ChessGPT V1.0

Thomas Yeung	yeungks@uci.edu
Yixuan Jing	yixuanj9@uci.edu
Yian Lin	yianl8@uci.edu
Tangqin Zhu	tangqinz@uci.edu
Rishi Tirumala	rtiruma1@uci.edu

Affiliation: Team ChessGPT

Table of Contents

0. Glossary:

0.1 Chessboard:
A chessboard is a game board used to play chess. It consists of 64 squares, 8 rows by 8 columns, on which the chess pieces are placed.

1. Rank: The eight horizontal rows of the chess board numbered from 1 to 8.

2. File: The eight vertical columns of the chess board lettered from A to H.

0.2 Pieces:
A chess piece is a game piece that is placed on a chessboard to play the game of chess. It can be either white or black, and it can be one of six types: king, queen, rook, bishop, knight, or pawn.

1. Pawn
   Moves forward one square, but captures diagonally one square. On its first move, it has the option to move forward two squares.

2. Knight
   Moves in an "L" shape: two squares in one direction and then one square perpendicular to that, or one square in one direction and then two squares perpendicular. Can jump over other pieces.

3. Bishop
   Moves diagonally any number of squares. Each bishop starts on either a light or dark square and remains on that color for the entire game.

4. Rook
   Moves horizontally or vertically any number of squares. Special move: "castling," in which the rook and the king move simultaneously under certain conditions.

5. Queen
   Combines the power of the rook and bishop, moving any number of squares horizontally, vertically, or diagonally.

6. King
   Moves one square in any direction.

0.3 Special moves:

1. "En Passant": when a pawn is moved two squares from the starting position and ended in the same rank as the enemy pawn, the enemy pawn can capture it by moving to the first square of its path. This can only be done in the very next round.
2. "Castling": the king moves two squares toward the nearest rook and the rook moves to the other side, right next to the king. Doable when none of the two pieces can be moved previously and no pieces are between them.

0.4 Conditions:

1. Check:
   A check occurs when a player's king is under immediate threat of capture by one or more of the opponent's pieces on their next move. The player in check must make a move to remove the threat.

2. Checkmate:
   Checkmate refers to the situation where there is no move for the player possible which would get his king out of check. Then the player loses.

3. Stalemate:
   A situation where the player whose turn it is to move has no legal move and their king is not in check. It results in a draw.

4. Draw:
   A situation in which neither player can force a win. Draws occur under several conditions, including stalemate, insufficient material, threefold repetition, and the fifty-move rule.

5. Promotion:
   Promotion is the replacement of a pawn with a new piece when the pawn is moved to its last rank. The player replaces the pawn immediately with a queen, rook, bishop, or knight of the same color.

0.5 Others:

1. Structure: A structure (or Struct) is a user-defined data type in C that allows you to combine different data types (integers, floats, other structs, etc.) into a single logical unit. It is commonly used to group related variables together.

2. Function: A function is a block of code that performs a specific task. Functions allow for code reusability, better organization, and easier debugging.

3. API: An API (Application Programming Interface) refers to a set of functions and data structures provided by a library or service that developers can use to perform specific tasks, enabling them to write code without having to implement complex details from scratch. These APIs provide a layer of abstraction, allowing developers to interact with hardware, software components, or systems efficiently and effectively.

