode from the available from a list and depending of the chosen one, the map will be spawned. That only happens on the server side (host), and when a client connects, obtains the gamemode with a Command and a ClientRpc methods.

The script has a MatchStatus enumeration that defines the five phases of the match:

- WAITING_PLAYERS: There are not enough players to start the match, and is waiting to reach the minimum.
- PREPARING: The preparation phase. Where the necessary number of players is present and some operations to start the match are done.
- PLAYING: The match is in progress.
- FINISHED: The match is finished.
- RESULTS: The final results screen is shown.

During the WAITING_PLAYERS phase, as said the script remains on hold, showing the list of players connected on the screen, until the minimum number of players to start a match is reached.

When the minimum number of players are connected, the match changes to the PREPARING phase. On this phase the players are spawned to the corresponding team bases and the MatchManager object is spawned.

The MatchManager object is slightly different of the other spawned objects because it's not a networked object. That means every client has its own local MatchManager spawned.

The match manager is a script that manages the time and scores of the match. It has the time remaining to finish the match and the team scores. In addition, it decides which is the winner team.

The status of the MatchManager is synched periodically from the game manager, with some RPC calls, so all clients have the same time and scores as the server object.

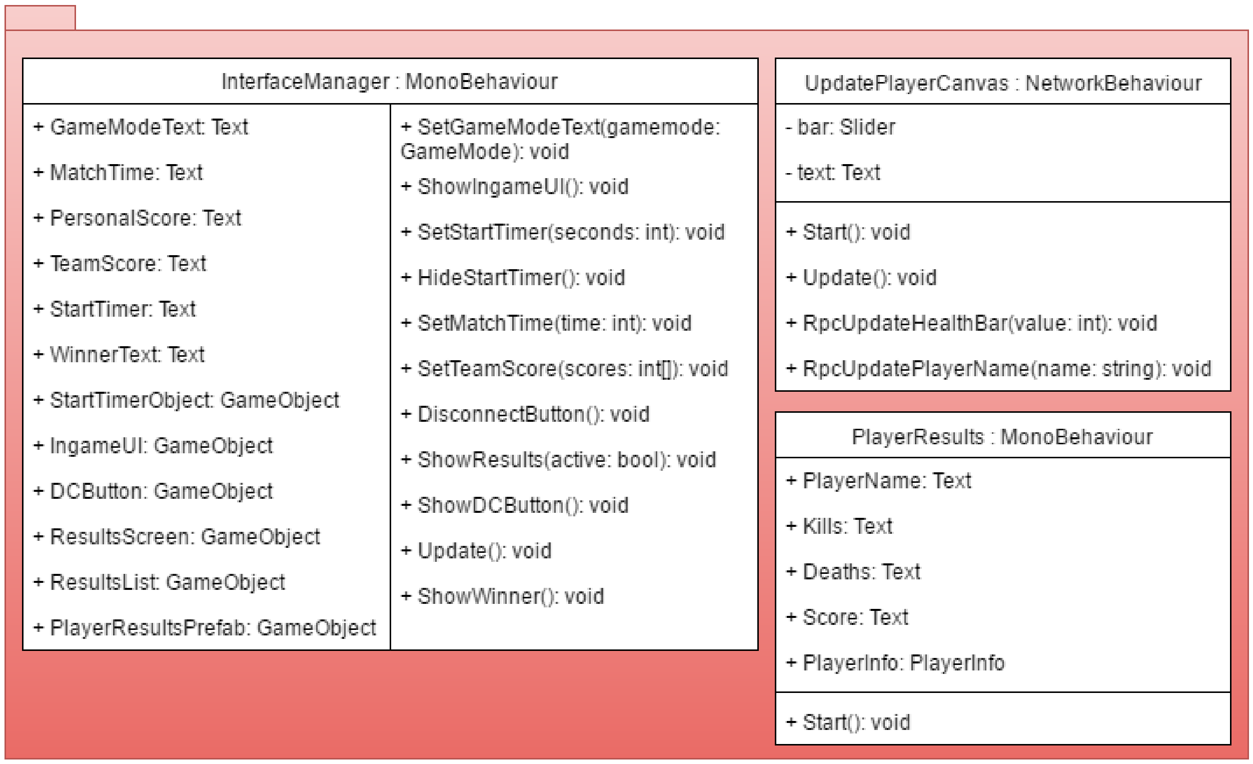
During the PREPARING phase, the match manager shows a countdown timer to start.

When that timer reaches 0, the match starts and the match timer start to count down. During that period the players can move and kill each others on the map while the scores are counted on the match manager. If some player dies, it's despawned to a neutral point located below the map disabling the movement component and changing the camera to a fixed place. After 5 seconds, it recovers the health and respawns again on the base.

When the time of the match manager reaches 0, the match will be over, and the FINISHED phase is set. This phase affects in the most part to the host because is designed to despawn all players and send the information to the database with the messages seen in the 3.3.3 section.

After doing that, the final phase is set: RESULTS. On this phase, the results screen is shown on the screen with the scores of all players, with a button for disconnect and return to the main menu.

Figure 30: Interface scripts class diagram.

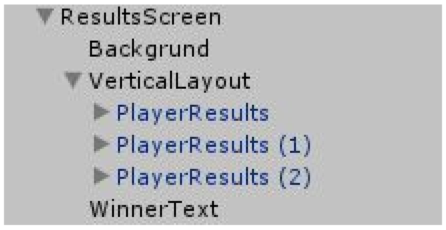

InterfaceManager : MonoBehaviour
+ GameModeText: Text
+ MatchTime: Text
+ PersonalScore: Text
+ TeamScore: Text
+ StartTimer: Text
+ WinnerText: Text
+ StartTimerObject: GameObject
+ IngameUI: GameObject
+ DCButton: GameObject
+ ResultsScreen: GameObject
+ ResultsList: GameObject
+ PlayerResultsPrefab: GameObject
+ SetGameModeText(gamemode: GameMode): void
+ ShowIngameUI(): void
+ SetStartTimer(seconds: int): void
+ HideStartTimer(): void
+ SetMatchTime(time: int): void
+ SetTeamScore(scores: int[]): void
+ DisconnectButton(): void
+ ShowResults(active: bool): void
+ ShowDCButton(): void
+ Update(): void
+ ShowWinner(): void
UpdatePlayerCanvas : NetworkBehaviour
- bar: Slider
- text: Text
+ Start(): void
+ Update(): void
+ RpcUpdateHealthBar(value: int): void
+ RpcUpdatePlayerName(name: string): void
PlayerResults : MonoBehaviour
+ PlayerName: Text
+ Kills: Text
+ Deaths: Text
+ Score: Text
+ PlayerInfo: PlayerInfo
+ Start(): void

All changes to the interface elements is managed by the component InterfaceManager. It is attached to the GameManager GameObject of the hierarchy, and has the control over all elements of the interface, for example, showing the results screen or changing the values of the time and scores.

