NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CZ3005 Artificial Intelligence

Lab Exercise 2: Reinforcement Learning

DSAI

U1821610C

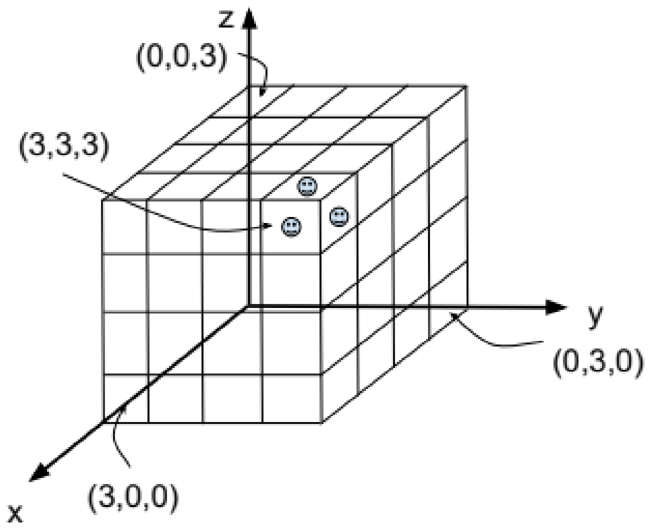
Liew Zhi Li (Sherna)

Contents

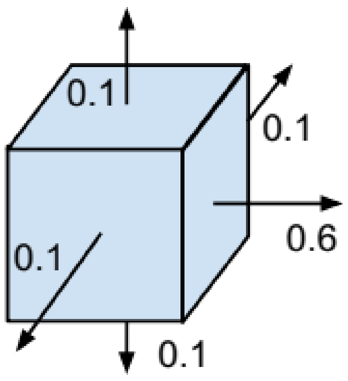
1 – Treasure Hunting in a Cube 2
2 – Q-Learning Algorithm 3
 2.1 – Epsilon-Greedy Algorithm 3
3 – Implementation of Q-Learning Algorithm 4
4 – Q-Table 6
5 – Plot of Learning Progress 10
6 – References 11

1 – Treasure Hunting in a Cube

This project is about applying reinforcement learning (RL) algorithm in the context of treasure hunting in a cube in a grid-world-based environment.



(a) 3D grid world. Smile faces represent terminal states which give reward 1.



(b) The illustration of transition, e.g., the intended action is RIGHT

Figure 1: Illustration of treasure hunting in a cube

The environment is a 3D grid world. The MDP is given in the table below.

State	3D coordinate: (x,y,z) indicates the current position of the agent. We begin with initial state (0,0,0). The terminal state is (3,3,3).
Action	Action space contains 6 actions: left, right, forward, backward, up, down.
Reward	At terminal state, reward = 1. At non-terminal states, reward = -0.1.
Transition	When the intended action is right, the probability of moving right = 0.6. There is a probability of 0.1 it will slip in the perpendicular direction. If the agent collides with a wall, it will remain in the same state.

2 – Q-Learning Algorithm

The RL algorithm used in this project is the Q-Learning algorithm.

Q-Learning is an off-policy, model-free RL algorithm based on the Bellman Equation. It allows the agent to make use of the environment's rewards to learn over time to decide on the best action to take in a given state.

The Q-value of a state action pair is the sum of the instant reward and the discounted future reward. A Q-table is used to store the Q-values for each state and action.

Initially, the Q-values in the Q-table are initialized to 0. As the agent is exposed to the environment, it receives different rewards by executing different actions. The Q-values in the Q-table will be updated using the equation:

$$Q(state, action) \leftarrow (1-\alpha)Q(state, action) + \alpha(reward + \gamma max_a Q(next\ state, all\ actions))$$

Alpha α	The learning rate is from 0 < α ≤ 1
Gamma γ	The discount factor is from 0 ≤ γ ≤ 1 It denotes how much importance we want to give to future rewards. When γ is high (close to 1), long-term rewards have more importance. When γ is low (close to 0), immediate rewards have more importance.

In the first portion $(1-\alpha)Q(state, action)$, we multiply a weight (1- α) to the old Q-value.

In the second portion $\alpha(reward + \gamma max_a Q(next\ state, all\ actions))$, we multiply α to the learned value. The learned value consists of the reward for taking the current action in the current state plus the discounted (γ) of the maximum reward from the next state once we take the current action.

Finally, we add the first and second portion together and the ← arrow assigns/updates the Q-value.