1. Software Architecture Overview

1.1 Main data types and structures

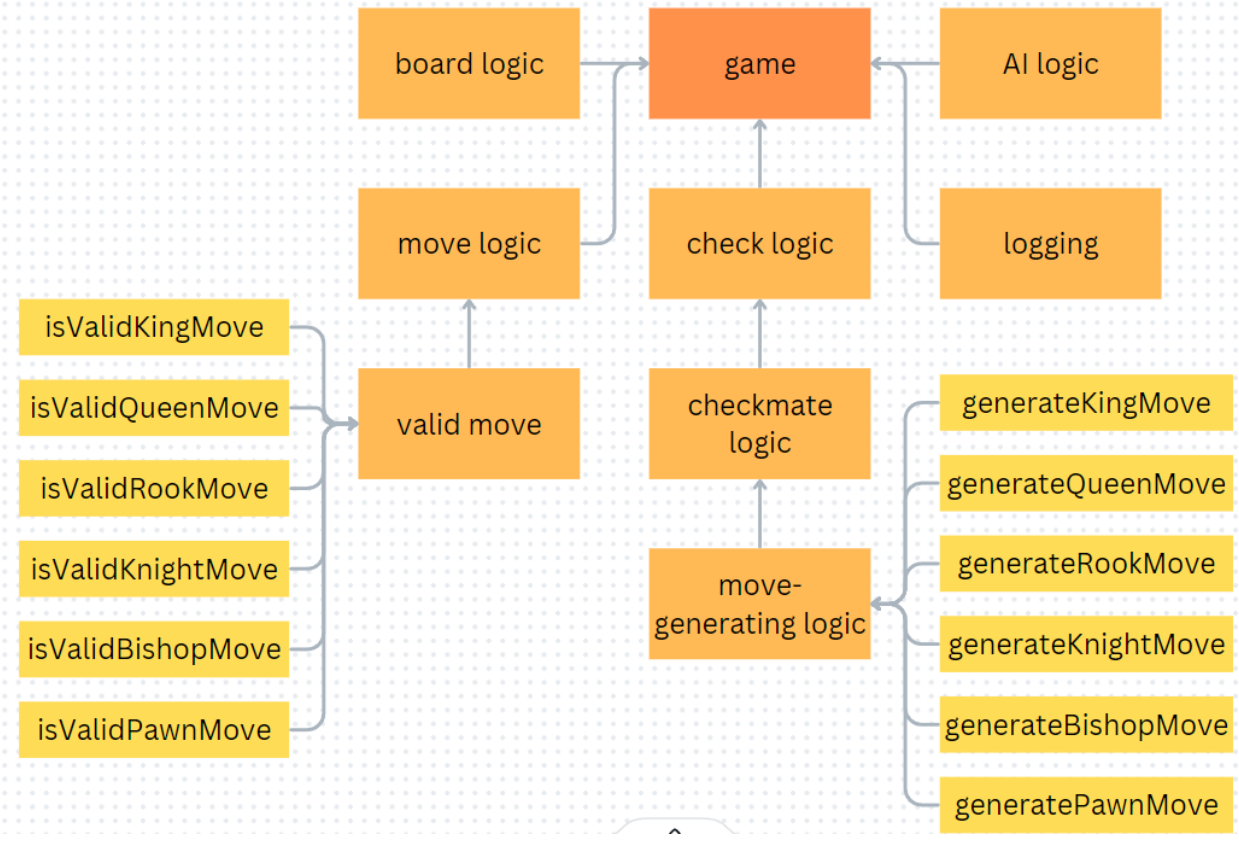
1. Struct: Square
The Square struct models a single square on the chessboard. It's essential for tracking what occupies each square on the board, be it a chess piece or nothing at all.

2. Struct: Piece
The Piece structure encapsulates all the necessary information about a chess piece, including its type (pawn, knight, bishop, rook, queen, king) and its color (black or white). This structure is crucial for managing the game's state and enforcing the rules specific to each piece's movement and capture abilities.

3. Struct: Move
The Move structure encapsulates all the information necessary to describe a move within a chess game. This includes the initial rows and columns, and final rows and columns of the move.

1.2 Major software components

• Diagram of module hierarchy



1.3 Module interfaces
API of major module functions:

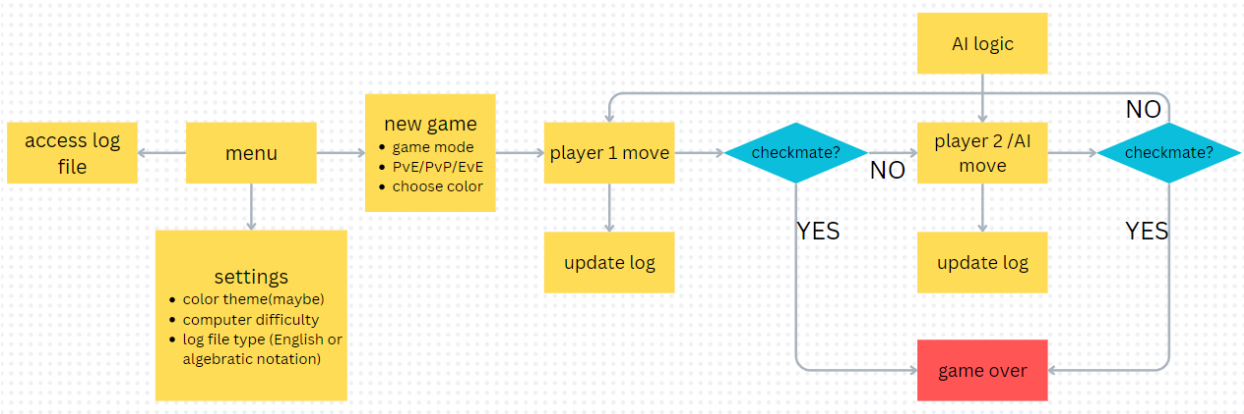
1. Home module
The Home Module acts as the gateway for players into the chess game. It is responsible for displaying the main menu, which includes options like "Play," "Settings," "Exit," and potentially "View Logs" or "Help." This module captures and processes user selections, directing them to the appropriate part of the application. In the "Settings" section, users can customize various aspects of the game, such as choosing a color scheme, setting difficulty levels for AI (if applicable), or configuring other game preferences. This module sets up the initial environment for the game to proceed.

