CHESS PROGRAM

Architecture Specification

Version 1.0

By:

Electric Lizards
Henry Samueli School of Engineering
University of California, Irvine

Bokor, Jacob	jbokor@uci.edu
Lam, Rayi	rayil@uci.edu
Nghi Nguyen	nghihn2@uci.edu
Shum, Ryan	rwshum@uci.edu
Lichen Wang	lichew1@uci.edu

Table of Contents

Development Glossary

AI: Artificial intelligence; an in-game entity that uses computer code to make decisions.

Doubly Linked List: A type of linked list that each node stores its data has two links. The first link points to the previous node and the second link points to the next node.

Bug: An error in computer program.

Datatype: An attribute of data which tells the compiler or interpreter how the programmer intends to use the data.

Struct: In C programming, a struct (or structure) is a collection of variables (can be of different types) under a single name.

String: An array of characters as a variable.

1. Software Architecture Overview

1.1. Main Data Types and Structures

- Board - 2D string Array
- Chess Piece - 2 character string
- Move Log - Doubly Linked List
    - MoveEntry struct: holds a board state after each move

1.2. Major Software Components

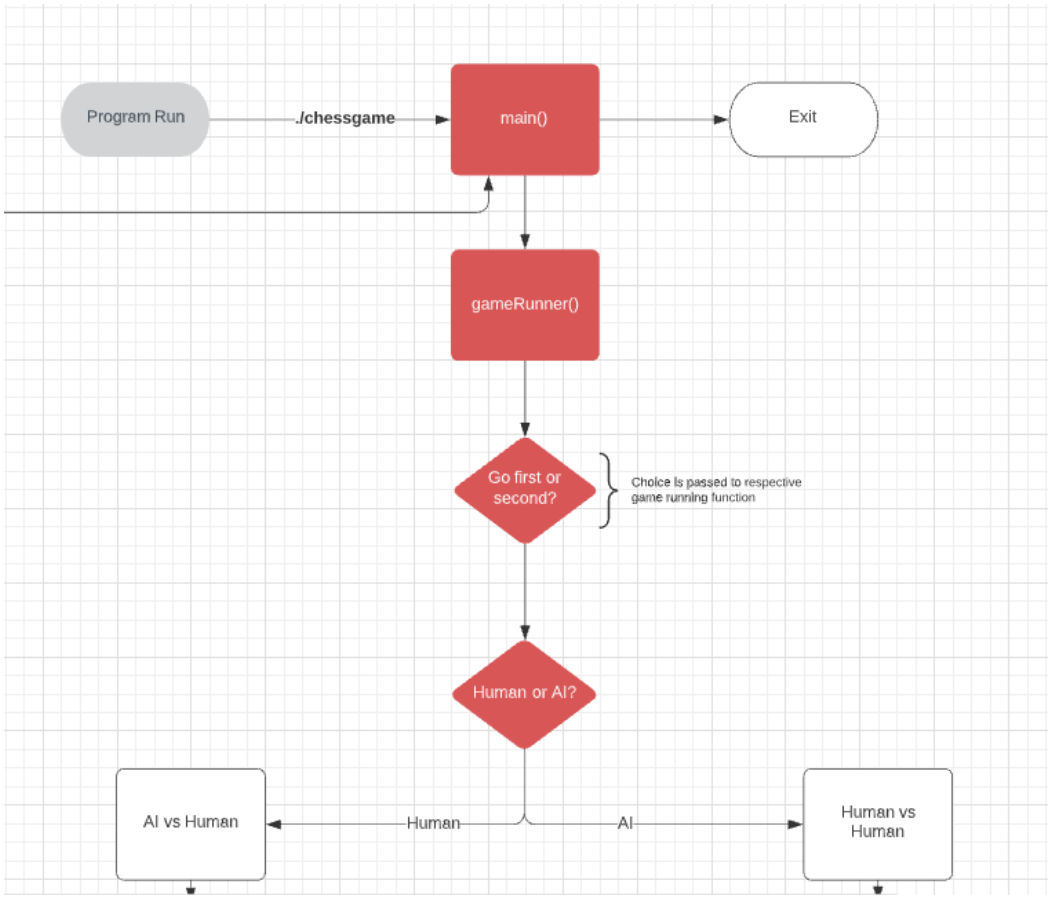
1.2.1. main.c
1.2.2. board.h , board.c
1.2.3. ai.h , ai.c
1.2.4. movelist.h , movelist.c
1.2.5. fileio.h, fileio.c
1.2.6. legalityCheck.h, legalityCheck.c