The results screen is shown at the beginning and at the end of the match automatically, but during the match the players can open that screen by holding the Tabulator key.

The results screen is composed by some prefab objects created on a Vertical Layout, and each one managed by the PlayerResults component that sets the values to the fields.

Figure 31: Results screen hierarchy.


ResultsScreen
Backgrund
VerticalLayout
PlayerResults
PlayerResults (1)
PlayerResults (2)
WinnerText

Finally, there is a script called CloseHook, present on all scenes that handles with the close of the application. This allows the interception of the close event and execute some code, like disconnection from a match or from the server before the application is completely closed.

Code 27: CloseHook method handling the application quit.

```
void OnApplicationQuit () {
        ConnectionManager.Instance.Disconnect();

        if(NetworkClient.active) {
                MatchMaker matchMaker = FindObjectOfType<MatchMaker>();
                matchMaker.DropConnection();
        }
}
```

4. Problems during the development

All projects have some problem during the development that needs to be addressed to continue and not get stuck.

During the development of this project, a very big problem appeared on the server application. that problem was a very high time consuming because was difficult to identify where it was an what was causing it.

By some reason, after sending a message from the server to the client (happened after sending the LOGIN_OK message), one of the two sides was closing the connection. The server on the moment of the close has been waiting to a new incoming messages, so the message appeared on the stack trace suggested that the problem was something with the Receive method, or something unknown done on the client (unknown because here any message appears).

After rewriting two times the whole server loop trying to now what was done incorrectly, the same error was still present. I downloaded a software called Wireshark that allows you to see the packets sent and received by any ip and port combination. With that program I seen that the client was sending a packet with the RST flag that was resetting the connection losing it on both sides, after receiving packet with the ACK flag with the data sent by the server.

After trying to know what's causing the problem causing that RST, I found other socket class.

I was originally using the TcpClient [26] and TcpListener [27], so I switched to the Socket [28] class i recently found, and the problem disappeared. At this moment I skill not known what really caused the problem, but probably was some bad configuration of the socket.

That consumed lot of time, more than expected, duplicating the expected time for the development of the server.

The game client took some time too, but not for problems, it was for the learning period of how the Unity networking system works. That took some time and in consequence, the demo was not completed with all the gamemodes and planned features like the in game shop, the friends list and the report system.

5. Tools used

During the development of the project multiple tools were used, most of them mentioned in the section 7 of the Chapter 2, but new ones have been used.

On the following table are the deliverables and the tools used for each one.

Table 10: New deliverables and tools.

Deliverables	Tools
D1: Technical Proposal	Documents by Google, GanttProject, Adobe Photoshop
D2: Database	PostgreSQL and pgAdmin III, Notepad++, Dia, Vertabello, Documents by Google, Draw.io
D3: Server app	Visual Studio Community 2015 (console C#), Wireshark
D4: Webpage	Notepad++, EasyPHP, Google Chrome
D5: Game Launcher	Visual Studio Community 2015 (visual C#)
D6: Videogame demo	Unity3D, Visual Studio Community 2015 (C#)
D7: Shared Library	Visual Studio Community 2015 (library C#)
D8: Memory	Documents by Google, GanttProject, Dia, Adobe Photoshop, Draw.io

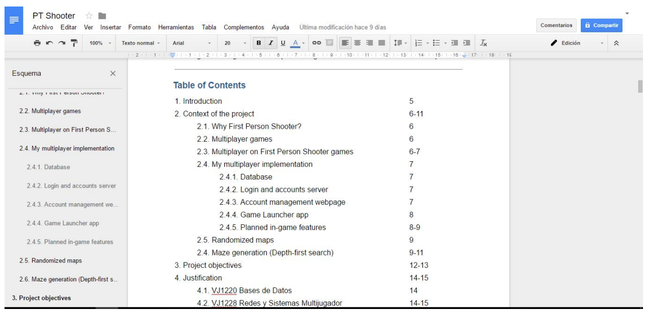
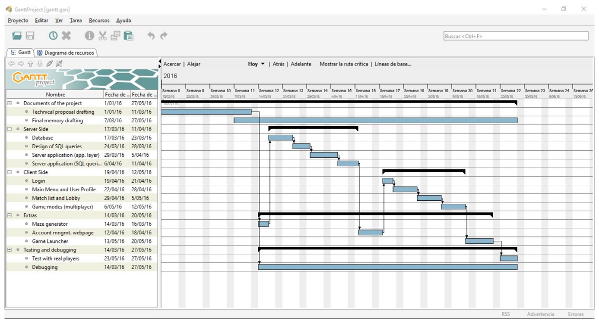

PT Shooter
Archivo Editar Ver Insertar Formato Herramientas Tabla Complementos Ayuda Última modificación hace 9 días
Comentarios
Compartir
Esquema

