CS3120-Machine Learning and Neural Computing
2021

Assignment: Reinforcement Learning with Python

Name: D.M.U.S.Dissanayaka

Index: s14320

Date: 22/08/2021

ABSTRACT

The main purpose of this assignment was to do coding for Reinforcement Learning with Python. The main objective of this assignment was to gain brief knowledge about Reinforcement Learning with Python and octave. In this experiment was included three parts which are practical the given code example, modelling the discussed example problem in the lecture note using Python and compare the output and results in Octave verse Python. And the third part was answered by modelling the Exercise problem of maze solving given in the lecture note. Jupiter notebook and Octave were used to analyze output and results. This software reduces the errors involved in experimenting as compared to those performed manually. This is because Colab notebook and Octave which lead to high precision and accuracy in the results obtained while analyzing outputs. In conclusion, Q-learning appears to be an intriguing idea and maybe considerably more entrancing than customary directed Artificial Intelligent since for this situation the machine is essentially gaining without any preparation how to play out a specific undertaking to enhance potential compensations.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Reinforcement Learning

Reinforcement Learning is a part of Artificial Intelligent. Here, specialists are self-prepared on remuneration and discipline instruments. It's tied in with making the most ideal move or way to acquire the greatest task and least discipline through perceptions in a particular circumstance. It goes about as a sign of positive and negative practices. The specialists are constructed that can see and decipher the background where is put and moreover. Furthermore, it can make moves and connect with it.

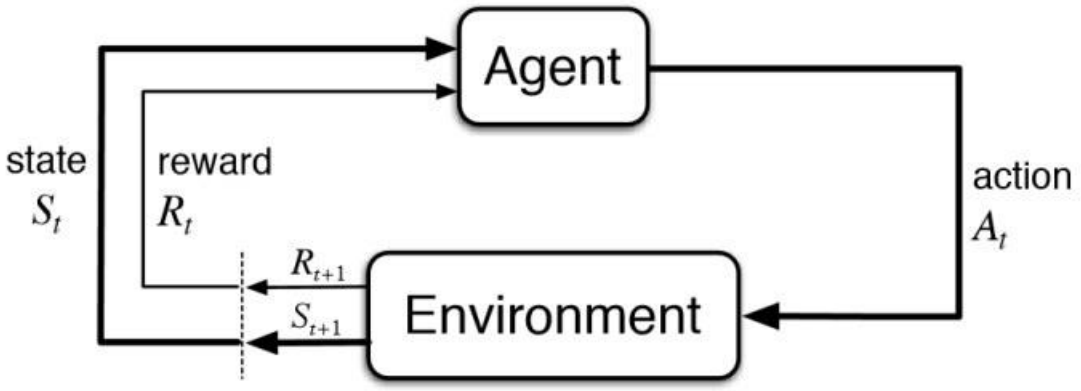


Figure 1- Basic Diagram of Reinforcement Learning

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

```
Set the gamma parameter (e.g. 0.8), and environment rewards in matrix R.
Initialize matrix Q to zero.
For each episode:
        Select a random initial state.
        Do While the goal state hasn't been reached.
                Select one among all possible actions for the current state.
                Using this possible action, consider going to the next state.
                Get maximum Q value for this next state based on all possible actions.
                Compute: Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]
                Set the next state as the current state.
        End Do
End For
```

Figure 2 The Pseudo-code for Q-Table update rule (reference: Lecture note)

2 METHODOLOGY

2.1 Part 01 - Python Q-learning Example

- The libraries were imported. And a points-list map was created and that represent each direction the bot could take.

```
import numpy as np
import pylab as plt

# map cell to cell, add circular cell to goal point
points_list = [(0,1), (1,5), (5,6), (5,4), (1,2), (2,3), (2,7)]
```

- After that, the starting point was assigned as 0 and the goal point was assigned as 7. Then the “networkx” graph was plotted.

```
goal = 7

import networkx as nx
G=nx.Graph()
G.add_edges_from(points_list)
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G,pos)
nx.draw_networkx_edges(G,pos)
nx.draw_networkx_labels(G,pos)
plt.show()
```

- Then the rewards graph was plotted by using the matrix version of the list of points map. Which was initialized the matrix to be the height and width of the points list and gave the initialize all values to -1.

```
# how many points in graph? x points
MATRIX_SIZE = 8

# create matrix x*y
R = np.matrix(np.ones(shape=(MATRIX_SIZE, MATRIX_SIZE)))
R *= -1
```

- Therefore, values were changed to be 0 if it was a viable path and 100 if it was a goal path and assigned zeros to paths and 100 to a goal-reaching point.

```
# assign zeros to paths and 100 to goal-reaching point
for point in points_list:
    print(point)
    if point[1] == goal:
        R[point] = 100
    else:
        R[point] = 0
```

```
if point[0] == goal:
        R[point[::-1]] = 100
    else:
        # reverse of point
        R[point[::-1]]= 0

# add goal point round trip
R[goal,goal]= 100

R
```

- The Q-learning matrix was created by using an equation of `Q[current_state, action] = R[current_state, action] + gamma * max_value` round the goal point with a learning rate (gamma) equal to 0.8.

```
Q = np.matrix(np.zeros([MATRIX_SIZE,MATRIX_SIZE]))

# learning parameter
gamma = 0.8

initial_state = 1

def available_actions(state):
    current_state_row = R[state,]
    av_act = np.where(current_state_row >= 0)[1]
    return av_act

available_act = available_actions(initial_state)

def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_act,1))
    return next_action

action = sample_next_action(available_act)

def update(current_state, action, gamma):

  max_index = np.where(Q[action,] == np.max(Q[action,]))[1]

  if max_index.shape[0] > 1:
      max_index = int(np.random.choice(max_index, size = 1))
  else:
      max_index = int(max_index)
  max_value = Q[action, max_index]

  Q[current_state, action] = R[current_state, action] + gamma *
max_value
  print('max_value', R[current_state, action] + gamma * max_value)

  if (np.max(Q) > 0):
    return(np.sum(Q/np.max(Q)*100))
  else:
    return (0)

update(initial_state, action, gamma)
```

- The function was trained and tested which would be run the update function 700 iterations for the Q-learning model. After that figure out the most efficient path and the graph of the scores versus the number of iterations was plotted.

```
# Training
scores = []
for i in range(700):
    current_state = np.random.randint(0, int(Q.shape[0]))
    available_act = available_actions(current_state)
    action = sample_next_action(available_act)
    score = update(current_state,action,gamma)
    scores.append(score)
    print ('Score:', str(score))

print("Trained Q matrix:")
print(Q/np.max(Q)*100)

# Testing
current_state = 0
steps = [current_state]

while current_state != 7:

    next_step_index = np.where(Q[current_state,]
        == np.max(Q[current_state,]))[1]

    if next_step_index.shape[0] > 1:
        next_step_index =
            int(np.random.choice(next_step_index, size = 1))
    else:
        next_step_index = int(next_step_index)

    steps.append(next_step_index)
    current_state = next_step_index

print("Most efficient path:")
print(steps)

plt.plot(scores)
plt.show()
```

2.2 Part 02- Example 01

2.2.1 Python (Colab Notebook)

- The libraries were imported. And a points-list map was created and that represent each direction the bot could take

```
[1]  import numpy as np
     import pylab as plt
```

- The points were mapped to certain paths. And a points-list map was created and that represent each direction the bot could take.

```
[2]  # map cell to cell, add circular cell to goal point
     points_list = [(0,4), (4,5), (4,3), (3,2), (3,1), (1,5)]
```

- After that, the starting point was assigned as 2 and the goal point was assigned as 5. Then the “networkx” graph was plotted.

```
goal = 5

import networkx as nx
G=nx.Graph()
G.add_edges_from(points_list)
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G,pos)
nx.draw_networkx_edges(G,pos)
nx.draw_networkx_labels(G,pos)
plt.show()
```

- The reward table was created with instant reward values. (-1) value was assigned to the null or no link path, 0 value was assigned to the linked path and 100 value was assigned to the goal state.

```
[4]  # how many points in graph? x points
     MATRIX_SIZE = 6

     # create matrix x*y
     R = np.matrix(np.ones(shape=(MATRIX_SIZE, MATRIX_SIZE)))
     R *= -1
```

```
[5]  # assign zeros to paths and 100 to goal-reaching point
     for point in points_list:
         print(point)
         if point[1] == goal:
             R[point] = 100
         else:
             R[point] = 0

         if point[0] == goal:
             R[point[::-1]] = 100
         else:
             # reverse of point
             R[point[::-1]]= 0

     # add goal point round trip
     R[goal,goal]= 100
```

- The Q-learning matrix was created around the goal point with a learning rate (gamma) equal to 0.8. After that initial state was given.

```
Q = np.matrix(np.zeros([MATRIX_SIZE,MATRIX_SIZE]))

# learning parameter
gamma = 0.8

initial_state = 1
```

- All available actions were returned in the state given as an argument. After those available actions were given in the current state then all the available actions within the range were given by using this function. Finally sample next action was printed.

```
def available_actions(state):
    current_state_row = R[state,]
    av_act = np.where(current_state_row >= 0)[1]
    return av_act

available_act = available_actions(initial_state)

def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_act,1))
    return next_action

action = sample_next_action(available_act)
```

- The Q matrix was updated to the path selected and the Q. The Q-learning matrix was created by using the Q learning algorithm formula,
`Q[current_state, action] = R[current_state, action] + gamma * max_value`
of round the goal point with learning rate (gamma) equal to 0.8. Therefore, the defined function was updated in the Q matrix according to the max_index.

```
def update(current_state, action, gamma):

  max_index = np.where(Q[action,] == np.max(Q[action,]))[1]

  if max_index.shape[0] > 1:
      max_index = int(np.random.choice(max_index, size = 1))
  else:
      max_index = int(max_index)
  max_value = Q[action, max_index]

  Q[current_state, action] = R[current_state, action] + gamma * max_value
  print('max_value', R[current_state, action] + gamma * max_value)

  if (np.max(Q) > 0):
    return(np.sum(Q/np.max(Q)*100))
  else:
    return (0)

update(initial_state, action, gamma)
```

