Obstacle Chess

RW144 Project 2023

August 5, 2023

1 Game Overview

Obstacle Chess is a variation of the traditional boardgame of chess. Please see `https://en.wikipedia.org/wiki/Rules_of_chess` for a description of the (original) chess rules. It is played on the original 8x8 chessboard with an extended set of pieces; the extensions are walls, trapdoors, and mines.

- A wall can be placed *between* two squares. Pieces cannot move through a wall, but a knight can jump over a wall, and diagonal moves (e.g., by a bishop) can go past the wall but not through two adjacent walls that block the path of the piece's move (see Figure 1). Note that a king is not in check if the attacking piece is blocked by a wall, for example the king in Figure 1 is not in check by the bishop.

- A trapdoor can be placed on a square; it is initially closed and invisible. Pieces can move over a trapdoor with no effect, however if a piece moves onto a square with a trapdoor, the trapdoor opens and becomes permenently visible and the piece is removed from the board. Any piece that moves onto an open trapdoor is also removed from the board.

- A mine can be placed under a square; it is invisible. If a piece moves onto a square with a mine, the mine explodes and is removed, and all standard pieces on the square and its immediately adjacent eight neighbors are removed, unless a wall is placed between the mine and the piece. Walls remain standing after the explosion, trapdoors remain unchanged, and a mine under a neighboring square does not explode.

Each player has three walls, one trapdoor, and one mine. Walls can be placed on the board at any time, instead of a traditional chess move. Note that placing a wall is treated the same as moving a pawn, i.e., resets the halfmove counter. The trapdoor and the mine are placed secretly before the game starts. A trapdoor can only be placed on the middle four ranks (i.e., ranks 3 to 6); a mine can only be placed on the middle two ranks (i.e., rank 4 and 5). Players are

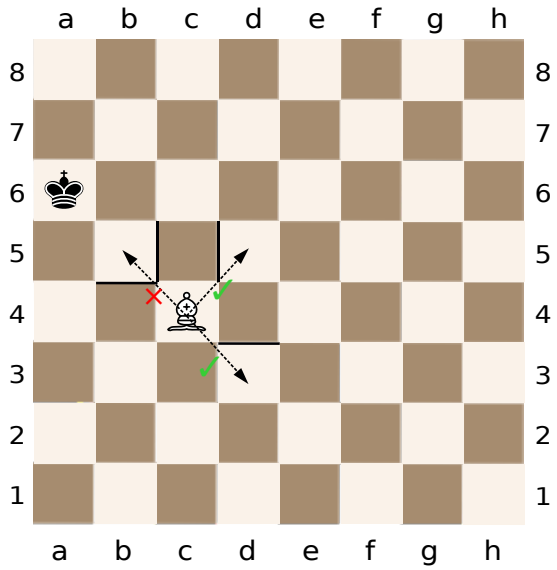

Figure 1: Valid and invalid moves passing walls. Walls are shown by the black lines. Arrows denote a move of the bishop. Valid moves are indicated by tick marks (✓), invalid moves by cross marks (✗). Note: the black king on a6 is *not* in check by the bishop.

not required to place their trapdoor or mine, but if they haven't placed them in the beginning, they cannot do so once the game has started. If a king is removed either by an exploding mine or by moving onto a trapdoor, the game is ended by checkmate. If both kings are removed at the same time by an exploding mine, the game ends in a draw by stalemate.

2 Project Description

The goal of the project is to design and create a full implementation of obstacle chess, including file input and output for boards and games, rule checking, and a complete visualization.

