# Server-side Design Decisions:

Server waits for 2 clients to connect to it, then starts the game session. Game session implements Runnable interface, so that every game session can be run in a separate thread.

When a game session is started, the server generates two random game fields for each of the players. I have written an algorithm that places players’ ships randomly. The algorithm selects random tile, checks that it is empty, checks that the tiles around are empty, randomly selects direction of ship placement and tries to put ship there, if every cell around is empty.

Classes Ship and Tile are used to represent ships and ship parts accordingly. Ship class stores number of cells alive and overall state of ship – dead or alive. Tile class stores direction, which the tile is facing.

Players’ turns are controlled by server:

1. Server notifies player 1 to start the game
2. Player 1 makes his move
3. If it was a hit, turn continues
   1. Send player 1 turn results to player 2
4. If it is a miss, turn ends
5. Notify player 2 to make his move

* After every turn server checks if one of the players has won

# Client-side Decisions:

When client is run, the player chooses to connect to server and start the game new instance of GameBoard class is created and run in a separate thread – this had to be done, because if it was run on the same thread, the new JFrame with game contents would be displayed empty for some reason.

The graphical interface is constructed, then the game loop is called – it receives messages from server, then using a set of if statements it does appropriate action for specific commands (like when server sends “continue” command, client activates listeners and waits for player to make his move).

Player input is detected using MouseListener, so whenever it is this player’s turn and he clicks mouse data is sent to server.

# GUI Issues:

The client generates a small window when it is run, that has a button to connect to server and label to display status information. When a game is started, a new window is generated with game board contents.

Game board consists of the following elements:

* Upper panel, divided into two panels with grid layout
* Inside each panel there are multiple labels with icons, to represent current ships’condition (players can clearly see which ships are still alive) – both player’s and opponent’s.
* Bottom panel with grid layout
* Inside each cell there is a label with icon to represent a tile

I have used 17 image files to display different conditions of a tile and 14 image files to represent different conditions of ships.

Boolean variable is used to disable mouse listener while it is not player’s turn.

# Network protocol details:

Client-server networking model with sockets is used, as this was required by the specifications of the coursework.

ServerSocket and two Sockets (one for each player) are on server side. One socket is at client side.

ObjectInputStream and ObjectOutputStream are used for client-server communication, because they allow multiple sets of data to be passed in one message, thus reducing number of times sending and receiving messages.

# Exception Handling Approach:

For every exception that is caught in the code, the message is printed into console, so that the problem could be identified straight away. For some occasions, like when server is shut down while a game is in progress, messages are sent to clients to inform them. Not all such exceptions have been covered though.

# Room for improvement:

Message exchanging between clients and server can be moved to a separate thread, so that messages could be sent/received whenever it is necessary. Thus exceptions could be handled much more easily and the overall response time would be reduced.

There is no manual ship placement mechanism in the game. This could be achieved by using the same JLabels with icons, MouseListener along with a modified version of the code, which is checking if randomly generated ship position is correct.