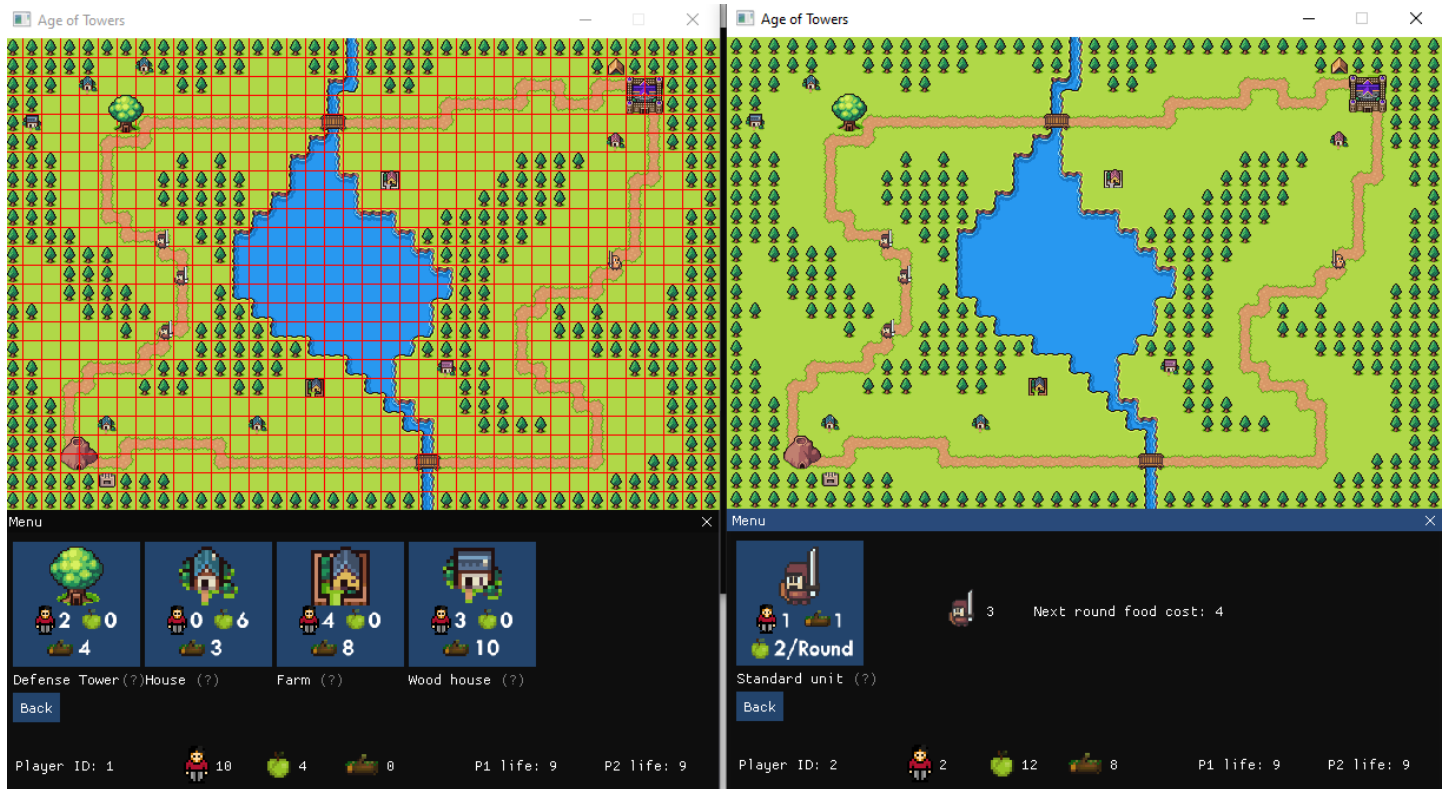


# Age of Towers

## Project 3 - Advanced Games Console Projects

### Multi-threaded network game



Diego Ochando Torres  
(SHU ID: 30022981)

## Contents

<b>GDD</b>	<b>1</b>
Game concept	1
Game rules and mechanics	1
Players	1
Resources	1
Buildings	1
Units	2
Rounds	2
Game controls and menus	2
<b>TDD</b>	<b>4</b>
Network communications	4
Multithreading	6
External dependencies	8
Further work	8

# GDD

## Game concept

This game was inspired by Age of Empires II and different Tower Defense games, resulting in a mix of mechanics of both genres.

