ChessCraft v3.0 Software Specification

Produced by ChessCrafters
Affiliated with University of California, Irvine

Authors:
Aileen Kim
Bryan Melendez
Gavin Nguyen
Matthew Quach
David Schoening

Table of Contents

Glossary

- *Application Programming Interface (API)* - A way for two or more software programs to communicate with each other
- *Doubly linked list* - A data structure that holds sequentially linked nodes. Each node has a link to the previous and next node.
- *Enumerate* - A user-defined data type that assigns names to integral constants
- *Game Log* - a text file that logs all moves made during the game
- *Member (Struct Member)* - A variable found inside a structure (See *Structure*)
- *Structure (struct)*- A user-defined data type that is used to group related variables into one place. (See *member*)
- Text-based User Interface - A form of User Interface where the user interacts with the program using typed commands. The output is text printed in the terminal.
- *User Interface* - The point of interaction between the user and the program

1. Software Architecture Overview

1.1 Main data types and structures

Structures:

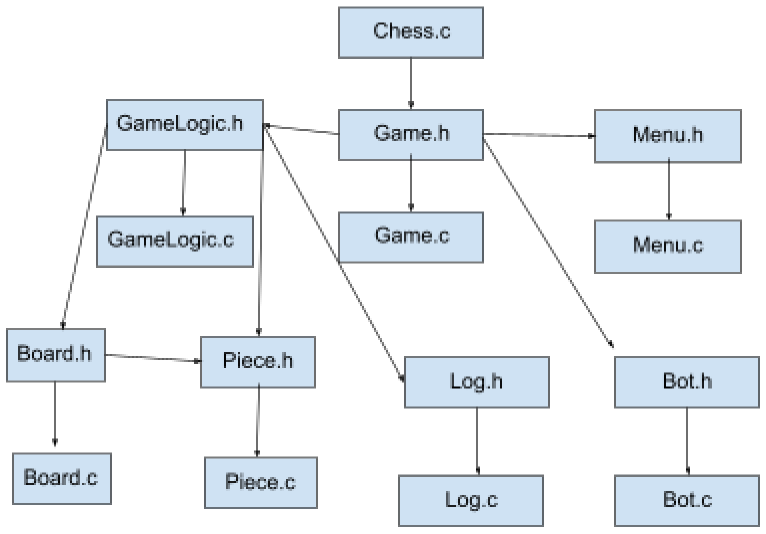
- Bot
    - Difficulty difficulty;
    - Coordinates start_location;
    - Coordinates end_location;
    - PieceColor color;
- Coordinates
    - int column
    - int row
- Game
    - Location* gameboard[8][8];
    - LogList *game_log;
    - CurrentPlayer player;
    - PieceColor color;
    - int game_mode;
    - int check;
    - int mate;
- Location
    - Piece* piece;
    - int valid_move;
    - int check_move;
    - int castle;
    - int en_passant;
- Log
    - LogList *list;
    - char move_entry[20];
    - char timestamp[30]
    - Log *next;
    - Log *prev;
- LogList
    - Log *first
    - Log *last
- Piece
    - PieceType type
    - PieceColor color
    - int first_move

Enumerates

- PieceType
    - PAWN
    - KNIGHT
    - BISHOP

    - ROOK
    - QUEEN
    - KING
- PieceColor
    - WHITE
    - BLACK
- CurrentPlayer
    - PLAYER1
    - PLAYER2
- Difficulty
    - EASY
    - MEDIUM
    - HARD

