Assignment 4: Search Part 4
"Chezz" Part II.

Summary

For this assignment you will be writing a program that plays Chezz.

Note: the rules below are subject to change during the evaluation of A4, to ensure a fair balance between white and black, to prevent some pieces from dominating gameplay (overly powerful classes may get NERFed), to result in more interesting game play, and for clarification of the rules. An objective of any rule change will be to minimize the impact of having to rewrite code.

Chezz Rules

Chezz follows the same basic rules as chess (which can be found at http://www.uschess.org/content/view/7324) with the following variations:

Pieces

The following pieces are used:

- The Flinger (a.k.a. catapult) (F) – is a piece that moves like a king but cannot capture (I.e. it can never move onto an occupied square). In addition, it can fling a friendly adjacent (8-neighbourhood) piece from the piece's starting position to a location that is in the same direction as the direction from the adjacent piece to the catapult (I.e. the "flung" piece goes over the catapult and keeps going in the same direction). The "flung" piece can move any distance. It can land on an empty square or on an opposing piece (in which case the opposing piece is captured, and the flung piece is also destroyed). There is one exception: a "flung" piece cannot capture a King. A catapulted piece can fly over top of any pieces. A piece cannot be flung off the board.
- The Peon (P) – moves and captures like a regular chess pawn (except that it can only move 1 square forward on its first move, and the *en passant* capture move is not allowed).
- The Knight (N) - moves and captures like a regular chess knight.
- The Cannon (C) - can move one square forwards, backwards, left or right only (like a one-step rook). It cannot capture an opposing piece (I.e. it cannot move to an occupied square). Instead of moving it can fire a cannonball in one of the four diagonal directions (sort of like a bishop). If it fires the cannonball, then all pieces (friend or foe) on the diagonal are removed from the diagonal square next to the cannon to the edge of the board. The Cannon cannot fire a cannonball that doesn't hit any pieces (i.e. it cannot make a null move).
- Queen (Q) - moves and captures like a regular chess queen.
- King (K) - moves and captures like in regular chess (excl. castling move).
- Zombie (Z) – can move one square forwards, backwards, left or right only (like a one-step rook or like the cannon). It can move onto an enemy piece, capturing that piece (and removing it from the board).

- Bishop (B) – moves and captures like in regular chess.
- Rook (R) – moves and captures like in regular chess.

The parenthesized letters after the piece name indicate how the piece is represented in the chessboard files below.

Extra rule at the end of a player's turn - Contagion:
Once a player has completed their move, any ***enemy*** pieces in the 4 squares (forward, backward, left, right) adjacent to the player's zombies are replaced by additional ***player*** zombies (i.e. they change colour). The king and other zombies are immune to being turned into a zombie but can be captured by a zombie if it moves onto it.

End of the game:
"Checkmate" occurs when the opposing King is captured (as opposed to being placed in a position of checkmate).

Promotion of peons:
If a peon makes it to the end of the board it automatically becomes a zombie. Note that this occurs after pieces are turned into zombies by contagion, so that a freshly promoted peon doesn't turn its neighbours into zombies until the next turn.

Winning
1. "Checkmate" occurs when the opposing King is captured or your king is launched onto an enemy piece, killing your own king (not a great move).
2. A player loses the game if they make an illegal move.
3. A player loses the game if they run out of time (see below).
4. If both Kings are hit by a cannonball at the same time, the game is a draw.

Operation
Your program will operate by reading a board file from standard input ("run < board"). **NOTE:** it should not open a file. It will not be given a filename as an argument. You must supply either script called "run" that will run your code, or an executable file called run.

The input file will consist of a first line that indicates whose turn it is (white or black), followed by three integers. The first two integers represent the total time your program has used on previous moves and the total time allowed (a.k.a. *n*), respectively. If your program takes more than the remaining time available (total time minus used time), it will be disqualified. Both times are measured in milliseconds and represented as an integer. The final value, <move-no>, represents the number of moves previously played in the game.

The next line, is an opening, brace character ("{").

