CS220 - Arch. from a Prog. Perspective
Project 1: Chess Engine

Due: 03/31/2020 11:59pm

1 Introduction

In this project, you will develop a fully functional chess engine capable of solving chess puzzles. Particularly, two types of puzzles must be solved. 1) Mate in 1, and 2) Mate in 2. Before you begin, familiarize yourself with the rules of the chess game[1].

1.1 Forced mate in 1

Given an initial position and the color that is to move (white or black), you are to find a single move that would deliver checkmate to the opponent. Depending on the position, it is possible that more than one Mate-in-1 moves exist. Finding any one of them is acceptable. For example in Figure 1, if white to move, there exist 2 mate-in-1s. The first is Qb3-b8 and the second is Qb3-g8. Either solution is acceptable.

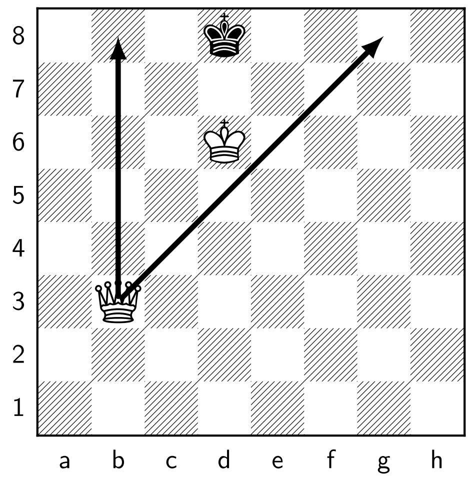


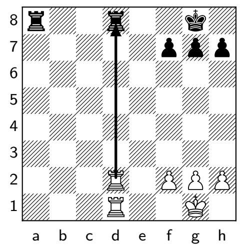
Figure 1: White to move and mate in 1

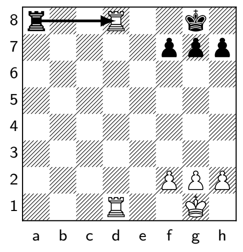
1.2 Forced mate in 2

Given an initial position and the color that is to move, you are to find a move such that irrespective of what the opponent moves, you will be able to deliver a mate-in-1 after your

[1]Chess rules: https://www.chesscentral.com/pages/first-moves-in-chess/the-rules-of-chess.html

opponent's move. For example, in Figure 2, if white to move, Rd2-d8 capture with a check is a forced mate in 2. Because the only other move is Ra8-d8 and that is followed by a mate in 1 by Rd1-d8.


(a) Initial position, Rd2-d8


(b) Ra8-d8

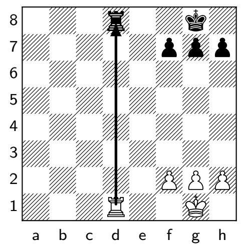

(c) Forced mate in 1, Rd1-d8

Figure 2: White to move and mate in 2

2 Data Representation

You are to use the following data representation to represent boards of individual players.

```
/* Player structure. Each player has a board to represent Rook, Bishop, Night
    , Queen, King and Pawns.
* The structure also contains a bit vector to represent possible castling
    options. */
typedef uint8_t      Flags;
typedef uint64_t       Board;
typedef unsigned int   Bool;

#define TRUE       1
#define FALSE      0

#define NO_CASTLE     0
```

```
#define CASTLE_KING    1
#define CASTLE_QUEEN   2

struct player_state {
  Flags castle_flags;
  Board r, n, b, q, k, p;
};
typedef struct player_state PlayerState;
```

Notice how we have created a Bool type although it doesn't exist in C. This is not a true boolean type, but a typedef'd integer. This will allow us to use TRUE and FALSE as return values to indicate success or failure. Additionally, you are to use the following Move structure to represent a move:

```
typedef unsigned int     Pos;
#define UNKNOWN_POS      ((unsigned int) (-1))

enum _PlayerColor {WHITE=1, BLACK=0};
typedef enum _PlayerColor PlayerColor;

struct move {
  Pos from;
  Pos to;
  Piece piece;
  Piece promotion_choice;
  struct move *next_move;
};
typedef struct move Move;
```

Additionally, the program will maintain a few globals to reflect the different pieces of relevant information.

```
/* The enpassant square */
extern Pos ep_square;

/* The player structures. One each for black and white */
extern PlayerState player[2];

/* The color of the current player */
extern PlayerColor CurrentPlayer;
```

3 High-Level Design

At a high level, the program must accept a position as input from the user along with information regarding: 1) whose turn it is to play, 2) who can castle what sides (i.e., can black/white castle king/queen sides), 3) en passant square, if any, and 4) whether this is a mate-in-1 or mate-in-2 puzzle. The input will be provided as a string in a file. Since we are considering forced mates in 1 and 2 moves, you can afford to brute force for solution. That is, consider each legal available move and explore to see if that is a solution.