Table of Contents

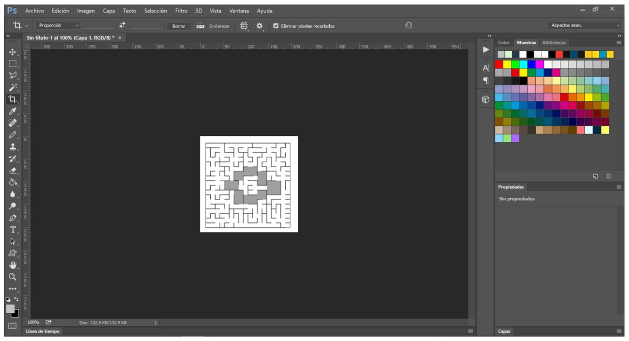
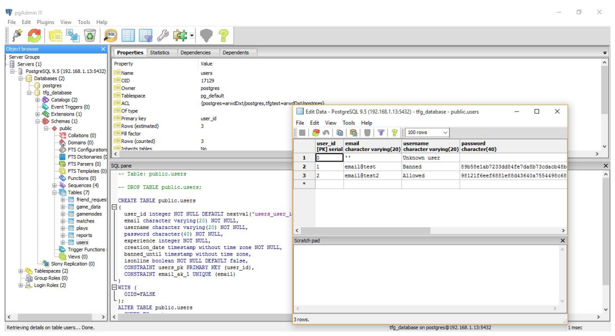
1. Introduction	5
2. Context of the project	6-11
2.1. Why First Person Shooter?	6
2.2. Multiplayer games	6
2.3. Multiplayer on First Person Shooter games	6-7
2.4. My multiplayer implementation	7
2.4.1. Database	7
2.4.2. Login and accounts server	7
2.4.3. Account management webpage	7
2.4.4. Game Launcher app	8
2.4.5. Planned in-game features	8-9
2.5. Randomized maps	9
2.4. Maze generation (Depth-first search)	9-11
3. Project objectives	12-13
4. Justification	14-15
4.1. VJ1220 Bases de Datos	14
4.2. VJ1228 Redes y Sistemas Multijugador	14-15



Nombre	Fecha de ...	Fecha de ...
Documents of the project	1/01/16	27/05/16
Technical proposal drafting	1/01/16	11/03/16
Final memory drafting	7/03/16	27/05/16
Server Side	17/03/16	11/04/16
Database	17/03/16	23/03/16
Design of SQL queries	24/03/16	28/03/16
Server application (app. layer)	29/03/16	5/04/16
Server application (SQL queri...	6/04/16	11/04/16
Client Side	19/04/16	12/05/16
Login	19/04/16	21/04/16
Main Menu and User Profile	22/04/16	28/04/16
Match list and Lobby	29/04/16	5/05/16
Game modes (multiplayer)	6/05/16	12/05/16
Extras	14/03/16	20/05/16
Maze generator	14/03/16	16/03/16
Account mngmt. webpage	12/04/16	18/04/16
Game Launcher	13/05/16	20/05/16
Testing and debugging	14/03/16	27/05/16
Test with real players	23/05/16	27/05/16
Debugging	14/03/16	27/05/16

Documents by Google (left): Used for writing the Technical Proposal and this Memory. I use this on all the documents that I write instead of other office tools because It's easy to use, has the same or similar tools of the others and ease the sharing of documents.

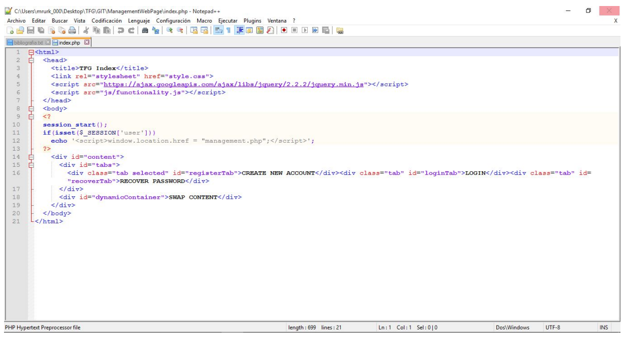
GanttProject (right): Was used to create the Gantt diagram of the Technical Proposal. I never used it before, but is very easy to use.

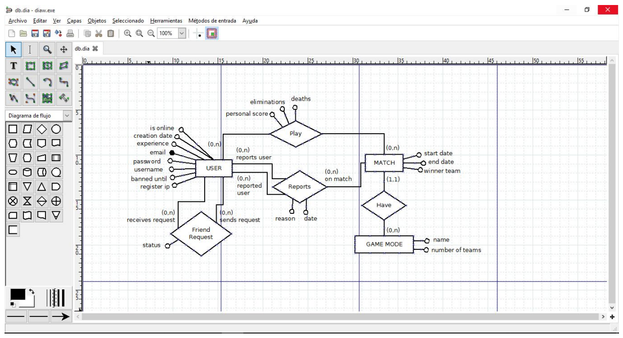
```
-- Table: public.users

-- DROP TABLE public.users;

CREATE TABLE public.users
(
  user_id integer NOT NULL DEFAULT nextval('users_user_i
  email character varying(20) NOT NULL,
  username character varying(20) NOT NULL,
  password character(40) NOT NULL,
  experience integer NOT NULL,
  creation_date timestamp without time zone NOT NULL,
  banned_until timestamp without time zone,
  isonline boolean NOT NULL DEFAULT false,
  CONSTRAINT users_pk PRIMARY KEY (user_id),
  CONSTRAINT email_ak_1 UNIQUE (email)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.users
```

Adobe Photoshop (left): Used only to create the graphics of the Maze Generator section of the Technical Proposal. A very powerful tool to edit images.

PostgreSQL and pgAdmin III (right): The database manager selected and the admin included on it. The admin was used to design and test the queries and the functionality of the database.

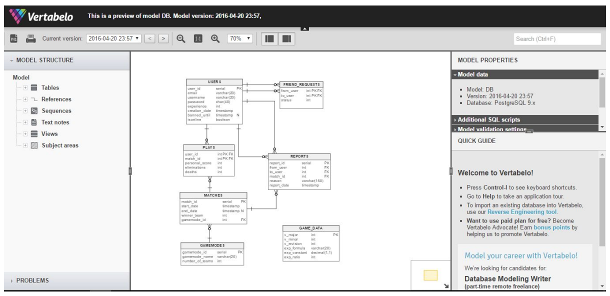
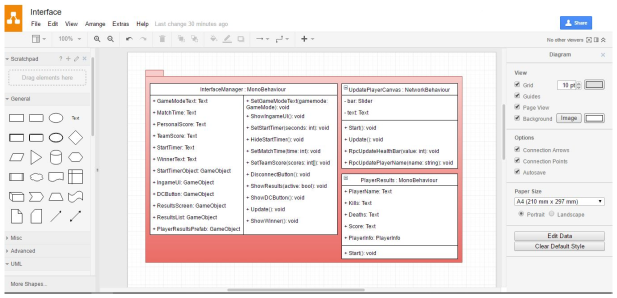


```
<html>
  <head>
    <title>TFG Index</title>
    <link rel="stylesheet" href="style.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.2/jquery.min.js"></script>
    <script src="js/functionality.js"></script>
  </head>
  <body>
  <?
  session_start();
  if(isset($_SESSION['user']))
    echo '<script>window.location.href = "management.php";</script>';
  ?>
    <div id="content">
      <div id="tabs">
        <div class="tab selected" id="registerTab">CREATE NEW ACCOUNT</div><div class="tab" id="loginTab">LOGIN</div><div class="tab" id=
        "recoverTab">RECOVER PASSWORD</div>
      </div>
      <div id="dynamicContainer">SWAP CONTENT</div>
    </div>
  </body>
</html>
```



