# Artificial Intelligence and Machine Learning Fundamentals

**Solution 1:** Activity 1: Generating All Possible Sequences of Steps in the tic-tac-toe Game

This section will explore the combinatoric explosion possible when two players play randomly. We will be using a program, building on the previous results that generate all possible sequences of moves between a computer player and a human player. Determine the number of different wins, losses, and draws in terms of action sequences. 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 boards. This way, we will have all of the nodes of a decision tree in each depth:

```
def all_moves_from_board(board, sign):
move_list = []
for i, v in enumerate(board):
if v == EMPTY_SIGN:
move_list.append(board[:i] + sign + board[i+1:])
return move_list
```

2. The decision tree starts with [ EMPTY_SIGN * 9 ], and expands after each move: all_moves_from_board_list( [ EMPTY_SIGN * 9 ], AI_SIGN )
3. The output is as follows:

```
['X........',
'.X.......',
'..X......',
'...X.....',
'....X....',
'.....X...',
'......X..',
'.......X.',
'........X']
['XO.......',
'X.O......',
'X..O.....',
'X...O....',
'X....O...',
'X.....O..',
'X......O.',
.
.
.
.
'......OX.',
'.......XO',
'O.......X',
'.O......X',
```

```
'..O.....X',
'...O....X',
'....O...X',
'.....O..X',
'......O.X',
'.......OX']
```

4. Let's create a filter_wins function that takes the ended games out from the list of moves and appends them in an array containing the board states won by the AI player and the opponent player:

```
def filter_wins(move_list, ai_wins, opponent_wins):
for board in move_list:
won_by = game_won_by(board)
if won_by == AI_SIGN:
ai_wins.append(board)
move_list.remove(board)
elif won_by == OPPONENT_SIGN:
opponent_wins.append(board)
move_list.remove(board)
```

5. In this function, the three lists can be considered as reference types. This means that the function does not return a value, instead but it manipulating these three lists without returning them.
6. Let's finish this section. Then with a count_possibilities function that prints the number of decision tree leaves that ended with a draw, won by the first player, and won by the second player:

```
def count_possibilities():
board = EMPTY_SIGN * 9
move_list = [board]
ai_wins = []
opponent_wins = []
for i in range(9):
print('step ' + str(i) + '. Moves: ' + \
str(len(move_list)))
sign = AI_SIGN if i % 2 == 0 else OPPONENT_SIGN
move_list = all_moves_from_board_list(move_list,
sign)
filter_wins(move_list, ai_wins, opponent_wins)
print('First player wins: ' + str(len(ai_wins)))
print('Second player wins: ' +
str(len(opponent_wins)))
print('Draw', str(len(move_list)))
print('Total', str(len(ai_wins) + len(opponent_wins)
+ \
len(move_list)))
```

7. 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 the ended games from the move list.
8. Then execute the number of possibilities to experience the combinatoric explosion.

```
count_possibilities()
```

9. The output is as follows:

```
step 0. Moves: 1
```

```
step 1. Moves: 9
step 2. Moves: 72
step 3. Moves: 504
step 4. Moves: 3024
step 5. Moves: 13680
step 6. Moves: 49402
step 7. Moves: 111109
step 8. Moves: 156775
First player wins: 106279
Second player wins: 68644
Draw 91150
Total 266073
```

As you can see, the tree of board states consists of 266,073 leaves. The count_possibilities function essentially implements a breadth first search algorithm to traverse all the possible states of the game. Notice that we do count these states multiple times, because placing an X on the top-right corner on step 1 and placing an X on the top-left corner on step 3 leads to similar possible states as starting with the top-left corner and then placing an X on the top-right corner. If we implemented a detection of duplicate states, we would have to check less nodes. However, at this stage, due to the limited depth of the game, we omit this step.