AI – Project 3

Rules:

- Plagiarism of any kind will result in failure for the course. No exceptions. If you are not sure whether or not you are plagiarizing from the web, classmates, or any other person/source, it is your responsibility to clarify it with the professor before submitting the code.
- You can be in a team with at most another student from the class. You are allowed to discuss the assignment only with your team member or the professor.

  If you have registered for the undergraduate version of the course, then your partner should also have registered for the undergraduate version of the course.

  If you have registered for the graduate version of the course, then your partner should also have registered for the graduate version of the course.

  It is ok if one member is registered for CSC and the other one for EE.
- You are allowed to write the code in C, C++, Java, Matlab, Python, or any other programming language. You should submit (via Blackboard) all your source files and a README.txt file which should describe how to compile and run the code. The README.txt file should also contain the names and email address of each team member.

  You will receive 0 points if the code does not compile, i.e., generates errors during the compilation. If your team has two members, only one team member should submit.
- Submit a printout of your code in class. Submit the source code, README, and report as one zipped directory via Blackboard.

1 Intelligent Tic-Tac-Toe on a General Board

Tic-tac-toe can be played on a general nrRows x nrCols board where a player wins by having nrToWin of its elements consecutively in a row, column, diagonal, or anti-diagonal. The traditional tic-tac-toe corresponds to the case where nrRows = nrCols = nrToWin = 3.

In this project, you will implement an alpha-beta search for the general tic-tac-toe game. The interface should be as follows:

- TTT nrRows nrCols nrToWin player1Type player2Type depth

where

- player1Type is a string corresponding to 'human', 'random', or 'AI'
- player2Type is a string corresponding to 'human', 'random', or 'AI'
- depth is a positive integer representing the evaluation depth for the alpha-beta search.

Your program should then display the tic-tac-toe board and prompt the player (if human) to make a move. If the player is 'random' or 'AI', your program should automatically make the move. Your program should display the board before and after each move. You can use simple text for displaying the board, e.g.,

```
X E E
O E X
O E X
```

where X corresponds to the first player, O corresponds to the second player, and E denotes an empty square.

For the random player, just select an empty slot at random.

For the AI player, you will implement alpha-beta search. Remember that as part of the alpha-beta search you will have to implement an evaluation function. Feel free to find and use information available on research articles and the Internet in general. Make sure to properly cite all sources that you have used. You are however not allowed to copy code.

1.1 How to test the quality of your alpha-beta search and evaluation function

First play on the 3x3 board. Your program should not lose any games. If you allow it a winning opportunity, it should be able to find it.

Play also on larger boards with different values for nrToWin. The program should be able to beat you most of the time. If you are not very good with tic-tac-toe, then try to find a friend who is good and see if your program can beat your friend. It should.

Your AI program should always beat the random player.

1.2 Report

In your report, you should show a plot (or multiple plots) of how your AI program plays against the random player. First try a 3x3 board with 3 elements to win. Vary the depth from 1 to the highest value that you can go (with your program still being able to make a move in less than 30s). For each depth, play AI vs Random 10 times. Create a table which shows the depth, number of games played, nr. times AI won, nr. times game ended in a tie, and nr. times AI lost.

Repeat this process for a 5x5 board with 4 elements to win.

Repeat this process for a 7x7 board with 4 elements to win.

Repeat this process for a 10x10 board with 5 elements to win.

2 Additional Requirements for Graduate-Course Enrollment

Implement alpha-beta with transposition tables. Add this as another player type, called ‘AItable’. Play your ‘AI’ against ‘AItable’. For the same depth, you should get the same results (as ’AI’ would be as strong as ‘AItable’). The advantage of transposition tables is that you can be more efficient. So, you are likely to go to higher depths.

Repeat the steps in section 1.2 but now using ‘AItable’ as the player.