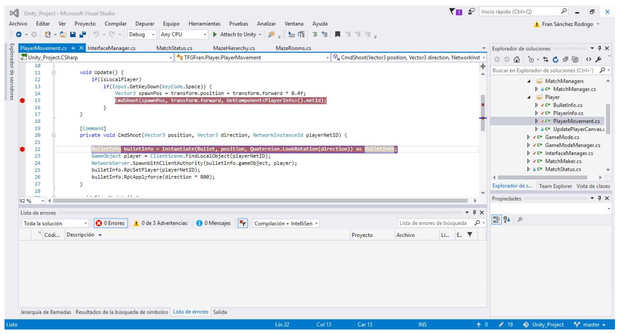
Notepad++ (left): A powerful plain text editor with code formatting for lot of programming languages. Used for the creation for the web page and edit the database.

Dia (right): Used only for the database entity-relationship diagram. It's a graphic editor free and easy to use.

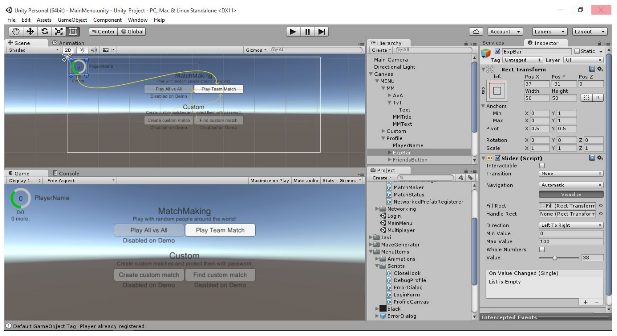



Vertabelo (left): A database designer easy to use that allows to create databases in a visual way and then export the sql code. It supports PostgreSQL, MySQL, MS SQL Server 2012, SQLite and more.

Draw.io (right): A graphic editor for Google Drive. It allows the creation of all type of graphics and was used to create the all the present on this memory. All class diagrams were used following a UML class diagram reference [29].

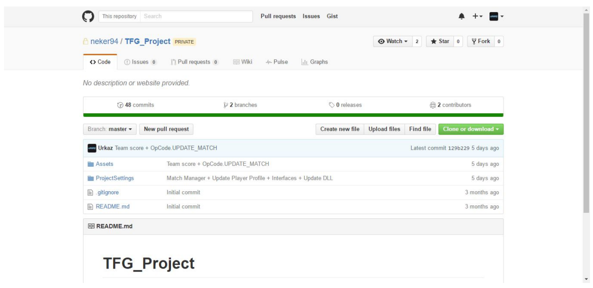


```
void Update() {
    if(isLocalPlayer)
        if(Input.GetKeyDown(KeyCode.Space)) {
            Vector3 spawnPos = transform.position + transform.forward * 0.4f;
            CmdShoot(spawnPos, transform.forward, GetComponent<PlayerInfo>().netId);
        }
}

[Command]
private void CmdShoot(Vector3 position, Vector3 direction, NetworkInstanceId playerNetID) {

    BulletInfo bulletInfo = Instantiate(Bullet, position, Quaternion.LookRotation(direction)) as BulletInfo;
    GameObject player = ClientScene.FindLocalObject(playerNetID);
    NetworkServer.SpawnWithClientAuthority(bulletInfo.gameObject, player);
    bulletInfo.RpcSetPlayer(playerNetID);
    bulletInfo.RpcApplyforce(direction * 800);
}
```



Visual Studio Community 2015 (left): Used for developing the projects and program the code needed for them using C#. Used too for debugging the Unity project with the integration plugin it has.

Unity3D (right): A powerful 3D game engine that was used to create the game demo.



neker94 / TFG_Project PRIVATE

No description or website provided.

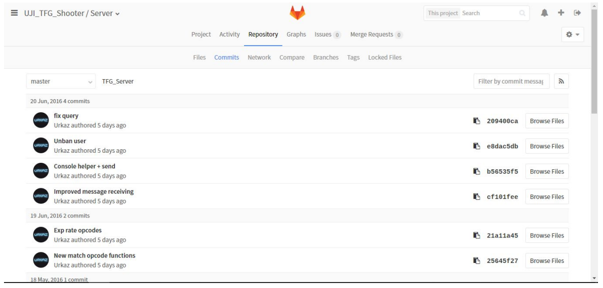
48 commits 2 branches 0 releases 2 contributors

Urkaz Team score + OpCode.UPDATE_MATCH Latest commit 129b229 5 days ago

Assets	Team score + OpCode.UPDATE_MATCH	5 days ago
ProjectSettings	Match Manager + Update Player Profile + Interfaces + Update DLL	5 days ago
.gitignore	Initial commit	3 months ago
README.md	Initial commit	3 months ago

README.md

TFG_Project



UJI_TFG_Shooter / Server

master TFG_Server

20 Jun, 2016 4 commits

fix query Urkaz authored 5 days ago	209400ca	Browse Files
Unban user Urkaz authored 5 days ago	e8dac5db	Browse Files
Console helper + send Urkaz authored 5 days ago	b56535f5	Browse Files
Improved message receiving Urkaz authored 5 days ago	cf101fee	Browse Files

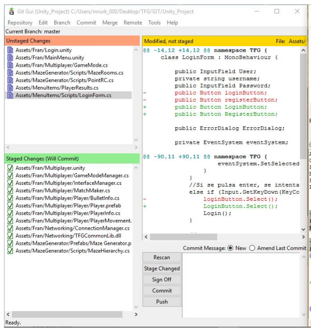
19 Jun, 2016 2 commits

Exp rate opcodes Urkaz authored 5 days ago	21a11a45	Browse Files
New match opcode functions Urkaz authored 5 days ago	25645f27	Browse Files

18 May, 2016 1 commit

GitHub (left): The git repository used for The Unity Project at the start. Lately it was transferred to GitLab.

