TheQueensChess ONLINE V2

Team Name: Queen’s Gambit

Team 18

Jonathan Sugijoto, Eesha Jain, Devin Reyes, Jixin Gong,

Software Title: TheQueensChess ONLINE

Version: V2

Affiliation: University of California, Irvine

Glossary

AI: Stands for ‘Artificial Intelligence.’ This refers to a computer program that can seemingly mimic human actions by being programmed to ‘think’ like humans.

Bishop Move: The bishop can move diagonally in any direction given that no pieces are not blocking it.

Castling: If the king and the rook have not moved from their starting positions, and the spaces between them are empty, the king moves two in the direction of the rook, and the rook ‘jumps’ one square over the king (all horizontally).

Char: shorthand for ‘character’

Chat Rooms - An area of the internet where computer network users can communicate

Check: When a player is in check, this means that the king is in danger, but not yet defeated. A move must be made to get the king out of check. The king cannot remain in check.

Checkmate: Checkmate is when the king is in a position where he is no longer able to defend himself, and therefore the game is lost.

Client Software - A specific application that can be used to communicate to another piece of software over a network

Double Linked List: A list of stored information that can iterate both ways

En Passant: When a pawn moves diagonally to capture an adjacent piece

Extranet - An intranet that can be partially accessed by authorized users, allowing a secure exchange of information between businesses

Int: An integer number; no decimals.

Internet - A global computer network providing a variety of information and communication facilities comprised of interconnected networks

Intranet- A local area network, restricted communications network

King Move: The king can move one step in any direction and cannot jump over pieces

Knight Move: The Knight can move in an L formation. It is the only piece that can jump over other pieces (with the exception of a king and a rook in castle). An ‘L formation’ means 1 in front and 3 to the right or left, OR 3 in front, and 1 to the right or left.

Linked List: A list of stored information that only iterates one way

Login - Process by which an individual gains access to a computer system by identifying and authenticating themselves

Modularization: The practice of segmenting code into different files, or ‘modules’, in order to increase efficiency

Network - A system to connect computers together in order to share information

Pawn Move: The pawn can move either one or two spaces vertically in front of itself on the first move, and from there on can only move one space in front of itself, or diagonally to kill. It can also move diagonally to kill on the first move, if the opportunity presents itself. The pawn cannot move diagonally if it is not killing.

Rook Move: The rook can move vertically and horizontally in any direction given that no other pieces are blocking it.

Queen Move: The queen can move in any direction as long as no pieces are blocking her. She cannot jump over the pieces.

Server - A computer program that manages accessibility within a centralized network

Stalemate: When neither side is able to attack or defend; a draw.

World Wide Web - System of interlinked hypertext documents accessed via the Internet

1. Installation

1.1 System Requirements

1. This application will be run off of the UCI EECS Servers, which is running CentOS version 6.10
2. The program requires 1M

1.2 Setup and Configuration

1. Unzip the tar file with the command “tar -xvzf Chess.tar.gz”
2. Run the makefile command “make’’
3. Depending on who is running the server and the client, run either:
    a. ./bin/server <port>
    b. ./bin/client <server name> <port>
    c. ./bin/client <server name> <port>
    d. Now both clients can play
4. The server name will be whatever server the “server runner” is connected to (EX: zuma, bondi, etc) and the port will be specified by the server.
5. To login to the server use the username: “Queens” and the Password “Gambit”
   E.g:
   Username: Queens
   Password: Gambit

6. NOTE: if you make a wrong move in the client client game, the program stops working and you have to restart the game. Furthermore, the game does not print that the board is in check, but the board still ‘knows’ that it is in check.

1.3 Uninstalling

1. Delete the file that you unzipped from the tarball by changing directories out of it with:
    a. ‘cd ..’
2. Remove the unzipped file with:
    a. ‘rm -r Chess’
3. Remove the tarball with:
    a. ‘rm Chess.tar.gz’

2 Client Software Architecture Overview

2.1 Main Data Types and Structures

There will be:

- int
- Char
- Socket structs

2.2 Major Software Components

Client.c, Client.h

2.3 Module Interface

Client.c: Client launcher that will handle connecting to an open hosting executable. The server will be hosting the game and relaying the information that the client needs in order to participate in the game that is going to be held.
Client.h: Header file for the client file

2.4 Overall Program Control Flow

We plan to have the client launch their executable and connect to an already running host. The host will then be able to lead the client through the menus and finally into the game. Disconnecting from the host can be done at any time by stopping the program. The two clients that are connected to the server can play against each other and pick colors.

