Implementation of Algorithms:

General Details:

Created a class “MDPSolver” and defined various fields to store helpful information such as the number of States, number of Actions, end states, discount factor and transitions.

Maintained an adjacency list-like data structure to store transitions. Every state had a list of transitions corresponding to every action at its index in the adjacency list for faster access. Different functions (corresponding to different algorithms) are called to solve the MDP as required(supplied as an argument). The default algorithm used is Linear Programming(“lp”). Values of terminal states are set as 0 and never get updated as they have no entry in the adjacency list.

Value Iteration:

Value functions corresponding to states are set to 0 initially. All states are iterated over, and if the value function corresponding to a state increases for a particular action, the value function and policy are updated accordingly, and the updates keep happening until values don't increase by margins above the threshold limit of 10^{-10}.

Howard’s Policy Iteration:

We initialize and start with a random policy and a value function corresponding to that policy. We iterate over all actions, and if we find an improving action for the state, we update the action for that state and value function for the next iteration (two separate value functions are maintained, one corresponding to the value function before the iteration and one after it). We find improving actions for all states in every iteration until no more improving states exist.

Linear Programming:

Value functions for all states are computed by formulating linear constraints as covered in class and feeding the objective function to the PuLP solver. Then Q-value is calculated from every state for every action to find the most optimal action(max Q-value for that action) and, hence, compute the optimal policy.

Policy Evaluation:

The value function for a particular policy is computed by solving the Bellman equations with an iterative approach.

The results obtained from all three algorithms are almost equal(to the precision of 10^{-10}). Linear programming works best for higher discount factors (above 0.98) beating value iteration and policy iteration ,who find convergence difficult because of the high amount of time spent on exploration.

Football Encoder:

An Encoder class is defined with fields ‘p’ and ‘q’ and a dictionary and a list to maintain names and order of states. 8194 states are defined in total. 8192 states represent the position of player1, the position of player 2, the position of the opponent and an indicator whether possession is with player 1 or 2. Two extra states are defined to represent a goal state(terminal state when the goal is scored) and a failure state(terminal state when possession is lost). Number of actions are 10 (4 movement actions for player 1 and 4 for player 2, 1 for shooting and 1 for passing). The success state is labeled 0 and the failure state 1.

For generating transitions from a given state, first, the position of the defender is updated according to probabilities presented in the opponent policy file and thereafter, the action is performed and the probability calculated for the same. 10 different action functions are defined , each performing one of the actions. For passing, the co-linearity conditions are checked whether it affects the pass, and for shooting, the opponent's presence in front of the goal is checked.

Since the value function to be reported is the number of expected goals from each state, the rewards for all transitions except the ones that lead to the success state(scoring of a goal) are set to 0 and the reward for transition to the success state is set to 1. The discount factor is also set to 1, as it’s an episodic task and we don’t want to discount expected goals from a future state.

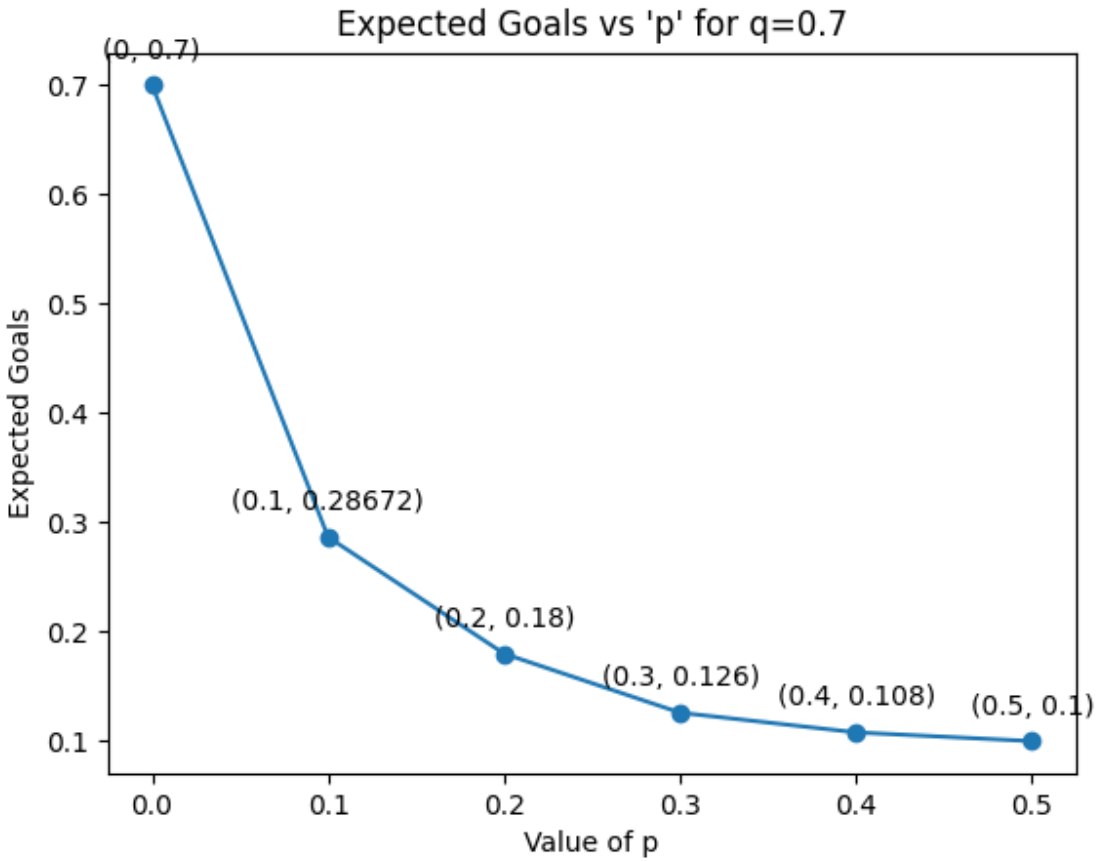

Fig.1 Expected Goals vs ‘p’

