CS 747, IIT Bombay
Assignment #2

Prof. Shivaram Kalyanakrishnan

Vibhav Aggarwal (190050128)

Task 1

I have used a list based representation to store the transitions, instead of the matrix to save memory and reduce computation time. Here are some of the implementation details for the different algorithms:

- **Value Iteration:** The value function is initialized to 0 for all states. While computing `Q` value for some state `s` and action `a`, if it so happens that this state-action pair is not present in the MDP file for some reason (which means that the MDP is incomplete), then its `Q` value is assigned to $-\infty$ to ensure that this action is never taken. For terminal states, the action `0` is always chosen.
- **Howard PI:** The initial policy is set in the following manner: each non-terminal state `s` is assigned to any action `a` such that there is at least one transition from `s` to `a` in the MDP file. For terminal states, the action is always set to `0`.
  For evaluating the value function for a policy, I have used matrix multiplication to simultaneously solve a linear system of n equations (where n is the number of states).
  In case of multiple improving actions, I am taking the first one.
- **Linear Programming:** The set of `nk` constraints is added to the solver as discussed in class. Additionally, for each terminal state `s`, a constraint `V[s] == 0` is also added to ensure that the terminal states take 0 value.

Task 2

Formulation of MDP:

Let's say that we are generating an MDP for player 1 using some policy of player 2.

- **States:** If player 1's state file has n states, then the MDP has $n+1$ states (indexed from 0) - the first n states correspond to the given states and the last state is a terminal state.
- **Actions:** There are 9 actions - 0 to 8 - one for each cell.
- **Transition probability and Reward:** If the player attempts to take an action in a cell which is not 0, then the resulting state is the same state with probability 1 and reward -10. This ensures that a player never takes an illegal move in an optimal policy.
  For a valid action, if the game draws or player 1 loses, then the resulting state is n with probability 1 and reward 0.
  For all other valid actions, the next state is decided on the basis of player 2's policy file. So, in this case, we can have multiple next states with probabilities as specified in the file. The reward is 0, except in the case when player 2 loses after taking some action, where the reward is assigned to 1.
- **MDP type:** It's episodic since the game always ends after a finite number of moves.
- **Discount factor:** It is set to 1 since we are interested in winning the game ultimately.

Task 3

Running the code:

Simply run the code `python3 task3.py` while keeping `task3.py` in the same folder where `data` folder is present. This will create a folder with the name `policies` with files having names of the format `opt_x_y.txt`. `opt_x_y.txt` denotes the optimal policy for player `x` against the opponent in `y`-th iteration. So, the converged policies will be named `opt_1_10.txt` and `opt_2_10.txt`. To change the initial policies, change the variables `P1_INITIAL_POLICY` and `P2_INITIAL_POLICY`. Exactly one of them must be set to `None`.

Observation:

I observed that on all the four initial policies, the policies converge such that player 2 wins on the converged policy. Also, the converged policies for all test runs were exactly same for both the players. This is the terminal output:

```
❯ python task3.py
Starting with a policy for P1 present at ./data/attt/policies/p1_policy2.txt

Iteration:  1
P1 policy differs at 1578 states
P2 policy differs at 451 states

Iteration:  2
P1 policy differs at 95 states
P2 policy differs at 16 states

Iteration:  3
P1 policy differs at 16 states
P2 policy differs at 0 states

Iteration:  4
P1 policy differs at 0 states
P2 policy differs at 0 states

Iteration:  5
P1 policy differs at 0 states
P2 policy differs at 0 states

Iteration:  6
P1 policy differs at 0 states
P2 policy differs at 0 states

Iteration:  7
P1 policy differs at 0 states
P2 policy differs at 0 states

Iteration:  8
P1 policy differs at 0 states
P2 policy differs at 0 states

Iteration:  9
P1 policy differs at 0 states
P2 policy differs at 0 states

Iteration:  10
P1 policy differs at 0 states
P2 policy differs at 0 states
```

Convergence:

The above observation leads one to believe that the policies do converge, irrespective of the starting policy used. However, it also depends on the implementation. For example, if ties were broken randomly, then the policies might not converge, otherwise, they must converge. This is because, in the converged policies which I obtained, player 2 won. This means, there exists at least one policy for player 2 which cannot lose. When player 2 finds one such policy, it does not change its policy further (if ties are not broken randomly) and consequently, player 1 does not change its policy either.