Reinforcement Learning Assignment

CAP-6671 Spring 2020

Sabaina Haroon

Q-Learning in Maze World

For this assignment, I have implemented a maze solver of 10x10 grid using Q learning. The algorithm learns a Q-table during learning phase. Q table has total number of states as rows and actions as columns. For this game we had 100 states, each representing location of a block in the maze and 4 actions that can be taken by the robot from any unoccupied location. Actions are up, down, left and right. Q table learns which action is best to take by the robot against any state in the maze. Following are the details of techniques used to learn this algorithm.

Transition and Reward functions

- **Transition Function:** Transition function takes as input, current state and an action and proposes a new state for robot. Transition function is used in this algorithm to add stochasticity, or uncertainty to the robot's environment. This function takes the robot where it wants to go 60 percent of the times, 30 percent of the times it is guided to another direction to explore and 10 percent of the times it stays where it is. If the new action takes robot out of the maze, the robot stays put at the current position.

```
def transition(action):
    chance = random.uniform(0.0, 1.0)
    if chance < 0.6:
        action = action
    elif chance < 0.7:
        action = (action + 1) % 4
    elif chance < 0.8:
        action = (action + 2) % 4
    elif chance < 0.9:
        action = (action + 3) % 4
    else:
        return current_pos, action  # not moving/ not taking any action
```

Figure 1: Transition function for maze Solver Robot

- **Reward Function:** There are 25 obstacles placed around the maze at 25 different locations. Robot faces a penalty of -100 points if it hits the obstacle, gets reward of 100 points for reaching goal state and 0 points for moving in any other direction.

```
def get_reward(reward):
    if obstacle:
        reward -= 100
    if goal:
        reward += 100
    else:
        reward += 0
```

Figure 2: Reward function pseudocode

Exploration Policy

Robot is using **Epsilon Greedy Exploration policy** for this game. When a robot is exploring its surrounding, rather than moving in a set path it wanders around to learn better about the environment. Q learning generates a table to guide the robot how to go from one state to another. Using the Q table to move the robot in maze is defined as exploitation. Greedy exploration policy helps the robot to explore the environment at the beginning knowing more about the obstacles and allowed maneuvers since Q-table is zero or empty at the beginning. After some time when the robot has learnt about the environment, we use Q table for some of the end moves in the environment. Stochasticity is still added to the environment while using Q-table; since robot selects next state from transition function where despite exploiting 60% of the time robot explores the environment randomly for the other 40 percent.

Epsilon is set to 1 at the beginning of the game. Robot explores the environment when value of epsilon is higher. Value of epsilon is decreased exponentially after each game by a value of **3e-4.** A game ends if robot has reached to goal or has been wandering around the maze for past 150 steps.

Q-Learning Rule Implementation:

Q learning rule is implemented for maze solving task using Bellman equation's modified form with learning rate and discount factor incorporated in it. This rule is used to find values for Q table. Value for an action against a state is calculated using previous value for taking such action, reward for next state and maximum of all the actions against next state subtracted the value for current state's action. Below is the Q-learning rule used for this game.

$$Q(s,a) = Q(s,a) + \alpha(r(s') + \gamma.[\max(Q[s']) - Q(s,a)])$$

- **s** is current state
- **a** is action selected to go to new state from s
- **s'** is new state
- **r**(s') returns reward for new state
- max(Q[s']) gives largest value of all the actions taken from new state
- Q (s, a) is value of taking an action in state s obtained from Q table
- $\boldsymbol{\alpha}$ is learning rate. This learning rate decides how much we want the robot to learn. It has value between 0 and 1. 0 means Q table is not updated at all, 1 means robot is learning to reach the goal in a much quicker way.
- $\boldsymbol{\gamma}$ is discount factor. It also has value between 0 and 1 and is used to control the robot to not be guided only by current reward but also by future rewards.

Since robot will never be standing in obstacle state, Q table with obstacle states will have entire row as zero. Therefore, when an action transits a robot from current state to an obstacle state, Q Learning Rule is implemented in a slightly modified way using the equation below.

$$Q(s,a) = Q(s,a) + \alpha(r(s') + Q(s,a))$$

Parameters:

No of Episodes: I am using 600 episodes for robot to learn the stochastic maze world. Number of episodes is not empirical rather guided by epsilon value; robot keeps on learning until the value of epsilon decreases from 1 to 0 with an exponential decay. An episode ends if 150 steps have been taken or robot has reached the goal. Increasing number of episodes make robot to learn more confidently.

