Index no: 18000045
S.A.T.N.Abeysinghe

I have explained the code and the algorithms better in the provided links of google colab.

Question 1)

https://colab.research.google.com/drive/1_aJzWoh67VfGdi_X1yXDMw_YlmyxIqF8?usp=sharing

```
import numpy as np
import pylab as plt
points_list = [(0,1), (1,5), (5,6), (5,4), (1,2), (2,3), (2,7)]
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()
MATRIX_SIZE = 8

R = np.matrix(np.ones(shape=(MATRIX_SIZE, MATRIX_SIZE)))
R *= -1
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:
        R[point[::-1]]= 0
R[goal,goal]= 100
```

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

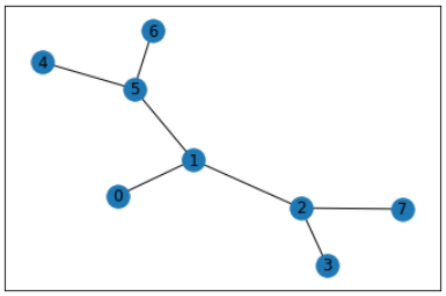
```
    scores.append(score)
    print ('Score:', str(score))
print("Trained Q matrix:")
print(Q/np.max(Q)*100)
current_state = 0
steps = [current_state]


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()
```

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

```
[22] points_list = [(0,1), (1,5), (5,6), (5,4), (1,2), (2,3), (2,7)]
```

```
[23] 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()
```

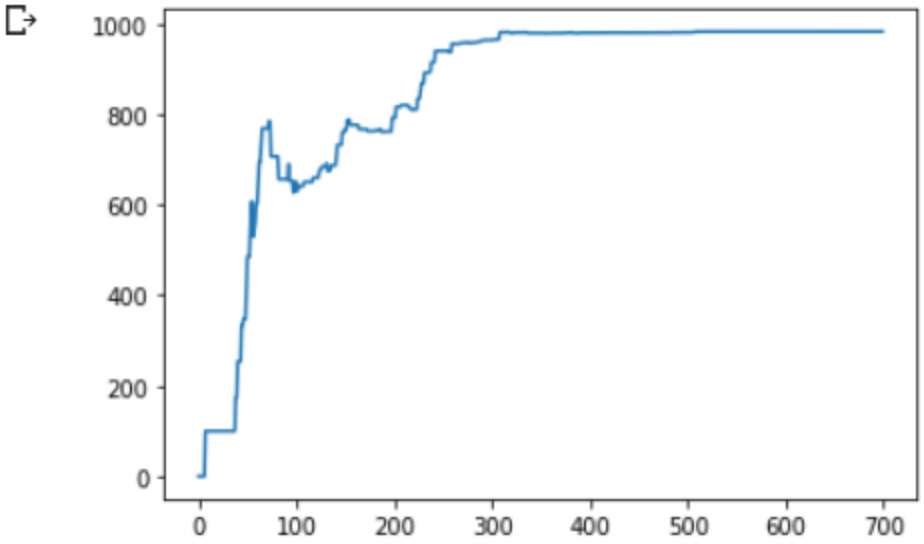


```
[38]
        steps.append(next_step_index)
        current_state = next_step_index
```

```
[39] print("Most efficient path:")
     print(steps)
```

```
Most efficient path:
[0, 1, 2, 7]
```

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



Q2)

https://colab.research.google.com/drive/1qN6YgA3XXxFvyb73YO4uzjIHjMXJOD2y?usp=sharing

```
import numpy as np
import pylab as plt
points_list = [(1,3), (1,5), (2,3), (3,4), (4,0), (4,5)]
```

```
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()
MATRIX_SIZE = 6

R = np.matrix(np.ones(shape=(MATRIX_SIZE, MATRIX_SIZE)))
R *= -1
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:
        R[point[::-1]]= 0
R[goal,goal]= 100
R
Q = np.matrix(np.zeros([MATRIX_SIZE,MATRIX_SIZE]))
gamma = 0.75

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)
available_act
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)
action
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)
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)
current_state = 0
steps = [current_state]


while current_state != 5:
```

