Programming Assignment 2
Report

Kanak Yadav, 20D070044

October 15, 2023

1 Task 1

1.1 MDP class

In the file planner.py, I have created an MDP class to facilitate the MDP planning procedure. It has the following variables: S, A, end, T, TR, mdptype, and discount.

S, A, end, mdptye, and store the number of states, actions, the end states, the type of MDP, and the discount factor γ respectively.

The state and actions are represented by whole numbers less than S and A respectively. S and A may be used to represent the set of these states and actions respectively as well.

T is a nested list with dimensions S x A, and every element is a list of tuples containing the possible final state and transition probability values.

TR is a 2d array of shape S, A containing the summation of the product of transition probability and reward, for a given state, action pair across all possible reachable states (or all state but the transition probabilities would be zero), i.e.,

$$TR[s,a] = \sum_{s' \in S} T(s,a,s')R(s,a,s')$$

The variables are assigned while reading the MDP file.

1.2 Value Iteration

The Value Iteration algorithm is the successive application of the Bellman Optimality Operator:

$$B^*(X)(s) = \max_{a \in A} \left(\sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma X(s')] \right)$$

For implementation consider

$$temp[s,a] = \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma X(s')]$$

$$= \sum_{s' \in S} T(s,a,s')R(s,a,s') + \gamma \sum_{s' \in S} T(s,a,s')X(s')$$

$$temp[s,a] = TR[s,a] + \gamma \sum_{s' \in S} T(s,a,s')X(s')$$

Now, V, and π (initialized to zeros) and updated as:

$$\pi(s) = \arg\max_{a \in A}(temp[s,a])$$

$$V(s) = temp[s, \pi(s)]$$

When every element of the updated V(s), gets close (based on a tolerance) to the old V(s), the iteration stops and the algorithm returns both V and π

1.2.1 Observations

The algorithm takes longer to run when the tolerance is low and also particularly for the test case where the MDP is episodic and the discount factor is 1, so the value function ends up having elements greater than 500.

1.3 Howard's Policy Iteration

Howard's Policy Iteration iterates over the policy based on the action value function:

$$Q[s,a] = \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V(s')]$$

for a fixed V (in other words a fixed policy π). The calculation is implemented in a similar fashion as above.

For implementation, π is initialized to 0 and updated as:

$$\pi = \arg\max_{a \in A}(Q[s,a])$$

This update step updates all the improbable states to the actions with the maximum Q value and all the non-improvable states remain the same.

The value function is computed using successive applications of the Bellman Operator for the fixed policy π:

$$B^{\pi}(X)(s) = \sum_{s' \in S} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma X(s')]$$

Again, the summation is implemented using the TR array as shown above, and a tolerance is used for the stopping criterion.

The stopping criterion for the algorithm is when there is no change in π.

1.3.1 Observations

The algorithm takes slightly longer to run than value iteration on my machine.

1.4 Linear Programming

The Linear Programming problem formulation used is the same as done in the class:

$$\begin{aligned} &\text{maximize} && -\left(\sum_{s\in S} V(s)\right) \\ &\text{subject to} && V(s) \geq \sum_{s'\in S} T(s,a,s')[R(s,a,s') + \gamma V(s')], \forall s \in S, a \in A \end{aligned}$$

The LP has S variables and S*A constraints with V* being the sole optimizer (i.e., unique solution).

The LP is solved using the PuLP library. To form the constraints I have used LpAffineExpression, which takes an iterable with LpVariables and their coefficients as pairs to return an expression.

After obtaining V* from the LP solver, π^* is computed as:

$$\pi^*(s) = \arg\max_{a\in A} \left(\sum_{s'\in S} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]\right)$$

1.4.1 Observations

While this algorithm was the fastest for the test cases in task-1, it was significantly slower than the other algorithms for task-2 on my machine. Hence, I have set the default algorithm for the planner to be Value Iteration.

1.5 Policy Evaluation

Policy evaluation (Value function computation) is done by the successive application of the Bellman Operator (as described above) for the policy described in the file.

2 Task 2

2.1 MDP Formulation

2.1.1 States

The state string containing the positions of the players, the defender, and the possession of the ball are mapped to the state numbers 1 to 8192 as:

$$\text{state index} = 2(256(\text{b1} - 1) + 16(\text{b2} - 1) + \text{r} - 1) + \text{p}$$

where b1, b2, and r are the positions of the players and the defender respectively (each from 1 to 16), and p is the possession indicator (1 or 2).

There are two end states: 0 for the end state where a goal is made and the reward is 1 and 8193 for all other end states with reward 0.

Hence, there are a total of 8194 states in the MDP. We need two end states because otherwise, there would be a situation where for that same s, a, s' combination, there would be two values for the reward and also the transition probability.

2.1.2 Actions

Actions are taken to be the same as given in the problem statement.

2.1.3 Discount Factor

The discount factor γ is taken to be 1 because the MDP is episodic, and we want the value function to represent the expected number of goals scored, which is the case only if we take γ to be 1.

2.1.4 Reward Function

Only the shooting action may result in a reward of 1, i.e. when we transition to the state 0 (a successful goal), and for all other transitions, the reward is 0.

2.2 Transition Function

I have defined separate functions for printing the transitions as required by the planner Python script. These functions are called after reading and decoding each line of the opponent's policy file.

