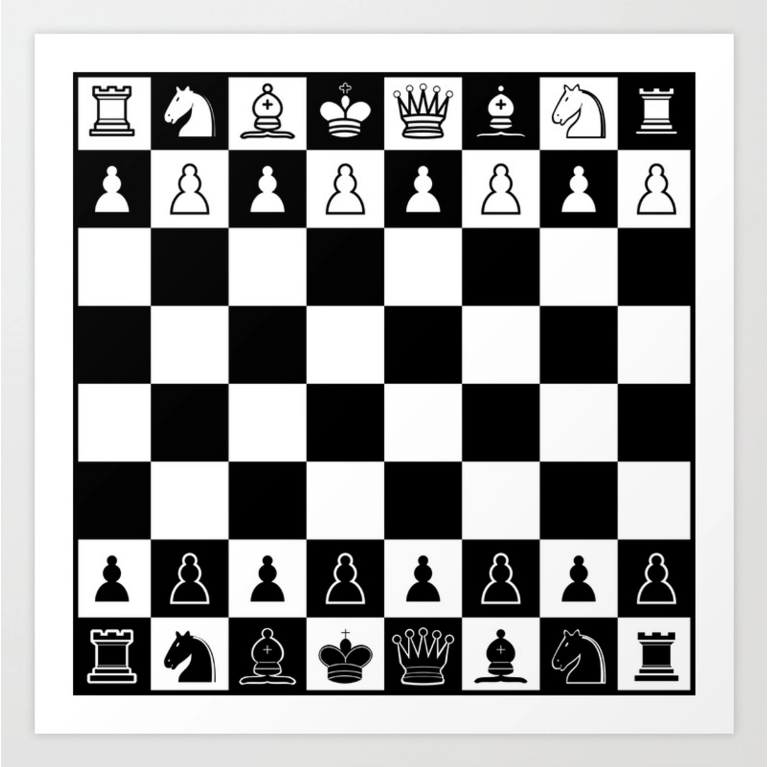
r/chess

Software Specification



Authors: Joseph Balardeta, Kevin Lee, Ryan Kim, Li-Yu Ho, Aidan Woods

Software Version: 1.0 (OFFICIAL RELEASE)

Affiliation: r/chess Team

Table of Contents

Glossary

“.c” - Describing a C file that contains a certain amount of functions pertaining to the program
“Enum” - Data type where all values we know are preset, but still part of the same variable.
“Linux” - The platform this project was programmed on, a basic operating system.
“Modules” - Modules contain small parts of the function that are slowly built out into the bigger picture of the program.
“Program” - Referring to the Chess program itself.
“Software” - Describes the set of .c files that make up the program.
“Struct” - Describes how certain elements of the game are defined.
“tasks” - Describes different elements of the program, like the main interface, the AI, etc. (also in reference to the partitioning for the group).

The Pieces

- The queen can move horizontally, vertically, and diagonally across the board.
- A rook can move horizontally and vertically across the board.
- A bishop can move diagonally across the board.
- A knight can jump to eight different squares which are two steps forward plus one step sideways from its current position.
- The king can move in any direction, but only one step at a time. Also, the king must never move into check.
- A pawn can move only forward towards the end of the board, but captures sideways. From its initial position, a pawn may make two steps, otherwise only a single step at a time. If the pawn reaches the end of the board, it is automatically promoted to another piece (usually a queen).

Other Terms

- Check - Check is a condition that occurs when a player’s king can be captured by the opponent on the next turn. A player must get out of check in the next move.
- Checkmate - Checkmate occurs when a player’s king is checked and there is no possible move for the king to escape the check. If a player’s king is checkmated, the player loses the game.
- File - Each column on the chess board is called a file.
- Rank - Each row on the chess board is called a rank.
- Castle - Castling allows the two pieces, the king and a rook to move at the same time in one move. The king can either move two spaces to the left or to the right while the rook desired to perform castling jumps across to the other side of the king, ending up being right next to the king. In order to castle though,

    - The king and the rook you want to castle with cannot have moved.
    - There must not be any pieces in between the king and the rook
    - The king cannot be in check.
    - The king cannot be in check after castling has taken place.
- En passant - May occur right after one player moves a pawn two squares forward and the opposing pawn could have captured the same pawn had it moved one square forward.
- Promotion - A pawn can be promoted to any other piece when it gets to its last rank.

