Chess-pionage v2024.04.29

Team 7: PretendGINEERS

Mharlo Borromeo

Tuaha Khan

Jack Lu

Calvin Nguyen

Mervin Nguyen

Peter Nguyen

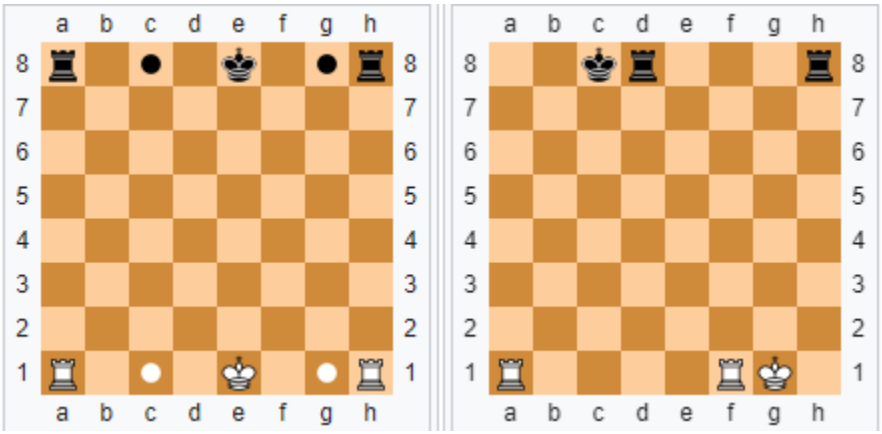
University of California, Irvine : EECS 22L S24

Table of Contents

Glossary

Castling - The castling maneuver can only be executed by a king and a rook if the king and the corresponding rook have not been moved from their starting positions before executing the move. The king will move two spaces towards the rook, either on the kingside or queenside of the chessboard, while the corresponding rook will move besides the king towards the center of the board (“hop over” the king). In order to be eligible to perform this maneuver, there must be no other pieces between the two pieces. Additionally, the king cannot castle out of check, into check, or through check.

En Passant - The en passant maneuver can only be performed by a pawn if the opposing pawn has moved two squares on the previous turn to be adjacent to the left or right of an opposing pawn, where the pawn has the right to perform “en passant” only in the immediate next turn, traveling diagonally to the space behind the opposing pawn and capturing it. If the move is not executed immediately, en passant is not eligible to be performed anymore.



Pieces in Chess

- Pawn (8) chess piece that moves one or two tiles that can capture a piece diagonally one tile left or right side.
- Rook (2) chess pieces that move forward or backward both horizontally and vertically.
- Bipshop (2) chess pieces that can move diagonally.
- Knight (2) chess pieces that can move in a L shaped pattern (3 tiles in horizontal or vertical direction, and 1 tile left or right of the main direction) either forward, backward, side to side and left or right .
- Queen (1) chess pieces that move horizontally, vertically and diagonally, like how the rook and bishop can.
- King (1) chess pieces that can move 1 tile in any direction, if no available space to move, result in a checkmate and the player loses.

C Technical Terminology

API - ‘Application programming interface’, refers to the way components/modules of a computer program are connected together.
Array - A collection of similar data type items stored contiguously in memory locations. Arrays can have different dimensions to them (such as 2D arrays, which will be used in this program).
Compile - The transformation of source code into machine code that can be executed by a computer, which is performed by a special program called a compiler.
Enum - A special kind of data type defined by the user where constant integers (starting from 0) are assigned to names.
GUI - ‘Graphical user interface’, refers to how the program will appear to the user and the environment that the user will interact with and within.
Graphics Library - A set or collection of functions and tools that can be used to perform tasks related to graphics and visual user interfaces.
Pointer - A variable that holds the memory address of a different variable of similar data types.
Struct - A composite data type that defines a grouped set of variables under one name in a single, continuous block of memory.