The pseudocode of Q-Learning algorithm is as follows:

```
1. Initialize Q-values in the Q-table to 0, set α=0.5, γ =0.99, ε = 0.01
2. Loop for each episode until converge
    a. Loop for each step of the episode
        i. Agent takes action according to take_action() for that state
        ii. Agent gets from the environment the reward and the next state
        iii. Update Q-value according to the equation
        iv. Set next state as the current state
        v. If the goal state is reached, terminate = True, then end
```

2.1 – Epsilon-Greedy Algorithm

In take_action(), Epsilon-Greedy Algorithm is used to balance between choosing exploration and exploitation. Exploration allows the agent to improve its current knowledge about each action. Exploitation exploits the agent's current action-value estimates to get the most reward.

Epsilon ε denotes the chance between choosing a random action on each step versus choosing the best action based on the current Q-value.

3 – Implementation of Q-Learning Algorithm

The implementation of the Q-Learning algorithm is in test.py.

```
import argparse
import random
from environment import TreasureCube
import numpy as np

class QLAgent(object):
    def __init__(self):
        self.action_space = ['left','right','forward','backward','up','d
own'] # in TreasureCube
        self.Q = np.zeros([4, 4, 4, 6]) # x, y, z, action
        # Hyperparameters
        self.gamma = 0.99
        self.alpha = 0.5
        self.epsilon = 0.01

    def take_action(self, state):
        # Explore action space: epsilon = 0.01 means 1% of the time, ran
domly pick the action from action_space
        if random.uniform(0, 1) < self.epsilon:
            action = random.choice(self.action_space)
        else:
            # Exploit learned values: 99% of the time, look up Q-
table with the highest action value for this state
            state = tuple([int(char) for char in list(state)])
            action = self.action_space[np.argmax(self.Q[state])]
        return action

    def train(self, state, action, next_state, reward):
        # convert state into a list of int, append action to list, make
into tuple
        state = [int(char) for char in list(state)]
        state.append(self.action_space.index(action))
        state_action = tuple(state)

        # retrieve old_value from Q-table
        old_value = self.Q[state_action]

        # convert next_state into a list of int, make into tuple
        next_state = tuple([int(char) for char in list(next_state)])

        # retrieve next_max from Q-table
        next_max = np.max(self.Q[next_state])

        # update new_value according to Q-Learning algorithm
        new_value = (1 - self.alpha) * old_value + self.alpha * (reward
+ self.gamma * next_max)

        # update new_value in Q-table
        self.Q[state_action] = new_value
```

```python
def test_cube(max_episode, max_step):
    env = TreasureCube(max_step=max_step)
    agent = QLAgent()

    # For plotting the learning progress
    episode_records = []

    for epsisode_num in range(0, max_episode):
        state = env.reset()
        terminate = False
        t = 0
        episode_reward = 0
        while not terminate:
            action = agent.take_action(state)
            reward, terminate, next_state = env.step(action)
            episode_reward += reward
            # you can comment the following two lines, if the output is 
too much
            # env.render() # comment
            # print(f'step: {t}, action: {action}, reward: {reward}') # 
comment
            t += 1
            agent.train(state, action, next_state, reward)
            state = next_state
        print(f'epsisode: {epsisode_num}, total_steps: {t} episode rewar
d: {episode_reward}')
        episode_records.append(episode_reward)
    with np.printoptions(precision=4, suppress=True):
        print(agent.Q) # print Q table with nice format
    import matplotlib.pyplot as plt
    plt.plot(episode_records)
    plt.ylabel('episode reward')
    plt.xlabel('episode no.')
    plt.show()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Test')
    parser.add_argument('--max_episode', type=int, default=500)
    parser.add_argument('--max_step', type=int, default=500)
    args = parser.parse_args()

    test_cube(args.max_episode, args.max_step)
```

