Introduction to computer networking

First part of the assignment

Academic year 2018-2019

Abstract

In this assignment, students will have to implement a client-server application using Java Sockets.
The server is a game platform, i.e. it allows a user to play the game of Battleship. The client will interact with the user to display the game status and ask for new location to hit, and discuss with the server to verify the location guessed by the user, or to request a new game.
Students will work alone using the Java language.

Sections 1 and 2 define your assignment objectives. Sections 3 and further are an executive summary of the technologies we use.

The **Hard Deadline** is March, 24th, 2019.

1 Battleship platform

As stated in the abstract, you will implement a Battleship platform using Java Sockets. The server waits for TCP connections on a given port, receives requests through that

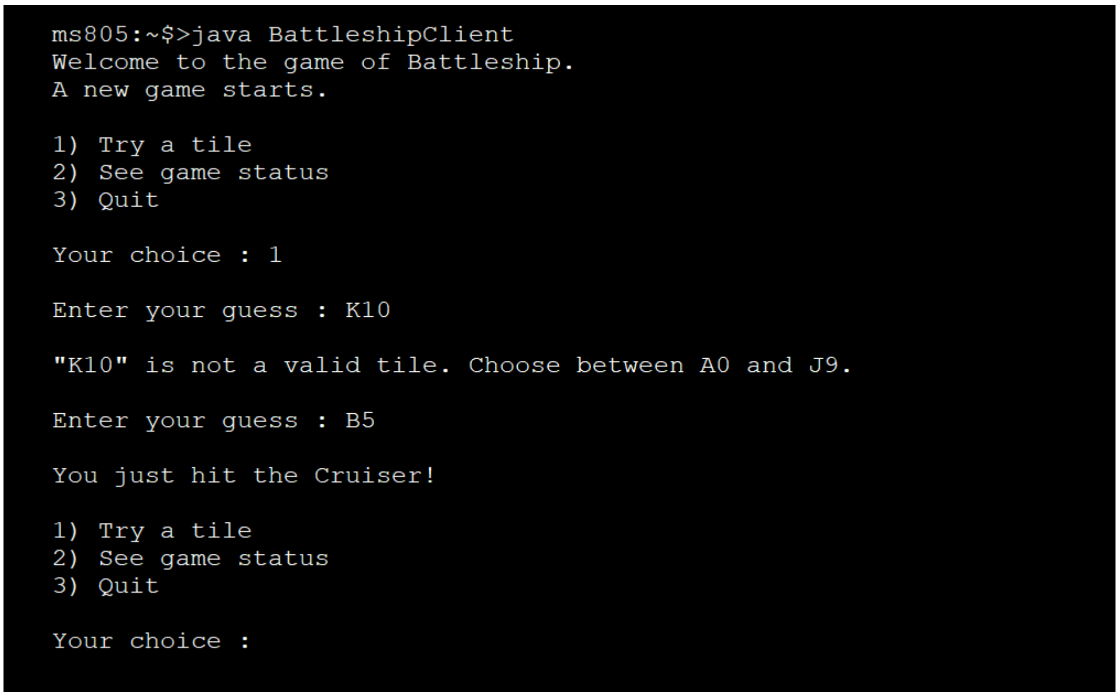
Image taken from www.youtube.com

connection when established, potentially updates its internal state accordingly, and responds with a formatted response following the custom-made Battleship protocol.

The server will be up for new connections and be able to handle concurrency. Each client plays its own game. The session scope is the TCP connection (identified by the client IP address and port).

The client will interact with the user using terminal I/Os, send requests to the server, read its responses, and update its display accordingly.

Figure 1 shows an example of a potential output from the client, interacting with the user and with the server.



```
ms805:~$>java BattleshipClient
Welcome to the game of Battleship.
A new game starts.

1) Try a tile
2) See game status
3) Quit

Your choice : 1

Enter your guess : K10

"K10" is not a valid tile. Choose between A0 and J9.

Enter your guess : B5

You just hit the Cruiser!

1) Try a tile
2) See game status
3) Quit

Your choice :
```

Figure 1: Example of a potential console output.

