Software Specification

Minor Carlson

Version 1.0

Team 21:
Anthony Flores
Chenyu Wang
Davis Nguyen
Nathan Bae
Timothy Chan

Table of Contents

Glossary

Chess Pieces:

Pawn: The Pawn can only move forward. It can only attack diagonally. On its first move it can either move forward once or twice. Once it reaches the completely other side of the board it is able to transform into either a Bishop, Knight, Queen, or Rook.

Rook: The Rook can move vertically or horizontally for as many spaces it wants given there isn't any piece in the way. It can attack any piece horizontal or vertical.

Knight: The Knight can move in L shape so either two moves vertically and one move horizontally or two moves horizontally and one move vertically. It is able to jump over pieces. It can attack any piece it lands on.

Bishop: The Bishop can move diagonally as many spaces as it wants. It can not jump over pieces and it attacks diagonally as well.

Queen: The Queen can move diagonally, vertically, or horizontally as many spaces as it wants. It can not jump over pieces and it attacks diagonally, vertically, or horizontally.

King: The King can move diagonally, vertically, or horizontally but for only one space. It can not jump over pieces and it attacks diagonally, vertically, or horizontally.

Chess Moves and Outcomes:

Castling: Is a special move where the Ring and Rook simultaneously move towards each other. The king moves two squares towards the rook and the rook moves right next to the king on the opposite side. Castling can only be done when the King and Rook have yet to move and there are no pieces between them.

Enpassant: this special move involves the pawns. This move consists of a pawn being able to capture a pawn right after the pawn has taken two moves to avoid capture. The capture must be done right after the pawn's two space evasion. The capturing pawn can diagonally attack the opposing pawn at the square right behind the opposing pawn.

Check: When the king is being threatened it is called check. When an opposing piece has the opportunity to capture the King in the next turn if no preventative action is taken this is called

“Check”. Check forces the threatened player to take action to prevent his king from being captured in the next turn.

Checkmate: When in Check the defending player has no protective action available to save their King.

Draws:

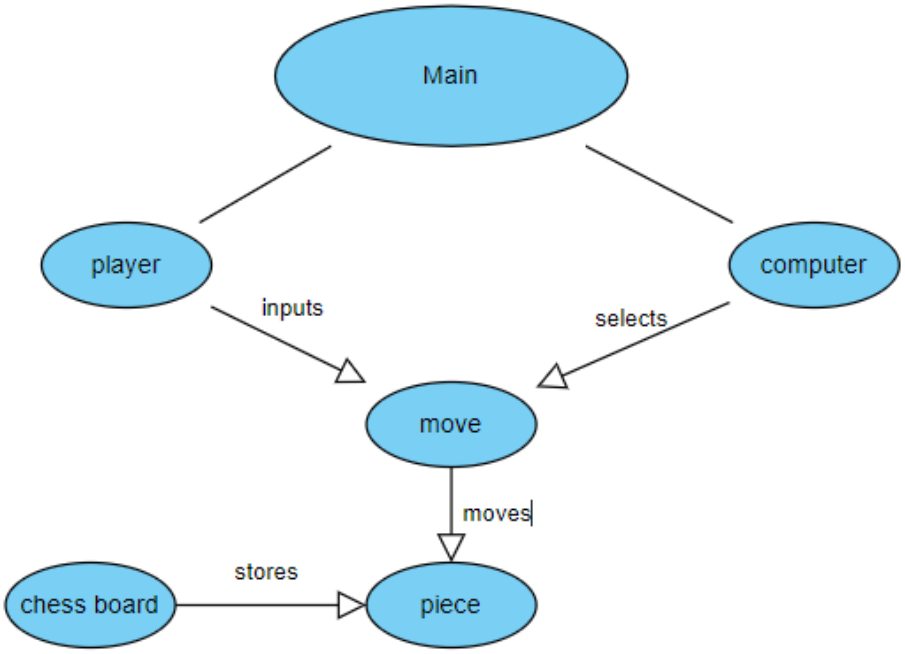
Stalemate: This occurs when a player has no possible moves and is not in Check.
Impossibility of Checkmate: This occurs when there aren't enough pieces in each of the players to produce a Checkmate.
75-move-rule: If no pawn moves or capture occurs within 75 moves, it is an automatic draw.
Draw agreement: Players are allowed to agree on a draw at any point in the game.

