Chess

Software Architecture Specification

Fig. 1

Team 10: **Engineers in Pajamas**

Arain, Hafsah Aymen

Chang, Jeremy Raphael

Li, Hui Quan

Padmawar, Aadi

Parvez, Ayesha

Yan, Yu Chen

University of California, Irvine

Version 1.0

Table of Contents

Glossary

- **AI:** artificial intelligence, when a computer performs an activity that is thought to require human intelligence
- **Algorithm:** sequence of steps used to perform calculations or solve a problem, finite
- **API:** stands for application programming interface, how certain features of an application can be accessed
- Array: contains multiple values of the same type, arranged in brackets, e.g. [3,5,1,0,100,3]
- **CentOS:** type of Linux distribution
- **Character:** a data type, smallest range, strings usually represented with this
- Command line: used to process commands, can be used to handle files, install and uninstall packages, etc. usually without the use of a graphical interface
- **Control Flow:** shows the order of the program, i.e. which functions/instructions are called when
- **Data Type:** there are two categories of data types, primary (dictates the range of the data and can be unsigned or signed, e.g. char, int, short, long, float, double) and derived (dictates how the information is grouped together, e.g. arrays, structures, pointers, etc.)
- **Enumerators:** user defined, can be used to assign certain names to a set of values
- **Function:** a block of instructions, takes in certain variables (but doesn't have to), performs calculations on them and can return a value if needed
- **Github:** website that hosts repositories in order for people to collaborate on and share programs
- **Integer:** a data type, used to represent whole numbers
- **Linux:** type of operating system
- **Module:** entire program is split into several files, or modules, each module contains some of the functions necessary for the program

- **Module hierarchy:** used to show the relationship between the modules of the program
- **Pointer:** points to the memory address of another variable
- **String:** a word/words, represented as an array of characters
- **Structure:** user defined, can be used to dictate how a collection of variables are grouped together
- **Untar:** extract files from tarball

1. Software Architecture Overview

1.1 Main data types and structures

Main Data Types and Data Structures

- Arrays
- Characters
- Enumerators
- Integers
- Pointers
- Structures
- Two-Dimensional Array

Main Structures

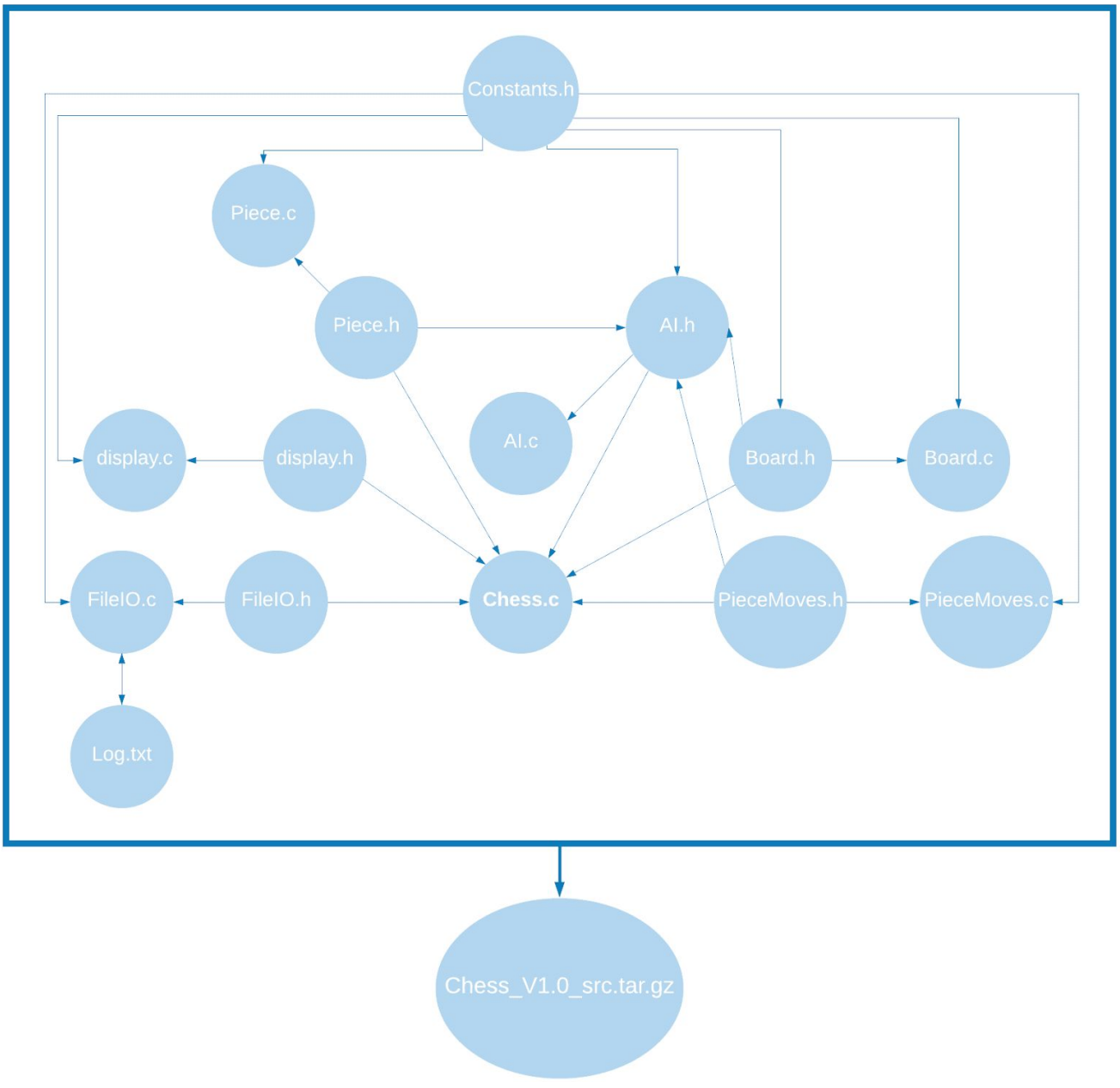
- **Board**
    - The Board Structure contains a character array type location that stores the current location. two integers type x ,y stores the row and column of that location, and a Piece type object piece at that location.
- **Pieces**
    - The Pieces structure contains a character array to store the name of the piece. An enum type color stores the piece's color, a Board type object PossibleMoves stores all the possible moves the piece can make, a pointer to a function that calculates all the possible moves the