1.1 Rules

On game start the server will place 5 ships at random on the 10x10 play area. Each ship occupies a number of consecutive tiles on the grid, arranged either horizontally or vertically. Ships cannot overlap. Ships are the following : Carrier (5 tiles), Battleship (4 tiles), Cruiser (3 tiles), Submarine (3 tiles) and Destroyer (2 tiles).

At each iteration, the user selects a tile where (s)he thinks a ship is present. The client software receives the tile position from the user and sends it to the server. The server responds with the name of the ship that was hit, or with “splash”, meaning that no ship was present at this position.

The game ends either when all ships have been sunk, or when at least one ship is not sunk after 70 tries, or if the user quits prematurely.

Note that this is a simplified version of Battleship in the sense that the user does not have ships of his own, (s)he just tries to sink the server ships in the minimum amount of tries.

1.2 Launch

The software is invoked on the command line with no additional arguments:
`java BattleshipServer` for the server
and
`java BattleshipClient` for the client.

The server listens on port 2*xxx* – where *xxx* are the last three digits of your ULg ID.

1.3 Incorrect or incomplete commands

When dealing with network connections, you can never assume that the other side will behave as you expect.

It is thus your responsibility to check the validity of the client’s request (and respond with the appropriate error code if something goes wrong) and to ensure that a malevolent person won’t make your server freeze by initiating a TCP connection and keeping it open for an indefinite period of time (see Section 4 for a counter-measure).

2 Guidelines

- You will implement the programs using Java 1.8, with packages `java.lang`, `java.io`, `java.net` and `java.util`,
- You will ensure that your program can be terminated at any time simply using CTRL+C, and avoid the use of ShutdownHooks
- You will not manipulate any file on the local file system.
- You will ensure your main class is named BattleshipServer (resp. BattleshipClient), located in BattleshipServer.java (resp. BattelshipClient.java) at the root of the archive, and does not contain any `package` instruction.

- On the server output console, you will display the position of the ships, for easier testing.
- You will not cluster your program in packages or directories. All java files should be found in the same directory.
- You will ensure that your program is fully operational (i.e. compilation and execution) on the student's machines in the lab (ms8xx.montefiore.ulg.ac.be).

Submissions that do not observe these guidelines could be ignored during evaluation.

Your commented source code will be delivered no later than 24th of March (**Hard deadline**) to the Montefiore Submission platform (`http://submit.run.montefiore.ulg.ac.be/`) as a .zip package.

Your program will be completed with a .pdf report (in the zip package) addressing the following points:

Software architecture: How have you broken down the problem to come to the solution? Name the major classes and methods responsible for requests processing.

Multi-thread coordination: How have you synchronized the activity of the different threads?

Limits: Describe the limits of your program, esp. in terms of robustness.

Possible Improvements: This is a place where you're welcome to describe missing features or revisions of these specifications that you think would make the server richer or more user-friendly.

3 The Battleship Protocol

The Battleship Protocol (BP) is a custom-made protocol designed for this assignment only. This protocol is *binary-oriented*, meaning that all exchanges are custom packed and not intended to be human-readable (as opposed to "text-oriented" protocols such as HTTP that is based on readable lines of text).
Each request and reply is made of a header of 2 bytes, possibly followed by more data.

Figure 2 is a visual summary of the protocol.

3.1 BP Header

For each new request and each response, the message is preceded by a header of 2 bytes. The description of each byte follows.

1. A version number, allowing several versions of the protocol to be supported. In this assignment, the protocol version is 1.
2. A number identifying the type of message.

3.2 BP Method field

The second field in the header determines the meaning of the request/response type. The possible values, and their meaning, are described in the following subsections.

3.2.1 Client requests

- 0 : Request a new game.
- 1 : Request a shoot at a given position. The position is given after the header. You will take advantage of the fact that only 100 values can be given and encode this in only one byte.
- 2 : Request the list of already proposed tiles and, for each one, the name of the ship that was hit if any.

3.2.2 Server response

- 1 : A new game was started.
- 2 : The tile position was received. The following byte indicates what ship has been hit, or 0 if no ship was hit.
- 3 : List request received. The next byte indicates how many positions were already tested. For each one, the reponses contains 1 additional byte, the ship that was hit (or 0 if none).
- 4 : Request error.