1.2 Major software components



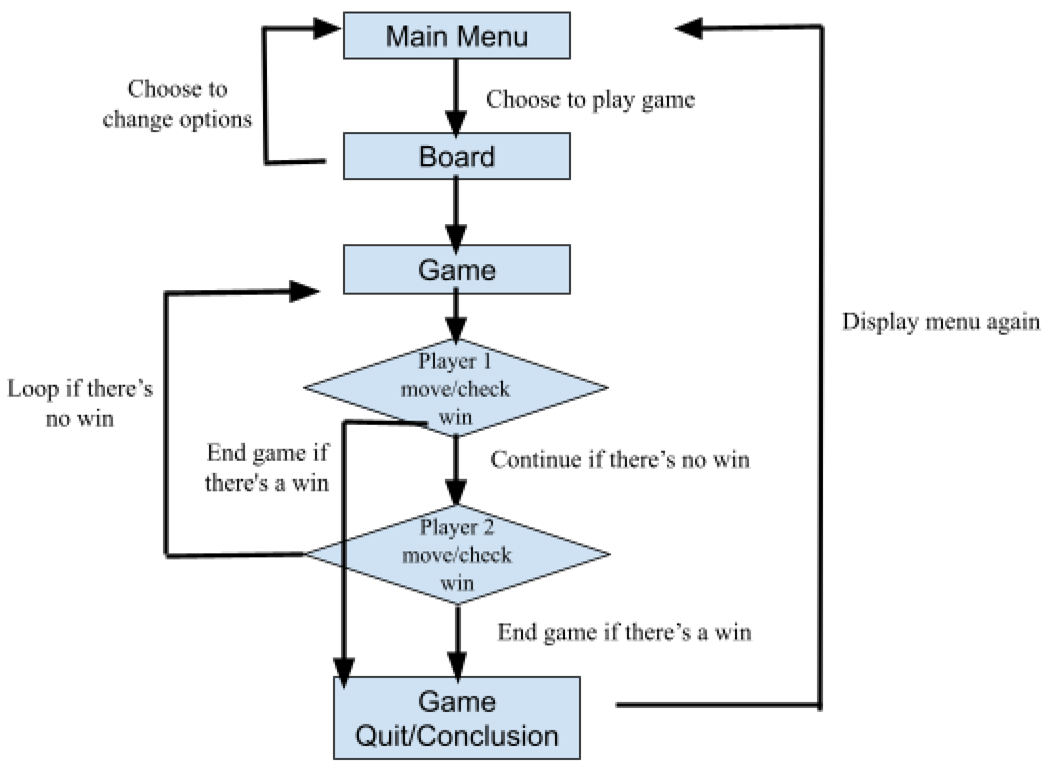
1.3 Module interfaces

- Board Module
    - void InitializeGameboard(Location* gameboard[8][8])
        - Creates a game board, placing all 32 pieces in their initial positions
- Game Logic Module
    - int Check(Game* game, Location* gameboard[BOARD_SIZE][BOARD_SIZE])
        - Determines if the king is in check, given the state of the game
    - int MovePiece(Location *game_board[BOARD_SIZE][BOARD_SIZE], Coordinates start_place, Coordinates end_place, Game* game)
        - Moves a piece, given that the new location is a valid move
- Piece Module
    - int CalculateMoves(Location* gameboard[][8], Coordinates coords)
        - Indicates whether or not each location is a valid move
- Menu Module

    - int Menu1(int play_option)
        - Prints the initial game menu, prompting user to choose a game mode
    - int Menu2(int game_option)
        - Prints the second game menu, prompting the user to select piece color
    - int Menu3(int bot_num)
        - Prints the third game menu, prompting the user to select a difficulty level
    - int Menu4()
        - Prints the fourth game menu, prompting the user to choose play to start the game
- Game Log Module
    - void AddMove(LogList *list, char *move)
        - Adds the most current move to the log of moves
- Bot Module
    - int EasyMove(Bot* bot, Game* game)
        - Makes a random, valid move
    - int MediumMove(Bot* bot, Game* game, int bot_turn)
        - Opens each game using basic book moves
    - int HardMove(Bot* bot, Game* game, int bot_turn)
        - Opens each game using the stonewall chess opening

1.4 Overall program control flow

1. Main Menu
    a. Initial menu with game options is displayed to welcome user
    b. User input is taken for game settings
2. Board
    a. If user chooses to start a game, the initial chess board is displayed using the earlier selected game settings
3. Game (looped until game ends)
    a. User input is taken for the location of the piece they want to move
    b. Legal moves for that location are calculated
    c. Updated board with highlighted legal moves is displayed
    d. User input is taken for the move
    e. Move is added to game log
    f. Update board with the new move is displayed
    g. Check for win
    h. Computer/opponent makes move
    i. Updated board with the new move is displayed
    j. Check for win
4. Game Quit / Game Conclusion
    a. Once game is won or quit, the result is displayed
    b. Main menu with game options is displayed again
    c. User input is taken to continue or quit the program