4 – Q-Table

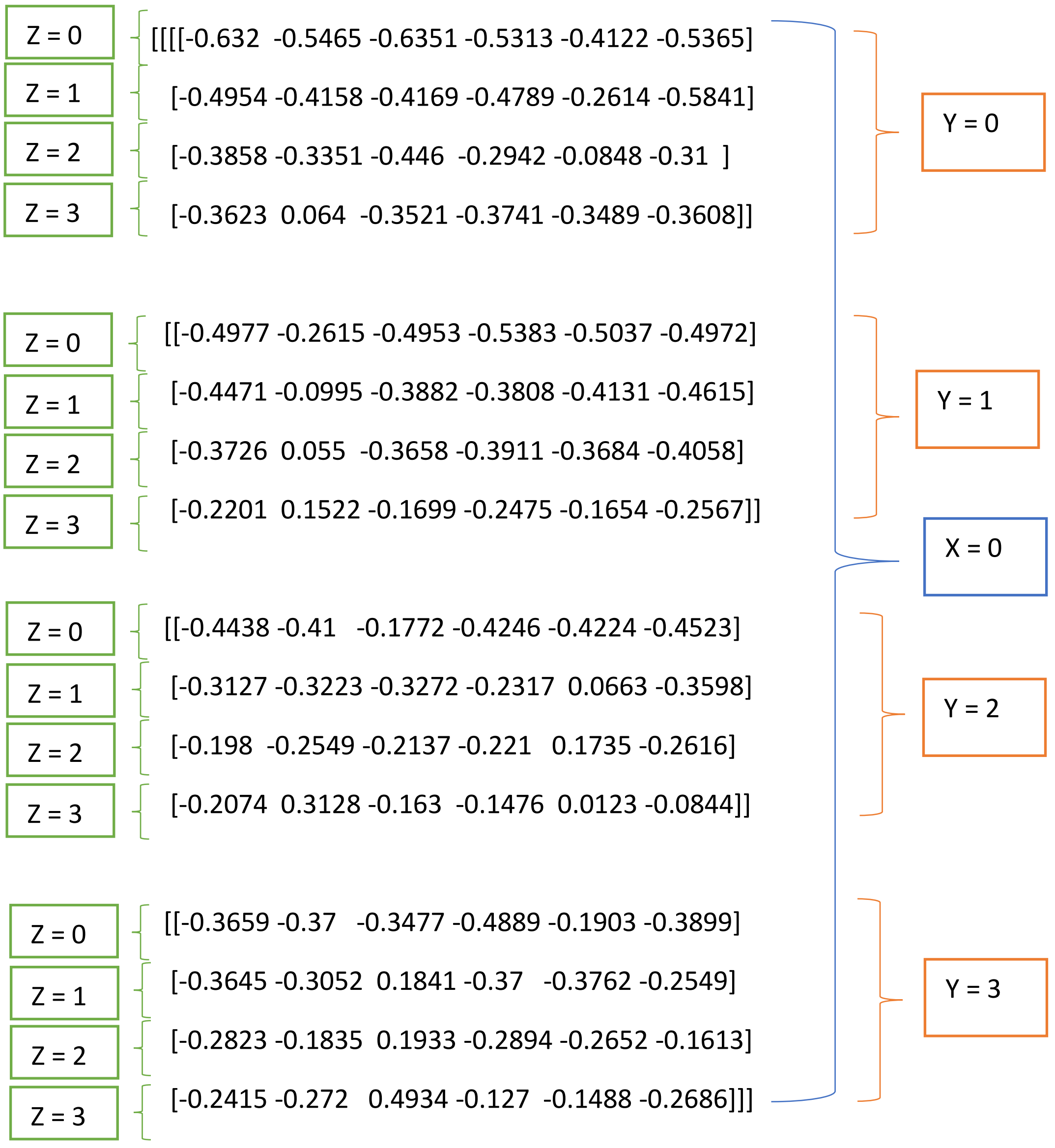
We can run the above file using this command, specifying max_episode = 5000 and max_step = 500.

```
python test.py --max_episode 5000 --max_step 500
```