Software Architecture Overview

Main data types and structures

struct Game - the main storage unit of the game, this will contain the game board (2D array) and info about the players like who's turn it is, etc.

struct ChessPiece - the main storage unit for a chess piece in memory. This will include info such as the piece's amount of moves, color, and piece type.

struct Move - includes the color of the piece moved, where it started at, and where it's going to.

enum PieceType - makes the type of the piece easier to read in code by matching a chess piece type to an associated number.

enum Color - makes the color of the player/CPU easier to read in code by matching a color name to a number.

enum PLAYER_TYPE - makes the "type" of player easier to read in the code by matching a number with the computer and the actual player.

Major software components

ChessPiece.c - Contains functions for creating, deleting, and promoting the pieces.

Move.c - Prompts for moves the performs moves.

AI.c - The brains of our chess robot, this software component works in conjunction with our MoveValidator.c component to ensure that our artificial chess player makes the best move every time.

UI.c - This part of our program is what makes everything visual for the user, this is important not just for the user's experience, but also for debugging the program more easily.

Game.c - Creates the board and the pieces in initial positions and deals with the main game loop.

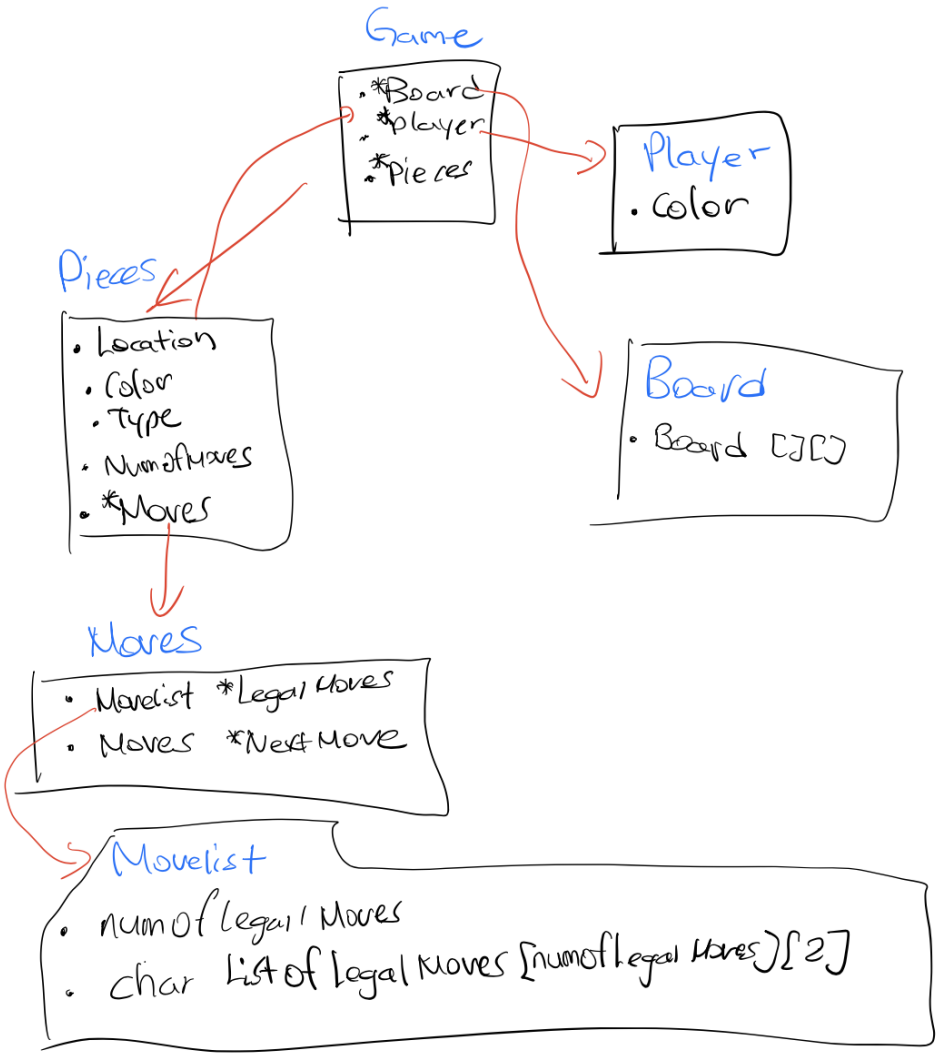
r_chess.c - The file that puts everything together. This is where the main loop of our game resides and also where most of the software components are put to use.

