Tic-tac-toe game RL Agent

Problem Statement

Note: Please go through the inventory management problem (MDP and the Q-learning code) carefully to solve the assignment.

One of the most popular and enduring games of all time is **Tic-Tac-Toe**. Because of its familiarity, this game is often used as a starting example to mathematically analyze a decision-making process. Its brevity makes it a perfect game to illustrate the rewards of thinking ahead and learning the consequence of each decision.

There are many variants of Tic-Tac-Toe. The most classic one is of X's and O's, where each player aims to place three of their marks in a horizontal, vertical, or diagonal row in a 3x3 grid.

The other popular variant of this game is **Numerical Tic-Tac-Toe.** Instead of X's and O's, the numbers 1 to 9 are used. In the 3x3 grid, numbers 1 to 9 are filled, with one number in each cell. The first player plays with the odd numbers, the second player plays with the even numbers, i.e. player 1 can enter only an odd number in the cell while player 2 can enter an even number in one of the remaining cells. Each number can be used exactly once in the entire grid. The player who puts down 15 points in a line - (column, row or a diagonal) wins the game.

It is recommended that you play the game here for more clarity.

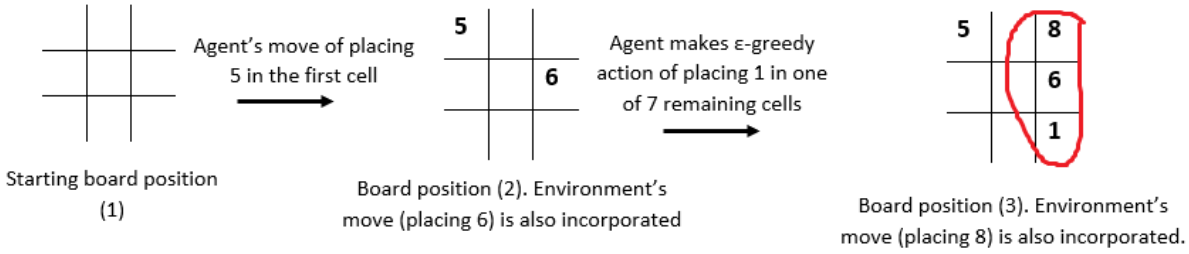
Rules of the Game:

1. The game will be played on a 3x3 grid (9 cells) using numbers from 1 to 9. Each number can be used exactly once in the entire grid.
2. There are two players: one is the Reinforcement Learning (RL) agent and other is the environment.
3. The RL agent is given odd numbers {1, 3, 5, 7, 9} and the environment is given the even numbers {2, 4, 6, 8}

4. Each of them takes a turn. The player with odd numbers always goes first.
5. At each round, a player puts one unused number on a blank spot.
6. The objective is to make 15 points in a row, column or a diagonal. The player can use the opponent's numbers in the grid to make 15.
7. The game terminates when any one of the players makes 15.

In this assignment, you need to build an RL agent that learns to play Numerical Tic-Tac-Toe with **odd numbers** (the agent will always make the first move). You need to train your agent using **Q-Learning**. The environment is playing randomly with the agent, i.e. its strategy is to put an even number randomly in an empty cell. If your agent wins the game, it gets 10 points, if the environment wins, the agent loses 10 points. And if the game ends in a draw, it gets 0. Also, you want the agent to win in as few moves as possible, so for each move, it gets a -1 point.

Following is a sample episode for your reference:



In this episode, the **environment wins** as it is able to make 15 first (8+6+1). After the agent places 1 in one of the grids, the environment rewards it (with a negative reward of -1) and makes a next move of placing 8 in one of the remaining cells.

Goals

You are given two files: 'TCGame_Env.py' and 'TicTacToe_Agent.ipynb'. The first one is the environment file and the second one is the agent file. You use the environment that is created in TCGame_Env.py to write the learning algorithm.

You need to accomplish following in this assignment:

1. Create an **MDP for Numerical Tic-Tac-Toe** game. The basic framework for this is:
    1. Initialise the state
    2. Define the action space for each state. (Be careful in defining actions. The actions are not the same for each state)
    3. Define the winning states: the sum of three numbers in a row, column or diagonal is 15.
    4. Define the terminal states (win,tie,loss)
    5. Build the reward structure as below:
        - +10 if the agent wins (makes 15 points first)
        - -10 if the environment wins
        - 0 if the game ends in a draw (no one is able to make 15 and the board is filled up)
        - -1 for each move agent takes
    6. Define a step function which takes in an input of the agent's action and state; and outputs the next state and reward. (Make sure you incorporate environment's move in the next state).
    7. **For your reference: TCGame_Env.py file is provided with a basic structure of the code.** The functions (and the comments) will provide an intuition of how the MDP would be formulated. Codes of a few functions are also provided to give you more sense on how to proceed with the MDP. **Note:** Using this framework is not compulsory, you can create your own framework and functions as well.
2. Build an **agent that learns the game by Q-Learning**. You can choose the hyperparameters (epsilon (decay rate), learning-rate, discount factor) of your choice. For that, you can train the model iteratively to obtain a good combination of hyperparameters. You won't be evaluated on your choice of the hyperparameters. You need to submit only the final model.
    - While **updating the Q-values**, if the next state is a terminal state, then the Q-values from that state are 0. (No action is possible from that state)
    - For a 64-bit system with 8GB RAM, it takes ~30 minutes to run 5Mn episodes.
3. **Q-values convergence-** check whether Q-values learnt by the agent have converged or not. Sample 4 state-action pairs and plot it with the number of episodes to understand the convergence.