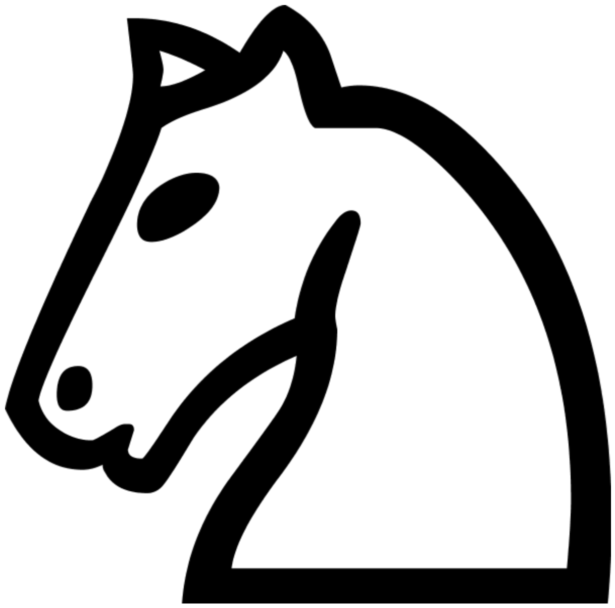
White Knights

Software Architecture Specifications



Producers:
Brian Le
Tushar Jain
Gianpaolo Gutierrez
Rochelle Naing
Jinren Zhou

Affiliated with UCI

Version 1.0 Final Release

Table of Contents

Software Architecture Overview

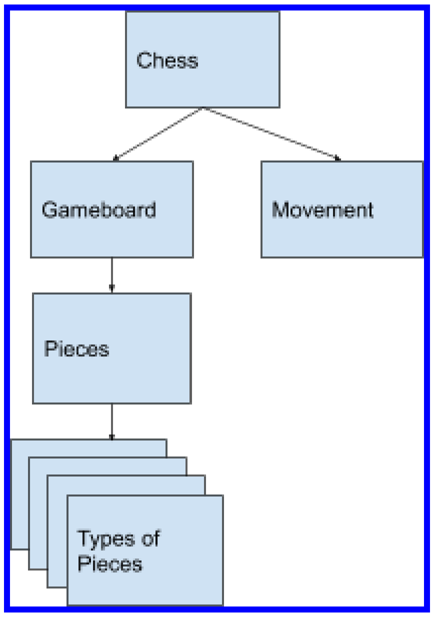
1.1 Main Data types and Structure:

For the chess pieces, a structure is implemented to record their piece's data for name, viable movements, and location. For the chessboard, a 2D array is implemented to create all the squares of the chessboard.

1.2 Major Software Components:

The game creates a gameboard that initializes pieces on the board. It then loops by players moving pieces until there is a winner or a draw.

1.2.1 Diagram of Module Hierarchy:



1.3 Module Interface:

ChessBoard Module:

- *void startBoard(PIECE *Board[8][8]);*
- *void copyBoard(PIECE *Board1[8][8], PIECE *Board2[8][8]);*
- *void setPiece(PIECE *piece, PIECE *(*ptr)[8], int color, TYPE *piecetype, int x, int y);*
- *void updateMoves(PIECE *Board[8][8]);*

Piece Module:

- *void updatePawn(PIECE *p);*
- *void updateRook(PIECE *p);*
- *void updateKnight(PIECE *p);*
- *void updateBishop(PIECE *p);*
- *void updateQueen(PIECE *p);*
- *void updateKing(PIECE *p);*
- *int checkPiece(PIECE *p, int newx, int newy);*
- *int checkSquare(PLAYER *player, int x, int y);*
- *void movePiece(PIECE *Board[8][8], PIECE *p, int newx, int newy);*
- *int EnPassant(PIECE *p, int newx, int newy);*
- *int Check(PIECE *Board[8][8], int color);*
- *int Mate(PIECE *Board[8][8], int color);*
- *void PawnPromotion(PIECE *p, TYPE *typelist[6]);*

Player Module:

- *PLAYER *createPlayer(int color, PIECE *Board[8][8]);*
- *void updatePlayer(PLAYER *player, PIECE *Board[8][8]);*
- *void pawnAttack(PLAYER *player, PIECE *piece);*

- *void rookAttack(PLAYER *player, PIECE *piece);*
- *void knightAttack(PLAYER *player, PIECE *piece);*
- *void bishopAttack(PLAYER *player, PIECE *piece);*
- *void queenAttack(PLAYER *player, PIECE *piece);*
- *void kingAttack(PLAYER *player, PIECE *piece);*