The main objective is to destroy the castle of the enemy by buying units and placing towers to defend your castle, as well as other buildings to generate different kinds of resources to improve your civilization and win easily.

## Game rules and mechanics

### Players

Each player has different visual aspects for their buildings and units, but they all cost and behave the same way. Each player starts with a castle with 10 lives, when an enemy unit reaches the castle, the player will lose 1 life point.

In the rare case that both players reach 0 hit points at the same time, both of them will win the game.

### Resources

Both players have their own amount of resources, divided into 3 kinds, people, food and wood. These resources can be used to construct buildings to increase the resource income or to defend the paths, as well as buying units to attack the enemy castle.

Each building has its own cost and benefits, the units have an initial cost and an extra cost at their spawn.

### Buildings

Buildings can only be placed in the player's territory, divided by a river in the map. Each building has their own functionality and rules:

**Defense tower** - It will drop caltrops on each two rounds of the game in the surrounding road tiles. The caltrops will hit the units for 1 hit point and be destroyed until the next two rounds. If multiple towers affect a road tile, the hit points of the caltrops won't combine. It needs 2 people and 4 wood to be built. Can only be built in a tile with an adjacent road tile.

**House** - When a house is constructed it will add 4 people units to the player resources. It needs 6 pieces of food and 3 pieces of wood to be built.

**Farm** - It will add resources to the player by each round of the game. The number of resources will depend on the grass tiles surrounding the farm, being 8 the max resources. It needs 4 people and 8 wood to be built.

**Woodhouse** - It will add resources to the player by each round of the game. The number of resources will depend on the tree tiles surrounding the woodhouse. It needs 3 people and 10 pieces of wood to be built.

## Units

The units will spawn at the hut near to the players' castle and they will travel the path to reach the enemy castle. If a unit reaches the enemy castle, the castle will lose one hit point. Each player will start with one unit and they can acquire more in the units menu paying 1 person and 1 wood for each. Each time a unit spawns it will consume two pieces of food (except for the first one of each round), if the player does not have enough food it will not spawn more units, but they will spawn in the next round if there is enough food.

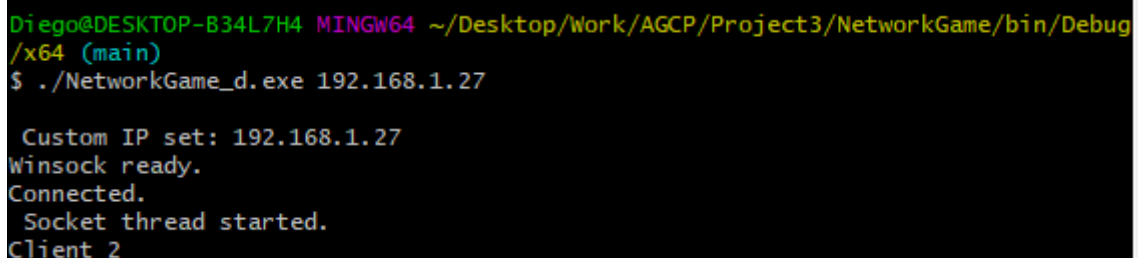
## Rounds

A new round starts when all the units of both players are killed or have reached the enemy castle. At the beginning of each round the buildings will give the resources to the owner player, the towers will drop their caltrops and a new wave of units will be spawned (Each unit consumes two pieces of food when spawning except the first one).

## Game controls and menus

### Connection

Before starting a client, the server must be started, after that you can open a client and it will connect directly to the local host. If you execute the client from a console you can send a custom IP as a parameter (the IPv4 of the PC where the server is running, it only will work in local networks unless you open port 50000, I only tried on my local network but it should work).



```
Diego@DESKTOP-B34L7H4 MINGW64 ~/Desktop/Work/AGCP/Project3/NetworkGame/bin/Debug
/x64 (main)
$ ./NetworkGame_d.exe 192.168.1.27

Custom IP set: 192.168.1.27
Winsock ready.
Connected.
Socket thread started.
Client 2
```

### Waiting menu

If a connection to the server cannot be established, a message will appear saying it to the player.

If the connection was successful, a message will say that you're waiting for another player to join the game.

If both players successfully connect, the server will send a message to both clients telling them to start the game, and the menu will change to the main one.

