# Nima Vahdat 610397163 (HW1)

In [ ]:
```python
import numpy as np
import random
```

- Here we set our algorithm parameter

$$Q^{new}(s_t,a_t) \leftarrow (1-\alpha)\cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg( \underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}} \bigg)}^{\text{learned value}}$$

In [2]:
```python
num_episodes = 10000
max_steps_per_episodes = 20
learning_rate = 0.1
discount_rate = 0.99

exploration_rate = 1
max_exploration_rate = 1
min_exploration_rate = 0.01
exploration_decay_rate = 0.01
```

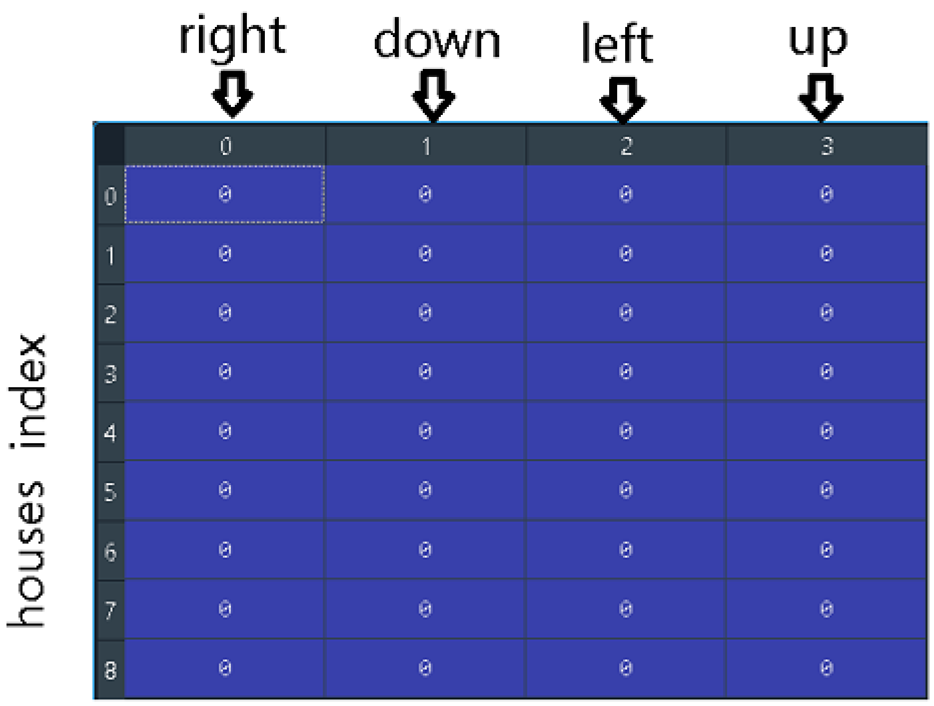
- Just set up few things to consider

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Moon_Sun_Star is showing the starting position and Moon_Sun_End the end position of moons and stars based on indexes above

moves array define proper moves from one index to another, If it is 1, you can go to another house

Q table will be represented like this:




| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 |

In [3]:

```python
# Define the actions
actions = [0,1,2,3,4,5,6,7,8]

# Define the Moon and Sun position at start
Moon_Sun_Start = {
    '0' : '*',
    '1' : '*',
    '2' : '*',
    '3' : '/',
    '4' : '*',
    '5' : '/',
    '6' : '/',
    '7' : ' ',
    '8' : '/'
}

# Define the Moon and Sun position at the end
Moon_Sun_End = {
    '0' : ' ',
    '1' : '/',
    '2' : '*',
    '3' : '/',
    '4' : '*',
    '5' : '/',
    '6' : '*',
    '7' : '/',
    '8' : '*'
}

# Define proper moves
moves = np.array([[0,1,0,1,0,0,0,0,0],
                  [1,0,1,0,1,0,0,0,0],
                  [0,1,0,0,0,1,0,0,0],
                  [1,0,0,0,1,0,1,0,0],
                  [0,1,0,1,0,1,0,1,0],
                  [0,0,1,0,1,0,0,0,1],
                  [0,0,0,1,0,0,0,1,0],
                  [0,0,0,0,1,0,1,0,1],
                  [0,0,0,0,0,1,0,1,0]])


rewards_all_episodes = []
q_table = np.array(np.zeros([9,4]))
```