2. Installation

2.1 System requirements, compatibility

To run ChessCraft v1.0, you must own a computer/laptop with the minimum system requirements:

- Storage space of at least 10MB
- At least 1GB of RAM
- Linux Operating System
- GCC Compiler
- GNU Make
- C11
- Evince Document Viewer (or any PDF viewer)

Libraries:

- Math Library
- Ctype Library
- String Library
- Stdlib Library
- Time Library

2.2 Setup and configuration

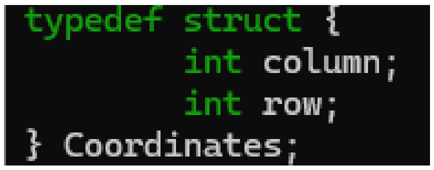
1. Navigate to desired location to store chess source code
   `cd {desired location}`
2. Copy source code file to the new directory
   `cp ~/{old location}/Chess_V1.0_src.tar.gz .`
3. Unpackage program files
   `tar xvzf  Chess_V1.0src.tar.gz`
4. View the software manual
   `evince chess/doc/chess_software.pdf`

2.3 Building, compilation, installation

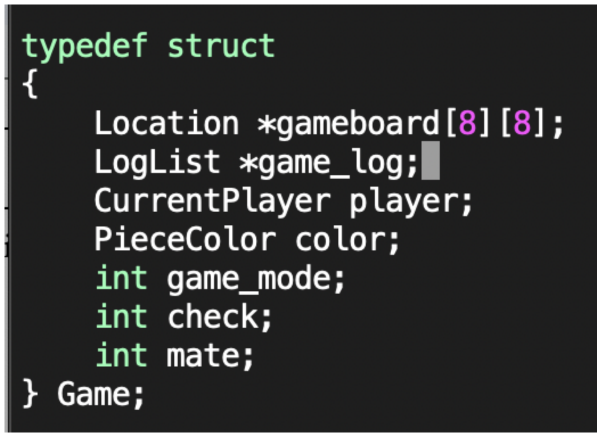
1. Build the complete program
   `make all`
2. Build the debug version
   `make test`
3. Remove executables and object files
   `make clean`
4. All other make rules can be viewed in Makefile
   `vim Makefile`

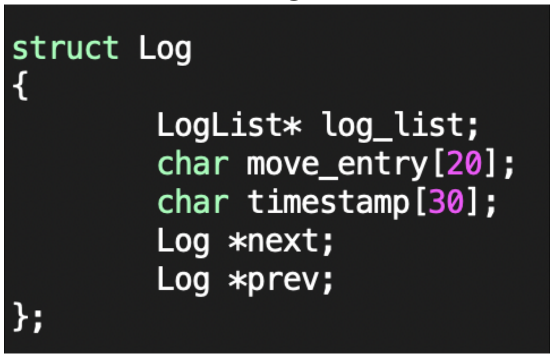
3. Documentation of packages, modules, interfaces

3.1 Detailed description of data structures

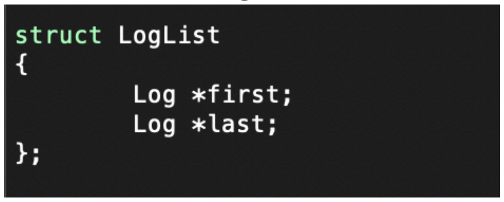
Structure Name	**Description**
Coordinates  `typedef struct { int column; int row; } Coordinates;`	Stores the row and column of a specific location
Member Name	**Description**
int column	Contains the column number from 0 to 7
int row	Contains the row number from 0 to 7