piece can make and an Integer type points to represent the point it's worth.

- **Player**
    - The player structure contains a character array type name stores the name of the player and an enum type color stores the side/color of the play(black or white).

1.2 Major software components

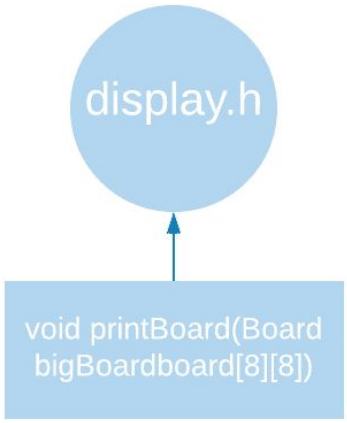
Diagram of Module Hierarchy



1.3 Module interfaces

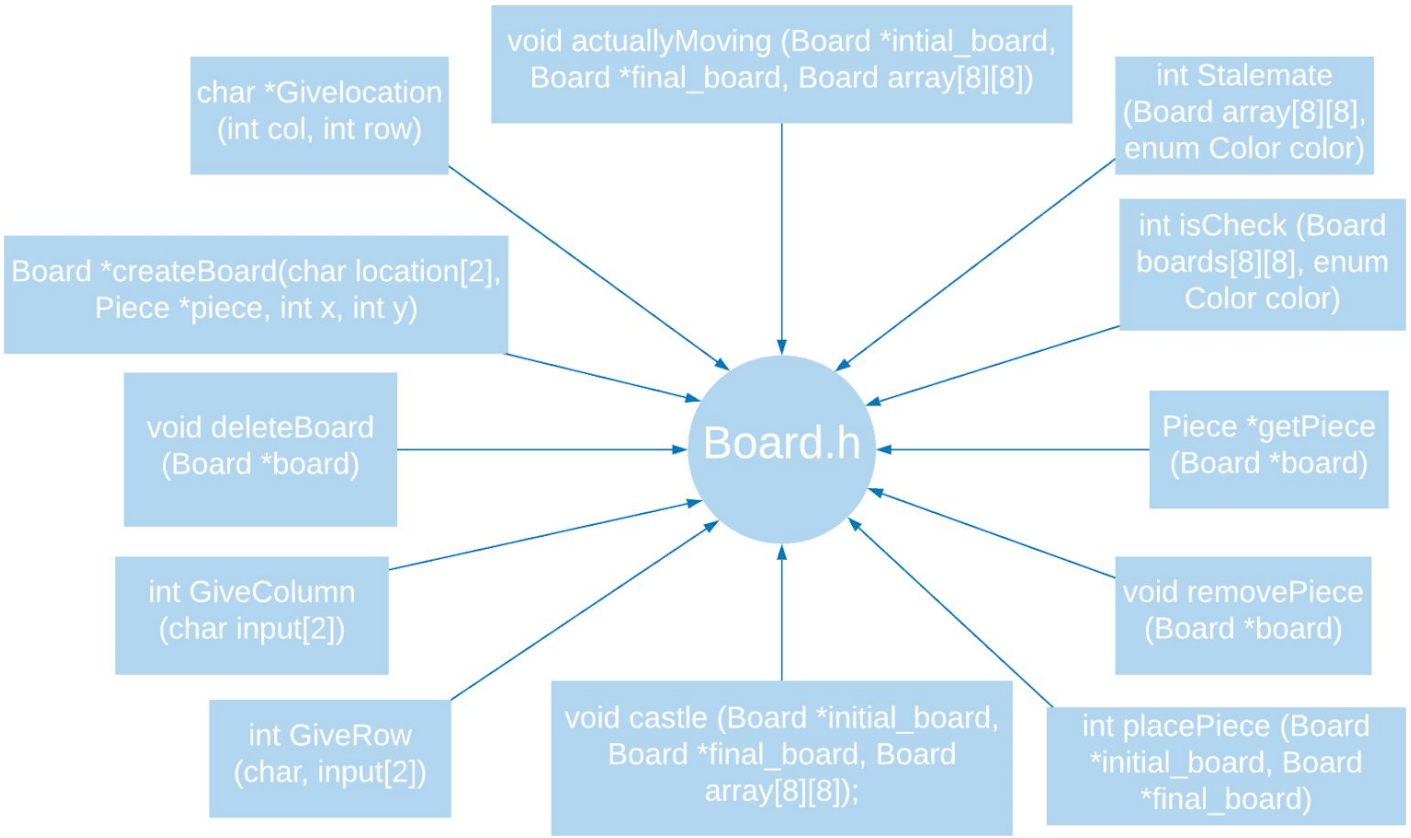
API of major module functions

display.h

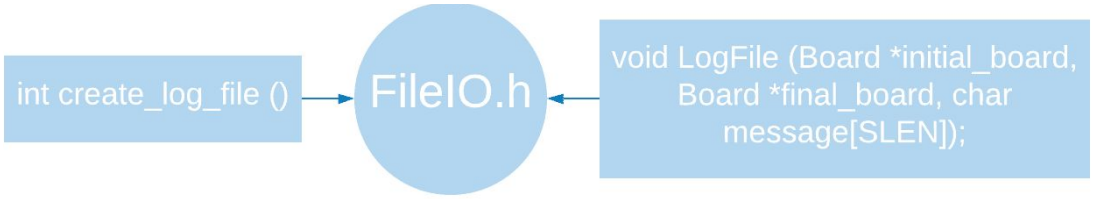


AI.h

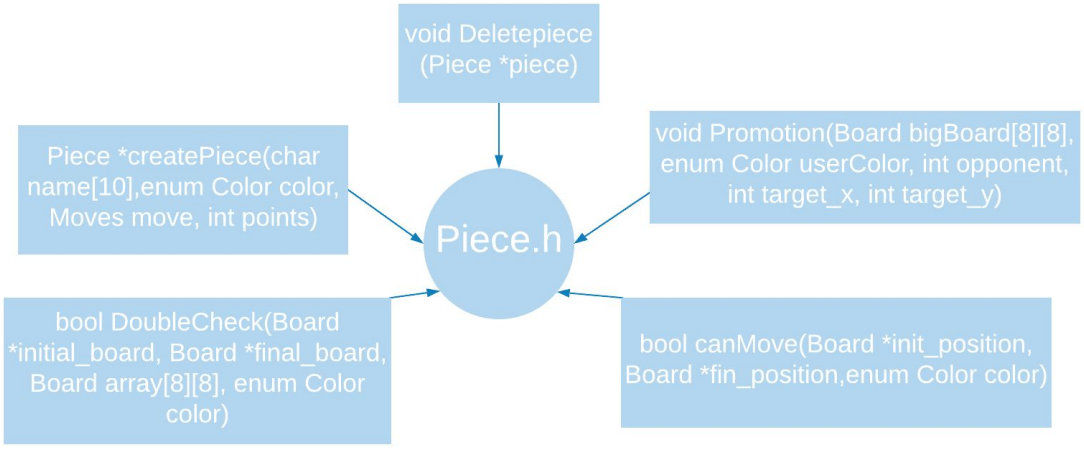
Board.h



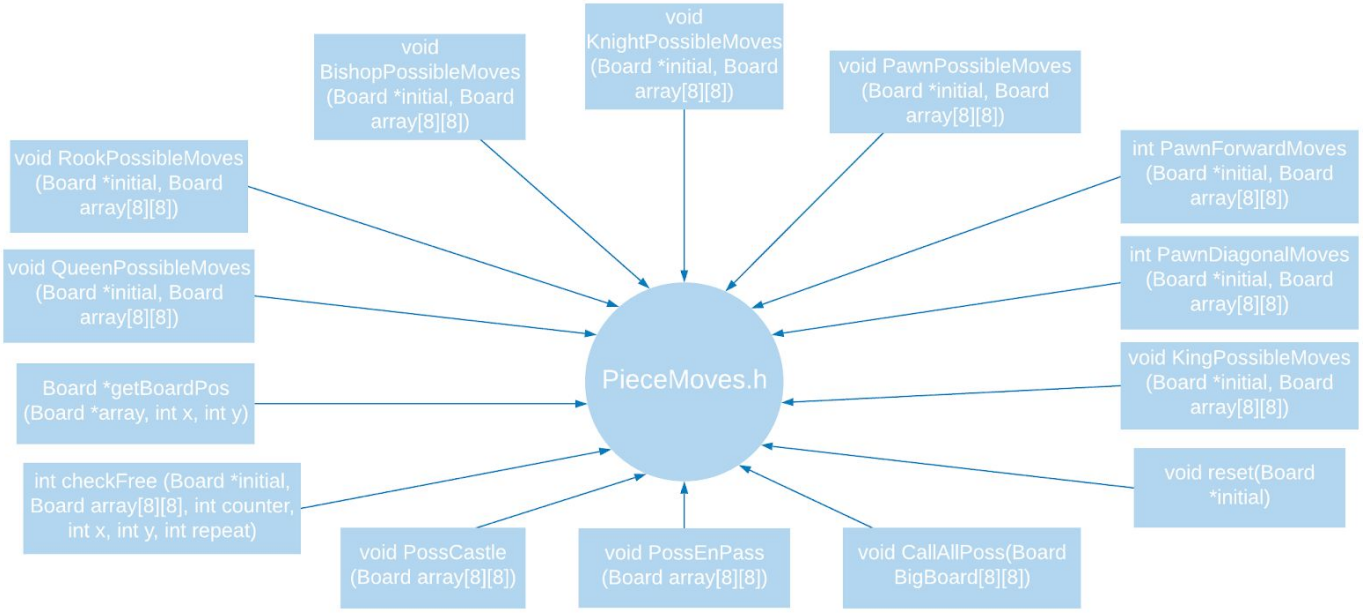
FileIO.h



Piece.h

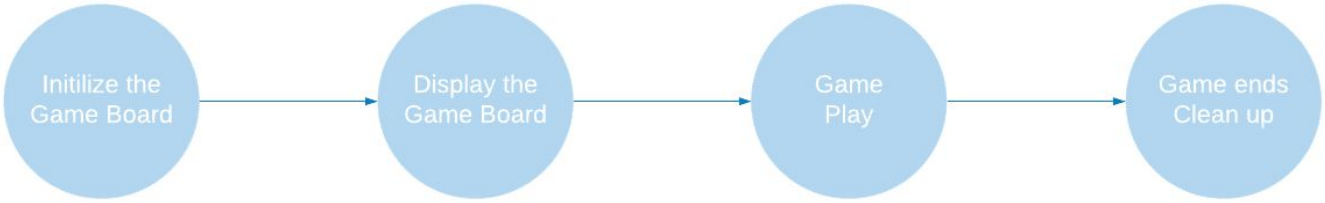


PieceMoves.h

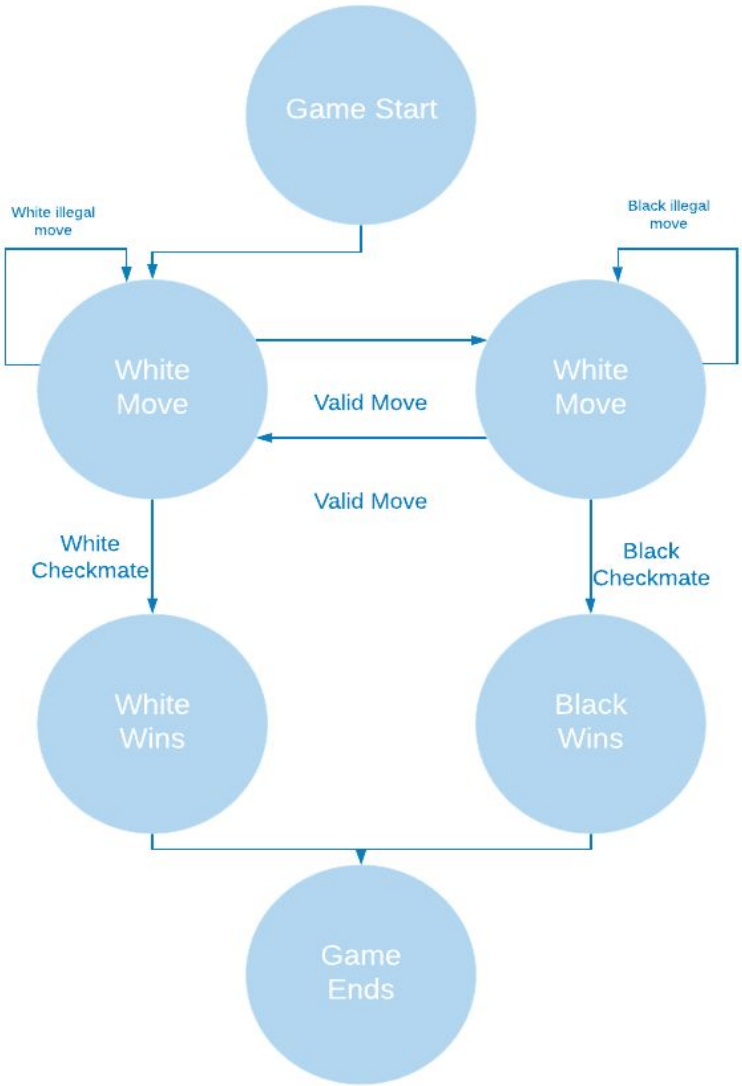


1.4 Overall Program Control Flow

Main Control Flow



Game Play Control Flow



2. Installation

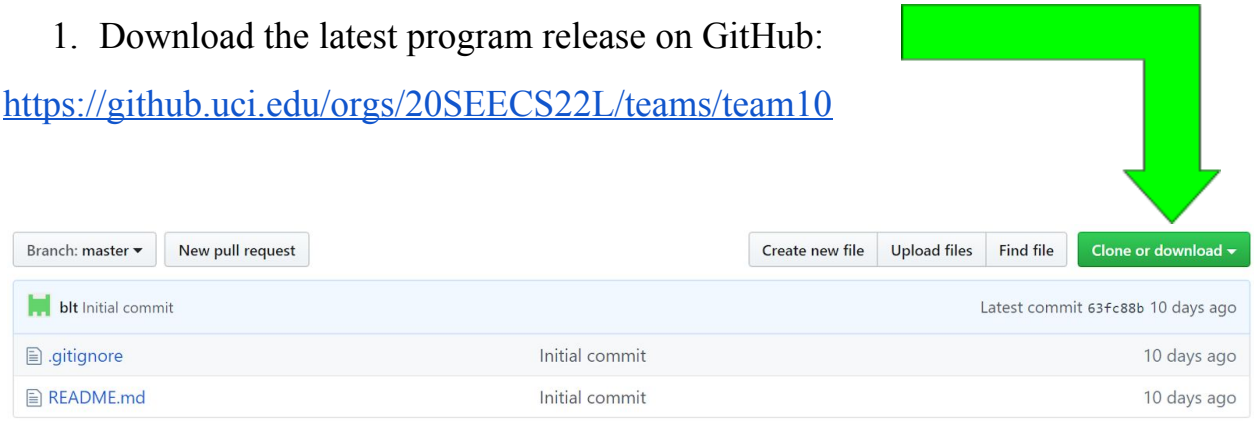
2.1 System requirement

- RedHat Linux that runs on CentOS release 6.10 or later
- 64-bit CPU
- Enough storage for program

2.2 Setup and Configuration

