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

Advanced Analysis of Algorithms
2021 Assignment: Bonus Section

1 Introduction

For those who are interested in testing their work against others, this is the document for you. **If not, you can stop reading here.** In this section, you will provide a text-based interface to your AI agent, which will allow us to run your code against an opponent in round robin tournaments.

2 API

Your code should read in text from standard input and execute behaviour depending on what was just read in as input. Your agent should keep track of the current position of the board, so if you have made a move, your internal board should reflect that that moves has been made. The protocol is as follows:

- `newgame` When your agent receives this string, it should setup any necessary data structures for a new game.
- `position <fen>` When your agent receives this string, it should setup the given position on its internal board. The position may be the start position, or it may be a completely random one.
- `moves ...` This is a list of moves, separated by a space, that your agent should execute. This is used to tell your agent what moves your opponent has just played.
- `go <seconds>` When your agent receives this command, it should print out its proposed move within the time limit provided. Your agent may take run for at most that time — if it does not output a move within the given timeframe, it will forfeit the match. The amount of time is specified in seconds

2.1 Example Interaction

Below is an example of interaction between two programs, `A` and `B`, and the game manager `GM`. The arrow `->` indicates input being provided from `GM` to one of the programs.

```
GM -> A: newgame
GM -> B: newgame
GM -> A: position 2ele1z/ppppppp/7/7/7/PPPPPPP/2ELE1Z w 0
GM -> B: position 2ele1z/ppppppp/7/7/7/PPPPPPP/2ELE1Z w 0
GM -> A: go 60
A: d2d3
GM -> B: moves d2d3
GM -> B: go 60
B: d6d5
GM -> A: moves d6d5
GM -> A: go 60
...
```

3 Submission

Submit to Moodle two files. The first file is a text file containing your team name and members who worked on the submission. The second file is a zip file containing your code. You may submit multiple C++ and header files, but you must provide a makefile that, when `make` is executed, will compile your code.

Your code should be able to be run on Ubuntu (so please check this before submission). Your program should also not use more than 6GB of RAM and 8 threads when operating, regardless of the depth it searches to.