2. Game module
The Game Module is responsible for the game's mechanics. It initializes the board by placing pieces in their starting positions and displays the board to the user. This module reads the user input, which could be in the form of moves specified in standard chess notation (e.g. e2 to e4). It validates these moves for both format (correct notation) and legality. Once a move is validated, it is executed and the board state is updated. This module has an iteration that continuously checks the state of the game, including check, checkmate, stalemate, or draw conditions. The game loop within this module continually updates the game state and user interface until the game concludes.

3. AI module
The AI Module controls the computer opponent, making it a critical component for games played against the computer. While the specific algorithms and implementation details are still under development, the module will likely assess the current board state and generate a suitable move based on the selected difficulty level. Integration with the Game Module ensures that AI moves are validated and executed within the same legal and game rule framework as human moves.

4. Log module
The Log Module serves as the historical record for chess games played within the application. It tracks and stores detailed game logs, including the history of moves, timestamps for game start and end, outcomes (win/loss/draw), and possibly player names or identifiers.

1.4 Overall program control flow



2. Installation

2.1 System Requirement

System Requirements	
RAM	512MB, or more
CPU	Intel Pentium, AMD K5, or better
Memory	2-3GB, or more
OS	Linux
Display	1024*768 screen resolution or higher

2.2 Setup and configuration

1. Verify that the computer either runs on Linux or has a virtual machine that runs on Linux.
2. Download the software files and open the software.

Building, compilation, installation

1. Download Chess_V1.0_src.tar.gz
2. Use command "tar -xvzf Chess_V1.0_src.tar.gz"
3. Use command "cd Chess_V1.0_src.tar.gz"
4. Use command "make"
5. Use command “cd bin”
6. Use command "./chess"

2.3 Uninstalling

Delete all the software files.

3. Documentation of packages, modules, interfaces

3.1 Detailed description of data structures

1. Structure for squares of the chessboard: Struct Squares
   Attributes:
   - color: the color of the square, with data type int, 1 for white and 0 for black
     Might be used for GUI representations or certain chess variants.
   - *content: a pointer that points to the chess piece inside it, or NULL if the square is empty.
   - col: an integer that represents which column the square is in.
   - row: an integer that represents which row the square is in.

   The content of the square will point to the corresponding chess piece if it is occupied, and will point to NULL if the square is empty. The attributes col and row aren't in use yet but might be used to generate logs. The color attribute isn't added yet.

```
struct Square {
    struct Piece* currPiece;
    int col;
    int row;
};
```

2. Structure for chess pieces: Struct Piece
   Attributes:
   - type: a character that represents one of the 6 types of chess pieces.
   - color: an integer that represents one of the two colors.
   - firstMove: an integer that records whether the piece has made its first move yet.
     - To determine special moves such as two squares from start position, En Passant, etc.

```
struct Piece{
    int color;// 0 = white, 1 = black
    int pieceType;// 0 = pawn, 1 = rook, 2 = bishop, 3 = knight, 4 = queen, 5 =
king
    bool firstMove;
};
```

3. Structure for making a move by a player: struct Move
   Attribute:
   - initialRow: An int representing the starting row of the move.
   - Initial_column: An int representing the starting column of the move.

- Final_row: An int representing the ending row of the move.
- Final_column: An int representing the ending column of the move.