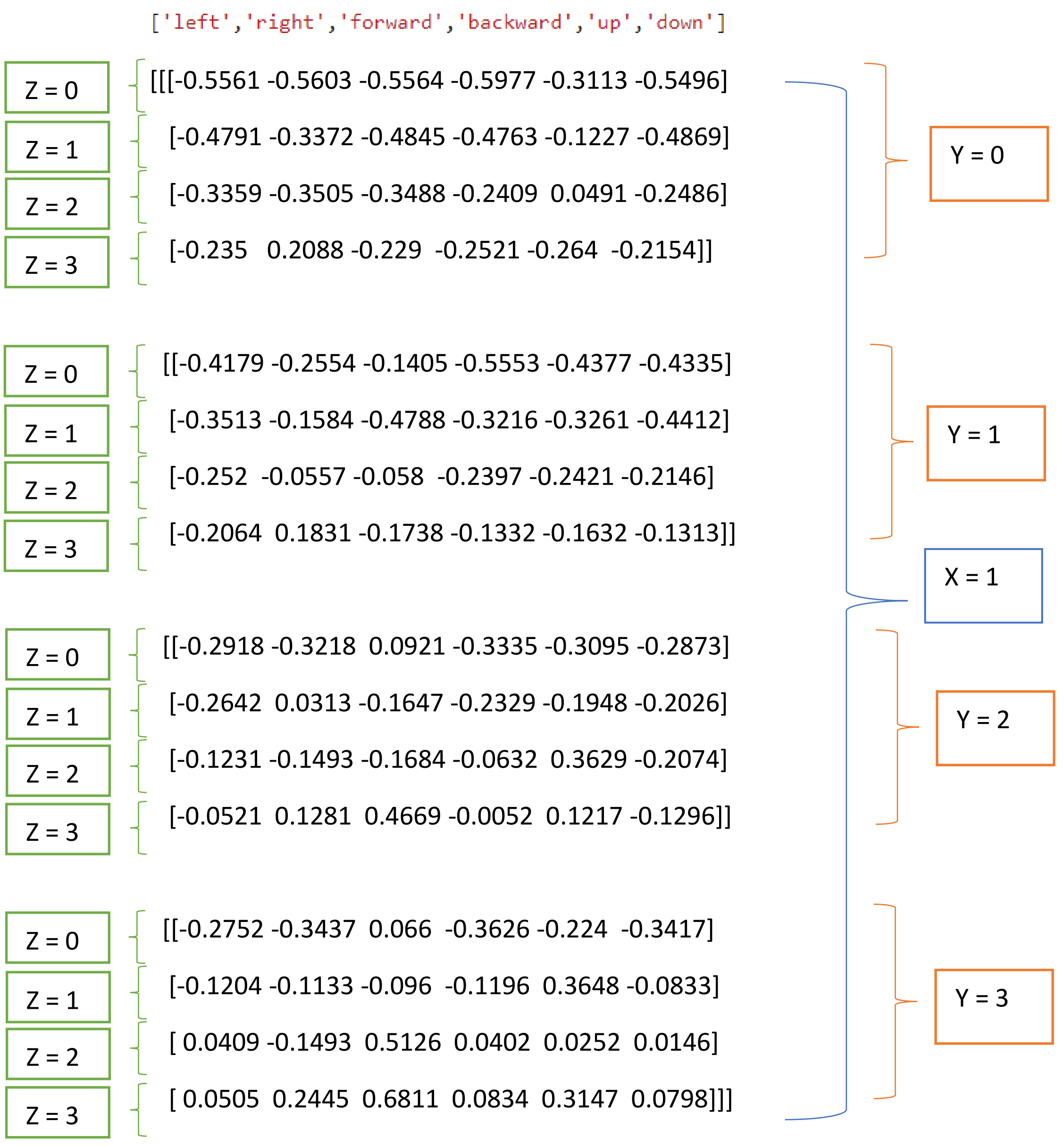
After running the above command, the output shows the Q-table, which is a 4D array as shown.

The columns from left to right are the actions: Left, Right, Forward, Backward, Up, Down.

