Seminar/Case Studies

Q- learning

Name - Nimesh Choudhary
Batch - B. Tech CSE -1 2019
Registration Id - 190C2030043

What is Q - Learning?

- Q - learning is a model - free reinforcement learning algorithm which is used to learn the value of an action and it uses the Q - value.
- In Q - learning Q refers to the quality which computes the expected reward from an action taken in a given state.
- Q - learning is a value - based method which is used to inform an agent which action he should take.

s0 ---> S1(a0,q0) ---> S2(a1,q1)

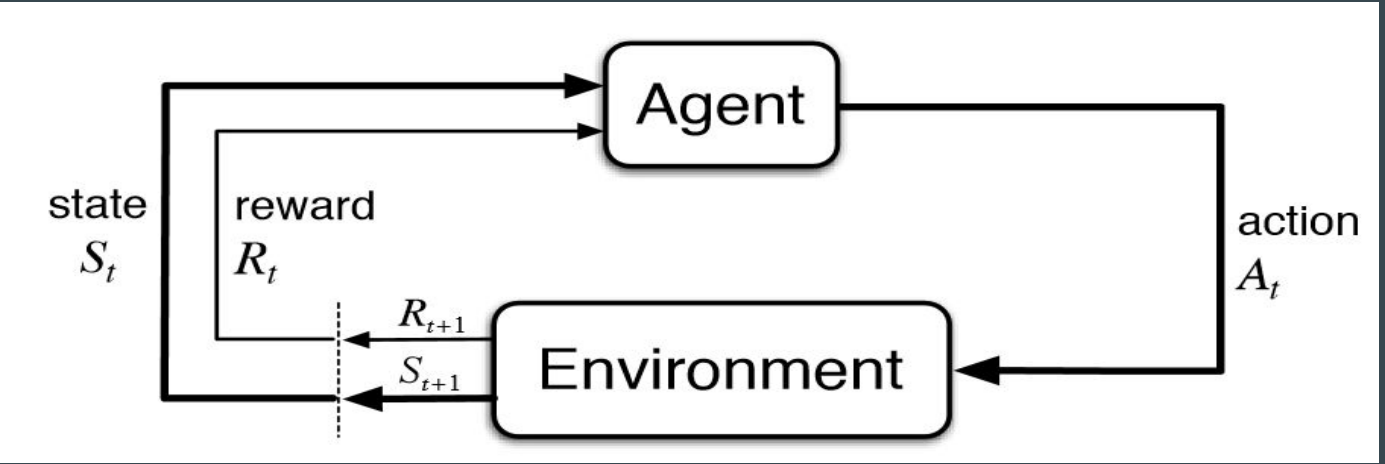
Objective of Q - learning

The main objective of q-learning is

- to learn an optimal policy for an arbitrary environment
- to learn a numerical evaluation function defined over state and actions
- What evaluation function should agents attempt to learn?

Reinforcement Learning

Reinforcement Learning is an area of Machine Learning to get the maximize reward from an action in a particular State.

