ECOLE DES MINES DE SAINT ETIENNE
CAMPUS G. CHARPAK PROVENCE

Rapport

System Programming - Client-Server Application Project

Gevorg ISHKHANYAN
Gaëtan Houllier

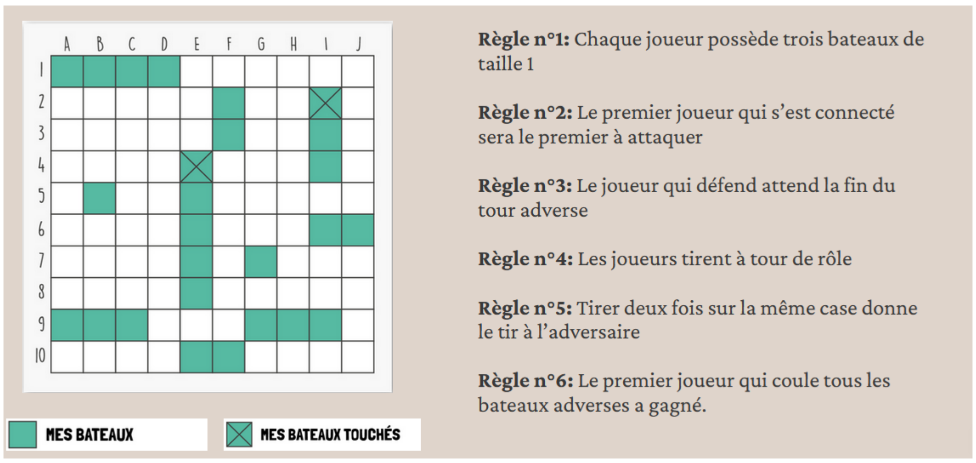
15 septembre 2024

Table des matières

1 Introduction

As part of this project, we decided to implement a battleship game. Battleship, also commonly known as "Hit and Sink," is a strategy game in which two players compete to sink each other's ships. It first appeared around World War I and consists of two phases. The first phase involves each player placing their ships on a **10x10 grid** visible only to themselves. In our case, we will limit it to **3 ships**. Then, each player takes turns choosing a square. If the square contains an opponent's ship, the ship is hit ; otherwise, the shot falls into the water. The game ends when one of the players has lost all their ships.

Therefore, in this project, the goal is to allow two players, also known as clients, to connect to the same network and face each other in a game of battleship. In this project, we will detail two codes : the **client.c** code and the **serveur.c** code, which will allow the organization of this iconic game between two players. Figure 1 provides a graphical illustration of the gameplay and outlines the rules we chose to adopt.



Règle n°1: Chaque joueur possède trois bateaux de taille 1

Règle n°2: Le premier joueur qui s'est connecté sera le premier à attaquer

Règle n°3: Le joueur qui défend attend la fin du tour adverse

Règle n°4: Les joueurs tirent à tour de rôle

Règle n°5: Tirer deux fois sur la même case donne le tir à l'adversaire

Règle n°6: Le premier joueur qui coule tous les bateaux adverses a gagné.

FIGURE 1 – Graphical illustration of a battleship grid with the adopted rules

2 Project Structure

2.1 The serveur.c Code

The **serveur.c** code manages the flow of the game. Based on what it receives from the clients, it determines which message to display on the clients' terminals. It is composed of four parts.
This diagram perfectly describes the parts of the server code :

