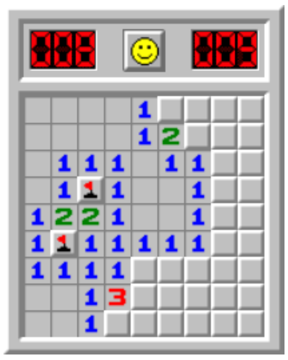
Introduction to computer networking

First part of the assignment

Academic year 2024-2025
Updated 04/10/24


[1]

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 MineSweeper.
The client will interact with the user to display the game status, reveal cells and flag cells.
Students will work alone using the Java language.

Sections 1 and 2 define your assignment objectives. Sections 3 and 4 summarize the technologies we use. Section 5 describes the practical guidelines.

The **Hard Deadline** is October 27th, 2024.

1 Minesweeper platform

As stated in the abstract, you will implement a Minesweeper platform using Java Sockets. The server waits for TCP connections on a given port, receives requests through that connection when established, potentially updates its internal state accordingly, and responds with a formatted response following the custom-made Minesweeper protocol.

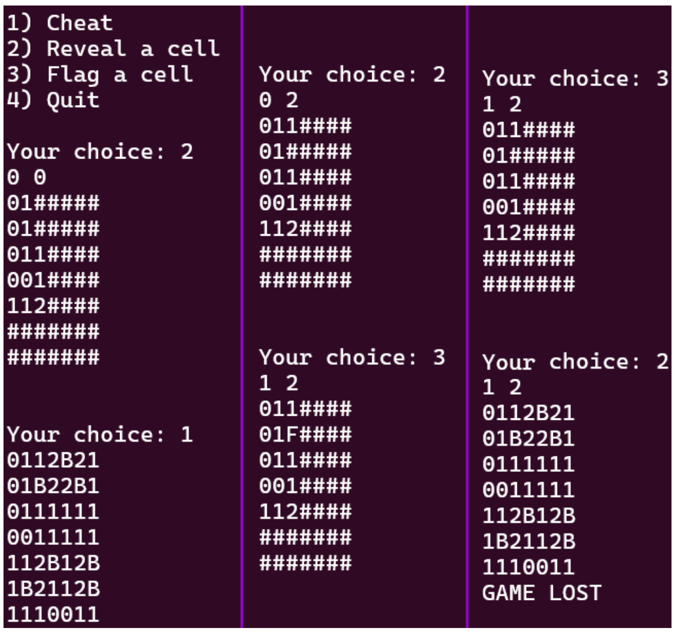
The server will be up for new connections and be able to handle concurrency. Each client plays its own game. The session scope is the TCP connection (identified by the

[1]Image taken from minesweeper.online

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. You are free to design your own user interface as long as the message exchange with the server respect the Minesweeper Protocol (cfr. 2).



```
1) Cheat
2) Reveal a cell
3) Flag a cell
4) Quit

Your choice: 2
0 0
01#####
01#####
011####
001####
112####
#######
#######


Your choice: 1
0112B21
01B22B1
0111111
0011111
112B12B
1B2112B
1110011
```

```
Your choice: 2
0 2
011####
01#####
011####
001####
112####
#######
#######


Your choice: 3
1 2
011####
01F####
011####
001####
112####
#######
#######
```

```
Your choice: 3
1 2
011####
01#####
011####
001####
112####
#######
#######


Your choice: 2
1 2
0112B21
01B22B1
0111111
0011111
112B12B
1B2112B
1110011
GAME LOST
```

Figure 1: Example of a potential console output.

1.1 Rules

1.1.1 Overview

On game start, the server will generate a 7x7 grid and randomly place 7 bombs within it. The objective of the user is to reveal all the cells that are not bombs (i.e. locating all the bombs).

At each iteration, the user can decide to either reveal a cell or flag a cell. While revealing cells makes the game progress, flagging cells is just a visual help for users. The client software can help the user build its request and send it to the server. Upon receiving a request, the server updates the game status accordingly and sends back the updated grid to the client.

The game ends either when all the safe cells have been revealed, or when a bomb has been revealed, or if the user quits prematurely.

1.1.2 Grid creation

At the start of each game, the server creates a 7x7 grid, randomly placing 7 bombs in different cells. Each cell can be identified by its row and column. Rows and columns are 0 indexed. For example, Table 1 has cell (1,5) that contains a hint 2 and cell (3,0) contains a F (for flag).

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	1	1	2	1
2	1	1	0	1	#	#	#
3	F	1	0	1	2	#	#
4	#	2	1	1	#	#	#
5	#	#	#	#	#	#	#
6	#	#	#	#	#	#	#

Table 1: Minesweeper grid example

The server ensures the following:

- Random Bomb Placement: There are 7 bombs that are placed randomly across the grid.
- Hint Placement: Each cell that is not a bomb contains the number of adjacent (sides and diagonals) cells that contain a bomb
- Safe First Reveal: The first cell revealed by the user will never contain a bomb. Thus, you should place the bomb after the bombs at the time of the first reveal. Be sure to respect the reveal rules cfr 1.1.3 for the first reveal.