1. Download the latest program release on GitHub:

https://github.uci.edu/orgs/20SEECS22L/teams/team10



2. Allow permission to compile the file
3. Untar (or unzip) the downloaded file through linux command line:

```
% gtar xvzf Chess_V1.0_src.tar.gz

% evince chess/doc/Chess_UserManual.pdf

% cd chess

% make

% make test
```

2.3 Uninstalling

1. Run ‘make clean’ on linux command line:

```
% make clean
```

3. Documentation of Packages, Modules and Interfaces

3.1 Data Structures and Data types

- Arrays
    - Multiple one and two dimensional arrays have been used in the program to store data such as pieces of each color, locations that the piece can move to and all the locations on the chess board.
    - Example:

```
// array of white pieces
Piece *WhitePieces[16];

// array of black pieces
Board boardArray[8][8];
```

- Type definitions
    - Moves

```
typedef void *(*Moves)(struct Board *initial, struct
Board *array);
```

    - possibleMoves

```
typedef void *(possibleMoves)(Board *board_struct);
```

- Structures
    - Piece
        - Defined in Constants.h, this structure is used to create objects that represent pieces in the game.
        - It contains an int variable to store points, a character array for the name of the piece, an enumerator that defines the color of the piece and an array of type int that uses 1 or 0 to indicate that a location is available or occupied, respectively.
        - The structure also contains a pointer to a function that defines all possible moves that the piece can make and two int variables hasMoved and lastMoved. hasMoved is incremented by 1 every time the piece is moved, while lastMoved is 1 if the piece was the last piece to be moved.

```
struct Piece{
      char PieceName[10];
      enum Color PieceColor;
      int points;
      int *PossibleMoves;
      Moves move;
      int hasMoved;
      int lastMoved;
};
```

- Board
  - Defined in Constants.h, this struct contains two int variables to store the row and column of a location, an a pointer of type Piece that refers to the piece currently placed at that location and a character array that stores the current location. It also contains WhiteHit and BlackHit which store 1 if a white or black piece can hit a specific square, respectively.

```
struct Board{
      Char location[2];
      int row;
      int col;
      Piece *piece;
      int WhiteHit;
      int BlackHit;
}Board;
```

- Enumerator
  - Color
    - Defined in Constants.h, this enumerator assigns a value of 0 to White and 1 to Black.

```
// White = 0, Black = 1
enum Color{
      White,
      Black
};
```

- Other data types that have been used include int, bool, double, char and void.

3.2 Functions

Board.c

`Board *createBoard(char location[2], Piece *piece, int x, int y)`
Creates and returns a board structure.

`void deleteBoard(Board *board)`
De-allocates the board structure from the heap

`int actuallyMoving(Board *initial_board, Board *final_board, Board array[8][8])`
Moves a piece from one place to another.

