CSCE 580: Artificial Intelligence
Coding Homework 2: Adversarial Search

Due: 2/18/2022 at 11:59pm

In this homework, you will implement heuristic minimax search for the game Connect Four. Since searching until the end of the game would be too time consuming, we need to use a heuristic function to evaluate positions in the game that are not terminal states. You will be designing your own heuristic function to accomplish this. There will be extra credit to whoever designs the best heuristic function.

When debugging, use this code to set a breakpoint.

```
import pdb
pdb.set_trace()
```

Your code must run in order to receive credit.

Do not change the signature of the function. 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.

Heuristic Function (30 pts)

Implement `heuristic`. Given a particular state, the heuristic function should evaluate how good the position is for Max with higher values meaning the position is better for Max.

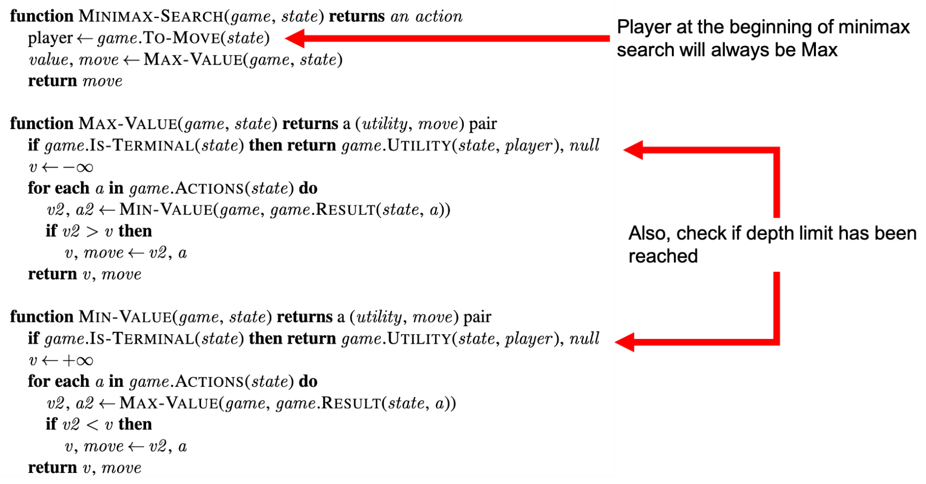
Extra Credit (30-50 pts)

Those who have a heuristic function that is as good as, or better than, a secret heuristic function which will not be described until after the due date, will receive 30pts extra credit on this assignment. The student whose heuristic function is better than the secret heuristic function and wins an elimination style tournament will receive 50pts extra credit.

Heuristic Minimax Search (70 pts)

Implement `heuristic_minimax_search`. Heuristic minimax search extends minimax search, shown in Figure 1, to the case where there is a maximum search depth where each level in the search corresponds to a ply. If we reach this depth and the state is not terminal, then we will evaluate the state with a heuristic function.

Hint: When implementing your code, 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.



```
function MINIMAX-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state)
  return move

function MAX-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
    if v2 > v then
      v, move ← v2, a
  return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
    if v2 < v then
      v, move ← v2, a
  return v, move
```

Player at the beginning of minimax search will always be Max

Also, check if depth limit has been reached

Figure 1: Minimax Search

Running the Code

You can play the game by clicking on the column in which you would like to place a piece. To get a hang of the game, you can play against an agent that simply plays randomly:

```
python run_proj2.py --opponent random
```

To play against your heuristic minimax implementation, run:

```
python run_proj2.py --opponent minimax --depth <depth>
```

When the depth argument reaches 4, your agent should start playing noticeably better. **Your agent should take no longer than 5 seconds to make a move when doing a minimax search of depth 4 using your heuristic function**.

What to Turn In

Turn in your implementation of `coding_hw/coding_hw2.py` to Blackboard. You can add helper functions to `coding_hw/coding_hw2.py`, if need be.