3 Server Software Architecture Overview

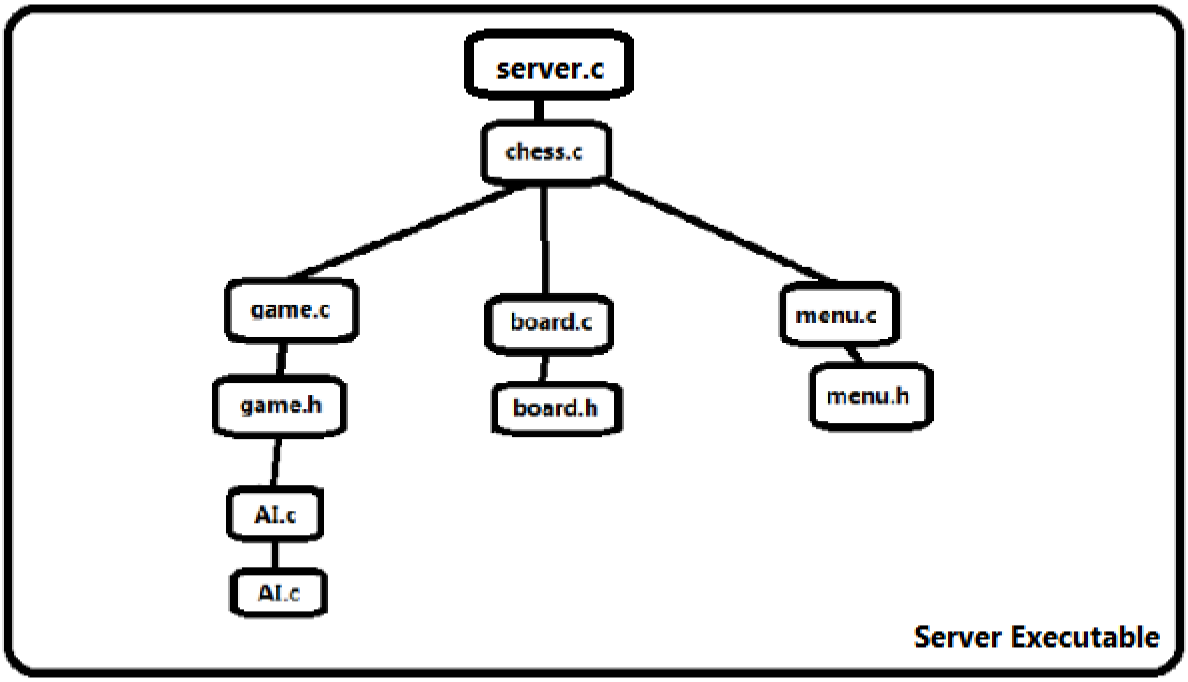
3.1 Main Data Types and Structures

There will be:

- Piece Structs
- Singly Linked Lists
- Doubly Linked Lists (MoveList/Take Back Support)
- int
- Char
- Socket structs

3.2 Major Software Components

AI.c, AI.h, board.c, board.h, chess.c, game.c, game.h, menu.c, menu.h, server.c, client.c, server.h, client.h, makefile.



3.3 Module Interface

AI.c: Implementation of AI computation functions against the player.
AI.h: Header file of AI.c.
board.c: Setting up the board and the location of pieces as well as all functions related to piece structures and movement, such as checking move validity, capture, and check.
board.h: Header file of board.h.