### Main menu and resource and life indicators

After the game starts the main menu will appear with two different buttons, the 'buildings' and 'units' buttons will open the corresponding menu to acquire new buildings or units.

The player ID, player resources and both players' lives will appear continuously while the game is still running.

### **Buildings menu**

In the buildings menu a grid will appear on the map to help you identify the map tiles and build easily. When the menu is open a building will appear in the mouse position and if you click and have enough resources the building will be built in the mouse place. You can select different buildings by pressing the buttons of the menu, the cost of each one is detailed in the buttons and a tooltip will appear when hovering them, indicating the benefit that the building will grant to the player. Each player will start with 1 house, 1 farm and 1 woodhouse.

### **Units menu**

In the buildings menu you can acquire new units for the next round of the game, each player will start with one unit. Acquiring a new unit will cost 1 person and 1 wood, although when each of them spawns in the next round they will consume 2 pieces of food.

The total number of acquired units will appear in this menu, as well as the total cost of food for the next round (the first unit will never use food).

Units will spawn while they have enough food, the units that don't have food will wait in the hut until the next round.

### **Win/Lose menus**

When a player wins, a message will be sent to the server and this will send it to the other player.

The game will end and the winner will show a win screen while the loser will show a lose screen.

In the case that both players win at the same time, it will result in a draw and both of them will win the game.

### **Disconnection menu**

If a player closes the game or the connection is dropped, a message will appear telling this to the player, and the server will close automatically. You have to close the game and start a new server and clients to keep playing.

# TDD

## Network communications

### Communication protocol

It may sound weird, but the game works with the TCP communication protocol in real time.  
(code from network\_helpers.h)

```
static int initSocket(SOCKET* sock) {  
    *sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);  
    if (*sock == INVALID_SOCKET) {  
        WSACleanup();  
        return EXIT_FAILURE;  
    }  
  
    return 0;  
}
```

Using TCP implicitly means using three-way handshake, as well as checking for missing data transmissions is not necessary using TCP, as it implicitly checks this itself. As multi-threading was required for this project, this a good use for the different threads, as we can leave the network communication in a separate thread without leaving the game stuck while sending and receiving commands.

The network works in real time using TCP with a variable number and kinds of commands. At the beginning of each loop of the network thread, each player will send a CommandListHeader (network\_data.h) containing the sender id and the number of commands of the command list.

```
struct CommandListHeader {  
    int sender_id;  
    int cmd_count;  
};
```

