CPSC 481 - Artificial Intelligence

Project 1, Fall 2022 - due 1st October 2022

In this project you will implement a game of Tic Tac Toe using Minimax (Alpha-Beta Pruning is optional).

There are two goals for this project

1. Exploring the concept of search problems with adversaries.
2. Learning the basics of python programming language.

Teams

The project must be completed with a team of three to four students. You are free to make your own team.
Teams should be formed by Thursday, 22th September 2022.

In order to submit the project as a team, you must join a group for that project in canvas.

- Several groups have been pre-created by the instructor, and your team must join one of them for your submission. Projects cannot be submitted by groups that you create yourself.
- The first student to join the group would automatically become the leader.
- Groups on canvas are specific to each Project, so you need to join a group for each project separately.
- If there are no empty groups on canvas, contact the instructor via email.

Getting started

- The starter code is available under Assignments-> Projects-> Project 1. Download this from canvas.
- This project uses “**pygame**”. Install pygame on your computer using the following command “pip install pygame”.

Understanding the Project

There are two main files in this project: “runner.py” and “tictactoe.py”. “tictactoe.py” contains all of the logic for playing the game, and for making optimal moves. “runner.py” has been implemented for you, and contains all of the code to run the graphical interface for the game. Once you’ve completed all the required functions in “tictactoe.py” you should be able to run “python runner.py” to play against your AI!

Let’s see “tictactoe.py” to get an understanding of what’s provided. First, we define three variables: X, O and EMPTY to represent possible moves of the board.

The function “initial_state” returns the starting state of the board. For this problem, I’ve chosen to represent the board as a list of three lists (representing the three rows of the board), where each internal list contains three values that are either X, O or EMPTY. What follows are functions that are left for you to implement!

Requirements

Complete the implementations of player, actions, result, winner, terminal, score and minimax.

- The **player** function should take a “board” state as input, and return which player’s turn it is (either X or O)
    - In the initial game state, `X` gets the first move. Subsequently, the player alternates with each additional move.
    - Any return value is acceptable if a terminal board is provided as input (i.e., the game is already over).
- The **actions** function should return a set of all of the possible actions that can be taken on a given board.
    - Each action should be represented as a tuple (*i, j*) where ‘i’ corresponds to the row of the move (0,1 or 2) and ‘j’ corresponds to which cell in the row corresponds to the move (also 0, 1 or 2).
    - Possible moves are any cells on the board that do not already have an ‘X’ or an ‘O’ in them.
    - Any return value is acceptable if a terminal board is provided as input.

- The **result** function takes a “board” and an “action” as input, and should return a new board state, without modifying the original board.
    - If action is not a valid action for the board, your program should raise an exception.
    - The returned board state should be the board that would result from taking the original input board, and letting the player whose turn it is make their move at the cell indicated by the input action.
    - Importantly, the original board should be left unmodified: since Minimax will ultimately require considering many different board states during its computation. This means that simply updating a cell in “board” itself is not a correct implementation of the “result” function. You’ll likely want to make a deep copy of the board first before making any changes.
- The **winner** function should accept a “board” as input, and return the winner of the board if there is one.
    - If the X player has won the game, your function should return `X`. If the O player has won the game, your function should return `O`.
    - One can win the game with three of their moves in a row horizontally, vertically, or diagonally.
    - You may assume that there will be at most one winner (that is, no board will ever have both players with three-in-a-row, since that would be an invalid board state).
    - If there is no winner of the game (either because the game is in progress, or because it ended in a tie), the function should return “None”.
- The **terminal** function should accept a “board” as input, and return a boolean value indicating whether the game is over.
    - If the game is over, either because someone has won the game or because all cells have been filled without anyone winning, the function should return “True”.
    - Otherwise, the function should return “False” if the game is still in progress.
- The **score** function should accept a terminal “board” as input and output the score of the board.
    - If X has won the game, the score is “1”. If O has won the game, the score is “-1”. If the game has ended in a tie, the score is “0”.
    - You may assume “score” will only be called on a “board” if “terminal(board)” is True.
- The **minimax** function should take a “board” as input, and return the optimal move for the player to move on that board.

    - The move returned should be the optimal action (*i, j*) that is one of the allowable actions on the board. If multiple moves are equally optimal, any of those moves is acceptable.
    - If the “board” is a terminal board, the “minimax” function should return “None”.

For all functions that accept a “board” as input, you may assume that it is a valid board (namely, that it is a list that contains three rows, each with three values of either `X`, `O`, or `EMPTY`). You should not modify the function declarations (the order or number of arguments to each function) provided.

Once all functions are implemented correctly, you should be able to run “python runner.py” and play against your AI. And, since Tic-Tac-Toe is a tie given optimal play by both sides, you should never be able to beat the AI (though if you don’t play optimally as well, it may beat you!)

Rubrics

1. Correctness - 20 points
2. Readability - 20 points
3. Documentation - 10 points

The individual contribution of each team member will be confidentially evaluated by the other team members. You will be evaluated individually on the following criteria

1. Discussion, Research and Problem Solving
2. Code Implementation
3. Checking in timely with other members
4. Willingness to help other team members

Submission

Only **ONE** submission is required per team. Do not submit your project via email. You must submit the projects through CANVAS.