Structure Name	**Description**
Game	Stores important game data

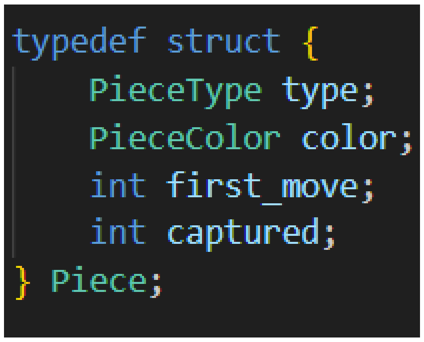
 ```typedef struct { Location *gameboard[8][8]; LogList *game_log; CurrentPlayer player; PieceColor color; int game_mode; int check; int mate; } Game;```	
Member Name	**Description**
Location* gameboard[8][8]	2D array that points to a location struct
LogList *game_log	List that stores game log entries
CurrentPlayer player	Stores data about who is the current player
PieceColor color	Stores data to tell which player has the white or black pieces.
int game_mode	Stores data to tell which game mode the user chose.
int check	Stores data to see whether a player is in check or not
int mate	Stores data to tell whether a player has checkmated another player or not.

Structure Name	**Description**
Log  ```struct Log { LogList* log_list; char move_entry[20]; char timestamp[30]; Log *next; Log *prev; };```	Stores data for a move recorded in the Game Log
Member Name	**Description**

LogList* log_list	Points to the list it belongs to
char move_entry[20]	Character string of the move
char timestamp[30]	Character string of timestamp of move
Log* next	Points to the entry that comes after
Log* prev	Points to the entry that comes before

Structure Name	**Description**
LogList 	List of all the logs (recorded moves) for a game
Member Name	**Description**
Log* first	Points to the first Log in the list
Log* last	Points to the last Log in the list

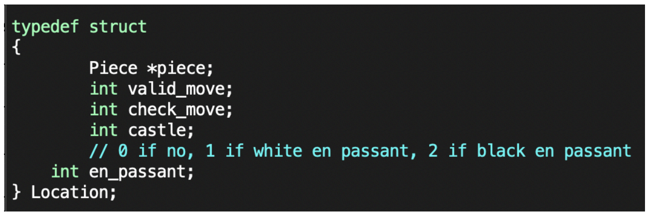
```
struct LogList
{
        Log *first;
        Log *last;
};
```

Structure Name	**Description**
Piece 	Contains information about specific pieces
Member Name	**Description**
PieceType type	Variable of enum type that stores a value corresponding to a type of chess piece
PieceColor color	Variable of enum type that stores a value corresponding to white or black piece color