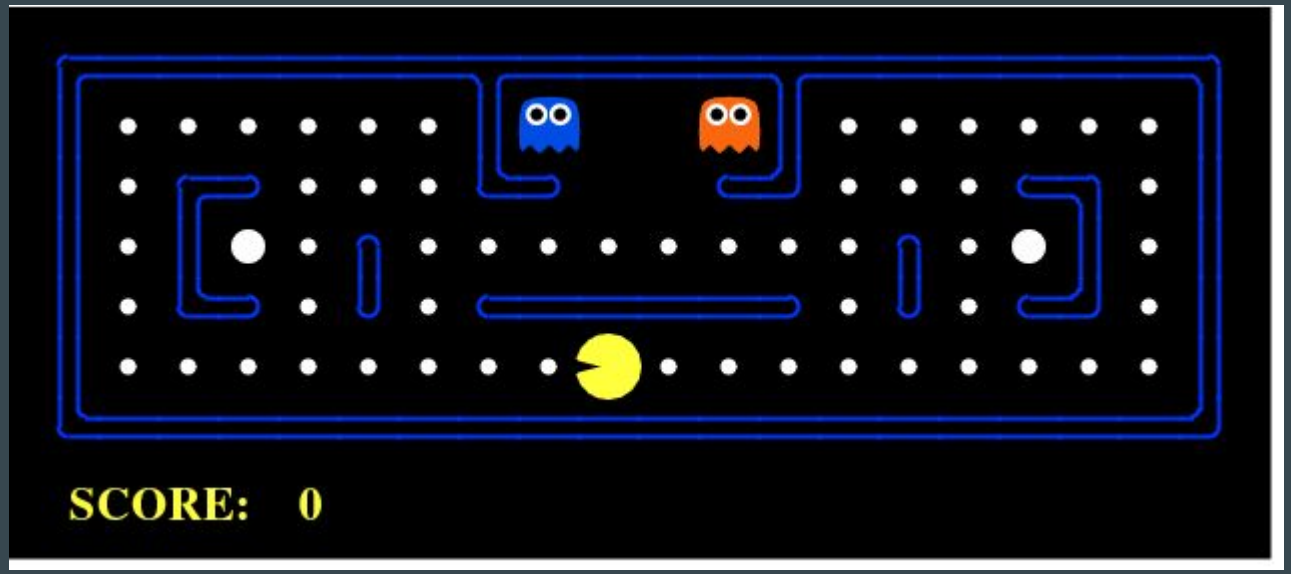


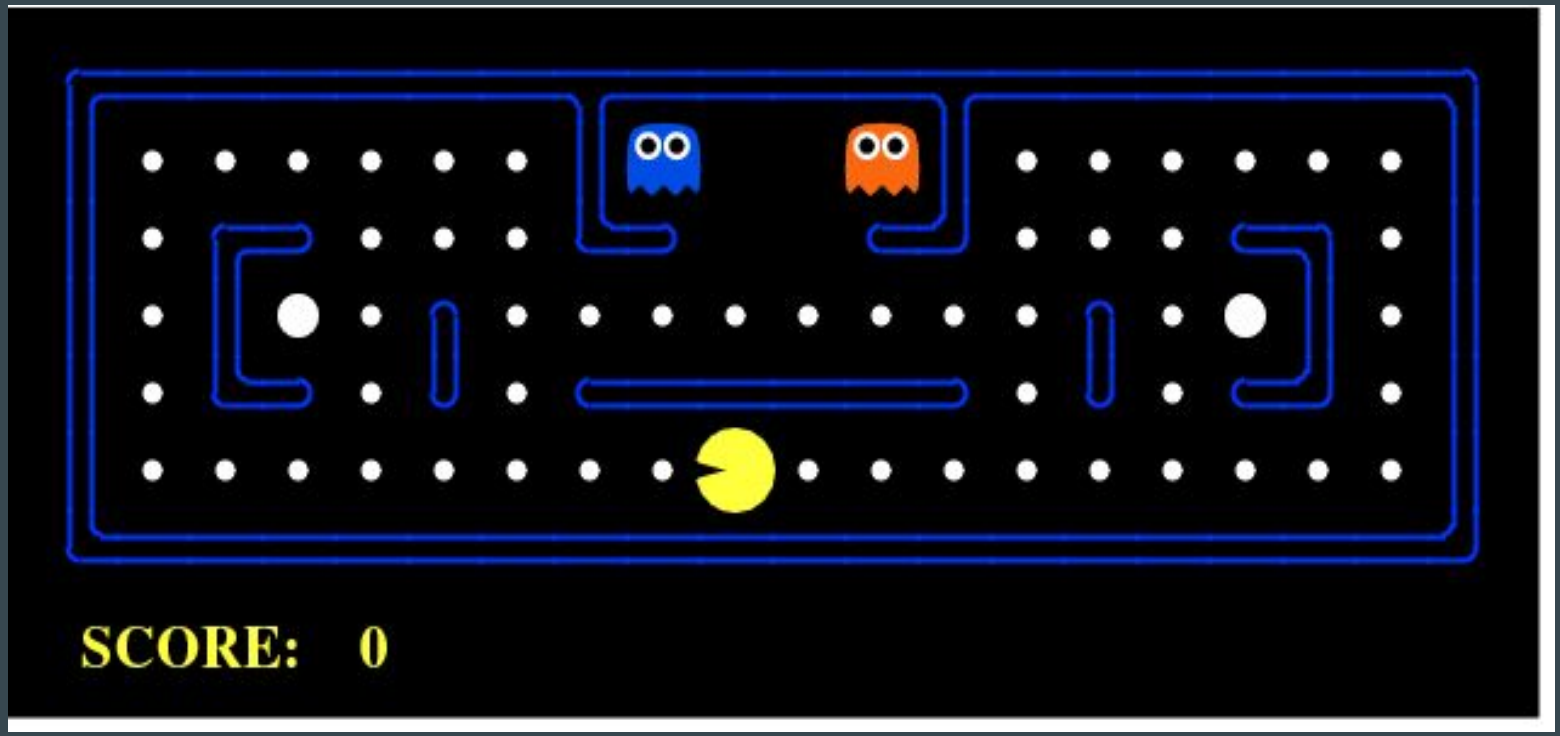
Common Terms Used in Reinforcement Learning

- Environment : - In which the agent can operates
- State : - current situation of the agent
- Reward :- Feedback from the environment
- Value :- Future reward that an agent would receive by taking an action on particular state
- Policy :- f: S -> A {where S represents the state and A represents the actions}

Example : PacMan Game

- The grid like structure shown in the below image is a Environment.
- Current State as shown below.
- The goal of an agent is to eat the food in the grid while avoiding
- Reward -> {Punishment, winning}




SCORE: 0

What is Q - Learning?