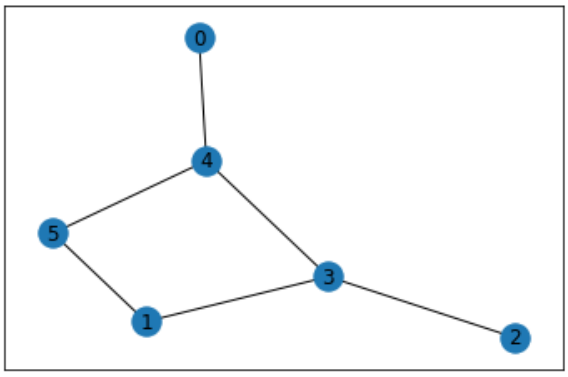
```
    next_step_index = np.where(Q[current_state,]
        == np.max(Q[current_state,]))[1]

    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()
```

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

```
[88] points_list = [(1,3), (1,5), (2,3), (3,4), (4,0), (4,5)]
```

```
[89] 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()
```




```
[91] MATRIX_SIZE = 6

     R = np.matrix(np.ones(shape=(MATRIX_SIZE, MATRIX_SIZE)))
     R *= -1
```

```
[94] R
```

```
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.]])
```

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

```
gamma = 0.75

initial_state = 1
```

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

```
[98] available_act = available_actions(initial_state)
     available_act
```

```
array([3, 5])
```

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

```
[100] action = sample_next_action(available_act)
      action
```

```
5
```

[101]
```
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)
```

[102]
```
update(initial_state, action, gamma)
```

```
max_value 100.0
100.0
```

```
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))
```

```
max_value 224.99729729273 78
Score: 942.1854729452043
max_value 399.99519518708945
Score: 942.1854729452043
max_value 399.99519518708945
Score: 942.1854729452043
max_value 224.9972972927378
Score: 942.1863988838147
max_value 224.9972972927378
Score: 942.1863988838147
max_value 224.9972972927378
Score: 942.1863988838147
max_value 399.99519518708945
Score: 942.1863988838147
max_value 299.9963963903171
Score: 942.1863988838147
max_value 399.99519518708945
Score: 942.1867992897003
max_value 224.9972972927378
Score: 942.1867992897003
max_value 299.9963963903171
Score: 942.1867992897003
max_value 168.74797296955336
```

```
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))
```

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

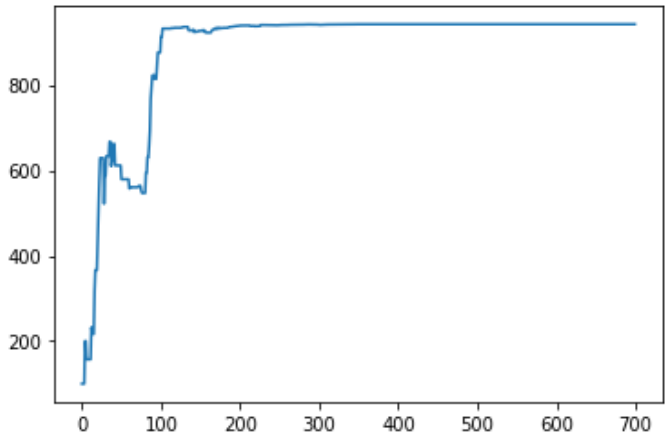
```
Trained Q matrix:
[[  0.           0.           0.           0.          74.99977829
    0.        ]
 [  0.           0.           0.          56.24983372   0.
  100.        ]
 [  0.           0.           0.          56.24983372   0.
    0.        ]
 [  0.          74.99938415  42.18715358   0.          74.99977829
    0.        ]
 [ 56.24953811   0.           0.          56.24983372   0.
   99.99970439]
 [  0.          75.           0.           0.          74.99938415
  100.        ]]
```

```
current_state = 0
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
```

```
[106] print("Most efficient path:")
      print(steps)
```

```
Most efficient path:
[0, 4, 5]
```

```
[107] plt.plot(scores)
      plt.show()
```