1. Software Architecture Overview

1.1 Main Data Types and Structures

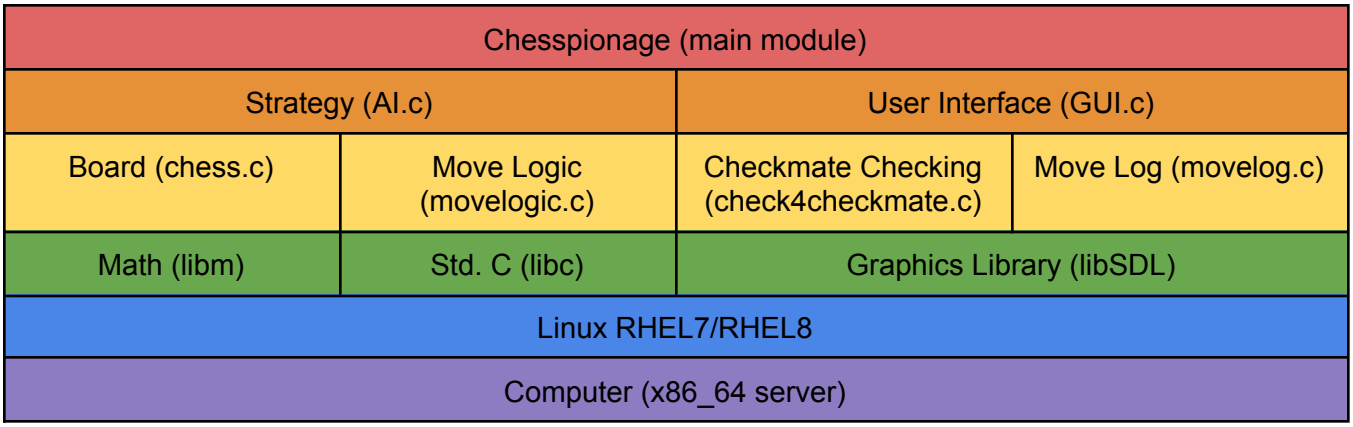
Data Types

- **PieceType**
- **PieceColor**

Data Structures:

- **Board**
- **Piece**
- **Move**

1.2 Major Software Components



1.3 Module Interfaces

- Board initialized for view, movements and last movement for both sides (chess.c).
- Legal moves, check if the move made by both players or AI is valid and not out of bounds (movelogic.c).
- AI should have evaluation(game status), move generation(check what move is currently legal to make), search algorithm(what is the best move possible) (AI.c).
- A module to check checkmate for both players (check4checkmate.c).
- Move log to display all moves made in a game (movelog.c).

1.4 Overall Program Control Flow

1. Program is started by the user in the main menu after launching the executable.
2. User chooses color (assume user = white, AI = black for following program control flow).
3. White player then makes a move.
4. The move is analyzed and the board is updated accordingly.
    a. If the move is invalid (the piece cannot make that kind of move, move is out of bounds, move is on top of another piece of the same color, etc), white player is prompted to make another move.
    b. If the move results in a checkmate, then the white player will win.
5. The black AI player analyzes the white player's move, and then performs calculations based on the weighting of chess pieces in order to determine the best move possible. The board is updated accordingly.
    a. The AI will only consider moves that are valid.
    b. If the move results in a checkmate, then the black player will win.
6. If no checkmate is reached after both players have had a turn, loop back to step 3.
7. Users can exit game at any time via exit button.
8. Exiting will lead back to main menu, where player can start a new game.

