Chess Game

Ciobotaru Mihai

Universitatea Alexandru Ioan Cuza, Facultatea de Informatica

Abstract. This papers aims to describe the implementation of a chess game played on different systems. The main focus of this project is to create an efficient networking protocol trough which the clients and the concurrent server communicate.

Keywords: Chess · Computer Networking · Games · SFML · C/C++.

1 Introduction

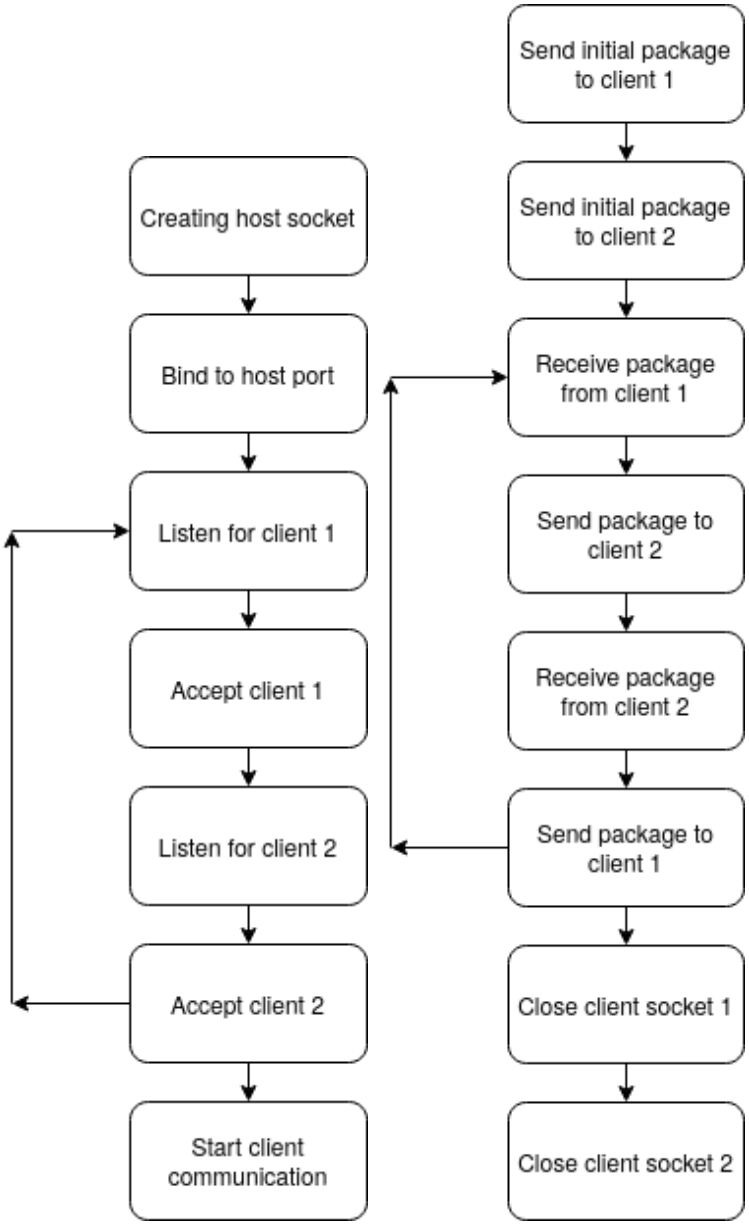
The project aimed to be accomplished is called "ChessC" and "ChessS". This project consists of the client and server side of a chess game played across different systems. The main requirements of this application are creating an attractive interface for the user to play and making an efficient protocol to communicate with the concurrent server.