`void castle(Board *initial_board, Board *final_board, Board array [8][8])`

`Piece *getPiece(Board *board)`
Returns the piece that is contained in the board struct

`void removePiece(Board *board)`
Changes the pointer of the piece in the board struct to NULL

`int placePiece(Board *inital_board, Board *final_board)`
Changes the pointer of the piece in the board struct to parameter given

`int isCheck(Board boards[8][8], enum Color color)`
Called when the king is getting attacked. Returns 0 or 1 based on whether it is in check. The parameter is the pointer of the board array

`int GiveRow(char input[2])`
Returns the row value from the user input. (ex A4 gives 3)

`int GiveColumn(char input[2])`
Returns the column value from the user input. (ex A4 gives 0)

`Char *Givelocation (int col, int row)`
Returns string location from x and y locations (ex 0, 0 gives A1)

`int Stalemate(Board array[8][8], enum Color color)`
Returns 1 if it is stalemate/if the color side has no possible moves

AI.c

```
double BestMove(Board b_array[8][8], enum Color MyColor, enum
userSide, int opponent)
```
Calculates the AI's move and does it. The parameter are board array and computer's color It will also print the moves for AI.

display.c

```
void printBoard(Board bigBoard[8][8])
```
Displays the board using the pointer of the board array.

FileIO.c

```
int createLogFile()
```
creates text file for log of moves

```
void LogFile(Board *initial_board, Board *final_board, char
message[SLEN])
```
Writes the move that happened during that turn into log.txt.

Piece.c

```
Piece *createPiece(char name[10],enum Color color, Moves move, int
points)
```
Creates a piece structure which contains the name of the piece, contains the color and the amount of points the piece is worth (for AI purposes). The Moves is a pointer to the correct possible moves function of that piece. (ex a pawn will use PawnPossibleMoves function)

```
void Deletepiece(Piece *piece)
```
Deletes piece structure from the heap

```
bool canMove(Board *init_position, Board *fin_position,enum Color
color)
```
Checks if the piece at the initial board position can move to the final board position.

```
bool DoubleCheck(Board *initial_board, Board *final_board, Board
array[8][8], enum Color color);
```
Checks if it is a possible moves

```
void Promotion(Board bigBoard[8][8], enum Color userColor, int
opponent, int target_x, int target_y);
```
Checks if a pawn can promote

PieceMoves.c

```
void BishopPossibleMoves(Board *initial, Board array[8][8])
```
Fills in possiblemoves array contained in piece struct with pointers of board struct that a bishop at the parameter can move to.

```
void RookPossibleMoves(Board *initial, Board array[8][8])
```
Fills in possiblemoves array contained in piece struct with pointers of board struct that a Rook at the parameter can move to.

```
void KnightPossibleMoves(Board *initial, Board array[8][8])
```
Fills in possiblemoves array contained in piece struct with pointers of board struct that a Knight at the parameter can move to.

```
void QueenPossibleMoves(Board *initial, Board array[8][8])
```
Fills in possiblemoves array contained in piece struct with pointers of board struct that a Queen at the parameter can move to.

```
void KingPossibleMoves(Board *initial, Board array[8][8])
```
Fills in possiblemoves array contained in piece struct with pointers of board struct that a King at the parameter can move to.

```
void PawnPossibleMoves(Board *initial, Board array[8][8])
```
Fills in possiblemoves array contained in piece struct with pointers of board struct that a Pawn at the parameter can move to.

```
int PawnForwardMoves(Board *initial, Board array[8][8], int counter,
int x, int y)
```
Checks if pawn can move forward

```
int PawnDiagonalMoves(Board *initial, Board array[8][8], int counter,
int x, int y)
```
Checks if pawn can capture a piece

```
void CallAllPoss(Board BigBoard[8][8])
```
Updates PossibleMoves for all pieces

`void PossCastle(Board array[8][8])`
Checks if the king can castle

`void PossEnPass(Board array[8][8])`
Checks if pawns can en passant

3.3 Chess Program Functions and Features

3.3.1 Starting the game

The game will ask players if they want to play against the computer or against another player:

*Please note that the computer portion for choosing moves has not been implemented yet, so the player should choose to play against another player:

```
Would you like to play against another (player or computer)?:
```

The player should enter “player”.

```
Would you like to play against another (player or computer)?: player
```

The game will ask Player 1 to choose a side.

```
Please enter which side Player 1 would like to be on (white or black)?:
```

Player 1 will respond by typing “white” or “black”

```
Please enter which side Player 1 would like to be on (white or black)?: white
```

The game will prompt the players with a message reminding them of whose turn it is.