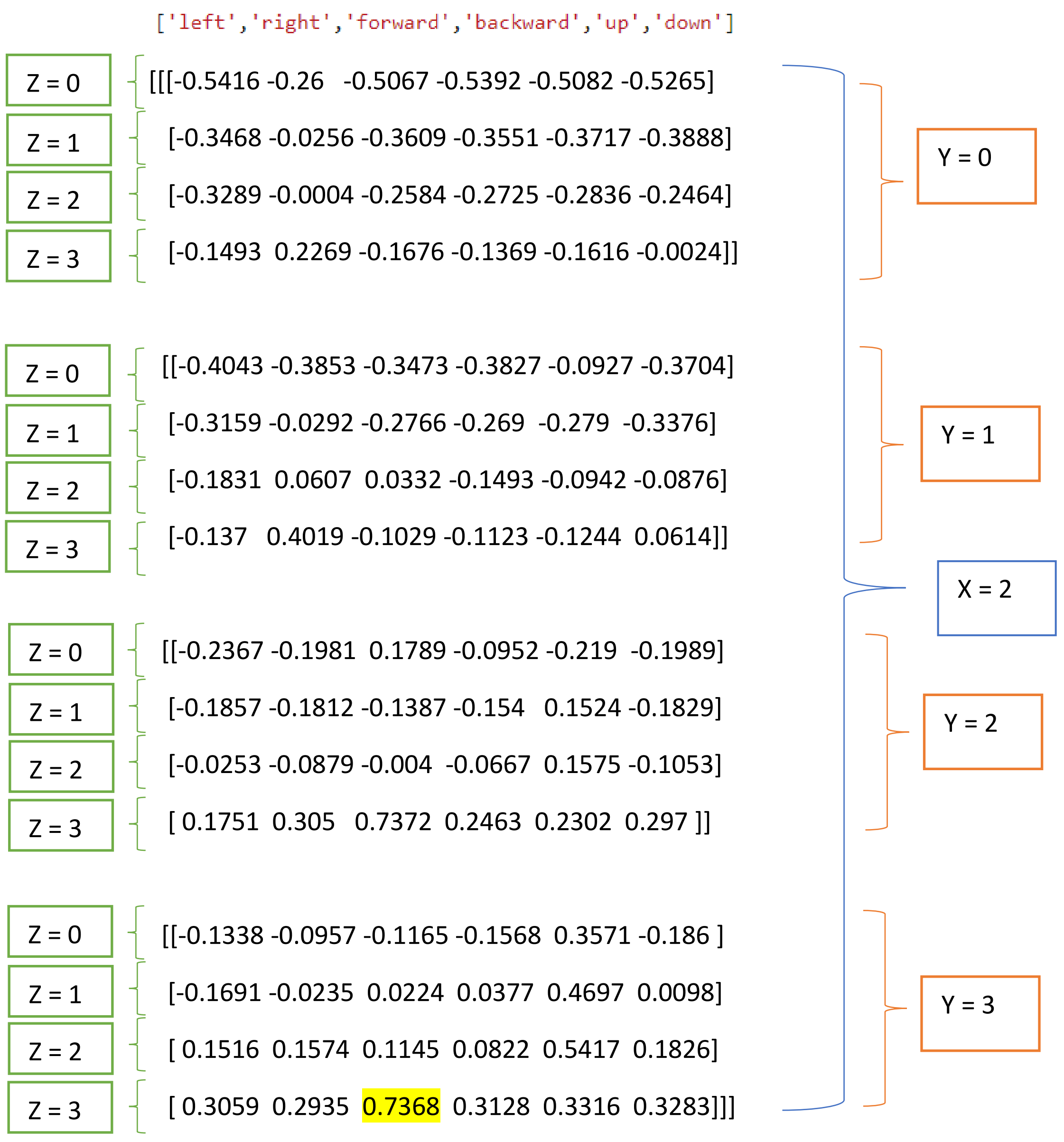
```
['left','right','forward','backward','up','down']
```



