Agent. The RL algorithm that learns from trial and error

Environment. The world through which the agent moves

Action (A): All the possible steps that the agent can take

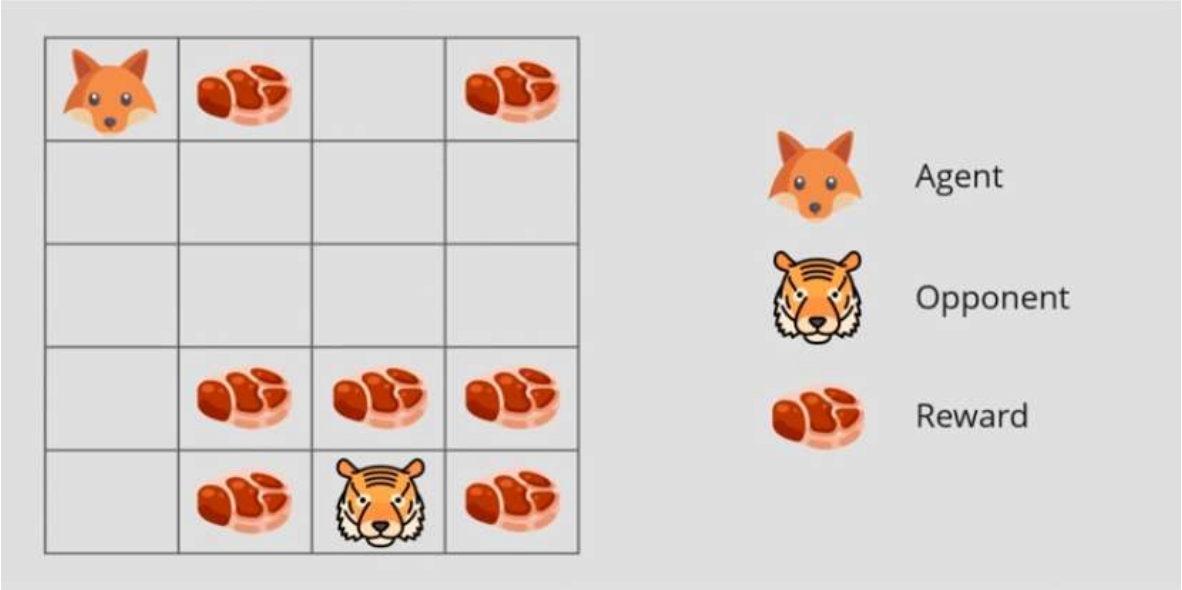
State (S): Current condition returned by the environment

Reward (R): An instant return from the environment to appraise the last action

Policy (π): The approach that the agent uses to determine the next action based on the current state

Value (V): The expected long-term return with discount, as opposed to the short-term reward R

Action-value (Q): This is similar to Value, except, it takes an extra parameter, the current action (A)



Exploitation is about using the already known exploited information to heighten the rewards

Exploration is about exploring and capturing more information about an environment

Markov Decision Process

The mathematical approach for mapping a solution in reinforcement learning is called Markov Decision Process (MDP)

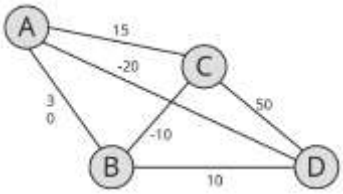
PARAMETERS INVOLVED IN MDP

- Set of actions, A
- Set of states, S
- Reward, R
- Policy, π
- Value, V

MARKOV DECISION PROCESS EXAMPLE – SHORTEST PATH PROBLEM

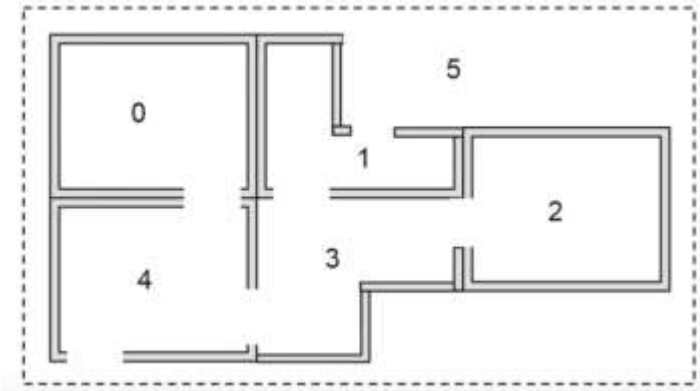
Goal: Find the shortest path between A and D with minimum possible cost