```
typedef struct {
    PieceType type;
    PieceColor color;
    int first_move;
    int captured;
} Piece;
```

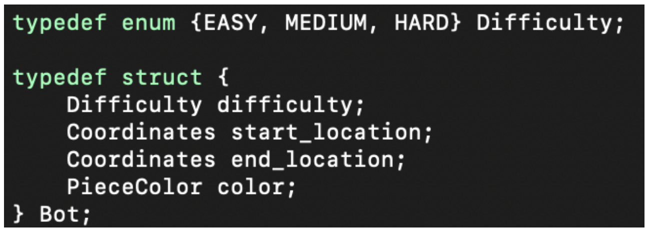
int first_move	Either 0 or 1 depending if a piece has been moved or not
int captured	Set to 1 when piece is captured

Structure Name	**Description**

Location

`typedef struct`
`{`
`        Piece *piece;`
`        int valid_move;`
`        int check_move;`
`        int castle;`
`        // 0 if no, 1 if white en passant, 2 if black en passant`
`    int en_passant;`
`} Location;`
	Stores data pertaining to each specific location on the game board

Member Name	**Description**
Piece* piece	Pointer to a piece object
int valid_move	Is set to 1 when a location is the valid endpoint for the current move
int check_move	Is set to 1 if there is a check
int castle	Is set to 1 if there is castling
int en_passant	Is set to 0 if there is no en passant, set to 1 if white made en passant, and 2 if black made en passant

Structure Name	**Description**

Bot

`typedef enum {EASY, MEDIUM, HARD} Difficulty;`
 	
`typedef struct {`
`    Difficulty difficulty;`
`    Coordinates start_location;`
`    Coordinates end_location;`
`    PieceColor color;`
`} Bot;`
	Contains all the information regarding the AI computer bot

Member Name	**Description**

Difficulty difficulty	Stores the difficulty of the bot based off of the user's selection
Coordinates start_location	Stores the starting coordinates of the piece the bot is intended to move from
Coordinates end_location	Stores the end coordinates of the piece the bot is intended to move to
PieceColor color	Stores data to tell if the bot has the white or black pieces.

3.2 Detailed description of functions and parameters

- Board Functions
  - void InitializeGameboard(Location* gameboard[8][8])
    - Initializes the game board 2D array to store all 32 pieces in their correct place using Piece objects and their respective colors and piece types.
  - char GetPieceColor(Location* gameboard[8][8], int x, int y)
    - Takes in a location within the game board 2D array and returns the initial of the color of that piece as a single character. If NULL, the function returns nothing.
  - char GetPieceType(Location* gameboard[8][8], int x, int y)
    - Takes in a location within the game board 2D array and returns the initial of the type of that piece as a single character. If NULL, the function returns nothing.
  - void PrintBoard(Location* gameboard[8][8])
    - Calls both GetPieceColor(gameboard, x, y) and GetPieceType(gameboard, x, y) and prints out the current state of the game board using text through the terminal.
  - void DeleteBoard(Location* gameboard[8][8])
    - Once called, it deallocates and frees all memory associated with the board and its Location structure contents, specifically calling DeletePiece(gameboard).
  - void DeletePiece(Piece* piece)
    - Takes in an object of the Piece structure, and once asserted that this piece exists, frees the memory connected to the piece.
- Bot Functions
  - int GetBotMove(Bot* bot, Game* game, int bot_turn)
    - Depending on the selected AI difficulty in the main menu, the function will execute a move for the AI using one of the Move functions listed below. If in case the bot cannot make a move, it will result in an error.
  - int EasyMove(Bot* bot, Game* game)
    - Gets a piece from a random position on the current gameboard
    - Calculates the potential moves that the chosen piece has, and executes a random move
  - int MediumMove(Bot* bot, Game* game, int bot_turn)
    - Sets up a chess opening using basic book moves

        - Once the opening has been completely set up, the bot will start using random legal moves in the same manner as the EasyMove function
    - int HardMove(Bot* bot, Game* game, int bot_turn)
        - Sets up a stonewall chess opening
        - After completing the stonewall opening, it will make random legal moves as in EasyMove and MediumMove
    - void GetRandomCoordinates(Coordinates* coords)
        - Creates a Coordinates object, setting the row and column to random coordinates on the 8 by 8 grid using the rand() function modulo 8.
    - void GetRandomPosition(Bot* bot, Coordinates *coords, Location* gameboard[BOARD_SIZE][BOARD_SIZE])
        - Locates a location on the board that contains a piece the same color as the current player.
        - Works as long as the piece can legally move.
    - void GetRandomMove(Bot* bot, Coordinates* coords, Location* gameboard[BOARD_SIZE][BOARD_SIZE])
        - Sets up a random move for the bot to make using GetRandomCoordinates.
- Game Functions
    - void PlayGame(Game, Bot bot)
        - Calls InitializeGameboard(game.gameboard) and PrintBoard(game.gameboard) to initialize a chess match and get user input for the first move.
        - Calls PrintBoard(game.gameboard) in a while loop to handle turns between players afterwards.
        - Users can input piece moves and access an options menu for pausing/unpausing, undoing moves, quitting, and hints.
        - If the user chooses a game option involving AI, PlayGame takes in a Bot object and allows the computer to make moves based on the selected difficulty.
    - int CapturePiece(Game)
        - Function returns an integer value determining if a piece was captured or not
    - void CapitalizeUserInput(char *user_selection, int move_step)
        - Takes in the user's move and capitalizes it
    - void GetUserInput(char *user_selection, int move_step);
        - Prompts and scans the user's input of their desired move, whether it be a piece on the board or one of the implemented option commands.