- Q - learning is a model - free reinforcement learning algorithm which is used to learn the value of an action and it uses the Q - value.
- In Q - learning Q refers to the Quality how useful a given action is in gaining future reward.
- Q - learning is a value - based method which is used to inform an agent which action he should take.

s0 ---> S1(a0,q0) ---> S2(a1,q1)

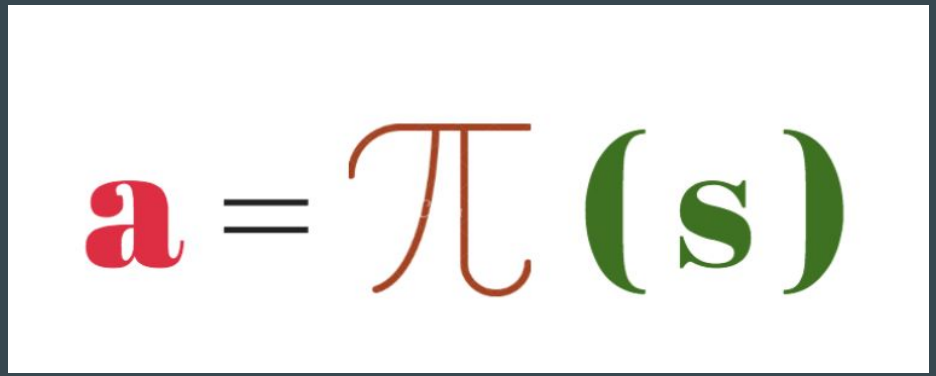
Approaches to Solve Reinforcement Learning

- There are two approach to solve Reinforcement Learning :-

  1. Policy - Based approach

  2. Value - Based approach

Policy - Based Approach :- In this approach we need to optimize the policy. The policy basically defines how agent behaves:



Value - Based Approach :- In this approach our goal is to optimize the value function *V(s)*. So, Value function tells us the maximum expected future reward the agent shall get at each state.

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s \right]$$

Expected

Reward discounted

Given that state

The value of each state is the total amount of the reward an RL agen can expect to collect over the future form a particular state.
So agent will use the above function to select which state to choose at each step.

Types of Algorithm in Reinforcement Learning

There are two types of algorithm that are used in Reinforcement Learning -

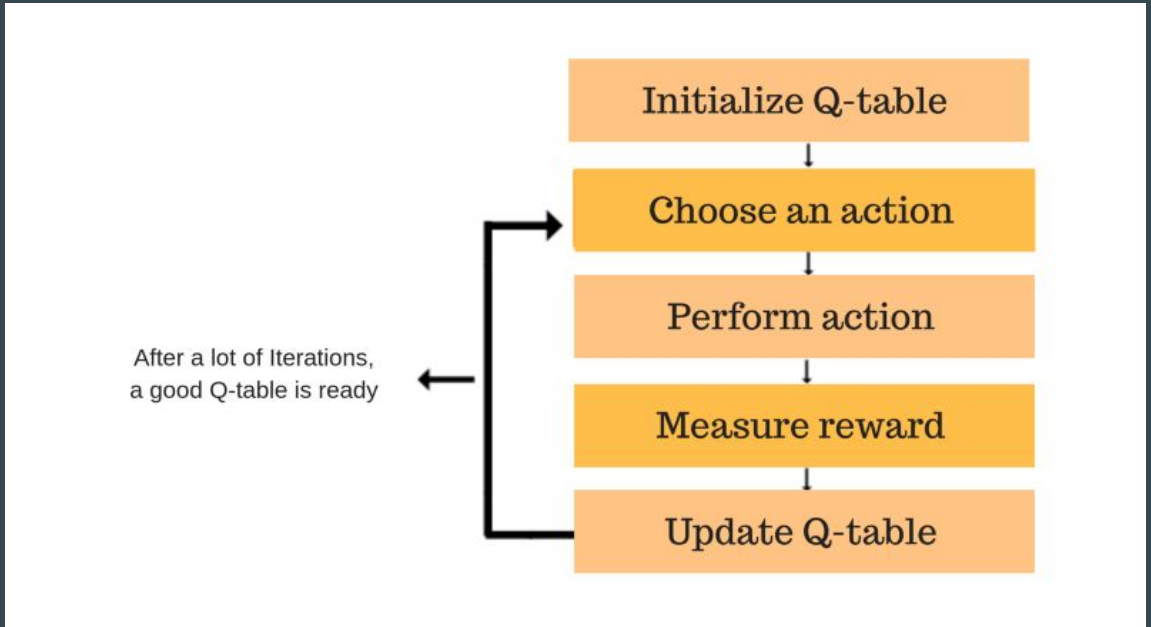
1. Model - Free Algorithm
2. Model - based Algorithm