2.2.1 Movement

We iterate over the action 0 to 7, find the new x, and y coordinates of the player to be moved, and check if they are inside the playing field or not. If they are not, we print a transition to the end state with a reward of 0.

If they are inside the field, then we check if the player moving is the one having the possession.

If they have possession then we check for the tackling situation.

The probability of success of the move is calculated for every possible movement of R separately, while the probability of failure of the move needs to be accumulated over the possible movements of R because the next state in the failure to move would be the same (8193).

2.2.2 Pass

For every position of R, we need to check if it is in between the two players, as the probability would be halved, then if the pass is successful we get a transition, and if it is unsuccessful, we accumulate the probability for the same reasoning as in movement.

2.2.3 Shoot

For shooting both there are only two possible transitions, one to each end state.

For all possible movements of R, we check if it is standing in front of the goal, and then accumulate the probability of a successful goal. Then the transition to both the end states (with and without goal) are printed using this probability and its complement (1 - probability).

2.3 Analysis

The game ends with a win when the reward is 1, which is the case when a goal is scored. Thus, the probability of winning given a starting state, is precisely the value function of that state, because the value function describes the expected number of goals from that state.

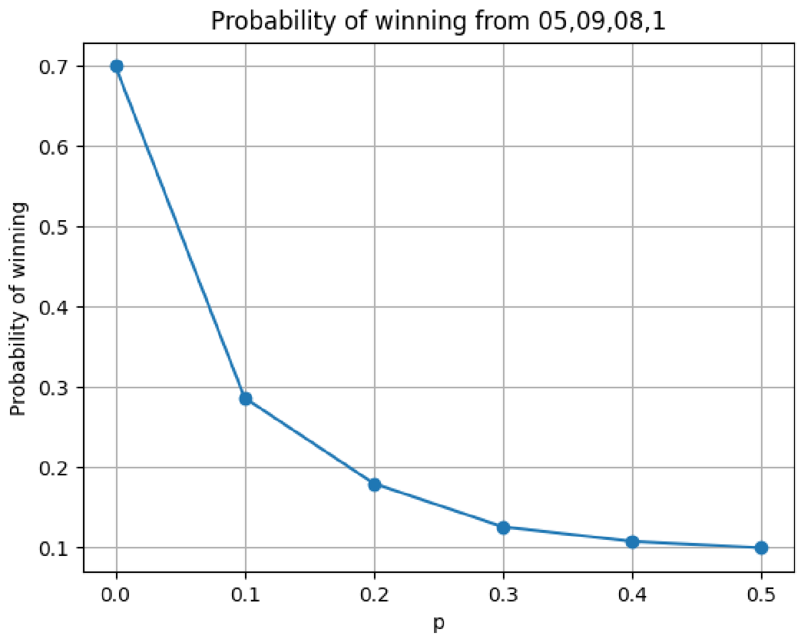

Figure 1: Variation of the probability of winning with p for fixed q (= 0.7)

As p increases, the probability of the player losing the game while moving increases (probability of losing possession directly and probability of failure when tackling occurs).

Thus, a lower p means that the players are better able to have possession of the ball and continue playing (surviving being tackled).

This results in a higher probability of winning (expected number of goals).

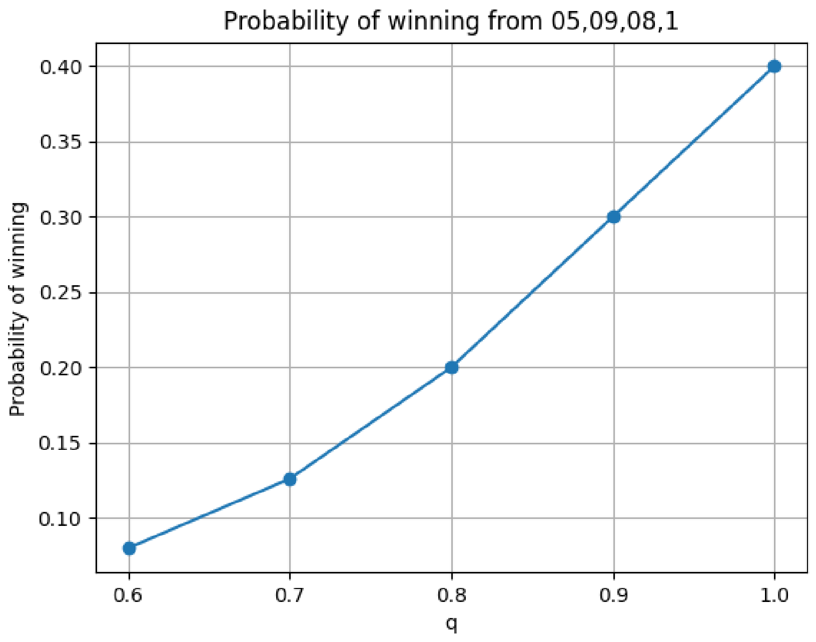

Figure 2: Variation of the probability of winning with q for fixed p (= 0.3)

As q increases, the probability of a successful pass and also that of a successful shoot increases.

Thus, with a higher q, the players are able to score more goals, which increases the probability of winning.

Yes, both of these variations are intuitive, as I have already explained that the parameters p and q represent the players' movement weakness and passing and shooting strengths respectively. Thus, a lower p and a higher q result in higher probability of winning.