    - int CheckUserInput(int *user_selection[], Game* game, int move_step)
        - Checks if user input matches any options and handles option functions for canceling moves, quitting, and hints.
    - void ExecuteUserInput(Coordinates user_start_coordinates, Coordinates user_end_coordinates, int input_type, int *quit, int *move_step, int *finish_move, Game* game)
        - Takes in the user's input and executes the move or option that they chose. Using int input_type, the function performs the selected option or move.
    - void Cancel((Location* gameboard[BOARD_SIZE][BOARD_SIZE], int *move_step)

        - Cancels a move after a user has successfully chosen a piece they can legally move. If this is not satisfied, the function informs the user that they are unable to cancel.
    - Coordinates GetCoordinates(Location* location[][8], PieceColor player_color)
        - Gets the coordinates of the user input and returns a Coordinate struct with a row and column integer value.
    - void GetRandomHint(Game* game)
        - When called, the user will receive a legal move they can make based on the current state of the game. It will state both the potential piece to move and the location to move it to in algebraic notation.
- GameLogic Functions
    - int CalculateMoves(Game* game, Location* gameboard[BOARD_SIZE][BOARD_SIZE], Coordinates coords, int for_check)
        - Using the coordinates passed through, it finds the piece located at those coords and calculates the number of possible moves that specific piece can legally make in the current state of the board.
        - Filters out all invalid moves that cause a piece to be in check.
    - int FilterMoves(Game* game, Location* gameboard[BOARD_SIZE][BOARD_SIZE], Coordinates coords)
        - Passing through a specified piece, this function filters out all moves that could put themselves in check.
        - Creates a copy of the state of the gameboard passed through, and checks for right and left side castling and if the king is currently in check.
        - Returns the number of moves filtered out.
    - void CopyGameBoard(Location* old_board[BOARD_SIZE][BOARD_SIZE], Location* new_board[BOARD_SIZE][BOARD_SIZE])
        - Taking in two 2D arrays of the Location structure, it allocates memory the same size of memory and copies the original board into the new, copied board.
    - int Check(Game* game, Location* gameboard[BOARD_SIZE][BOARD_SIZE])
        - Looks throughout the gameboard to find the king of the opposite color from the players current turn
        - Then locates the type of piece that was last played and calculates the move
        - If the recently moved piece can move where the king is on its next turn, the king is in check
        - If the king is in check, sets the game.check = 0
    - int MovePiece(Location *game_board[BOARD_SIZE][BOARD_SIZE], Coordinates start_place, Coordinates end_place, Game* game)
        - Takes the coordinates of a piece chosen by the user (start_place) and moves that piece to the new coordinates (end_place).
        - Accounts for potential errors, involving the end_place not existing, the move being an invalid move, and the piece trying to capture a piece of the same color
        - Checks for multiple scenarios of castling, en passant, if the move is capturing another piece, or if it is just a normal move.
        - If valid, will record the move in the log.

    - int Castle(Location *game_board[BOARD_SIZE][BOARD_SIZE], Piece *king, Coordinates new_king, Game* game)
        - Takes the current users color and attempt at moving the king
        - Checks if the user has decided to try and castle kingside or queenside
        - Then checks if castling is still an option by checking if either piece has moved
        - If castling is possible, the king and rook will be placed in the correct places for castling after
        - If not, the user will be notified this move is invalid and another should be chosen
    - int EnPassant(Location *game_board[BOARD_SIZE][BOARD_SIZE], Piece* pawn, Coordinates end_place)
        - If valid, the function allows enPassant by the user. Depending on the color of the opponent, EnPassant will delete the opposing pawn, returning an error if it does not work.
    - int ClearEnPassant(Game* game)
        - Traverses through the entire game board and sets the en_passant member of each location struct equal to zero.
    - int PawnPromotion(Location *game_board[BOARD_SIZE][BOARD_SIZE], CurrentPlayer player)
        - Allows pawn promotion if a user's move is valid.
    - int CheckMate(Game *game, Location* gameboard[BOARD_SIZE][BOARD_SIZE], int same_side)
        - Calculates the amount of moves that could happen to put the king in check
        - It accounts for every situation that could put the king in check, including if the king can move anywhere, if the user can capture any of the opponent pieces causing check, if one of the pieces can block check, etc.
    - void CapturePiece(Location *game_board[BOARD_SIZE][BOARD_SIZE], Piece* moved_piece, Coordinates end_place)
        - Using the Coordinates at end_place, this deletes the piece of the opposing color being captured, as well as assigns the piece performing the capture to the new location.
    - void ClearBoard(Location *gameboard[BOARD_SIZE][BOARD_SIZE])
        - Traverses through the gameboard and sets both valid_move and check_move of each Location object to 0.
- Game Log Functions
    - LogList *NewLogList(void)
        - Allocates memory for a new list
        - Returns the newly created list
    - void DeleteMove(Log *log)
        - Frees memory allocated for Log (given by the parameter)