2.1 Command line interface

The system must support a non-graphical mode where it takes the names of an input board file, an output board file, and an optional game file, and executes the specified sequence of moves in the game file (if available) on the input board and writes the resulting board to the output board file. For example:

```
python3 obstacleChess.py inboard outboard ingame
```

reads and validates the input board given in the file `inboard`, then reads and executes the moves given in the file `ingame` over this board, and if the moves are valid, writes the resulting board to the file `outboard`. Your program must write any error messages to the standard error channel, and may write informative messages to the standard output channel. If your program encounters any error it must write the appropriate error message below and terminate (without writing an output board file). Note that a game must terminate on encountering checkmate or stalemate, however a game may continue to be played if a draw by fifty moves or threefold repetition could be claimed by a player. Error messages must use the following format:

- `ERROR: illegal board at` *p* The board is illegal; the position p (in the coordinate notation described in Section 2.3.1) is the first position (*starting from a8 and following the same order as in the board file*) where this can be determined.
- `ERROR: illegal board at status line` The combination of board and status line is illegal; the error can only be determined when the status line is read.
- `ERROR: illegal move` *m* The move m (in the move notation described in Section 2.3.1) is illegal.
- `INFO: check` The preceding move has ended in a board where one of the players is in check.
- `INFO: checkmate` The preceding move has ended in a board where one of the players is checkmate.
- `INFO: stalemate` The preceding move has ended in a board where one of the players is stalemate.
- `INFO: draw due to fifty moves` The preceding move satisfies the conditions of the fifty moves rule, and the current player could claim a draw.
- `INFO: draw due to threefold repetition` The preceding move satisfies the conditions of the threefold repetition rule, and the current player could claim a draw.

2.2 Input and output of boards

The game must allow boards to be read from and written to a file. Board files that are read must be validated and invalid board files must be reported (see Section 2.2.1 for validation rules). Each board file consists of exactly 9 non-comment lines; each of the first eight lines consist of a sequence of consecutive characters describing the state of the squares in the rank from file a to file h (i.e., left-to-right from white's perspective), starting with rank 8 (i.e., black's base rank) and using the following notation:

- white king (‘K’), queen (‘Q’), rook (‘R’), knight (‘N’), bishop (‘B’), pawn (‘P’)
- black king (‘k’), queen (‘q’), rook (‘r’), knight (‘n’), bishop (‘b’), pawn (‘p’)
- hidden trap door (‘D’)
- open trap door (‘O’)
- mine (‘M’)
- mine and hidden trap door on the same field (‘X’)
- west wall (‘|’)
- south wall (‘_’)
- empty square (‘.’, i.e., a full stop).

If walls are placed both left (i.e. west) and below (i.e. south) of a square, this must be denoted by ‘|_’, i.e., the west wall sign must precede the south wall sign. Walls must come before the character of the square which the wall is placed on.

The final line contains status information for the game; the different status blocks are separated by a single space character.

- Active colour. ‘w’ means white moves next, ‘b’ means black.
- Number of remaining walls for white
- Number of remaining walls for black
- Castling availability: four status blocks of one character each describing which of the castling moves white and black can still execute, in the order white kingside, white queenside, black kingside, black queenside. A ‘+’ denotes that the corresponding castling move is still available, a ‘-’ that it is no longer available because one of the pieces has moved already. You must *only* check that the the entries are ‘+’ or ‘-’, not that the king and corresponding rooks are on the right squares.
- En passant target square: If the last move was a two-square move by a pawn, this is the coordinate position “behind” the pawn, otherwise it is a ‘-’. You must *only* check that the target square is a well-formed field position (i.e., a1 - h8).
- Halfmove clock: This is the number of halfmoves since the last capture, pawn advance or wall placement. This is used to determine if a draw can be claimed under the fifty-move rule.

The board file may contain arbitrarily many comment lines anywhere in the file that are ignored; these are denoted by a ‘%’-sign at the beginning of the line. See Listing 1 below for an example of a board file.

```
% Board file with 4 walls, 2 mines and 1 trapdoor.
rnbqkbnr
ppp|_ppppp
........
..X.....
.|.....M.
........
PPPP_PPPP
RNBQKBNR
% status line below, note spacing
w 1 1 + + + + - 0
```

Listing 1: Example of a board file given as input.

