Team 13 Developer Manual

Version: 1.0.0

Beta Release

University of California, Irvine
EECS 22L

Justin Han
Raymond Yu
Vivek Hatte
Daisuke Otagiri
Hao Ming Chiang

Table of Contents

Glossary

Application Program Interface (API) - a defined set of functions that enables data exchange between software applications. The API determines how programs should interact between the operating system or protocol.

Data Types - a list of variable types each piece, position, and color will represent.

Graphical User Interface (GUI) - an interactive user interface that includes graphical elements such as windows, icons and buttons. GUI enables programs to become user-friendly.

Installation - a method to setup the program to run later

Log file - a file that records the events that occur while the software runs.

Module Diagram - a diagram that displays the structure of the program and the connection between each module of the program

Module Interface - declares global objects (values, type definitions, exception definitions) that a module exports to make available for other modules. Modules in the program can refer to these globals using qualified identifiers.

Program Control Flow - the steps the program will execute in given user input once program opened

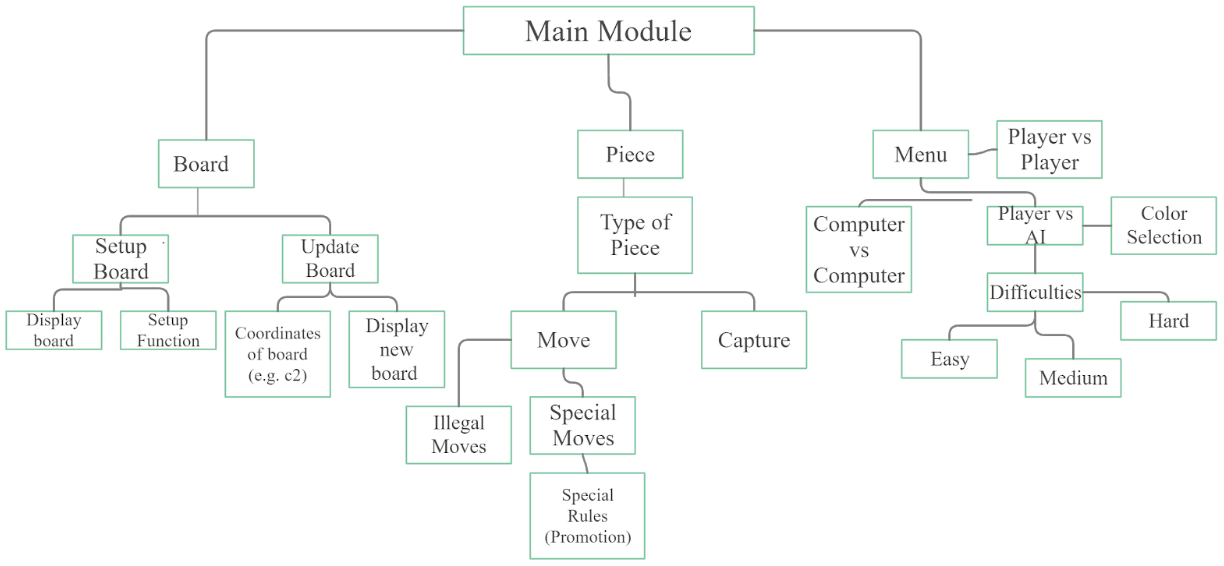
Syntax - a style an event will be formatted in that will be structured into the program

Software Architecture Overview

Main Data Types and Structures

- Main Data Types:
  - Arrays
  - Characters
  - Integers
- Board (2D integer array)
- Column letters (1D integer array)
- Chess piece names (1D integer array)

Major Software Components



(diagram created via *creately.com*)

Modules:

- main
- menu/settings
- Chess Rules
- Moves
- Special moves
- board
- AI_Algorithmll

Module Interfaces

main.c

- int main(void);

menu.c

- int menu_returned_value (void);

ChessRules.c

- void ChessRules(void);

Chess.c

- int playerVScomputer(int menu_returned_value);
- int playerVSplayer ();

