CSCE 580: Artificial Intelligence
Coding Homework 2: Adversarial Search
Due: 2/7/2024 at 11:58pm

In this homework, you will implement an AI agent that can play Connect Four.

Your code must run in order to receive credit.

Do not change the signature (including the name) of the functions provided in the file. Your code must accept the exact arguments and return exactly what is specified in the documentation.

Installation

We will be using the same `conda` environment as in Homework 0.

The entire GitHub repository should be downloaded again as changes were made to other files. You can download it with the green "Code" button and click "Download ZIP".

Helper Functions

Your code will be implemented from the point of view of Max.

`env.is_terminal(state)`: Returns True is the state is terminal and False, otherwise.

`env.utility(state)`: Returns the utility of a terminal state. There is a utility of 1,000,000 if Max wins, a utility of -1,000,000 if Min wins, and a utility of 0 if it is a draw. Only call this function on terminal states.

`env.get_actions(state)`: Returns all legal actions for that particular state

`env.next_state(state, action)`: Returns the next state given the current state and action

`state.get_lines()`: Returns a list of numpy arrays that represent all of the rows, columns, and diagonals of the board. You may not need this, but you may use it, if you like.

`state.grid`: Is the configuration of the board. If the entry at index i, j is 0, then no piece is there. If it is 1, then Max's piece is there. If it is -1, the Min's piece is there.

Connect Four (100 pts)

Implement `make_move`. When an agent's `make_move` is called, it will always be that agent's turn. You can use any search method that you like. You may search the internet for ideas for search methods and/or heuristic functions, however, **you must cite all of your sources and you cannot copy any code, whatsoever. Your agent should take no longer than 5 seconds to make a move**.

One possible search method is heuristic minimax search. Minimax search is shown in Figure 1. Each search depth corresponds to a ply. For heuristic minimax search, if the maximum depth is reached and the state is not terminal, then the state is evaluated with a heuristic function. Hint: If you do heuristic minimax

search, to recognize when search has reached the maximum depth, you can pass a depth argument that you decrement at each depth. When that depth reaches 0, then you know you have reached the maximum depth. A depth of 4 should result in an agent that plays well.

You can play the game by clicking on the column in which you would like to place a piece. To play against your agent, run:
`python run_assignment_2.py --opponent human`

To have your agent play four games against a random agent with each person getting a turn to go first, run:
`python run_assignment_2.py --opponent random`

Your agent should beat random 100% of the time. Your agent should also play competitively against a heuristic minimax search with depth 4 and a simple heuristic that counts the number of pieces in a row and subtracts the number Min has from the number Max has.

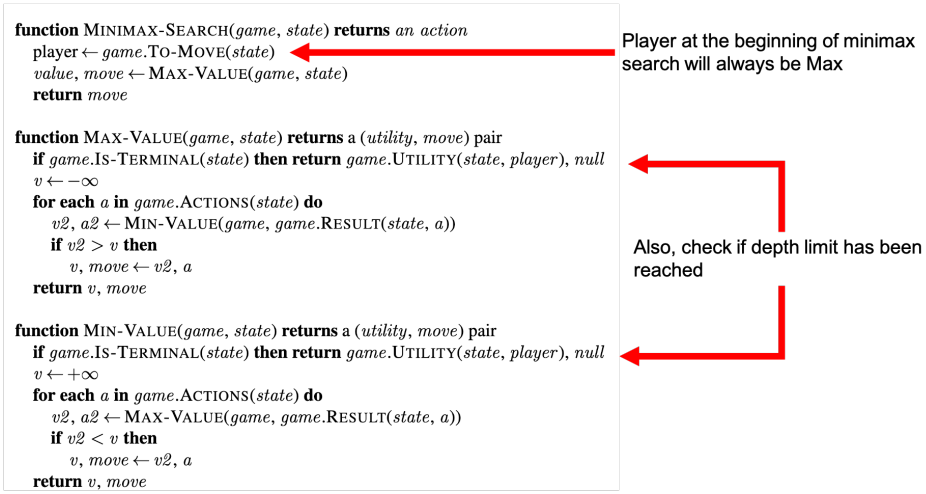

Figure 1: Minimax Search

Extra Credit (up to 50 pts)

Those who have an agent that is as good as, or better than, a secret agent, which will not be described until after the due date, will receive 30pts extra credit on this assignment. Furthermore, students may receive up to 20pts extra credit based on the results of the elimination style tournament. Move time limits will still apply here.

What to Turn In

Turn in your implementation of `coding_hw/coding_hw2.py` to Blackboard.