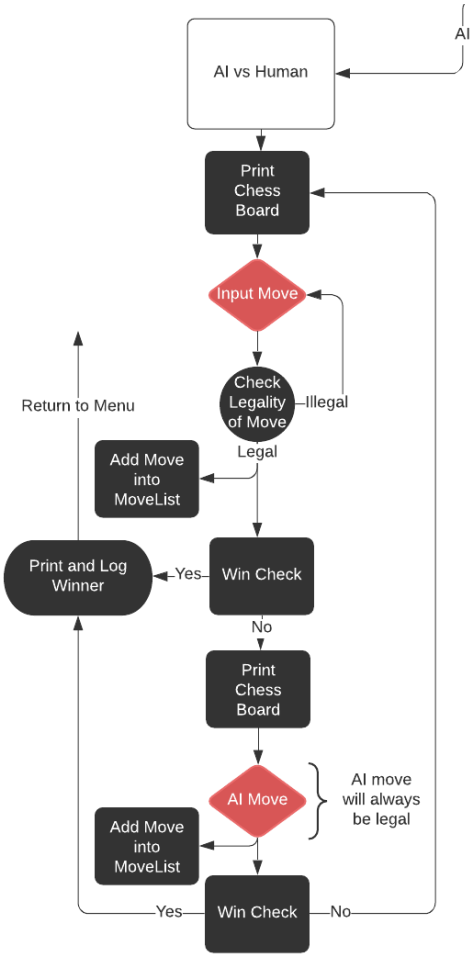
1.3. Module Interfaces

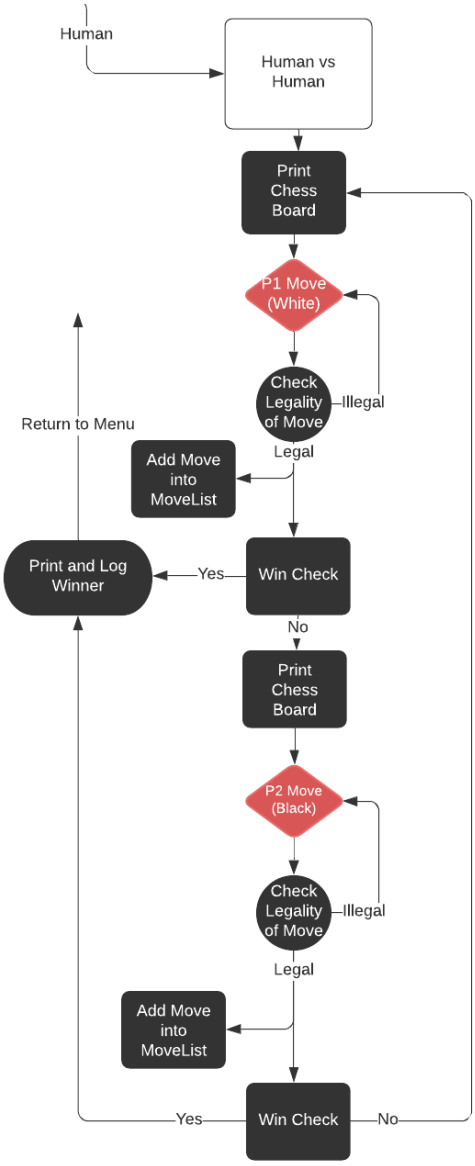
1.3.1. main.c
    1.3.1.1. main()
    1.3.1.2. printMenu()
1.3.2. board.h, board.c
    1.3.2.1. playerVsPlayer()
    1.3.2.2. playerVsAI()
    1.3.2.3. printBoard()
    1.3.2.4. playerInput()
1.3.3. ai.h, ai.c
    1.3.3.1. makeMove()
1.3.4. movelist.h, movelist.c
    1.3.4.1. newMoveEntry()
    1.3.4.2. deleteMoveEntry()
    1.3.4.3. createList()
    1.3.4.4. deleteList
    1.3.4.5. append()
    1.3.4.6. deleteNFromEnd()
    1.3.4.7. moveDifference()
1.3.5. fileio.h, fileio.c
    1.3.5.1. writeMove()
1.3.6. legalityCheck.h, legalityCheck.c
    1.3.6.1. winCheck()
    1.3.6.2. checkAndBlock()
    1.3.6.3. checkBlock()
    1.3.6.4. check()
    1.3.6.5. anyAvailableMoves()