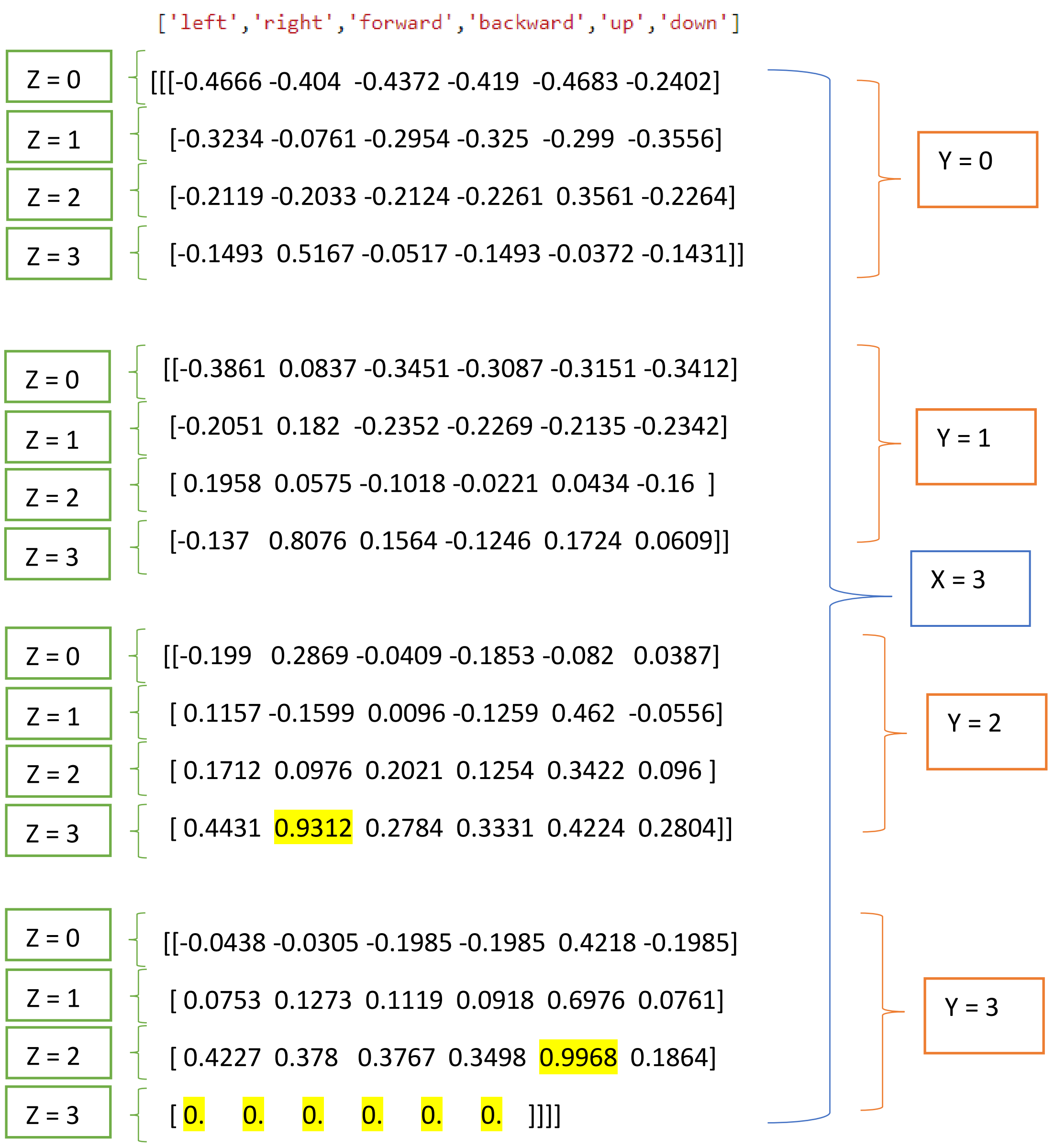
```
[[[[-0.632  -0.5465 -0.6351 -0.5313 -0.4122 -0.5365]
   [-0.4954 -0.4158 -0.4169 -0.4789 -0.2614 -0.5841]
   [-0.3858 -0.3351 -0.446  -0.2942 -0.0848 -0.31  ]
   [-0.3623  0.064  -0.3521 -0.3741 -0.3489 -0.3608]]

  [[-0.4977 -0.2615 -0.4953 -0.5383 -0.5037 -0.4972]
   [-0.4471 -0.0995 -0.3882 -0.3808 -0.4131 -0.4615]
   [-0.3726  0.055  -0.3658 -0.3911 -0.3684 -0.4058]
   [-0.2201  0.1522 -0.1699 -0.2475 -0.1654 -0.2567]]

  [[-0.4438 -0.41   -0.1772 -0.4246 -0.4224 -0.4523]
   [-0.3127 -0.3223 -0.3272 -0.2317  0.0663 -0.3598]
   [-0.198  -0.2549 -0.2137 -0.221   0.1735 -0.2616]
   [-0.2074  0.3128 -0.163  -0.1476  0.0123 -0.0844]]

  [[-0.3659 -0.37   -0.3477 -0.4889 -0.1903 -0.3899]
   [-0.3645 -0.3052  0.1841 -0.37   -0.3762 -0.2549]
   [-0.2823 -0.1835  0.1933 -0.2894 -0.2652 -0.1613]
   [-0.2415 -0.272   0.4934 -0.127  -0.1488 -0.2686]]]
```



```
['left','right','forward','backward','up','down']
[[[-0.5561 -0.5603 -0.5564 -0.5977 -0.3113 -0.5496]
  [-0.4791 -0.3372 -0.4845 -0.4763 -0.1227 -0.4869]
  [-0.3359 -0.3505 -0.3488 -0.2409  0.0491 -0.2486]
  [-0.235   0.2088 -0.229  -0.2521 -0.264  -0.2154]]

 [[-0.4179 -0.2554 -0.1405 -0.5553 -0.4377 -0.4335]
  [-0.3513 -0.1584 -0.4788 -0.3216 -0.3261 -0.4412]
  [-0.252  -0.0557 -0.058  -0.2397 -0.2421 -0.2146]
  [-0.2064  0.1831 -0.1738 -0.1332 -0.1632 -0.1313]]

 [[-0.2918 -0.3218  0.0921 -0.3335 -0.3095 -0.2873]
  [-0.2642  0.0313 -0.1647 -0.2329 -0.1948 -0.2026]
  [-0.1231 -0.1493 -0.1684 -0.0632  0.3629 -0.2074]
  [-0.0521  0.1281  0.4669 -0.0052  0.1217 -0.1296]]

 [[-0.2752 -0.3437  0.066  -0.3626 -0.224  -0.3417]
  [-0.1204 -0.1133 -0.096  -0.1196  0.3648 -0.0833]
  [ 0.0409 -0.1493  0.5126  0.0402  0.0252  0.0146]
  [ 0.0505  0.2445  0.6811  0.0834  0.3147  0.0798]]]
```