2 Technologies used

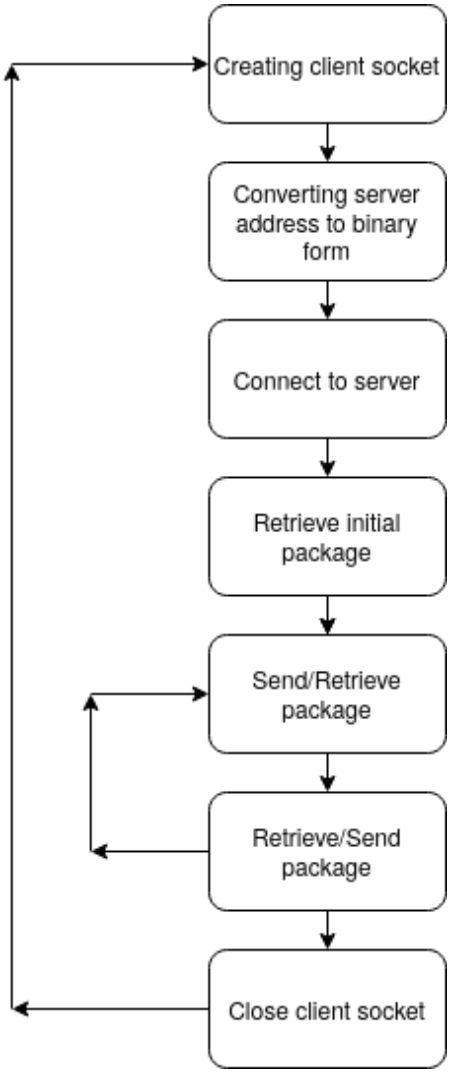
For the graphical side of the project I decided to use SFML - Simple and Fast Multimedia Library. SFML is a cross-platform library designed to provide simple application programming interface (API) having bindings in C. For the transport layer protocol, the best option is to use TCP - Transmission Control Protocol. TCP is a connection-oriented transport protocol that send the information correctly and keeps track of the packets‘s orders. This concept is essential because the game relies on loyal sending and receiving of packets in correct order.More than that, when sending a package, the protocol checks that the message was sent correctly and ensures that the parties are still connected. As for the assets used, I have displayed the pieces and the board‘s tiles with the help of John Pablok‘s chess PNGs.

3 Application architecture

3.1 Server diagram



3.2 Client diagram



3.3 Network protocol

Server side The server first creates a socket and binds it‘s address and port to it in order for the clients to connect. After that the server enters in a loop designed for listening to clients. When the server successfully accepted two clients it creates a new thread for their communication and goes back at the start of the loop to listen to new clients. All of these details described are presented in the diagram above in the left and the other diagram depict the communication between the clients. The protocol starts by sending to each client the color of

pieces he plays and, by sending these initial packages, lets the clients that there was found another client to play with. The first player to connect is always the white player and the other is the black player. After sending the initial packages, it enters the loop of communication. First the server receives a package from the first client and sends it back to the second. The second client sends a package after that to the server and the server sends it back to the first client. This goes on until the server receives an exit package.

Client side Firstly the client creates it's socket and converts the server address to binary form. After it has done that, it enters in a loop that it will exit only when it would have successfully connected to the server. Once connected to the server it waits for the initial package that it lets him know that it has found another player to play with. It process the initial package in order to know which color it is. If it is white, naturally, he will move first, then the other player and so on until a player wins. After a client makes a move,it will send the board info to the server and wait to receive board info back from the server in order to move again. While waiting to receive the player can not move because it is not their turn(their turn comes once again when the other players finishes makes a move). Once the game has finished, the client has two choices to exit or to play another game. There also exists the possibility of exiting the game in the middle of it. When exiting prematurely, the client sends the server a package in which it lets him know. The server sends that package to the other client to let him know that he was disconnected (the server disconnects the other player and exits communication when one of the players exited). When this happens, the client has the same possibility as before: to search for a new game or to exit.

Packages Packages have either 4 bytes or 256 bytes. Initial packages have 4 bytes representing each an int. The initial package sent to the first client is number 2 which tells the client he is the color white and, for the second client, the package contains the number 1 meaning the color black. Anytime the client quits, he sends an exit package of 4 bytes size representing the number 0 as an int to the server. When the server receives such a packages he sends the same package with the same size to the other client and after that closes the client sockets quitting the communication. When such a number is read by the client, the player can not move any pieces anymore and a popup with the message "Disconnected" appears. The popup offers the possibility to chose to search for a new game or to exit the app through two buttons. There exists one last type of package, the 256 bytes packages. This package is an 64 long int array encrypting for each tile in the board what piece is on it. This package is sent to the server after a client has been done. When the server receives such a package, he sends it back without processing it further to the other client. When a board package is received, the client knows the other player has finished it's move and it is his turn now. Each client has a matrix of the board and when a board packages is received, it replaces the matrix it has locally with the one received.