GitLab (right): Similar to GitHub, but with the option to create groups of repositories. Here the repositories for the server, webpage, launcher, shader library and database were created on a group.



Git Gui (Unity_Project) C:/Users/mrurk_000/Desktop/TFG/GIT/Unity_Project

Current Branch: master

Unstaged Changes

- Assets/Fran/Login.unity
- Assets/Fran/MainMenu.unity
- Assets/Fran/Multiplayer/GameMode.cs
- Assets/MazeGenerator/Scripts/MazeRooms.cs
- Assets/MazeGenerator/Scripts/PointRC.cs
- Assets/MenuItems/PlayerResults.cs
- Assets/MenuItems/Scripts/LoginForm.cs

Staged Changes (Will Commit)

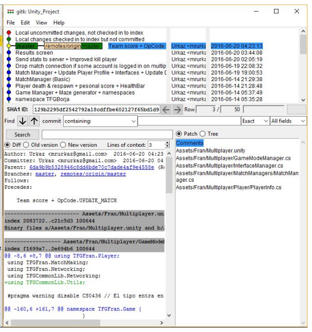
- Assets/Fran/Multiplayer.unity
- Assets/Fran/Multiplayer/GameModeManager.cs
- Assets/Fran/Multiplayer/InterfaceManager.cs
- Assets/Fran/Multiplayer/MatchMaker.cs
- Assets/Fran/Multiplayer/Player/BulletInfo.cs
- Assets/Fran/Multiplayer/Player/Player.prefab
- Assets/Fran/Multiplayer/Player/PlayerInfo.cs
- Assets/Fran/Multiplayer/Player/PlayerMovement.
- Assets/Fran/Networking/ConnectionManager.cs
- Assets/Fran/Networking/TFGCommonLib.dll
- Assets/MazeGenerator/Prefabs/Maze Generator.p
- Assets/MazeGenerator/Scripts/MazeHierarchy.cs

Modified, not staged File: Assets/

```
@@ -14,12 +14,12 @@ namespace TFG {
     class LoginForm : MonoBehaviour {

         public InputField User;
         private string username;
         public InputField Password;
-        public Button loginButton;
-        public Button registerButton;
+        public Button LoginButton;
+        public Button RegisterButton;

         public ErrorDialog ErrorDialog;

         private EventSystem eventSystem;

@@ -90,11 +90,11 @@ namespace TFG {
                         eventSystem.SetSelected
                     }
                 }
                 //Si se pulsa enter, se intenta
                 else if (Input.GetKeyDown(KeyCo
-                    loginButton.Select();
+                    LoginButton.Select();
                     Login();
                 }
```

Commit Message: (● New) ○ Amend Last Commit

Ready.



gitk: Unity_Project

```
Local uncommitted changes, not checked in to index
Local changes checked in to index but not committed
master remotes/origin/master Team score + OpCode.    Urkaz <mrurka  2016-06-20 04:23:13
Results screen                                       Urkaz <mrurka  2016-06-20 03:44:08
Send stats to server + Improved kill player          Urkaz <mrurka  2016-06-20 02:05:19
Drop match connection if some account is logged in on multip  Urkaz <mrurka  2016-06-19 22:08:32
Match Manager + Update Player Profile + Interfaces + Update [  Urkaz <mrurka  2016-06-19 19:00:53
MatchManager (Basic)                                 Urkaz <mrurka  2016-06-14 21:29:38
Player death & respawn + pesonal score + HealthBar   Urkaz <mrurka  2016-06-14 21:28:48
Game Manager + Maze generator + namespaces           Urkaz <mrurka  2016-06-14 05:37:49
namespace TFGBorja                                   Urkaz <mrurka  2016-06-14 05:35:28
```

SHA1 ID: 129b2295df2542792a18cdffbe602127f65bd1d9 Row 3 / 50

Find commit containing: Exact All fields

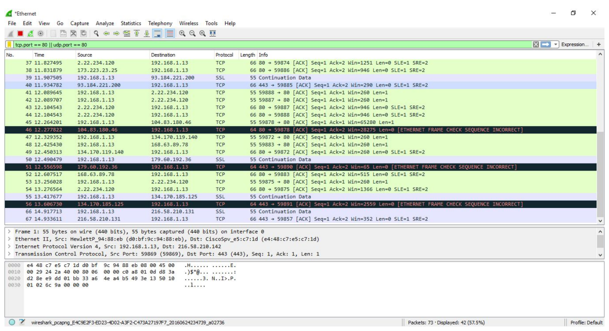
(● Diff) ○ Old version ○ New version Lines of context: 3

(● Patch) ○ Tree

```
Author: Urkaz <mrurkaz@gmail.com>  2016-06-20 04:23
Committer: Urkaz <mrurkaz@gmail.com>  2016-06-20 04
Parent: 6da9b9b5328946c8dd6bde70c7dade4af9e4558e (R
Branches: master, remotes/origin/master
Follows:
Precedes:

    Team score + OpCode.UPDATE_MATCH

----------------------- Assets/Fran/Multiplayer.un
index 2083722..c21c5d3 100644
Binary files a/Assets/Fran/Multiplayer.unity and b/

----------------- Assets/Fran/Multiplayer/GameMode
index f1699a7..2e694b6 100644
@@ -8,6 +8,7 @@ using TFGFran.Player;
 using TFGFran.MatchMaking;
 using TFGFran.Networking;
 using TFGCommonLib.Networking;
+using TFGCommonLib.Utils;

 #pragma warning disable CS0436 // El tipo entra en

@@ -160,6 +161,7 @@ namespace TFGFran.Game {
                 }
```

Comments

- Assets/Fran/Multiplayer.unity
- Assets/Fran/Multiplayer/GameModeManager.cs
- Assets/Fran/Multiplayer/InterfaceManager.cs
- Assets/Fran/Multiplayer/MatchManagers/MatchManager.cs
- Assets/Fran/Multiplayer/Player/PlayerInfo.cs