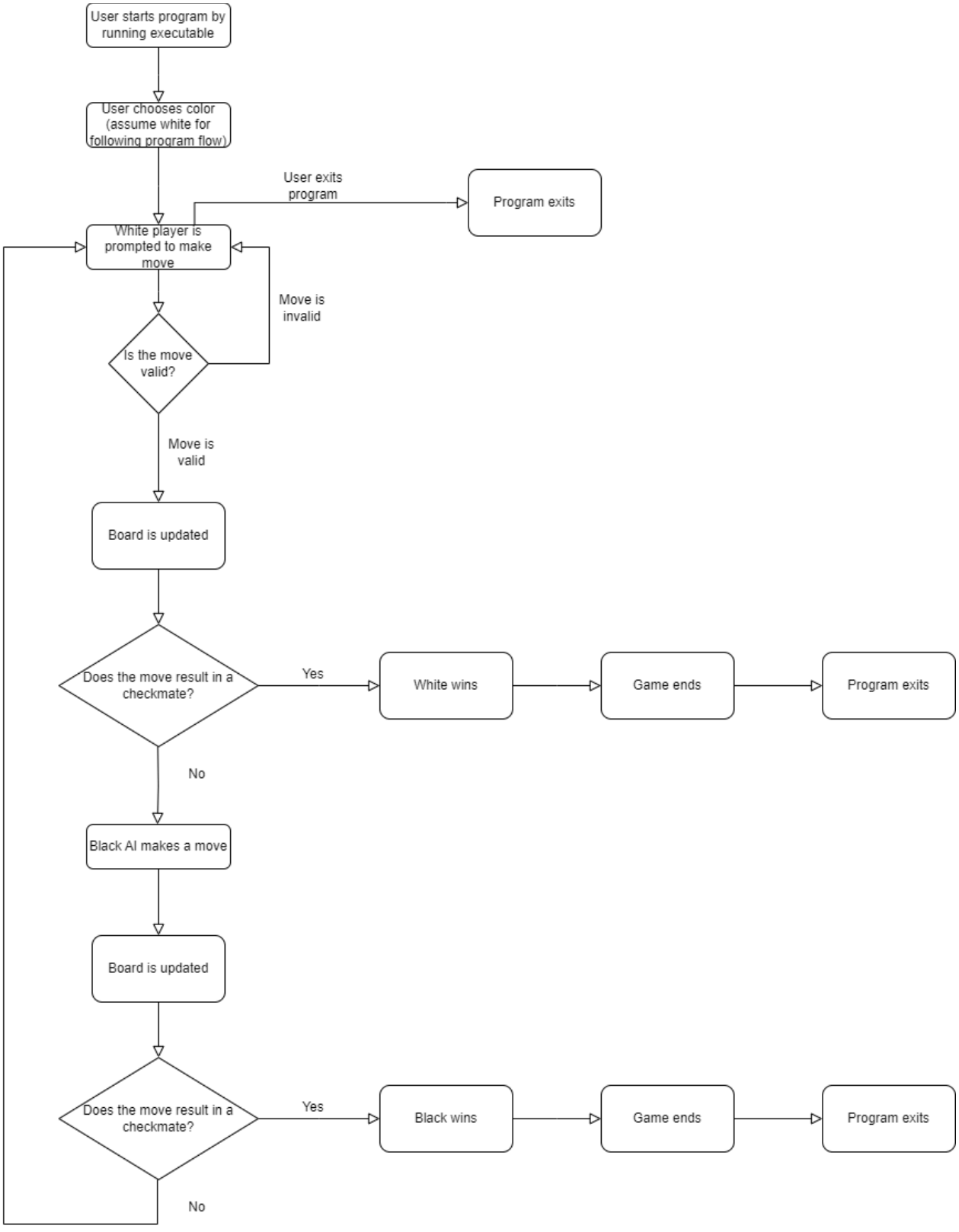


Figure 1: Diagram of program flow.

2. Installation

2.1 System Requirements and Compatibility

Hardware	Minimum Specification
CPU	Any x86_64 processor with clock speed of at least 1GHz
RAM	At least 1 GB
Disk Space	At least 500 MB
Operating System	Linux (RHEL7/RHEL8)

2.2 Setup and Configuration

- The installation of the program requires some knowledge of navigating Linux via terminal interface.
- Prerequisite of having SDL 1.2 library installed. (https://www.libsdl.org/release/SDL-1.2.15/docs/html/index.html, https://github.com/libsdl-org/SDL-1.2 for more information and installation instructions, respectively.)

2.3 Building, Compilation, Installation

- Installation (basic user, no building or compiling necessary)

- Download Chess.tar.gz compressed folder.
- Navigate to the directory where the Chess.tar.gz folder is located after downloading(for example, 'cd Downloads'), and then run command (in terminal) ‘gtar xvzf Chess.tar.gz’ in order to unpack/unzip the file.
- Navigate within the newly unzipped folder in the terminal ('cd chess'), and then navigate to the ‘bin’ folder in order to run the program executable (‘./chesspionage’).