4 Implementation details

Board and piece encoding

```
tableMatrix = {
        {4,3,2,5,6,2,3,4},
        {1,1,1,1,1,1,1,1},
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},
        {-1,-1,-1,-1,-1,-1,-1,-1},
        {-4,-3,-2,-5,-6,-2,-3,-4}
};
```

In the image shown above, we can see how we encode the board in the app. Each number represents a piece and the it‘s index represents the position it has on the board. Negative numbers stand for the color white pieces and the positive ones for the color black. Number 0 is the ”null” piece - meaning no piece exists there. Numbers 1, 2, 3, 4, 5 and 6 are the pawn, the bishop, the knight, the rook, the queen and the king.

Package struct

```
struct package{
    int type;
    std::vector<int> data;
};
```

Packages have a struct of their own. Once the client is done processing the information received from the server, it stores it in a package struct. The package struct stores the type of packages and the data(that can vary from

type to type). There are 3 types of packages : 0 - exit package, 1 - initial package, 2 - board package.

Client package retrieving

```
void TCP_client::retrive_package(package* info) {
    std::vector<int> processed_package;
    int package[64] = {};
    int bytes_received;
    bytes_received = recv( fd: client_socket,  buf: package ,  n: sizeof(package),  flags: 0);
    if(bytes_received < 0){
        perror( s: "[-]Could not receive.\n");
    }
    else if(bytes_received == 0){
        std::cout<<"[+]Server has disconnected.\n";
        processed_package.push_back(0);
        info->type = 0;
        info->data.push_back(0);
    }else if(bytes_received == 4){
        if(package[0] == 0){
            std::cout<<"[+]Shutting down request accepted.\n";
            info->type = 0;
            info->data.push_back(0);
        }else {
            info->type = 1;
            info->data.push_back(package[0]-1);
            turn = info->data[0];
            isConn = true;
        }
    }else {
        turn = true;
        info->type = 2;
        for(int i = 0;i<64;++i){
            info->data.push_back(package[i]);
        }
    }
}
```

This function is the one that retrieves a package from the server and processes the data give to store it further in a package variable.

Client package sending

```
void TCP_client::send_board(std::vector<std::vector<int>> matrix){
    turn = false;
    int board[64];
    for(int i = 0;i<8;++i)
        for(int j = 0;j<8;++j)
        {
            board[i*8+j]=matrix[i][j];
        }
    int bytes_sent = send( fd: client_socket, buf: board,  n: sizeof(board), flags: 0);
    if(bytes_sent < 0){
        perror( s: "[-]Could not send!\n");
    }
}

void TCP_client::send_exit() {
    int exit = 0;
    int bytes_sent = send( fd: client_socket, buf: &exit,  n: sizeof(exit), flags: 0);
    if(bytes_sent < 0){
        perror( s: "[-]Could not send!\n");
    }
    isConn = false;
    close( fd: client_socket);
}
```

These two functions manage the sending of packages from the client to the server. The client being able to send either an exit package or a board package. In the case of sending the board package, it first converts the board matrix into an array because the client uses the board as a matrix but the packages is an array.

