CS747 Assignment 2 report
Akshay Iyer- 190070006

Task 1-

Value iteration- I implemented value iteration using np arrays. To check for convergence I used the np.dot function. If (sum> max), I updated the value of the max index (sum here is the same sum in the value iteration method taught in class). Hence, in case two value functions for different actions are the same, the value of pi[i] is given by the **first action** for that state, which maximises our reward. (This is how I broke ties when multiple actions were optimal).

Howards policy iteration- I implemented this by first using a while loop to calculate the different values of Vpi till they converged. Then I used those values of V[pi] to calculate our current optimal state. Then I found the Qpi for a transition from our current state through that action.
I then found all the other Qpi's and compared them to the Qpi of our current optimal state. If we found any Qpi greater than our earlier optimum for Qpi, I updated the pi[s] for that state s and added a break function. Then I moved on and improved the value of pi[s] for the next state in line. I did this till no state had any improvable states left. Hence we got the optimum value of pi*[s] and V*(s) for all states s. Here I always choose to improve to the **first state** which has Qpi greater than that of our current optimal pi. (This is how I broke ties when multiple actions were better than the current action we take).

Linear programming- I used the pulp library for this part of the assignment. I fed all the linear programming equations and constraints to the solver, which returned the values of V*(s) appropriately.

Once I got the values of all my V*(s), I iterated through all my actions for every state, and found out which action's sum(of terms as done in class) over all s' gave us the maximum value for every state. Again, like in the vi case I used if (sum>max) as my condition. Hence we see that for multiple optimum actions giving the same value of Qpi, we take the **first such action** only as our optimum action (This is how I broke ties when multiple actions were optimal). We then assign the value pi[s]= that optimum action. Hence we get our V* and pi*.

I kept my default algorithm as value iteration. (Used if the argument is returned as none).

Observation- Linear programming is the fastest. Then value iteration is the second fastest, as there is only one convergence criterion to be met. Then Howards policy iteration is the slowest as it needs to achieve convergence to get Vpi first, and then continue till the action from every state is its optimum action to get our V* and pi*.

Task 2-

This was a very interesting assignment. First assign t and r = 0 for all s,a,s'. I also created a new state which is the terminal state indicating that the game has ended.

I iterated through all the states of the agent in its states file.

At every state of the agent, I iterated through the set of all valid actions (i.e. putting the number at places where there was a 0 in the state. If there was a non 0 value that move would have been invalid). Then for every valid action, I replaced the 0 at that place with the number of the agent. Lets call this new state ns (nextstate).

Note that ns depends ONLY on the action taken by the agent.

Then I check if this is a terminal state, i.e. one where either of the players has one, or there is a draw. I did this using my function checkin which returns the player who won, or a 0 if game goes on, or a 5 if the game is drawn.

If its a terminal state, it means that our agent could not have won. Because its either a draw, or the game ends if the person who just played gets 3 in a line and loses.
Hence we check if the opponent won, if he did, then we say that p(transition to terminal state) for that state and for that action is 1 (as ns only depends on action) . Similarly if there is a draw in that turn, t[s][a][terminal state]=1 and reward for transition remains 0. This is as no it does not matter what the opponent does, as long as the agent takes that action from its state, it will not win, and the game will end.

If ns is a non terminal state, we check the valid actions that opponent can take with ns. Thus only those actions which replace a location (with value 0 at that location previously). Thus after adding the opponent number to the location let us call it fs (finalstate).

Now we check if finalstate is a terminal state. If agent has won at finalstate, we assign
p(s, a, termini state)= (probability of opponent playing that action), and reward =1.

This is as ns decided only by action a of the agent. The transition probability **given an action played by agent**, is decided purely by the policy of the opponent.

If its a draw, we say p(s,a,terminal state)= (probability of opponent playing that action), but our reward still remains 0 (as agent has not won).

If the game has not ended even at fs, we say that p(s, a, s') = (probability of opponent playing that action). Where s' is our state fs. Our reward for that action and transition still remains 0 as agent has not won.

Thus we keep doing this for every state in the state file of the agent.

We did all this, as we needed a reward and probability that only depends on the initial state of the agent, action taken by agent and final state of the agent. Hence based on the action taken by the agent for a given initial state of the agent, we have to find the final state and probability and reward of that transition using out opponents policy. Hence we did all the above.

In decode.py, in some cases, where it was impossible for the agent to win, the maximum V* and pi* dictated an invalid action, i.e. one where there was no 0 to replace at that location. Hence in such a case I gave every valid action an equal probability, such that all their probabilities added to 0. Hence this was how I resolved this issue here.

Task 3-

The outputs for the 10 successive updated policies of player 1(p1) and player 2(p2), are in the folder additionalfiles.
Player 1's successive policies are name p1iter1, p1iter2….. p1iter10, with p1iter10 as the final converged policy of player 1.
Player 2's successive policies are name p2iter1, p2iter2….. p2iter10, with p2iter10 as the final converged policy of player 2.