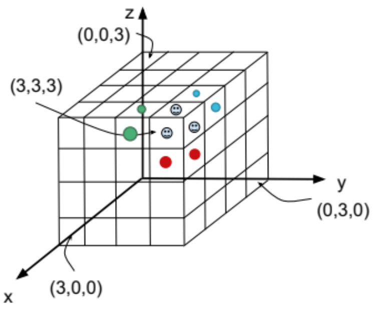
```
['left','right','forward','backward','up','down']
[[[-0.5416 -0.26   -0.5067 -0.5392 -0.5082 -0.5265]
  [-0.3468 -0.0256 -0.3609 -0.3551 -0.3717 -0.3888]
  [-0.3289 -0.0004 -0.2584 -0.2725 -0.2836 -0.2464]
  [-0.1493  0.2269 -0.1676 -0.1369 -0.1616 -0.0024]]

 [[-0.4043 -0.3853 -0.3473 -0.3827 -0.0927 -0.3704]
  [-0.3159 -0.0292 -0.2766 -0.269  -0.279  -0.3376]
  [-0.1831  0.0607  0.0332 -0.1493 -0.0942 -0.0876]
  [-0.137   0.4019 -0.1029 -0.1123 -0.1244  0.0614]]

 [[-0.2367 -0.1981  0.1789 -0.0952 -0.219  -0.1989]
  [-0.1857 -0.1812 -0.1387 -0.154   0.1524 -0.1829]
  [-0.0253 -0.0879 -0.004  -0.0667  0.1575 -0.1053]
  [ 0.1751  0.305   0.7372  0.2463  0.2302  0.297 ]]

 [[-0.1338 -0.0957 -0.1165 -0.1568  0.3571 -0.186 ]
  [-0.1691 -0.0235  0.0224  0.0377  0.4697  0.0098]
  [ 0.1516  0.1574  0.1145  0.0822  0.5417  0.1826]
  [ 0.3059  0.2935  0.7368  0.3128  0.3316  0.3283]]]
```



```
['left','right','forward','backward','up','down']
[[[-0.4666 -0.404  -0.4372 -0.419  -0.4683 -0.2402]
  [-0.3234 -0.0761 -0.2954 -0.325  -0.299  -0.3556]
  [-0.2119 -0.2033 -0.2124 -0.2261  0.3561 -0.2264]
  [-0.1493  0.5167 -0.0517 -0.1493 -0.0372 -0.1431]]

 [[-0.3861  0.0837 -0.3451 -0.3087 -0.3151 -0.3412]
  [-0.2051  0.182  -0.2352 -0.2269 -0.2135 -0.2342]
  [ 0.1958  0.0575 -0.1018 -0.0221  0.0434 -0.16  ]
  [-0.137   0.8076  0.1564 -0.1246  0.1724  0.0609]]

 [[-0.199   0.2869 -0.0409 -0.1853 -0.082   0.0387]
  [ 0.1157 -0.1599  0.0096 -0.1259  0.462  -0.0556]
  [ 0.1712  0.0976  0.2021  0.1254  0.3422  0.096 ]
  [ 0.4431  0.9312  0.2784  0.3331  0.4224  0.2804]]

 [[-0.0438 -0.0305 -0.1985 -0.1985  0.4218 -0.1985]
  [ 0.0753  0.1273  0.1119  0.0918  0.6976  0.0761]
  [ 0.4227  0.378   0.3767  0.3498  0.9968  0.1864]
  [ 0.      0.      0.      0.      0.      0.    ]]]]
```

From the Q-table, we can see that for example:

At [3,3,3] as indicated by the smiley faces in the diagram, the action values are all 0 because it is the terminal state.