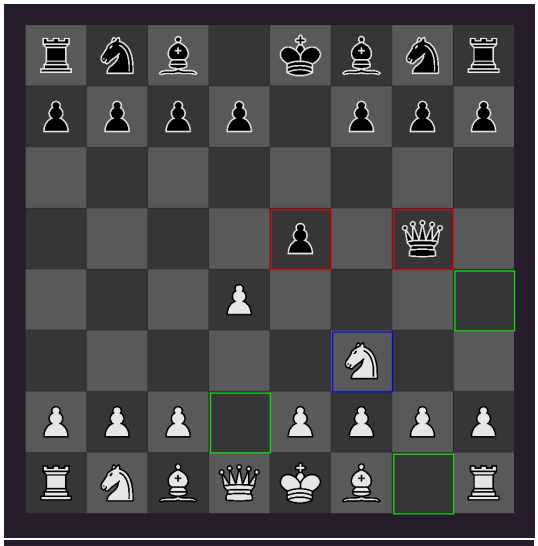
Server package processing

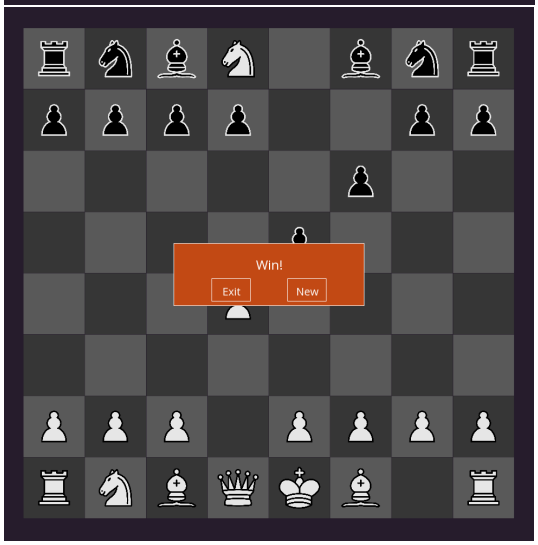
```
/// receiving
int package[64] = {};

bytes_received = recv(client_socket, package , sizeof(package), 0);

if(bytes_received < 0){
    perror("[-]Could not receive.\n");
}
else if(bytes_received == 0){
    cout<<"[+]Client has disconnected.\n";
    break;
}
else if(bytes_received == 4){

    /// sending exit signal

    if(package[0] == 0){
        cout<<"[+]Shutting down request accepted.\n";
        int* exit = new int;
        *exit = 0;
        bytes_sent = send(client_socket_2,exit,sizeof(int),0);
        if(bytes_sent < 0){
            perror("[-]Could not send!\n");
        }
    }

    break;
}else {
    /// sending table packages

    bytes_sent = send(client_socket_2,package,sizeof(package),0);
    if(bytes_sent < 0){
        perror("[-]Could not send!\n");
    }

}
```

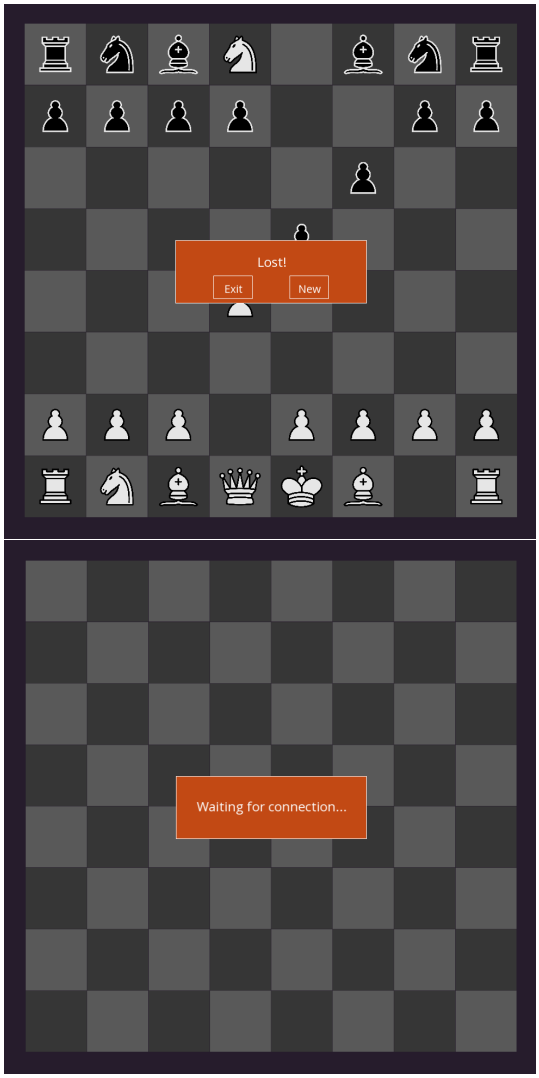
In this photo, it is depicted how the server receives packages and interprets them. The client and the server differentiate the types of packages through the size of them. If the server receives a 4 bytes sized packages, it know it is an exit signal, sends to the other client the signal and then exits the communication loop. If else, it knows it is a board packages and just sends it to the other client and goes further. Because the client can receive an exit package and an initial package which have both 4 bytes, it know that if the package contains the numbers 0 is an exit signal, if else, is an initial package.