Display Module:

- *void printBoard(PIECE *Board[8][8]);*

Log Module:

- *void writefile(PIECE *p, int newx, int newy);*
- *int convertX(int x);*
- *int convertY(int y);*
- *void readfile(void);*

Conversion Module:

- *int convertLetter(char oldletter);*
- *int convertNumber(int oldnumber);*

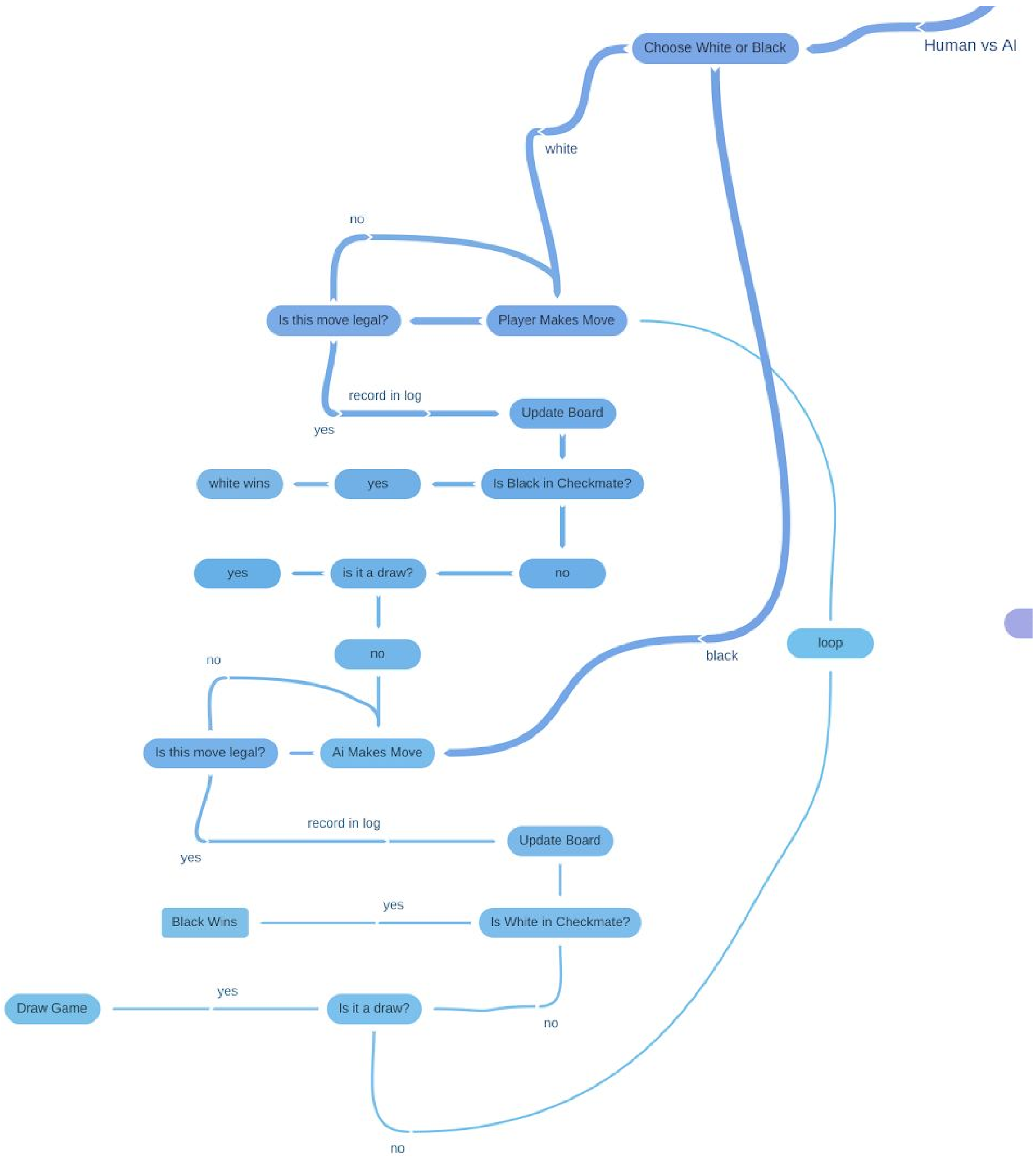
AI Module:

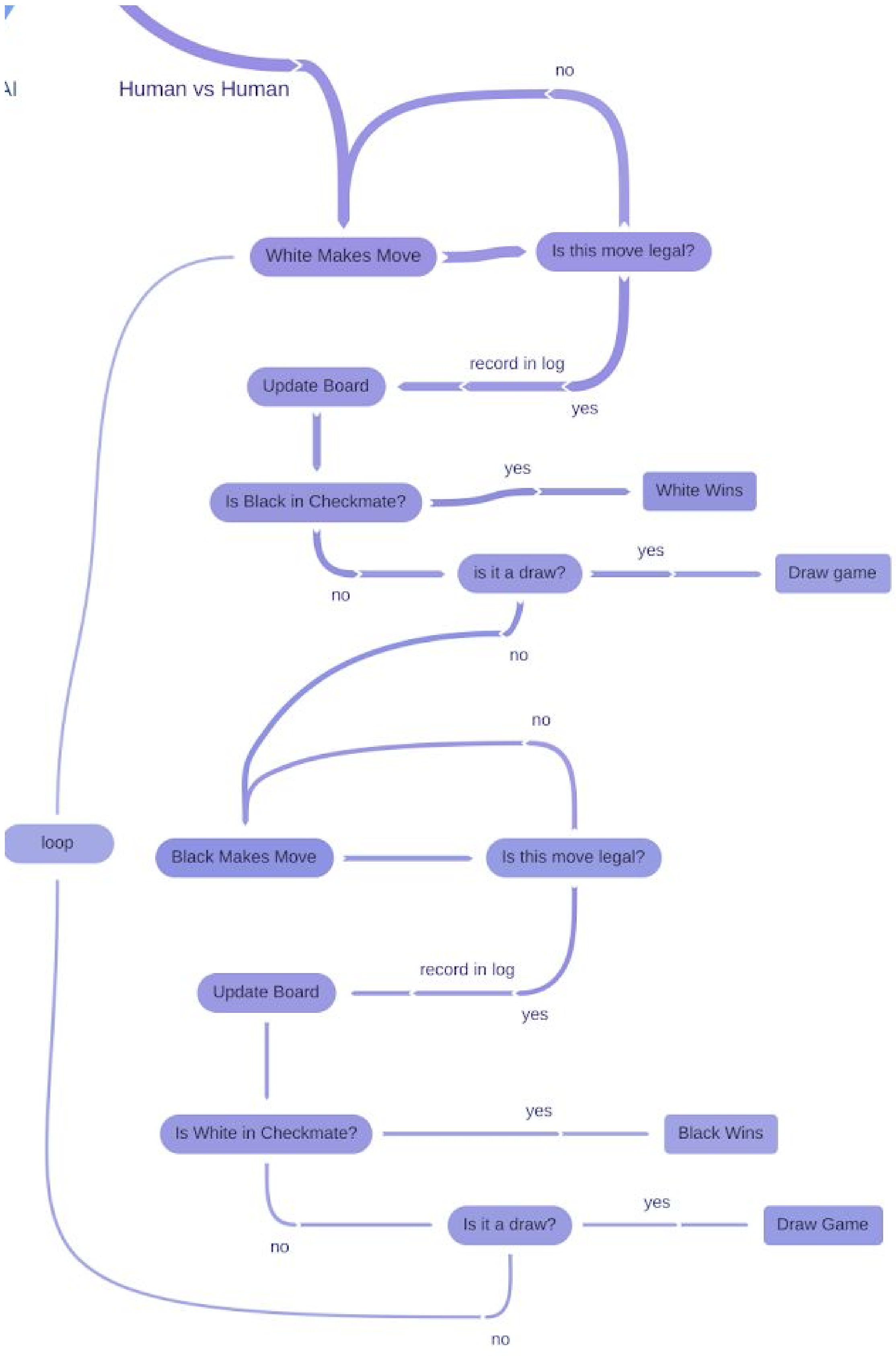
- *void aifunction(PIECE *Board[8][8], int color, int *level, int *oldx, int *oldy, int *newx, int *newy);*

1.3.1 API of Major Module Functions

- We need "stdio.h" for our file related input/output functions.
- We need "stdlib.h" for our memory allocation and freeing functions.

