A 'five-in-a-row' (gomoku) player

For this project you will implement a 'Five-in-a-row' player.

THE RULES OF THE GAME

'Five-in-a-row' is a board game played on an *N* x *N* square board. The goal of the game is to get at least five stones in a row, be it horizontal, vertical or diagonal. Of course, the game can end in a draw, if the players agree, or in the case the whole board is full of stones while no player has five in a row.

The black player begins the game by placing a stone inside a square on the board (by convention roughly at the center of the board). Then, players take turns and each player places a stone anywhere on the board.

GUIDELINES

You'll have to make two major decisions: the algorithm to use for the game player and the data structure(s).

- *Data structures*. One possible (and intuitive) representation for the board is as an *N* x *N* byte array. All cells in the byte array are initialized at 0 (no stone has been placed yet) and can have two values: -1 or 1 depending on whether the white or the black have placed a stone on it.
- *Algorithms*: The MiniMax algorithm can be used for two-player board games, such as tic-tac-toe, checkers, chess, go, and so on. These games have two things in common: they are logic games (i.e., they can be described by a set of rules and premises – as a result it is possible to know from a given point in the game, what are the next available moves) and are 'full information games' (i.e., each player knows everything about the possible moves of the adversary).

  You can find a description of the MiniMax algorithm here, here, here and in the textbook. To improve the performance of your player your may want to take a look alpha-beta pruning optimizations. This describes one set of ideas for the position evaluation function.

THE INPUTS/OUTPUTS OF YOUR PROGRAM

- *Conventions and notations*: The lower, left corner of the board has the coordinates: "0 0". Moves are denoted by "line column". Thus a move "2 3" is on the third line and fourth column of the board (since we start counting the lines and the columns at 0)
- *Command line arguments*: <program_name> [options]

```
-h, --help  show this help message and exit
-f          If specified, then the player plays black
            and makes the first move, otherwise the computer
            moves first.
-s SIZE     Board size (should be between 6 and 10)
-b          The board is printed after the computer moves
            (as we've discussed implementation is optional for this)
```

- *Input*: The game waits for opponent's moves to be entered from standard input as a pair of numbers (the line and the column) separated by space(s). There can b ean

arbitrary number of spaces at the before and after. Each move (i.e., line column pair) is entered on a different line.

- *Output*: The player outputs its moves in the same format as the inputs. If computer looses, it outputs "I lost" and exits.

IMPLEMENTATION CONSTRAINTS

- You can use C, Java or Python to implement your player.
- Your player must run on ssh-linux.ece.ubc.ca (you will deliver the executable as well as the source code and the makefile to produce the executable).
- Your player must 'think' at most one minute for a move. The player is not allowed to 'think' (i.e., consume CPU cycles) while the opponent thinks.

DEADLINE AND DELIVERABLES

- Deadline: see website.
- Deliverables: code, executables, short report. (Details will follow)