*Ethernet

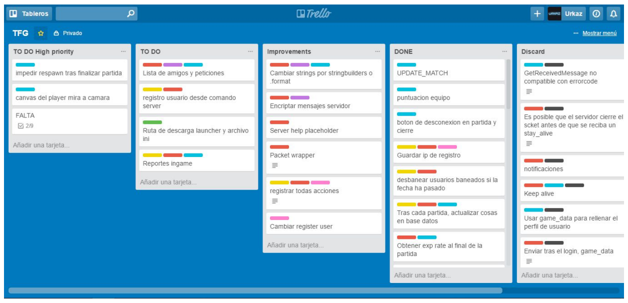
tcp.port == 80 || udp.port == 80

```
No.  Time       Source           Destination      Protocol Length Info
 37 11.827495  2.22.234.120     192.168.1.13     TCP      66 80 → 59874 [ACK] Seq=1 Ack=2 Win=1251 Len=0 SLE=1 SRE=2
 38 11.831879  173.223.23.25    192.168.1.13     TCP      66 80 → 59886 [ACK] Seq=1 Ack=2 Win=946 Len=0 SLE=1 SRE=2
 39 11.907505  192.168.1.13     93.184.221.200   SSL      55 Continuation Data
 40 11.934782  93.184.221.200   192.168.1.13     TCP      66 443 → 59885 [ACK] Seq=1 Ack=2 Win=290 Len=0 SLE=1 SRE=2
 41 12.089645  192.168.1.13     2.22.234.120     TCP      55 59888 → 80 [ACK] Seq=1 Ack=1 Win=260 Len=1
 42 12.089707  192.168.1.13     2.22.234.120     TCP      55 59887 → 80 [ACK] Seq=1 Ack=1 Win=260 Len=1
 43 12.104543  2.22.234.120     192.168.1.13     TCP      66 80 → 59887 [ACK] Seq=1 Ack=2 Win=946 Len=0 SLE=1 SRE=2
 44 12.104543  2.22.234.120     192.168.1.13     TCP      66 80 → 59888 [ACK] Seq=1 Ack=2 Win=946 Len=0 SLE=1 SRE=2
 45 12.264201  192.168.1.13     104.83.180.46    TCP      55 59878 → 80 [ACK] Seq=1 Ack=1 Win=65280 Len=1
 46 12.277822  104.83.180.46    192.168.1.13     TCP      64 80 → 59878 [ACK] Seq=1 Ack=2 Win=28275 Len=0 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
 47 12.329352  192.168.1.13     134.170.119.140  TCP      55 59872 → 80 [ACK] Seq=1 Ack=1 Win=260 Len=1
 48 12.425430  192.168.1.13     168.63.89.78     TCP      55 59883 → 80 [ACK] Seq=1 Ack=1 Win=260 Len=1
 49 12.450313  134.170.119.140  192.168.1.13     TCP      66 80 → 59872 [ACK] Seq=1 Ack=2 Win=260 Len=0 SLE=1 SRE=2
 50 12.490479  192.168.1.13     179.60.192.36    SSL      55 Continuation Data
 51 12.556598  179.60.192.36    192.168.1.13     TCP      64 443 → 59890 [ACK] Seq=1 Ack=2 Win=65 Len=0 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
 52 12.607517  168.63.89.78     192.168.1.13     TCP      66 80 → 59883 [ACK] Seq=1 Ack=2 Win=515 Len=0 SLE=1 SRE=2
 53 13.256028  192.168.1.13     2.22.234.120     TCP      55 59875 → 80 [ACK] Seq=1 Ack=1 Win=260 Len=1
 54 13.276564  2.22.234.120     192.168.1.13     TCP      66 80 → 59875 [ACK] Seq=1 Ack=2 Win=1366 Len=0 SLE=1 SRE=2
 55 13.417677  192.168.1.13     134.170.185.125  SSL      55 Continuation Data
 56 13.606730  134.170.185.125  192.168.1.13     TCP      64 443 → 59891 [ACK] Seq=1 Ack=2 Win=2559 Len=0 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
 66 14.917713  192.168.1.13     216.58.210.131   SSL      55 Continuation Data
 67 14.933611  216.58.210.131   192.168.1.13     TCP      66 443 → 59857 [ACK] Seq=1 Ack=2 Win=352 Len=0 SLE=1 SRE=2

> Frame 1: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface 0
> Ethernet II, Src: HewlettP_94:88:eb (d0:bf:9c:94:88:eb), Dst: CiscoSpv_e5:c7:1d (e4:48:c7:e5:c7:1d)
> Internet Protocol Version 4, Src: 192.168.1.13, Dst: 216.58.210.142
> Transmission Control Protocol, Src Port: 59869 (59869), Dst Port: 443 (443), Seq: 1, Ack: 1, Len: 1

0000  e4 48 c7 e5 c7 1d d0 bf  9c 94 88 eb 08 00 45 00   .H...... ......E.
0010  00 29 24 2a 40 00 80 06  00 00 c0 a8 01 0d d8 3a   .)$*@... .......:
0020  d2 8e e9 dd 01 bb 33 a6  4e a4 b5 49 3e 13 50 10   ......3. N..I>.P.
0030  01 02 6c 9a 00 00 00                               ..l....
```

wireshark_pcapng_E4C9E2F3-ED23-4D02-A3F2-C473A27197F7_20160624234739_a02736 | Packets: 73 · Displayed: 42 (57.5%) | Profile: Default

Git SCM (left): Git client used to manage the local repositories and update the remote ones with the new changes.

Wireshark (right): Program that allows you to see the packets sent and received on your computer. They can be filtered by protocol and ip for example and their content can be seen to create a trace. Used when the problem with the sockets, mentioned on the last section, to see what was happening with the packets.



TFG · Privado

TO DO High priority
- impedir respawn tras finalizar partida
- canvas del player mira a camara
- FALTA 2/9

TO DO
- Lista de amigos y peticiones
- registro usuario desde comando server
- Ruta de descarga launcher y archivo ini
- Reportes ingame

Improvements
- Cambiar strings por stringbuilders o .format
- Encriptar mensajes servidor
- Server help placeholder
- Packet wrapper
- registrar todas acciones
- Cambiar register user

DONE
- UPDATE_MATCH
- puntuacion equipo
- boton de desconexion en partida y cierre
- Guardar ip de registro
- desbanear usuarios baneados si la fecha ha pasado
- Tras cada partida, actualizar cosas en base datos
- Obtener exp rate al final de la partida

Discard
- GetReceivedMessage no compatible con errorcode
- Es posible que el servidor cierre el scket antes de que se reciba un stay_alive
- notificaciones
- Keep alive
- Usar game_data para rellenar el perfil de usuario
- Enviar tras el login, game_data

Trello: Tool used for organizing tasks and things to do between all projects on panels using cards, similar to using post-its on a wall on the real life.

