Artificial Intelligence and Machine Learning Fundamentals

Activity 1: Generating All Possible Sequences of Steps in a Tic-Tac-Toe Game

This activity will explore the combinatoric explosion that is possible when two players play randomly. We will be using a program, building on the previous results, that generates all possible sequences of moves between a computer player and a human player. Assume that the human player may make any possible move. In this example, given that the computer player is playing randomly, we will examine the wins, losses, and draws belonging to two randomly playing players:

1. Create a function that maps the **all_moves_from_board** function on each element of a list of board spaces/squares. This way, we will have all of the nodes of a decision tree.
2. The decision tree starts with **[EMPTY_SIGN * 9]**, and expands after each move. Let's create a **filter_wins** function that takes finished games out of the list of moves and appends them in an array containing the board states won by the AI player and the opponent player.
3. Then, with a **count_possibilities** function that prints the number of decision tree leaves that ended with a draw, were won by the first player, and were won by the second player.
4. We have up to 9 steps in each state. In the 0th, 2nd, 4th, 6th, and 8th iteration, the AI player moves. In all other iterations, the opponent moves. We create all possible moves in all steps and take out finished games from the move list.
5. Then, execute the number of possibilities to experience the combinatoric explosion.

As you can see, the tree of board states consists of 266,073 leaves. The **count_ possibilities** function essentially implements a BFS algorithm to traverse all the possible states of the game. Notice that we count these states multiple times because placing an X in the top-right corner on step 1 and placing an X in the top-left corner on step 3 leads to similar possible states as starting with the top-left corner and then placing an X in the top-right corner. If we implemented the detection of duplicate states, we would have to check fewer nodes. However, at this stage, due to the limited depth of the game, we'll omit this step.
A **decision tree** is very similar to the data structure examined by **count_ possibilities**. In a decision tree, we explore the utility of each move by investigating all possible future steps up to a certain extent. In our example, we could calculate the utility of the first moves by observing the number of wins and losses after fixing the first few moves.

Note: The root of the tree is the initial state. An internal state of the tree is a state in which a game has not been ended and moves are possible. A leaf of the tree contains a state where a game has ended.