α: Value of learning rate used for this game is 0.01. To make sure that robot learns the environment after exploring the environment sufficiently Q table is filled slowly with a value of 0.01. Increasing this value fills the Q table quickly but doesn't get enough chance of exploring the environment.

γ: Value of discount factor used for this game is 0.9. Keeping this value high allowed robot to find optimal paths for reaching the goal. I tried training the game with smaller values of gamma, but with this value robot learnt policy with minimum steps to reach the goal.

Heatmap of the final Q-table (post-learning):

Results in this report are generated using following 10x10 grid map of maze. But my program can also generate random maze sequences and works for all the possible placements of obstacles in the maze.

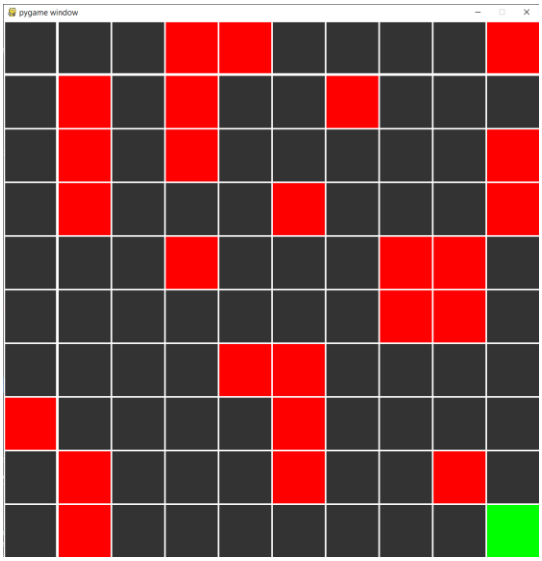

Figure 3: 10x10 Maze. Red Blocks are obstacles. Green block is goal

State	Up	Down	Left	Right
1	0	0	0	0
2	0	-24	0	0
3	0	0	0	-21
4	0	0	0	0
5	0	0	0	0
6	0	0	-23	0
7	0	-17	0	0
8	0	0	0	0
9	0	0	0	-24
10	0	0	0	0
11	0	0	0	-22
12	0	0	0	0
13	0	0	-13	-22
14	0	0	0	0
15	-10	0	-10	0
16	0	0	0	-13
17	0	0	0	0
18	0	0	-31	0
19	0	0	0	0
20	-9	-11	0	0
21	0	0	0	-19
22	0	0	0	0
23	0	0	-13	-17
24	0	0	0	0
25	0	0	-17	0
26	0	-20	0	0
27	-44	0	0	0
28	0	0	0	0
29	0	0	0	-24
30	0	0	0	0
31	0	0	0	-30
32	0	0	0	0
33	0	0	-25	0
34	-9	-8	0	0
35	0	0	0	-15
36	0	0	0	0
37	0	2	-41	0
38	0	-26	0	0
39	0	-10	0	-11
40	0	0	0	0
41	0	0	0	0
42	-31	0	0	0
43	0	0	0	-24
44	0	0	0	0
45	0	0	-13	0
46	-16	0	0	0
47	0	6	0	-45
48	0	0	0	0
49	0	0	0	0
50	-14	0	-11	0

State	Up	Down	Left	Right
51	0	0	0	0
52	0	0	0	0
53	0	0	0	0
54	-13	0	0	0
55	0	-17	0	0
56	0	-27	0	1
57	0	15	0	-50
58	0	0	0	0
59	0	0	0	0
60	0	4	-18	0
61	0	-30	0	0
62	0	0	0	0
63	0	3	0	0
64	0	2	0	-31
65	0	0	0	0
66	0	0	0	0
67	0	29	-43	0
68	-18	2	0	0
69	-26	0	0	2
70	0	21	0	0
71	0	0	0	0
72	0	-30	-26	0
73	0	0	0	8
74	0	17	0	0
75	-26	2	0	-21
76	0	0	0	0
77	0	48	-41	1
78	0	18	1	0
79	0	-22	0	9
80	0	53	0	0
81	-26	0	0	-18
82	0	0	0	0
83	0	1	-27	0
84	0	30	0	1
85	0	19	0	-41
86	0	0	0	0
87	3	67	-53	8
88	0	61	3	-35
89	0	0	0	0
90	5	85	-36	0
91	0	0	0	-36
92	0	0	0	0
93	0	0	-41	11
94	2	0	0	45
95	1	0	8	59
96	-72	0	11	70
97	21	0	24	80
98	23	0	30	89
99	-73	0	41	99
100	0	0	0	0