In this problem,



- Set of states are denoted by nodes i.e. {A, B, C, D}
- Action is to traverse from one to another {A -> B, C -> D}
- Reward is the cost represented by each edge
- Policy is the path taken to reach the destination {A -> C ->D}

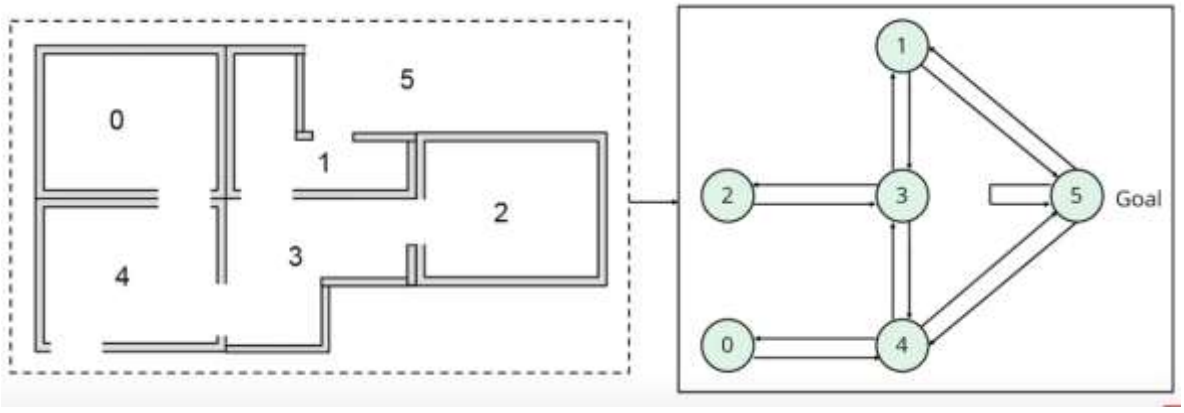
Q-LEARNING ALGORITHM {Reinforcement Learning}



- 5 rooms in a building connected by doors
- each room is numbered 0 through 4
- The outside of the building can be thought of as one big room (5)
- Doors 1 and 4 lead into the building from room 5 (outside)

GOAL: Get outside of the building. (Shortest path 0, … , 4 -> 5)

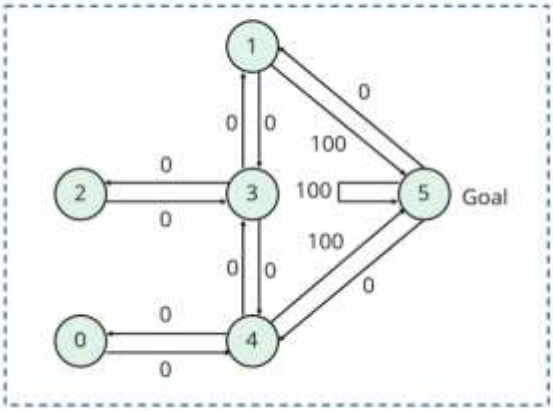
Graph representation



As the rooms 1 and 4 are connected to the 5, or the goal, we assign values to this rooms with a high reward. Like 100, while the rooms left have a reward of 0:

Next step is to associate a reward value to each door:

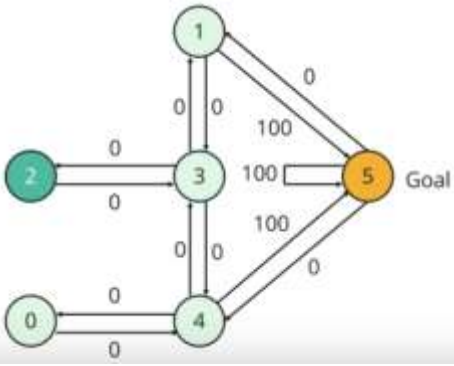
- doors that lead directly to the goal have a reward of 100
- Doors not directly connected to the target room have zero reward
- Because doors are two-way, two arrows are assigned to each room
- Each arrow contains an instant reward value



States are represented as rooms. There is a total of 5 states (S).

