Assignment # 1, Tic-Tac-Toe Implementation with Minimax Algorithm

19 Mar 2025, Total Marks: 10

Objective:
In this assignment, you will implement a Python-based Tic-Tac-Toe game where one player is a human, and the other is controlled by an AI. The AI should play optimally using the Minimax algorithm to make decisions.

Getting Started:

- Download the distribution code provided on GCR.
- Once in the directory for the project, run pip3 install -r requirements.txt to install the required Python package (pygame) for this project.

Understanding the Project:

This project has two main files: runner.py and tictactoe.py.

- **tictactoe.py** contains all the game logic, including how the game is played and how the AI makes its moves.
- **runner.py** has already been written for you and contains the code to run the game's graphical interface.

Once you finish implementing the required functions in tictactoe.py, you can run the game by using the command python runner.py and play against your AI!

In the tictactoe.py file, we start by defining three variables: X, O, and EMPTY, which represent the possible moves on the board.

The function initial_state will return the starting state of the game. For this project, we represent the board as a list containing three lists (each representing a row), where each inner list holds three values: X, O, or EMPTY.

The remaining functions are up to you to implement!

Specifications:

Complete the implementations of player, actions, result, winner, terminal, utility, 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 utility function should accept a terminal board as input and output the utility of the board.
  - If X has won the game, the utility is 1. If O has won the game, the utility is -1. If the game has ended in a tie, the utility is 0.
  - You may assume utility 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!)

Submission Instructions for the Assignment:

1. **Upload your Code on GitHub:**
   - Create a repository "Artificial Intelligence Assignments" on your GitHub account.
   - Upload all the code files for the Tic-Tac-Toe project to the repository.
   - Ensure your code is well-organized and all necessary files are included.
2. **Create a Zip File:**
   - Prepare a zip file that contains:
     - All the code files from your project (including both .py files, such as tictactoe.py, and any other necessary files).
     - A demo video showing the game running. The video should demonstrate you playing 2 to 3 games against the AI.

        - Record your screen while you interact with the game, ensuring the AI's responses are visible as well.
        - The video should clearly show that the AI is playing optimally and that the game is working as expected.

3. **Submit the Zip File:**
    - Upload the zip file containing the code and the demo video to the Google Classroom.
4. **Ensure Your Submission Includes:**
    - A link to your GitHub repository containing the project code.
    - A zip file with the project code and the demo video.

Optional

Implementing Alpha-Beta Pruning in the Minimax algorithm is optional but encouraged for extra efficiency.