- The function was trained and tested which would be run the update function 10,000 iterations for the Q-learning model. Then the numbers of iterations were scored. After that, the maximum value of the Q table was found, and the Q table was normalized and trained using that value.

```
# Training
scores = []
for i in range(10000):
    current_state = np.random.randint(0, int(Q.shape[0]))
    available_act = available_actions(current_state)
    action = sample_next_action(available_act)
    score = update(current_state,action,gamma)
    scores.append(score)
    print ('Score:', str(score))

print("Trained Q matrix:")
print(Q/np.max(Q)*100)
```

- The Q-table was tested and figure out the most efficient path. After that, the graph of the scores versus the number of iterations was plotted for10000 iterations for the Q-learning model.

```
# Testing
current_state = 2
steps = [current_state]

while current_state != 5:

    next_step_index = np.where(Q[current_state,]== np.max(Q[current_state,]))[1]

    if next_step_index.shape[0] > 1:
        next_step_index =int(np.random.choice(next_step_index, size = 1))
    else:
        next_step_index = int(next_step_index)

    steps.append(next_step_index)
    current_state = next_step_index

print("Most efficient path:")
print(steps)

plt.plot(scores)
plt.show()
```

2.2.2 Octave

- The gamma parameter (0.8), and environment rewards in matrix R was set.
- The matrix Q was initialized to zero.
- A random initial state was for each episode.
- The goal state had been reached using a do-while loop and one among all possible actions was selected for the current state.
- After that, the possible action was considered for going to the next state.
- The maximum Q value for this next state based on all possible actions was taken and computed Q-table using the formula of Q (state, action) = R (state, action) + Gamma * Max [Q(next state, all actions) then next state was set as the current state.
- The Q table was computed until it was getting converged. Finally, the maximum value of the Q-table wax was analyzed, and the Q table was divided by it.

```
clc;
clear;
clear all;
R= [-inf,-inf,-inf,-inf,0,-inf;
    -inf,-inf,-inf,0,-inf,100;
    -inf,-inf,-inf,0,-inf,-inf;
    -inf,0,0,-inf,0,-inf;
    0,-inf,-inf,0,-inf,100;
    -inf,0,-inf,-inf,0,100]

gamma = 0.8; % the Gamma (learning parameter).
goalstate = 6; % the goal state is 5
q= zeros(size(R));
 for episode=1:10000
   %selec a random initial state=2
   y=randperm(size(R,2));
   state = y(1)

  while state~= goalstate %find the all posible actions from the state
    actions= find(R(state,:)>=0);
    if size (actions,2)>0 %select one action randomly
      i=randperm(size(actions,2));
      action= actions(i(1));
    end
    %Return a column vector with the max values of each rows
    qMax = max(q, [],2);
    %compute the q values
     q(state, action) = R(state, action) + gamma.*qMax(action);
     % Transition to the next state
     state=action;

  end
 end
```

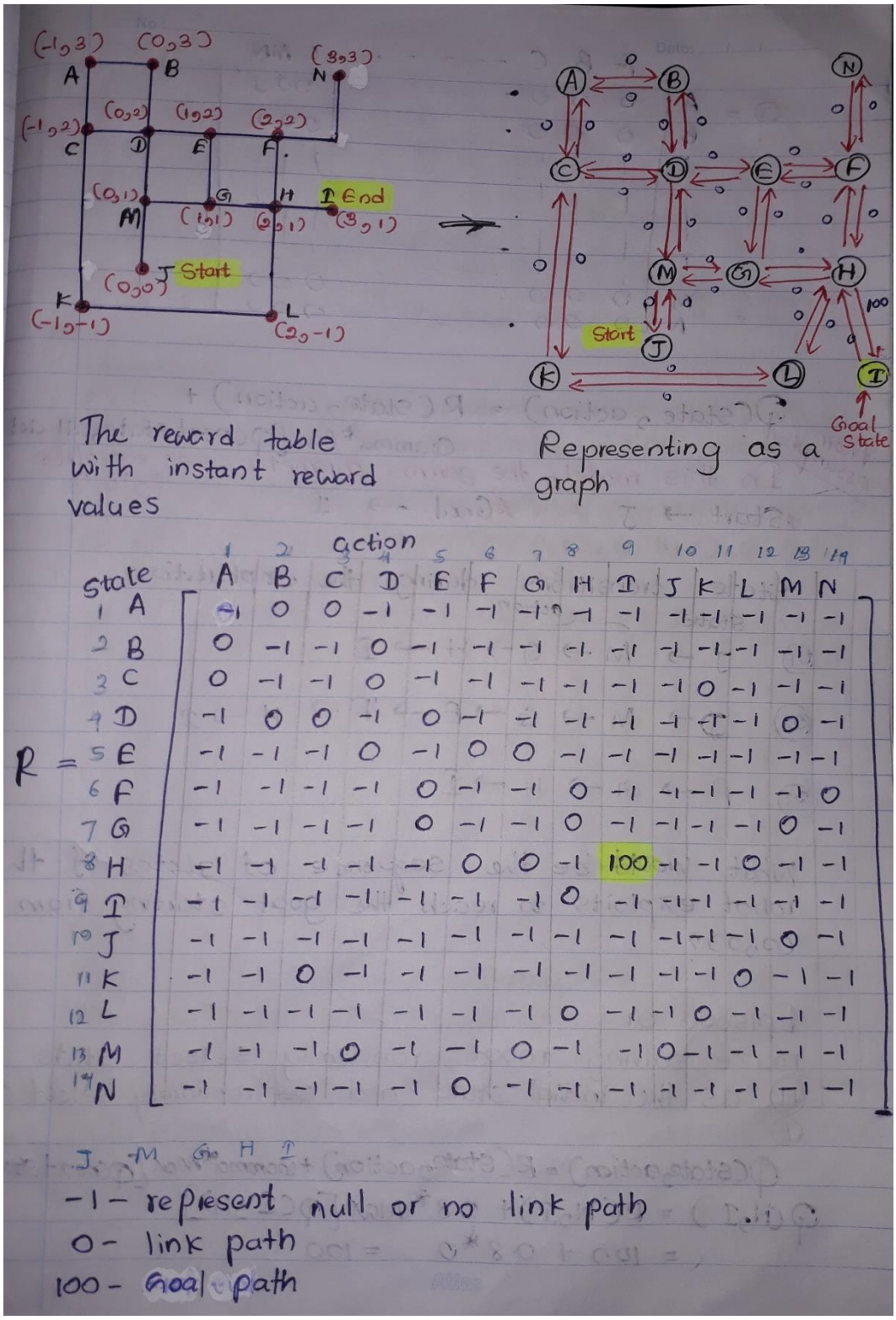
```
% normalized q
          g=max(qMax);
          if g>0,
          q= 100*q./g;
          end
          disp(q)

disp('The q maximun values are:')
disp(qMax);
disp('The most effient path is:')
state=3;
disp(state-1);
while state~=goalstate
  [mx,action] =max(q(state,:));
  state =action;
  disp(state-1)

end
```

2.3 Part 03- Maze solving modelling

- The matrix Q was initialized to zero.
- A random initial state was for each episode. In this part, three types of initial states and the same goal state were given and analyzed the most efficient path for each state.
- The goal state had been reached using a do-while loop and one among all possible actions was selected for the current state.
- After that, the possible action was considered for going to the next state.
- The maximum Q value for this next state based on all possible actions was taken and computed Q-table using the formula of Q (state, action) = R (state, action) + Gamma * Max [Q(next state, all actions) then next state was set as the current state.
- The Q tables were computed until they were getting converged. Finally, maximum values of the Q-tables were analyzed, and Q tables were divided by it.



state	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
B	0	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
C	0	-1	-1	0	-1	-1	-1	-1	-1	-1	0	-1	-1	-1
D	-1	0	0	-1	0	-1	-1	-1	-1	-1	-1	-1	0	-1
E	-1	-1	-1	0	-1	0	0	-1	-1	-1	-1	-1	-1	-1
F	-1	-1	-1	-1	0	-1	-1	0	-1	-1	-1	-1	-1	0
G	-1	-1	-1	-1	0	-1	-1	0	-1	-1	-1	-1	0	-1
H	-1	-1	-1	-1	-1	0	0	-1	100	-1	-1	0	-1	-1
I	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1
J	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1
K	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1
L	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	0	-1	-1	-1
M	-1	-1	-1	0	-1	-1	0	-1	-1	0	-1	-1	-1	-1
N	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1