- Advanced Installation (advanced user, requires different folders in order to build and compile properly).

- Download Chess_src.tar.gz compressed folder.
- Navigate to the directory where the Chess_src.tar.gz folder is located after downloading(for example, 'cd Downloads'), and then run command (in terminal) ‘gtar xvzf Chess_src.tar.gz’.
- Navigate within the newly unzipped folder in the terminal ('cd chess'), and then run the command ‘make all’ in order to build and compile the Chess executable within the ‘bin’ folder.

- Optionally, utilize commands ‘make clean’ in order to remove all object files and executable files created by the ‘make/make all’ command.
- Optionally, command ‘make test’ can also be used to create executables ‘test_rulecheck’ and ‘test_boarddisplay’ in order to check all programmed rules of the chess game and the displaying/updating of the chessboard within the terminal, respectively (navigate to 'bin' folder, use './test_rulecheck' and/or '/test_boarddisplay' to run executables).
- Optionally, use command ‘make gtar’ in order to create a tar of the source code package (compress the source code package) that will execute the macro ‘gtar cvzf Chess_src.tar.gz’ along with all bundled files.
- Navigate within the ‘bin’ folder and then run the executable by using the command ‘./chess’.

- Uninstallation (for both basic and advanced users)

- Navigate to the folder where Chess.tar.gz was uncompressed within terminal ('cd Downloads', then 'cd chess'), taking extra care that there are no other contents in the folder besides the files that were unzipped from the Chess.tar.gz file, then use 'rm -f *' to remove all files within the downloaded and unzipped folder without needing to authorize the deletion of each file at a time.

- See installation instructions for more information.

3. Documentation of packages, modules, interfaces

3.1. Detailed description of data structures

`pieceType(board.h)`

pieceType is an enumerated structure under struct Piece that is used to denote the type of piece, as well as an option for “EMPTY” to denote an empty square.

```
typedef enum {
    PAWN,
    ROOK,
    KNIGHT,
    BISHOP,
    QUEEN,
    KING,
    EMPTY
} pieceType;
```

`color(board.h)`

color is an enumerated structure under struct Piece that is used to denote the color of the piece (BLACK or WHITE), and also contains an option for “EMPTY” in case there is no piece on that square, therefore there is no color.

```
typedef enum{
    BLACK,
    WHITE,
    EMPTY
} pieceColor;
```

`stateOfGame(board.h)`

stateOfGame is an enumerated structure under struct Piece that is used to denote the state that the game is currently in. It contains an option for WHITE or BLACK winning, as well as a DRAW state and a CONTINUE state that denotes the game is still being played.

```
typedef enum{
    WIN_WHITE,
    WIN_BLACK,
```

```
    DRAW,
    CONTINUE
}stateOfGame;
```

promotionType (board.h)

Promotion type is a struct to represent the piece types available for a pawn to be promoted to whenever a pawn reaches the opposite side of the board. It contains options for queen, rook, bishop, and knight promotion, as well as the option for no promotion if the player so chooses to not promote their pawn.

```
typedef enum
{
  QUEEN_PROMOTION,
  ROOK_PROMOTION,
  BISHOP_PROMOTION,
  KNIGHT_PROMOTION,
  NO_PROMOTION
} promotionType;
```

Piece(board.h)

Piece is a struct used to represent a piece on the board (2D array) where each element in the array is denoted by a piece type. The struct contains the color, type of piece, as well as row and column integers to represent the coordinate of the piece on the board.

```
struct Piece{
    pieceType piece;
    pieceColor color;
    int row;
    int col;
}
```

Board(board.h)

Board is a struct with an 8x8 2D array of pointers to Piece, to represent the 8x8 chess board as well as fill its squares with pieces.

```
struct Board{
    Piece *board[8][8];
}
```

Move(game.h)

Struct to move pieces on the board, containing a variable int for the first pair of coordinates and the coordinates for the destination the piece will move to.

```
struct Move{
    int x0;
    int y0;
    int x1;
    int y1;
};
```

gameState(game.h)

Struct to denote the state of the game and check whether white has won, black has won, stalemate has occurred, or game is still in progress.

```
struct Game
{
  stateOfGame gameState;
};
```

3.2: Detailed description of functions and parameters.

