UNIVERSITY OF TEXAS AT ARLINGTON
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

4308/5360 **ARTIFICIAL INTELLIGENCE I** **FALL 2023**

PROGRAMMING ASSIGNMENT 2 (100 POINTS)
ASSIGNED: 10/24/2023 DUE: 11/28/2023

This assignment constitutes 15% of the course grade. You must work on it individually and are required to submit the Python programs described below along with a PDF report.

1 Introduction

In this assignment you will implement an agent that plays the **max-connect-4** game using a depth-limited version of the minimax algorithm with alpha-beta pruning. The game is played on a 6×7 grid with 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. Figure 1 shows the first few moves of a game.

It is best to visualize the board as standing upright. We assign a number to every row and column as follows. Columns are numbered from left to right with numbers $0, 2, \ldots, 6$. Rows are numbered from bottom to top with numbers $0, 2, \ldots, 5$. 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 (i.e., 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 not handle computer graphics, thus there will not be red and green pieces. Instead, one player will place 1's on the board and will be called the '1-player'. The other player will place 2's on the board and will be called the '2-player'. 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 your program may be either the 1-player or the 2-player, which will be specified by the state saved in the input file.*

As part of this assignment 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 therefore is a meaningful measurement of the efficiency of your implementation.

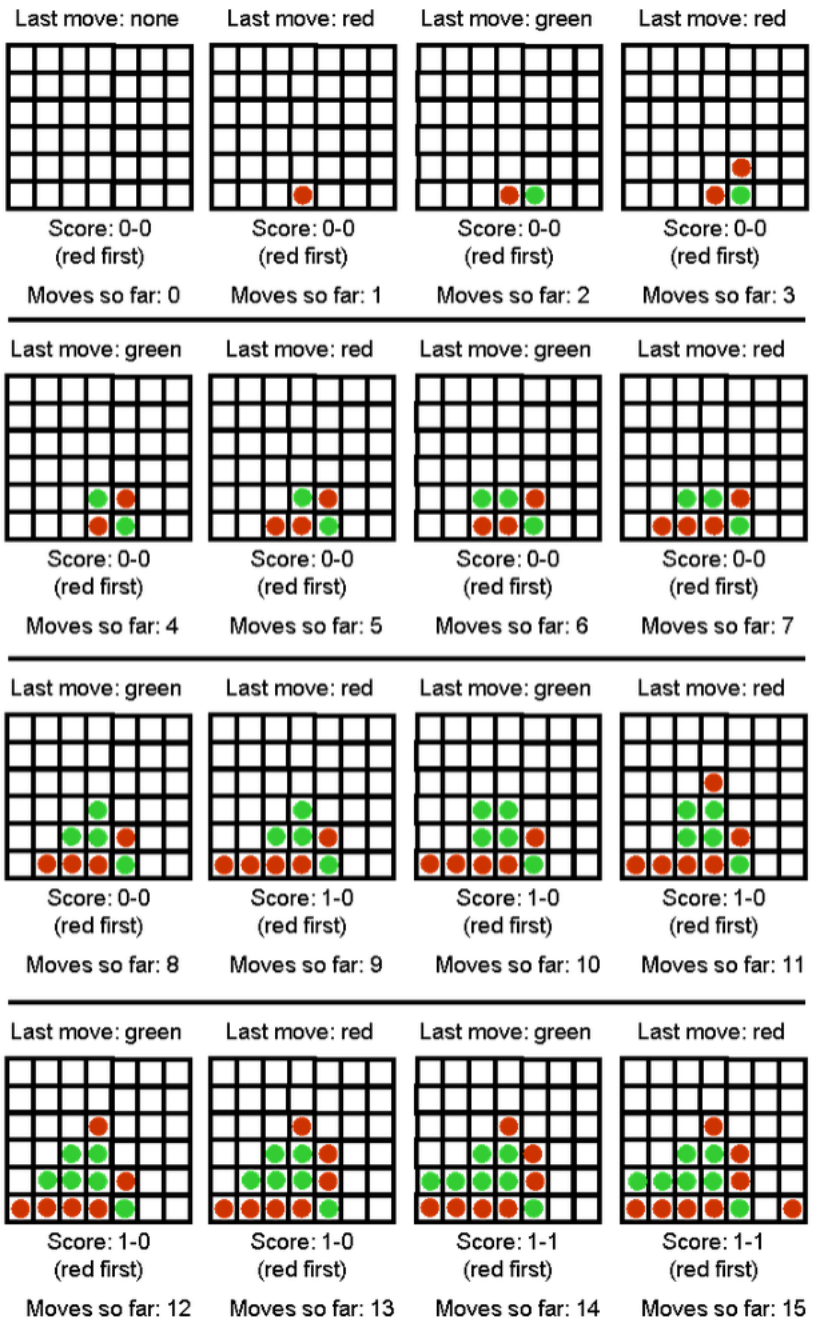

Figure 1: The first 15 moves of a max-connect-4 game.