1.1.3 Reveal

When a cell (*row*, *col*) is revealed the outcome is one of the following:

- The cell was already revealed and nothing happens.
- The cell contains a hint that is not a 0, the cell is revealed.
- The cell contains a hint that is 0, the cell is revealed and all adjacent cells are revealed following the rules of this section.
- The cell contains a bomb, all cells are revealed and the game is lost.

If all cells that do not contain a bomb are revealed, the game is won.

1.1.4 Flagging

As previously mentioned flagging cells is just a visual help for users. They can even flag cells before their first reveal. When a cell (*row*, *col*) is flagged the outcome is one of the following:

- The cell was already revealed and nothing happens.
- The cell is unrevealed and unflagged, the cell is flagged.
- The cell is flagged, the cell is unflagged.

1.2 Launch

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

The server listens on port $2xxx$ – where xxx are the last three digits of your ULiege student 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 3 for a counter-measure).

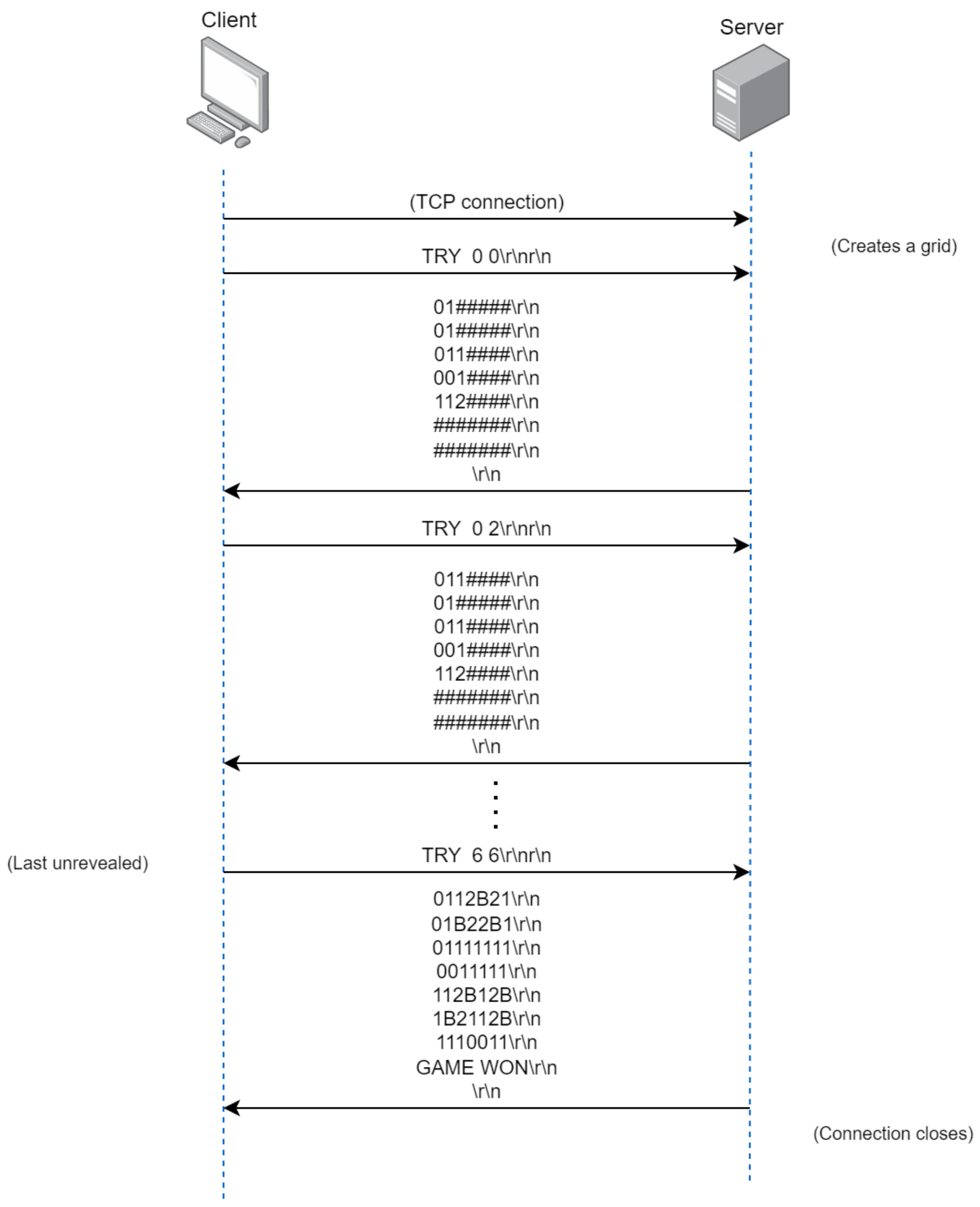
2 The Minesweeper Protocol

The Minesweeper Protocol (MP) is a custom-made protocol designed for this assignment only. This protocol is *text-oriented*, meaning that all exchanges are intended to be human-readable (as opposed to "binary" protocols such as IP, TCP and BitTorrent that feature custom information packing). This protocol is a multiline protocol, each request/response consists of a few lines of **uppercase** characters ending with "\r\n" (CRLF) and ends in an empty line (i.e. each request ends in "\r\n\r\n").