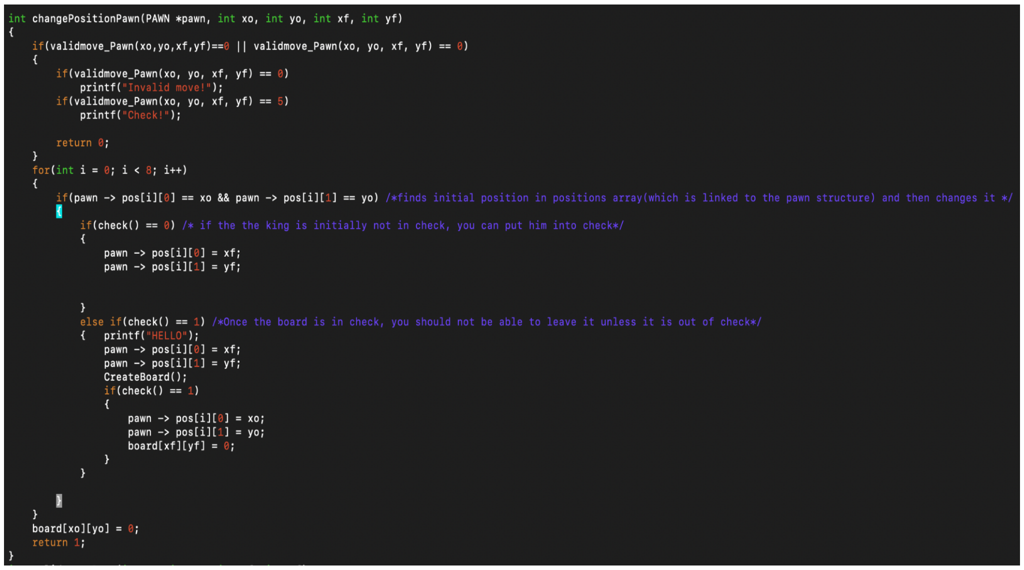
chess.c: Calls the menu function and includes menu options.
game.c: Main file that connects all the files together. Has HumanVsHuman and HumanVsAI functions.
game.h: Header file of game.c
menu.c: Print the menu.
menu.h: Header file of menu.c
makefile: Generates the executable program.
Server.c: Server side C file that manages user and connection

3.4 Overall Program Control Flow

We plan to have the program flow from the initial setup of the server on the hosting computer which will then wait for a connection from the client executables. Once there is a connection, the server/host will be able to choose a gamemode and change the settings, preferably referencing the chat while doing so in order to accommodate the client as well. Finally, once a game starts between the host and client, there will be back and forth of chess exchanges until there is a winner. From a winner, it will return to the main menu and we have come full circle.

4 Documentation of Packages, Modules, Interfaces

4.1 Detailed Description of Data Structures



```c
int changePositionPawn(PAWN *pawn, int xo, int yo, int xf, int yf)
{
    if(validmove_Pawn(xo,yo,xf,yf)==0 || validmove_Pawn(xo, yo, xf, yf) == 0)
    {
        if(validmove_Pawn(xo, yo, xf, yf) == 0)
            printf("Invalid move!");
        if(validmove_Pawn(xo, yo, xf, yf) == 5)
            printf("Check!");

        return 0;
    }
    for(int i = 0; i < 8; i++)
    {
        if(pawn -> pos[i][0] == xo && pawn -> pos[i][1] == yo) /*finds initial position in positions array(which is linked to the pawn structure) and then changes it */
        {
            if(check() == 0) /* if the the king is initially not in check, you can put him into check*/
            {
                pawn -> pos[i][0] = xf;
                pawn -> pos[i][1] = yf;


            }
            else if(check() == 1) /*Once the board is in check, you should not be able to leave it unless it is out of check*/
            {   printf("HELLO");
                pawn -> pos[i][0] = xf;
                pawn -> pos[i][1] = yf;
                CreateBoard();
                if(check() == 1)
                {
                    pawn -> pos[i][0] = xo;
                    pawn -> pos[i][1] = yo;
                    board[xf][yf] = 0;
                }
            }

        }
    }
    board[xo][yo] = 0;
    return 1;
}
```

Figure 4.10 For our functions that move pieces, we check that the piece move is valid, and we check that if a king is already in check, no moves that keep the king in check remain valid.

For our data structures, we have one header and one c file for the game, menu, AI,and board. We also have one main C file. In our header file for the board, we have the board and piece structures, as well as our global piece variables. We are also including client and server files. These will be major components of our project.