Rules.c - Validates moves. It should take care of all the special moves including castle, en passant and promotion. It also checks if a player's king is checked or checkmated.

MoveList.c - Creates list of moves for each instance of a game.

Log.c - Creates a log file (.txt) and logs every move of a player of a single game.

Basic Module Hierarchy



Module Interfaces

Board.h - Defines constants.

Move. h - Header file for Move.c. Defines the Move structure.

AI.h - Header file for AI.c.

UI.h - Header file for UI.c.

Game.h - Header file for Game.c. Defines the Game structure and PLAYER_TYPE enum.

ChessPiece.h - Header file for ChessPiece.c. Defines the ChessPiece structure, Color enum, and PieceType enum.

Rules.h - Header file for Rules.c.

Log.h - Header file for Log.c.

MoveList.h - Header file for Movelist.c. Defines MoveList structure and MoveEntry structure.

Overall Program Control Flow

First, the user will be prompted to enter an option in a menu when the program starts up. There will be multiple options, one being to play against the computer, another being for the computer to play against itself, an option to exit the program, an option to view log files, and an option to uninstall the game.

If the user selects the player vs computer option, then the user will be prompted to select their color for the game. After that, the game will be played, with moves being logged to a human readable file. Once the game is over, the user will be sent back to the main menu.

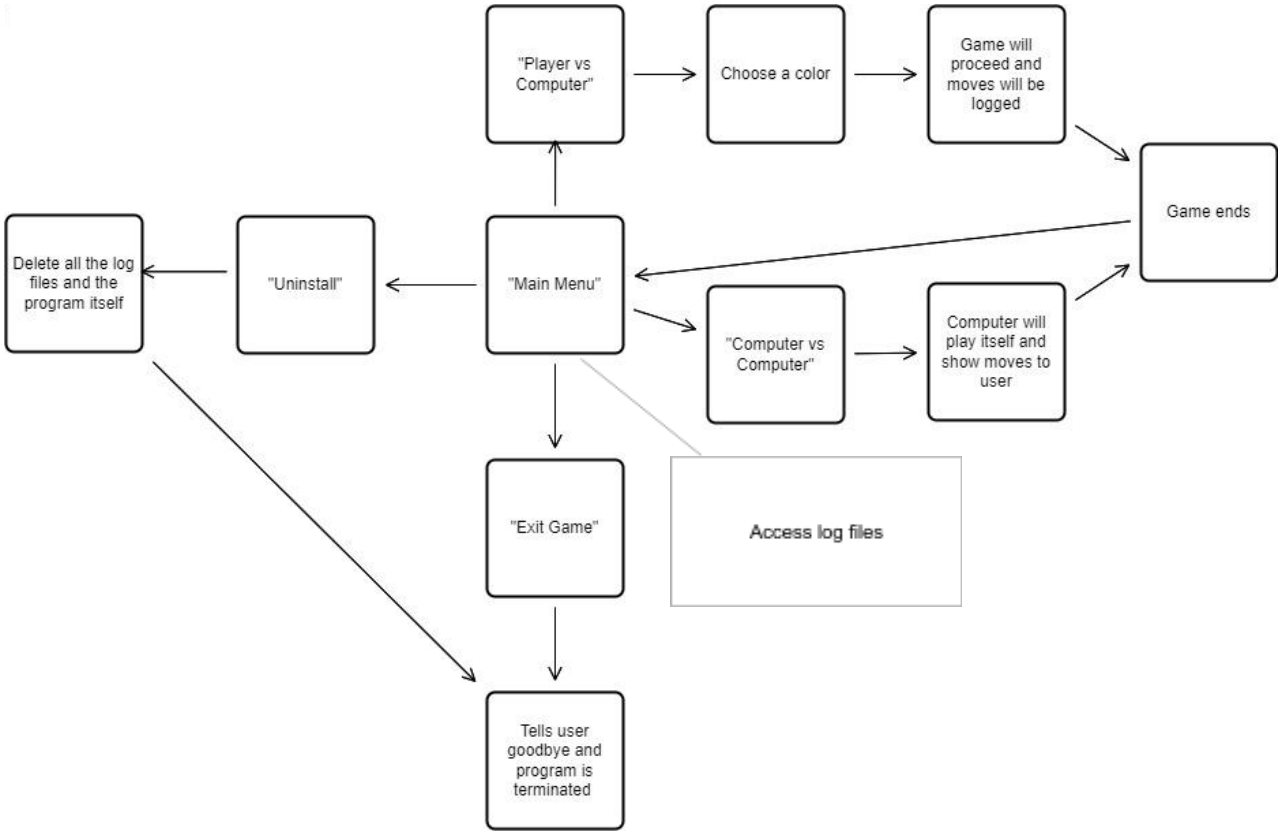
If the user selects the computer vs itself option, the computer will play a game against itself, showing its moves to the user. Once that game is over, the program will return to the main menu.