A function will translate user input into destination rows and columns, and these information will be stored in this structure for code cleanness.

```
struct Move{
    int initialCol;
    int initialRow;
    int destinationCol;
    int destinationRow;
};
```

3.2 Detailed description of functions and parameters

• Function prototypes and brief explanation*

Board Functions

- void initializeBoard(struct Square gameBoard[8][8])

Initializes the chess board with all pieces in their starting positions. Allocates memory for each piece and sets the respective properties such as type, color, and first move status.

- void deepCopyBoard(struct Square gameBoard[8][8], struct Square gameBoardCopy[8][8])

Creates a deep copy of the board for scenarios like checking future move consequences without altering the main game state.

- void printBoard(struct Square gameBoard[8][8])

Prints the current state of the board to the console. Displays each piece on the board using a character symbol that represents the type and color of the piece.

Move Functions

- bool isValidMove(struct Square gameBoard[8][8], const struct Move *move, int currColor)

Determines if a move is legal by checking the bounds of the board, ensuring the piece exists and matches the current player's color, and then delegating to specific move validation functions based on the piece type.

- bool isValidMoveFormat(const char *input)

Validates the format of a user's input move. Checks that the input string correctly represents a move in chess notation (e.g., e2e4).

- void getPlayerMove(struct Move *move)

Prompts the user to enter a move, checks for the correct format using isValidMoveFormat, and then converts the input string into a move structure.

- void makeMove(struct Square gameBoard[8][8], const struct Move *move, int currColor)

Executes a move if it is valid, updating the board state accordingly by moving the piece and clearing its original location.

Check functions:
- bool isValidBishopMove(struct Square gameBoard[8][8], const struct Move *move);
- bool isValidKnightMove(struct Square gameBoard[8][8], const struct Move *move);
- bool isValidRookMove(struct Square gameBoard[8][8], const struct Move *move);
- bool isValidQueenMove(struct Square gameBoard[8][8], const struct Move *move);
- bool isValidPawnMove(struct Square gameBoard[8][8], const struct Move *move);
- bool isValidCastle(struct Square gameBoard[8][8], const struct Move *move);
- bool isValidKingMove(struct Square gameBoard[8][8], const struct Move *move);
Check the validity of the moves of different pieces

- bool possibleCheck(struct Square gameBoard[8][8], int currColor)
Checks if the current player's king is in check by simulating the opponent moves to the king's position.

- void generateLegalMoves(struct Square gameBoard[8][8], int startRow, int startCol, struct Move moves[], int *numMoves, int currColor)
Generates all legal moves for a piece at a specified position, storing them in an array of moves. It delegates to specific move generation functions based on the piece type.

-void generateRookMoves(struct Square gameBoard[8][8], int startRow, int startCol, struct Move moves[], int *numMoves, int currColor);
-void generateQueenMoves(struct Square gameBoard[8][8], int startRow, int startCol, struct Move moves[], int *numMoves, int currColor);
-void generateKingMoves(struct Square gameBoard[8][8], int startRow, int startCol, struct Move moves[], int *numMoves, int currColor);
-void generatePawnMoves(struct Square gameBoard[8][8], int startRow, int startCol, struct Move moves[], int *numMoves, int currColor);
-void generateKnightMoves(struct Square gameBoard[8][8], int startRow, int startCol, struct Move moves[], int *numMoves, int currColor);
-void generateBishopMoves(struct Square gameBoard[8][8], int startRow, int startCol, struct Move moves[], int *numMoves, int currColor);
Generate the possible moves for different pieces.

-bool isStalemate(struct Square gameBoard[8][8], int currColor);
-bool isEnPassant(struct Square gameBoard[8][8], const struct Move *move, int currColor);
-bool canEscapeCheck(struct Square gameBoard[8][8], int currColor);
-bool outOfCheck(struct Square gameBoard[8][8],struct Move *move,int currColor);
Check for special conditions or special moves.

AI functions:
-int evaluateBoard(struct Square gameBoard[8][8],int currColor);
This function evaluates the current state of a chess board based on various piece positions and strategic tables. It returns an integer score that represents the board's value from the

perspective of the current player. It includes checks for specific situations like checks and stalemates and evaluates material and positional advantages using predefined scoring tables for different pieces.

-int pieceCounter(struct Square gameBoard[8][8],int pieceType,int currColor);
This function counts the number of pieces of a specific type and color on the board. It iterates through the board and increments a counter whenever a piece matching the specified type and color is found.

