Mini Project 1

Date Assigned: 09/15/2021 - Due Date: 10/12/2021 (11:59:00 PM)

	Full Name	Group No. & Group Representative	Banner ID
1-			
2-			

Please Follow the submission instructions. Points will be taken off if not followed properly.

- Type your full name and banner ID on the above table. Any Blackboard submission needs to have this page as its cover letter.
- All submissions must be made through the Blackboard. Submissions via Email are not accepted.
- This is a group project. You must collaborate with your groupmate(s). Each group must make **only one** submission **by the group representative**.
- Individual submission by a group member will lead to a **zero** for all the group members.
- You are not allowed to share your answers with other groups. Your code will be checked by plagiarism detection tools and any similarity score above **30%** will lead to a zero.
- Your group just needs to submit one **tictactoe.py** file to the blackboard. Do not compress the file. Just one python file is all each group needs to turn in.

Feel free to contact the instructor or the teaching assistant for any help. Our contact information can be found in the course syllabus on the blackboard.

Before you start:

1- Make sure you have **Python** installed. You need to use Python 3 for this project. Visit https://www.python.org/downloads/ for more information on how to download and install python.

2- Make sure you have **pip** installed. Visit https://pip.pypa.io/en/stable/quickstart/ for further information on how to install or use pip.

3- For this project, you need **pygame** module installed. You can find useful instructions on how to install pygame by visiting:
https://www.pygame.org/wiki/GettingStarted#Pygame%20Installation

Project files:

This project consists of two **python** files (i.e. runner.py and tictactoe.py) and one **tiff** file (i.e. OpenSans-Regular.ttf).

Do not manipulate runner.py and OpenSans-Regular.ttf. You just need to implement the methods/functions in the tictactoe.py file and upload it to the Blackboard.

While implementing the functions in tictactoe.py, make sure that you do not change functions' signature (I.e. functions' name and their arguments). Otherwise, your code does not produce the expected results properly and your group will miss points on that.

You are **not allowed** to add extra methods or functions to the code. Just implement the existing ones.

Grading:

You need to implement an agent which plays tictactoe using minimax algorithm. Grading criteria include:

1- Your code should run without raising any error(s).

2- Your code should be able to play the game as an intelligent agent.

3- Your code needs to be properly documented. Use brief comments to explain how different parts of your code work. Comments help the TA (Teaching Assistant) give you partial credit when your code does not work properly.

4- There are 8 different functions in tictactoe.py. Every function needs to be implemented separately according to the following explanations.

Project Explanation:

In tictactoe.py we have three variables: **X**, **O**, and **EMPTY**, to represent possible moves of the board.

The function **initial_state** returns the starting state of the board. The board is presented 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.

There are 7 more functions named **player**, **actions**, **result**, **winner**, **terminal**, **utility**, and **minimax**. You need to implement them all. In what follows, each function has been explained briefly.

- 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 (i.e. the goal) 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 an 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 do not play optimally as well, it may beat you!)