Model - Free Algorithm :- An algorithm which does not use the transition probability distribution function also known as Reward function like Markov Decision process to estimate the optimal policy

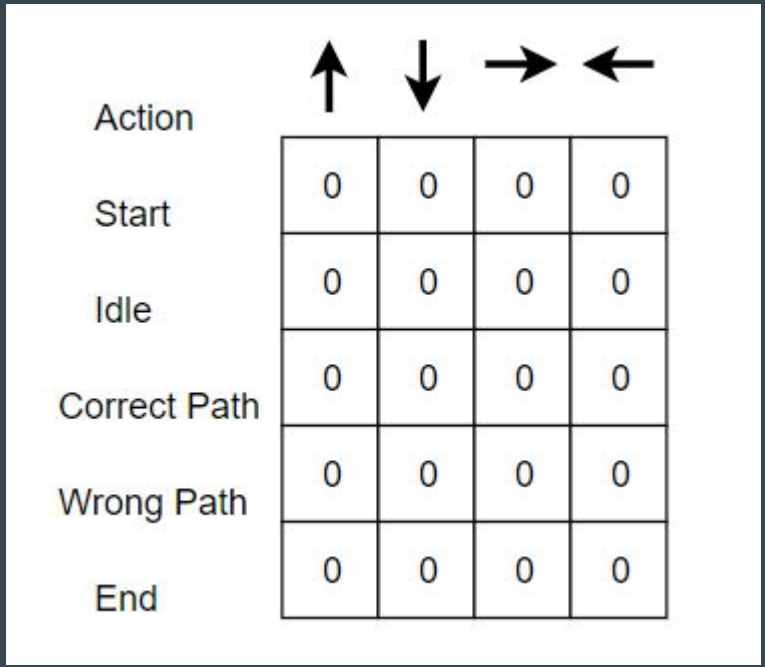
Model - Based Algorithm :- An algorithm which uses the transition function to estimate the optimal Policy

Q - Learning Definition

- Q - Table is the data structure used to calculate the maximum expected future reward for an action at each state.
- Works on Iterative process



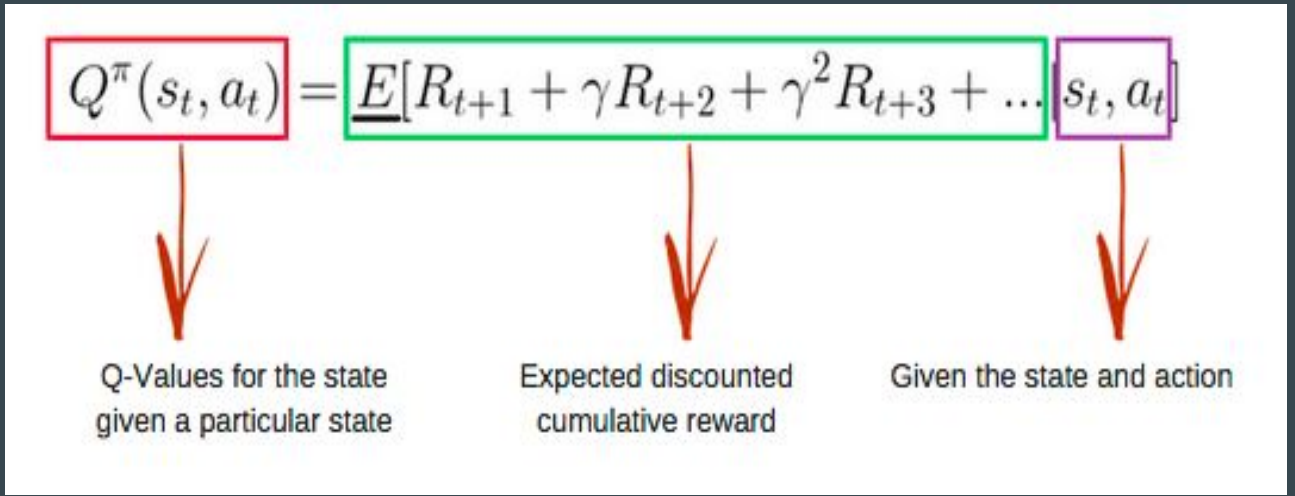
- Initially all values are 0s.
  - At every Iterative approach we need to update the Q - table
  - So Q - table [n X m] where n are columns of the table which tells the number of action and m are rows which tells the number of states.