The movement of these rooms or states represents the action (A).

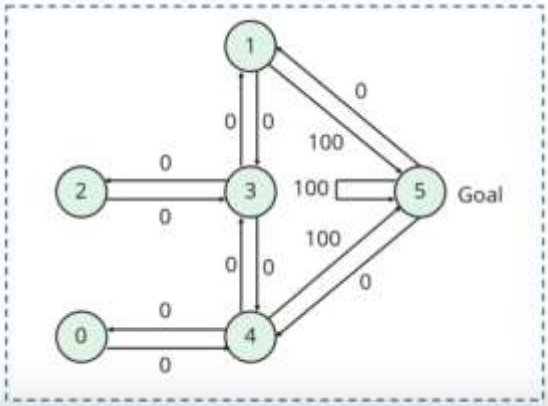
In the figure, a state is depicted as a node, while “action” is represented by the arrows



Example (Agent traverse from room 2 to room5):

1. Initial state = state 2
2. State 2 -> state 3
3. State 3 -> state (2, 1, 4)
4. State 4 -> state 5

We can put the state diagram and the instant reward values into a reward table, matrix R where the -1 values are null values, and the possible states (rows) are the same as the actions (columns). The right way to represent a reward in this table is (State, Action) (4, 5) => 100 while (5,4) => 0.



Action

State	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

$R =$

The -1's in the table represent null values

There is another matrix called the Q-Matrix, which represents the memory of what the agent has learned through experience.

- The rows of matrix Q represent the current state of the agent
- Columns represent the possible actions leading to the next state
- The formula to calculate the Q-Matrix:

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

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{old\ value} + \underbrace{\alpha}_{learning\ rate} \cdot "Q^{current}(state, action)"$$

$$Q => Quality\ of\ the\ actions$$

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

NOTE. The Gamma parameter has a range of 0 to 1 (0 <= Gamma >1).

- If Gamma is closer to zero, the agent will tend to consider only immediate rewards (Exploit).
- If Gamma is closer to one, the agent will consider future rewards with greater weight (Q-value -> Exploration).

Q-LEARNING ALGORITHM

1. Set the gamma parameter, and environment rewards in matrix R
2. Initialize matrix Q to zero
3. Select random initialize state
4. Set initial state = current state
5. Select one among all possible actions for the current state
6. Using this possible action, consider going to the next state
7. Get maximum Q value for this next state based on all possible actions
8. Compute: Q(state, action) = R(State, action) + Gamma*Max[Q(next state, all actions)]
9. The Q Matrix is updated.
10. Repeat above steps until current state = goal state (Gamma zero?)

The first step is to set the value of the learning parameter Gamma = 0.8, and the initial state as Room 1. The nodes at the right are what is known as Markov Decision Trees.

Next, initialize matrix Q as a zero matrix:

- From room 1 you can either go to room 3 or 5, let's select room 5.
- From room 5, calculate maximum Q value for this next state based on all possible actions:

$$Q(1,5) = R(1,5) + 0.8 * Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8 * 0 = 100$$

Then the matrix Q gets updated (We finish an episode)

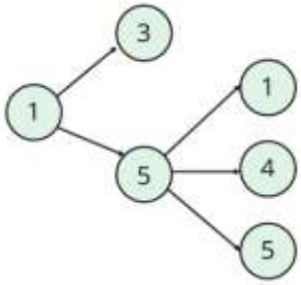
Q(1,5) = R(1,5) + 0.8 * Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8 * 0 = 100

$Q =$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100



For the next episode (second), we start with a randomly chosen initial state, i.e. state 3

- From room 3 you can either go to room 1,2 or 4, let's select room 1.
- From room 1, calculate maximum Q value for this next state based on all possible actions:

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

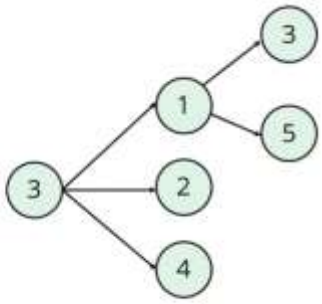
Q(3,1) = R(3,1) + 0.8 * Max[Q(1,3), Q(1,5)]= 0 + 0.8 * [0, 100] = 80
The matrix Q get's updated

$Q =$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	80	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100



For the next episode (third), the next stage, 1, now becomes the current state. We repeat the inner loop of the Q learning algorithm because state 1 is not the goal state.

