Artificial Intelligence and Machine Learning Fundamentals

Activity 4: Connect Four
In this section, we will practice using the **EasyAI** library and develop a heuristic. We will be using the game Connect 4 for this. The game board is seven cells wide and seven cells high. When you make a move, you can only select the column in which you drop your token. Then, gravity pulls the token down to the lowest possible empty cell. Your objective is to connect four of your own tokens horizontally, vertically, or diagonally, before your opponent does, or you run out of empty spaces.
The rules of the game can be found at https://en.wikipedia.org/wiki/Connect_Four.
We can leave a few functions from the definition intact. We have to implement the following methods:

- **__init__**
- **possible_moves**
- **make_move**
- **unmake_move (optional)**
- **lose**
- **show**

1. We will reuse the basic scoring function from Tic-Tac-Toe. Once you test out the game, you will see that the game is not unbeatable, but plays surprisingly well, even though we are only using basic heuristics.
2. Then, let's write the **init** method. We will define the board as a one-dimensional list, like the Tic-Tac-Toe example. We could use a two-dimensional list too, but modeling will not get much easier or harder. We will generate all of the possible winning combinations in the game and save them for future use.
3. Let's handle the moves. The possible moves function is a simple enumeration. Notice that we are using column indices from 1 to 7 in the move names, because it is more convenient to start column indexing with 1 in the human player interface than with zero. For each column, we check whether there is an unoccupied field. If there is one, we will make the column a possible move.
4. Making a move is similar to the possible moves function. We check the column of the move and find the first empty cell, starting from the bottom. Once we find it, we occupy it. You can also read the implementation of the dual of the **make_move** function: **unmake_move**. In the **unmake_move** function, we check the column from top to bottom, and we remove the move at the first non-empty cell. Notice that we rely on the internal representation of **easyAi** so that it does not undo moves that it has not made. If we didn't, this function would remove one of the other player's tokens without checking whose token was removed.
5. As we already have the tuples that we have to check, we can mostly reuse the lose function from the Tic-Tac-Toe example.
6. Our last task is to implement the show method that prints the board. We will reuse the Tic-Tac-Toe implementation and just change the variables.
7. Now that all of the functions are complete, you can try out the example.

Feel free to play a round or two against the opponent. You can see that the opponent is not perfect, but it plays reasonably well. If you have a strong computer, you can increase the parameter of the Negamax algorithm. I encourage you to come up with a better heuristic.