Mathematics: the Q - learning algorithm

Q - function :-

- The Q - function uses the Bellman equation and takes two input: s {state} and a {action}



Before learning the algorithm of Q - learning we need to know what is evaluation function Q(s,a) which is used to calculate the maximum cumulative future reward we get from a particular state s after applying action a on it. So, Q - function is defined by

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a))$$

so, the above equation is also known as the bellman Equation.
The key problem is to find a appropriate way to estimating training values for Q, which gives the intermediate reward r spread over time which uses the iterative approach not recursively So the relation between Q and V*is.

$$V^*(s) = \max_{a'} Q(s, a')$$

Then from above relationship we can write as

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a')$$

So the recursive definition of Q provides the basis for an algorithm that iteratively approximates Q which is denoted by Q' which refers to the learner's hypothesis. So the agent observe the state s and choose some action at and get some reward r=r(s,a) and the next state will be s'=(s,a)and then we will update the entry for Q' according to the rule

Q learning algorithm

For each s, a initialize the table entry $\hat{Q}(s, a)$ to zero.
Observe the current state s
Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s, a)$ as follows:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

- $s \leftarrow s'$

Applications of Q - learning

- Online Web Auto Configuration
- News Recommendations
- Traffic Signal Control

Challenges

- Q learning works only in an environment having discrete and finite states and action space.
- So, generally Q - learning uses Q - table data structure to optimize the solution. When the number of states are very high then it is very difficult to use the Q - table so for that we need to use the Deep neural network instead of Q - table.

Challenges

- Works only in environment having discrete and finite states and action space.

Automobile Factory Analogy

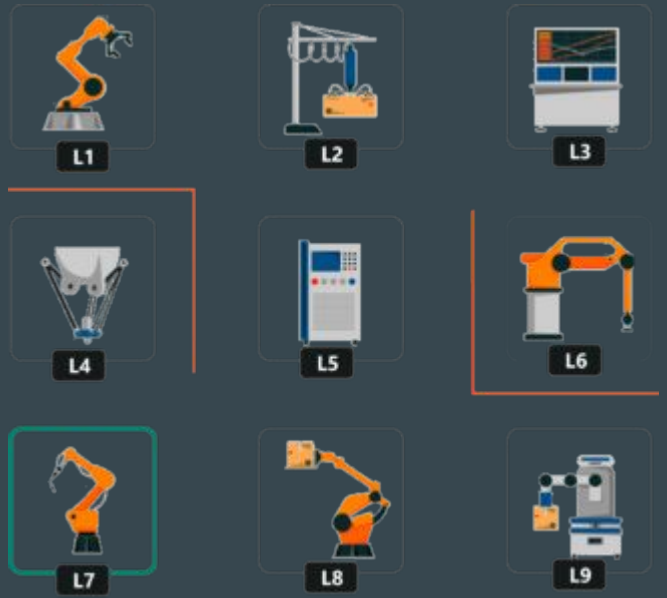
There are different part are located within the factory in 9 stations.

The parts are wheels, dashboard,engine and so on.

So Factory Master has prioritzed the location where wheels are being installed as the highest priority.

And there are some hurdle between the station as shown in the figure

For example, you can’t reach directly L5 from L4 so for reaching that we have to path through location L7 and L8.



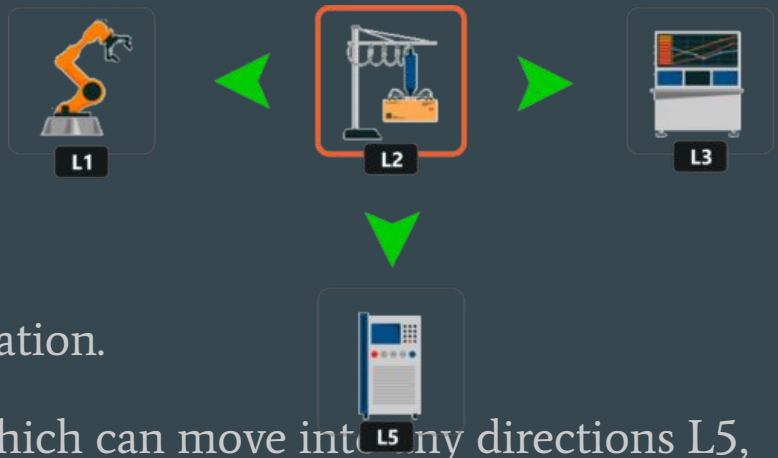
In this,

Location -> L1 L2 L3 L4 L5 L6 L7 L8 L9

States -> 0 1 2 3 4 5 6 7 9

So generally state here is a particular instance

is called its state.



