TheQueensChess V3

Team Name: Queen's Gambit

Team 18

Jonathan Sugijoto, Eesha Jain, Devin Reyes, Jixin Gong, Michael Nguyen*

Software Title: TheQueensChess

Version: V3

Affiliation: University of California, Irvine

*Extenuating Circumstances: did not contribute

Table of Contents

Glossary

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

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

Int: An integer number; no decimals.

Char: shorthand for ‘character’

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

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

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.

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

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).

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

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.

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.

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

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

1 Software Architecture Overview

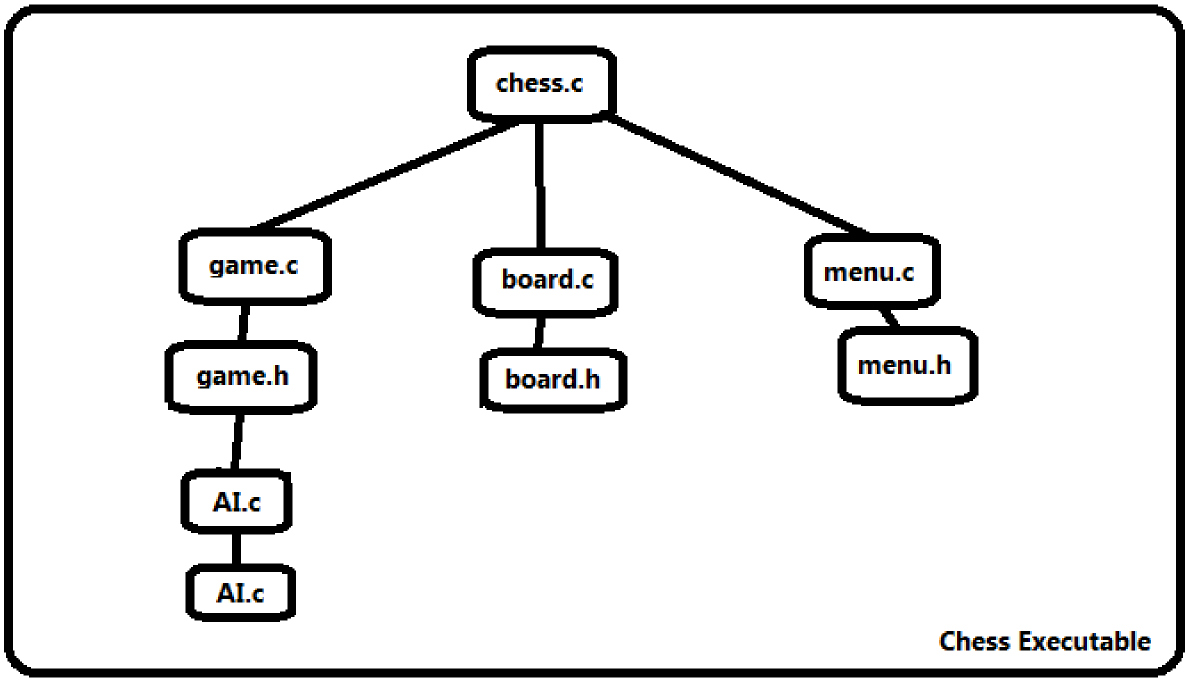
1.1 Main Data Types and Structures

There will be:

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

1.2 Major Software Components

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



1.3 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.

1.4 Overall Program Control Flow

We plan to have the program flow from the menu, to the settings, to the game, to the AI or Human players. Our Makefile demonstrates the flow we have for our program.

2. Installation

2.1 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 17M

2.2 Setup and Configuration

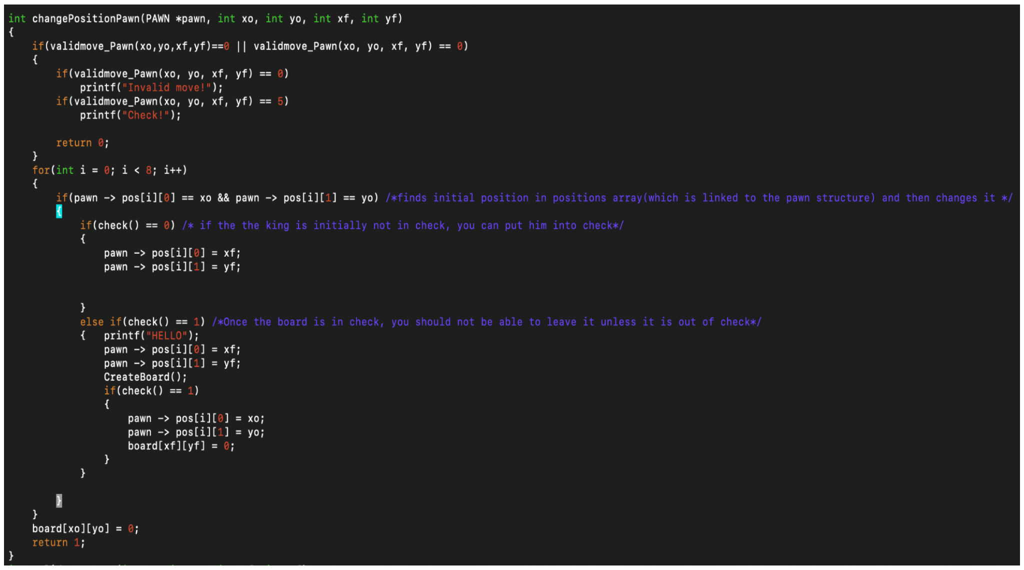
1. Unzip the tar file with the command “tar -xvzf Chess_Beta_src.tar.gz”
2. Run the makefile command “make’’
3. Run the executable with “./bin/chess”