Q3)

https://colab.research.google.com/drive/1om0bHfuz-ESQUdblefG-wJngEQ11p-vM?usp=sharing

```
import numpy as np
import pylab as plt

import random as random
```

```
Rtable = {(-1,3):{'down': 0, 'left': -1, 'right': 0, 'up':-1},
          (0,3):{'down': 0, 'left': 0, 'right': -1, 'up':-1},
          (3,3):{'down': 0, 'left': -1, 'right': -1, 'up':-1},
          (-1,2):{'down': 0, 'left': -1, 'right': 0, 'up':0},
          (0,2):{'down': 0, 'left': 0, 'right': 0, 'up':0},
          (1,2):{'down': 0, 'left': 0, 'right': 0, 'up':-1},
          (2,2):{'down': 0, 'left': 0, 'right': 0, 'up':-1},
          (3,2):{'down': -1, 'left': 0, 'right': -1, 'up':0},
          (0,0):{'down': -1, 'left': -1, 'right': -1, 'up':0},
          (0,1):{'down': 0, 'left': -1, 'right': 0, 'up':0},
          (1,1):{'down': -1, 'left': 0, 'right': 0, 'up':0},
          (2,1):{'down': 0, 'left': 0, 'right': 100, 'up':0},
          (3,1):{'down': -1, 'left': 0, 'right': -1, 'up':-1},
          (-1,-1):{'down': -1, 'left': -1, 'right': 0, 'up':0},
          (2,-1):{'down': -1, 'left': 0, 'right': -1, 'up':0}}
```

```
Qtable = {(-1,3):{'down': 0, 'left': 0, 'right': 0, 'up':0},
```

```
            (0,3):{'down': 0, 'left': 0, 'right': 0, 'up':0},
            (3,3):{'down': 0, 'left': 0, 'right': 0, 'up':0},
            (-1,2):{'down': 0, 'left': 0, 'right': 0, 'up':0},
            (0,2):{'down': 0, 'left': 0, 'right': 0, 'up':0},
            (1,2):{'down': 0, 'left': 0, 'right': 0, 'up':0},
            (2,2):{'down': 0, 'left': 0, 'right': 0, 'up':0},
            (3,2):{'down': 0, 'left': 0, 'right': 0, 'up':0},
            (0,0):{'down': 0, 'left': 0, 'right': 0, 'up':0},
            (0,1):{'down': 0, 'left': 0, 'right': 0, 'up':0},
            (1,1):{'down': 0, 'left': 0, 'right': 0, 'up':0},
            (2,1):{'down': 0, 'left': 0, 'right': 0, 'up':0},
            (3,1):{'down': 0, 'left': 0, 'right': 0, 'up':0},
            (-1,-1):{'down': 0, 'left': 0, 'right': 0, 'up':0},
            (2,-1):{'down': 0, 'left': 0, 'right': 0, 'up':0}}


gamma = 0.75

initial_state = (-1,3)


def available_actions(state):
  current_state_row=Rtable.get(state)
  valid_actions =[]
  for key,value in current_state_row.items():
    if value >= 0:
     # print(key)
      valid_actions.append(key)
  return valid_actions
  #print (current_state_row)
  #print(valid_actions)

available_act = available_actions(initial_state)
available_act


def sample_next_action(available_actions_range):
    next_action = random.choice(available_actions_range)
    return next_action
```

```
paths = {(-1,3):{'down': (-1,2), 'left':-1, 'right': (0,3), 'up':-1},
          (0,3):{'down': (0,2), 'left': (-1,3), 'right':-1, 'up':-1},
          (3,3):{'down': (3,2), 'left': -1, 'right': -1, 'up':-1},
          (-1,2):{'down': (-1,-1), 'left': -1, 'right': (0,2),
'up':(-1,3)},
          (0,2):{'down': (0,1), 'left': (-1,2), 'right': (1,2),
'up':(0,3)},
          (1,2):{'down': (1,1), 'left': (0,2), 'right': (2,2), 'up':-1},
          (2,2):{'down': (2,1), 'left': (1,2), 'right': (3,2), 'up':-1},
          (3,2):{'down': -1, 'left': (2,2), 'right': -1, 'up':(3,3)},
          (0,0):{'down': -1, 'left': -1, 'right': -1, 'up':(0,1)},
          (0,1):{'down': (0,0), 'left': -1, 'right': (1,1), 'up':(0,2)},
          (1,1):{'down': -1, 'left': (0,1), 'right': (2,1), 'up':(1,2)},
          (2,1):{'down': (2,-1), 'left': (1,1), 'right': (3,1),
'up':(2,2)},
          (3,1):{'down': -1, 'left': (2,1), 'right': -1, 'up':-1},
          (-1,-1):{'down': -1, 'left': -1, 'right': (2,-1), 'up':(-1,2)},
          (2,-1):{'down': -1, 'left': (-1,-1), 'right': -1, 'up':(2,1)}}



c = (2,-1)
a = 'left'
xx = paths.get(  c ).get(a)
print(xx)


def update(current_state, action, gamma):

  action_state = paths.get(  current_state ).get(action)
  print(action_state)
  action_state_row_of_Qtable= Qtable.get(action_state)

  max_value = max(action_state_row_of_Qtable.values())
```

```