EasyPHP Webserver (14.1): A software with a built-in Apache server with PHP support and MySQL database creation with PHPMyAdmin installed on it. For the project only the Apache server with a modification on the PHP configuration to support PostgreSQL was used.

Google Chrome: The web browser used to test the web page.

Windows Remote Desktop: Remote desktop tool included on windows to manage a remote desktop. As the server, database and web page were on another computer than the client, this tool was used to configure all the necessary on that computer to make them work on a local network.

Windows Snipping Tool: Tool included on Windows to create screenshots or snippets of the screen. Used to create the most screenshots and figures from the different projects and softwares.

6. Conclusions

In conclusion, most of the objectives are completed, but not all because the problems explained on the section 4 of this chapter.

• **OBJ1: Develop a multiplayer environment similar to that in some important games of the industry.** The necessary tools were created, but not all the features were implemented on the game client, so this global objective can be marked as partially completed.

• **OBJ2: Expand the knowledge of Unity3D to apply them in a professional project.** I learned a lot of how the multiplayer system of Unity works, and now I know how to create scenes and scripts to create multiplayer games, or adapt single player ones to be played with multiple players online. This objective is completed.

• **OBJ3: Learn and experiment with new technology. Self learning.**

Similar to the OBJ2, but applied to the rest of the projects. I improved a lot my knowledge of the sockets and the client handling with the TCP protocol, and send and download files. In addition, I learned how to send formularies with javascript without refreshing the page thanks to the post method and javascript. And finally, now I meet more tools to work with in the future. This objective is completed.

• **OBJ4: Implement a database with good performance and table relations.**

The database was created with the help of the book from the subject of the degree, and the queries done on the server were made with prepared statements to increase the security, so this objective can be marked as completed.

• **OBJ5: Create a stable server communication framework.**

The server communication is explained on the sections 3.3 and 3.4 of this chapter. The method is stable and tested when creating the game client so this objective can be marked as completed.

• **OBJ6: Create a multiplayer game modes.**

The game have a MatchMaker as the objective said on the chapter 2 section 3, but not all game modes were implemented so this objective can be marked as partially completed.

• **OBJ7: Create a functional registration and account management webpage.**

The webpage is fully functional and has the basic tools to manage an account, like change or recover password, change email, download the game client and register new users. This objective is completed.

• OBJ8: Create a tool that helps to maintain the client always updated.

This objective refers to the Game Launcher, that downloads the last version of the game if there is any update, maintaining the version consistent between all clients, so this objective is completed.

CHAPTER 4: USER MANUAL

This chapter will consist on a brief explanation of how the project is configured and how it works.

The first thing we should mention, is that the project configuration contains two different computers in the equation. One of the computers is acting as a server and has all the software necessary installed on it to do it. And the other is where the client is executed.

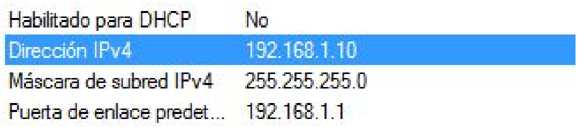
Both computers are connected to the same local network and have access to the internet.

The local network should have the following properties:

- Gateway: 192.168.1.0
- Mask: 255.255.255.0

The computer acting as server should have the static IP 192.168.1.10. On this computer the softwares for the database (PostgreSQL) and the webpage (EasyPHP) should be installed.

Figure 32: Network configuration of the server.

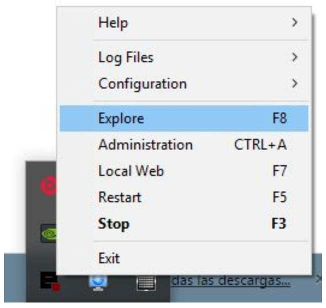


Habilitado para DHCP	No
Dirección IPv4	192.168.1.10
Máscara de subred IPv4	255.255.255.0
Puerta de enlace predet...	192.168.1.1

On the database a new user should be created with name and password "tfgtest", and a new database called "tfg_database". On this database the sql code for the creation of the tables and the initial inserts should be executed. This operations can be done on the pgAdmin III provided with the PostgreSQL installation.

When running EasyPHP will place it on the windows tray. Right clicking on it will show the options, and selecting Explore will open the folder where the files of the webpage should be.

Figure 33: EasyPHP Explore option.



In addition, on the Configuration menu, we should configure PHP to be able to communicate with PostgreSQL databases, and allow the short open tag "<?" instead of "<?php". On the file that will open, do the following changes:

Table 11: PHP.ini modifications

`short_open_tag = Off`
`//...`
`;extension=php_openssl.dll`
`;extension=php_pdo_pgsql.dll`
`;extension=php_pgsql.dll`
`;extension=php_openssl.dll`
	`short_open_tag = On`
	`//...`
	`extension=php_openssl.dll`
	`extension=php_pdo_pgsql.dll`
	`extension=php_pgsql.dll`
	`extension=php_openssl.dll`

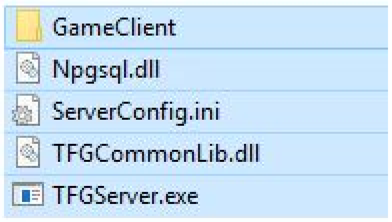
In addition to the php configuration, a single change should be done on the Apache config to allow the connection from another computer.

Table 12: Apache httpd.conf modifications.

`Listen 127.0.0.1:80`	`Listen 127.0.0.1:80`
	`Listen *:80`

The server application should be (or at least I recommend it) in a folder, without any other software or extra files. The game client should be places on a folder called "GameClient". Everything on that folder will be downloaded by the launcher when downloading or updating the client.