Figure 4: Heat Map of Q-table; ***Left:*** for first 50 states, ***Right:*** for last 50 states

Two example action sequences generated using your policy

Sequence 1

['left', 'left', 'down', 'left', 'down', 'down', 'down', 'down', 'down', 'down', 'down', 'right', 'right', 'right']

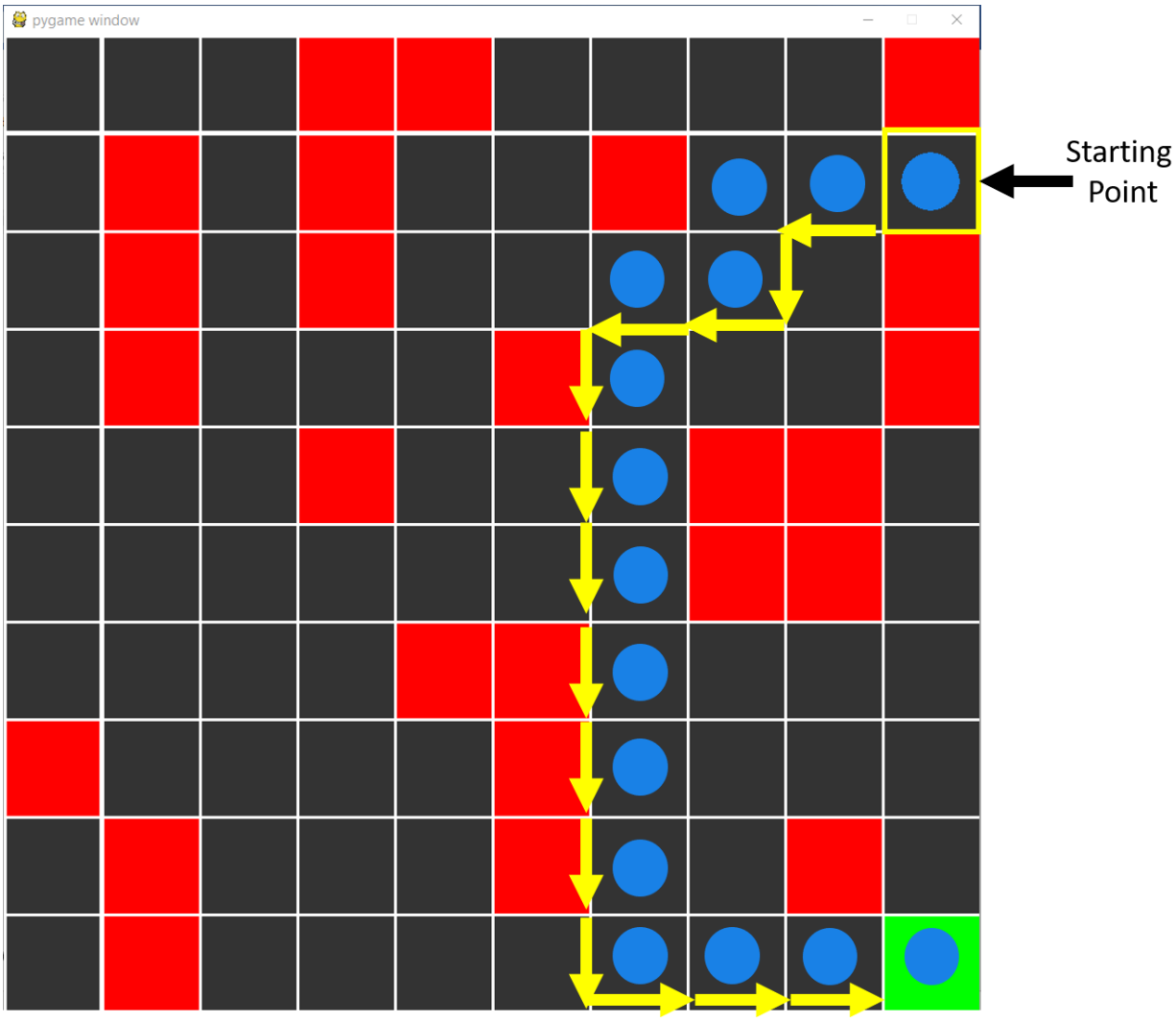

Figure 5: Action sequences generated using trained Policy

Sequence 2

['right', 'down', 'down', 'right', 'down', 'right', 'down', 'down', 'right', 'right', 'right', 'right', 'right', 'right']