  max_keys = [k for k, v in action_state_row_of_Qtable.items() if v ==
max_value]

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

  if len(max_keys) > 1:
      max_key = random.choice (max_keys)
  else:
      max_key = random.choice (max_keys)
      #max_index = int(max_index)
  #max_value = Qtable[action_state, max_key]
  max_value = Qtable.get(action_state).get(max_key)

  #Qtable[current_state, action] = Rtable[current_state, action] + gamma *
max_value
  Qtable[current_state][action] = Rtable[current_state][ action] + gamma *
max_value
  print('max_value', Rtable[current_state][action] + gamma * max_value)

  maxx=0;
  return 0

 # for key,value in Qtable.items():
 #   for k,v in value.items():
 #     if v>maxx:
 #       maxx=v

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


update(initial_state, action, gamma)


KeyList = list(Qtable.keys())
KeyList
```

```
scores = []

for i in range(700):
    current_state = random.choice(KeyList)
    available_act = available_actions(current_state)
    action = sample_next_action(available_act)

    print ('Score:', str(score))
    score  = update(current_state,action,gamma)
   # scores.append(score)


Qtable


current_state = (-1,3)
steps = [current_state]
direction = []


while current_state != (3,1):

    current_row = Qtable.get(current_state)
    max_value = max(current_row.values())
    max_keys = [k for k, v in current_row.items() if v == max_value]

    if len(max_keys)>1 :
      max_key = random.choice (max_keys)
    else:
      max_key = random.choice (max_keys)# ekai tiyenne eka enawa

    next_state = paths.get(  current_state ).get(max_key)
    steps.append(next_state)
    direction.append(max_key)
    current_state = next_state


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

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

      import random as random
```

```
Rtable = {(-1,3):{'down': 0, 'left': -1, 'right': 0, 'up':-1},
          (0,3):{'down': 0, 'left': 0, 'right': -1, 'up':-1},
          (3,3):{'down': 0, 'left': -1, 'right': -1, 'up':-1},
          (-1,2):{'down': 0, 'left': -1, 'right': 0, 'up':0},
          (0,2):{'down': 0, 'left': 0, 'right': 0, 'up':0},
          (1,2):{'down': 0, 'left': 0, 'right': 0, 'up':-1},
          (2,2):{'down': 0, 'left': 0, 'right': 0, 'up':-1},
          (3,2):{'down': -1, 'left': 0, 'right': -1, 'up':0},
          (0,0):{'down': -1, 'left': -1, 'right': -1, 'up':0},
          (0,1):{'down': 0, 'left': -1, 'right': 0, 'up':0},
          (1,1):{'down': -1, 'left': 0, 'right': 0, 'up':0},
          (2,1):{'down': 0, 'left': 0, 'right': 100, 'up':0},
          (3,1):{'down': -1, 'left': 0, 'right': -1, 'up':-1},
          (-1,-1):{'down': -1, 'left': -1, 'right': 0, 'up':0},
          (2,-1):{'down': -1, 'left': 0, 'right': -1, 'up':0}}
```

```
[524] Qtable = {(-1,3):{'down': 0, 'left': 0, 'right': 0, 'up':0},
          (0,3):{'down': 0, 'left': 0, 'right': 0, 'up':0},
          (3,3):{'down': 0, 'left': 0, 'right': 0, 'up':0},
          (-1,2):{'down': 0, 'left': 0, 'right': 0, 'up':0},
          (0,2):{'down': 0, 'left': 0, 'right': 0, 'up':0},
          (1,2):{'down': 0, 'left': 0, 'right': 0, 'up':0},
          (2,2):{'down': 0, 'left': 0, 'right': 0, 'up':0},
          (3,2):{'down': 0, 'left': 0, 'right': 0, 'up':0},
          (0,0):{'down': 0, 'left': 0, 'right': 0, 'up':0},
          (0,1):{'down': 0, 'left': 0, 'right': 0, 'up':0},
          (1,1):{'down': 0, 'left': 0, 'right': 0, 'up':0},
```

```
[525] gamma = 0.75

      initial_state = (-1,3)
```

```
      def available_actions(state):
        current_state_row=Rtable.get(state)
        valid_actions =[]
        for key,value in current_state_row.items():
          if value >= 0:
           # print(key)
            valid_actions.append(key)
        return valid_actions
        #print (current_state_row)
        #print(valid_actions)
```

```
[527] available_act = available_actions(initial_state)
      available_act
```