1.3.6.6. legalMove()
1.3.6.7. checkPawn()
1.3.6.8. checkRook()
1.3.6.9. checkKnight()
1.3.6.10. checkBishop()
1.3.6.11. checkQueen()
1.3.6.12. checkKing()
1.3.6.13. Void pawnPromotion()

1.4. Overall Program Control Flow




AI
AI vs Human
Print Chess Board
Input Move
Check Legality of Move
Illegal
Legal
Return to Menu
Add Move into MoveList
Print and Log Winner
Yes
Win Check
No
Print Chess Board
AI Move
AI move will always be legal
Add Move into MoveList
Yes
Win Check
No


Human
Human vs Human
Print Chess Board
P1 Move (White)
Check Legality of Move
Illegal
Legal
Return to Menu
Add Move into MoveList
Print and Log Winner
Yes
Win Check
No
Print Chess Board
P2 Move (Black)
Check Legality of Move
Illegal
Legal
Add Move into MoveList
Yes
Win Check
No

2. Installation

2.1. System Requirements, Compatibility

2.1.1. System Requirements

Operating System: Linux
Storage Requirements: 10mb

2.1.2. Compatibility

Linux EECS Server

2.2. Setup and Configuration

The entirety of the program will be within a tarball file (.tar.gz). Once extracted, the program will be in a folder called ./chessgame/ located at the location where the tarball was extracted. Navigate to the root directory of the program and proceed to the following step.

2.3. Building, Compilation, Installation

Once in the root directory, running the “make” command will compile and combine the modules files into one executable file. The makefile will compile the program in compliance with the ANSI C99 Standard for all of the modules.

After compiling, the generated executable can be run with command “ ./chessgame “.

3. Documentation of Packages, Modules and Interfaces

3.1. Detailed description of data structures

3.1.1. Critical snippets of source code

- Board - 2D string Array

The chess board of the game will be a 2D string array that is of length 64 - 8 rows and 8 columns for the corresponding letter and number of the board location.

- Chess Piece - 2 character string

The chess piece of the game will be a 2-character-string with the first letter specifying the color of the piece and the other letter specifying the name of the chess unit.

- Move Log - Doubly Linked List
    - MoveEntry struct

```
typedef struct MoveEntry entry;
struct entry {
        MoveList* list;
        char gameBoard[8][8][2];
        entry *next;
        entry *last;
        int lastMove[4]
};

typedef struct MoveList{
        Entry* first;
        Entry* last;
}
```

The MoveEntry struct will hold a 2D string array of the game board, a pointer to the next MoveEntry, and a pointer to the previous MoveEntry. Specifically, the state of the game board will be stored in each entry. The lastMove will hold the last move that a player or AI has made in terms of integer array coordinates.