- We set our reward function this way that

  The function checks that we have reached the goal and returns the score and result

  we refresh MN also (MN shows location of moons and suns for each episode)

In [4]:

```python
# Reward function
def situation(state, new_state, MN):
    global Moon_Sun_End
    MN[str(state)], MN[str(new_state)] = MN[str(new_state)], MN[str(state)]
    point = 0
    for i in range(9):
        if MN[str(i)] == Moon_Sun_End[str(i)]:
            point += 1
    if point == 9:
        return point, True
    return 0, False
```

- Q-Learning In this section based on whether we want to explore or exploite

We set new state and then after getting the reward, update Q table using this equation

$$Q^{new}(s_t,a_t) \leftarrow (1-\alpha)\cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg( \underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}} \bigg)}^{\text{learned value}}$$

If we reached our goal the route will be printed. At the end of each episode we update exploration rate using exponential decay.

In [ ]:
```
for episode in range(num_episodes):
    MN = Moon_Sun_Start.copy()
    route = [7]
    state = 7
    done = False
    reward_current_episode = 0
    for step in range(max_steps_per_episodes):

        # Exploration-Exploitation trade-off
        exploration_rate_threshold = random.uniform(0, 1)
        if exploration_rate_threshold > exploration_rate and np.argmax(q_table[state,:]) != 0:
            action = np.argmax(q_table[state,])
            if action == 0:
                new_state = state + 1
            elif action == 1:
                new_state = state + 3
            elif action == 2:
                new_state = state - 1
            else:
                new_state = state - 3
        else:
            playable_actions = []
            for j in range(9):
                if moves[state, j] > 0:
                    playable_actions.append(j)
            new_state = np.random.choice(playable_actions)
            while(len(route) >= 2 and new_state == route[-2]):
                new_state = np.random.choice(playable_actions)

        # Computing reward and Situation
        reward, done = situation(state, new_state, MN)

        # Updating Q table based on action and reward
        if new_state > state:
            if state + 1 == new_state:
                action = 0 #right
            else:
                action = 1 #down
        else:
            if state - 1 == new_state:
                action = 2 #left
            else:
                action = 3 #up
        q_table[state, action] = q_table[state, action] * \
            (1 - learning_rate) + learning_rate * (reward + discount_rate * np.max(q_table[new_state, :]))

        # Upadting route and checking if we reached the goal or not
        route.append(new_state)
        state = new_state
        reward_current_episode += reward
        if done == True:
            print("\n YEEEEEEEEES! ")
            print(route)
            break



    # Updating exploration rate
    exploration_rate = min_exploration_rate + (max_exploration_rate - min_exploration_rate) * \
        np.exp(-exploration_decay_rate * episode)

    rewards_all_episodes.append(reward_current_episode)
```

YEEEEEEEEES!

[7, 4, 1, 2, 5, 4, 1, 0, 3, 6, 7, 8, 5, 2, 1, 0]

YEEEEEEEEES!

[7, 8, 7, 4, 5, 2, 1, 0, 3, 6, 7, 8, 5, 4, 1, 0]

YEEEEEEEEES!

[7, 4, 5, 2, 1, 0, 3, 6, 7, 8, 5, 4, 1, 0]

YEEEEEEEEES!

[7, 4, 1, 0, 3, 6, 7, 4, 5, 2, 5, 8, 7, 4, 1, 0]

YEEEEEEEEES!

[7, 4, 5, 2, 1, 0, 3, 6, 7, 8, 5, 4, 1, 0]

YEEEEEEEEES!