1.4 Overall Program Control Flow





AI
Human vs Human
no
White Makes Move
Is this move legal?
record in log
Update Board
yes
yes
Is Black in Checkmate?
White Wins
yes
is it a draw?
Draw game
no
no
no
loop
Black Makes Move
Is this move legal?
record in log
Update Board
yes
yes
Is White in Checkmate?
Black Wins
yes
Is it a draw?
Draw Game
no
no

Installation

2.1 System Requirements:

- Hardware PC: Hardware x86_64 server
- Operating System Required: Linux OS
- Third Party Software Required
  - GCC
- Dependent Libraries : Math Library (Math.h), SDL for graphical interface.
- Xming

2.2 Setup and Configurations:

1) Open PuTTy (log into a server).
2) Type command *'cd'* (Change the current directory) to find the correct folder which the game was downloaded into.
3) Extract the src code by typing '*tar -xvzf Chess_V1.0_src.tar.gz'*.
4) Then type *'cd Chess_V1.0_src'* to change directories into the source file.

2.3 Building, Comparison, Installation

After downloading and extracting the necessary files, follow the steps below for installation.

1) Type '*make*' command to build the program.
2) Type *'./bin/Chess'* to run the program.
   OR
1) Type '*make test'* command to build test program
2) Type *'./bin/ChessTest'* to run the program

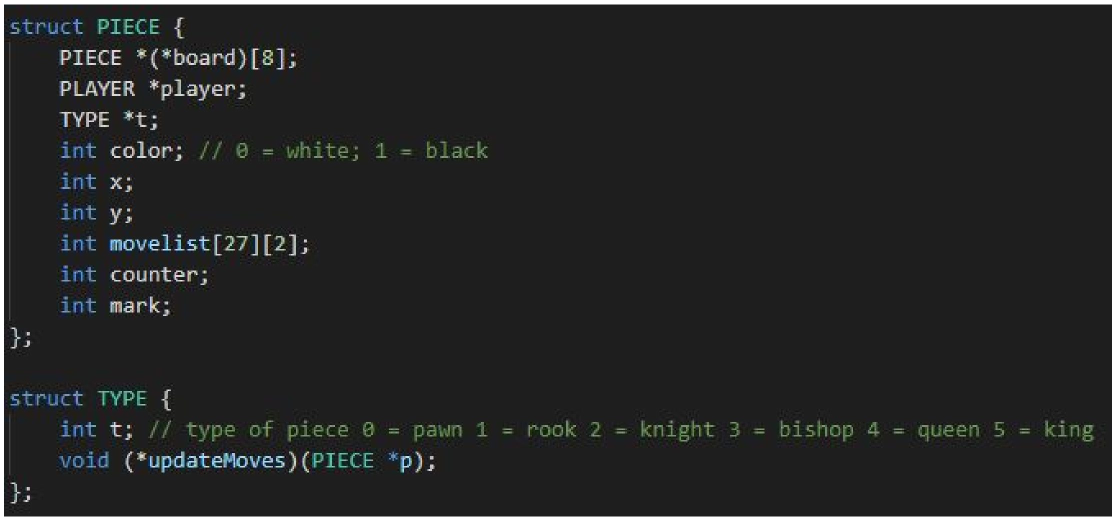
The test environment automatically picks ai vs ai and continues until checkmate or draw.

Documentation of Packages, Modules, Interfaces

3.1 Detailed Description of the data structures :

1) The chessboard has a 2D array with pointers to each chess piece structure.
2) The piece structure will have information of each corresponding chess piece with a list of their viable moves sets, color, piece type, and index number.
3) The board will be handled as a 2D array that acts like a grid for the chess pieces to be placed on.

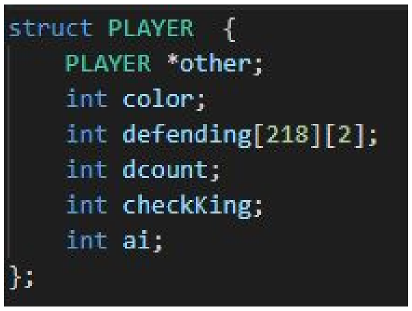
3.1.1 Critical Snippets of Source Code:



```
struct PIECE {
    PIECE *(*board)[8];
    PLAYER *player;
    TYPE *t;
    int color; // 0 = white; 1 = black
    int x;
    int y;
    int movelist[27][2];
    int counter;
    int mark;
};

struct TYPE {
    int t; // type of piece 0 = pawn 1 = rook 2 = knight 3 = bishop 4 = queen 5 = king
    void (*updateMoves)(PIECE *p);
};
```