3.2. Detailed description of functions and parameters

3.2.1. Function prototypes and brief explanation

- main.c
    - int main()
        - Parameters: None
        - Return: Returns integer 0
        - Process: The main function will loop through the menu with the player vs. player and player vs. AI chess games.
    - void PrintBoard()
        - Parameters: None

        - Return: None
        - Process: The function prints out the main menu of the chess program.
- board.h, board.c
    - void playerVsPlayer(char gameBoard[8][8][2], FILE* file)
        - Parameters: The gameBoard is the board array with all the chess pieces on it. The file is the test file to write in the game replay.
        - Return: None
        - Process: The playerVsPlayer() function will loop through the human vs. human chess game. It will use the printBoard() function to print the current state of the game and the playerInput() function to take in user inputs. The board will then be printed and the next player, black, will input his/her move. After each move, the legalMove() and winCheck() function will be called to see if there is an illegal move or if there is a win. If there is an illegal move, the user will be prompted to choose again. The loop will continue until there is a win in which the winning player will be displayed. The game replay will also be available in a text file and the program will exit to the menu.
    - void playerVsAI(char gameBoard[8][8][2], FILE* file)
        - Parameters: The gameBoard is the board array with all the chess pieces on it. The file is the test file to write in the game replay.
        - Return: None
        - Process: The function will loop through the human vs. AI game. It will use the printBoard() function to print the current state of the game. Then, the playerInput() function will take in user inputs. The board will then be printed and the AI will then make its move. After each move, the legalMove() and winCheck() function will be called to see if there is an illegal move or if there is a win. If there is an illegal move, the user will be prompted to choose again. The loop will continue until there is a win in which the winning player will be displayed. The game replay will also be available in a text file and the program will exit to the menu, and the program will exit to the menu.
    - void printBoard(char gameBoard[8][8][2])
        - Parameters: gameBoard array. It holds the current state of the board in where all the chess pieces are.
        - Return: None
        - Process: The function prints the chess board along with all of the chess pieces in its respective positions.
    - bool playerInput(moveList *m, char gameBoard[8][8][2], char player)
        - Parameters: The moveList is a linked list that holds board states. The gameBoard array will hold the current state of the board with all the chess pieces in their positions. The player character is either ‘w’ or ‘b’ depending on the player color.
        - Return: If the player has exited the game or not

        - Process: The function will update the chess board according to player input. It will prompt the user for two coordinates - one for the starting position and one for the ending position. If the user inputs “exit,” the program exits the game and goes back to the main game menu. The input move and updated board will be added to the moveList.
    - int lastMoveConvert(char move[4], char from[2], char to[2], int lastMove[4])
        - Parameters: The move is the input move that the user inputs. The from and to arrays are the coordinates of the move the user makes. The lastMove is the int board value.
        - Return: Returns 0
        - Process: Converts the input character array of ASCII to the correct ASCII decimal values as character.
    - void suggestedMove(char gameBoard[8][8][2], char player)
        - Parameters: The gameBoard array will hold the current state of the board with all the chess pieces in their positions. The player character is either ‘w’ or ‘b’ depending on the player color.
        - Return: void
        - Process: prints a suggested move based on the AI function

- ai.h, ai.c
    - void makeMove(char gameBoard[8][8][2], char player)
        - Parameters: gameBoard is the board
        - Return: Void
        - Process: In playerVsAI, the function will be used in order to figure out the first possible move the AI could make.
- movelist.h, movelist.c
    - entry *newMoveEntry(moveList *m, char gameBoard[8][8][2])
        - Parameters: gameBoard is the board state to be saved
        - Return: returns pointer to newly created entry
        - Process: Creates a new MoveEntry pointer that will hold current state boards
    - void deleteMoveEntry(entry *e)
        - Parameters: e is the entry to be deleted
        - Return: void
        - Process: properly free the memory that the entry takes
    - moveList* createList()
        - Parameters: Nothing
        - Return: moveList pointer
        - Process: allocates memory for new moveList
    - void deleteList(moveList *m)
        - Parameters: m is the list to be deleted
        - Return: void
        - Process: properly free the memory that the list takes
    - void append(char gameBoard[8][8][2])