2.2.1 Board validation

A board file is considered invalid if and only if it violates any one of the following constraints:

1. There must be one and only one king of each colour on the board, i.e. one black king and one white king.
2. The number of obstacles both placed and remaining must not exceed the total amount available. In particular:
   (a) There must be no more than two mines and two trapdoors on the board (this includes mines and trapdoors placed on the same field).
   (b) There must be no more than six walls in total, including walls on the board and remaining walls indicated in the status line.
3. Mines can only be placed on the middle two ranks (ranks 4 and 5)
4. Trapdoors can only be placed on the middle four ranks (ranks 3 to 6).
5. Pawns cannot appear in ranks 1 and 8.
6. There must be no more than 16 pieces of each colour (black and white) on the board, with the following restrictions:
   (a) There may be no more than 8 pawns.
   (b) If there are 8 pawns there may be no more than 2 rooks, 2 knights, 2 bishops and one queen.
   (c) If there are fewer than 8 pawns, there may be additional rooks, knights, bishops or queens equal to the number of missing pawns (due to potential promotion), and up to a maximum of 9 queens or 10 rooks, knights or bishops. The total number of additional queens, rooks, knights, and bishops that can be on the board depends on the number of pawns remaining on the board. (For example, you cannot have a king, two queens and ten rooks, even though the number of

queens and rooks is each within the limits, and the total number of pieces is less than 16.)

2.2.2 Error positions

- Extra characters at the end of a rank line must be reported at the last square in that rank. For example, extra characters at the end of the third non-comment line in the input board file must be reported at h6.
- Too few characters in an rank line must be reported at the square that should have been read next. For example, if there are only 5 non-wall characters on the second non-comment line in the input board file, this must be reported at f7.
- Any errors encountered on the ninth non-comment line in the input board file must be reported as an error in the status line. This includes structural errors (such as incorrect characters like '`Z`' etc.) as well as semantic errors (such as too many walls).
- Missing kings must be reported at h1

2.3 Input and output of games

Games can be read from and written to a file. Each game file contains a sequence of moves in coordinate notation (see Section 2.3.1 below). Each individual move must be on a separate line. The first move is supposed to be by white; if the file describes a partial game with black moving first, the first line must be '`...`'. The first four lines of a game file may contain the placement of trapdoors and mines, using the move notation described in Section 2.3.1. If the players do not place the same number of trapdoors and mines, lines of the form '`...`' must be used to denote the missing placements. Game files may also contain comment lines, which follow the same format as for the board files.

Note that the validity of the moves in a game file can only be interpreted in the context of a board description, whether inital or read from a file. Your system must check whether the moves are legal when it reads in the game file.

2.3.1 Coordinate Notation

Moves must be represented by coordinate notation (see `https://en.wikipedia.org/wiki/Chess_notation`). Each move consists of its starting coordinate, immediately followed by a '`-`', immediately followed by its end coordinate. Coordinates consist of a lower case letter (a-h) denoting the file, immediately followed by a digit (1-8) denoting the rank. A kingside castling move is denoted by '`0-0`', a queenside castling move by '`0-0-0`'. Promotion must be indicated by an immediately trailing '`=`*x*', with *x* denoting (in the notation described in Section 2.2) the piece to which the pawn is promoted; pawns can only be promoted to queens, rooks, knights, or bishops.

Placing a trapdoor at coordinate *xy* denoted by '`D`***xy***', placing a mine at *xy* by '`M`***xy***'. Placing a wall to the left (or west) of *xy* is denoted by '|***xy***' and placing a wall below (or south of) *xy* by '_***xy***'.

2.4 Visualization

You must implement a visualization of the chessboard using the StandardDraw library that must support board initialization and an interactive game play mode. The board initialization must allow a visual board setup and loading of a board file by providing a file selection dialog.