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

```
[528] def sample_next_action(available_actions_range):
          next_action = random.choice(available_actions_range)
          return next_action
```

```
[529] action = sample_next_action(available_act)
      action
```

```
'right'
```

```
[528] def sample_next_action(available_actions_range):
          next_action = random.choice(available_actions_range)
          return next_action
```

```
      action = sample_next_action(available_act)
      action
```

```
      'right'
```

```
[530] paths = {(-1,3):{'down': (-1,2), 'left':-1, 'right': (0,3), 'up':-1},
               (0,3):{'down': (0,2), 'left': (-1,3), 'right':-1, 'up':-1},
               (3,3):{'down': (3,2), 'left': -1, 'right': -1, 'up':-1},
               (-1,2):{'down': (-1,-1), 'left': -1, 'right': (0,2), 'up':(-1,3)},
               (0,2):{'down': (0,1), 'left': (-1,2), 'right': (1,2), 'up':(0,3)},
               (1,2):{'down': (1,1), 'left': (0,2), 'right': (2,2), 'up':-1},
               (2,2):{'down': (2,1), 'left': (1,2), 'right': (3,2), 'up':-1},
               (3,2):{'down': -1, 'left': (2,2), 'right': -1, 'up':(3,3)},
               (0,0):{'down': -1, 'left': -1, 'right': -1, 'up':(0,1)},
               (0,1):{'down': (0,0), 'left': -1, 'right': (1,1), 'up':(0,2)},
               (1,1):{'down': -1, 'left': (0,1), 'right': (2,1), 'up':(1,2)},
               (2,1):{'down': (2,-1), 'left': (1,1), 'right': (3,1), 'up':(2,2)},
               (3,1):{'down': -1, 'left': (2,1), 'right': -1, 'up':-1},
               (-1,-1):{'down': -1, 'left': -1, 'right': (2,-1), 'up':(-1,2)},
               (2,-1):{'down': -1, 'left': (-1,-1), 'right': -1, 'up':(2,1)}}
```

```
[531] c = (2,-1)
      a = 'left'
      xx = paths.get(  c ).get(a)
      print(xx)
```