If the user selects the option to exit the program, the program will tell the user goodbye and exit.

If the user selects the option to view the log files, the program will display the .txt log file.

If the user selects the option to uninstall the program, then the program will delete all of its log files and itself, after that it will exit.

Control Flow Chart



Installation

System requirements, compatibility

- Linux OS running gcc version 1.7.1 or newer
- 50mb of disk space
- 50kb of free RAM

Setup and configuration

There is no setup required besides the instructions in the **Building, compilation, and installation** section.

Building, compilation, and installation

Building/Compilation

To set up r/chess, Type ‘tar -xvzf Chess V1.0 src.tar.gz’, then ‘cd Chess V1.0 src’. Launch the “r_chess” executable file in order to launch the game.

Installation

To uninstall the program, launch the “r_chess” executable file. Then in the menu, select the option that says “Uninstall” in order to uninstall and exit the program.

Documentation of packages, modules, interfaces

Detailed description of data structures

/* 0 represents a player-controlled side of the board, 1 represents a computer-controlled side of the board */
typedef enum {HUMAN = 0, COMPUTER = 1} PLAYER_TYPE;

/* 0 represents the white side of the board, 1 represents the black side of the board */
typedef enum {WHITE = 0, BLACK} Color;

/* numbers 0-5 represent each piece */
typedef enum {PAWN = 0, KNIGHT, BISHOP, ROOK, QUEEN, KING} PieceType;

/* Rank and file */
```
typedef struct {
    int r1; // rank to go from
    int f1; // file to go from
    int r2; // rank to go to
    int f2; // file to go to
    Color who;
} MOVE;
```

/* Contains the entire chess board, and the descriptions of each player */
```
typedef struct{
    ChessPiece *board[8][8]; // represents the game board
    PLAYER_TYPE PLAYERW; // the player controlling the white side
    PLAYER_TYPE PLAYERB; // the player controlling the black side
} Game;
```

/* Contains the type of piece, color, and position on board */
```
typedef struct{
    PieceType p_type;
    Color color;
    char name[3];
} ChessPiece;
```

```
/* Contains everything regarding moves of pieces */
typedef struct{
        MENTRY *First;
        MENTRY *Last;
        unsigned int Length;
} MoveList MLIST;

/* Contains new moves */
typedef struct{
        MLIST *mList;
        MENTRY *Next;
        MENTRY *Prev;

        MOVE *Move;

        int Points;
} MoveEntry MENTRY;
```