Figure 1: "Code snippet of the data structure of PIECE"



```
struct PLAYER  {
    PLAYER *other;
    int color;
    int defending[218][2];
    int dcount;
    int checkKing;
    int ai;
};
```

Figure 2: "Code snippet of the data structure of PLAYER"

3.2 Detailed Description of Functions and Parameters

ChessBoard Module:

- *void startBoard(PIECE *Board[8][8]);* - ***Initializes board with chess pieces in their initial position***
- *void copyBoard(PIECE *Board1[8][8], PIECE *Board2[8][8]);* ***- Copies Board1 to Board2***
- *void setPiece(PIECE *piece, PIECE *(*ptr)[8], int color, TYPE *piecetype, int x, int y);* - ***Sets the initial information of the starting pieces with the following color, type, and location***
- *void updateMoves(PIECE *Board[8][8]);* - ***updates each piece position***

Piece Module:

- *void updatePawn(PIECE *p);*
- *void updateRook(PIECE *p);*
- *void updateKnight(PIECE *p);*
- *void updateBishop(PIECE *p);*
- *void updateQueen(PIECE *p);*
- *void updateKing(PIECE *p);*

These functions above have their own set of movement rules and update the pieces' movelist based on the movement of that certain piece.

- *int checkPiece(PIECE *p, int newx, int newy);* -***Checks if the new piece can move to x and y and returns 1 if it is.***
- *int checkSquare(PLAYER *player, int x, int y);* - ***Checks if x,y location is attacked by a player and returns 1 if it is.***
- *void movePiece(PIECE *Board[8][8], PIECE *p, int newx, int newy);* - ***Properly moves a piece to the desired position based on user input of x and y coordinates.***

- *int EnPassant(PIECE *p, int newx, int newy); -****Looks at the new location and checks if an En Passant move is possible and returns 1 if it is.***
- *int Check(PIECE *Board[8][8], int color); -* ***Checks if the king is in Check and returns 1 if it is.***
- *int Mate(PIECE *Board[8][8], int color); -* ***Checks if there are any legal moves that the player of that color can make***
- *void PawnPromotion(PIECE *p, TYPE *typelist[6]); -* ***Promotes the pawn to the type given from typelist***

<u>Player Module:</u>

- *PLAYER *createPlayer(int color, PIECE *Board[8][8]); -* ***Initializes player with certain colors***
- *void updatePlayer(PLAYER *player, PIECE *Board[8][8]); -* ***Updates the available moves of the player***

These functions below add all the moves that piece can attack into the 2D array called defending.

- *void pawnAttack(PLAYER *player, PIECE *piece);*
- *void rookAttack(PLAYER *player, PIECE *piece);*
- *void knightAttack(PLAYER *player, PIECE *piece);*
- *void bishopAttack(PLAYER *player, PIECE *piece);*
- *void queenAttack(PLAYER *player, PIECE *piece);*
- *void kingAttack(PLAYER *player, PIECE *piece);*

<u>Display Module:</u>

- *void printBoard(PIECE *Board[8][8]); -* ***prints board w/ updated positions***

Log Module:

- *void writefile(PIECE *p, int newx, int newy); -* ***Records and writes movements of pieces into a readable text file***
- *int convertX(int x);* ***To flip the given numbers around and start from 1 and return the new number***
- *int convertY(int y); -* ***Converts the corresponding numbers into letters of A-H and return those letters***
- *void readfile(void); -* ***If a log file already exists, it reads it in and continues writing the new logs into it***

Conversion Module:

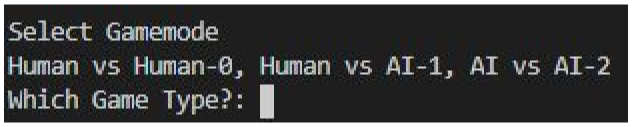
- *int convertLetter(char oldletter); -* ***Converts the given letter A-H to its corresponding number and return that number***
- *int convertNumber(int oldnumber); -* ***To flip the given numbers around and start from 1 and return the new number***

AI Module:

- *void aifunction(PIECE *Board[8][8], int color, int *level, int *oldx, int *oldy, int *newx, int *newy); -* ***Calculates the best move possible for the AI to perform.***

