CS747-Assignment 2
MDP Planning and Anti Tic Tac Toe

Task 1:

In Task1, we are supposed to implement the 3 algorithms to determine the value and optimal policy actions for the states.

Here, I have created a class **MDP** which takes in a path of the *mdp* file to initialise the MDP. There is another helper method runAlgo, which takes certain algorithms to run the algorithm required to find the required values.

Three functions are:

- `valueIteration()` - for the value iteration algorithm
- `howardPI()` - for the Howard Iteration algorithm
- `linearProgramming()` - for the Linear Programming algorithm

Note that, to tackle the issues of large data because of the large number of states and actions, instead of creating a 3-dimensional matrix of size S x A x S, I have created a 2-Dimensional matrix of size S x A, where each element of the matrix is a list of tuples, where the tuples are (s', probability, reward).

This helped me to work around a large amount of data easily without unnecessary computations.

Important points to note about the implementation:

1. I have ensured that every terminal state achieves 0 Value by assigning 0 value at the end.
2. The precision for the value iteration algorithm is taken care of till the 9th decimal value for absolute correctness.
3. As the algorithms use the np.argmax function which resolves ties by returning the lowest index with max value, to ensure that the Policy values for any state aren't the actions that actually have 0 probability, I assigned a very high negative value for the Q[s][a] of those pair of state and action.

Observations:

- The value iteration algorithm takes the highest amount of time among the 3 algorithms. This is certainly because the other algorithms make use of the available inbuilt functions to solve certain mathematical equations and also because we have ensured that the precision is kept very high.
- In general, the three algorithms work pretty fast and even the computation for 2500 states and 30-35 actions were completed in under 40 seconds.

Task 2:

Task 2 involved the understanding of the use of MDP and the state transitions, and realise the importance of correct transitions and identifying the correct terminal state.

Encoder File:

- The file takes in the policy file of the opponent and the valid states of the player as inputs from command line. To ensure the order of the states, I have created a map, also called dictionary in python, to create a mapping between the state, that is a string and the index.
- The indices have been assigned taking the order of the input file into consideration.
- Further, the file contains two auxiliary functions.
    - isTerminal(): This function returns whether the given state is a terminal state or not. If it is a terminal state, then who has won the game or has the game ended in a draw.
    - nextState(): This function takes the current state and a certain action and returns the next state if the action is a valid move on the current state.
- The above functions were useful while calculating the transition probabilities and the rewards.
- For the sake of simplicity and also intuitively, the terminal states have been combined as a single state. The only difference being the reward which is 1 if the player wins and 0 if the player loses or the game ends in a draw.

Please refer to the code for further details.

Decoder File:

- The code takes the value and policy file generated by running the planner over the output file of the encoder. The code also takes the states file and the player ID of the player.
- The purpose of the decoder is simply to assign the probability 1 to the action which is given out by the planner in the value-policy file.

Task 3:

The policies converge very soon with the default algorithm of the planner which is probably because of the high value precision of the Howard Algorithm.