        - Parameters: gameBoard is the board state to be saved
        - Return:void
        - Process: appends a new gamestate to the end of the movetracking
    - void deleteNFromEnd(moveList *m, int n)
        - Parameters: m is the current move list, e is the entry you want to delete, n is the number of moves from the end you want to delete.
        - Return: none
        - Process: deletes the last n entries in the MoveList, and re-assigns the last pointer for the moveList.
    - void moveDifference(char *o,char gameBoard1[8][8][2], char gameBoard2[8][8][2])
        - Parameters: takes two game boards, and a 5 char string for the output
        - Return:move in chess notation
        - Process: returns the move difference between board1 and board2
    - void printList(moveList *m)
        - Parameters: takes a moveList
        - Return: none
        - Process: prints the entire list

- fileio.h, fileio.c
    - int replayGame(FILE *file, moveList *m, char gameOpponent)
        - Parameters: The file is the file that will be written into. The movelist is the list that contains all of the board states. The gameOpponent will be 1 for a player vs AI game and a player vs player game otherwise.
        - Return: Return 0
        - Process: Writes title of file, date, and all of the moves the player(s) made.
    - Int revertPrintNotation(char asciiMove[4], int lastMove[4])
        - Parameters: The ascii move is the input move in ascii. The last move is the decimal values.
        - Return: Return 0
        - Process: Converts ascii move to chess notation from int.
    - void fprintBoard(char gameBoard[8][8][2], FILE *file)
        - Parameters: gameBoard array. It holds the current state of the board in where all the chess pieces are. The file is the file that will be written into.
        - Return: None
        - Process: The function prints the chess board along with all of the chess pieces in its respective positions into the file.
- legalityCheck.h, legalityCheck.c
    - bool winCheck(char board[8][8][2])
        - Parameters: gameBoard is the board at the time of running the function.
        - Return: True or False if there is a winning player
        - Process: checks for white win, black win, or stalemate or draw.
    - bool checkAndBlock(int kPosRow, int kPosCol, char board[8][8][2], char color);

        - Parameters: kPosRow and kPosCol is the king position row and column respectively of the opposing king side color. The board is the current chess board with the chess pieces in their positions. The color is either ‘w’ or ‘b.’ The function will check if the color side will have a check or not.
        - Return: Returns a boolean if a check is on the board and if there are no other possible moves from the opposing side that could take or block the checking piece.
        - Process: The function checks if there is a check on the king position passed in. It also checks if there is a possible counter to the check like blocking the checking piece from capturing the king or taking the piece itself.
    - bool checkBlock(char board[8][8][2], char color, int array[8][2]);
        - Parameters: The board is the current chess board with the chess pieces in their positions. The color is either ‘w’ or ‘b’ for white or black respectively. The array is a set of coordinates of the path where a certain piece is attacking the king in check.
        - Return: Returns a boolean if any piece of the color is able to move to that location.
        - Process: Once there’s a check on a king, the color that has their king at check needs to see if there is a piece that could capture or block the checking piece from capturing the king.
    - bool check(int kPosRow, int kPosCol, board[8][8][2], char color);
        - Parameters: kPosRow and kPosCol is the king position row and column respectively of the opposing king side color. The board is the current chess board with the chess pieces in their positions. The color is either ‘w’ or ‘b.’ The function will check if the color side will have a check or not.
        - Return: Returns a boolean if a check is on the board.
        - Process: The function checks if there is a check on the king position passed in.
    - bool anyAvailableMoves(char board[8][8][2], char color, char enemyColor);
        - Parameters: The board is the current chess board with the chess pieces in their positions. The color is either ‘w’ or ‘b’ for white side or black side respectively. The enemyColor is the opposing side’s color.
        - Return: Returns a boolean to see if there are any possible moves left on the board.
        - Process: The function checks if there are any possible moves left on the board. The function will be used to check for stalemate - since for a stalemate to happen, there are no more legal moves left on the board.
    - bool legalMove(char[2] from, char[2] to, char board[8][8][2])
        - Parameters: from and to are the coordinates of the movement, the gameBoard is the board at the time of making the move
        - Return: True or False if there is a valid move
        - Process: Validates whether a passed move is legal
    - bool checkPawn(char from[2], char to[2], char board[8][8][2])

