Protocol requirements:

- Text protocol over TCP transport protocol.

- without encryption

- when nothing happens (no player does anything), nothing is sent

\* the exception might be a random ping message

- every request gets a response (even if it is only a one-character confirmation that the operation was successful)

Application requirements

- ● Applications are translated when submitted with a standard automatic translation tool (make, ant, maven, scons, ...; not bash-script, not manual gcc/javac, not via IDE)

- ● no libraries are allowed for networking and message serialization - your code solves this by itself using only BSD sockets (server) and native support in a standard library (client; Java, C#, ...); not even the C++2y network interface is allowed

- ● application code is appropriately structured into modules or classes

- ● the code is sufficiently and reasonably documented with comments

- ● applications (server and client) are stable, they do not crash on segfault (and others), all exceptions are handled, applications do not stall (e.g. deadlocks)

- ● the number of players in the game is limited only by the rules of the game; however, it must always be played by 2 people (so we can test it)

- ● once logged in, the player is taken to a "lobby" with rooms; the player has the option to select a room and enter it (if he/she does not exceed the player limit); alternatively, he/she is placed in a queue and waits for the game room to fill up

- ● the game allows you to recover from failures caused by unexpected client shutdowns, short-term or long-term network unavailability

\* according to the rules of the game, then either

\*\* waits for the player to return (the game is paused)

\*\* does not wait for player to come back, game continues and after player reconnects to next round of game (but player is still recovered after connection)

\*\* does not wait for player to return, game continues (but player status is restored when connected)

\* short-term unavailability should not force the player to manually try to connect (the client will do everything automatically)

\* prolonged inaccessibility should already work the other way around (and contain a message for the player)

\* all players in a given game must be aware of opponent's outages (short term, long term)

\* a player who is unavailable for an extended period of time is removed from the game; the game may close the room (according to the rules, usually there is no other way) and bring the active opponent back into the lobby

- ● Players are moved back to the "lobby" after the game ends

- ● both applications must work without the need to restart them (e.g. after several games)

- ● both applications process invalid network messages; in the event of an error, disconnection from the other party occurs

\* ● random data that does not match the protocol (e.g. with /dev/urandom)

\* messages that match the protocol format, but contain explicitly invalid data (e.g. moving a piece to square -1)

\* messages in an invalid game state (e.g. a move when the player is not in play/on his move, etc.)

\* messages with invalid input data according to the game rules (e.g. chess - diagonal rook move)

\* (Optional) applications can contain an invalid message counter and won't shut down immediately after the first invalid message, but only after e.g. three.

- (Optional) Both applications have some form of log for recording player status information, games, error messages, etc.

Server:

- The server is capable of serving several game rooms in parallel without interfering with each other (both in terms of gameplay and synchronisation)

- (optional) number of rooms (limit) to be set at the start of the server or in the configuration file

- (Optional) total limit of players (total game and lobby) is limited; this can also be set at server startup or in the configuration file.

- ● you may also specify the IP address and port on which the server will be listening (parameter or configuration file; not hardcoded)

Client:

- ● Client implements a graphical user interface (Swing, JavaFX, Unity or other according to the capabilities of the selected language and environment) (not a console)

- ● Client allows you to specify the address (IP or host name) and port to connect to the server

- ● the user interface is independent of the counterparty response - it does not get stuck, e.g., during a server connection or while sending a message/waiting for a response

- ● player and client are uniquely identified by nickname (we do not allow collisions) \* [optional] if you want, you can also implement a simple registration (nickname + password) to resolve collisions

- ● all user-entries are treated for invalid values

\* the same applies, for example, to the processing of moves in a game (e.g. in chess, so that the rook cannot move diagonally, etc.)

- ● The client always shows the current status of the game on the current board, the nicknames of the other players, who is on the move, if a player is unavailable, etc.

- ● The client visibly informs about server inaccessibility - at game start, in the lobby, in the game

- ● the client visibly informs about the inaccessibility of the opponent - in-game

● the client visibly informs about the inaccessibility of the opponent - in the game:

- the server runs on one GNU/Linux computer, the client runs on one GNU/Linux computer and one MS Windows computer

- the handover process will necessarily include (under the guidance of a student)

o translating the client and server

o running with different parameters

o running one complete game without faults (simulations) and without invalid data

o Ability to respond to failures (both applications; as specified)

o Ability to handle invalid data (both applications; as specified)

o check load on system resources