Assignment 4: Search Part 4

Summary

For this assignment you will write a program that plays "anti-chess". Your program will be evaluated by letting it compete with other students' programs.

Anti-chess rules

Anti-chess follows the same basic rules as chess (which can be found here) with the following exceptions:

- The "Special Moves" (Castling and En Passant) are not allowed.
- There is no "Check" or "Checkmate".
- The King has no special meaning and can be taken just like any other piece (with play continuing without it).
- If you are able to take any piece of your opponent, but you select a move which does not capture any piece, you have lost the game. If there is more than one piece that can be taken, you can chose which of those pieces you wish to take.
- ***If you submit a move that is not legal, you immediately lose the game.***
- ***If your opponent is unable to move any piece, then you immediately lose the game.***
- ***All promotion of pawns will be to Queens.***
- You have a total of *n* seconds (see below to find out what *n* is) to complete all the moves in the game. If you ever exceed the limit of *n* seconds total, you immediately lose the game.
- If you lose all your pieces, you have won the game.
- The game ends when a player loses all their pieces or exceeds the *n* second total time limit (there are no stalemates).

n

I will select a value of *n* that allows suitable time to search for good moves, but not so much that the game can be solved. The value is subject to change. The initial value for *n* is 60.

I/O

Your program will select one move each time it is run. It will read from the standard input stream a file describing a chess board and write to standard out a selected move. The input file will be formatted as follows:

```
<what-colour>
<piece-at-0,0><piece-at-1,0><piece-at-2,0><piece-at-3,0><piece-at-4,0><piece-at-5,0><piece-at-6,0><piece-at-7,0>
<piece-at-0,1><piece-at-1,1><piece-at-2,1><piece-at-3,1><piece-at-4,1><piece-at-5,1><piece-at-6,1><piece-at-7,1>
<piece-at-0,2><piece-at-1,2><piece-at-2,2><piece-at-3,2><piece-at-4,2><piece-at-5,2><piece-at-6,2><piece-at-7,2>
<piece-at-0,3><piece-at-1,3><piece-at-2,3><piece-at-3,3><piece-at-4,3><piece-at-5,3><piece-at-6,3><piece-at-7,3>
<piece-at-0,4><piece-at-1,4><piece-at-2,4><piece-at-3,4><piece-at-4,4><piece-at-5,4><piece-at-6,4><piece-at-7,4>
<piece-at-0,5><piece-at-1,5><piece-at-2,5><piece-at-3,5><piece-at-4,5><piece-at-5,5><piece-at-6,5><piece-at-7,5>
<piece-at-0,6><piece-at-1,6><piece-at-2,6><piece-at-3,6><piece-at-4,6><piece-at-5,6><piece-at-6,6><piece-at-7,6>
<piece-at-0,7><piece-at-1,7><piece-at-2,7><piece-at-3,7><piece-at-4,7><piece-at-5,7><piece-at-6,7><piece-at-7,7>
<time-used>
<total-time>
<move-no>
```

The value of <what-colour> can be either 'B' or 'W' to indicate that your program

should play black or white respectively. The values <piece-at-i,j> indicate which piece is located at position (*i,j*). Each value can be one of the following values:

- 'p' - white pawn
- 'P' - black pawn
- 'r' - white rook
- 'R' - black rook
- 'n' - white knight
- 'N' - black knight
- 'b' - white bishop
- 'B' - black Bishop
- 'k' - white king
- 'K' - black king
- 'q' - white queen
- 'Q' - black queen
- ' ' (blank) - square is unoccupied

The values of <time-used> and <total-time> represent the total time your program has used on previous moves and the total time allowed (a.k.a. *n*), respectively. Both times are measured in milliseconds and represented as a value that can be read by a **scanf("%d",&val)** statement. The final value, <move-no>, represents the number of moves previously played in the game. The initial board would be represented as:

```
W
RNBQKBNR
PPPPPPPP




pppppppp
rnbqkbnr
0
60000
0
```

Each line in the board file will be terminated by a single new-line (unix-style) character. Thus, the first line will consist of 2 characters, the next eight lines will consist of 9 characters each and the last 3 lines will be comprised of however many characters are required to represent the integers, plus one new-line per line.

An example board file is found in **`board.txt`** in the repo.

Your program should write to standard output your selected move in the following format:

`<start-column>,<start-row>-<end-column>,<end-row>`

Here <start-column>,<start-row> represents the initial position of the piece that you are moving, while <end-column>,<end-row> represents the final position of the piece that you are moving. Your program should add a newline after the coordinates.

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 anti-chess 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.

Evaluation
A part of your program's evaluation will be based on how well it performs relative to your classmates' programs. On March 14th, a "ladder-style" competition will be held. Each program will play white once and black once against each opponent.

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.

If enough students submit practice code before the 14th, a pre-season scrimmage competition (which will not affect your grade) may be conducted.

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

Submission Instructions
Submit via git, as usual.

Results
Stay tuned.