Figure 3-The reward table for maze solving robot

- The gamma parameter (0.8), and environment rewards in matrix R was set. The reward matrix size was (14X14).

```
clc;
clear;
clear all;% -inf = no link between the points
R= [-inf,0,0,-inf,-inf,-inf,-inf,-inf,-inf,-inf,-inf,-inf,-inf,-inf;
    0,-inf,-inf,0,-inf,-inf,-inf,-inf,-inf,-inf,-inf,-inf,-inf,-inf;
    0,-inf,-inf,0,-inf,-inf,-inf,-inf,-inf,-inf,0,-inf,-inf,-inf;
    -inf,0,0,-inf,0,-inf,-inf,-inf,-inf,-inf,-inf,-inf,0,-inf;
    -inf,-inf,-inf,0,-inf,0,0,-inf,-inf,-inf,-inf,-inf,-inf,-inf;
    -inf,-inf,-inf,-inf,0,-inf,-inf,0,-inf,-inf,-inf,-inf,-inf,0;
    -inf,-inf,-inf,-inf,0,-inf,-inf,0,-inf,-inf,-inf,-inf,0,-inf;
    -inf,-inf,-inf,-inf,-inf,0,0,-inf,100,-inf,-inf,0,-inf,-inf;
    -inf,-inf,-inf,-inf,-inf,-inf,-inf,0,-inf,-inf,-inf,-inf,-inf,-inf;
    -inf,-inf,-inf,-inf,-inf,-inf,-inf,-inf,-inf,-inf,-inf,-inf,0,-inf;
    -inf,-inf,0,-inf,-inf,-inf,-inf,-inf,-inf,-inf,-inf,0,-inf,-inf;
    -inf,-inf,-inf,-inf,-inf,-inf,-inf,0,-inf,-inf,0,-inf,-inf,-inf;
    -inf,-inf,-inf,0,-inf,-inf,0,-inf,-inf,0,-inf,-inf,-inf,-inf;
    -inf,-inf,-inf,-inf,-inf,0,-inf,-inf,-inf,-inf,-inf,-inf,-inf,-inf;]
gamma = 0.8; % the Gamma (learning parameter).
goalstate = 9; % the goal state is I
q= zeros(size(R));
 for episode=1:10000
   %selec a random initial state=2
   y=randperm(size(R,2));
   state = y(1)
  while state~= goalstate %find the all posible actions from the state
    actions= find(R(state,:)>=0);
    if size (actions,2)>0 %select one action randomly
      i=randperm(size(actions,2));
      action= actions(i(1));
    end
    %Return a column vector with the max values of each rows
    qMax = max(q, [],2);
    %compute the q values
     q(state, action) = R(state, action) + gamma.*qMax(action);
     % Transition to the next state
     state=action;

  end
 end
% normalized q
         g=max(qMax);
         if g>0,
         q= 100*q./g;
         end
         disp(q)

disp('The q maximun values are:')
disp(qMax);
disp('The most effient path is:')
state=10; %initial state is J=(0,0)
%state=4; %initial state is D=(0,3)
%state=5; %initial state is E=(1,2)
disp(state);
while state~=goalstate
  [mx,action] =max(q(state,:));
  state =action;
  disp(state)
end
```

3 RESULTS AND ANALYSIS

3.1 Part 01- Python Q-learning Example

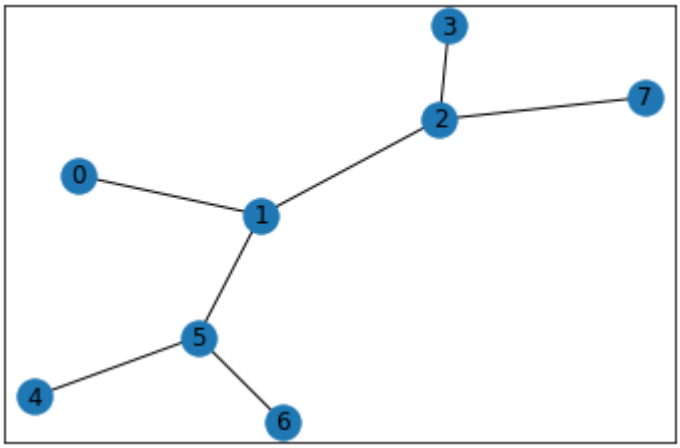


Figure 4- The figure of networkx graph

In this graph, the points-list map was represented each direction the bot could be taken. According to the above graph was visualized everything with networkx graphs instead of the format was allowed to show complexity. The starting point was 0, the final goal point is 7 and other extra added points were connected in between them. The figure of the networkx graph was shown that the best efficient path was [0,1,2,7]. In the event was meshed a story into this pursuit. The bot was searching for honey, it was attempting to discover the hive and stay away from the processing factory. Therefore, at that point was made the rewards graph was which is the matrix version of the rundown of focuses map.

```
R

(0, 1)
(1, 5)
(5, 6)
(5, 4)
(1, 2)
(2, 3)
(2, 7)
matrix([[ -1.,   0.,  -1.,  -1.,  -1.,  -1.,  -1.,  -1.],
        [  0.,  -1.,   0.,  -1.,  -1.,   0.,  -1.,  -1.],
        [ -1.,   0.,  -1.,   0.,  -1.,  -1.,  -1., 100.],
        [ -1.,  -1.,   0.,  -1.,  -1.,  -1.,  -1.,  -1.],
        [ -1.,  -1.,  -1.,  -1.,  -1.,   0.,  -1.,  -1.],
        [ -1.,   0.,  -1.,  -1.,   0.,  -1.,   0.,  -1.],
        [ -1.,  -1.,  -1.,  -1.,  -1.,   0.,  -1.,  -1.],
        [ -1.,  -1.,   0.,  -1.,  -1.,  -1.,  -1., 100.]])
```

Figure 5- The result of the reward table

According to the above matrix, the x-axis was represented or where the bot was currently located, and the y-axis was the possible next action. Furthermore, it was introduced the framework to be the height and width of the focus points list such as 8 and initially gave all qualities too (- 1). Finally, the reward table was stated the goal point round trip which is `R[goal, goal]= 100`

```
max_value 0.0
0
```

Figure 6- The output of the Q-learning matrix was stated as the goal point round trip which is R[goal, goal]= 100

```
Score: 982.4755139390292
max_value 204.44743467681158
Score: 982.4755139390292
max_value 255.77435819315943
Score: 982.4755139390292
max_value 255.77435819315943
Score: 982.5185572596226
Trained Q matrix:
[[  0.          63.98870995   0.           0.           0.
    0.           0.           0.        ]
 [ 51.19096796   0.          79.98588744   0.           0.
   51.14792464   0.           0.        ]
 [  0.          63.98870995   0.          63.98870995   0.
    0.           0.         100.        ]
 [  0.           0.          80.           0.           0.
    0.           0.           0.        ]
 [  0.           0.           0.           0.           0.
   51.19096796   0.           0.        ]
 [  0.          63.98870995   0.           0.          40.91833971
    0.          40.95277437   0.        ]
 [  0.           0.           0.           0.           0.
   51.19096796   0.           0.        ]
 [  0.           0.          79.98588744   0.           0.
    0.           0.         100.        ]]
Most efficient path:
[0, 1, 2, 7]
```

Figure 7- The output result of trained Q-matrix and most efficient path

After training the Q matrix it could be found the maximum Q value for each column then using those values the most efficient path was found. 0->1->2->7

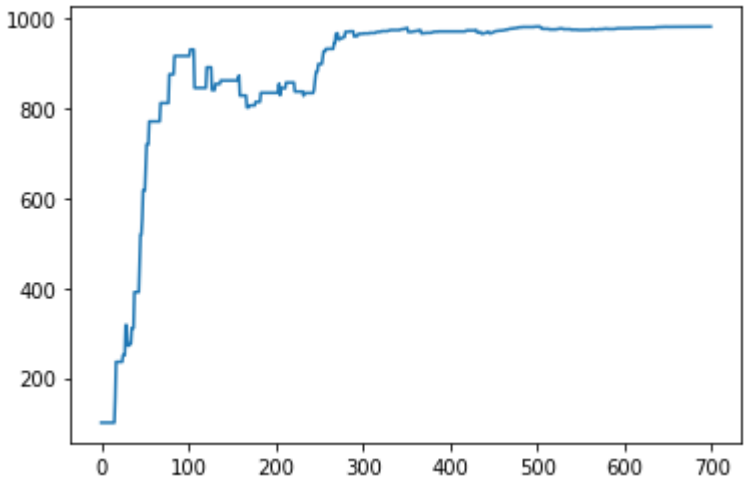


Figure 8-The graph of scores versus the number of iterations

This is the graph of the scores versus the number of iterations. This graph shows that the most efficient path is given at least having there are 400 iterations until it is converged to a solution. Therefore, this graph is very useful to find the minimum number of iterations for converging.

3.2 Part 02- Example 01

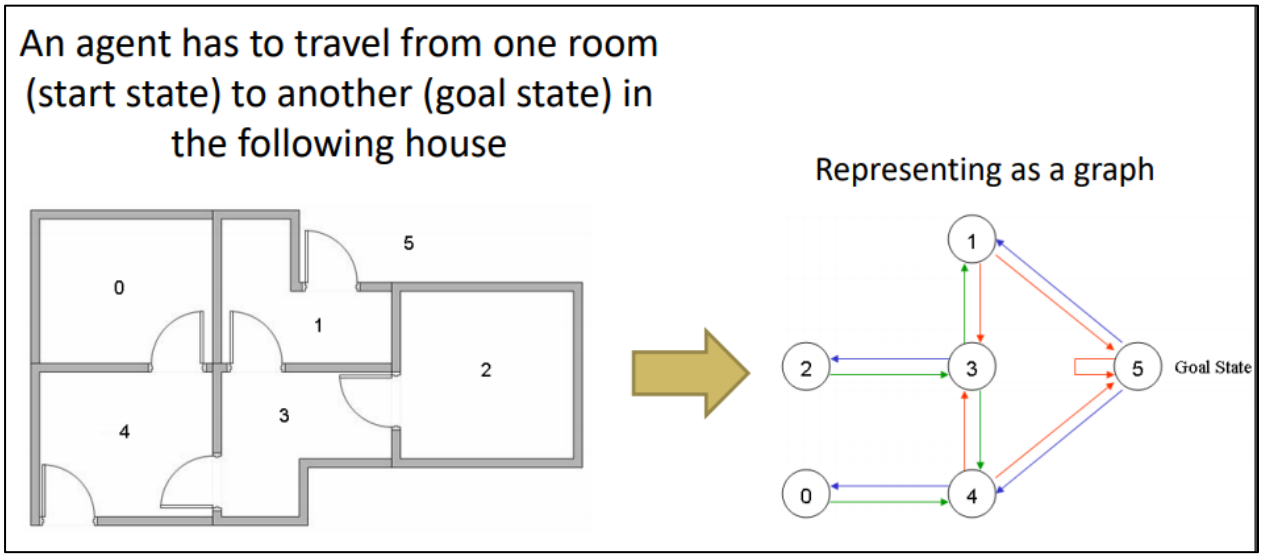


Figure 9-The graph of the path in example 02

According to this example was considered 2 was the starting state and 5 was the goal state. Then the available path would be found using this graph such as [2,3,1,5] and [2,3,4,5].

