

Socket Programming and Concurrency Assignment 2

by Rhys Davies

Submission date: 10-May-2025 01:18PM (UTC+0100)

Submission ID: 258689628

File name:

24078_Rhys_Davies_Socket_Programming_and_Concurrency_Assignment_2_664977_2113722681.docx
(175.91K)

Word count: 1235

Character count: 7229

Socket Programming and Concurrency Report – Implementation of Battleships Game Using C++

Model Code: ACCA6028

Rhys Davies | 2008896

May 2025

Contents

Socket Programming and Concurrency Report – Implementation of Battleships Game Using C++	1
Introduction	2
1. Project Scope and Objectives	2
2. Application Design	2
2.1 Client-Server Architecture	2
2.2 Game Flow	3
3. Implementation Details	3
3.1 Networking	3
3.2 Gameplay	3
4. Best Practices and Considerations	4
4.1 Modular and Reusable Code	4
4.2 Memory Safety and Robustness	5
5. Conclusion	5
6. References	6
No table of figures entries found.	
Table 1: Summary of Networking Functions and Responsibilities	3
Table 2: Gameplay Functionality	3

Introduction

This document describes the development of a simple Battleships game played in a network environment coded using C++. The program has a multithreaded server and two client programs that transmit messages using TCP. The objective is to recreate the traditional gameplay experience in an easily understood turn-based format and demonstrate the use of low-level networking features and adherence to software best practices.

1. Project Scope and Objectives

The Battleships game should:

- Allow two players to compete against each other.
- Use server side logic to manage game state and ship placement.
- Use raw socket communication using WinSock.
- Provide feedback to players in a way that is clear and user-friendly.
- Use a minimal interface to minimise complexity.
- Stay true to the well-known battleships gameplay as much as possible including the rules, ship layout, turn taking, hit and miss logic.

2. Application Design

2.1 Client-Server Architecture

Server: All gameplay logic and two socket connections are managed by a single threaded server. The server arranges each player's ships on a 10x10 grid at random after the two players have been linked. Each player then takes turns sending moves to the server, which decides whether the player's selected coordinate hits or misses. Players receive feedback from the server informing them of their hits or misses.

The server sends feedback to the client letting the players know if they hit or missed.

Client: Each player must open an instance of the client which is a console-based application. The client connects to the server, displays the player's own board and an empty enemy board, and allows the user to enter coordinates. Clients send moves and await responses from the server, updating the board only based on server instructions. The

result of the player's hits/misses are recorded on the enemy's board so they can keep track of their moves.

2.2 Game Flow

- Both players connect to the server by opening a console.
- Ships are automatically placed on each player's board by the server.
- Each player takes turns guessing coordinates of the other player's ships.
- The server checks the move, updates game state, and sends the result back to the client.
- The game continues until one player sinks all enemy ships and wins the game.

3. Implementation Details

3.1 Networking

The application uses WinSock for TCP communication using the following.

Table 1: Summary of Networking Functions and Responsibilities

Component	Functions Used	Purpose
Server	bind(), listen(), accept()	Sets up the server socket and listens for incoming connections and accept clients.
Client	connect()	Establishes a connection to the server.
Server & Client	send(), recv()	Exchanges game-related messages including moves, results and board layout updates.
Server	Separate game loop thread	Allows game logic to run in independently of connection handling. [1]

3.2 Gameplay

Table 2: Gameplay Functionality

Feature	Description
Boards	Each player has two 10x10 grids. one for their own ships, one for tracking guesses of the opponent's board.
Ships	There are five ships per player of different lengths: – Carrier (5) – Battleship (4) – Cruiser (3) – Submarine (3)