1.1 Interactive Mode

In the interactive mode, the game should run from the command line with the following arguments:

`python maxconnect4 interactive [input_file] [computer-next/human-next] [depth]`

For example,

`python maxconnect4 interactive input1.txt computer-next 7`

is broken down as follows:

- Argument `interactive` specifies that the program is to be run in interactive mode.
- Argument `[input_file]` specifies an input file that contains the initial board state. This allows us to start the program from a non-empty board state. If the input file does not exist, then the program should create an empty board state and start again from there.
- Argument `[computer-next/human-next]` 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, i.e., this argument specifies the depth of the search tree. Basically, this argument controls the amount of time it takes for the computer to make a move.

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

1. If computer-next then goto 2, else goto 5.
2. Print the current board state and score. If the board is full, then exit.
3. Choose and make the next move.
4. Save the current board state in a file called computer.txt (in the same format as input file).
5. Print the current board state and score. If the board is full, then exit
6. Ask the human to make a move (make sure that the move is valid, otherwise repeat the request to the human). The human should specify a move by entering a column number ranging from 0 (for the leftmost column) to 6 (for the rightmost column).
7. Save the current board state in a file called human.txt (in the same format as input file).
8. Goto 2.

1.2 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 instance, the program simply makes a single move and terminates. Concretely, the program should do the following:

1. Read the input file and initialize the board state and current score similar to interactive mode.
2. Print the current board state and score. If the board is full, then exit.

3. Choose and make the next move.
4. Print the current board state and score.
5. Save the current board state to the output file *in exactly the same format used for the input files.*
6. Exit.

1.3 Python Sample Code

The Python sample code associated with this assignment is contained in the following files: maxconnect4.py and MaxConnect4Game.py. The sample code needs an input file to run. The following sample input files are included: input1.txt and input2.txt. You are free to make other input files to experiment with as long as they follow the same format. In the input files, a '0' stands for an empty position, a '1' indicates for a piece played by the first player, and a '2' denotes a piece played by the second player. The last number in the input file indicates which player plays *next* and *not* which player played last.

A command line example that runs the program, assuming that you have input1.txt saved in the same directory, is the following:

```
python maxconnect4.py one-move input1.txt output1.txt 10
```

The sample code implements a system playing max-connect-4, in one-move mode only, by making random moves. While the AI part of the sample code leaves much to be desired (your assignment is to fix that), the code can get you started by showing you how to represent and generate board states, how to save/load the game state to and from files in the desired format, and how to count the score (although faster score-counting methods are possible).

1.4 Measuring Execution Time

In your report, please note the time it takes for your program to make a move on an empty board for different depths. You should report the time for all depths starting from $depth = 1$, along with the smallest depth in which it takes longer than 30 seconds to make a move. On a Linux-based operating system, you can measure the execution time of your program by inserting the command `time` at 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 then execute the following:

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

Your output will look something like this:

```
real  0m0.003s
user  0m0.002s
sys   0m0.001s
```

1.5 Submission Instructions

Files to submit: The program in this assignment must be written in Python. You must submit your source code and report as follows. Create a directory with your last name followed by an underscore followed by your student ID

number, e.g. 'smith_1010101010'. Then, place the files in this directory, zip the directory, and upload the zip file to Canvas.

Evaluation: Your code will be graded for technical correctness. The report must contain your name and student ID number. The weight distribution for the assignment is the following:

- (30 points) Minimax implementation. If you implement minimax with no alpha-beta pruning and do not include depth-limiting, then you will only get 30 points.
- (30 points) Alpha-beta pruning implementation. If you implement minimax with alpha-beta pruning, but do not include depth-limiting, then you will get 60 points (30 for minimax, 30 for alpha-beta pruning).
- (30 points) Depth-limited minimax implementation. If you correctly implement depth-limited minimax, including alpha-beta pruning, then you will get 90 points (30 points for the depth-limited implementation, 30 points for the alpha-beta pruning implementation, and 30 points for the minimax implementation). For full credit, you need to come up with a reasonable evaluation function to be used in the context of depth-limited search. A "reasonable" evaluation function is defined to be an evaluation function that allows your program to consistently beat a random player. An obvious choice for the evaluation function is to use the score at each state, i.e., subtract the MIN player's points from the MAX player's points. However, you are free to try other evaluation functions.
- (10 points) The report must describe the implementation of the program, known bugs and deficiencies, show how to run the program, and provide the terminal session output.

Academic dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you to submit your own work only; please don't let us down. If you do, we will pursue the strongest consequences available to us. The Python sample code is copyright protected and cannot be made publicly available without written consent from the author.

1.6 Extra Credit

Up to 20 points of extra credit are available via the discretion of the grader. Extra credit can be received for novel evaluation functions, faster score-counting methods, bug fixes, etc. Any attempts at extra credit must be clearly documented in your report.