CSCE 580: Artificial Intelligence
Project 2: Adversarial Search
Due: 2/23/2021 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. It will be your best friend.

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

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.

`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.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, evaluate the state such that the function returns a higher value if Max appears to be winning and a lower value if Min appears to be winning.

To get full credit, you can implement a basic heuristic function that counts the number of instances that Max has three pieces in a row and subtract from that the number of instances that Min has three pieces in a row.

However, you can get extra credit for implementing a more sophisticated heuristic function.

Extra Credit (30 pts)

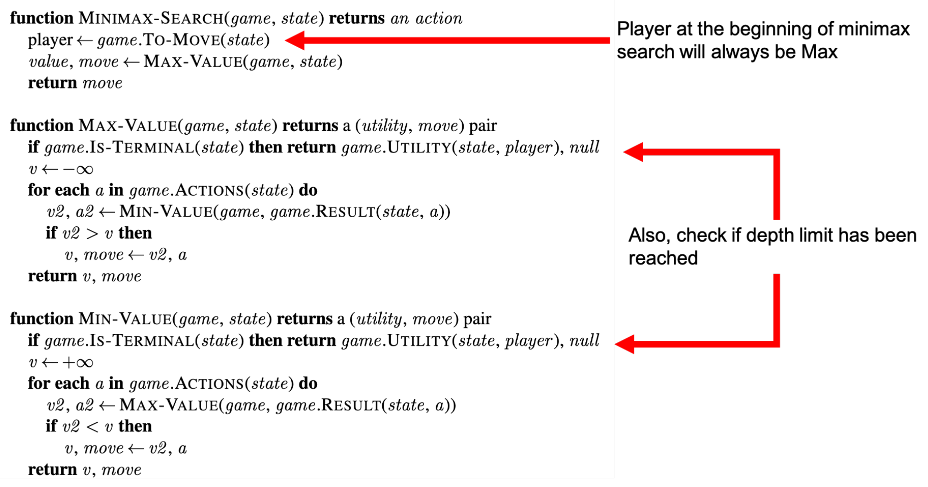
Those who have a heuristic function that is as good as, or better than, a secret heuristic function which will not be described until after the due date, will receive 30pts extra credit on this assignment.

Partial extra credit will be given to those who create a heuristic function that is better than the default heuristic function.

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, then your agent should start playing noticeably better. However, it will also take longer to make a decision.

What to Turn In

Turn in your implementation of `proj_code/proj2.py` to Blackboard.

You can add helper functions to `proj_code/proj2.py`, if need be.