Game Playing Algorithms / Propositional Logic

The task in this programming is to implement an agent that plays the Max-Connect4 game using search. Figure 1 shows the first few moves of a game. The game is played on a 6x7 grid, with six rows and seven columns. There are two players, player A (red) and player B (green). The two players take turns placing pieces on the board: the red player can only place red pieces, and the green player can only place green pieces.

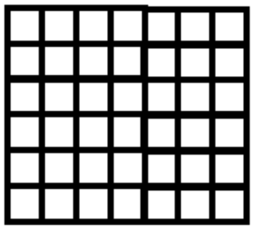
It is best to think of the board as standing upright. We will assign a number to every row and column, as follows: columns are numbered from left to right, with numbers 1, 2, ..., 7. Rows are numbered from bottom to top, with numbers 1, 2, ..., 6. When a player makes a move, the move is completely determined by specifying the COLUMN where the piece will be placed. If all six positions in that column are occupied, then the move is invalid, and the program should reject it and force the player to make a valid move. In a valid move, once the column is specified, the piece is placed on that column and "falls down", until it reaches the lowest unoccupied position in that column.

The game is over when all positions are occupied. Obviously, every complete game consists of 42 moves, and each player makes 21 moves. The score, at the end of the game is determined as follows: consider each quadruple of four consecutive positions on board, either in the horizontal, vertical, or each of the two diagonal directions (from bottom left to top right and from bottom right to top left). The red player gets a point for each such quadruple where all four positions are occupied by red pieces. Similarly, the green player gets a point for each such quadruple where all four positions are occupied by green pieces. The player with the most points wins the game.

Your program will run in two modes: an interactive mode, that is best suited for the program playing against a human player, and a one-move mode, where the program reads the current state of the game from an input file, makes a single move, and writes the resulting state to an output file. The one-move mode can be used to make programs play against each other. Note that THE PROGRAM MAY BE EITHER THE RED OR THE GREEN PLAYER, THAT WILL BE SPECIFIED BY THE STATE, AS SAVED IN THE INPUT FILE.

As part of this, you will also need to measure and report the time that your program takes, as a function of the number of moves it explores. All time measurements should report CPU time, not total time elapsed. CPU time does not depend on other users of the system, and thus is a meaningful measurement of the efficiency of the implementation.

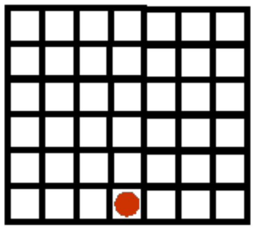
Last move: none



Score: 0-0
(red first)

Moves so far: 0

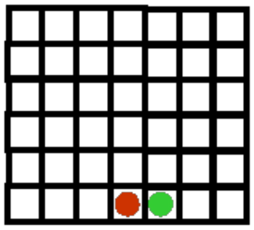
Last move: red



Score: 0-0
(red first)

Moves so far: 1

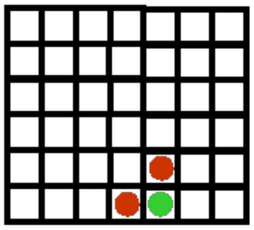
Last move: green



Score: 0-0
(red first)

Moves so far: 2

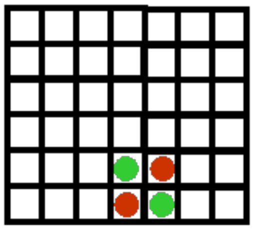
Last move: red



Score: 0-0
(red first)

Moves so far: 3

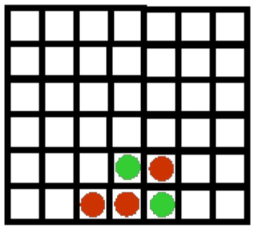
Last move: green



Score: 0-0
(red first)

Moves so far: 4

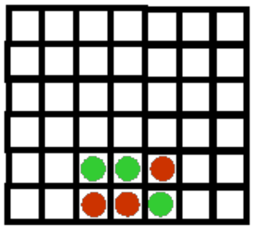
Last move: red



Score: 0-0
(red first)

Moves so far: 5

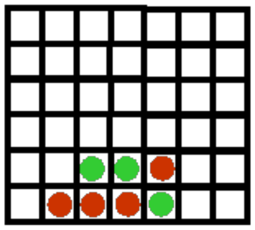
Last move: green



Score: 0-0
(red first)

Moves so far: 6

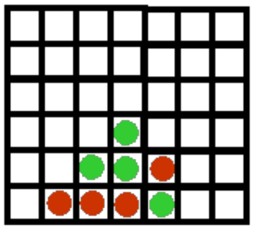
Last move: red



Score: 0-0
(red first)