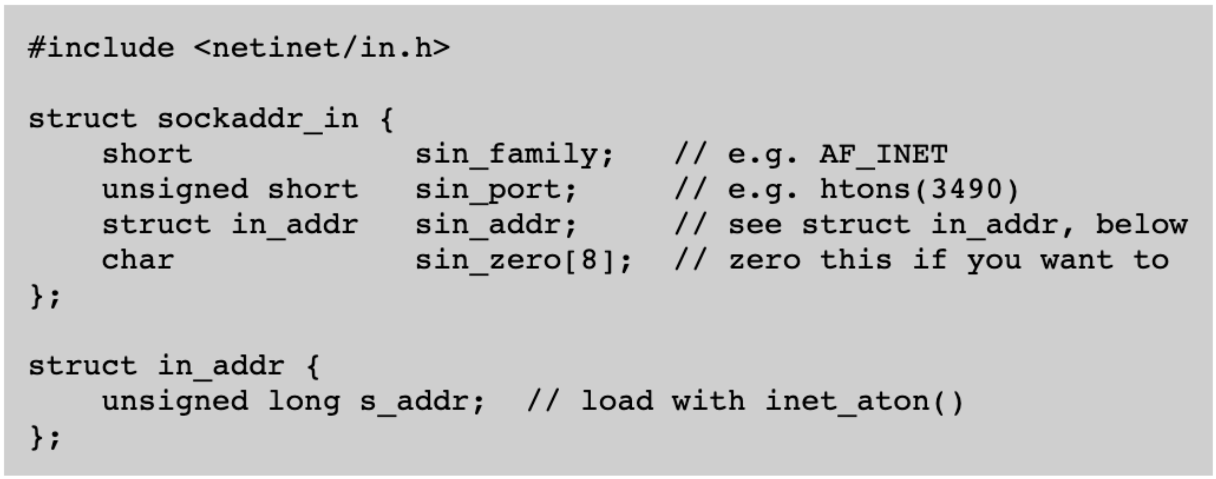
We also have piece structures for each piece. For example, the pawn structure holds the locations of each of the 8 pawns and the color, and the character for the pawn, and the same goes for each piece and a char. We are also going to have a list with all the piece structures in it (a single-linked list) that will store all of the pieces and their location.

If the piece has been removed from the board, we will set its location to -5,-5

Each structure will have a set color and location assigned to each chess piece. By assigning each chess piece its own structure we are able to add them to a list of chess pieces so that we can go through the list when needed to find legal moves or even remove pieces from the board. Allowing for modularity throughout the program. The structs are incomplete, and still need some more information such as piece name, etc.

We have shown a snippet of one of our “change position” functions. The code iterates through the positions array given in the struct to find the piece the user wants to move. Then, the ‘valid position’ function is called, checking if the user has made a valid move. Then, if the board is not in check, we let the piece move. If the board is initially in check, we check that no move that does not get the piece out of check is allowed as per the chess rules.

For our client and server files, we would like to expand our software to include multiple users where each user can have their own account to play chess on. Users should be able to connect to a server and play against each other.



```
#include <netinet/in.h>

struct sockaddr_in {
    short            sin_family;   // e.g. AF_INET
    unsigned short   sin_port;     // e.g. htons(3490)
    struct in_addr   sin_addr;     // see struct in_addr, below
    char             sin_zero[8];  // zero this if you want to
};

struct in_addr {
    unsigned long s_addr;  // load with inet_aton()
};
```

Figure 4.21

This is the main data structure we will be referring to for most of our code. This socket structure holds the type of IP address the client/host is on and then what port they are connecting to. Next we take in the actual address of the client/host to keep for future use to startup the connection.

As we can see above, this structure holds all information necessary for us to establish our server in the first place. By being able to quickly access and manipulate the data in this struct, we can properly set up our server to work with all users and create a stable connection for those who use our program.

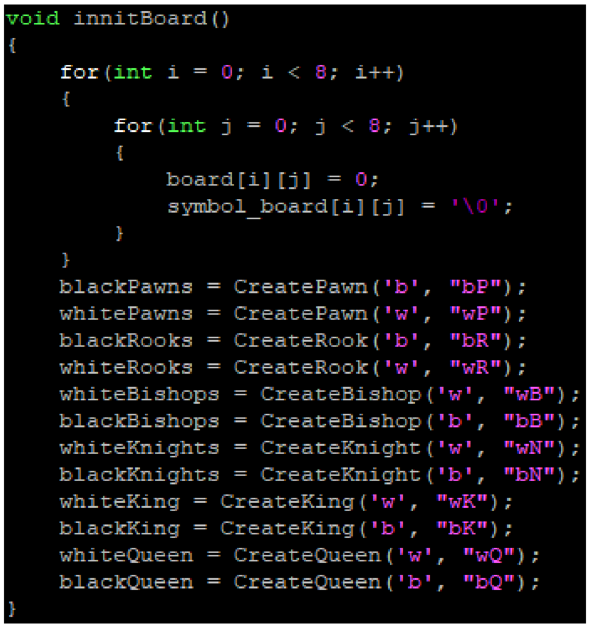
4.2 Detailed Description of Functions and Parameters

Three functions that we used to initialize and draw the board are innitBoard, getBoard and CreateBoard.