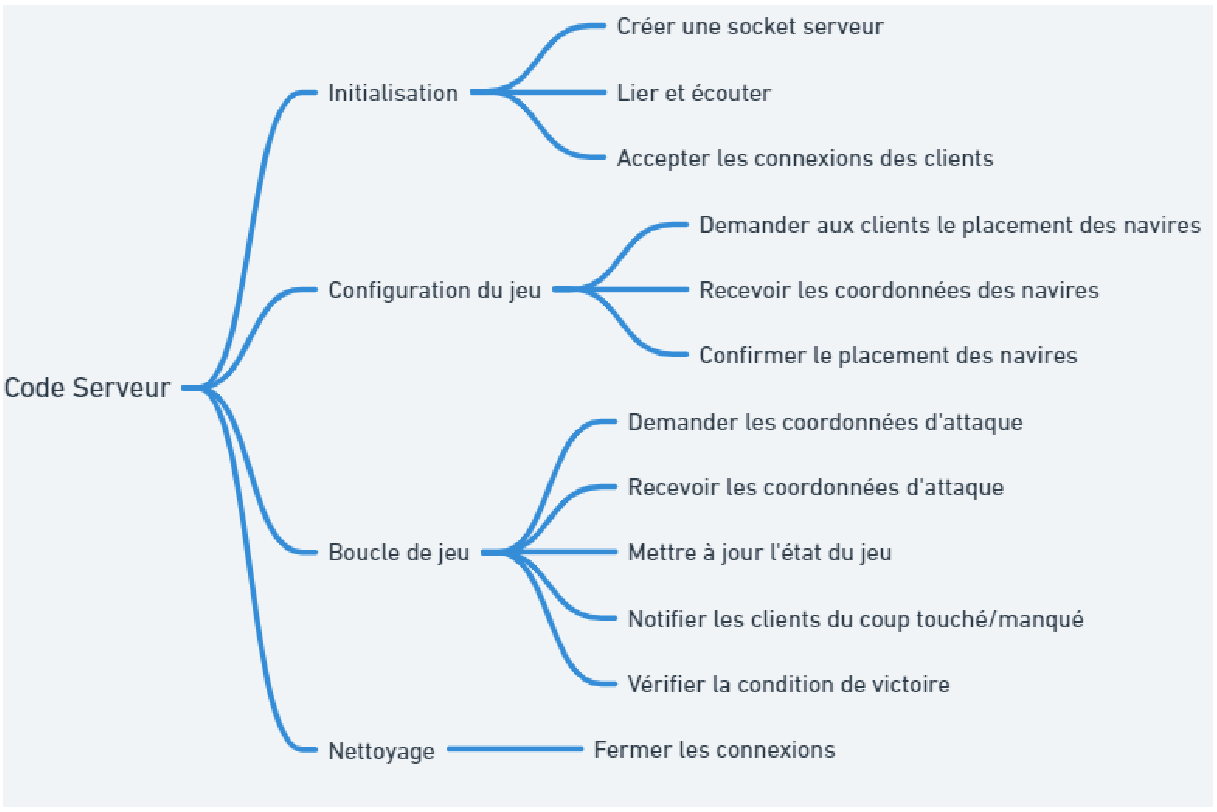

Code Serveur
Initialisation
Créer une socket serveur
Lier et écouter
Accepter les connexions des clients
Configuration du jeu
Demander aux clients le placement des navires
Recevoir les coordonnées des navires
Confirmer le placement des navires
Boucle de jeu
Demander les coordonnées d'attaque
Recevoir les coordonnées d'attaque
Mettre à jour l'état du jeu
Notifier les clients du coup touché/manqué
Vérifier la condition de victoire
Nettoyage
Fermer les connexions

FIGURE 2 – Diagram of the **serveur.c** code operation

2.2 The client.c Code

The **client.c** code manages all communications between the player and the server, which provides directives. It retrieves input from the client on their machine, sends it to the server, which in turn gives further instructions. It consists of three parts representing the three stages :