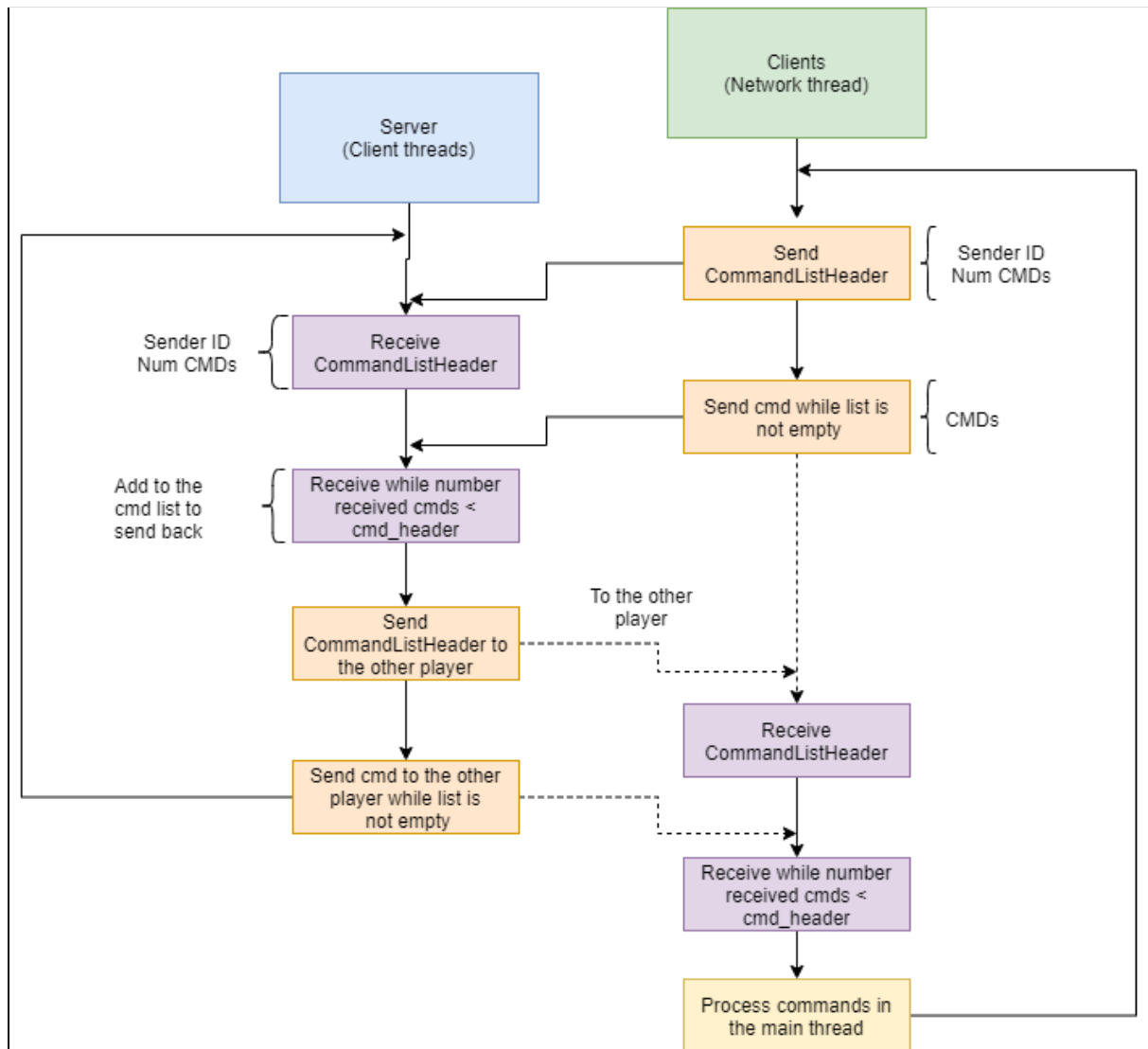
Each player will fill their command list in the update stage of the main thread, and their commands will be sent one by one until the list is empty. The client threads of the server will be waiting to receive these commands, storing them and sending them to the other player using the same process.

Finally, the other player network thread receives a first command header containing the commands that he has to receive, and after that it enters in a while loop and fills a receive command list that will be executed in the update stage of the main thread.

```
// -----  
// Used to store the commands and execute them later when the net thread is not running  
// When the net thread is running it will fill these lists instead, in parallel with the main thread  
class CommandList {  
public:  
    CommandList() {  
        commands_ = std::vector<Command*>(0);  
    }  
  
    ~CommandList() { commands_.clear(); }  
  
    std::vector<Command*> commands_;  
};  
// -----
```

Look in client\_main.cpp and server\_main.cpp to better understand this architecture.

The communication protocol explained before is shown in the image below.



Different kinds of data packages are used, they contain only integer numbers and they are fit inside a union struct to minimize the space and send/receive them all in the same way (network\_data.h).

```
// Package to send, can contain different information, identified later by the receiver
struct DataPackage {
    DataPackageKind package_kind;

    union {
        Client client;
        CommandListHeader header;
        StartGame start;
        EndGame end;
        UnitsEnd units_end;
        CastleLife castle_life;
        BuildData build;
        UnitData unit;
    };
};
```

