Sheel Shah, 19D070052, CS747 Assignment 2 report

Task 1:
Most of this has been implemented in mdp.py. The MDP class has Q_from_V, pi_from_Q, and V_from_pi functions that are used to convert from pi to V or the other way around. All these are implemented straightforwardly from the definitions taught.

For value_iteration, we apply the bellman optimality operator (in vectorized form) till consecutive V and V' have rms difference less than epsilon = 1e-10. There is also a cap on the number of iterations, just in case.

For howards_policy_iteration, we set pi_ = q_pi.argmax(axis=1). So if there are actions that have better Q value, we choose the one with the highest Q. Otherwise, the current action has the best Q, and hence argmax chooses that itself. We keep repeating until pi and pi_ are the same. Tie breaking is the default followed by numpy argmax, which is the action with the lowest index among those that are tied.

For linear_programming, I created S variables for each V(S_i), and then created constraints to be added to the problem iteratively.

Task 2:
I create two extra states. The loss/draw terminal state, and the win terminal state. These states are the largest two indices and are mentioned as end states (but not used, since planner doesn't really use the terminal states). The transitions are normally transcribed. For each state action pair, the following is done:

if action is illegal, create transition to loss-state with -100 reward.

else, the state that the opponent will see because of this state + action, is created. Now based on the opponent's policy: 1. If game continues, the transition is printed with corresponding probability. 2. If game ends, this actions probability is added to one of win_prob / draw_prob which accumulate the win and draw probabilities. After accounting for all actions of the opponent, the cumulative win and draw transitions are printed.

in case the state created for the opponent is not in its policy, this implies the game ended by state + action. This implies the game has ended in a loss/draw and hence the appropriate transition is printed.

Task 3:
Yes, the policy always converges. It converges to an "ultimate policy", which is one which will win in all situations where it is possible to guarantee a win against all opponent policies. This implies that player 2 always wins from the blank board state too, no matter how player 1 were to play. These policies are such that they complement each other. So an ultimate policy for p1 is an optimal policy when p2 follows its ultimate policy, and the other way round too.

Proof: Let us say that we start with p2's policy p20, where pij is policy of player i in step j. This gives us p11 and that gives us p21. We know that the Value functions of these players are complementary in a way. So if p1 has a high value for a state, p2 will have a low value for the state that follows from this state and the action that p1 chooses. V of p20 under p11 is lower than V of p21 under p11. This is because p21 is the optimal policy against p11 and hence has the highest V for each state. Now p12 has to be such that V of p12 under p21 is higher than that of p11 under p21 (by the same argument). Hence value of p21 under p12 is lower that its value under p11 (since values are complementary).
Therefore V_p20_p11 <= V_p21_p11 >= V_p21_p12.
Hence p2's policy's value first increases then decreases, and this keeps on happening, and can only stop when it becomes ultimate.
Similarly, V_p10_p21 <= V_p11_p21 >= V_p11_p22 implying V_p21_p10 >= V_p21_p11 <= V_p22_p11.
Therefore V_p20_p11 <= V_p22_p11 which means V of p2 must increase in net after the increase and decrease, and this guarantees convergence to the ultimate policy.