School of Information and Physical Sciences
University of Newcastle
SENG4500 –Network & Distributed Computing
Assignment 2

Due using the submission facility of the Canvas Assignment facility: 11:59 p.m. 23/10/22

In this assignment we are going to build a peer-to-peer client implementation of the game Battleships. To understand the rules of Battleships I suggest you first review the guide at https://static.kidspot.com.au/cm_assets/79310/battleship-game-highres1-pdf-20160118152656.pdf.

We will be playing on a 10x10 grid with 5 ships as defined in the list of ships at the bottom of the PDF above. Numbers across the top and letters down the side. When referencing a position in the grid we will reference the letter first, then the number – i.e. A3 will be the 1st row, 3rd column of the grid.

The high-level requirements for the assignment are as follows. These are broken down in more detail below.

1. The "clients" must be able to locate each other using a UDP-based broadcast packet.
2. A newly started client will monitor the network for 30 seconds looking for "NEW PLAYER" messages being sent to the network broadcast address.
3. If no messages are spotted then it will send one itself, before looping back to step 2.
4. If a "NEW PLAYER" message is spotted then the receiver (Player 2) will connect back to the sender (Player 1) over TCP to establish the command channel for the game.
5. At this point both clients can setup their data structures for the grid and place their boats.
6. The clients will then take turns sending "Fire" messages over TCP, with Player 1 going first.
7. The clients will respond with either "MISS", "HIT", "SUNK" or "GAME OVER" messages.

The clients will be command-line only. No GUI is expected. The clients will expect two parameters to be passed in as command-line arguments – the BROADCAST address and the BROADCAST port number to be used in the "NEW PLAYER" messages.

The ships will be placed randomly, taking care not to overlap ships. This can be done however you like – even just an ugly loop that keeps picking a random starting position and a direction until it manages to find a placement that doesn't clash with an existing ship.

The only user-input we are wanting from the client is for them to enter in the co-ordinates fore the FIRE command. You can implement this UX however you like – but I imagine you would just use standard in/standard out at the terminal.

As for the output to the user, you should at each turn print out two grids – one showing the status of your ships and one showing the status of the responses received from the other playing – i.e. the miss and hit's. You can print these two grids however you like. We also expect basic output on the screen to let us know how the game is progressing in response to the messages defined below.

Note that we are only expecting a SINGLE CODE BASE be submitted that can handle being both Player 1 and Player 2 depending on whether the code sent or received the "NEW PLAYER" message in steps 2 and 3.

Below is a table of message definitions that match the requirements above.

Message Type	Channel	Definition
NEW PLAYER	UDP	This message contains the string "NEW PLAYER", followed by a colon, followed by the TCP port number that the client is listening on for the game command channel. The TCP port number will be randomly selected by the client from within a range of 5000-6000. For example – `NEW PLAYER:5001`
FIRE	TCP	Player 1 sends the first fire message. Afterwards they wait to receive one of the messages below, before waiting to receive Player 2's FIRE message. We then loop. This message contains the string "FIRE", followed by a colon, followed by the grid position.

		For example – `FIRE:A3`
MISS	TCP	The players respond with a MISS message if the received FIRE message lands in a grid position that does not contain a ship. This message contains the word "MISS", followed by a colon, followed by the grid position we just checked For example – `MISS:A3`
HIT	TCP	A player responds with a HIT message if the received FIRE message lands in a grid position that DOES contain a ship but that ship is NOT fully sunk by this hit. This message contains the word "HIT", followed by a colon, followed by the grid position that was just hit. For example – `HIT:A3`
SUNK	TCP	A player responds with a SUNK messasge if the received FIRE message lands in a grid position that DOES contain a ship and it's the last grid position that has fully sunk that ship. This message contains the word "SUNK", followed by a colon, followed by the grid position we checked, followed by another colon, followed by the name of the ship that was sunk. For example – `SUNK:A3:Battleship`
GAME OVER	TCP	A player responds with a GAME OVER messasge if the received FIRE message lands in a grid position that DOES contain a ship and it's the last grid position that has fully sunk that ship, and it's the last ship to be sunk for that player. This message contains the words "GAME OVER", followed by a colon, followed by the grid position we checked, followed by another colon, followed by the name of the ship that was sunk. For example – `GAME OVER:A3:Battleship` At this point both clients will exit after gracefully shutting down.

You are not required to implement any error handling beyond ensuring that at the user-interface level the coordinates that the user enters are within the valid range. There is no "ERROR" message defined at the TCP level.

This assignment is worth 20 marks of your final result for the course.

You are to submit all your source code in a ZIP, including a README with instructions on how to run your game and any issues with your final submission.

Dr. Mark Wallis