The board setup must provide a quick setup mode for the initial board according to the original rules, and a graphical setup mode for arbitrary boards. It must support the secret placement of the trapdoors and mines by the players after the board setup is complete. The board initialization must check that the board setup or loaded board is legal according to the obstacle chess rules, and produce a descriptive error message if a rule is violated.

In the interactive game mode, players may move pieces by clicking, on the board visualization, the source square containing the piece to move, and then the destination square. The interface must refuse illegal moves. The original chess moves remain valid, except for the restrictions introduced by walls.

The visualization must use the standard graphical symbols (see `https://en.wikipedia.org/wiki/Rules_of_chess`) to represent the pieces. You must define your own visual representation for walls and open trapdoors. The processes of castling, promotion, capturing, trapdoor opening, and mine explosion must be visualized accordingly. The visualization must keep track of whose turn it is, allow players to resign or offer and accept a draw, and notify the players of check, checkmate, and stalemate positions, and of draws according to the "threefold repetition" and "fifty move" rules. It must also provide a game log showing the sequence of moves using the obstacle chess notation (see Section 2.3.1). The game log must keep the placement of trapdoors and mines secret until the game is finished. The visualization must allow the current board or game log to be written to a file, for which a file selection dialog must be provided.

The visualization must allow players to step back and forth through the moves played so far (with an additional "home" button to return to the current state), and to change the game from any move; note that the latter step is not reversible, i.e., after changing an earlier move all later moves are deleted.

2.5 Optional extensions

The project includes the following optional extensions that you can attempt for extra marks:

Better board validation The board validation detailed in Section 2.2.1 is an under-approximation of the actual legally allowed chess boards (i.e. it allows some boards that are not valid according to the original chess rules). Implement more stringent checks such as not allowing two kings next to

each other, checking if the castling availability in the status line matches the board, checking the en passant target square is legal according to the given board, etc. NOTE: ONLY IMPLEMENT THIS IN THE THIRD AND FINAL SUBMISSION, IMPLEMENTING THIS IN THE FIRST OR SECOND SUBMISSION WILL CAUSE YOU TO LOOSE MARKS DUE TO INCORRECT FUNCTIONALITY.

Better visualization If a piece is moved, the visualization smoothly moves the piece in a movie-like fashion across the board, rather than simply removing the piece from its old position, and replacing it at its new position.

If a piece is captured, or a trapdoor opens, or a mine explodes, the process is visualized in a movie-like fashion.

Tutorial mode If a player clicks on a piece on the board, the visualization highlights the legal moves for this piece.

Networked version Your system allows two players on different computers to play against each other. Each player sees the same board (modulo hidden trapdoors and mines); they can only move their own pieces, but see the moves of their opponent.

Move sequence finder The system accepts two board descriptions as input, and outputs a legal sequence of moves that transform the the first board into the second, or reports that this is not possible.

Mechanic turk You integrate an existing standard chess playing engine (e.g., GNU chess) as opponent. Note that this requires you to check that the suggested moves are legal under the Obstacle Chess rules as well. Alternatively, you implement and integrate your own engine.

Note that these are "open-ended" and can be very challenging. You should attempt them only if you have addressed the project requirements and your base system is running very stable.

3 Progress and Submission

The project must be implemented in Python. For each submission deadline you MUST submit your code on SunLearn under the relevant project submission link.

3.1 Submissions

For each of the project's three different submission deadline, your submission MUST contain the complete source code for your project, not just the files changed or added in comparison to the previous submission.

3.1.1 Submission 1

In the first submission, you must implement board input, validation and output by the command line mode. Your program must have a `obstacleChess.py` file that can be run from the command line with the command:

```
python3 obstacleChess.py inboard outboard
```

where `inboard` and `outboard` can be any input and output file locations. Note that the optional game file argument is not required in this submission, and can thus be ignored until submission 2.