[531]
```
c = (2,-1)
a = 'left'
xx = paths.get(  c ).get(a)
print(xx)
```

```
(-1, -1)
```

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

 action_state = paths.get(  current_state ).get(action)
 print(action_state)
 action_state_row_of_Qtable= Qtable.get(action_state)

 max_value = max(action_state_row_of_Qtable.values())
 max_keys = [k for k, v in action_state_row_of_Qtable.items() if v == max_value]

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

 if len(max_keys) > 1:
     max_key = random.choice (max_keys)
 else:
     max_key = random.choice (max_keys)
     #max_index = int(max_index)
 #max_value = Qtable[action_state, max_key]
 max_value = Qtable.get(action_state).get(max_key)

 #Qtable[current_state, action] = Rtable[current_state, action] + gamma * max_value
 Qtable[current_state][action] = Rtable[current_state][ action] + gamma * max_value
 print('max_value', Rtable[current_state][action] + gamma * max_value)

 maxx=0;
 return 0
```

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

 action_state = paths.get(  current_state ).get(action)
 print(action_state)
 action_state_row_of_Qtable= Qtable.get(action_state)

 max_value = max(action_state_row_of_Qtable.values())
 max_keys = [k for k, v in action_state_row_of_Qtable.items() if v == max_value]

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

 if len(max_keys) > 1:
     max_key = random.choice (max_keys)
 else:
     max_key = random.choice (max_keys)
     #max_index = int(max_index)
 #max_value = Qtable[action_state, max_key]
 max_value = Qtable.get(action_state).get(max_key)

 #Qtable[current_state, action] = Rtable[current_state, action] + gamma * max_value
 Qtable[current_state][action] = Rtable[current_state][ action] + gamma * max_value
 print('max_value', Rtable[current_state][action] + gamma * max_value)

 maxx=0;
 return 0

# for key,value in Qtable.items():
#   for k,v in value.items():
#     if v>maxx:
#       maxx=v

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

```
[533] update(initial_state, action, gamma)
```

```
KeyList = list(Qtable.keys())
KeyList
```

```
[(-1, 3),
 (0, 3),
 (3, 3),
 (-1, 2),
 (0, 2),
 (1, 2),
 (2, 2),
 (3, 2),
 (0, 0),
 (0, 1),
 (1, 1),
 (2, 1),
 (3, 1),
 (-1, -1),
 (2, -1)]
```

```
scores = []

for i in range(700):
    current_state = random.choice(KeyList)
    available_act = available_actions(current_state)
    action = sample_next_action(available_act)

    #print ('Score:', str(score))
    score  = update(current_state,action,gamma)
   # scores.append(score)
```

[537]
```
current_state = (-1,3)
steps = [current_state]
direction = []


while current_state != (3,1):

    current_row = Qtable.get(current_state)
    max_value = max(current_row.values())
    max_keys = [k for k, v in current_row.items() if v == max_value]

    if len(max_keys)>1 :
      max_key = random.choice (max_keys)
    else:
      max_key = random.choice (max_keys)# ekai tiyenne eka enawa

    next_state = paths.get(  current_state ).get(max_key)
    steps.append(next_state)
    direction.append(max_key)
    current_state = next_state
```

[538]
```
print("Most efficient path:")
print(steps)
print(direction)
```

```
Most efficient path:
[(-1, 3), (-1, 2), (-1, -1), (2, -1), (2, 1), (3, 1)]
['down', 'down', 'right', 'up', 'right']
```