Figure 2 is an example of a potential exchange between the different entities using the protocol.

2.1 Client requests

The client will either reveal a cell, flag a cell, or cheat to see the revealed grid, or will quit. It will thus send either:



```
(TCP connection)
TRY  0 0\r\nr\n
01#####\r\n
01#####\r\n
011####\r\n
001####\r\n
112####\r\n
#######\r\n
#######\r\n
\r\n
TRY  0 2\r\nr\n
011####\r\n
01#####\r\n
011####\r\n
001####\r\n
112####\r\n
#######\r\n
#######\r\n
\r\n
⋮
TRY  6 6\r\nr\n
0112B21\r\n
01B22B1\r\n
01111111\r\n
0011111\r\n
112B12B\r\n
1B2112B\r\n
1110011\r\n
GAME WON\r\n
\r\n
```

Figure 2: Example of the Minesweeper Protocol.

- "TRY *row col*" where *row* and *col* are **NUMBERS**, to reveal a cell
- "FLAG *row col*" where *row* and *col* are **NUMBERS**, to flag a cell
- "CHEAT" to get the revealed grid
- "QUIT" to signal to the server that it closes the connection

Note that all these requests ends in "\r\n\r\n".

2.1.1 Server responses

Most responses contain a *grid* that has the following format:

- If a cell is revealed and contains a hint, its value is the value of the hint i.e. a number between 0 and 8 included.
- If a cell is unrevealed, its value is "#".
- If a cell is flagged, its value is "F".
- If a cell is a bomb, its value is "B".
- Each line ends in \r\n.

The full response of the server will depend of the client's request but also the game state.

When receiving "CHEAT", if the game started, the server will respond with *grid* where all cells are revealed. If the game has not stated yet the server will respond with "GAME NOT STARTED\r\n".

When receiving "QUIT", the server will close the connection to the client.

When receiving a "TRY" request, the response will depend on the cell that is revealed. If the game is not over, the server respond with an updated *grid*. If the game is over with a win, the server respond with a revealed *grid* + "GAME WON\r\n". If the game is over with a loss, the server respond with a revealed *grid* + "GAME LOST\r\n". When the game is finished and the server send the response, the connection closes.

When receiving a "FLAG" request, the server respond an updated *grid*.

Additionally, if the *row* and/or *col* of a "TRY" or "FLAG" request is out of range, the server respond with "INVALID RANGE\r\n".

Finally, for any other request that it might receive, the server will answer with "WRONG" and keep the connection open.

3 Multi-Threading

A very simple solution for the server's code would be to accept a connection, handle the requests, 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, forwards 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. In addition, using the method `setSoTimeout` on the socket will ensure that a client that stays silent for too long will trigger an exception at server-side.

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.

4 Stream-oriented protocols (TCP)

The MP 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`.
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. MP achieves this by having a delimiter (CRLF) after each request. In practice, you should thus use a class that takes multiple reads into account (such as `BufferedReader`) or perform multiple reads if using a low-level `read`.

5 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 MinesweeperServer (resp. Minesweeper-Client), located in MinesweeperServer.java (resp. MinesweeperClient.java) at the **root of the archive**, and does not contain any `package` instruction. You are allowed to create more classes and java files than these two imposed ones.
- 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 27th October 2024 (**Hard deadline**) to the Montefiore Submission platform (`https://submit.montefiore.uliege.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. A diagram of classes is highly recommended.

Program Logic: How does your server work? Sequentially describe which operations it performs and mention the location in your source code where a given operation is implemented.

TCP stream : How did you read the requests from the TCP stream? You are allowed to provide the source code that handles this operation.

Multi-thread organisation: How did you achieve multi-threading in your program? You may also include source code to justify. Did you synchronize the activity of the different threads?

Robustness: How did you make your server robust? How did you manage the different types of exceptions that your server might receive?

6 Plagiarism

Plagiarism is the practice of taking someone else's work or ideas and passing them off as one's own. As of ULiege regulation, plagiarism is a fraud and is sanctioned with a grade of 0/20 for the course. In particular, giving his/her source code to another student, or copying the source code from another student submission, past or present, is strictly prohibited. Copying from an external source is allowed if the part that is copied is well identified and the source quoted. Note that, in that case, the quoted material will not award any points in the grading. In case of suspicion of fraud, an oral examination will be conducted to determine if the project is indeed the student's work.

Good programming...