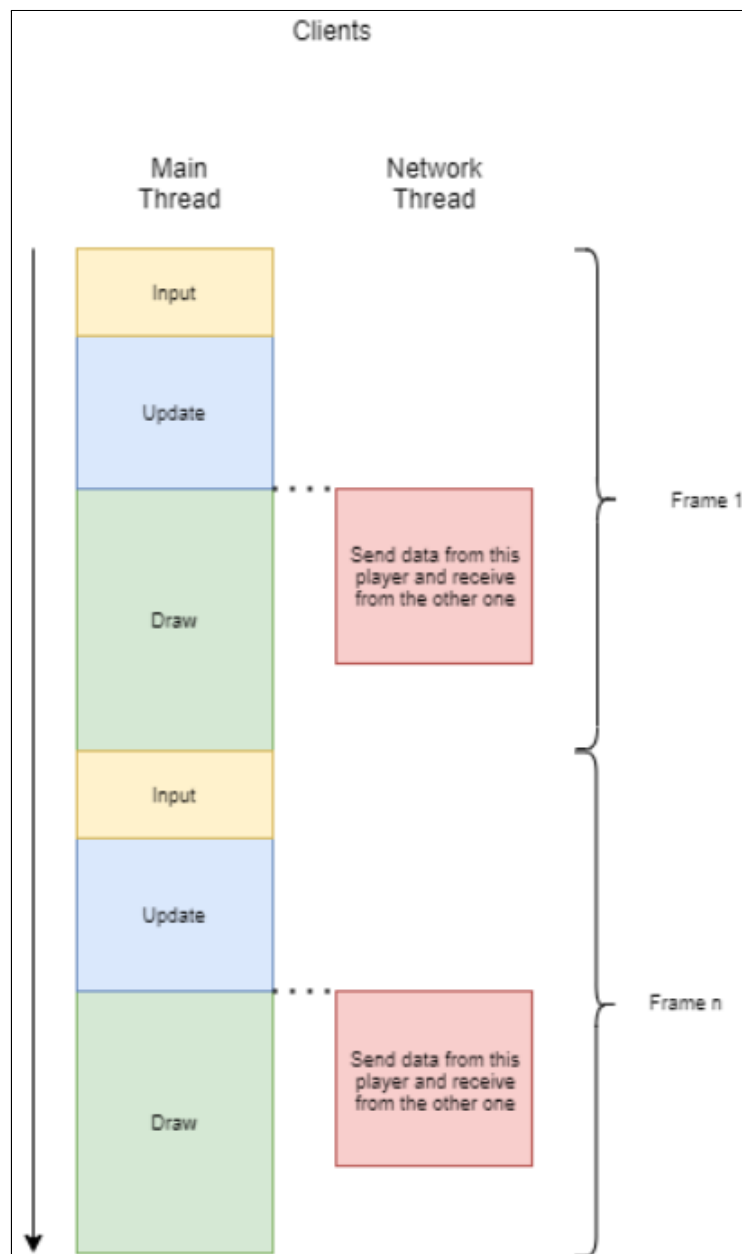
# Multithreading

## Clients

The multithreading scheme implemented in the clients works as shown in the left image. In the update stage of the main thread the command list is filled with all the things that need to be updated on the other client, right after the update stage ends, the network thread start to send the commands on the list and receive what have been sent meanwhile by the other player, using TCP means that the packages will just wait until they are received, without needing to check for missing data.