- **setPiece(struct Board *board, int file, int rank, pieceType piece, pieceColor color);**
    - The purpose of the setPiece function is to allow each piece to be initialized and set on the board struct.
    - Parameters: The parameters of setPiece are the board, file, rank, piece type, and piece color.
    - This function is used in tandem with initializeBoard to set the pieces where they need to be

- **initializeBoard(struct Board *board);**
    - The primary purpose of the initializeBoard function is to fill the specified board up with the necessary pieces for both sides to hold a chess game. This includes all of the pawns, bishops, knights, rooks, queens and kings.
    - This function utilizes the setPiece function to set each type of piece and color in the 2d array.

- Parameters: InitializeBoard will use a pointer to an instance of a Board, so that it knows which board to fill up.

- **`printBoard(struct Board *board);`**
  - The primary purpose of the printBoard function is to present the current configuration of the chessboard in a human-readable format to the user.
  - Parameters: The parameter Board is a pointer to the actual chessboard being used within the game, stored as a 2-dimensional array. The function utilizes this input to generate the visual representation of the board.

- **`isValidMove(struct Board *board, Move *move, pieceColor color);`**
  - This function checks if the current move executed by the player is valid or not.
  - Contains two parameters: the move input by the player, and the board in which the move occurred.
  - Returns 1 if move is valid and returns 0 if move is invalid

- **`pieceMove(struct Board *board, Move *move, pieceColor color);`**
  - The primary purpose of the movePiece function is to move a specific piece within the chessboard. Before this function is executed, the function isValidMove will be called.
  - Although the function is described here as 'pieceMove', each individual piece will have its own move function (eg. pawnMove, rookMove, etc.).
  - Parameters: The parameters of movePiece will be the specific moves that the piece can make, and the Board that it is on.

- **`capture(struct Board *board, struct Move *move);`**
  - The primary purpose of this function is to capture a piece in the game. If the player's move lands on an enemy piece, that piece will be removed from the board and kept aside on display.
  - Parameters: The parameters of capture is the struct Move, which contains the coordinates of the piece that the player moved, as well as the board that the chess piece is on.

- **`isCheck(struct Board *board, pieceColor color);`**
    - The primary purpose of the isCheck function is to analyze the board and check to see if a check has occurred, that is the King is able to be attacked by a piece.
    - Parameters: The parameters of this function are a pointer to the Board struct to analyze whether a check has occurred and the color of the king to be analyzed if it is in check.
    - The function will return 1 if check has occurred, or 0 if no check has occurred
    - If a check has occurred, no other move other than moving the king out of check is valid.

- **`isCheckMate(struct Board *board, );`**
    - The primary purpose of this function is to check the current state of the board to see whether a checkmate has occurred or not
    - Parameters: The function contains a pointer to the Board struct to analyze whether a checkmate has occurred
    - Utilizes isCheck function to see if check has occurred, and then checks if there are any valid moves for the king. If not, the function will return 1. Otherwise, the function will return 0.

- **`isCastle(Move *move, struct Board *board, pieceColor color);`**
    - The purpose of the isCastling function is to implement the special chess rule of Castling within the game. “Castling” is when a king and rook piece have not moved, they are in the same rank, there is no other piece between them, the king is not in check, and that the king does not pass through or finish on a square that is attacked. The function will check all of these conditions and if there is a valid king and rook piece.
    - Parameters: The parameters of isCastling will be the two pieces, namely a king and rook piece, as well as the board that these pieces are on.

- **`isPromotion(Move *move, struct Board *board);`**
    - The purpose of this function is to check if a promotion has occurred on the board.
    - Promotion occurs when a pawn moves all the way to the other end of the board. This function checks whether all requirements for promotion have been satisfied.
    - Parameters: The parameters are the move the player executes, as well as the struct Board

- **`promotePawn(struct Board *board, int promoteChoice);`**

- The purpose of this function is to promote a pawn to a user designated piece type after reaching the opponent's side of the board.
- This function utilizes the isPromotion function in order to provide for error checking.
- Parameters: the parameters are the struct Board that the game is being played on, as well as the player choice of what piece type they would like to promote the pawn to (eg 1 for queen, 2 for rook, etc).

