CSC 470: CS III
Project #3: Battleship Game, Network Edition

Objective:

Practice developing a large C++ application, networking, and git.

Starter Code:

The GitHub Classroom assignment includes a template repository with the following files:

- Completed **server** application
  - network_packet.h # DO NOT MODIFY THIS FILE
  - server.cpp # DO NOT MODIFY THIS FILE
- Starter **test_client** application
  - test_client.cpp
  - Makefile
  - .gitignore

Project Details:

Design and implement the classic Battleship game. Your game must support network play.

Network Game Server

Upon each new network connection, the game server replies with a CONNECT packet containing the game client's player number.

Otherwise, the game server simply retransmits each received packet to every connected game client, except the game client which originated the packet.

Network Game Client

By default, the game client connects to a game server running on "localhost". The game client can also connect to a game server running on a different host by specifying the host address as the first command line parameter when starting the game client.

The Board

You shall implement a Board class that represents the state of the 10x10 game board. Your game will need to represent the 2 grids from the classic game:

- The **ocean grid**, on which you place your 5 ships and record "hits" by your opponent's shots.

    - In the physical game, each “hit” is represented by a red peg, but it’s not necessary to record your opponent’s misses. You are welcome to represent your opponent’s misses if you wish.
- The **target grid**, which represents your shot history against your opponent’s ocean.
    - In the physical game, each “hit” is represented by a red peg, while each “miss” is represented by a white peg.

You can represent both grids within a single Board object, or you can instantiate separate Board objects representing the 2 grids, or you can define separate classes for the ocean and target grids. That design decision is up to you.

The Ships

Just as in the classic board game, your game shall support 5 ships:

- carrier 5 holes
- battleship 4 holes
- destroyer 3 holes
- submarine 3 holes
- patrol boat 2 holes

Text-Based User Interface

You are encouraged to implement a text-based user interface, with user input through the keyboard and displaying a text representation of the playing grids. Players alternate turns taking shots at their opponent’s ocean. A shot is transmitted as a network packet to the game server. After each shot, the opponent’s game client will respond with a network packet representing whether it was a “miss”, a “hit”, or a “hit and sunk”.

Graphical User Interface

Bonus credit will be awarded if a graphical display and/or graphical input is implemented.

More than 3 Players

If more than 2 game clients connect to the game server, the game should work in multiplayer mode without any additional development effort. The game server will retransmit each shot to every game client. Every game client will send its “miss”, “hit”, or “hit and sunk” report, which will be received by all other game clients. Multi-player mode should “just work”. The server will support up to 30 simultaneous game clients (defined by a constant in server.cpp).

Development Plan

Task 1

Design and implement a class (or separate classes) which represent the 10x10 ocean grid and the 10x10 target grid. Instantiate the 2 grids.

Task 2

Design and implement a user interface to initially place your 5 ships on user-specified board locations on your **ocean grid**.

Task 3

Design and implement code to display the **ocean grid** view of your board.

Task 4

Design and implement code to display the **target grid** view of your opponent's board.

Task 5

Design and implement code to receive network packets and execute them. After executing each packet, update the displayed state of the ocean and/or target grids. Note: you can use the "test_client.cpp" application to transmit packets to your game client to test it.

Task 6

Design and implement code to display a menu and accept commands from your keyboard. Send the commands as network packets to the game server.

Project Presentation:

Each student will give a short presentation describing their class project. Your goal should be to demonstrate and describe in 10 minutes (8 minutes presentation + 2 minutes Q&A) what you did and why it is interesting. Describe the guts of your project using 4-8 slides. The slides should clearly present the goals, challenges, approach, implementation, and results of your project. In addition, your presentation may include any number of screenshots showing your game in use.

You should also start a live demo on one of the computers in the Linux lab.

Written Report:

Each student should submit a six- to ten-page written final report. The written report should contain descriptions of the goals and challenges of your project. You should write detailed descriptions of the approach you've chosen, the implementation hurdles you've encountered, the features you've implemented, and the results you've generated. Please do not be vague in your written descriptions. Following is a suggested outline.

- Introduction
  - Goal
    - What did I try to do?
  - Problem Statement
    - Why is it hard?
    - How hard is it?
- Approach
  - What approach did I try?
  - Under what circumstances do I think it should work well?
  - Why do I think it should work well under those circumstances?
- Methodology
  - What pieces had to be implemented to execute my approach?
  - For each piece ...
    - Were there several possible implementations?
    - If there were several possibilities, what were the advantages/disadvantages of each?
    - Which implementation(s) did I choose? Why?
    - Which implementation(s) did I not choose? Why not?
    - What did I implement? <== Include detailed descriptions
- Results
  - What was I trying to test?
  - What experiments did I execute?
  - What different approaches/parameters did I vary in these experiments?
  - What objective measure of success did I use?
  - Provide quantitative results.
  - What do my results indicate?
- Discussion
  - Overall, is the approach I took successful?
  - What different approach or variant of this approach might be better?
  - What follow-up work should be done next?
  - What did I learn by doing this project?
- Conclusion
- Appendices (not included in the page count)
  - Concise build/install/use instructions. This might simply be your awesome README.md.
  - Representative screenshots of the running client and server applications.

What to turn in:

- GitHub Classroom repository
  - All files required to build and run the game server and game client. Your applications shall build by typing "make". There shall be no compilation or linkage warnings or errors.
  - An awesome README.md file
    - https://github.com/matiassingers/awesome-readme
    - https://github.com/abhisheknaiidu/awesome-github-profile-readme
- Canvas submission
  - A UML class diagram of the system (PDF)
  - Your detailed final report (PDF)
  - Your presentation slides (PDF)

Source Formatting Requirements:

- Start each source file with a comment containing your name, the name of the assignment, and a summary comment that explains what the file does.
- Format your code consistently.
- Comment your code (each method should describe its purpose, parameters, and return values).

Grading Rubric:

10	Design and implement a class (or separate classes) which represent the 10x10 ocean grid and the 10x10 target grid. Instantiate the 2 grids.
10	Design and implement a user interface to initially place your 5 ships on user-specified board locations on your **ocean grid**.
10	Design and implement code to display the **ocean grid** view of your board.
10	Design and implement code to display the **target grid** view of your opponent's board.
10	Design and implement code to receive network packets and execute them. After executing each packet, update the displayed state of the ocean and/or target grids. Note: you can use the "test_client.cpp" application to transmit packets to your game client to test it.
10	Design and implement code to display a menu and accept commands from your keyboard. Send the commands as network packets to the game server.
5	Proper UML class diagram
5	Block comment at top of each source file, formatting, indenting, source comments.
15	Good software development process as evidenced by GitHub commits and pushes. An awesome GitHub README.md.
10	6-10 page written project report.
5	10 minute in-class presentation and slides.