Moves.c

- int move(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
- int checkLegal(int piece, int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
- int checkPawn(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
- int checkRook(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
- int checkBishop(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
- int checkQueen(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
- int checkKing(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
- int checkKnight(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
- int check4Checkl(int board[][], int color, char piece);
- int checkmate(int board[][], int color, char piece);

special_moves.c

- int castleQueen(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
- int castleKing(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
- int EnPassant(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
- int pawnPromotion(int board[][], int nextRow, int piece);
- Yet to be implemented

board.c

- void pieceRep(const int piece, char* representation);
- void displayRow(int board[8][8], const int row);
- void displayBoard(int board[8][8]);
- void write2Log(int board[8][8], const int prevRow, const int prevCol, const int currRow, const int currCol);

conversion.c

- void convertChess2Array(int* arrayRow, int* arrayCol, char colLetter, int rowNum);
- int checkChessRowBounds(int rowNum);
- int checkChessColBounds(char colLetter);
- int colLetter2ColIndex(char colLetter);
- char colIndex2ColLetter(int colIndex);
- int arrayRow2ChessRow(int rowNum);

ai.c

- int easyDifficulty(void);
- int intermediateDifficulty(void);
- int expertDifficulty(void);
- Uses minimax algorithm for making difficult or easy
- **Yet to be implemented**

pvp.c

- int playerVSplayer(void);
- int conversion_x(char x_value);
- int conversion_y(char y_value);

pvc.c

- int playerVScomputer(int menu_returned_value);

Overall Program Control Flow

- Main Menu
    - User chooses game mode (Player vs Player, Player vs AI)
        - If Player vs AI is chosen
            - User chooses what side (Black or White)
- In game
    - Board is displayed with each piece
    - White player moves first
        - Display new board
            - If Black is in checkmate, White wins
    - Black goes next
        - Displays new board
            - If White is in checkmate, Black wins
    - Repeat with white moving next

Installation

System Requirements

- Any OS that allows you to SSH into the EECS linux servers
- Standard Library (C Programming)
- Math Library
- User Interface (e.g. GUI)
- C programming compiler (e.g. gcc)

Setup and Configuration

- Log into any of the uci eecs servers accessible
  - If you are at a remote location, you need to download cisco anyconnect and connect to the uci vpn
- Create a directory locally to keep the game file
- Type in the linux server:

```
> git Pull
> cd chess
> cd Team13
> cd pvp_stuff
> make clean
>make all
>./ChessGame
```

Uninstalling

- Login to the eecs server on an ssh client (putty, etc)
- Go to the directory with the project inside visible(will be in folder Team13)
- Then type

```
> cd pvp_stuff
> make clean
>cd
> cd chess
>cd Team13
> rm -r pvp_stuff
```

Documentation of Packages, Modules, Interfaces

Detailed Description of Data Structures

- Arrays
    - Int board[8][8] - stores current board state and changes every move. Integers specify the type of piece.

0 - empty space	
1 - white rook	-1 - black rook
2 - white knight	-2 - black knight
3 - white bishop	-3 - black bishop
4 - white queen	-4 - black queen
5 - white king	-5 - black king
6 - white pawn	-6 - black pawn

    - char col_letters[8] = {‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’}
        - Character array to symbolize board position
    - Char piece_name[6][7] = {“rook”, “knight”, “bishop”, “queen”, “king”, “pawn”} characters to represent the names of the pieces

Detailed Description of Functions and Parameters

- Function Prototypes and Brief Explanation

main.c

- Main function
    - Creates the board
    - Choose game mode
    - Sets difficulty
    - Set player color
    - Pvp mode

menu.c

- **List of Functions:**
    - void start_game(int x);
        - Activates given user input (mouse click)
        - Displays the modes
    - void ChessRules(void);
        - Prints out the rules of chess

- int playerVScomputer(int menu)
    - Takes in an integer which is a returned value from the main menu, and depending on that it will chose the color the player and the difficulty rating for the AI

- int playerVScomputer(int menu_returned_value)
    - Takes in the value of the menu returned value and chooses the difficulty and color side that is wanted

moves.c

- **List of functions (tentative)**
    - int move(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
        - Moves a piece from one location on the board to the next
    - int checkLegal(int piece, int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
        - Checks if a player's move is legal. This function will be inside the move function
        - Returns 1 if move is legal and 0 if it isn't.
    - int checkPawn(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
        - Inside the legal function there will be a check on certain pieces. In this case, the function checks if the pawn is able to move to a certain position on the board given the initial position and the final position and checks for pieces in between
        - Returns 1 if move is legal and 0 if it isn't.
    - int checkRook(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
        - Checks if user selected a horizontal and vertical square (no diagonal)
        - Can only move up to blocked square (if ally occupies)
        - Returns 1 if move is legal and 0 if it isn't.
    - int checkBishop(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
        - Checks if player moves diagonally
        - Checks if path is blocked
            - Capture if opponent piece
            - Piece can only move up until blocked space if ally piece
        - Returns 1 if move is legal and 0 if it isn't.
    - int checkQueen(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
        - Checks if the queen move is legal