Software Architecture Overview

1.1 Main Data types and Structures

Struct Board{Piece Board[8][8], bool whiteHasCastled, bool blackHasCastled }
Struct Piece{char *name[3],PieceType t_piece, Player player, bool moved}
Enum PieceType{Pawn, Knight, Bishop, Queen, King, …}
Enum Player{black, white}
Struct Move:{Location location_src,Location location_dest,Piece captured_piece}
Struct Location{char rank, char file}
Log: text file

1.2 Major Software components

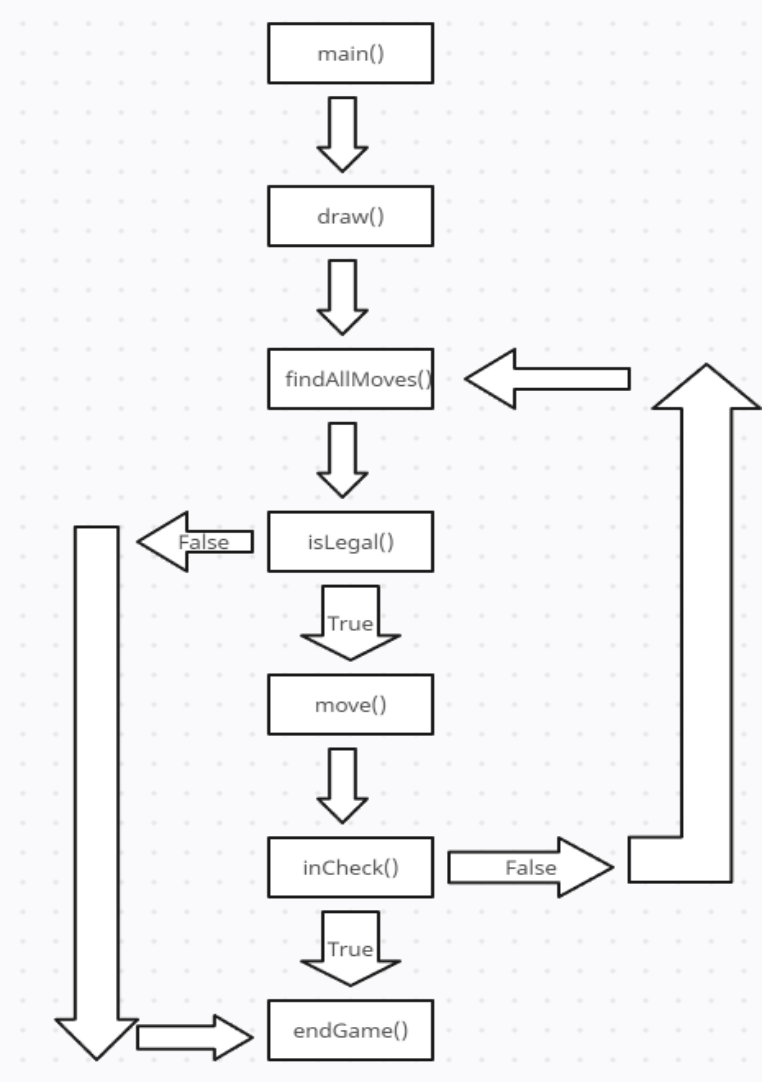


1.3 Module Interfaces

- int main()
- void move(Move move)
- Move* computer_move(Player player, Board* board)
- Moves* findAllMoves(Player player, Board* board)
- bool isLegal(Move* move, Board* board)
- bool inCheck(Board* board)
- void draw(Board* board)
- void endgame()
- Void recordMove(Move* move, Player player)

1.4 Overall program control flow

After opening the chess program, the user is asked to choose between two game modes through the use of the main function. The first game mode is Player vs. Player while the second game mode is Player vs. AI. Once the user chooses a game mode, they will be asked if they want to start black or white. After choosing the side, the game will begin. The chess board will then be generated with the function draw(). The function findAllMoves() will be used for the program to read the user's input. Function isLegal() will then be used to decide if the user input is considered a legal move. If it is a legal move, the function move will move the chess piece towards the desired location. However, if it is an illegal move, the game will end. After the legal move, the function inCheck() will check if the user input leads to a checkmate or go back to reading the user input. Once inCheck() becomes true, the game will end and the winner will be announced.



Installation

2.1 System requirements, compatibility

A functional computer (Either Windows or Mac) that is capable of running Linux. The machine can run this program in both a 32-bit and 64-bit operating system. This program requires little processing power to function. Additionally, an insignificant amount of hard drive space is required. It should be compatible with other computers running a similar program to be able to function against each other.