- **`movePiece(struct Move *move, struct Board *board);`**
  - The purpose of this function is to move a piece on the board to a new position and fill the old position with an EMPTY piece.
  - The function handles errors due to using the isValidMove function beforehand, thus it does not inherently have any error checking.
  - Parameters: The parameters are the move that the player desires to be executed, as well as the struct Board that the game is taking place on.

- **`generateLegalMoves(struct Board *board, pieceColor color, Moves moves[], int *moveCount);`**
  - The purpose of this function is to generate a 1-dimensional array of all possible legal moves for a certain color/side.
  - The function has no error checking and is a void function, but will update the moves[] array with entries of possible moves that can be made.
  - Parameters: The parameters of the function are the struct Board, the color of the side to be analyzed, the moves[] array that all the possible moves will be stored into as discrete entries, and the pointer to moveCount, which will be the amount of entries/possible moves inside the array.

- **`aiTurn(struct Board *board, pieceColor color);`**
  - The purpose of the function is to function as the AI or CPU that the player will play against in the player versus computer gamemode.
  - The function is a void function, but will cause the AI to make a move from its set of moves generated by generateLegalMoves().
  - Parameters: The parameters of the function are the struct board, as well as the color of the side that the AI is playing as in that instance of a chess game.

3.3: Detailed description of input and output formats

User Input of a Move:
The user will be prompted with this message at the start of the game:

```
Enter the coordinates of the piece you would like to move,
followed by the coordinates of the square you would like
to move it to (make sure to use a space):
a2 a4
```

The user will then enter the rank and file (in that order) of the piece they would like to move, and the rank and file of the square they would like to move the piece to, with a space in between. The isValid function will be called to validate the move, and if a value of 1 is passed, the piece will move.

Syntax of a move recorded in the log file:
The log file of each move will utilize the same notation for moves as the user sees in the terminal, displaying the starting coordinates, the ending coordinates, and the color and piece type of the piece being moved.

```
2 2 2 4 White Pawn
2 7 2 5 Black Pawn
2 1 3 3 White Knight
1 7 1 5 Black Pawn
```

4. Development plan and timeline

4.1: Partitioning of tasks

Weeks 1 and 2:
- User Manual
- Software Specification
- Begin header files, data type, structure implementation
- Implement board

Weeks 3 and 4:
- Finish header files, data type, structure implementation
- Implement user input collection and processing
- Implementation of move checking functions
- Start AI implementation
- Start GUI implementation

Week 5:
- Finish AI implementation
- Finish checkmate and check implementation

- Add ability for player to choose colors
- Add additional “nice to haves”
    - player can choose AI difficulty between easy and normal
    - GUI
- Test and validate game functionality

4.2: Team member responsibilities

Mharlo Borromeo - main chess function, board/chess function (promotion, castling), instruction manual/software spec, makefile, general debugging
Tuaha Khan - board/chess functions (board functions and move validation), main chess function, instruction manual/software spec, general debugging
Jack Lu - AI, board/chess functions (move validation, castling), instruction manual/software spec, general debugging
Calvin Nguyen - main chess function, instruction manual/software spec, AI, general debugging
Mervin Nguyen - AI, movelog, software validation, instruction manual/software spec
Peter Nguyen - documentation, header files(data types, structure implementation), instruction manual/software spec, makefile, general debugging, AI

Back Matter

Copyright Information

Index

References

https://en.wikipedia.org/wiki/Castling - for image illustrating castling within glossary.
https://en.wikipedia.org/wiki/En_passant - for image illustrating en passant within glossary.
https://www.pinterest.com/pin/296252481738255706/ - for images of chessboard
https://commons.wikimedia.org/wiki/Category:PNG_chess_pieces/Standard_transparent - for images of chess pieces