- From room 1 you can either go to room 3 or 5, let's select room 5.
- From room 5, calculate maximum Q value for this next state based on all possible actions:

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

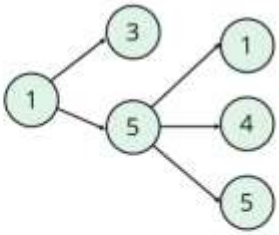
Q(1,5) = R(1,5) + 0.8 * Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8 * 0 = 100
The matrix Q remains the same since, Q(1,5) is already fed to the agent

$Q =$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	80	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

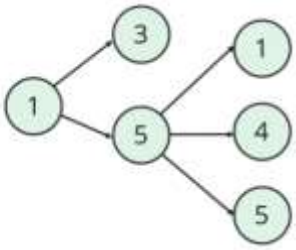


For the same episode (third), and the next stage (second), 1, now becomes the current state. We repeat the inner loop of the Q learning algorithm because state 1 is not the goal state.

- From room 1 you can either go to room 3 or 5, let's select room 5.
- From room 5, calculate maximum Q value for this next state based on all possible actions:

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

Q(1,5) = R(1,5) + 0.8 * Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8 * 0 = 100
The matrix Q remains the same since, Q(1,5) is already fed to the agent



$Q =$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	80	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

In [2]:
```
import numpy as np
# R matrix
R = np.matrix([[-1,-1,-1,-1, 0,-1], # (S,A) -> R(0,4) = 0
               [-1,-1,-1, 0,-1,100],# (S,A) -> R(1,3) = 0; R(1,5) = 100
               [-1,-1,-1, 0,-1,-1], # (S,A) -> R(2,3) = 0
               [-1, 0, 0,-1, 0,-1], # (S,A) -> R(3,1) = 0; R(3,2) = 0; R(3,4) =
0
               [ 0,-1,-1, 0,-1,100],# (S,A) -> R(4,0) = 0; R(4,3) = 0; R(4,5) =
100
               [-1, 0,-1,-1, 0,100]])# (S,A) -> R(5,1) = 0; R(5,4) = 0; R(5,5)
= 100

# Q matrix
Q = np.matrix(np.zeros([6, 6]))

# Gamma (Learning rate).
gamma = 0.8


# This function returns all available actions in the state given as an argument
def available_actions(state):
    current_state_row = R[state]    # Returns a matrix ([[ -1,  -1,  -1,   0,
-1, 100]])
    av_act = np.where(current_state_row >= 0)[1] # where([[ -1,  -1,  -1,   0,
-1, 100]]) >= 0
                                                 # [0] -> array([0, 0]), [0] ->
array([3, 5])
    return av_act # return all available actions greater or equal than zero | a
v_act = array([3, 5])
                                               #![0] for values # [1] for positio
ns


# This function updates the Q matrix according the path selected and the Q
# learning algorithm
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 learning formula       R [row, column]
    Q[current_state, action] = R[current_state, action] + gamma * max_value

#--------------------------------------------------------------
# Training
# Train over 10 000 iterations, (Re-iterate the process above).
for i in range(10000):
    current_state = np.random.randint(0, int(Q.shape[0]))
    available_act = available_actions(current_state)
    action = int(np.random.choice(available_act, size=1))
    update(current_state, action, gamma)

print("Training Q Matrix:")
print(Q / np.max(Q) * 100)
```

```
Training 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. ]]
```

In [3]:
```
# Testing

# Goal state = 5
# Best sequence path starting from 2 -> 2, 3, 4, 5

current_state = 1
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 elected sequence of steps
print("Selected path: ")
print(steps)
```

```
Selected path:
[1, 5]
```

In [ ]:
```

```