```
import numpy as np    # for creating arrays and matrices
import gym            # for using AI labraries.
import random         # generating random numbers
```

```
env = gym.make("Taxi-v3")
env.render()

# INFORMATION       POSSIBLE VALUES         Total # POSSIBLE VALUES

# TaxiRow            0, 1, 2, 3, 4          5
# TaxiCol            0, 1, 2, 3, 4          5
# Destination        R, G, B, Y             4
# PassngLoc          R, G, B, Y, onBoard    5


###############Possbile Actions ################
################################################
################# North #######################
#################   Up    #####################
##### West <=Left <TAXI>  Right => East  #######
#################  Down  #######################
################# South #######################
################################################
############ Pickup /  Dropoff ################
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
+---------+
```



```
# Actions:( 1: go north, 2: go south, 3: go west, 4: go east,
#           5: pick up passenger, 6: drop off passenger )
action_size = env.action_space.n
```

```
print("Action size ", action_size)

state_size = env.observation_space.n   # state space size= 5 * 5 * 4 * 5 = 500
print("State size ", state_size)
```

```
Action size  6
State size  500
```

```
# For each state, action initialize the table entry Q'(s, a) to zero.
qtable = np.zeros((state_size, action_size))
print(qtable)
```

```
[[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. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]
```

```
total_episodes = 50000        # Total # of steps(episodes) for Q-Learning prob.
total_step_pickup = 100       # Total #of steps(episodes) since pickup passendge
max_steps = 99                # Max steps per episode

learning_rate = 0.7           # Learning rate
gamma = 0.618                 # Discounting rate [0,1], it is important.
                              # it determines the rewerd of immediate and future.

# Exploration parameters
epsilon = 1.0                 # Exploration rate
                               xploration probability at start
                               inimum exploration probability
decay_rate = 0.01             # Exponential decay rate for exploration prob
```



```
# 2 For life or until learning is stopped
for episode in range(total_episodes):
    # Reset the environment
    state = env.reset()
```

```
    step = 0
    done = False

    for step in range(max_steps):
        # 3. Choose an action a in the current world state (s)
        ## First we randomize a number
        exp_exp_tradeoff = random.uniform(0,1)

        # If this number > greater than epsilon --> exploitation
        # (taking the biggest Q value for this state)
        if exp_exp_tradeoff > epsilon:
            action = np.argmax(qtable[state,:])

        # Else doing a random choice --> exploration
        else:
            action = env.action_space.sample()

        # Take the action (a) and observe the outcome state(s') and reward (r)
        new_state, reward, done, info = env.step(action)

        # Update Q(s,a):= Q(s,a) + lr [R(s,a) + gamma * max Q(s',a') - Q(s,a)]
        qtable[state, action] = qtable[state, action] + learning_rate * (reward
                + gamma * np.max(qtable[new_state, :]) - qtable[state, action])

        # Our new state is state
        state = new_state


        # If done : finish episode
        if done == True:
```



```
                                need less and less exploration)
    epsilon = min_epsilon + (max_epsilon - min_epsilon)*np.exp(-decay_rate*episode)
```

```
env.reset()
rewards = []

```

```
for episode in range(total_step_pickup):
    state = env.reset()
    step = 0
    done = False
    total_rewards = 0
    #print("****************************************************")
    #print("EPISODE ", episode)

    for step in range(max_steps):
        # UNCOMMENT IT IF YOU WANT TO SEE OUR AGENT PLAYING
        env.render()

        # Take the action (index) that have the maximum expected future reward
        # given that state
        action = np.argmax(qtable[state,:])

        new_state, reward, done, info = env.step(action)

        total_rewards += reward

        if done:
            rewards.append(total_rewards)
            print ("Drop-off achieved Score", total_rewards)
            break
        state = new_state
env.close()
print ("Score over time: " +  str(sum(rewards)/total_step_pickup))
```

```
|R: | : :G|
| : | : : |
| : : : : |
```

Saving...

```
  (Pickup)
+---------+
|R: | : :G|
|_: | : : |
| : : : : |
| | : | : |
|Y| : |B: |
```

```
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
|_: : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+

|R: | : :G|
| : | : : |
| : : : : |
|_| : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
Drop-off achieved Score 11
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
```

Saving...

```
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
```

```
  (South)
+---------+
|R: | : :G|
| : | : : |
```

```
# The Program lists taxi T[s, a] and the R[s, a].
############# Q-Learning Taxi Problem V3 ##########
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.

Saving...