At [3,3,2] as indicated by the red dots in the diagram, the optimal action is Up, as the highest Q-value among all 6 actions is 0.9968.

At [3,2,3] as indicated by the green dots in the diagram, the optimal action is right, as the highest Q-value among all 6 actions is 0.9312.

At [2,3,3] as indicated by the blue dots in the diagram, the optimal action is forward, as the highest Q-value among all 6 actions is 0.7368.

5 – Plot of Learning Progress

After running the above command, the program can also generate the plot of the learning progress of episode rewards vs episode no. This is done using the matplotlib library.

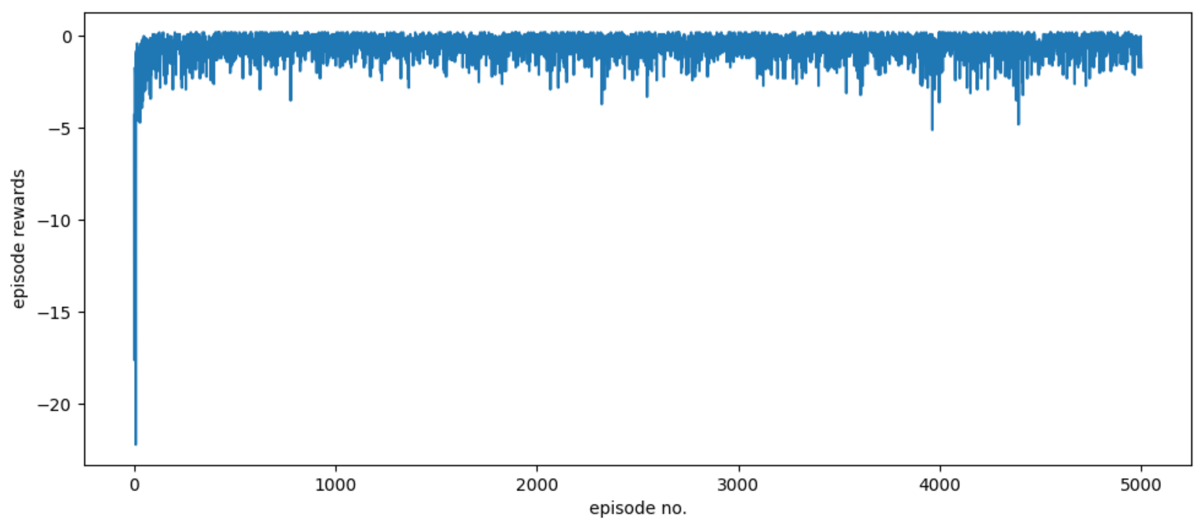


Fig. 1: Episode Rewards vs Episode No.

We can see that in the beginning few episodes, the episode reward values varied from 0 to -20 sharply. This is because the agent is still learning.

After a short amount of time of about 100 iterations all the way to 5000 iterations, the episode reward values have improved greatly and varied from between 0 and -5. This can be due to two reasons:

1. **Epsilon-Greedy Algorithm**
    a. We set Epsilon ε = 0.01.
    b. This means there is a 1% chance it will choose a random action on each step.
    c. 99% of the time, it will look up the Q-table for this state and choose the action with the highest value.
2. **Environment**
    a. There is a 40% chance it will 'slip' and not go in the direction it intends to because of the transition model.
    b. If the intended action is right, there is a 10% chance to slip in each of the perpendicular directions (forward, backward, up, down).
    c. If the intended action is up, there is a 10% chance to slip in each of the perpendicular directions (left, right, forward, backward).

6 – References

The following are the online resources that aided in my understanding and accomplishment of this project.

[1] S. Kansal and B. Martin, "Reinforcement Q-Learning from Scratch in Python with OpenAI Gym", Learndatasci.com, 2020. [Online]. Available: https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/. [Accessed: 17- Oct- 2020].

[2] "Q-Learning Explained - A Reinforcement Learning Technique", Deeplizard.com, 2020. [Online]. Available: https://deeplizard.com/learn/video/qhRNvCVVJaA. [Accessed: 17- Oct- 2020].

[3] "Q-learning", En.wikipedia.org, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Q-learning. [Accessed: 17- Oct- 2020].

[4] "Epsilon-Greedy Algorithm in Reinforcement Learning - GeeksforGeeks", GeeksforGeeks, 2020. [Online]. Available: https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/. [Accessed: 17- Oct- 2020].