1. COMMUNICATION PROTOCOL DESCRIPTION

Protocol communication is simple and working over TCP. Data delivery confirmation is handled via TCP itself. Communication protocol begins after client connection to the server. After that server waits for client input about game state:

- In case of new game request client sends single ASCII zero symbol “0”. Which means to server new game request.
- In case of connection to existing game request client sends ASCII representation of game code (authentication symbol of game ID). This code should be provided from one client to another to play together. In case of successful connection server sends ASCII “START” to client.

In case of some fail (game with provided code does not exist or already has 2 players) server responds with ASCII “UNKNOWN” which means client should try again. This brings client back to connection state. After successful game establishment, server provides the way for clients to communicate with each other being just a tunnel between them.

Typical exchange data looks like the following:

Client sends ASCII “BOARD” followed by zero byte 0x00 followed by current board state followed by zero byte 0x00 followed by current board visual state. Those states are used by clients to represent the board in-game and visually.

2. IMPLEMENTATION DESCRIPTION

Server implementation:

Server contains of couple instances – class ChessServer and structure ChessClient each representing corresponding part of the game. ChessClient holds client name and client socket. ChessServer contains unsorted hash map of connected clients and unsorted hash map of ongoing games represented as pairs of clients. During client connection to the game lookup of game hash map takes place to verify the client is able to connect to required game. Each game runs in separate thread preventing from accidental game intersections.

Client implementation:

This Python code is a simple implementation of a chess game using the PyGame library. Here's an overview of its functionality:

Chess Pieces Class (Piece): Represents a chess piece with attributes like team, type, image, and whether it can be killed by another piece.

JSON Encoder for Piece Class (PieceEncoder): Custom JSON encoder to serialize the Piece class objects.

Initialization of Chess Pieces: Instances of various chess pieces (pawn, king, rook, bishop, queen, knight) are created for both black (b) and white (w) teams.

Initial Arrangement of Pieces on the Chessboard: Pieces are arranged on the chessboard according to the standard starting position.

Functions for Chess Moves (pawn_moves_b, pawn_moves_w, king_moves, rook_moves, bishop_moves, queen_moves, knight_moves): These functions determine the valid moves for different types of chess pieces.

Functions for Board Initialization and Display:

- create_board: Initializes the chessboard.
- convert_to_readable: Converts the board to a readable string format.
- deselect: Resets 'x's and killable pieces.
- highlight: Highlights valid moves on the board.

Pygame Window Setup: Pygame window initialization with necessary configurations.

Node Class (Node): Represents a node in the chessboard grid.

Functions for Chessboard Grid (make_grid, draw_grid, update_display, Find_Node): These functions handle the creation, drawing, and updating of the chessboard grid.

Functions for User Interface (UI) State Handling (main): The main game loop where different UI states (connection, menu, code, authentication, game, done) are managed. UI elements like input boxes, buttons, and text are drawn based on the current state. Mouse and keyboard events are handled accordingly. Connection Setup and Communication: The game can be set up to connect to a server, create a new game, or join an existing game. The code uses sockets to establish connections and exchange data between the client and server.

Game State Handling (GAME_STATE): Chess moves are processed based on user input and sent to the opponent through the established connection. The board state is updated based on received data from the opponent.

Exception Handling: Exception handling for potential errors during socket communication.

Main Function Call (`main(WIN, WIDTH)'): Invokes the main game loop with the Pygame window and width parameters.

Overall, this code provides a functional chess game that can be played between two clients connected over a network. The game state and moves are communicated between the clients using sockets.

3. COMPILATION END EXECUTION

For server: only CMake v3.1.0+ required and any C++17 -compatible compiler (Clang/GCC). None of additional libraries used for simplicity. To build server one should create “build” folder and then call cmake command from it as follow (from ChessServer source dir):

```
mkdir ./build
cd ./build
cmake ..
cmake --build .
cd ../
```

Then, server can be started as following (with sudo cause on some linux distributions networking is a privileged operation requiring super user) from ChessServer source dir:

```
sudo ./build/ChessServer
```

For client: only Python v3.8+ required. Code actively uses PyGame python library alongside some standard libraries (json, socket, sys, time, traceback). PyGame installed as following:

```
pip install pygame (or pip install -r ./requirements.txt)
```

To start client execution only following command is required (from ChessClient source dir):

```
python ./ChessClient.py
```

All you have to do then is to specify server IP (IP of the machine in LAN which is running ChessServer). Port is by default set to 8080 and should be changed in server code if required!