    - void DeleteLogList(LogList *list)
        - Loops through the list (given by the parameter) and calls DeleteMove to free each list member
        - Frees memory for the list
    - void AddMove(LogList *list, char *move)

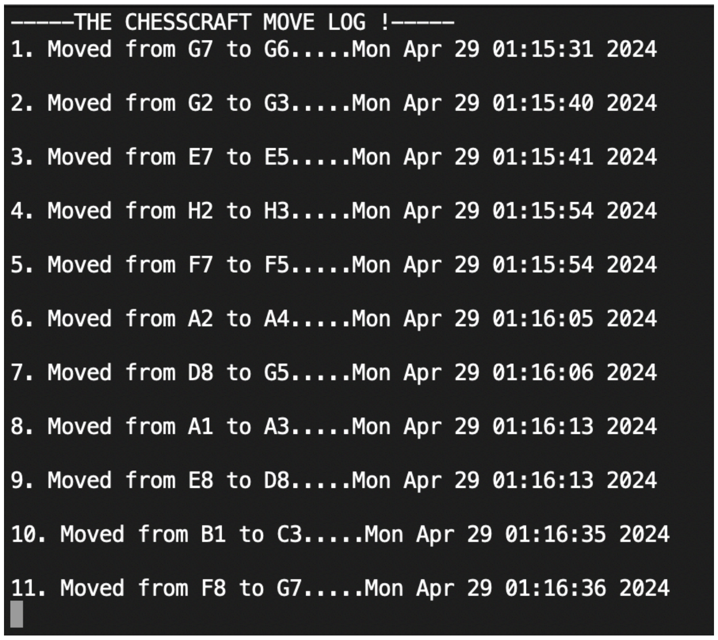
        - Allocates memory for a new Log
        - Assign the string parameter to the Log’s move_entry member
        - Append LOG to the list (given by the parameter)
        - Calls WriteLog to rewrite the text file with the new list
    - void UndoMove(LogList *list)
        - Deletes the most recent LOG in the list (given by the parameter)
        - Calls WriteLog to rewrite the text file with the new list
    - void WriteLog(LogList *list)
        - Opens log.txt to overwrite its contents
        - Print a title line
        - Loop through the list (given by the parameter) and print each Log’s move_entry and timestamp in a separate line
        - Close file
    - char* LogText(Coordinates start, Coordinates end)
        - Takes the Coordinates of the start and end position of a chess move
        - Converts the Coordinates to a string character that will be recorded into the log
- Menu Functions
    - int Menu1(int play_option)
        - Print initial menu
        - Prompt user to select game mode
        - Scan user input to store in a variable
        - If user input is invalid, prompt user again
        - Returns user input
    - int Menu2(int game_option)
        - Prompt user to select their piece color
        - Scan user input to store in respective variable
        - If user input is invalid, prompt user again
        - Returns user input
    - int Menu3(int bot_num)
        - Prompt user to select a difficulty level
        - Scan user input to store in respective variable
        - Returns user input
    - int Menu4()
        - After all other game options have been selected, this prompts the user to start the game or to go back and change their selected options.
        - Scans for user input, and if valid, will return the integer the user chose.

- Piece Functions
    - int CheckLocation(Location* location, PieceColor player_color)
        - Returns 0 if location is empty
        - Returns 1 if location contains piece of opposite color
        - Returns 2 if location contains piece of similar color
        - Returns -1 if error occurs

- int PawnMove(Location* gameboard[][8], Coordinates coords, int for_check, CurrentPlayer player)
  - Checks all possible locations for pawn movement
  - Sets valid_move = 1 when a valid location is determined
  - Returns 0 if no possible moves
  - Returns -1 if error occurs
  - Otherwise, returns the number of possible moves
- int KnightMove(Location* gameboard[][8], Coordinates coords, int for_check)
  - Checks all possible locations for knight movement
  - Sets valid_move = 1 when a valid location is determined
  - Returns 0 if no possible moves
  - Returns -1 if error occurs
  - Otherwise, returns the number of possible moves
- int BishopMove(Location* gameboard[][8], Coordinates coords, int for_check)
  - Checks both diagonals for valid locations to move bishop
  - Sets valid_move = 1 when a valid location is determined
  - Returns 0 if no possible moves
  - Returns -1 if error occurs
  - Otherwise, returns the number of possible moves
- int RookMove(Location* gameboard[][8], Coordinates coords, int for_check)
  - Checks all vertical and horizontal locations for valid rook moves
  - Sets valid_move = 1 when a valid location is determined
  - Returns 0 if no possible moves
  - Returns -1 if error occurs
  - Otherwise, returns the number of possible moves
- int QueenMove(Location* gameboard[][8], Coordinates coords, int for_check)
  - Combines rook and bishop logic to check all possible locations for queen
  - Sets valid_move = 1 when a valid location is determined
  - Returns 0 if no possible moves
  - Returns -1 if error occurs
  - Otherwise, returns the number of possible moves
- int KingMove(Location* gameboard[][8], Coordinates coords, int for_check)
  - Checks all possible location for king movement
  - Sets valid_move = 1 when a valid move is determined
  - Returns 0 if no possible moves
  - Returns -1 if error occurs
  - Otherwise, returns the number of possible moves
- int CheckCastling(Location *gameboard[][BOARD_SIZE], Coordinates coords)
  - Calculates if castling can be performed by first checking if one of the rooks have been moved, then checks if castling is available.
- int CheckEnPassant(Location *gameboard[][BOARD_SIZE], Coordinates coords)
  - Checks if a given pawn based off coords can make en passant
  - Uses en_passant value to see if the passing location is a valid move.

3.3 Detailed description of input and output formats

- Moves are inputted by the user using two locations. The first location input is the location of the piece that the player wants to move (e.g. a2). The second location input is the location that the player wants to move their piece to (e.g. a4). While the user inputs the location using a character string, the program converts this input into integers which are then used as members of the Coordinate data structure.
- Every move is recorded in the Game Log text file "log.txt". The moves are logged in the format "Moved from [location] to [location]". Next to each logged move, a timestamp of when the move was made is also printed. Each move is represented by the data structure Log in a doubly linked list. Two of the structure's members ae character strings of the move and timestamp that will be printed to the log (e.g. "a2 to a4" and "Mon Apr 29 10:00:00 2024).