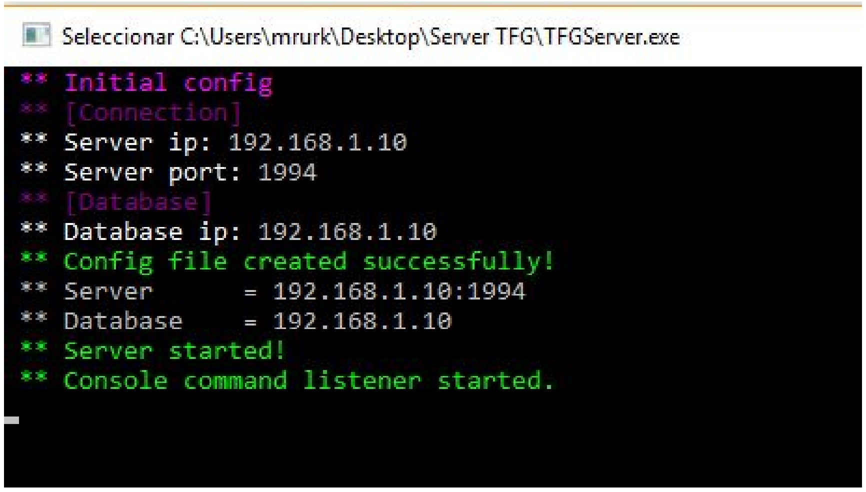
Figure 34: Server application folder.



GameClient
Npgsql.dll
ServerConfig.ini
TFGCommonLib.dll
TFGServer.exe

With this configured, the server can be opened. If it don't have an initial config stored, it will prompt the user to introduce the ip and port of the current computer and the ip where is the database (in this case both ip are 192.168.1.10 and the port 1994).

Figure 35: Server initial config.



```
Seleccionar C:\Users\mrurk\Desktop\Server TFG\TFGServer.exe
** Initial config
** [Connection]
** Server ip: 192.168.1.10
** Server port: 1994
** [Database]
** Database ip: 192.168.1.10
** Config file created successfully!
** Server      = 192.168.1.10:1994
** Database    = 192.168.1.10
** Server started!
** Console command listener started.
```

With all of this configured, now using other computer we should be able to access to the web page by writing the ip of the server on any browser. A new user can be created on the formulary, and when logged in, the launcher can be downloaded.

With the launcher we can download the game client and then, open it and start to play.

CHAPTER 5: BIBLIOGRAPHY AND ANNEXES

Bibliography

[1] Unity Documentation, Multiplayer and Networking.
http://docs.unity3d.com/Manual/UNet.html

[2] Mercedes Marqués. "Bases de datos". Col·lecció Sapientia. Primera edición, 2011. ISBN: 978-84-693-0146-3.

[3] Maze generation using "Randomized Depth First Search" algorithm.
https://en.wikipedia.org/wiki/Maze_generation_algorithm#Depth-first_search

[4] Final Degree Project Teaching Guide
https://e-ujier.uji.es/pls/www/gri_www.euji22883_html?p_curso_aca=2015&p_asignatura_id=VJ1241&p_idioma=ES&p_titulacion=231

[5] Microsoft Naming Guidelines
https://msdn.microsoft.com/en-us/library/ms229002.aspx

[6] Vertabelo - Design Your Database Online
http://www.vertabelo.com/

[7] An INI file handling class using C#
http://www.codeproject.com/Articles/1966/An-INI-file-handling-class-using-C

[8] The Dispatcher Pattern, Ashley Davis, What could possibly go wrong?
http://www.what-could-possibly-go-wrong.com/the-dispatcher-pattern/

[9] NuGet Package Manager
https://www.nuget.org/

[10] Npgsql NuGet plugin
https://www.nuget.org/packages/Npgsql/

[11] Launch external executable with C#
http://stackoverflow.com/questions/240171/launching-a-application-exe-from-c/240189#240189

[12] Costura.Fody NuGet plugin
https://www.nuget.org/packages/Costura.Fody/

[13] Iterate Through a Directory Tree
https://msdn.microsoft.com/en-us/library/bb513869.aspx

[14] GitHub repository of jquery.fileDownload
https://github.com/johnculviner/jquery.fileDownload

[15] FormGet, PHP Login Form with Sessions
https://www.formget.com/login-form-in-php/

[16] GitHub repository of PHPMailer
https://github.com/PHPMailer/PHPMailer

[17] GameDev, StackExchange, Experience formula
http://gamedev.stackexchange.com/questions/13638/algorithm-for-dynamically-calculating-a-level-based-on-experience-points#comment124885_13639

[18] Unity Manual, Using the NetworkManager
https://docs.unity3d.com/Manual/UNetManager.html

[19] Unity Manual, NetworkStartPosition
http://docs.unity3d.com/Manual/class-NetworkStartPosition.html

[20] Unity Manual, MatchMaker
http://docs.unity3d.com/520/Documentation/Manual/UNetMatchMaker.html

[21] Network Manager code
https://bitbucket.org/Unity-Technologies/networking/src/c26a8146968cbd61c2017b3e49327f3d2c3996fa/Runtime/NetworkManager.cs?fileviewer=file-view-default

[22] Network Manager HUD code
https://bitbucket.org/Unity-Technologies/networking/src/c26a8146968cbd61c2017b3e49327f3d2c3996fa/Runtime/NetworkManagerHUD.cs?at=5.2&fileviewer=file-view-default

[23] Unity Manual, NetworkIdentity
http://docs.unity3d.com/Manual/class-NetworkIdentity.html

[24] Unity Manual, NetworkTransform
http://docs.unity3d.com/Manual/class-NetworkTransform.html

[25] Unity Manual, Remote Actions
http://docs.unity3d.com/Manual/UNetActions.html

[26] TcpClient
https://msdn.microsoft.com/es-es/library/system.net.sockets.tcpclient(v=vs.110).aspx

[27] TcpListener
https://msdn.microsoft.com/es-es/library/system.net.sockets.tcplistener(v=vs.110).aspx

[28] C# Socket
https://msdn.microsoft.com/es-es/library/system.net.sockets.socket(v=vs.110).aspx

[29] UML Class Diagrams Reference (seen 21/jun/2016)
http://www.uml-diagrams.org/class-reference.html

Annex 1: Repository link

Link to GitLab group where all projects are located:
https://gitlab.com/groups/UJI_TFG