-int pieceEval(struct Square gameBoard[8][8], int evalBoard[8][8],int pieceType,int currColor);
This function calculates the score for pieces of a specific type and color on the board by summing up values from a given evaluation table. Each piece's position on the board contributes to the score based on the corresponding value in the evaluation table.

-int miniMax(struct Square gameBoard[8][8],int row, int col, int depth,int alpha, int beta,int currPlayer,bool maxPlayer);
The miniMax function is an implementation of the Minimax algorithm, which is used in game theory to determine the optimal move for a player in chess game.

Log functions:
-void addLog(int logType, int pieceType, int currColor, int startRow, int startCol, int destRow, int destCol);

The addLog function records details of chess moves and game events in a structured log format. It creates a natural language description of each move or event based on the type (regular move, capture, castling, etc.), and also formats an algebraic notation string that succinctly represents the move.

-void undoLog();
This function handles the cases where the user wants to undo the move. It deletes the previous move in the array.

Game functions:
-int startGame(int gameMode);
This function initializes and starts a chess game based on the selected game mode.

-bool redoMovePVE(struct Square board[8][8], struct Square backupBoard[8][8], int currentPlayer);
-bool redoMovePVP(struct Square board[8][8], struct Square backupBoard[8][8], int currentPlayer);
These functions allow players to redo their moves in games against the environment or another player, respectively, by restoring the game state from a backup board.

-void printMenu();
This function displays the main menu options to the player.

-void PlayerMode_1();
-int PlayerMode_2();
These functions manage different modes or scenarios in which players can engage with the game, likely differing in their settings or rules.

*Due to the vast amount of functions in the software, only essential functions are explained in the document.

3.3 Detailed description of input and output formats

-Syntax/format of a move input by the user

- Program prompts user for initial position and final position
  Example: `Enter your move (e.g., e2e4):`
  User input: `e2e4`

-Syntax/format of a move recorded in the log file
The log file can be viewed in natural language and algebraic notation.

- Natural Language
  - Piece Movement
    [color] [piece] moved from [initial position] to [final position].
    Example: `White pawn moved from e2 to e4.`
  - Capture
    [color] [piece] captured from [initial position] to [final position].
    Example: `White pawn captured from b4 to c5.`
  - Castling
    [color] performed a [piece] side castling.
    Example: `White performed a queen side castling.`
  - Promotion
    [color] pawn at [final position] promoted to [piece].
    Example: `White pawn at e8 promoted to queen.`
  - Check
    [color] is under check.
    Example: `Black is under check.`
  - Checkmate
    Checkmate for [color]!
    Example: `Checkmate for black!`
  - Stalemate
    Stalemate! Game Draw!
  - Win
    Game over! [color] wins!

Example: `Game over! White wins!`

- Algebraic Notation
  - Piece Movement
    Pawn: [final position]
    Example: `e4`
    Other pieces: [abbreviation of piece][final position]
    Example: `Qd8`
  - Capture
    Pawn: [initial file]x[finial position]
    Example: `exd5`
    Other pieces: [abbreviation of piece]x[final position]
    Example: `Bxe5`
  - Castling
    King-side Castling: `0-0`
    Queen-side Castling: `0-0-0`
  - Check
    Pawn: [final position]+
    Example: `d8+`
    Other pieces: [abbreviation of piece][final position]+
    Example: `Qd8+`
  - Checkmate
    Pawn: [final position]#
    Example: `d8#`
    Other pieces: [abbreviation of piece][final position]#
    Example: `Qd8#`
  - Stalemate
    `1/2 - 1/2`
  - Win
    White win: `1-0`
    Black win: `0-1`

4 Development plan and timeline

4.1 Partitioning of tasks

Date	Task
4/8-4/15	● Complete the software specification

	● Develop an initial prototype -Basic game logic -Develop features including log, special moves, stalemate
4/15-4/22	● Develop alpha version of the software -Implement AI module
4/22-4/29	● Prepare for team presentation ● Develop GUI module ● Final version

4.2 Team member responsibilities

Section	Status	Assigning to
Log file	Done	Yian, Aiden
Undo move	Done	Thomas
Stalemate	Done	Rishi
Special move	Done	Rishi
Game menu (Difficulty, etc.), add to main()	Done	Yian
Documentation	Done	Group
Makefile&test	Done	Alex
Section(Additional)	Status	Assigning to
AI	Done	Rishi
GUI	Done	Alex,Thomas

References

EECS 22L Slides
Credit: Professor Rainer Dömer, University of California, Irvine

Index