Moves so far: 7

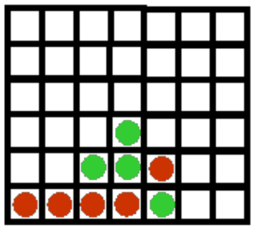
Last move: green



Score: 0-0
(red first)

Moves so far: 8

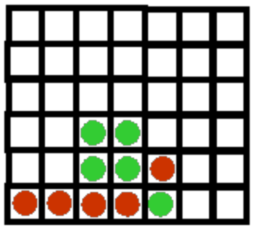
Last move: red



Score: 1-0
(red first)

Moves so far: 9

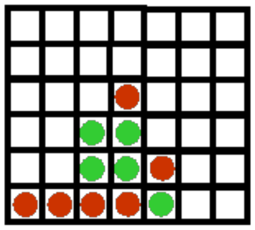
Last move: green



Score: 1-0
(red first)

Moves so far: 10

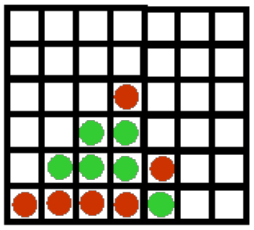
Last move: red



Score: 1-0
(red first)

Moves so far: 11

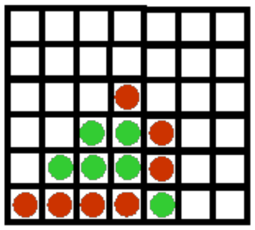
Last move: green



Score: 1-0
(red first)

Moves so far: 12

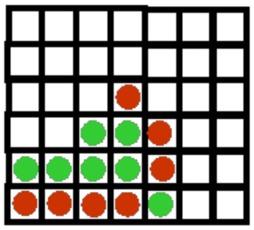
Last move: red



Score: 1-0
(red first)

Moves so far: 13

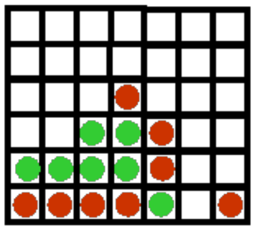
Last move: green



Score: 1-1
(red first)

Moves so far: 14

Last move: red



Score: 1-1
(red first)

Moves so far: 15

Figure 1: Sample Max-Connect Game (15 moves in)

Interactive Mode

In the interactive mode, the game should run from the command line with the following arguments (assuming a Python implementation, with obvious changes for C++ or other implementations):

Python maxconnect4 interactive [input_file] [computer-next/human-next] [depth]

For example:

Python maxconnect4 interactive input1.txt computer-next 7

- Argument interactive specifies that the program runs in interactive mode.
- Argument [input_file] specifies an input file that contains an initial board state. This way we can start the program from a non-empty board state. If the input file does not exist, the program should just create an empty board state and start again from there.
- Argument [computer-first/human-first] specifies whether the computer should make the next move or the human.
- Argument [depth] specifies the number of moves in advance that the computer should consider while searching for its next move. In other words, this argument specifies the depth of the search tree. Essentially, this argument will control the time takes for the computer to make a move.

After reading the input file, the program gets into the following loop:

1. If computer-next, goto 2, else goto 5.
2. Print the current board state and score. If the board is full, exit.
3. Choose and make the next move.
4. Save the current board state in a file called computer.txt (in same format as input file).
5. Print the current board state and score. If the board is full, exit.
6. Ask the human user to make a move (make sure that the move is valid, otherwise repeat request to the user).
7. Save the current board state in a file called human.txt (in same format as input file).

8. Goto 2.

One-Move Mode

The purpose of the one-move mode is to make it easy for programs to compete against each other, and communicate their moves to each other using text files. The one-move mode is invoked as follows:

Python maxconnect4 one-move [input_file] [output_file] [depth]

For example:

Python maxconnect4 one-move red_next.txt green_next.txt 5

In this case, the program simply makes a single move and terminates. In particular, the program should:

- Read the input file and initialize the board state and current score, as in interactive mode.
- Print the current board state and score. If the board is full, exit.
- Choose and make the next move.
- Print the current board state and score.
- Save the current board state to the output file **IN EXACTLY THE SAME FORMAT THAT IS USED FOR INPUT FILES**.
- Exit

Measuring Execution Time

You can measure the execution time for your program by inserting the word "time" in the beginning of your command line. For example, if you want to measure how much time it takes for your system to make one move with the depth parameter set to 10, try this:

```
time python maxconnect4 one-move red_next.txt green_next.txt 10
```