Key steps:

1. The input is parsed and stored into the PlayerState, CurrentPlayer and ep_square structures.
2. Then, depending on the puzzle, run_mate1() or run_mate2() functions are called. These functions will be written to solve the corresponding puzzle. They will both return a Move as a solution that comprises of the from square to the to square. This move will be written to a results file.
3. Several helper functions will be written to accomplish the job. You will implement a is_draw() and is_checkmate() function that returns TRUE or FALSE depending on whether the current state of the game is a draw or whether a king is currently in checkmate. The is_checkmate() function will allow us to identify a solution.
4. You will also write a function legal_moves() that generates a singly linked list of all legal moves available to a given color in a given position. It returns FALSE if no legal move is available. This function can be used to detect checkmate. A move is considered legal if after completion of the move, the king of its own color is not under check.
5. You will implement a function validate_and_move() that will validate and make a move. This function will also perform necessary captures, update ep_square, castling flags, etc.

6. You will implement necessary helper functions that will validate and perform castling, en passant capture, pawn promotion, etc.

4 Input and Output

The input board along with the color to play, castling status and en passant square will be represented in a single string. The string is formatted as follows:

1. The board representation comprises of lower case letters p, n, q, r, b and k that will be used to represent Black pieces and upper case letters P, N, Q, R, B and K that will be used to represent white pieces. Empty squares are represented by numbers, and each rank is separated by a '/'. The representation starts from a8-h8 as the first rank followed by '/', then a7-h7, then '/', and so on. For example, the string rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR represents the starting position on a chess board as in Figure 3.

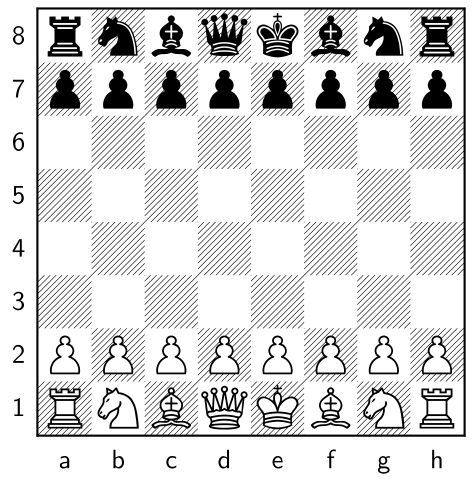


Figure 3: Starting position: rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR

2. The board representation is followed by a space followed by a single character 'w' or 'b' to represent whose turn it is, 'w' represents white, and 'b' represents black.

3. The active player color is followed by a single space followed by either a '-' to represent

no castling privilege for either player or a string comprising of a single K, single Q, single k or single q. K implies white is allowed to castle king side whereas q implies that black is allowed to castle queen side.

4. Finally, the castling flags are followed by a single space follwed by a square (e.g., e6) that represents the en passant square. Whenever a pawn moves two squares forward, the square between the starting and the ending position becomes the en passant square.

As an example, in the input: r3k2r/p4nb1/1p4Bp/2q1p1p1/pP1n4/P2Q2N1/5PP1/4K2R b Kq b3, the board is as shown in Figure 4 with Black to move. White can castle king side whereas Black can castle queen side, and square b3 is the en passant square. That is, if black wishes, this move and this move only, black can capture pawn on b4 by moving pawn a4 to b3. You will be provided a file with multiple input strings (one in each line),

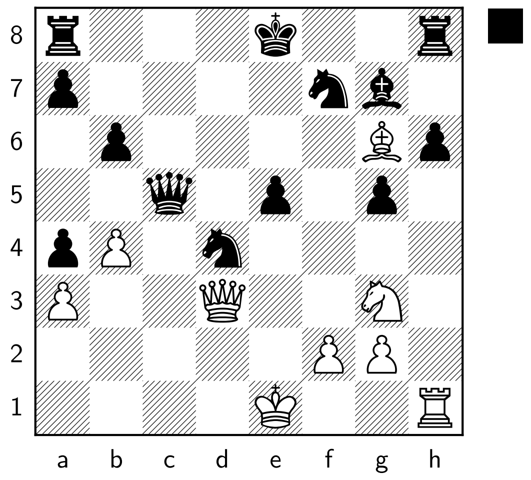


Figure 4: Black to move, White can castle king side and Black can castle queen side. En passant square is b3.

and your program must accept the string and execute the puzzle. The output move must be printed to another file (given through command line). The format of the output string should be starting_square-ending_square.