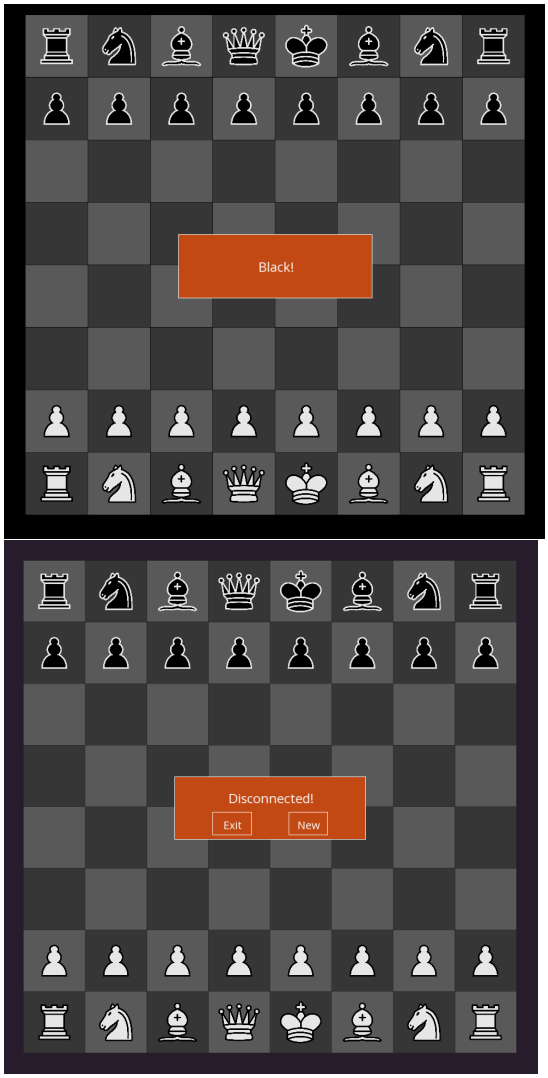
Other details The application has a series of popups that appear when a certain game state is met. While waiting for the initial package the game displays an "Waiting for connection" popup. When the initial package is retrieved, the application displays for 3 seconds a popup with the color that the player has. After the game ends, the application displays a popup with either the "Win" or "Lose" message and with 2 buttons to let the player choose to play again or to exit. If, in the middle of the game, the other player exits, there is displayed a popup with "Disconnected" message that, once again, gives the possibility to play again or to exit through two buttons. In the moment that a game ends, there are sent exit signals and all the clients sockets and the communication are ended. If the player chooses to play another game, then there are created new ones. It is worth noting that, when a popup is displayed, the player cannot move any pieces even though it is his turn to play.

5 Images






Lost!
Exit
New
Waiting for connection...



6 Conclusions

The project succeeded in meeting it‘s requirements: having a concurrent server, an efficient protocol and giving the possibility to play chess through a beautiful interface. These being said, the project is a good enough example of a chess game with client and server implementation.

References

1. LNCS Homepage, http://www.springer.com/lncs. Last accessed 4 Oct 2017

2. Geeks for geeks - multithreading,
https://www.geeksforgeeks.org/multithreading-in-cpp/.
3. SFML documentation, https://www.sfml-dev.org/documentation/2.5.1/.
4. John Pablok - chess pieces and board assets,
https://opengameart.org/content/chess-pieces-and-board-squares.
5. Jacob Sorber - computer networking, https://www.youtube.com/c/JacobSorber.
6. Lenuta Alboaie, Andrei Panu - computer networking,
https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php .