3.2.1 Python (Colab Notebook)

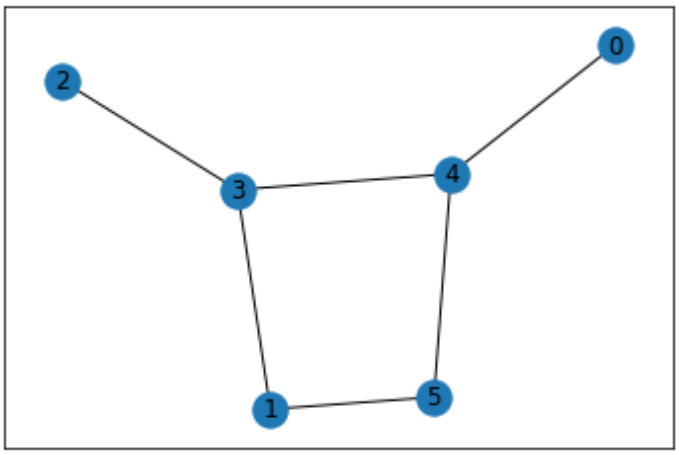


Figure 10- The figure of networkx graph for part 02

In this graph, the points-list map was represented each direction the room could be taken. According to the above graph was visualized everything with networkx graphs instead of the format was allowed to show complexity. In the event, an agent had to travel from one room (start state) to another (goal state) in the following house and it was attempting to discover the rooms and stay away from the processing house. Therefore, at that point was made the rewards graph was which is the matrix version of the rundown of focuses map. The starting point was 2, the final goal point is 5 and other extra added points were connected in between them. The figure of networkx graph was shown that the best efficient path was [2,3,1,5] or [2,3,4,5]. Therefore, when the agent came from room 2 to room 3 it was randomly selected room 3 or room 4. Finally, the agent reached the goal state of room 5.

```
R

(0, 4)
(4, 5)
(4, 3)
(3, 2)
(3, 1)
(1, 5)
matrix([[ -1.,  -1.,  -1.,  -1.,   0.,  -1.],
        [ -1.,  -1.,  -1.,   0.,  -1., 100.],
        [ -1.,  -1.,  -1.,   0.,  -1.,  -1.],
        [ -1.,   0.,   0.,  -1.,   0.,  -1.],
        [  0.,  -1.,  -1.,   0.,  -1., 100.],
        [ -1.,   0.,  -1.,  -1.,   0., 100.]])
```

```
max_value 0.0
0
```

Figure 11- The result of the reward table

According to the above matrix, the x-axis was represented or where the agent was currently located, and the y-axis was the possible next actions. Furthermore, it was introduced the

framework to be the height and width of the focus points list such as the (6X6) matrix because 0 to 5 rooms were included and initially gave all qualities too (- 1) for no linked rooms. Then 0 value was given to link rooms. Finally, the Reward table was stated the goal point round trip which is R[goal, goal]= 100

```
max_value 320.0
Score: 1007.2
max_value 500.0
Score: 1007.2
max_value 500.0
Score: 1007.2
max_value 400.0
Score: 1007.2
max_value 320.0
Score: 1007.2
max_value 256.0
Score: 1007.2
max_value 500.0
Score: 1007.2
Trained Q matrix:
[[  0.    0.    0.    0.   80.    0. ]
 [  0.    0.    0.   64.    0.  100. ]
 [  0.    0.    0.   64.    0.    0. ]
 [  0.   80.   51.2   0.   80.    0. ]
 [ 64.    0.    0.   64.    0.  100. ]
 [  0.   80.    0.    0.   80.  100. ]]
```

Figure 12- The output of the normalized the "trained" Q matrix in the collaboratory

According to the above figure of the trained Q-matrix was shown that each column had a unique maximum value and 0 values. Those maximum values were filtered to the descending order, [51.2->64->80->100].

```
Most efficient path:
[2, 3, 1, 5]
```

After analyzing the trained Q-matrix and networkx graph it could be examined the most efficient path [2->3->1->5].

The next page represents the graph of the scores versus the number of iterations. This graph shows that the most efficient path is given at least having there are 1000 iterations until it is converged to a solution. Therefore, this graph is very useful to find the minimum number of iterations for converging.

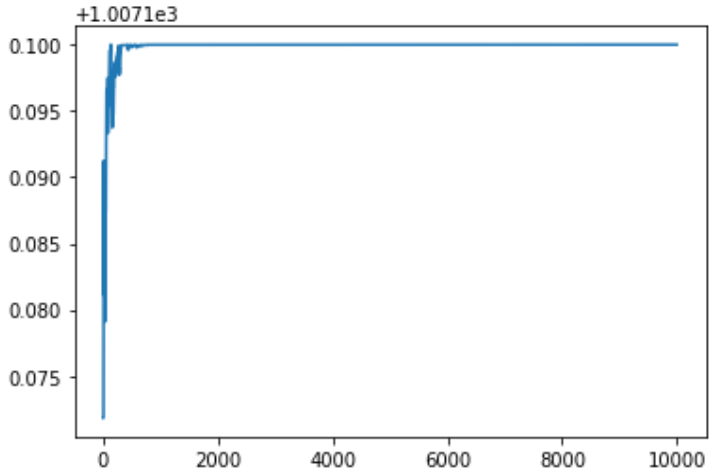


Figure 13- The graph of scores versus the number of iterations for part 02

3.2.2 Octave

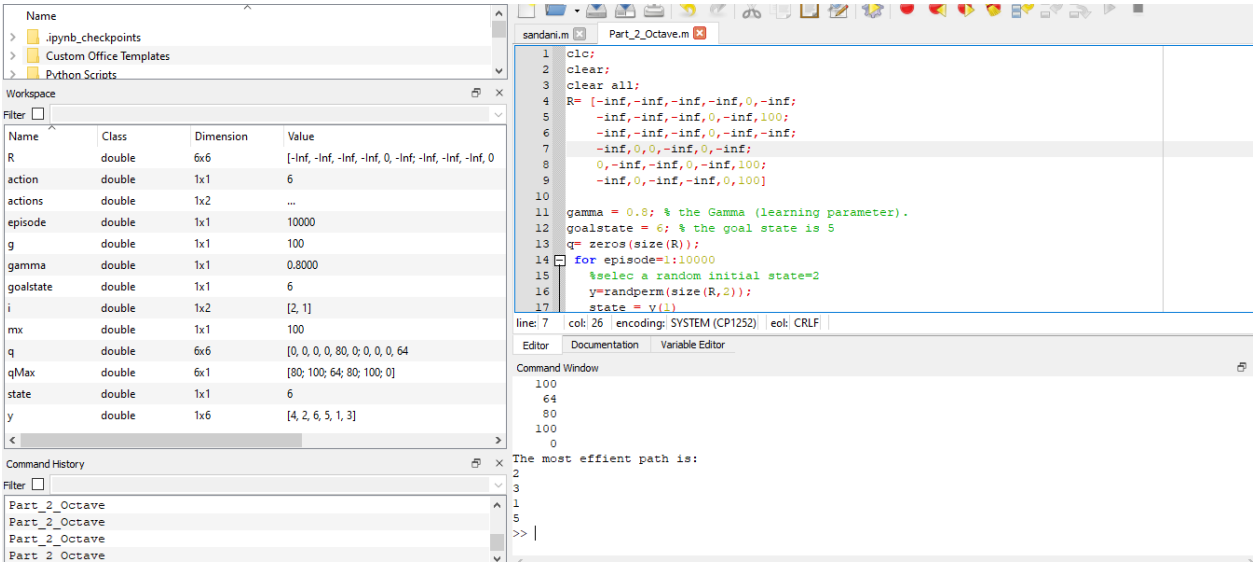


Figure 14- The working background of the octave

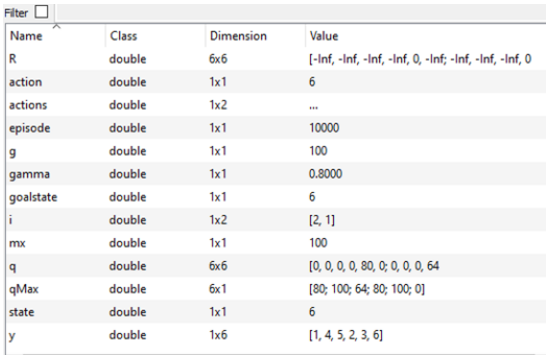


Figure 15- All variable was assigned for part 02

This figure shows the variables and dimensions. In this modelling, 6 points were defined as 0 to 5. Therefore, the Reward and Q matrix size was (6X6). The most efficient paths for each part were found by using each initial state and goal state.