        - Checks if the path its going is legal by adding all the values of the array in its path to the final location
        - Returns 1 if move is legal and 0 if it isn't.
    - int checkKing(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
        - Only 1 square in every direction
        - Cannot move in blocked space (unless opponent piece)
        - Returns 1 if move is legal and 0 if it isn't.
    - int checkKnight(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
        - Checks if player chose an "L" shape path
        - Doesn't check if path is blocked
        - Checks if landing square is blocked
        - Returns 1 if move is legal and 0 if it isn't.
    - int check4Check(int board[][]);
        - Checks if the King piece is in check
        - Returns 1 if king is in check and 0 if it isn't.
    - int checkmate(int board[][], int color, char piece);
        - King piece has no legal moves to make
        - Returns 1 for checkmate and 0 for no checkmate

special_moves.c

- En passant
    - void EnPassant(int board[8][8], int prevRow, int prevCol, int nextRow, int nextCol);
        - Checks condition of whether enemy pawn move 2 squares forward
        - Grants the legal move to capture for the opponent player
- castle king side
    - void castleKingSide(int board[8][8], int prevRow, int prevCol, int nextRow, int nextCol);
        - Moves the King to the space next to the rook
        - Moves rook to space next to king
- Castle queen side
    - void castleQueenSide(int board[8][8], int prevRow, int prevCol, int nextRow, int nextCol);
        - Moves the Queen to the space next to the rook
        - Moves rook to space next to Queen
- Pawn promotion
    - void pawnPromotion(int board[8][8]);
        - Only called when a pawn reaches the end of the board
        - Grants player choice to replace pawn with any piece

board.c

- void pieceRep(const int piece, char* representation);
    - Represents a piece (int) with its char representation (Ex. ‘wR’ for 1)
- void displayRow(int board[8][8], const int row);
    - Prints out an individual row. (Called by displayBoard)
- void displayBoard(int board[8][8]);
    - Prints out the entire board with formatting.
- void write2Log(int board[][], int prevRow, int prevCol, int nextRow, int nextCol);
    - Prints out to log of past moves taken
    - Prints out if a piece has been captured
    - Prints out if a players king is in check or checkmate

Detailed Description of Input and Output Formats

- Syntax/Format of a Move Input by the User
    - The move inputted by the user is based on a click event.
    - After each legal click event made by the user, an updated board will be displayed with the detailed description of the move displayed in the log. If a click event is illegal, then an illegal statement will be displayed in the log without updating the board.
- Syntax/Format of a Move Recorded in the Log File
    - The recorded log file is based on the inputted click event.
    - There will be 2 columns, one column for the move of black and the other column for the move of white.
    - The notation will be represented as the abbreviation of the piece the user moves followed by the final position of the piece.

King = K	x: capture
Queen = Q	kingside castling: 0-0
Bishop = B	queenside castling: 0-0-0
Knight = N	check: +
Rook = R	checkmate: ++
Pawn = no notation	pawn promotion: =

Ex. White moves the left knight from its original position to c3 and Black moves the pawn at position c7 to c5.

Display of the Log

	White	Black
1.	Nc3	c5
2.	.	.
3.	.	.

Development Plan and Timeline

Partitioning of Tasks

1. Structure: Creation of Board and Chess Pieces
2. Gameplay: Movement, capture, check and checkmate
3. AI: Computer generated moves
4. UI : Menu Interfaces
5. GUI

Team Member Responsibilities

Justin Han - main.c
Raymond Yu - board.c, move.c
Vivek Hatte - main.c
Daisuke Otagiri - moves.c, special_moves.c
Hao Ming Chiang - move.c

Copyright Notice

References

U.S. Chess Federation. “Let’s Play Chess. A Summary of the Moves of Chess.” 2004. PDF File

Index