5 Supported Features

The following features must be supported.

1. Full and correct set of rules of chess including piece movement.
2. Castling as defined by the input string. All castling rules apply.
3. En passant capture as defined by the input string (or based on game moves).
4. Draw due to stalemate.
5. Detection of checkmate.
6. Promotion of pawns. This is specially important when you consider what legal moves are available to a user. If a promotion is possible, all promoted pieces (Queen, Rook, Bishop or Night) become possibilities.

6 Resources

- Official FIDE chess rules: https://www.fide.com/FIDE/handbook/LawsOfChess.pdf
- C programming: The C Programming Language. 2nd Edition by Brian Kernighan and Dennis Ritchie.
- You may download a free chess engine (e.g., Stockfish) if it helps analyze solutions to puzzles.

7 What you are allowed to (and not to) use

You may use the following functions without a need to confirm with me or the TAs (i.e., strcpy, strcat, strlen, atoi, toupper, tolower, malloc, memset, memcpy, calloc, printf, scanf, fscanf, fprintf, sprintf, sscanf). You are also encouraged to use macros, structures, unions (if necessary), etc. If there is any other function you want to use, you can check with me and have it approved.

8 Compiling and running

You must compile your code with the following flags:

```
$ gcc <source files> -g -o chess -Wall -ansi
```

The -Wall option forces the compiler to report all warnings as errors. This will force the developer to not ignore warnings.

The program will be run as follows:

```
$ ./chess puzzles.txt solution.txt <mode>
```

Where 'mode' is 1 for mate in 1 and 2 for mate in two. The puzzle.txt file contains one position per line. The solution.txt output file is to be generated by the program.

9 Starting material and sample input/output

You are provided with a sample set of starting positions and the expected outputs for mate-in-2 puzzles. The expected output for a position is listed in the line immediately succeeding the position string. E.g.,

2bqkbn1/2pppp2/np2N3/r3P1p1/p2N2B1/5Q2/PPPPKPP1/RNB2r2 w - -
f3-f7

In the example in Figure 5, a mate-in-two is possible starting Qf3-f7. You are only required to provide the first move. Subsequent moves would be Ke8-f7 followed by Bg4-h5 checkmate. Your output is not required to print the character for the piece that is being moved. You are only required to print the start square followed by a hyphen followed by the end square.

You are also provided with some source files that could help you get started. Other than the required data structures, you are encouraged to, but are not required to follow the design provided.

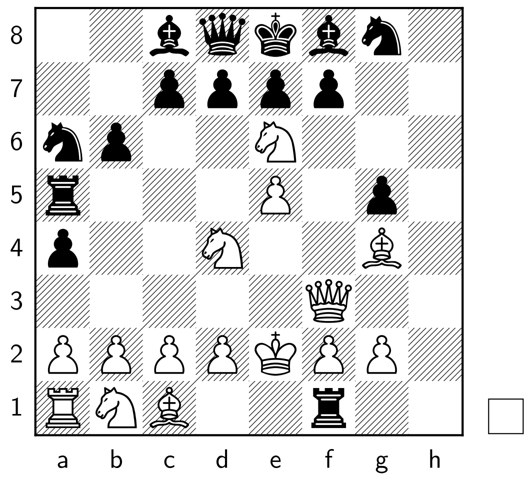


Figure 5: White to move and mate in 2. Soluton: f3-f7

10 Submission and Demo

Similar to assignment 1, you are required to upload your code at regular intervals into the github repo (which will be created shortly). Note that the commit must come from the person in the team who implemented the commit. This will help us keep track of regular commits (or progress) of not only all the teams but also all the members of the teams.

You will commit all your changes before the deadline. Your TA will download your code from github, compile it using the command above, and run your code on a large set of inputs. Based on how your code performs, you will be assigned a score and your score will be posted on mycourses. If you are not satisfied with your score, you are to set up an appointment to demo the project to your TA in person and clarify questions if any. If you are still not satisfied with your score, you may appeal the evaluation.