2.3 Building, Compilation, and Installation

1. Type ‘make’ after unzipping the .tar.gz file with “tar -xvzf Chess_Beta_src.tar.gz”
2. Run the executable by using command ./bin/chess.

3 Documentation of Packages, Modules, Interfaces

3.1 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 3.10 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. 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.

3.2 Detailed Description of Functions and Parameters

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

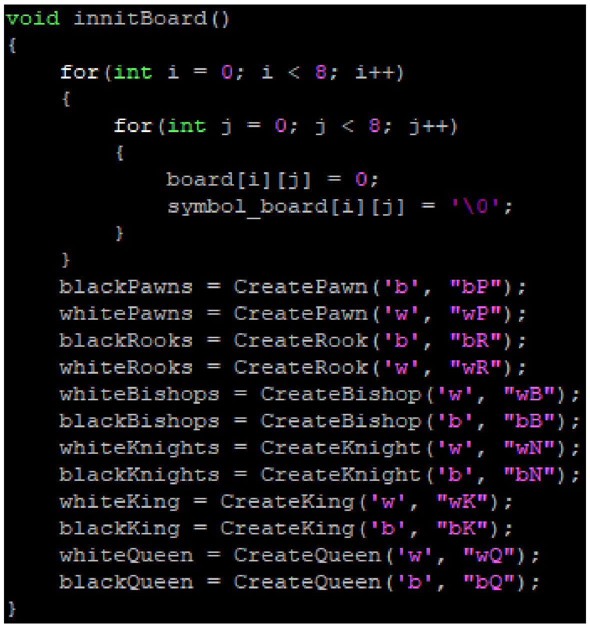
InnitBoard initializes each of the pieces for each color given the color and piece name as a parameter.

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.

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.

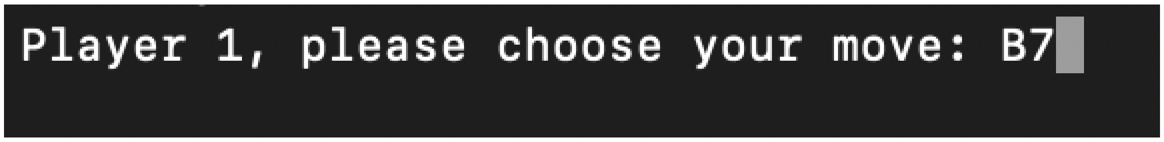


```
void innitBoard()
{
    for(int i = 0; i < 8; i++)
    {
        for(int j = 0; j < 8; j++)
        {
            board[i][j] = 0;
            symbol_board[i][j] = '\0';
        }
    }
    blackPawns = CreatePawn('b', "bP");
    whitePawns = CreatePawn('w', "wP");
    blackRooks = CreateRook('b', "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");
}
```

Figure 3.20: This function initializes our board

3.3 Detailed description of input and output formats:

Format of a move made by the user: (lower-case letter)(number)
The user types in the location they would like to go to. If that is not a valid location, then the computer returns an error message, and the player can go again.

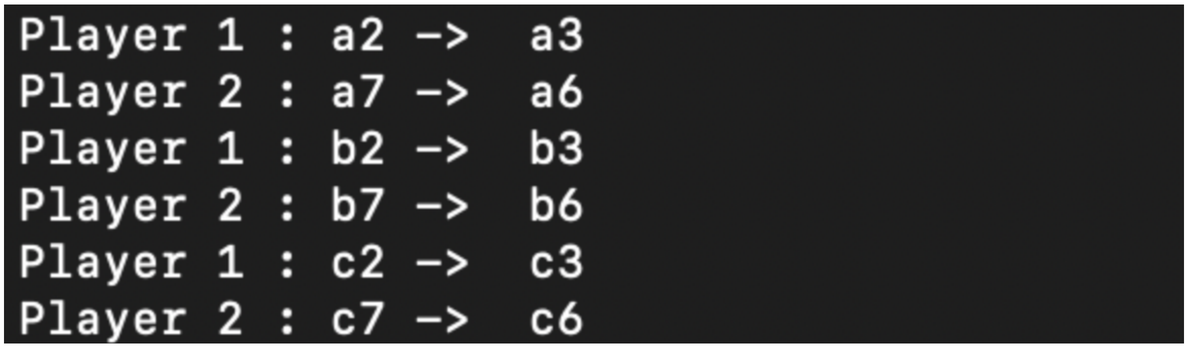


```
Player 1, please choose your move: B7
```