The metric I used to check how much the successive updations the policy of the two players differ, and hence to check convergence,
is the norm of the difference of the 2D arrays created using the probabilities of the state transition in the respective policy files. I used np.linalg.norm function for this.
Thus a norm of 0 indicated that the policies have converged, and do not change over successive iterations. In every iteration based on the player who is playing, I update the policy files respectively.

I have stored these errors per iteration in a text file. The naming convention is as follows-
p1_2p2_1error.txt is the file with the initial policy of p1 as the original file p1_policy_2.txt given as part of the problem statement for this assignment. Thus as p1 has policy 2, p1_2. Similarly we take p2_policy_1.txt as the policy file for player 2, hence p2_1. Thus our text file name is p1_2p2_1error.txt. Similarly for the other files too.

p1_2p2_1policy2.txt is the file that stores the final converged policy of ***player 2*** with initial policy of player 1 as p1_policy_2.txt, and player 2 as p2_policy_1.txt. (same convention as before).

Similarly p1_2p2_1policy2.txt is the file that stores the final converged policy of ***player 1*** with initial policy of player 1 as p1_policy_2.txt, and player 2 as p2_policy_1.txt. (same convention as before).

Let us take an example of the error file p1_2p2_1error.txt.

agent 1 38.64265231056375 agent 2 45.79301256742124
agent 1 38.64265231056375 agent 2 52.3450093132096
agent 1 38.458635787904214 agent 2 52.3450093132096
agent 1 38.458635787904214 agent 2 28.394541729001368
agent 1 21.700559571455425 agent 2 28.394541729001368
agent 1 21.700559571455425 agent 2 11.26203060435077
agent 1 7.946247991527646 agent 2 11.26203060435077
agent 1 7.946247991527646 agent 2 2.449489742783178
agent 1 0.0 agent 2 2.449489742783178
agent 1 0.0 agent 2 0.0
agent 1 0.0 agent 2 0.0
agent 1 0.0 agent 2 0.0
agent 1 0.0 agent 2 0.0
agent 1 0.0 agent 2 0.0
agent 1 0.0 agent 2 0.0

Hence we see that the policies of both agent 1 and agent 2 converge till their difference both becomes 0 (i.e. the policies of both agents have completely converged). The errors afterward continuously stay 0 for both agents, which shows us that the policies have completely converged and do not change anymore.

Similarly I have generated this same error file, as described above for different initial policies of both the agents and observed, that they still converge to the same final policy.

Final policy of agent 1 is stored in policy1.txt. The policy of agent 1 right before that is policy1_1.txt, which we observe is the same as policy1.txt. Hence converged.

Similarly, we see that final policy of agent 2 is stored in policy2.txt. Second last policy of p2 is stored in policy2_1.txt.

Hence we have proved that, as our error goes to 0 after a particular iteration, and stays there, our policies do converge. And because this happened for all the 4 different initialisations that I took (can be checked in the text files I mentioned above), I infer that this happens for all initial policies of player 1 and 2.

Now, to explain this using theory-

Now, let us assume that player 1 is the agent first, and we have a policy for player 2.
Thus based on player 2's policy, we get an optimal policy for player 1.

Now player 2 becomes the agent, and player 1 has a fixed policy.

We have certain states in player 2, where even though player 1 was free to choose its policy in the earlier step, it had no chance of winning. These are typically the cases where the optimal pi of player 1 leads to an invalid choice of action. Thus these states have not changed in player 2, and player 2 will still always win using its original policy for those states. Hence those rows in the policy file will not change at all. These rows will not change in any successive policy of player 2 too, as it will always win or draw using that.

Now since player 2 has fixed a few of its states near the terminal state forever, player 1 sees those states near the terminal state as fixed. And against a fixed policy of an opponent, we get a fixed policy for the agent. Thus the states right before the fixed states of player 2, become fixed states in the policy of player 1 too.

Hence similarly these previous states in successive iterations keep fixing themselves permanently until we get a constant policy for both player 1 and player 2 in all successive iterations. Hence our algorithm is said to be converged at this point.

Now, to prove that this converged algorithm is the optimal algorithm-

In the beginning-
Player 1 only fixes those algorithms where the opponent cannot win, or where player 1 wins itself. What our V* and pi* do is they maximise the reward of our player, which in this case is player 1. Hence it only fixes those cases where there is surely a win for player 1, maximising its reward, or where there is surely a draw, i.e. no matter what player 1 does it cannot win and the game ends on the next step.

Then player 2s policy maximises reward by fixing states in which player 2 can win, or there is surely a draw. This is only for transitions to fixed states of player 1, hence it will fix those states.

Then player 1 similarly optimises only those moves where it surely wins or draws, before the next fixed state it sees. Thus both keep optimising themself, until they reach an impasse.
At this point both agents have fixed all their states to achieve an optimal policy and changing any one of the states and actions would reduce their reward. Hence their policy is optimum and converged.

Note: This only works as the fixed states probabilities never change once they become fixed. This is as our algorithm assigns fixed probabilities for every transition. If our algorithm chose between multiple probability distributions for the actions whenever a state is encountered, none of our states would ever get fixed. Hence we would never reach convergence.