InnitBoard initializes each of the pieces for each color given the color and piece name as a parameter.

The get board function takes in the pieces as parameters and has a char symbol board and an int board. The int board is an 8x8 array that stores a 1 if the piece is white and a -1 if the piece is black. The symbol board holds the character for the piece.

CreateBoard recreates the board and symbol board after double checking the structures and referencing the board array we have passed around. This function is used extensively after the movement of pieces.



```
void innitBoard()
{
    for(int i = 0; i < 8; i++)
    {
        for(int j = 0; j < 8; j++)
        {
            board[i][j] = 0;
            symbol_board[i][j] = '\0';
        }
    }
    blackPawns = CreatePawn('b', "bP");
    whitePawns = CreatePawn('w', "wP");
    blackRooks = CreateRook('b', "bR");
    whiteRooks = CreateRook('w', "wR");
    whiteBishops = CreateBishop('w', "wB");
    blackBishops = CreateBishop('b', "bB");
    whiteKnights = CreateKnight('w', "wN");
    blackKnights = CreateKnight('b', "bN");
    whiteKing = CreateKing('w', "wK");
    blackKing = CreateKing('b', "bK");
    whiteQueen = CreateQueen('w', "wQ");
    blackQueen = CreateQueen('b', "bQ");
}
```

Another function is going to be the AI function. The AI function decides how to move their pieces by randomly generating a piece and then moving it to some valid position, also randomly.

A checkmate and check function has been completed as well. We will check if any of the pieces are able to kill the king in the next move. We will do this by making the pieces around the king hypothetically move one step forward. For checkmate, if the king or any pieces have no moves to make, then the king is in checkmate. However, if the king can defend himself, or if any other pieces can defend the king, the king is in check.

Figure 4.20: This function initializes our board for online play

We will have a function that allows the user to set password and username and store it somewhere so the user can always log in. Our function needs to handle logging into an existing account as well as creating a new account.

We also need to create a function that handles if the user wants to log onto the server and connects two users screens together so that the user can always see what the other person is doing.

UserLogin(): This function will provide the information necessary for us to check in a user into our game and have them authenticated for server use. This function will look into the text file made with all of our user information.

GetAddress(): To implement this function we hope to have the server keep track of what user is actually connecting to our server. By consistently checking in with the user we can ensure that the connection is stable and consistent throughout the game.

Timeout()/select(): Similar to the clock socket tutorial in lecture, we want to essentially timeout the server every time the user wants to use this our program so that it runs better in a client, host based program like our online chess game.

4.3 Detailed description of the communication protocol:

For the settings portion of our communication protocol, we want to have options for the user to create accounts, log in to an existing user account, and possibly even allow for customization similar to a social media application. By keeping a detailed record of each user that has connected to the server, the program can work closely with the client to create a seamless connection to our online chess program.

5 Development Plan and Partitioning of Tasks

5.1 Partitioning of tasks

Jonathan - Debugging, Github, and Part of the Server Code

Eesha - Client Code

Devin - Implementation of code into functions and part of the Client Code

Jixin - Server Code

5.2 Team Member Responsibilities

Jonathan - Take responsibility for the due dates, and helps with debugging

Eesha - Takes responsibility to make sure we are up to date with coding standards and helps with presentations

Devin - Takes responsibility to make sure that the files are all working efficiently together and to make sure all files are commented properly as well as makes sure to implement the client and server code into our functions

Jixin - Helps write helper functions for bits of the code that need them and gathers information on the sockets

Back Matter

6.1 Copyright

6.2 References

Kister, J., et al. "Experiments in chess." *Journal of the ACM (JACM)* 4.2 (1957): 174-177.

Shannon, Claude E. "XXII. Programming a computer for playing chess." *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 41.314 (1950): 256-275.

6.3 Index