4 Multi-Threading

A very simple solution for the server's code would be to accept a connection, handle the request, then go back to accepting new connections, and so forth.

Now, imagine that a client behaves badly and sends an incomplete request. The server would then be blocked and would not be able to handle new connections.

A simple solution to handle this problem is to separate the code logic into separate threads. The main thread will loop on the `accept()` method and, when it succeeds, forward the task to a new Thread. That way, even if one client is erroneous, it will only impact one of the server's Threads and other clients will be served correctly.

We will see in the second part of the assignment that this method also has a flaw, but will be considered as acceptable for this first part.

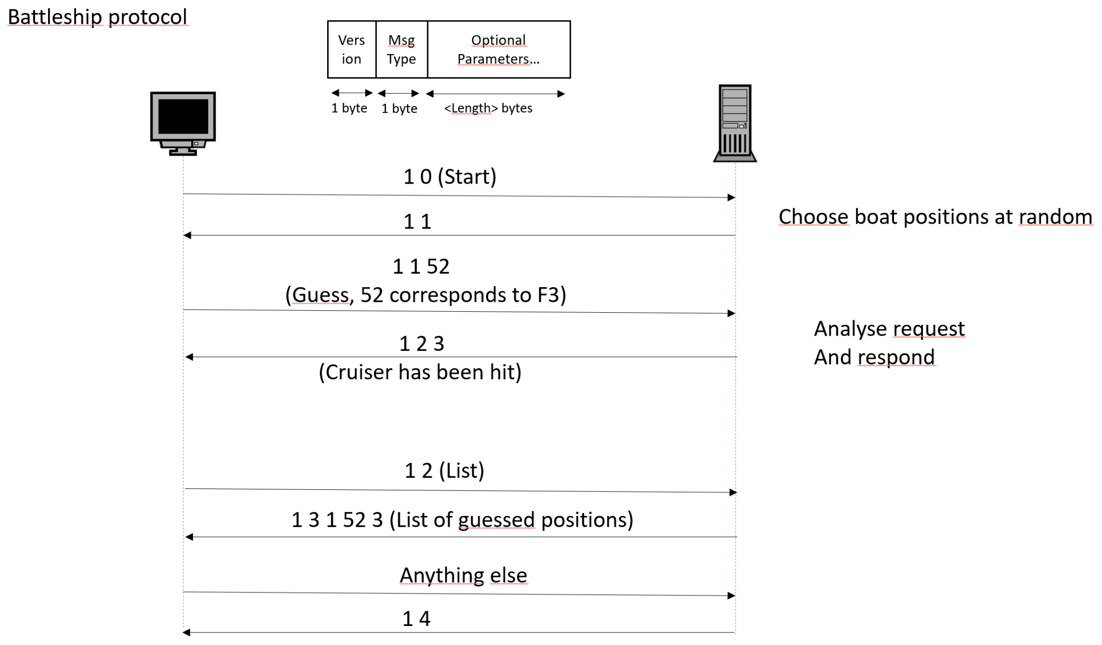


Figure 2: Summary of the Battelship Protocol.

5 Stream-oriented protocols (TCP)

The BP protocol relies on the TCP protocol, which is stream-oriented.

This has at least three implications:

1. You can be sure that the connection is lossless and that there won't be any packet re-ordering.
2. The sender could be *buffering* the output, i.e. the data that is requested to be sent might not have been effectively sent yet. This can be solved by calling the `flush()` method on the `OutputStream`. Nagle's algorithm can also prevent a packet from departing immediately. It can be deactivated using `setTcpNoDelay(true)`.
3. **The data requested to be sent with one call to a write method will not necessarily be received with only one read method from the other side.** Several reads could be necessary to recover the entire information. This is because we are reading from a stream, not messages (as we would with UDP, for instance). We thus have to find a way to delimit message boundaries on the stream. BP achieves this by having a constant-size header, whose "type" field determines how many bytes, if any, will follow. In practice, you should thus use a class that takes multiple reads into account (such as `BufferedInputStream`) or perform multiple reads if using a low-level `read`.

Good programming...