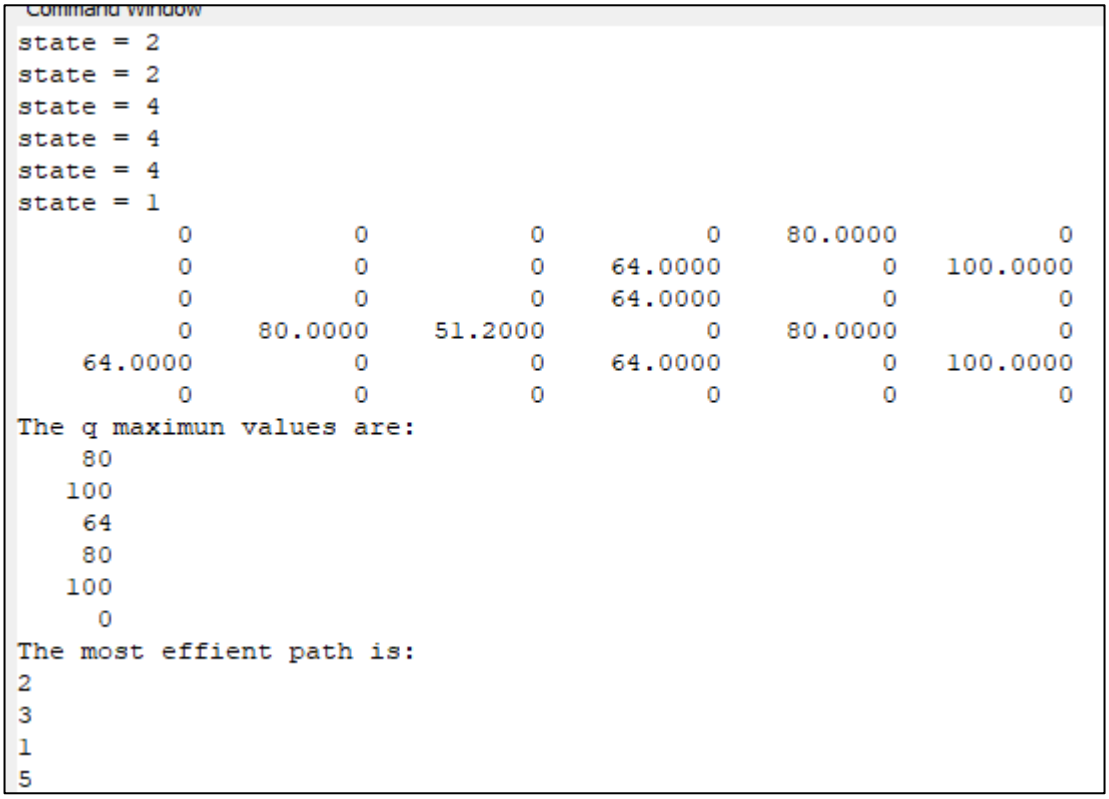


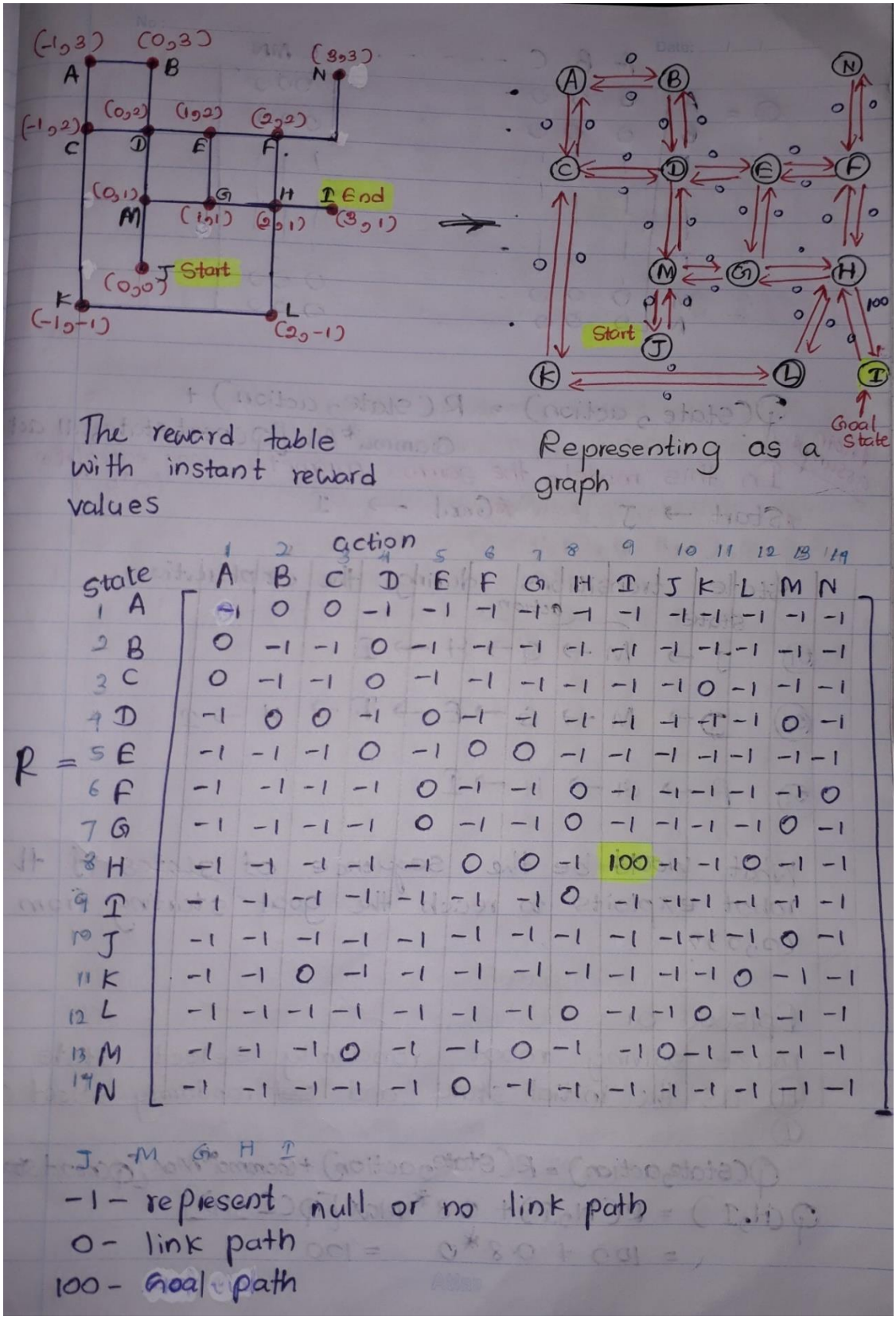
Figure 16- The output of the most efficient path and normalized the "trained" Q matrix in Octave software

According to the above figure of the trained Q- matrix was shown that each column had a unique maximum value and 0 values. But those outputs were given as row by row, therefore it was needed to get those maximum values column by column and they were filtered to the descending order, [51.2->64->80->100]. After analyzing the trained Q-matrix and networkx graph it could be examined the most efficient path [2->3->1->5].

3.2.3 The comparison of both result

Therefore, both the result and output of the python and octave were equal. Hence both results could be examined as the most efficient path [2->3->1->5]. When comparing both trained Q-table, it was clearly shown that both results were equal. Furthermore, the trained Q matrix has maximum values that were filtered to the descending order, and it was [51.2->64->80->100] in the python on google colab. And also, the descending order of the maximum values in the Q table for octave was [51.2->64->80->100]. This is because Colab notebook and Octave which lead to high precision and accuracy in the results obtained while analyzing outputs.

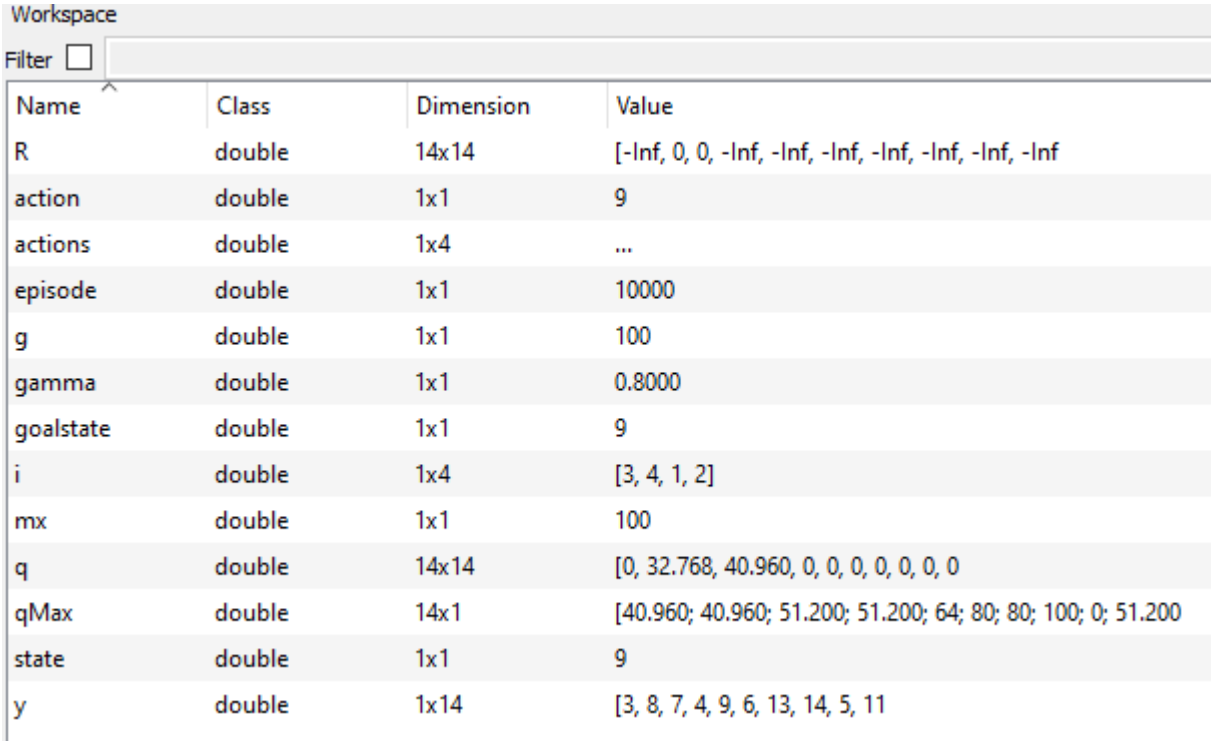
3.3 Part 03-Maze solving modelling



state	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1 A	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2 B	0	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3 C	0	-1	-1	0	-1	-1	-1	-1	-1	-1	0	-1	-1	-1
4 D	-1	0	0	-1	0	-1	-1	-1	-1	-1	-1	-1	0	-1
5 E	-1	-1	-1	0	-1	0	0	-1	-1	-1	-1	-1	-1	-1
6 F	-1	-1	-1	-1	0	-1	-1	0	-1	-1	-1	-1	-1	0
7 G	-1	-1	-1	-1	0	-1	-1	0	-1	-1	-1	-1	0	-1
8 H	-1	-1	-1	-1	-1	0	0	-1	100	-1	-1	0	-1	-1
9 I	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1
10 J	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1
11 K	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1
12 L	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	0	-1	-1	-1
13 M	-1	-1	-1	0	-1	-1	0	-1	-1	0	-1	-1	-1	-1
14 N	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1