	– Destroyer (2) The ships are placed randomly and do not overlap. [2]
Moves	Players enter coordinates (for example: D5). The server validates the move and processes it.
Feedback	The server sends responses after each move, for example: – “HIT: You hit a ship at D5.” – “MISS: The enemy missed at A1.” – “HIT: The enemy sank your Cruiser H3.”
Ship Sinking	Each ship has health equal to its length. When all segments are hit, the ship is sunk, and a message is sent to both players.
Turn Control	The server enforces turn order. After a player moves, they receive a “WAITING” message while the opponent receives “YOURTURN.”
Win Condition	When all ships of one player are sunk, the server sends a “GAME OVER: You win!” message to the winner and “GAME OVER: You lose.” to the loser.
Ship Visibility	Ships are only visible on the player's own board. Hits and misses are shown on the enemy board after receiving feedback from the server.
Input Validation	The server rejects invalid coordinates and repeated guesses, responding with relevant ERROR: or INFO: messages.

4. Best Practices and Considerations

The application design follows several software engineering principles to ensure its reliable, easy to understand and maintainable. Some of the principles are as follows.

4.1 Modular and Reusable Code

Functions such as:

- `parseCoord()` – safely parses coordinate strings into indices.
- `updateBoard()` – updates the appropriate board with a hit/miss marker.
- `sendTo()` – abstracts WinSock send logic.
- `printBoard()` – ensures a consistent, clear console UI.

Each of these functions are self-contained and reusable. This modularity reduces duplication, aids readability, and simplifies debugging. [3]

4.2 Memory Safety and Robustness

All buffers are explicitly set to zero before use with `memset()`, avoiding artefacts from residual memory. The server checks for disconnections by validating the result of `recv()` calls, and `WSACleanup()` is always called to release resources cleanly at program termination. Unexpected behaviour such as the client disconnecting mid-game is handled without crashing the server. [4]

5. Conclusion

This project successfully delivered a functioning two-player version of the classic Battleships game, built using C++ and WinSock. A reliable turn-based gameplay was developed using socket communication and via a client-server model, with the server being responsible for all core logic and coordination.

Throughout the development process, it was important that a consistent game state was maintained and that there was consistent communication between clients, and correct validation of all player inputs. The code was structured in a modular fashion with clear separation between the networking and gameplay components. This contributed to code that is relatively easy to read and understand which enhances the maintainability of the application.

The application was console based and even though there were specially designed graphical interfaces or extended multiplayer capabilities, the final result successfully replicates the traditional board game experience. The interface is simple, but it has enough information regarding hits and misses to enable effective gameplay and enable players to inform their hit strategy. The completed system provides a strong foundation for future developments with and it demonstrates good practice methods in both network programming principles and practical software design.

6. References

- [1] Guided Hacking, "C++ Winsock Networking Tutorial - Introduction," 16 February 2019. [Online]. Available: <https://www.youtube.com/watch?v=GiOgWfmWzPY>. [Accessed 05 May 2025].
- [2] WikiHow, "Battleship Game Setup, Rules & Strategy: A Complete Guide," 25 April 2025. [Online]. Available: <https://www.wikihow.com/Play-Battleship>. [Accessed 07 May 2025].
- [3] Medium, "Mastering User-Defined Functions in C++: Simplify and Modularize Your Code," 28 Jan 2025. [Online]. Available: <https://medium.com/@pkgmalinda/mastering-user-defined-functions-in-c-simplify-and-modularize-your-code-92608f6413ca>. [Accessed 08 May 2025].
- [4] J. Radigan, "-memory-safe C++ - Jim Radigan - CppCon 2022," 11 January 2023. [Online]. Available: <https://www.youtube.com/watch?v=ml4t-6bg9-M>. [Accessed 09 May 2025].
- [5] Microsoft, "Getting started with Winsock," [Online]. Available: <https://learn.microsoft.com/en-us/windows/win32/winsock/getting-started-with-winsock>. [Accessed 08 May 2025].
- [6] GeeksforGeeks, "Socket Programming in C," 23 April 2025. [Online]. Available: <https://www.geeksforgeeks.org/socket-programming-cc/>. [Accessed 08 May 2025].

Socket Programming and Concurrency Assignment 2

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

A good effort at a battleships game, well designed and implemented. Well done, Rhys.

80/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7