The following lines represent the chessboard, one line per piece. Each line consists of a square on the Chezz-board, given as a letter and a number, followed by a 2-character string in single quotes. The square and piece are separated by a colon and terminated by a comma.

- The square location is represented as a column, from “a” to “h” (right most), and a row, from 1 to 8. White begins in the low numbered rows, while black begins in the high numbered rows.
- The piece at a given location is given by a two-character string in single quotes. The first character represents the colour (w or b), while the second character represents the piece type (see above).

The next line in the file contains a closing brace character(“}“).

Because players sit on opposite sides of the board, the player playing white has the square labelled a1 at the bottom left of their board, while the player playing black has the square labelled h8 at the bottom left of their board.

An input file representing the initial board is:

```
w 0 60000 0
{
  a1: 'wF',
  a2: 'wP',
  a7: 'bP',
  a8: 'bF',
  b1: 'wN',
  b2: 'wP',
  b7: 'bP',
  b8: 'bN',
  c1: 'wC',
  c2: 'wP',
  c7: 'bP',
  c8: 'bC',
  d1: 'wQ',
  d2: 'wP',
  d7: 'bP',
  d8: 'bQ',
  e1: 'wK',
  e2: 'wZ',
  e7: 'bZ',
  e8: 'bK',
  f1: 'wB',
  f2: 'wP',
  f7: 'bP',
  f8: 'bB',
```

```
  g1: 'wN',
  g2: 'wP',
  g7: 'bP',
  g8: 'bN',
  h1: 'wR',
  h2: 'wP',
  h7: 'bP',
  h8: 'bR'
}
```

Each line in the board file will be terminated by a single new-line ('\n'; unix-style) character.

Your program should generate an output board file in the same file format and write it to the ***standard output*** (i.e. not to a file). The first line will be ignored but must be included to ensure correct parsing.

Outputting a board file that does not represent a legal board will result in your loss of the current game (see Winning 2.).

Other rules

Your program may use any datafiles you would like (with a reasonable limit on file size). These datafiles can be uploaded with your file submission or created by your program while running. The files will not be deleted between moves or between games, so your program must be prepared for this.

Your program must terminate and leave no residual processes after computing its move. Your program may not access any on-line resources (e.g. chess super computer, human expert) while computing its move.

Any attempts to win, by means other by playing chezz well, will result in disqualification. These include local denial of service attacks (e.g. filling the file system before your opponent's turn, spawning many processes before your opponent's turn, etc.).

Evaluation

A part of your program's evaluation will be based on how well it performs relative to your classmates' programs and my programs (**`firstlegal`**, **`lastlegal`**, **`randomlegal`**, and others). A "round-robin-style" competition will be held. Each program will play white once and black once against each opponent.

There may be opportunities to submit and test your code before the due date of the assignment. A competition will be held on the assignment due date. After the initial

competition, you will be permitted to submit a revised version of your program. New evaluations will be conducted every so often as new versions are submitted.

Your overall performance will be based on your average score over the various competitions.

Submission Instructions

For this assignment you will be uploading your code to the to gitlab as usual. The **`chezz-server`** will look for a script in the top level directory called "**`compile`**". It will run this script (you can make it call make, or if your code is script like python, it can do nothing). You must also supply a script called "**`run`**" that will run your code. Finally, you must supply a file called "**`name`**" (no extension). The contents of your **`name`** file will be the name of your program and will be displayed in tournament results. If you prefer your performance to remain anonymous do not include any identified (e.g. your personal name) in the **`name`** file. (NOTE: your instructor will have access to your name file, so you will not be anonymous to the instructor.)

All names must be "suitable for work", apolitical, non-discriminatory, inoffensive, and not associated with a real person other than yourself (if you chose to) etc. Please use common sense; if you doubt your ability to use common sense, ask the instructor.

To test your code, create an empty directory in a Linux system. Clone your repo into the empty directory. Also copy a board file "**`board.txt`**" to the same directory. "**`cd`**" to the directory. Then type "**`./compile`**", then type "**`./run < board.txt > board.out`**". At this point "**`board.out`**" should contain a valid board file.

You are encouraged to submit multiple times and revise your code.