```
-----THE CHESSCRAFT MOVE LOG !-----
1. Moved from G7 to G6.....Mon Apr 29 01:15:31 2024

2. Moved from G2 to G3.....Mon Apr 29 01:15:40 2024

3. Moved from E7 to E5.....Mon Apr 29 01:15:41 2024

4. Moved from H2 to H3.....Mon Apr 29 01:15:54 2024

5. Moved from F7 to F5.....Mon Apr 29 01:15:54 2024

6. Moved from A2 to A4.....Mon Apr 29 01:16:05 2024

7. Moved from D8 to G5.....Mon Apr 29 01:16:06 2024

8. Moved from A1 to A3.....Mon Apr 29 01:16:13 2024

9. Moved from E8 to D8.....Mon Apr 29 01:16:13 2024

10. Moved from B1 to C3.....Mon Apr 29 01:16:35 2024

11. Moved from F8 to G7.....Mon Apr 29 01:16:36 2024
```

4. Development plan and timeline

4.1 Partitioning of tasks

- Software *Alph*a Version Release - April 22, 2024
  - Text-based User Interface - board, menu, user prompts
  - Chess Pieces
  - Game Logic
  - Game Log - logs moves in basic format
  - Computer Strategy - select a random move from a list of possible moves
  - Testing and Documentation
- Software v.30 Release - April 29, 2024
  - Game Log - comprehensive game log with timestamps
  - Supports three different game modes (One player, Two Players, AI vs AI)
  - In-game features (Hints, Cancel Piece)
  - Computer Strategy - AI to select best possible move

        - Supports three difficulty levels (easy, medium, hard)
    - Testing and Documentation

4.2 Team member responsibilities

- Aileen Kim: Functions for main menu and handling user input, Functions for game log
- Bryan Melendez: Structures and functions dealing with Pieces
- David Schoening: Functions creating game logic
- Gavin Nguyen: Functions for handling game flow and managing user turns
- Matthew Quach: Functions to display and update board

Back matter

Copyright

References

All material is referenced from the EECS 22L Class taught at UC Irvine.

Index