Practical Work 2
DAI - HEIG-VD

Loïc Herman Massimo Steffani

I. Chess Communication Protocol

A. Overview

The Chess Communication Protocol (CCP) is a protocol for communicating chess games between chess engines and chess GUIs. It is designed to be simple and easy to implement, and to be able to represent all chess games.

B. Transport protocol

The CCP protocol uses text-based messages over a TCP (Transmission Control Protocol) connection to ensure the reliability of data transmission and must also use port 6343.
The initial connection must be established by the client.
The server can implement a matchmaking system where clients can play a game against any other client connected to the server. The requirement is that the server must be able to manage one game at a time, meaning 2 concurrent client connections, one for the white player and one for the black player.
The maximum number of connections per server is two times the amount of concurrent games the server is configured to run, meaning at least 2. Beyond this number, the server should not accept new connections until a spot is available.
Once the connection is established, the client should wait for the server to assign them a player color, this indicates the game is ready to start.
The game is played by turns, meaning that the server must wait for a message from the client before sending a message to the other client. Client A (assigned the white color) must send a move message to the server, then the server must send the move message to client B (assigned the black color). Client B must then send a move message to the server, then the server must send the move message to client A. This process repeats until the game is over.
The server must verify on each message recieved that the message is valid and that the client is not trying to cheat. If the message is invalid, the server must send an error message to the client.
If the played move requires a promotion (pawn on the top row), the client must send a promotion message to the server, but only after the move message is sent. The server will verify and relay the information to the other client.
When the game is completed, the clients should notify the server if they want to play another game, by sending a replay message. If both clients vote yes, the server replies to each client, and the game starts over.
A client can disconnect at any time, but the server must be able to handle this situation and notify the other client that the game is over by closing the other client's connection.

C. Messages

1) *Color:*
In order, the server will first send a `COLOR` message.

```
COLOR <WHITE|BLACK>
```

No response is expected, but the client that recieved the color white should prepare to send a move message
2) *Move:*
Request:
The client sends the following

```
MOVE <fromX> <fromY> <toX> <toY>
```

Response:
The server responds with `ERROR {code}` if an error or the move is invalid. The reference for the error codes is available in the appendix.
If the move is valid, the server does not respond, it will forward the move to the other client, the client should await a `MOVE` message from the server (the move of the other player).
3) *Promotion:*
Request:
The client sends the following

```
PROMOTION <0..4> <X> <Y>
```

Where `X` and `Y` correspond to the position of the peice and the numeric ordinal for the piece corresponds to the following list:

1. ROOK
2. KNIGHT
3. BISHOP
4. QUEEN

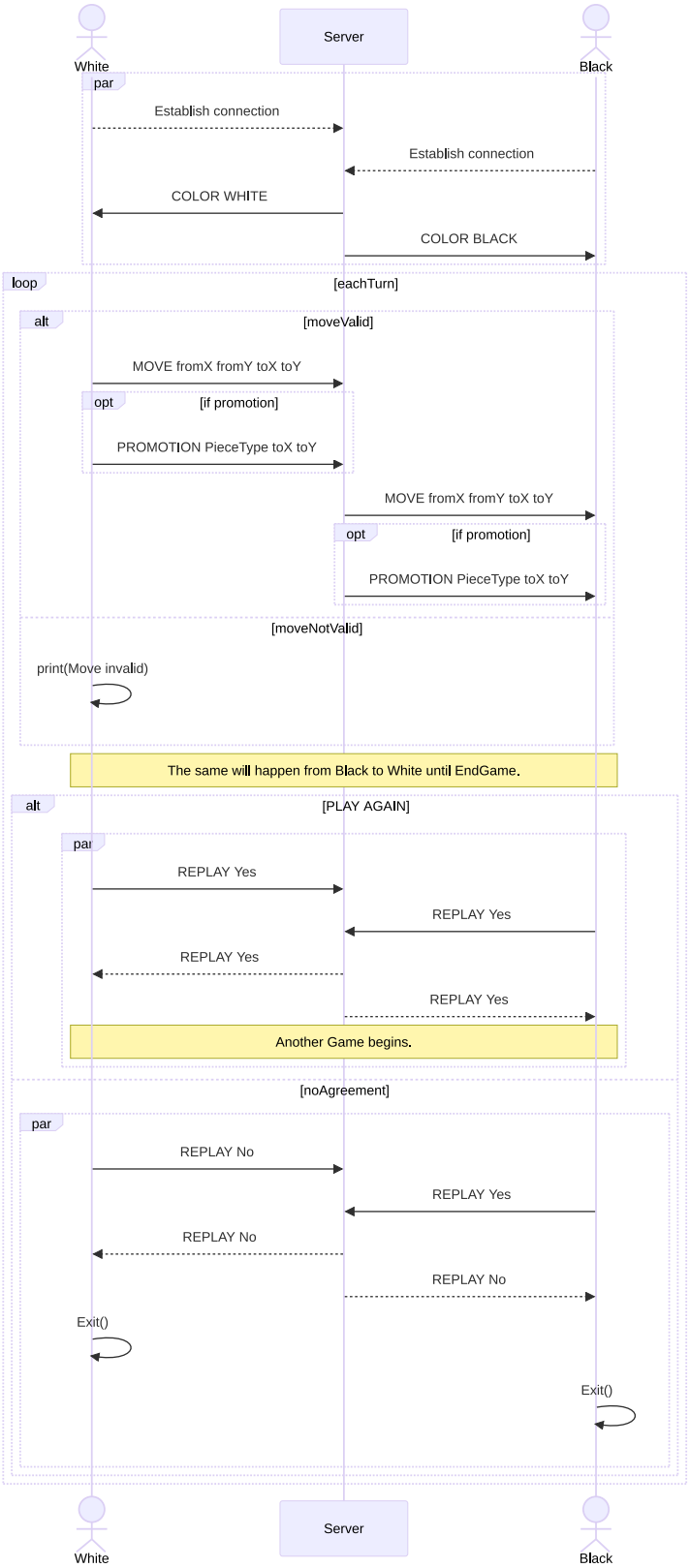
Response:
The server responds with `ERROR {code}` if an error occurs or the move is invalid. The reference for the error codes is available in the appendix.
If the move is valid, the server does not respond, it will forward the promotion to the other client, the client should await a `MOVE` message from the server (the next move from the other player).
4) *Replay:*
When the game ends, each client should send a replay message to the server.
Request:

```
REPLAY <Yes|No>
```

Response:
The server responds with `REPLAY <Yes|No>` to each client. If both clients vote yes, the game starts over, otherwise each client should close their connection to the server.

D. Example

The next figure contains an example of a game between two clients, the first client is assigned the white color, the second client is assigned the black color.
Edge cases are also described, such as a promotion, an invalid move, and a replay (one where both players agree, and one where at least one player disagrees).


White
Server
Black
par
Establish connection
Establish connection
COLOR WHITE
COLOR BLACK
loop
[eachTurn]
alt
[moveValid]
MOVE fromX fromY toX toY
opt
[if promotion]
PROMOTION PieceType toX toY
MOVE fromX fromY toX toY
opt
[if promotion]
PROMOTION PieceType toX toY
[moveNotValid]
print(Move invalid)
The same will happen from Black to White until EndGame.
alt
[PLAY AGAIN]
par
REPLAY Yes
REPLAY Yes
REPLAY Yes
REPLAY Yes
Another Game begins.
[noAgreement]
par
REPLAY No
REPLAY Yes
REPLAY No
REPLAY No
Exit()
Exit()
White
Server
Black

II. Appendix

A. Error codes