Alexander O’Callaghan

ST20066908

CIS6003 – Advanced Programming

Battleships Documentation

https://github.com/Zander-Git/Battleships-Assignment

Table of Contents

[Running the Program 3](#_Toc40098553)

[Design Choices 4](#_Toc40098554)

[UML Diagrams 5](#_Toc40098555)

[Use case 5](#_Toc40098556)

[Class diagrams 6](#_Toc40098557)

[Client Diagram 6](#_Toc40098558)

[Server Diagram 7](#_Toc40098559)

[References 8](#_Toc40098560)

# Running the Program

To run the program, please select both bat files to have them build and compile the game.

When open, the server’s interface will have a suggested port number (4444) already in place. Feel free to change this number. The client’s interface will have the same port number already placed and will also have the local I.P address already placed. If on the same machine, simply pressing “start” on the server, followed by “connect” in the client will connect the two.

Once running, please place your ships. Once you have done so, press the “ready” button and, if you’ve placed all your ships, you will notify the other player that you are ready and receive window indicating if they are also ready. If you have not placed all your ships yet, the console will output which ships you have yet to place.

Once both players are ready, the game will begin, and the client always starts first. The players will swap turns every time they miss a ship with their guess. If they hit an enemy ship they can fire again.

Once one player has hit all the other players ships, the game will display the winner and ask you to close the window.

Unfortunately, the second use case of the requirements were not met, where the server would act as a player and randomly guess coordinates and place ship. Though it wasn’t done, the already completed code is not that far from completing this as the placeShips function gives good guidance to what positions are acceptable for placing i.e not overlapping, withing bounds etc.

# Design Choices

The project is based on, and includes features, from a learning resource “ClientServerChatGUI” that was suggested to use a basis for our work. In my project, the sockets and ClientManager are the most like the resource.

While the program works, the backend took some compromises in terms of proper design structure.

If the project was to be done again, it is clear there is a lot of duplicated code that have been better managed. An example of this would be in the “SimpleClient” and “SimpleServer” implementation of how messages are received and given. Creating an abstract component for the sockets to extend would be a good solution to this.

At the start of the project, a lot of time was spent trying to implement a Model-View-Controller (MVC) approach to the structure of the classes. In the end I could not get such a structure working and abandoned it in favour of producing a working system, but the appeal of keeping classes and packages separate to their own area is very appealing. Knowing code related to the specifically the view packages is only concerned with itself and allowing a controller to provide all the connection between the view and the model logic is a very neat way of doing things.

# UML Diagrams

## Class diagrams

### Client Diagram

A close up of text on a white background

Description automatically generated

### Server Diagram

A close up of a map

Description automatically generated