Table 1-The table for points labelling

	Number	Points
A	1	(-1,3)
B	2	(0,3)
C	3	(-1,2)
D	4	(0,2)
E	5	(1,2)
F	6	(2,2)
G	7	(1,1)
H	8	(2,1)
I	9	(3,1)
J	10	(0,0)
K	11	(-1,-1)
L	12	(2,-1)
M	13	(0,1)
N	14	(3,3)



This figure shows the variables and dimensions of the maze solving model. In this modelling, 14 points were defined A to N. Therefore, Reward and Q matrix size were (14X14). The most efficient paths for each part was found by using each initial state and goal state.

1. **Start = (0, 0); Actions: UP, RIGHT, RIGHT, RIGHT**

Command Window

```
Command Window
state = 2
state = 4
state = 12
state = 8
state = 14
state = 7
state = 8
state = 8
state = 4
state = 14
state = 11
state = 7
state = 3
state = 4
state = 12
state = 2
state = 12
state = 5
state = 13
state = 8
state = 12
state = 3
 Columns 1 through 9:

         0    32.7680    40.9600         0         0         0         0         0         0
   32.7680          0         0    40.9600         0         0         0         0         0
   32.7680          0         0    40.9600         0         0         0         0         0
         0    32.7680    40.9600         0   51.2000         0         0         0         0
         0          0         0    40.9600         0   64.0000   64.0000         0         0
         0          0         0         0   51.2000         0         0   80.0000         0
         0          0         0         0   51.2000         0         0   80.0000         0
         0          0         0         0         0   64.0000   64.0000         0  100.0000
         0          0         0         0         0         0         0         0         0
         0          0         0         0         0         0         0         0         0
         0          0   40.9600         0         0         0         0         0         0
         0          0         0         0         0         0         0   80.0000         0
         0          0         0   40.9600         0         0   64.0000         0         0
         0          0         0         0         0   64.0000         0         0         0

    Columns 10 through 14:

         0          0         0         0         0
         0          0         0         0         0
         0    51.2000         0         0         0
         0          0         0   51.2000         0
         0          0         0         0         0
         0          0         0         0   51.2000
         0          0         0   51.2000         0
         0          0   64.0000         0         0
         0          0         0         0         0
         0          0         0   51.2000         0
         0          0   64.0000         0         0
         0    51.2000         0         0         0
   40.9600          0         0         0         0
         0          0         0         0         0
```

This is a Q-Learning Algorithm in Octave and its exploration would be done in random. After the 10000 episodes, the trained Q table was given like this.

```
The q maximun values are:
    40.9600
    40.9600
    51.2000
    51.2000
    64.0000
    80.0000
    80.0000
   100.0000
         0
    51.2000
    64.0000
    80.0000
    64.0000
    64.0000
The most effient path is:
10
13
7
8
9
>>
```

According to the above figure of the trained Q-matrix was shown that each column had a unique maximum value and 0 values. But those outputs were given as row by row, therefore it was needed to get those maximum values column by column and they were filtered to the descending order, [40.96->51.2->64->80->100]. After analyzing the trained Q-matrix and networkx graph it could be examined the most efficient path [10->13->7->8->9].

The most efficient path: (0,0)->(0,1)->(1,1)->(2,1)->(3,1)

Actions: UP, RIGHT, RIGHT, RIGHT

2. **Start = (0, 2), Actions: DOWN, RIGHT, UP, RIGHT, DOWN, RIGHT**

This is a Q-Learning Algorithm in Octave and its exploration would be done in random. After the 10000 episodes, the trained Q table was given like this. According to the above figure of the trained Q-matrix was shown that each column had a unique maximum value and 0 values. But those outputs were given as row by row, therefore it was needed to get those maximum values column by column and they were filtered to the descending order, [40.96->51.2->64->80->100]. After analyzing the trained Q-matrix and networkx graph it could be examined the most efficient path [4->5->6->8->9].

The most efficient path: (0,2)->(1,2)->(2,2)->(2,1)->(3,1)

Actions: RIGHT, RIGHT, DOWN, RIGHT

```
state = 11
state = 10
state = 14
state = 14
state = 8
         0   32.7680   40.9600         0         0         0         0         0         0         0         0         0         0         0
   32.7680         0         0   40.9600         0         0         0         0         0         0         0         0         0         0
   32.7680         0         0   40.9600         0         0         0         0         0         0   51.2000         0         0         0
         0   32.7680   40.9600         0   51.2000         0         0         0         0         0         0         0   51.2000         0
         0         0         0   40.9600         0   64.0000   64.0000         0         0         0         0         0         0         0
         0         0         0         0   51.2000         0         0   80.0000         0         0         0         0         0   51.2000
         0         0         0         0   51.2000         0         0   80.0000         0         0         0         0   51.2000         0
         0         0         0         0         0   64.0000   64.0000         0  100.0000         0         0   64.0000         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0   51.2000         0
         0         0   40.9600         0         0         0         0         0         0         0         0   64.0000         0         0
         0         0         0         0         0         0         0   80.0000         0         0   51.2000         0         0         0
         0         0         0   40.9600         0         0   64.0000         0         0   40.9600         0         0         0         0
         0         0         0         0         0   64.0000         0         0         0         0         0         0         0         0
The q maximun values are:
   40.9600
   40.9600
   51.2000
   51.2000
   64.0000
   80.0000
   80.0000
  100.0000
         0
   51.2000
   64.0000
   80.0000
   64.0000
   64.0000
The most effient path is:
4
5
6
8
9
>>
```

3. Start = (1, 2); Actions: DOWN,RIGHT, RIGHT

```
Command Window
state = 12
state = 9
state = 6
state = 8
state = 1
state = 14
         0   32.7680   40.9600         0         0         0         0         0         0         0         0         0         0         0
   32.7680         0         0   40.9600         0         0         0         0         0         0         0         0         0         0
   32.7680         0         0   40.9600         0         0         0         0         0         0   51.2000         0         0         0
         0   32.7680   40.9600         0   51.2000         0         0         0         0         0         0         0   51.2000         0
         0         0         0   40.9600         0   64.0000   64.0000         0         0         0         0         0         0         0
         0         0         0         0   51.2000         0         0   80.0000         0         0         0         0         0   51.2000
         0         0         0         0   51.2000         0         0   80.0000         0         0         0         0   51.2000         0
         0         0         0         0         0   64.0000   64.0000         0  100.0000         0         0   64.0000         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0   51.2000         0
         0         0   40.9600         0         0         0         0         0         0         0         0   64.0000         0         0
         0         0         0         0         0         0         0   80.0000         0         0   51.2000         0         0         0
         0         0         0   40.9600         0         0   64.0000         0         0   40.9600         0         0         0         0
         0         0         0         0         0   64.0000         0         0         0         0         0         0         0         0
The q maximun values are:
   40.9600
   40.9600
   51.2000
   51.2000
   64.0000
   80.0000
   80.0000
  100.0000
         0
   51.2000
   64.0000
   80.0000
   64.0000
   64.0000
The most effient path is:
5
6
8
9
```

This is a Q-Learning Algorithm in Octave and its exploration would be done in random. After the 10000 episodes, the trained Q table was given like this. According to the above figure of the trained Q, a matrix was shown that each column had a unique maximum value and 0 values. But those outputs were given as row by row, therefore it was needed to get those maximum values column by column and they were filtered to the descending order, [51.2->64->80->100]. After

analyzing the trained Q-matrix and networkx graph it could be examined the most efficient path [5->6->8->9].

The most efficient path: (1,2)->(2,2)->(2,1)->(3,1)

Actions: RIGHT, DOWN, RIGHT

Now, what would be the sequence of states if the robot exploits to reach the goal starting from (0, 0)?