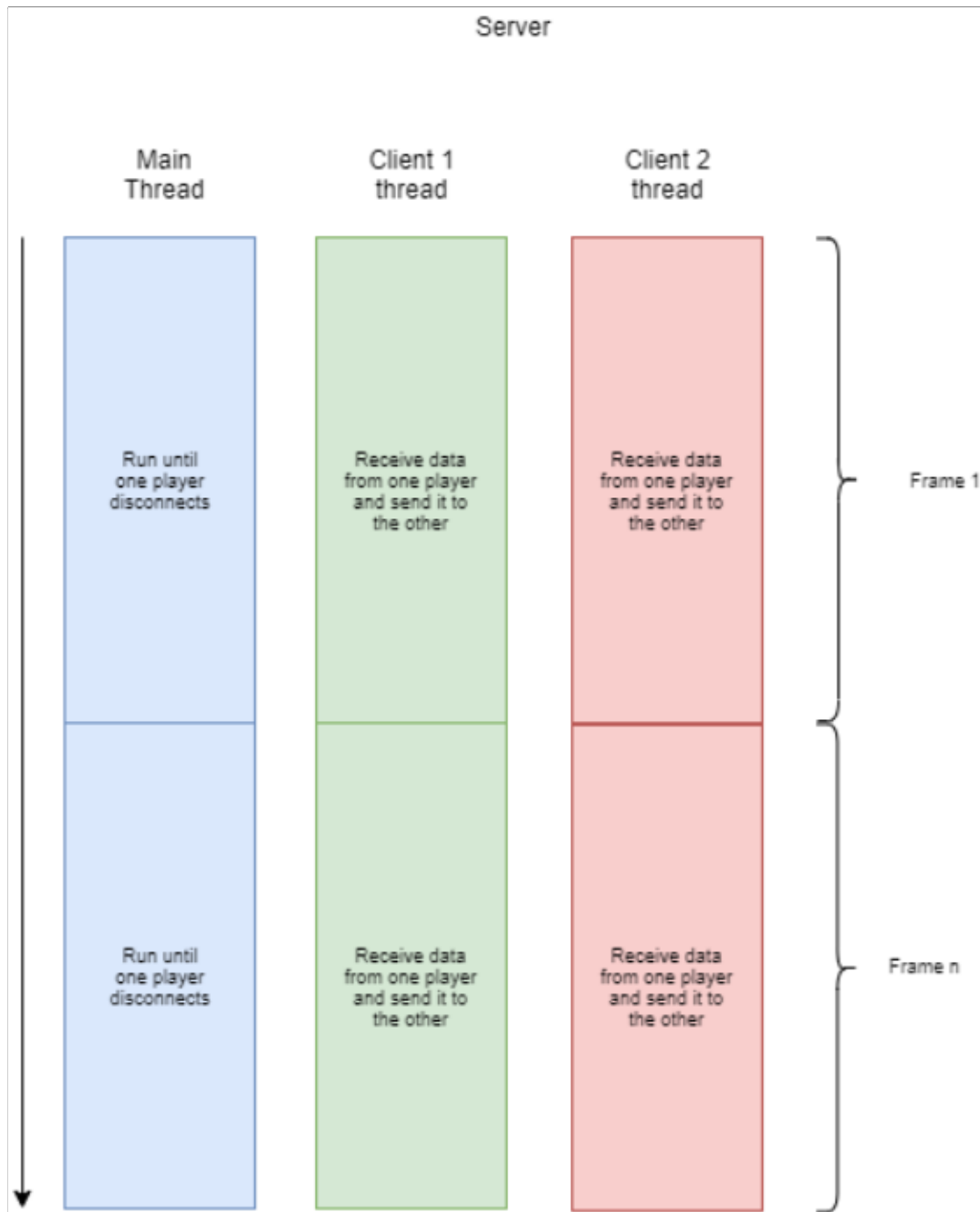
Doing this with TCP is only possible with multi-threading, otherwise the client will get stuck waiting for the packages. It is also possible due to the usage of a Command list, by sending first a unique command containing how many commands must send or receive the network thread, in this way the communications won't be blocked.

Using this scheme brings the possibility to not use mutexes, as the data will never be accessed at the same time.



## Server

The multithread scheme for the server is less complex, all the threads execute continuously, so a Mutex was used at certain points to prevent multiple threads accessing the same piece of data. The main thread waits for client connections until there are two of them connected, after that it will send a game start command to both of them and continue in an infinite loop waiting for a player to disconnect. When a client connects to the server a thread will be created, their function will be to receive the data for a player and send it to the other one, except for some data packages that are stored on the server until a condition is met.





## External dependencies

All the code for the game, networking and multithreading was specifically implemented for this project, although older code from other projects was added to include some extra functionalities.

### Own dependencies

**AI agents project from ESAT** - It is used for the deterministic path movement from the units of the players.

**Parts of an SDL 2D engine made at ESAT** - Used to have a base for the game objects to draw and update, some functionalities and modifications were added to adapt it for this project.

### External dependencies

*SDL*, *SDL image*, *SDL\_ttf*, *SDL\_imgui* for the 2D mini engine.

*ImGUI* for the UI and menus.

*GLM* for mathematics operations.

## Further work

Lots of extra ideas for the game were cut for time reasons, some of them are detailed below:

- **Better resource and unit balance.** I couldn't play too much with other people and I know all the tricks for playing, but some balancing to the costs and benefits from resources and to the tower attacks and units life, speed, etc... will benefit the gameplay totally.
- **More than two players.** The server and communication design is prepared to support more than two players. Although it wasn't done as it will require to create another map or tweak the existing and change some rules, which will have involved a lot of work dealing with a game of such size.
- **Different maps.** It will be easy to add more maps with the implemented tilemap system, although they require a lot of work balancing both sides of the map.
- **Unit and building improvements.** Like in almost all RTS games it would have been cool to improve the buildings and units by doing some research or something similar.
- **Tutorial.** An option to enable a tutorial for the first times you play will be very helpful with all the rules of the game.
- **Animations.** Adding animations to the units, buildings and scenery will improve the visual quality of the game a lot.
- **Audio.** Sound effects and music will help providing more feedback to the user. For example when an enemy building is created or when a unit reaches the enemy castle.