[7, 4, 1, 0, 3, 4, 5, 2, 1, 4, 5, 8, 7, 6, 3, 0]

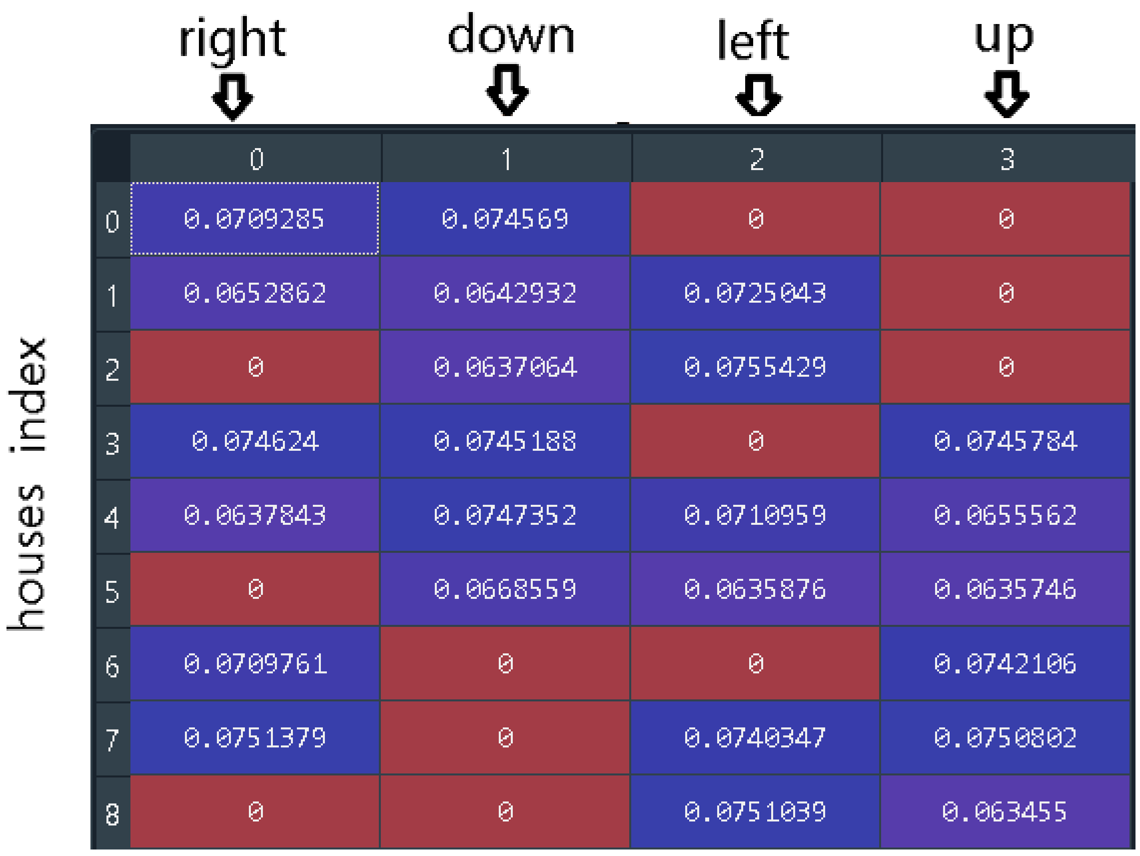
- Final Outputs

Finaly we check our q-table

In [6]:
```
print(" \n", q_table)
```

```
 [[0.01673581 0.01665507 0.         0.        ]
 [0.01677902 0.01673429 0.01668266 0.        ]
 [0.         0.01678469 0.01681967 0.        ]
 [0.01670197 0.01670986 0.         0.01669494]
 [0.01673067 0.01664929 0.01666069 0.01669771]
 [0.         0.00893354 0.01662282 0.01346833]
 [0.01671964 0.         0.         0.0166746 ]
 [0.01668542 0.         0.01667634 0.01667448]
 [0.         0.         0.00178249 0.01664781]]
```

Q table is showing this




| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0.0709285 | 0.074569 | 0 | 0 |
| 1 | 0.0652862 | 0.0642932 | 0.0725043 | 0 |
| 2 | 0 | 0.0637064 | 0.0755429 | 0 |
| 3 | 0.074624 | 0.0745188 | 0 | 0.0745784 |
| 4 | 0.0637843 | 0.0747352 | 0.0710959 | 0.0655562 |
| 5 | 0 | 0.0668559 | 0.0635876 | 0.0635746 |
| 6 | 0.0709761 | 0 | 0 | 0.0742106 |
| 7 | 0.0751379 | 0 | 0.0740347 | 0.0750802 |
| 8 | 0 | 0 | 0.0751039 | 0.063455 |