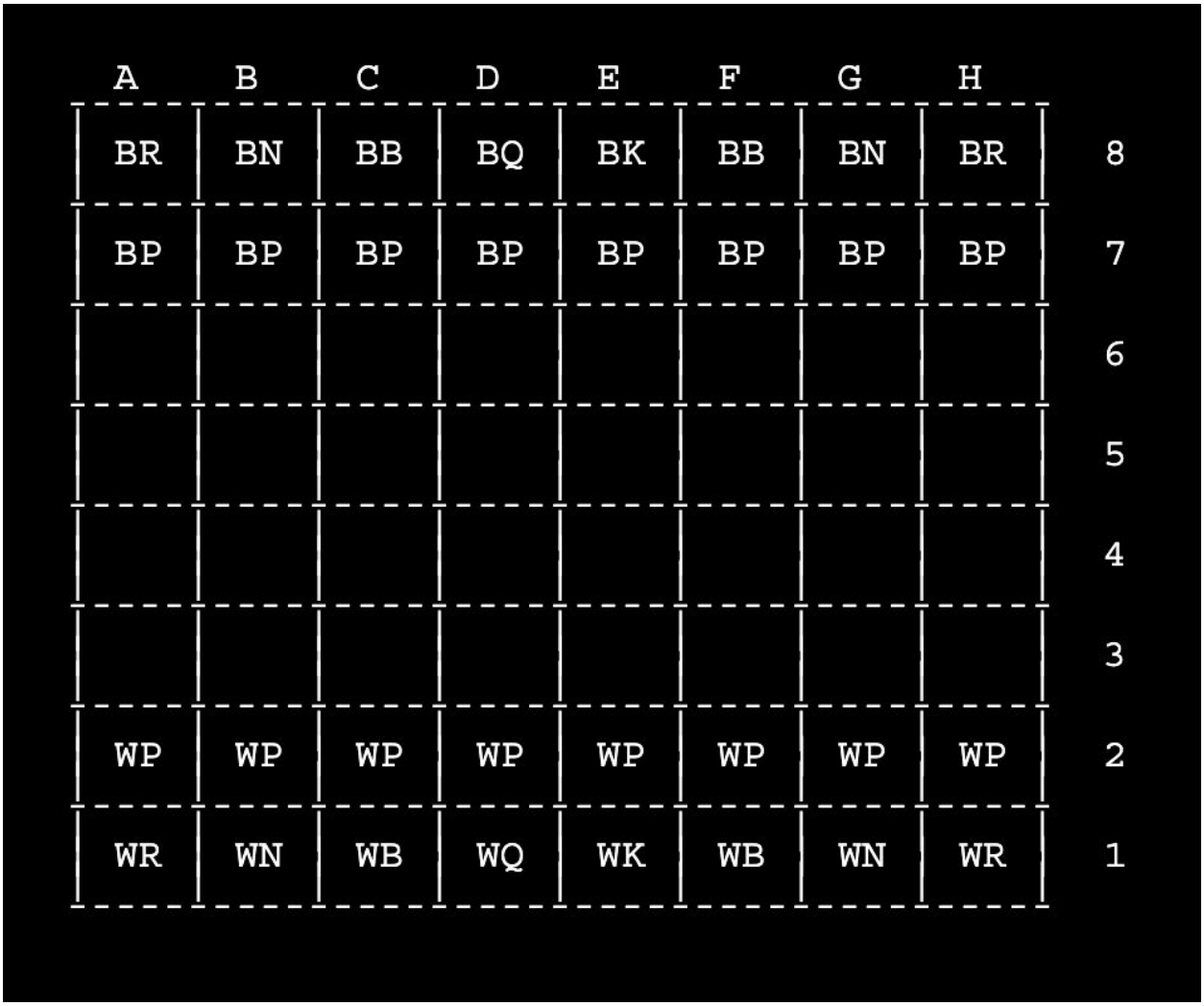
```
Player 1's turn to move!
```

```
Player 2's turn to move!
```

3.3.2 Chess board

An 8x8 chess board will be displayed with A-H labeled horizontally and 1-8 labeled vertically.

The player's chess will be in the bottom



3.3.3 Moves

The player whose turn it is will be prompted to make a move. First, they will be prompted to enter in the coordinates (capital letter followed by a number, e.g. A2, D5, G6) of the piece they want to move.

```
Enter the coordinates of the piece you would like to move (Use Capital Letters):
```

The player should respond by typing in correct coordinates:

```
Enter the coordinates of the piece you would like to move (Use Capital Letters): A2
```

Then, they will be prompted for the coordinates where they want to move the piece.

```
Enter the coordinates of where you would like to move the piece(Use Capital Letters):
```

The player should once again respond by entering in valid coordinates:

```
Enter the coordinates of where you would like to move the piece(Use Capital Letters): A3
```

3.3.4 Check and Checkmate

A message “Check!” or “Checkmate” will appear when check or checkmate is Performed.

```
Check!
```

```
Checkmate!
```

3.3.5 Human-readable log file

The log file contains 3 parts for each game played:

1. **Beginning:** this part will record the game started and which side the players are in.

```
Game started! Player(Black), PC(White)
```

2. **Moves:** this part will record all the moves made by the players.