Figure 3.30: The message that will be displayed when asking user for input

This is how we prompt the user to input a move, the board will appear above them to view the positions of pieces throughout the chess board. We will then convert these inputs into a number for a location for each piece on the board so that we can keep track of all moves and locations for each piece.

Format of a move in the recorded log file:
The log file is a doubly linked list that contains each move in integer format (row, column)



```
Player 1 : a2 ->  a3
Player 2 : a7 ->  a6
Player 1 : b2 ->  b3
Player 2 : b7 ->  b6
Player 1 : c2 ->  c3
Player 2 : c7 ->  c6
```

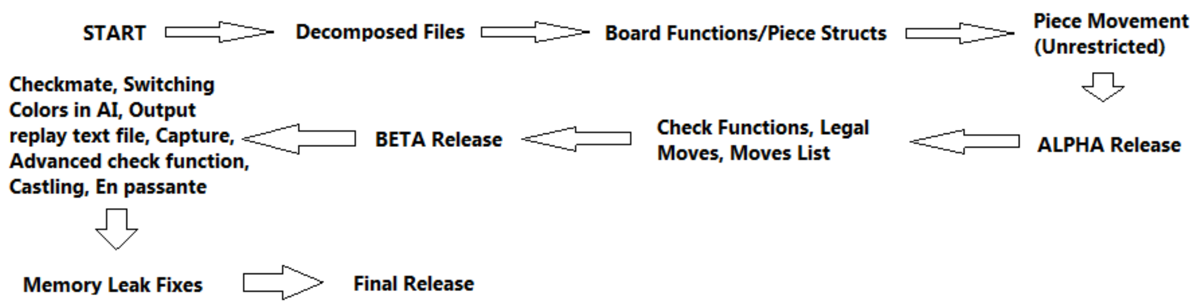
Figure 3.31: The output file that shows the move list to the reader

Above is an example of how the syntax of our log file could look like after a game. There will be more moves of course but this gives a general idea of the style we want to approach moving forward

4 Development Plan and Timeline

4.1 Partitioning of Tasks

As we move into the coding portion of our program, we will likely split up each task based on who is most comfortable with specific functions. By allowing each group member to work where they feel most comfortable we can optimize the coding process and move along with the project. Conversely, if one task is too difficult for one person to manage, we can work together in groups on these much larger functions.


START
Decomposed Files
Board Functions/Piece Structs
Piece Movement (Unrestricted)
ALPHA Release
Check Functions, Legal Moves, Moves List
BETA Release
Checkmate, Switching Colors in AI, Output replay text file, Capture, Advanced check function, Castling, En passante
Memory Leak Fixes
Final Release

4.2 Team Member Responsibilities

For our team we each have our own fields we can manage for this project.

Devin is the manager of the group so he will continuously check up on all the progress that has been made each week. He will also be in touch with all the group members to make sure they effectively contribute to the project. He has written the function to convert user input into usable information for the computer. He also wrote the moves list, and has worked on a large portion of checkmate.

Eesha is very familiar with the game of chess so she will handle helping everyone understand the game itself. She has created all of the piece structures and the functions related to them. She has also created the functions that move pieces around the board as well as delete pieces after the program is exited. She wrote the valid move functions for pawn, rook, and knight, and accounted for movement of pieces after the board is in check. She also wrote En Passante and our random AI.

Jixin will begin to focus on implementing the board and its pieces properly so that the group can work on the game and testing that it works in the coming weeks. He coded the part of the getBoard function that grabs position values and draws them, thereby putting them into a visual format for the user. He also wrote the valid move functions for bishop, king, and queen, and has completed castling.

Jonathan will be taking care of our weekly presentations, so he will control making the google doc that shows what we have accomplished the week before, and what we strive to finish by in the coming weeks. He also worked on the innitBoard and getBoard functions with Jixin. He also wrote the check function and continually debugged with fellow team members. He also handled most of the github management. Jonathan made sure that the program can quit at any time, created switching between colors as a setting and organized the manuals.

Michael has currently caught up on the written code and is working on checkmate with Devin.

Back Matter

5.1 Copyright

5.2 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.

5.3 Index