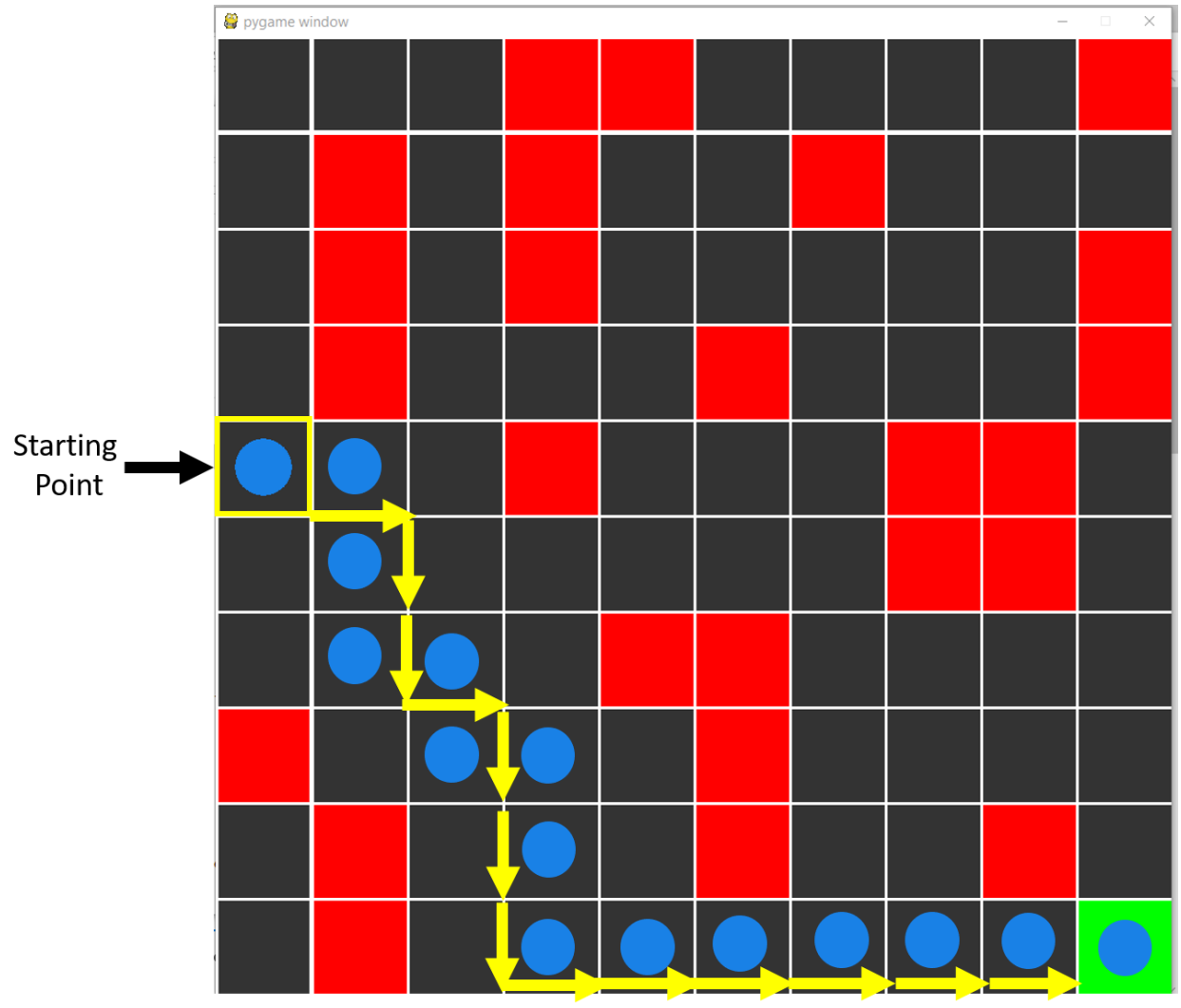

Figure 6: Action sequences generated using trained Policy