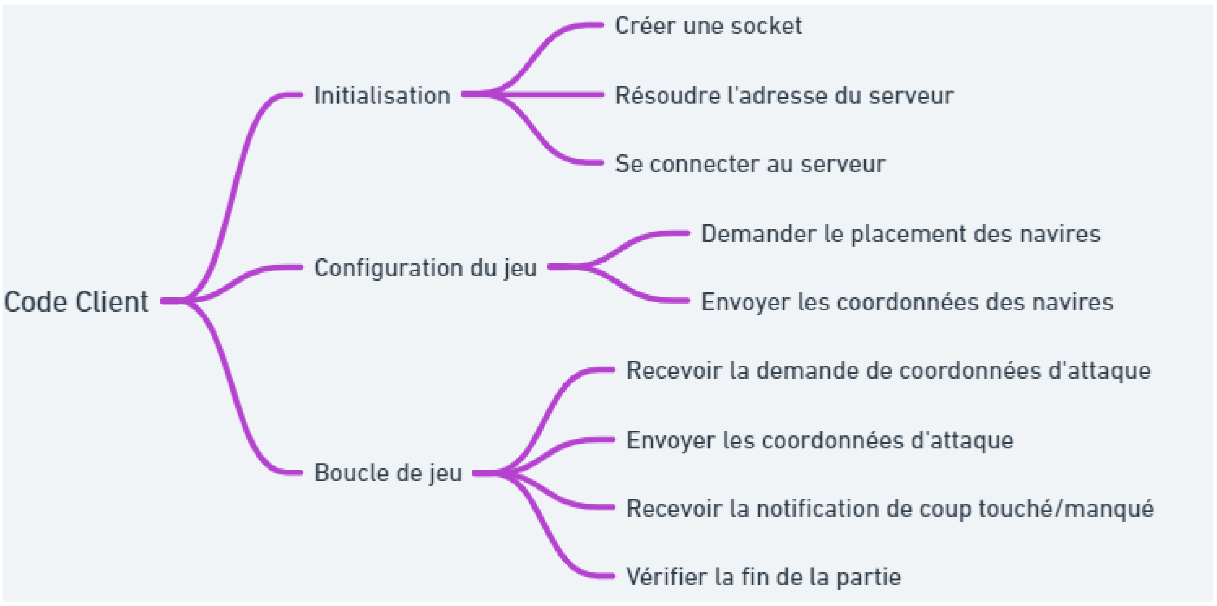

Code Client
Initialisation
Créer une socket
Résoudre l'adresse du serveur
Se connecter au serveur
Configuration du jeu
Demander le placement des navires
Envoyer les coordonnées des navires
Boucle de jeu
Recevoir la demande de coordonnées d'attaque
Envoyer les coordonnées d'attaque
Recevoir la notification de coup touché/manqué
Vérifier la fin de la partie

FIGURE 3 – Diagram of the **client.c** code operation

2.3 Overall Architecture of the Project

With an overview of the **client.c** and **serveur.c** codes, it is now possible to outline the project's overall structure. It is necessary to establish proper connections between the two codes so that the server can provide appropriate instructions to the players, and the client can relay the necessary information. Figure 4 illustrates when the client and server send and receive information to ensure smooth gameplay.