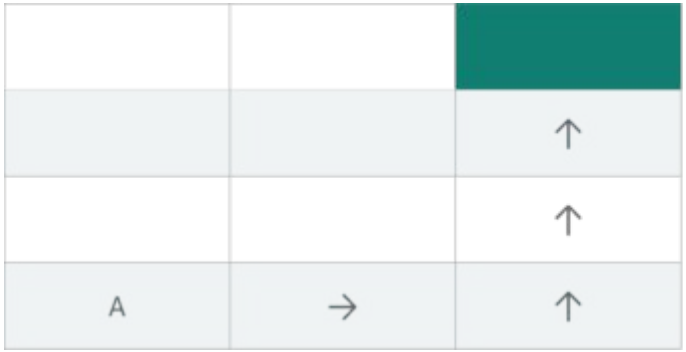
Actions -> moves made by the robots to any location.

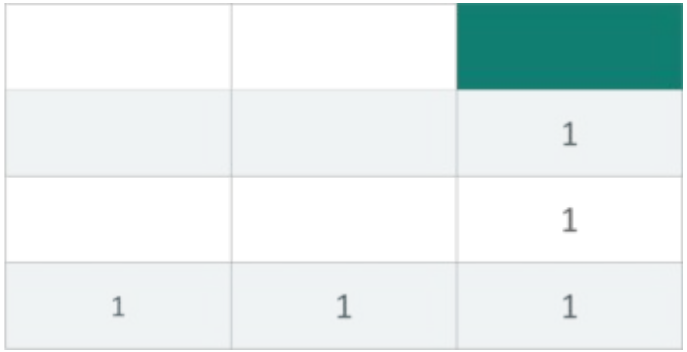
Let considered robot is located at location L2 which can move into any directions L5, L1 and L3.

So, in this case reward will be given to the robot for going directly from one state to another. So in above case if robot reach L5 directly from L2 and vice versa then reward of 1 will be provided.

	L1	L2	L3	L4	L5	L6	L7	L8	L9
L1	0	1	0	0	0	0	0	0	0
L2	1	0	1	0	0	0	0	0	0
L3	0	1	0	0	0	1	0	0	0
L4	0	0	0	0	0	0	1	0	0
L5	0	1	0	0	0	0	0	1	0
L6	0	0	1	0	0	0	0	0	0
L7	0	0	0	1	0	0	0	1	0
L8	0	0	0	0	1	0	1	0	1
L9	0	0	0	0	0	0	0	1	0

Now suppose robot need to go from A to B. It will chose the path which will yield you a positive reward.





So suppose robot start from A and in between where it can see two or more than two paths then how robot decide to take decision because robots doesn't have a memory like human beings. So in that case Bellman Equations comes into the pictures

$$V(s) = max(R(s,a) + \gamma V(s'))$$

Where:

- s = a particular state
- a = action
- s′ = state to which the robot goes from s
- γ = discount factor
- R(s, a) = a reward function which takes a state (s) and action (a) and outputs a reward value
- V(s) = value of being in a particular state

Imagine robot is on the orange block and needs to reach the green block which is destination. But if there will be some dysfunction in the robot which can be confused the robot to decide the path

So in this case we need to modify the decision-making process.

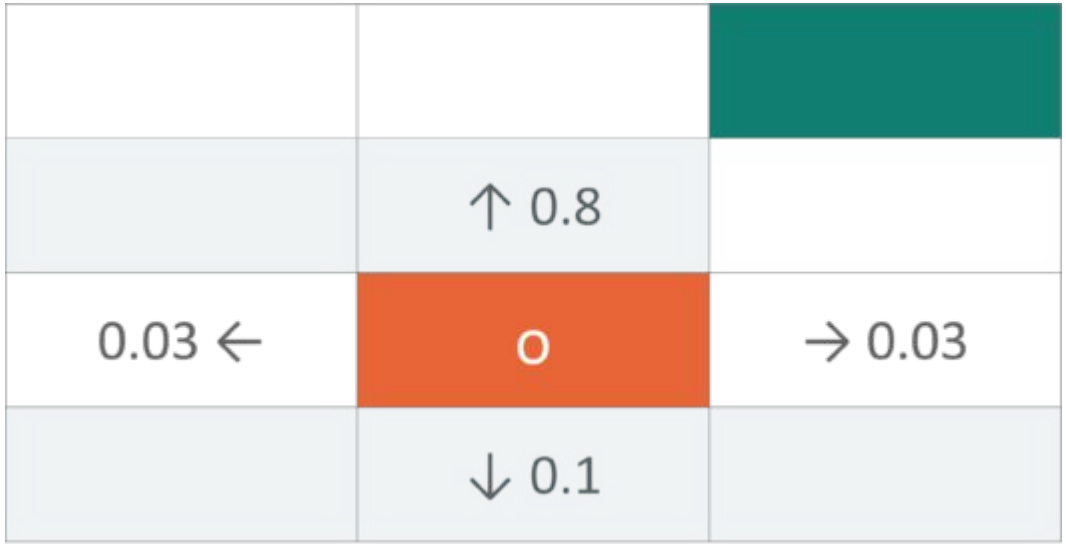
It has to Partly Random and Partly under robot's control.. So for that we need to change Bellman equation. So in that case we don't know the next state s1, so for that we will know all the possibilities of a turn and the change in the equation is