2.2 Setup and configuration

For the setup of the chess program, simply download the provided file and choose the location of where you want the program to be placed. Then simply run the executable file in order to open up the program.

2.3 Building, compilation, installation

To install the chess program, the user will compile the chess program through its tar.gz file. Type “compile program” for the Linux system to compile all the necessary files required for the program to function. The installation is finished after all the files are compiled. To uninstall the program, simply delete the files that were compiled from the tar.gz file.

Packages, Modules, and Features

3.1 Data Structures

```
#ifndef CHESS_GAME_H
#define CHESS_GAME_H

// Define the different piece types
enum PieceType {
  KING,
  QUEEN,
  ROOK,
  BISHOP,
  KNIGHT,
  PAWN,
  EMPTY
};

// Define the different players
enum Player {
  BLACK,
  WHITE,
  EMPTY
};

// Define a struct to represent a chess piece
struct Piece {
  char *name3[3];
  PieceType t_piece;
  Player player;
  bool moved;
  bool captured;
};

// Define a struct to represent a chess move
struct Move {
  Location location_src;
  Location location_dest;
  Piece captured_piece;
};

//Define a struct to store a list of moves
struct Moves {
  int size;
  Move moveList[];
};
```

```
// Define a struct to represent a chess board location
struct Location {
  int rank;
  int file;
};

// Define the chess board as an 8x8 array of Piece structs
struct Board{
    Piece* board[8][8];
    bool whiteHasCastled;
    bool blackHasCastled;
}

#endif // CHESS_GAME_H
```

3.2 Functions

int main()

This function:
1)sets up the board;
2)initializes the game;
3)creates a log file for the game;
4)lets the human player choose the sides;
5)runs the loop until the game ends if one player wins or an illegal move occurs.

void move(Move* move)

input: instance of struct move
Output: none
This function uses the input information to move a certain piece to a certain location, if there's capture involved, the capture will be automatically done and recorded in the log file

Move* computer_move(Player player, Board* board)

Input:none
Return value: instance of struct Move
Invoked in main() when it's the computer's turn to move.

Moves* findAllMoves(Player player, Board* board)

Input: location of board and all pieces, and what piece color is moving.

Return value: an instance of struct Move containing an array of all valid possible moves for each piece on the board for specified player color.

bool isLegal(Move* move, Board* board)

input: none
Return value: True if the move is legal, false otherwise
Output: print out the error if the move is illegal
This function takes the input data of the function move and evaluates whether it's a legal move. It'll print out information if it's not legal and end the game using endgame() Function

bool inCheck(Board* board)

Input: Board with current status
Return value: True if the board is in check, false otherwise
Output: Prints out warning to computer or to player if previous move resulted in a check
This function takes the input data of the previous move and evaluates whether that move resulted in a check. It will print a warning on the screen indicating that a check has happened and you must resolve the check

void draw(Board *board)

Input: struct board
Output: print out the board in the user interface
This function uses the board information to print out the board in the user interface

void endgame()

Input: none
Output: announcement about the result of the game
This function will be invoked in main() directly or in isLegal() if an illegal move occurs.

Void recordMove(Move* move, Player player)

Input: the move that has been made and the player that made the move
This function records the move into the log file

3.3 Input and Output Formatting

User input: The input would be read in the form of a 4 character string like “A1A4” It would be stored in variables to store the initial position and the resulting position
Output: The log file would be recorded in a text file. It would be 2 columns with the initial piece and position on the left and the resulting position on the left. This would Alternate between black and white.

Development Plan and Timeline

4.1 Partitioning of Tasks

All group members are tasked to complete the User Manual and Software Specification.

Data Structure, Game Modes, Log Files: Chenyu Wang
AI Creator: Davis Nguyen
User Input, Interface: Nathan Bae
Valid User Inputs: Anthony Flores
Checkmate and Ending Game: Timothy Chan

4.2 Team Member Responsibilities

The responsibilities of all team members will be divided upon completing the stated functions.

Anthony Flores: findAllMoves(), isLegal()
Chenyu Wang: main(), structure header file
Davis Nguyen: computer_move(), choose_AI_Move()
Nathan Bae: move(), draw()
Timothy Chan: inCheck(), endgame()

Back matter

• Copyright

• Error messages

Illegal Moves

- “There is no piece at selected location”
- “That piece cannot move to selected position”
- “That piece is not yours”

• Index

• References

http://www.wachusettchess.org/ChessGlossary.pdf
https://cdn.shptrn.com/media/mfg/1725/media_document/8976/Imp_ChessR.pdf?1401227342