Sequence 01:

The most efficient path: (0,0)->(0,1)->(1,1)->(2,1)->(3,1)

Actions: UP, RIGHT, RIGHT, RIGHT

Sequence 02:

(0,0)->(0,1)->(0,2)->(1,2)->(1,1)->(2,1)->(3,1)

Actions: UP, UP, RIGHT, DOWN, RIGHT, RIGHT

Sequence 03:

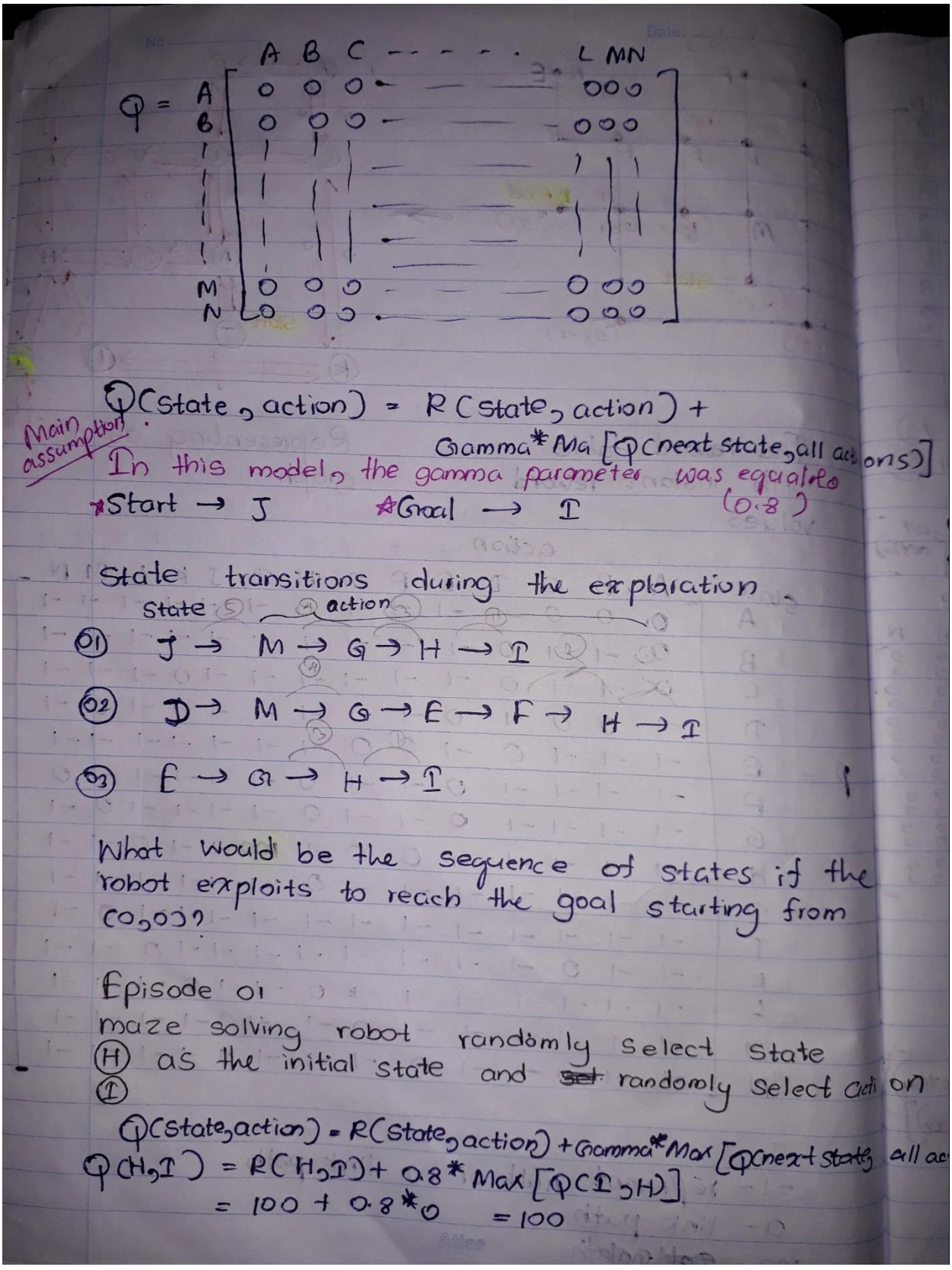
(0,0)->(0,1)->(1,1)->(1,2)->(2,2)->(2,1)->(3,1)

Actions: UP, RIGHT, UP, RIGHT, DOWN, RIGHT

Sequence 04:

(0,0)->(0,1)->(0,2)->(1,2)->(2,2)->(2,1)->(3,1)

Actions: UP, UP, RIGHT, RIGHT, DOWN, RIGHT



Q = | | A | B | C | ---- | L | MN |
|---|---|---|---|---|---|---|
| A | 0 | 0 | 0 | ---- | 0 | 0 0 |
| B | 0 | 0 | 0 | ---- | 0 | 0 0 |
| | | | | ---- | | |
| M | 0 | 0 | 0 | ---- | 0 | 0 0 |
| N | 0 | 0 | 0 | ---- | 0 | 0 0 |

Q(State, action) = R(State, action) + Gamma * Ma [Q(next state, all actions)]

Main assumption

In this model, the gamma parameter was equal to (0.8)

* Start → J * Goal → I

State transitions during the explaration

State → action

(01) J → M → G → H → I

(02) D → M → G → E → F → H → I

(03) E → G → H → I

What would be the sequence of states if the robot exploits to reach the goal starting from (0,0)?

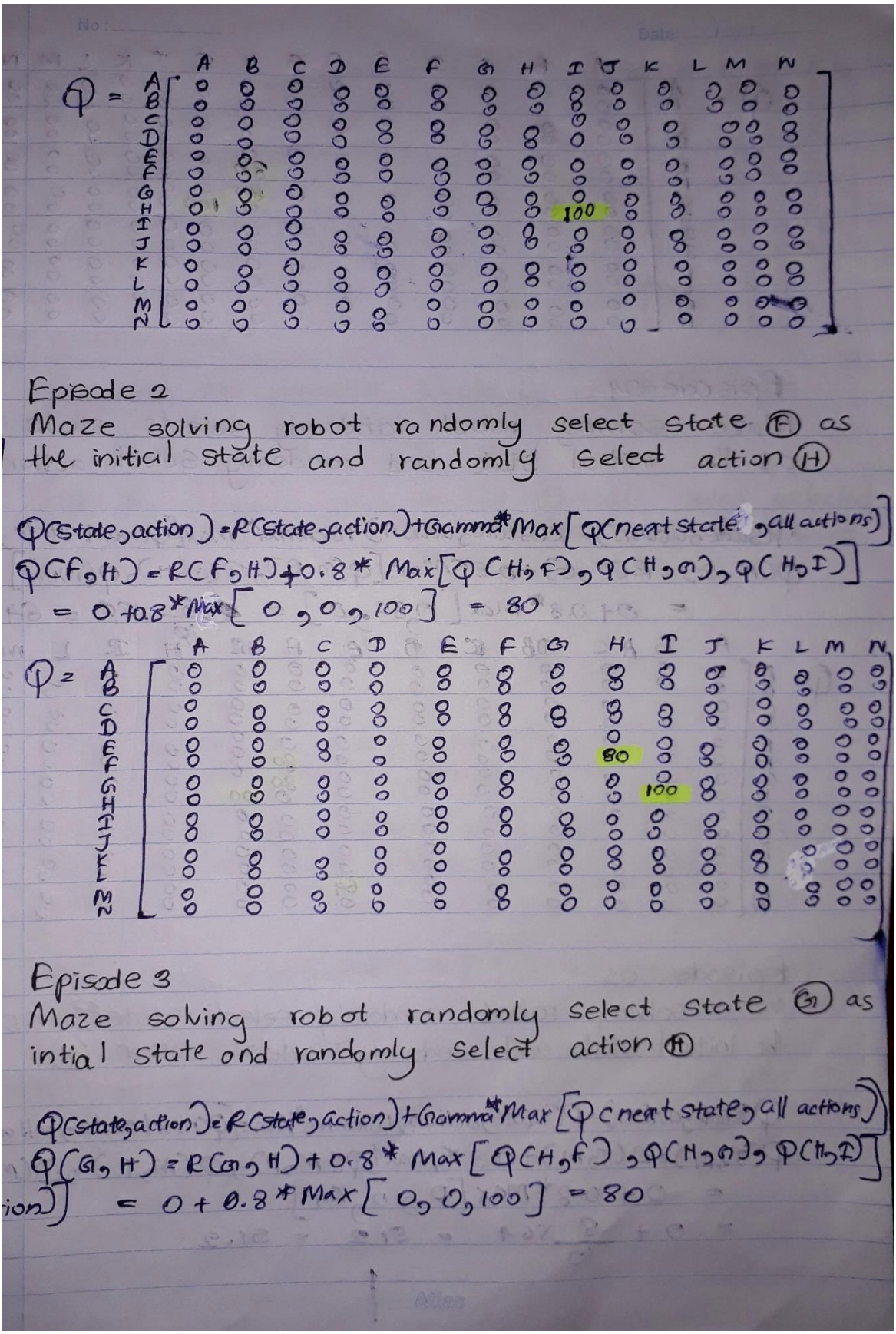
Episode 01

maze solving robot randomly select State (H) as the initial state and ~~set~~ randomly select action (I)

Q(State, action) = R(State, action) + Gamma * Max [Q(next state, all ac

Q(H, I) = R(H, I) + 0.8 * Max [Q(I, H)]

= 100 + 0.8 * 0 = 100



Q =

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	100	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Episode 2

Maze solving robot randomly select state (F) as the initial state and randomly select action (H)

$Q(State, action) = R(State, action) + Gamma * Max[Q(next\ state, all\ actions)]$

$Q(F, H) = R(F, H) + 0.8 * Max[Q(H, F), Q(H, G), Q(H, I)]$

$= 0 + 0.8 * Max[0, 0, 100] = 80$

Q =

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	80	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	100	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	0