3.3 Detailed Description of Input and Output Formats:

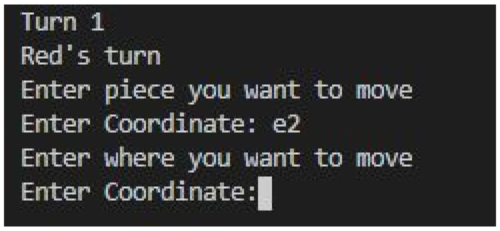
When the game begins, the user is given the option to play against a human or an AI.



```
Select Gamemode
Human vs Human-0, Human vs AI-1, AI vs AI-2
Which Game Type?:
```

Figure 3: Select a Gamemode

***If the user picks 0, then the following is prompted.

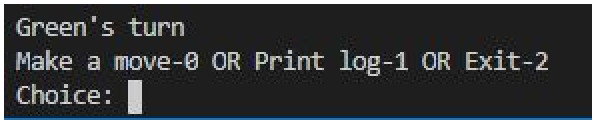


```
Turn 1
Red's turn
Enter piece you want to move
Enter Coordinate: e2
Enter where you want to move
Enter Coordinate:
```

Figure 4: How to Move

It starts with the turn number along with which player's turn it is.
In order to move a piece you first enter the coordinate of the piece you want to move followed by the coordinate of the place you want to move it. This move must be a valid move in accordance with the rules of chess or else the program will ask you for a different move.

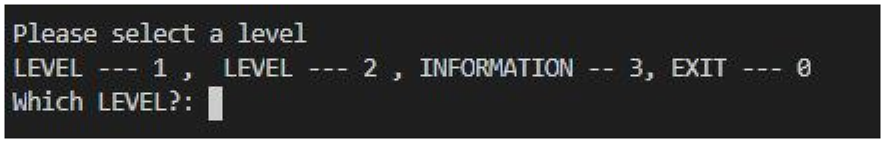
After the turn is finished, the turn is passed over to the next player and asks them if they want to continue making a move, view the log file, or exit the game.



```
Green's turn
Make a move-0 OR Print log-1 OR Exit-2
Choice:
```

Figure 5: Turn Options

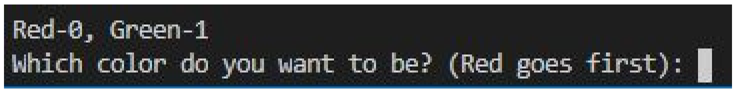
***If the user picks 1, then the level selection of the AI is prompted.



```
Please select a level
LEVEL --- 1 ,  LEVEL --- 2 , INFORMATION -- 3, EXIT --- 0
Which LEVEL?:
```

Figure 6: Select an AI Level

After a level is chosen, the user is then prompted for a color choice.

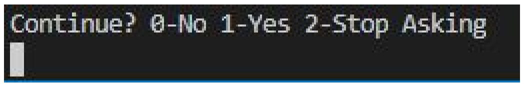


```
Red-0, Green-1
Which color do you want to be? (Red goes first):
```

Figure 7: Choose a Color

After choosing a color, the game follows the same format as when the user first picks 0.

***If the user picks 2, then the level selection of the AI is prompted (shown in Figure 6). After each round, the user is asked if they want to continue to the next round.



```
Continue? 0-No 1-Yes 2-Stop Asking
```

Figure 8: Continue Playing?

Development Plan and Timeline

4.1 Partitioning of Tasks

Tasks:

1. Creating the GUI
2. Creating the Main structure of the game
3. Implementing the rules of chess
4. Error Checking
5. Implementing an AI
6. Create User and Developers Manuals

Timeline:

Week 1: Setup team accounts and have an understanding of ‘CVS’ or ‘git’.

Week 2: Create initial versions of User Manual.

Week 3: Create initial versions of Developers Manual and Started Coding GUI and Main Structure.

Week 4: Release Alpha version of program.

Week 5: Develop AI for our program with Beta version release. Nearly all chess game functions should be implemented.

Week 6: Preparations for Pre-Season Tournament. AI development.

4.2 Team Member Responsibilities

Brian Le: Worked on the main structure and function of the game. Created the ASCII UI.

Tushar Jain: Working on creating a log file for the user and GUI. Updated manuals.

Gianpaolo Gutierrez: Revised coordinate system and user input. Prototyped GUI.

Rochelle Naing: Worked on moving the chess pieces, implementing capturing, En Passant, and Pawn Promotion

Jinren Zhou: Worked on AI and prototype for a GUI menu.

Copyright

References

https://lichess.org/

http://www.wachusettchess.org/ChessGlossary.pdf

Index