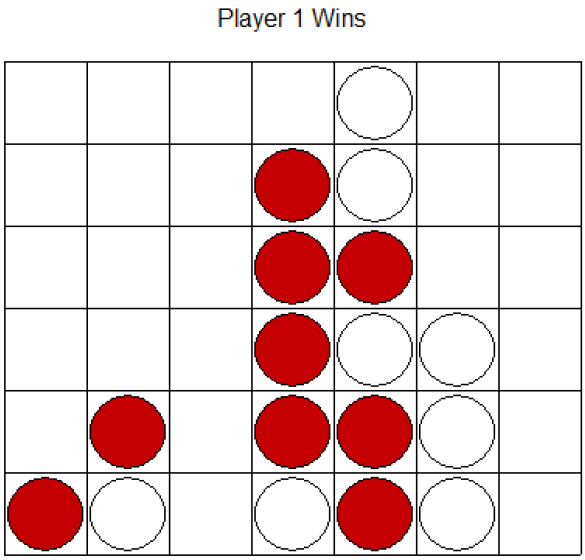
CPSC 4420/6420: Artificial Intelligence

Homework Project 3 (due Tuesday, October 12)



In this programming assignment, you will design agents for the zero-sum game Connect Four, in which two players take turns dropping a colored disc from the top into a 7x6 grid. The game is won by forming a horizontal, vertical, or diagonal line of four same colored discs. See here for more details.

Basic Requirements

After downloading the attached source code, you should be able to display the game environment and select different game modes including Agent vs Agent, and Agent vs Human by typing the following at the command line:

```
python connect4.py
```

You can run the human-vs-human version to review how the game is played. Your task is to modify connect4.py in order to implement three of algorithms covered in class: minimax, alpha-beta pruning, and expectimax. Note that as different algorithms differ only in the dispatching mechanism, if you have minimax working correctly, implementing the rest of the methods should be straightforward.

Question 1 (4 points) Minimax

Modify the `minimax` function in `connect4.py` to implement depth-limited minimax search. As it's not feasible to search the entire game tree, your code should limit the search to an arbitrary depth by using the GUI. Score the leaves of your minimax tree with the

supplied `evaluate` function in order to treat them as terminal nodes.

Question 2 (3 points) Alpha-Beta Pruning

Modify the `alphabeta` function in `connect4.py` to use alpha-beta pruning and allow more efficiently exploration of the minimax tree. You should be able to see a speed-up as the depth of the tree increases.

Question 3 (3 points) Expectimax

Minimax and alpha-beta assume that MAX plays against an adversary who makes optimal decisions. Modify
the `expectimax` function in `connect4.py` to model probabilistic behavior of opponents that may make suboptimal decisions. To do so, you will replace MIN nodes (Agent2 or Human) with chance nodes. To simplify your code, assume you will only be running against an adversary which chooses actions uniformly at random.

Important notes:

1. Depending on your GUI selection, the game can be played between two agents, or an agent and a human player, or an agent and a random player. When an agent needs to make its next move, it runs an adversarial search as a MAX player and its opponent is considered to be a MIN (or a CHANCE) player depending on the search algorithm that you run. A random player simply makes a random valid move.
2. The minimax algorithm always considers that the adversary tries to minimize the score of the MAX player that initiated the game search. The adversary never considers its own score at all during this process. Therefore, when evaluating the utilities of the nodes at the maximum tree depth, the evaluation should always be made from MAX's point of view.
3. The pesudocode provided in the slides only returns the best utility value. However, here, you need to select the policy, i.e., the action that is associated with this value. To do so, you should consider all valid actions for the MAX player at the root of the tree, and return the one that leads to the best value.
4. We have provided you with a decent evaluation function that allows for effective depth limited search. Feel free to edit the `evaluate` function, if you are able to come up with a better one. A better function means lower computation time and/or higher winning percentage as compared to ours for a fixed depth. In case you're up to the task, please include a brief ReadMe explaining your strategy.

Submission

Please submit your modified `connect4.py` file to Project/Assignment 3 on Canvas. You can work in pairs if you want to. If you do so, include both names in your report.

Getting Help

If you get stuck, please do not hesitate to contact the TA and the instructor for help, and stop by during office hours. We also encourage you to post questions and initiate discussions on Canvas and discord. Your colleagues are also there to help you.