        - Parameters: from and to are the coordinates of the movement, the gameBoard is the board at the time of making the move
        - Return: True or False if there is a valid move
        - Process: Validates whether a passed move of a pawn is legal
    - bool checkRook(char from[2], char to[2], char board[8][8][2])
        - Parameters: from and to are the coordinates of the movement, the gameBoard is the board at the time of making the move
        - Return: True or False if there is a valid move
        - Process: Validates whether a passed move of a rook is legal
    - bool checkKnight(char from[2], char to[2], char board[8][8][2])
        - Parameters: from and to are the coordinates of the movement, the gameBoard is the board at the time of making the move
        - Return: True or False if there is a valid move
        - Process: Validates whether a passed move of a knight is legal
    - bool checkBishop(char from[2], char to[2], char board[8][8][2])
        - Parameters: from and to are the coordinates of the movement, the gameBoard is the board at the time of making the move
        - Return: True or False if there is a valid move
        - Process: Validates whether a passed move of a bishop is legal
    - bool checkQueen(char from[2], char to[2], char board[8][8][2])
        - Parameters: from and to are the coordinates of the movement, the gameBoard is the board at the time of making the move
        - Return: True or False if there is a valid move
        - Process: Validates whether a passed move of a queen is legal
    - bool checkKing(char from[2], char to[2], char board[8][8][2])
        - Parameters: from and to are the coordinates of the movement, the gameBoard is the board at the time of making the move
        - Return: True or False if there is a valid move
        - Process: Validates whether a passed move of a pawn is legal
    - void pawnPromotion(int rowFrom, int columnFrom, char board[8][8][2])
        - Parameters: rowFrom and columnFrom are the coordinates of the movement, the gameBoard is the board at the time making the move.
        - Return: None
        - Process: it is being called in checkPawn, when the pawn reaches to the other side of the board it will ask if the user wants to switch the pawn for another piece.

3.3. Detailed description of input and output formats

3.3.1. Syntax/format of a move input by the user

The format of the move input by the user is expected to be composed by letter or number. In the menu operating, the user will use arabic numbers to choose whether he wants to begin on player vs player, player vs ai or setting etc. By entering two of the letter+number, like A3, the user is able to locate and move the chess.

3.3.2. Syntax/format of a move recorded in the log file

The game log file will include the sections from top to bottom: file information, game information, move list, and winner. The file information will include the version of the game, filename, and date. The player information will show if the player of the game is white or black and if the player is a human or AI. Then, the move list will list the moves each player took chronologically. Finally, the winner is displayed.

4. Development plan and timeline

4.1. Partitioning of tasks

- Create all files, data types, and makefile
- Build game logic and menu in main
- Game logic in PvP and PvAI
- AI, legal move check, and win check
- Write move list into file

4.2. Team member responsibilities

4.2.1. Jacob: Creation for all files, filling out the .h and .c files. Create makefile and build outline of board data types, including linked list structure. Work on file output of movelist.

4.2.2. Rayi: Working on doubly linked-list for move tracking

4.2.3. Nghi: Handling legal move check and print game board.

4.2.4. Ryan: Handing Player Vs Player and Player Vs AI game runner. Create win condition check.

4.2.5. Lichen: Working on game logic and menu in main. Handle legal move check.

Back Matter

Copyright

References

Contributors: Jacob Bokor
Rayi Lam
Ryan Shum
Nghi Nguyen
Lichen Wang

Affiliation: Henry Samueli School of Engineering
University of California, Irvine

Index