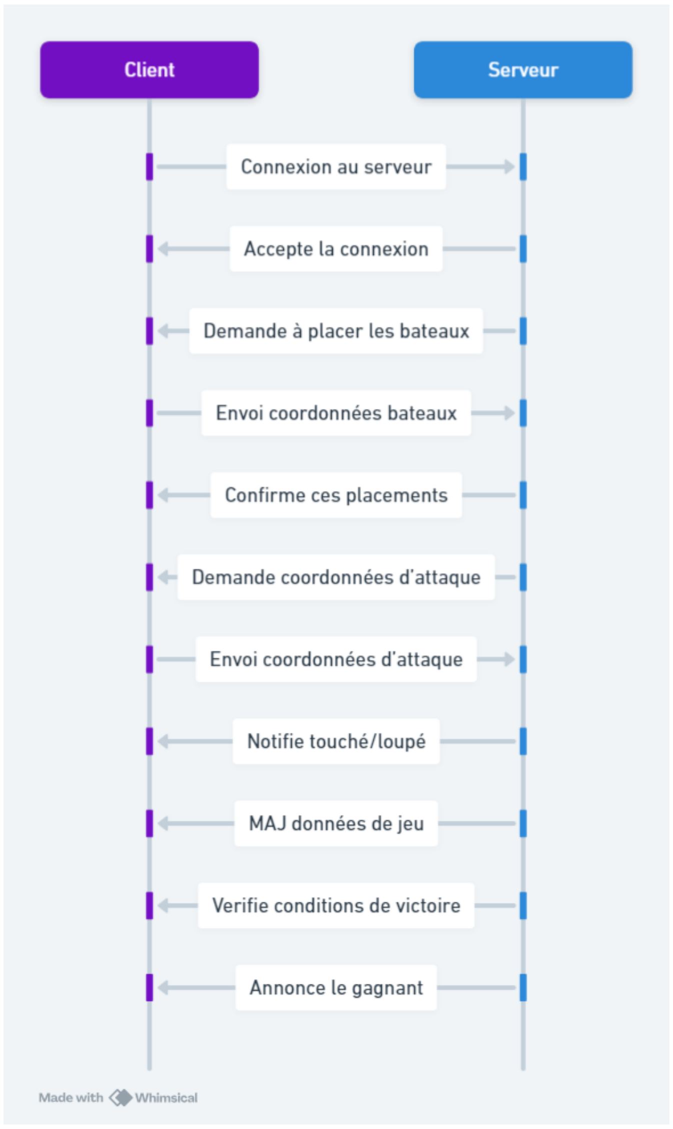


FIGURE 4 – Sequence diagram of the code operation

After compiling both codes and launching the programs, the client requests to connect to the server, which accepts. Once the player is accepted, the server prompts them to place their ships. In response, the **client.c** code sends the chosen coordinates for the ships, which are then validated by the server. Once both players have placed their ships, the server orchestrates the attack and defense phases for each player, continuously informing them after each phase about the status of their ships and the opponent's ships. Finally, when the victory conditions are met, the server announces the winner to both players.

3 User Guide

For the game to function properly, the code must first be compiled, then the **serveur_bataille_navale** program must be launched in the first terminal, and the **client_bataille_nav** program must be launched in two other terminals. To do this, follow these steps :

- **TO COMPILE :** Open a terminal in the Bataille_Navale folder and type '**make**' in the terminal.
- **TO LAUNCH THE PROGRAMS :**
  - Open three terminals in the same Bataille_Navale folder
  - In the first terminal, type the following command :
    ./serveur_bataille_navale n (with n as the port number)
      (If an error occurs, try a different port.)
  - In the other two terminals, type the following commands :
    ./client_bataille_navale localhost n (with n being the same port number)
      (If an error occurs, try a different port.)
  - EXAMPLE : ./serveur_bataille_navale 2024 (in terminal 1)
      ./client_bataille_navale 2024 (in the other two terminals)

Note : Both players can place their ships simultaneously, but to avoid bugs, ensure that the second player finishes placing their ships last.

4 Conclusion

4.1 Challenges Faced

Some parts of this project posed challenges, such as :

— Managing multiple client connections as it is important to ensure smooth gameplay under various network conditions. One solution was to implement multiple threads to allow both players to fill their grids simultaneously rather than waiting for the first player to finish.

— Debugging, particularly due to the frequent use of the blocking function read(), which is difficult to debug.

4.2 Areas for Improvement

Here are some suggestions for further refining the project :

— Improve the user interface, for example, by adding a graphical interface for better user immersion.

— Develop artificial intelligence to allow a player to play alone.

— Enhance security by encrypting communication between the client and server.

— Block the terminal of a player when it is not their turn. If a player enters shots when it is not their turn, the moves are stored in their "buffer" and executed when it is their turn. Thus, the player cannot cancel their move if they change their mind in the meantime.

— Check beforehand to ensure that ships do not overlap.

4.3 Acknowledgments

We would like to thank our professor **Xavier SERPAGGI** for his interesting lessons, as well as our classmate **Bastien GARNIER** for his debugging advice.