With increasing value of ‘p’, the chances of successful movement starts reducing, hence, shots have to be taken earlier(even if from far-off positions from the goal).

For p=0,0.1 and 0.2 the optimal action obtained is ‘1’ that is player1 with possession moves to the right(and distracts the defender out of the blocks in front of goal due to greedy defense policy) ,moves right twice again and finally shoots from one block away from goal. The number of expected goals has reduced with increasing ‘p’ because probability of successful movement has reduced.

For p=0.3 and 0.4, optimal action is ‘5’, player 2 moves to the right and the ball is passed to player 2 before shooting. Here player 2 is moved because movement without ball still has a higher chance of success as compared to action ‘1’.

For p=0.5, the optimal action is ‘9’ as ‘p’ is large enough to not waste turn on moving, which has a probability of failure of 0.5 for player without possession and failure probability 1 for player with possession.

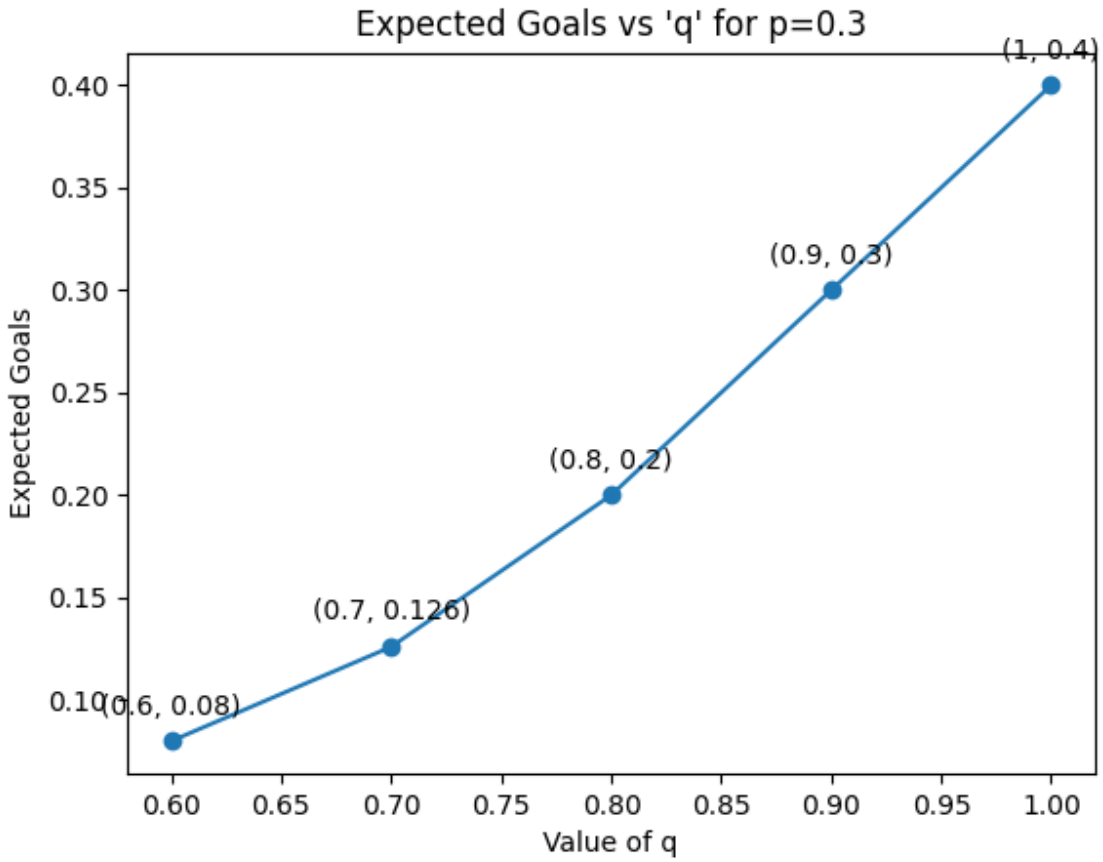

Fig.2 Expected Goals vs ‘q’

With increasing values of q,the scope for shooting from far blocks improves and hence shooting is preferred instead of risking failure with movement.
For q=0.6, the optimal policy is ‘1’, where player 1 moves 3 blocks right before shooting from right in front of goal.
For q=0.7, the optimal policy is ‘5’ where player 2 moves 1 block right, the ball is passed onto him and he shoots from 2 blocks out.
For q=0.8,0.9 and 1, the optimal policy is ‘9’ , where in q is large enough to take a shot from 3 blocks away from the goal, and the expected goal value is equal to ‘q’ itself.