$$V(s) = max(R(s, a) + \gamma V(s'))$$

$$V(s) = max(R(s, a) + \gamma \Sigma_{s'} P(s, a, s') V(s'))$$

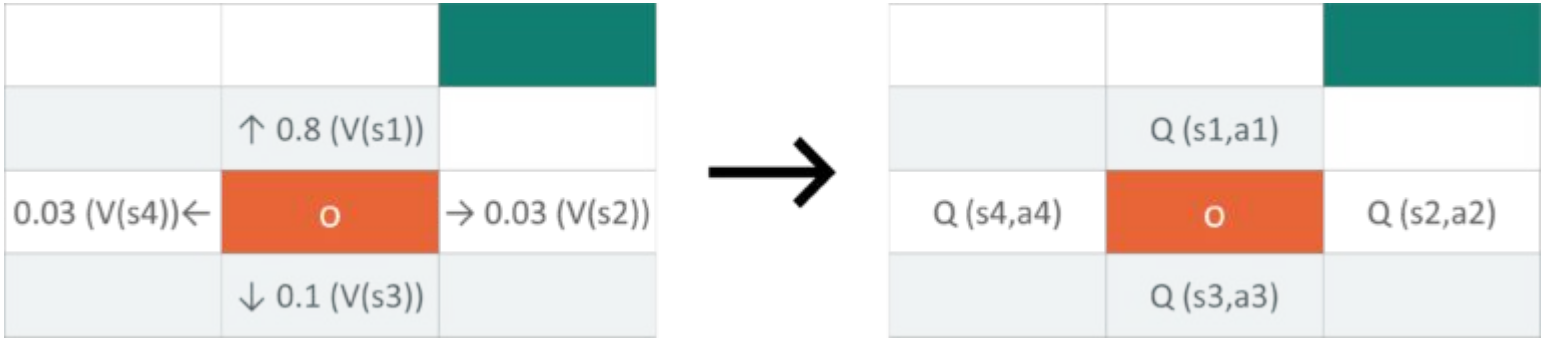
P(s,a,s') *:* Probability of moving from state *s* to *s'* with action *a*

Σ s' P(s,a,s') V(s') *:* Randomness Expectations of Robot



V(s) = max(R(s, a) + γ ((0.8V(room up)) + (0.1V(room down) + ….))

So, As we all know that Q - learning tells us quality of an action taken to move from one state to another state not the possible values of state..



So if we remove the max component from the above equation then it become only one step/move for possible action which is nothing but the quality of action

$$Q(s, a) = (R(s, a) + \gamma \Sigma_{s'} P(s, a, s') V(s'))$$

Temporal Difference to calculate the Q-values with respect to the changes in the environment over time.

$$TD(s, a) = (R(s, a) + \gamma \Sigma s' P(s, a, s') \max Q(s', a')) - Q(s, a)$$

We recalculate the new Q(s, a) with the same formula and subtract the previously known Q(s, a) from it. So, the above equation becomes:

$$Qt(s, a) = Qt\text{-}1(s, a) + \alpha TDt(s, a)$$

Q_t (s, a) = Current Q-value

Q_{t-1} (s, a) = Previous Q-value

```python
import numpy as np
```

```python
#Defines the states
location_to_state = {
    'L1': 0,
    'L2': 1,
    'L3': 2,
    'L4': 3,
    'L5': 4,
    'L6': 5,
    'L7': 6,
    'L8': 7,
    'L9': 8
}
```

```
rewards = np.array([[0,1,0,0,0,0,0,0,0],
                    [1,0,1,0,1,0,0,0,0],
                    [0,1,0,0,0,1,0,0,0],
                    [0,1,0,0,0,0,1,0,0],
                    [0,1,0,0,0,0,0,1,0],
                    [0,0,1,0,0,0,0,0,0],
                    [0,0,0,1,0,0,0,1,0],
                    [0,0,0,0,1,0,1,0,1],
                    [0,0,0,0,0,0,0,1,0]])
```

Rewards is an array which define the rewards after move from one state to another

```
state_to_location = dict((state,location) for location,state in location_to_state.items())
gamma = 0.75
alpha = 0.9
```