   Following the syntax :

(player): (piece) “from” (location1) “to” (location2) (additional_information)

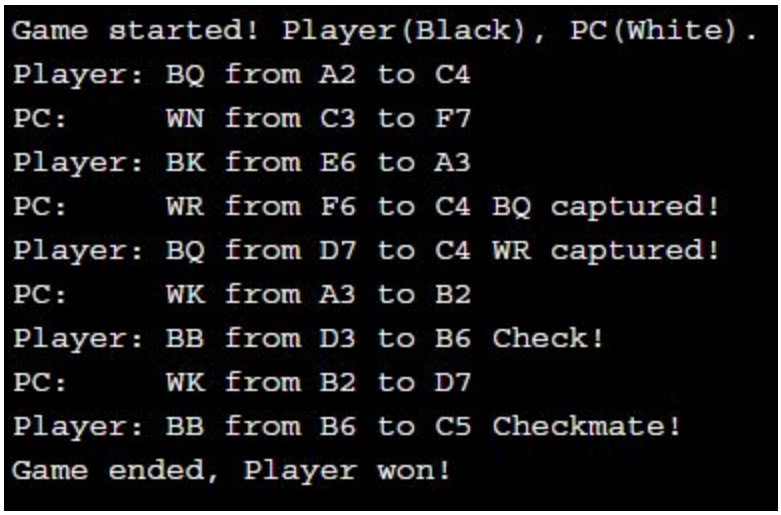
   Example of (additional_information):

   - (piece) captured!
   - Check!
   - Checkmate!

```
PC:     WR from F6 to C4 BQ captured!
Player: BQ from D7 to C4 WR captured!
PC:     WK from A3 to B2
Player: BB from D3 to B6 Check!
PC:     WK from B2 to D7
Player: BB from B6 to C5 Checkmate!
```

3. Ending: this part will record the game ended and which player won.

```
Game ended, Player won!
```



(log for a complete game)

4. Development Plan and Timeline

4.1 Partitioning of Tasks

During team meetings, we discussed how to best implement the program. After brainstorming on our own for a few days, we reconvened and presented our ideas on what data structures and functions would be needed, and then decided how to split the program up into modules.

We decided to first focus on completing the main code of the project first, and then focus on the AI needed for the computer as well as any extra features that we would want to implement.

After deciding on which functions would be needed in the project, we split the code into modules, so that our team could work on the project simultaneously. We also decided that we would use an array of structures to hold the board information and another array of structures to hold piece information.

Here is how we decided to split up and implement the chess program:

Main.c - calls on functions in other modules to actually implement the game of chess

Constants.h - all constants defined here, such as board width and height, etc.

FileIO.c - responsible for the creation of the log file and writing all moves of both players and the outcome of the game to the text file

FileIO.h - header file for FileIO.h

Piece.c - deletes pieces, creates pieces, finds possible moves for each piece type (pawn, bishop, etc.), and checks if a move made by user is valid

Piece.h - header file for Piece.c, structure for piece are defined here

Board.c - creates the board, moves pieces, checks for check and checkmate, etc.

Board.h - header file for Board.c, declares the board structure here

Display.c - prints the board and displays any error messages

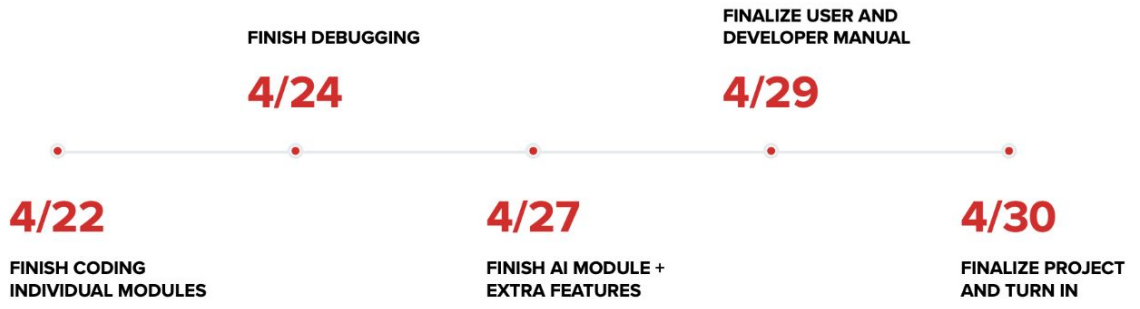
Display.h - header file for Display.c

AI.c - decides the best possible move to make to maximize the computer's chance of winning the game

AI.h - header file for AI.h, declares any functions or structures needed in AI.c

Makefile - compiles all modules, generates object files and an executable file that can be run to play chess

Here is our timeline for the project:



4.2 Team Members and Responsibilities

We split up the modules between all the members of the group as follows:

Ayesha - Piece.c, Piece.h, AI.c, AI.h

Aadi - PieceMoves.c, PieceMoves.h

Hafsah - Chess.c, Constants.h

Jeremy - FileIO.c, FileIO.h, Makefile

Li - display.c, display.h, AI.c, AI.h

Robert - Board.c, Board.h, AI.c, AI.h

Copyright

Printed by Engineers in Pajamas, in the United States of America

First Printing, 2020

Hafsah Aymen Arain

Jeremy Raphael Chang

Hui Quan Li

Aadi Padmawar

Ayesha Parvez

Yu Chen Yan

UC Irvine

Irvine, CA, 92697

References

Fig. 1. Balog, János. *Game of Chess*. 26 Feb. 2012.

Index

F

G

I

L

M

P

S

T

U