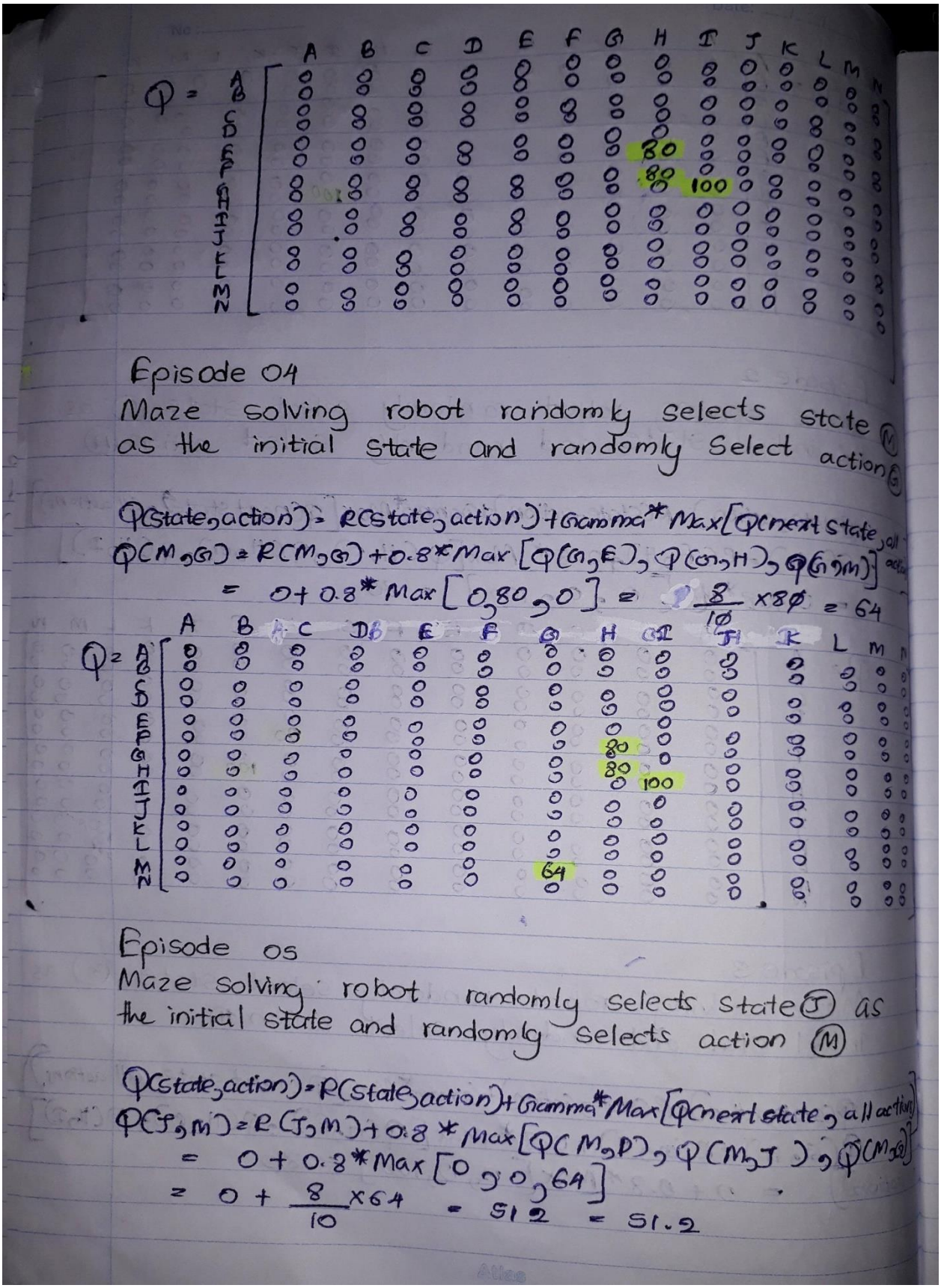
Episode 3

Maze solving robot randomly select state (G) as intial state and randomly select action (H)

$Q(State, action) = R(State, action) + Gamma * Max[Q(next\ state, all\ actions)]$

$Q(G, H) = R(G, H) + 0.8 * Max[Q(H, F), Q(H, G), Q(H, I)]$

$= 0 + 0.8 * Max[0, 0, 100] = 80$



Q =

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	80	0	0	0	0	0	0
F	0	0	0	0	0	0	0	80	100	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Episode 04

Maze solving robot randomly selects state (M) as the initial state and randomly Select action (G)

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all action)]

Q(M, G) = R(M, G) + 0.8 * Max[Q(G, E), Q(G, H), Q(G, M)]

= 0 + 0.8 * Max[0, 80, 0] = $\frac{8}{10}$ x 80 = 64

Q =

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	80	0	0	0	0	0	0
G	0	0	0	0	0	0	0	80	100	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	64	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Episode 05

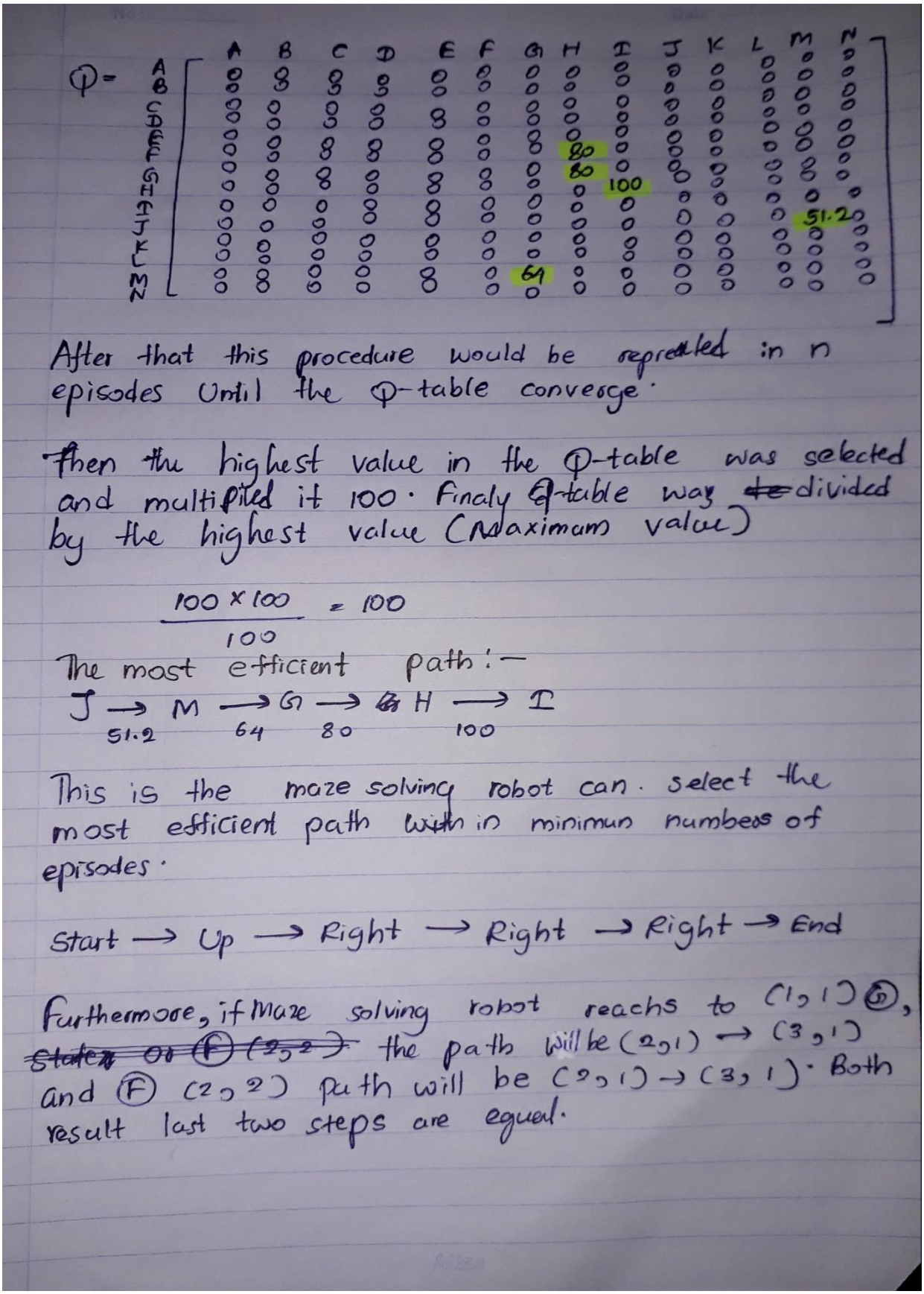
Maze solving robot randomly selects state (J) as the initial state and randomly selects action (M)

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all action)]

Q(J, M) = R(J, M) + 0.8 * Max[Q(M, P), Q(M, J), Q(M, G)]

= 0 + 0.8 * Max[0, 0, 64]

= 0 + $\frac{8}{10}$ x 64 = 512 = 51.2



After that this procedure would be repeated in n episodes Until the Φ-table converge.

Then the highest value in the Φ-table was selected and multiplied it 100. Finaly Q-table way ~~te~~ divided by the highest value (Maximum value)

$$\frac{100 \times 100}{100} = 100$$

The most efficient path :-

J → M → G → H → I
51.2 64 80 100

This is the maze solving robot can. select the most efficient path with in minimun numbers of episodes.

Start → Up → Right → Right → Right → End

Furthermore, if maze solving robot reachs to (1,1) Ⓖ, ~~state of Ⓕ (2,2)~~ the path will be (2,1) → (3,1) and Ⓕ (2,2) path will be (2,1) → (3,1). Both result last two steps are equal.

4 REFERENCES

- Analytics Vidhya. Introduction to Reinforcement Learning for Beginners. [online] Available at: https://www.analyticsvidhya.com/blog/2021/02/introduction-to-reinforcement-learning-for-beginners/ [Accessed 14 Augt.2021].
- ViralML.com. Reinforcement Learning - A Simple Python Example and A Step Closer to AI with Assisted Q-Learning. [online] Available at: https://www.viralml.com/video-content.html?v=nSxaG_Kjw_w [Accessed 14 Aug.2021].

5 Appendix

5.1 Part 01-

https://colab.research.google.com/drive/1S3QY83AeK6ayDDS15D3vq30IsuuDgDYT?usp=sharing

5.2 Part 02-

https://colab.research.google.com/drive/1G-1WhtMyppuJipjaSedx2jOctqz4wrjv?usp=sharing

https://drive.google.com/file/d/1p2GTVH0zYzFKCKoIcNeGf1ggABGvWbnN/view?usp=sharing

5.3 Part 03-

https://drive.google.com/file/d/1Tent5JDhhw2As9I3yNGeb7DoWe3p6xsu/view?usp=sharing