Your program must read in the board file given as input, perform board validation, and print the board to the output board file location given if the board is valid. If the program detects an illegal board file, the corresponding error message must be printed to the standard error channel, and no output board must be printed. WARNING: Board validation will contribute to the majority of this submission's marks, and thus failure to detect invalid boards will result in failing the first submission.

3.1.2 Submission 2

In the second submission, you must implement game file input, execution and output using the command line mode as in the first submission. Your program will therefore be run with the command:

```
python3 obstacleChess.py inboard outboard ingame
```

where `inboard`, `outboard` and `ingame` can be any input and output board and input game file respectively. Your program must support reading the moves of the game file, validation of each move, and execution of the moves. Any invalid moves must be reported to the standard error channel, at which point the game must be terminated and no output board must be printed. After the execution of all moves in the game file, the corresponding output board must be written to the output file location provided as in submission 1.

3.1.3 Submission 3

In the third submission you must implement the visualization described in Section 2.4, including board initialization, interactive game play mode, and move undo/redo.

4 Marking Scheme

4.1 Overview

The project will be marked out of 100, and contributes 50% to your RW144 overall course mark. Out of the 100 marks, 75 marks will be determined by

functional correctness (i.e., how well your system implements the rules of Obstacle Chess) which will include 25 marks for submission 1 and 50 marks for submission 2. The remaining 25 marks will be determined in submission 3 by the quality and completeness of the interactive mode and the visualization, and the general quality of your code (e.g., commenting and test cases).

Functional correctness (evaluated in submission 1 and 2) will be evaluated by an automated test script running your system's command line mode over a comprehensive test suite and automatically comparing the system's outputs to the expected outputs. Note that this requires your implementation to exactly follow the specifications; in particular, your implementation must be able to read and write board and game files. IF YOUR IMPLEMENTATION FAILS TO CORRECTLY READ INPUT BOARD FILES OR GAME FILES, OR FAILS TO CORRECTLY WRITE OUTPUT BOARD FILES YOU ARE LIKELY TO RECEIVE NO MARKS FOR FUNCTIONAL CORRECTNESS TESTING (AND WILL LIKELY FAIL THE PROJECT).

The quality and completeness of the interactive mode and the visualization, and the general quality of your code will be evaluated in submission 3 by a system demonstration of up to 20 minutes during week 12 or 13 of the semester.

4.2 Functional Correctness Testing

Functional correctness testing will follow a completely automatic process:

- We will download your source code from SunLearn for the corresponding submission and run the code from scratch. Your submission must contain all code required to build your system. You may not use any code that is not written by yourself or part of the textbooks standard libraries, numpy, pygame or the python standard libraries; you may, however, re-use and re-submit your own code from RW114.
- Your implementation must provide a `obstacleChess.py` file that provides a main method that implements the command line interface described above. Submissions that fail to run the `obstacleChess.py` file will receive no marks on functional correctness.
- We will run the `obstacleChess.py` file over a test suite of test cases consisting of a board input file, and a game file in submisison 2; we will automatically compare the generated board output file or any messages on the standard error channel against the predicted result. Your mark will be calculated as the percentage of passed test cases.

4.3 Demonstration

In the system demonstration we will evaluate the usability of your interactive mode. For each of the areas below, up to 5 marks will be given:

- completeness of the visualization, including castling, promotion, capturing, and mine explosion;

- usability of board setup, including placement of walls, trapdoors, and mines;
- input/output of board and game files;
- support for single-stepping through a played game (back and forth); and
- code quality.

As part of the demonstration, the demis will play a scripted game from an board input file and check the resulting log file. They may ask you about all aspects of your code, and may ask you to explain how you would implement certain smaller changes.

If you integrate an existing chess enigne as a mechanic turk optional extension in submission 3, your repository may contain its sources (if the license permits) or the binary, in a clearly labeled sub-directory.

4.4 Bonus Marks

Each of the optional extensions listed above is worth up to 5 marks; the number of bonus marks is capped at 20.