Assignment 3: Chezz (Part I)

Summary

> Chess is a two-player strategy board game played on a checkered board with 64 squares arranged in an 8×8 grid. The game is played by millions of people worldwide. Chess is believed to be derived from the Indian game chaturanga sometime before the 7th century.
>
> Wikipedia: https://en.wikipedia.org/wiki/Chess (accessed Feb. 10, 2025)

Chezz is also a two-player strategy board game played on a checkered board with 64 squares arranged in an 8×8 grid. The game is played by no-one. Chezz is known to be a "rip-off" of chess.

For the next two assignments you will be working on developing an AI player for a chess variant called Chezz.

For this assignment, you will be implementing a program that computes the entire set of valid moves for a board in Chezz. This is similar to the **`edges`** function from A1.

This assignment will form the basis for the next assignment, Assignment 4, in which you will write a program that plays Chezz. For the current assignment, your program will be evaluated based on the number of correct moves that it figures out (minus the number of any incorrect moves). For Assignment 4, your program will be evaluated by letting it compete with other students' programs.

Note: the rules below are subject to change between A3 and A4, and 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 (Version 1.00)

Unless otherwise noted, Chezz follows the same basic rules as chess (which can be found at http://www.uschess.org/content/view/7324).

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.

Operation:
Your program will operate by reading a board file from standard input – i.e. "`a4 < board`" - ***do not open a file within your program***. As output, it will produce a number of output files named "`board.000`", "`board.001`", "`board.002`", etc. Each file will contain a valid board that could occur on the following turn. Your program should generate files for all possible moves and their corresponding valid boards.

The input file will consist of a first line that indicates whose turn it is (white or black), followed by three integers that are reserved for future use – each will contain a single integer.

The next section matches the format of a JavaScript Object. Specifically:

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

The following lines represent the chessboard, one line per piece. Each line may be indented using spaces or tabs. Each line consists of a square on the Chezz-board, given as a letter and a number (e.g. c4), followed by a 2-character string in single quotes (e.g. 'bZ'). 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("**}**"). And the final 3 lines of the file are reserved for future use – each will contain a single integer.

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'
}
0
0
0
```

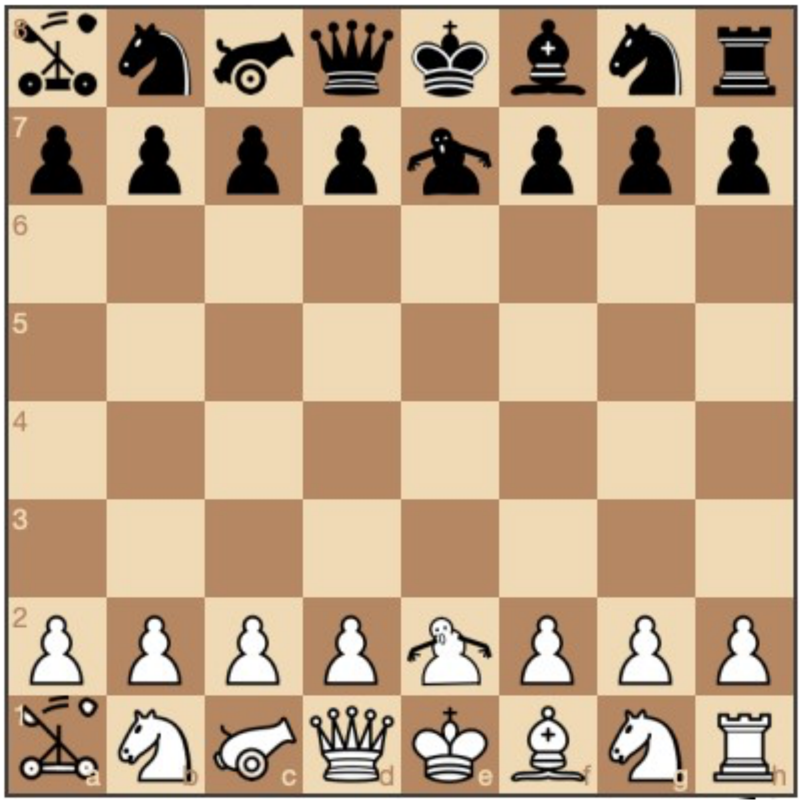
Each line in the board file will be terminated by a single new-line ('\n'; unix-style) character.

Hint for python programmers: You can quickly read the board file by using code similar to the following:

```
import sys;


a8=(0,7); b8=(1,7); c8=(2,7); d8=(3,7); e8=(4,7); f8=(5,7); g8=(6,7); h8=(7,7);
a7=(0,6); b7=(1,6); c7=(2,6); d7=(3,6); e7=(4,6); f7=(5,6); g7=(6,6); h7=(7,6);
a6=(0,5); b6=(1,5); c6=(2,5); d6=(3,5); e6=(4,5); f6=(5,5); g6=(6,5); h6=(7,5);
a5=(0,4); b5=(1,4); c5=(2,4); d5=(3,4); e5=(4,4); f5=(5,4); g5=(6,4); h5=(7,4);
a4=(0,3); b4=(1,3); c4=(2,3); d4=(3,3); e4=(4,3); f4=(5,3); g4=(6,3); h4=(7,3);
a3=(0,2); b3=(1,2); c3=(2,2); d3=(3,2); e3=(4,2); f3=(5,2); g3=(6,2); h3=(7,2);
a2=(0,1); b2=(1,1); c2=(2,1); d2=(3,1); e2=(4,1); f2=(5,1); g2=(6,1); h2=(7,1);
a1=(0,0); b1=(1,0); c1=(2,0); d1=(3,0); e1=(4,0); f1=(5,0); g1=(6,0); h1=(7,0);


colour, i1, i2, i3 = sys.stdin.readline().split()


board = { (i,j): '  ' for i in range(0,8) for j in range(0,8) }
board.update(  eval( sys.stdin.readlines()[:-3] ) )
```

An example board file is found in **`board.txt`** on CourseLink.

A visual representation of the board looks like this:



Grading

Your overall performance will be based on how many correct and how few incorrect successor boards you generate. Each board that you generate must have the same format as indicated above. You should transcribe the 3 integers from the first line without modification, and convert the first character from black to white, and vice versa. In addition to the initial board above, your program will be subjected to several other test boards (that will not be provided to you in advance).

For this assignment you are permitted and encouraged to share input game boards (test cases) with your classmates. You are also permitted to share the next board states that your program generates. But don't share code.

Submission Instructions

For this assignment you should create a script file called "`compile`" that should run in a standard Linux environment. This script should perform any compilation that may be required (e.g. if you wrote code in C or Java). If your code does not require compilation (e.g. it is a python script), you should still submit a "`compile`" script that does nothing. After running the "`compile`" script, there should be a program called "`a3`" in the directory. If you are using python, the `a3` program may be a bash script that calls the python3 interpreter and runs your python code, or you may use the shebang at the beginning of your script and turn on execute permission on your file.

To test if your submission is correct, I recommend you try the following.

1) create an empty directory,
2) copy a board file, `board.txt`, into the directory,
3) use git to retrieve your code from gitlab,
4) execute the following commands:

```
./compile
./a3 < board.txt
```

5) confirm that your directory contains all the board files representing the results of the possible moves.

Upload your code into the gitlab.