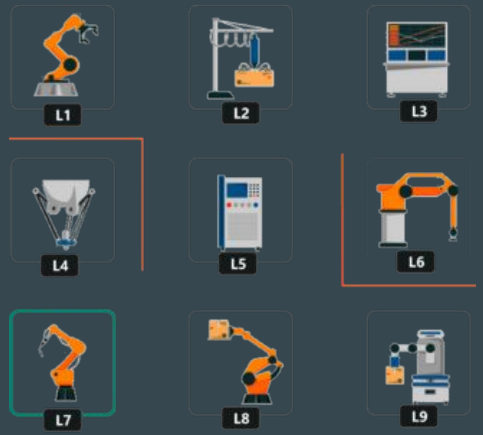
Where gamma and alpha tella the learning rate and discount factor

```python
def get_optimal_route(start_location, end_location):
    rewards_new = np.copy(rewards)
    ending_state = location_to_state[end_location]
    rewards_new[ending_state, ending_state] = 999

    Q = np.array(np.zeros([9, 9]))

    # Q-Learning process
    for i in range(1000):
        current_state = np.random.randint(0, 9)#used to pick up the random state
        playable_actions = []
        # So now we will check the reward value from current_state to next state j is greater than zero or not
        for j in range(9):
            if rewards_new[current_state, j] > 0:
                playable_actions.append(j) #reward value is greater than 0 then we will add into this list
        # Pick a random action that will lead us to next state
        next_state = np.random.choice(playable_actions) # now next_state will be the top most element of playable_actions list
        # Computing Temporal Difference
        TD = rewards_new[current_state, next_state] + gamma * Q[next_state, np.argmax(Q[next_state,])] - Q[
            current_state, next_state]
        # Updating the Q-Value using the Bellman equation
        Q[current_state, next_state] += alpha * TD

    # Initialize the optimal route with the starting location
    route = [start_location]
    # Initialize next_location with starting location
    next_location = start_location

    # We don't know about the exact number of iterations needed to reach to the final location hence while loop will be a good ch
    while (next_location != end_location):
        # Fetch the starting state
        starting_state = location_to_state[start_location]
        # Fetch the highest Q-value pertaining to starting state
        next_state = np.argmax(Q[starting_state,])
        # We got the index of the next state. But we need the corresponding letter.
        next_location = state_to_location[next_state]
        route.append(next_location)
        # Update the starting location for the next iteration
        start_location = next_location

    return route, Q
```

```
route, Q = get_optimal_route('L1','L9')
print("Optimal Path -> ",route)
print("Q-Table ->\n",Q)
```

Output->

```
Optimal Path ->  ['L1', 'L2', 'L5', 'L8', 'L9']
Q-Table ->
 [[   0.         1267.08330176    0.            0.            0.
     0.            0.            0.            0.         ]
 [ 951.31231932    0.          951.31215236    0.         1688.11106913
     0.            0.            0.            0.         ]
 [   0.         1267.08329304    0.            0.            0.
   714.47822447    0.            0.            0.         ]
 [   0.         1267.08330157    0.            0.            0.
     0.         1688.11678477    0.            0.         ]
 [   0.         1267.08330184    0.            0.            0.
     0.            0.         2249.48143351    0.         ]
 [   0.            0.          951.31239925    0.            0.
     0.            0.            0.            0.         ]
 [   0.            0.            0.         1267.08511677    0.
     0.            0.         2249.48905087    0.         ]
 [   0.            0.            0.            0.         1688.11107484
     0.         1688.11642553    0.         2997.9854017 ]
 [   0.            0.            0.            0.            0.
     0.            0.         2249.48905124 3995.99299882]]
```



OpenAI - Gym library's interface

So Gym provides different types of environments that we can plug into our code and test it. This library takes care of API which provides us all the information about the environment, the number of possible states and actions present in it.

So in this tutorial we are going to use the Taxi - v2 Gym environment.

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

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

Timestep: 1
State: 328
Action: 5
Reward: -10
```

```
Episode: 100000
Training finished.

Time steps taken: 1117
Penalties incurred: 363

Wall time: 30.6 s
```

So from these two we clearly see that earlier there were 363 penalties done by a taxi agent and now there are 0.0 penalties/100 episodes done by taxi agent. So as we clearly see from the evaluation, the agent's performance improved significantly which means it performed the correct pickup and drop off actions.

```
Results after 100 episodes:
Average timesteps per episode: 12.3
```

Code DEMO

```
-------------
|   |   |   |
-------------
|   | x |   |
-------------
|   |   |   |
-------------
Input your action row:2
Input your action col:2
-------------
|   |   |   |
-------------
|   | x |   |
-------------
|   |   | o |
-------------
-------------
|   |   |   |
-------------
|   | x |   |
-------------
|   | x | o |
-------------
Input your action row:0
Input your action col:1
```