Detailed description of functions and parameters

```
/* Continuously running loop that plays the game until the game meets its requirement to terminate */
int main();

/* Validates user-inputted moves. Returns true or false based on validation */
int isLegalMove(Game *game, MOVE *move);

/* Gets the move from the player then checks if it is legal */
MOVE *GetUserMove(Game *game);

/* Gets the move from the computer then checks if it is legal */
MOVE *GetAIMove(Game *game);

/* Returns a list containing all the legal moves */
MLIST *GetAllLegalMoves(Game *game);

/* Ranks all possible moves */
MLIST *RankMoves(Game *game, MLIST *mList);

/* Gets the best possible move that could be made*/
MOVE *GetBestMove(Game *game, MLIST *mList);

/* Checks if the king is being checked */
int isInCheck(Game *game);
```

```
/* Returns whether the king is checkmated */
/* If the list of legal moves contains no object, the king is being checkmated */
int isCheckmate(Game *game);

/* Returns the best move a piece can do from its list of legal moves */
MOVE *GetBestMove(Game *game, MLIST *mList);

/* Moves a piece on the board. */
void Move(Game *game, MOVE *move);

/* Checks if player's wanted move is valid for the selected piece */
int IsValidPieceMove(Game *game, MOVE *move);

/* Checks for all pieces that threaten the king */
int exposesKing(Game *game, MOVE *move);

/* Checks for if piece is obstructing another piece before a move is made */
int isObstructed(Game *game, MOVE *move);

/* Checks pieces in order to determine any are eligible for promotion */
void checkPromotions(Game* game);

/* Allocates a 2d array of integer board, the players and the chess pieces */
Game *CreateGame();

/* Frees the memory space for the board and pieces */
void DeleteGame(Game *game);

/* Responsible for a single instance of a game */
void GameLoop(int option);

/* Creates piece when player moves */
ChessPiece *CreatePiece(Color color, PieceType p_type);

/* Deletes piece after player moves */
void DeletePiece(ChessPiece *piece);

/* Promotes piece once it gets to the last rank */
void PromotePiece(ChessPiece *piece, PieceType p_type);

/* Determines the winner and cleans up memory */
void EndGame(Game *game);
```

```
/* Clones the previously played board */
Game *CloneGame(Game *game);

/* Creates log file of made moves in a single instance of a game */
void CreateLog();

/* Records player's moves */
void PlayerLog(Game *game, int Rank1, int File1, int Rank2, int File2);

/* Records computer's moves */
void ComputerLog(Game *game, int Rank1, int File1, int Rank2, int File2);

/* Opens log to the player */
void OpenLog();

/* Converts char into int for the board */
int charToInt(char c);

/* Creates list of moves */
MLIST *CreateMoveList();

/* Deletes the previous list of moves */
void DeleteMoveList(MLIST *mList);

/* Adds new move to the list of moves */
void AppendMoveEntry(MLIST *mList, MOVE *m);

/* Removes move from the list of moves */
void DeleteMoveEntry(MLIST *mList, int index);

/* Prints the main menu */
void PrintMenu();

/* Switch statement that deals with different main menu options */
void ProcessInput();

/* Deletes the program and the log file */
void Uninstall();

/* Prints the playable chess board */
void PrintBoard(Game *game);

/* Prints the playable chess board in reverse */
void PrintBoardR(Game *game);
```

Detailed description of input and output formats

The user will be asked to give multiple inputs describing the square of the piece they desire to move, and the square they want that piece to go to. For example, the user could say they desire to move the piece on B6, which could be a white knight, to D5. This is a valid move, and how generally input for moves from the user will be read. The moves the player makes will be read on input, and stored in a log file as program-specific short descriptions of themselves.

An example move input by the user would look like this (assuming the player has chosen to play as white):

Enter the position of the piece you want to move: a2
Enter the position of the square you want to move your piece to: a3

The log file will log this move like this:

Player moved white pawn from a2 to a3.

Development Plan and Timeline

Partitioning of tasks

Software/Documentation Subteam (2 members of the team)

- Maintain Software Specification and User Manual, keep them up to date with the current code
- Program some of the more minor aspects of the project
    - User interface (ASCII text)
    - Log file module (Human readable alongside chess notation)
    - Main loop (user menu with options)

Software Subteam (2 members of the team)

- Program the more advanced parts of the project
    - Chess strategy module (“artificial intelligence” chess bot)
    - Chess game rules and move validator (possible/legal moves)

Software Framework Manager (1 member of the team)

- In charge of organizing the code structure for the team and overseeing all code operations to keep things running smoothly.

All members:

- Testing of code (debugging, valgrind)
- Feedback (suggestions)

These tasks are subject to slight changes based on the workload that each subteam has.

Team member responsibilities

- All members are responsible for testing/compiling any new changes made.
- All members should inform others of said changes.
- All members should attend/participate in scheduled meetings regarding development.

License and Disclaimers

References

All references are from resources/material published by Professor Doemer (specifically for the EECS 22 and EECS 22L courses of WINTER 2022 and SPRING 2022)

Index