UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG
School of Computer Science & Applied Mathematics

COMS4033A/7044A Assignment
Move Selection

1 Introduction

We have so far implemented an approach to tracking possible board configurations. We now need to decide how to act on these potential states. To do this, we will make use of a standard chess engine called Stockfish so that we do not have to worry about writing code to play chess well (a completely separate assignment).

First, download the Stockfish executable from the internet (`https://stockfishchess.org/download/`) and place it in the same directory as your code. Rename the executable to `stockfish`. You can use the Python `chess` package to wrap the executable as follows:

```
import chess.engine
engine = chess.engine.SimpleEngine.popen_uci('./stockfish', setpgrp=True)
```

The automarker will assume that the executable is at this location with this name, so do not rename it or place it into a directory.

Before the end of your program, be sure to close the engine, otherwise your Python program will remain running:

```
engine.quit()
```

We will use this to decide on what move to make. However, we need to be careful because the aim here is to capture the king, which is an illegal move in standard chess and would cause Stockfish to crash.

To make the code compatible with the automarker, you will need to make a slight change before submitting. When submitting, please instead use the path `/opt/stockfish/stockfish`. Your code should therefore look like:

```
import chess.engine
engine = chess.engine.SimpleEngine.popen_uci('/opt/stockfish/stockfish', setpgrp=True)
```

2 Move Generation

Given a position encoded by a FEN string, your task is to generate the move to be played. A move should be decided by the following rules:

1. If the opposing king is attacked by one of our pieces, capture it
2. Otherwise, ask Stockfish for a move, giving it a time limit of 0.5 seconds.

Much of this code has already been implemented by the `Trout` bot here: `https://reconchess.readthedocs.io/en/latest/bot_create.html`. Feel free to use it.

Your Python program should output the move in the format of `<start-square><end-square>`, as in the examples below.

Sample Input #1

```
8/8/8/8/k7/8/7K/3B4 w - - 48 32
```

Sample Output #1

```
d1a4
```

Sample Input #2

```
k7/p2p1p2/P2P1P2/8/8/8/8/7K b - - 23 30
```

Sample Output #2

```
a8b8
```

Sample Input #3

```
rn3rk1/pbppq1pp/1p2pb2/4N2Q/3PN3/3B4/PPP2PPP/R3K2R w KQ - 7 11
```

Sample Output #3

```
h5h7
```

3 Multiple Move Generation

The final piece of the puzzle is to decide how to make moves when we have multiple potential states of the game. To do this, we will adopt the same approach as above across all possible boards. Write a Python program that takes in a series of boards, computes the move for each board, and then outputs the move that is most commonly recommended. If there are ties, the move that is first alphabetically should be displayed.

The first line of input is N, indicating the number of boards. N FEN strings follow, each on its own line.

Your Python program should output the move in the format of `<start-square><end-square>`, as in the examples below.

Sample Input #1

```
2
r1bqk2r/pppp1ppp/2n2n2/4B3/1b2P3/1P3N2/P1PP1PPP/RN1QKB1R b KQkq - 0 5
r1bqk2r/pppp1ppp/2n2n2/4N3/1b2P3/1P6/PBPP1PPP/RN1QKB1R b KQkq - 0 5
```

Sample Output #1

```
c6e5
```

Sample Input #2

```
4
8/3k2pn/7P/8/8/8/4K3/8 w - - 0 45
8/3k2pp/7P/8/8/8/4K3/8 w - - 0 45
8/4k1p1/7P/8/8/8/4K3/8 w - - 0 45
8/4k1p1/7P/7b/8/8/4K3/8 w - - 0 45
```

Sample Output #2

```
h6g7
```