Machine Learning
(Reinforcement Learning)

Hello!

I am Eslam Ahmed

I am a software engineer.

You can find me at jeksogsa@gmail.com

Hello!

I am Eman Ehab

I am a ML research engineer.

You can find me at
emanehab.ieee@gmail.com

Contents

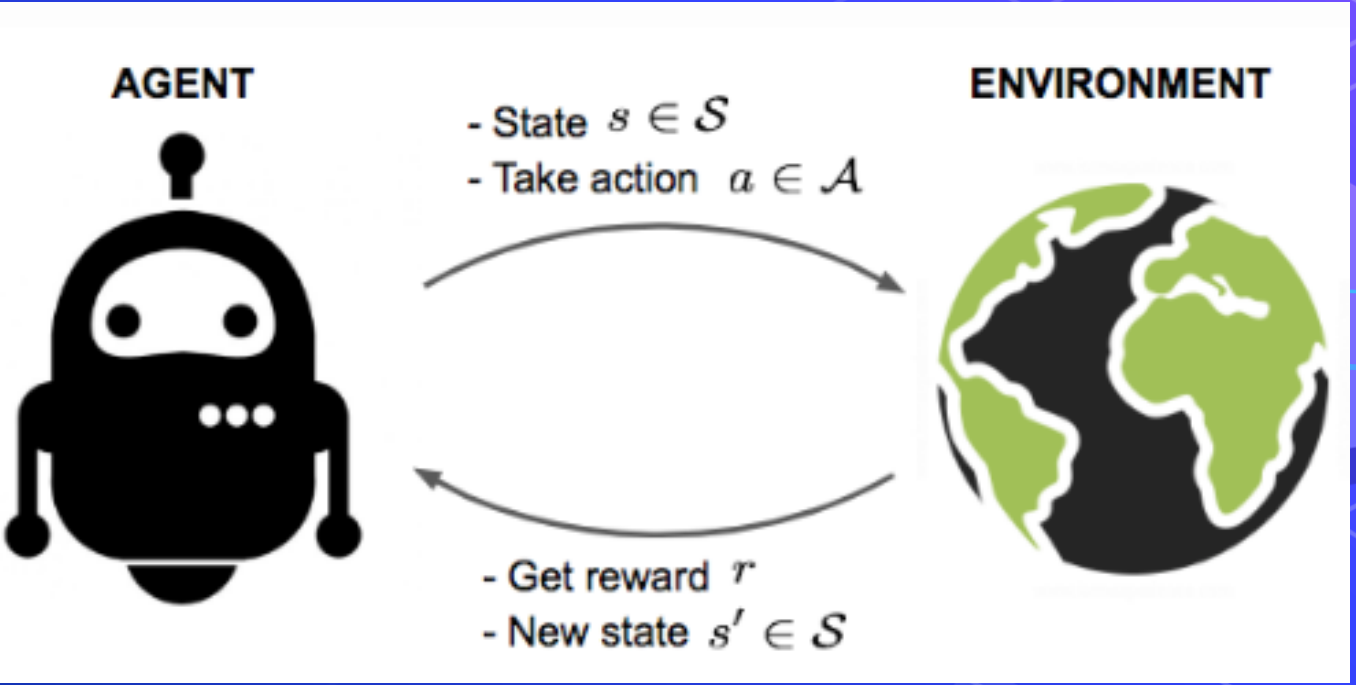
Contents

Machine Learning

- Supervised learning : Task Driven (Classification)
- Unsupervised learning : Data Driven (Clustering)
- Reinforcement learning :
  - Close to human learning.
  - Algorithm learns a policy of how to act in a given environment.
  - Every action has some impact in the environment, and the environment provides rewards that guides the learning algorithm.

Contents

What is RL ?


AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

What is RL ?

3azooz (Agent)

What is RL ?

What is RL ?

What is RL ?

What is RL ?

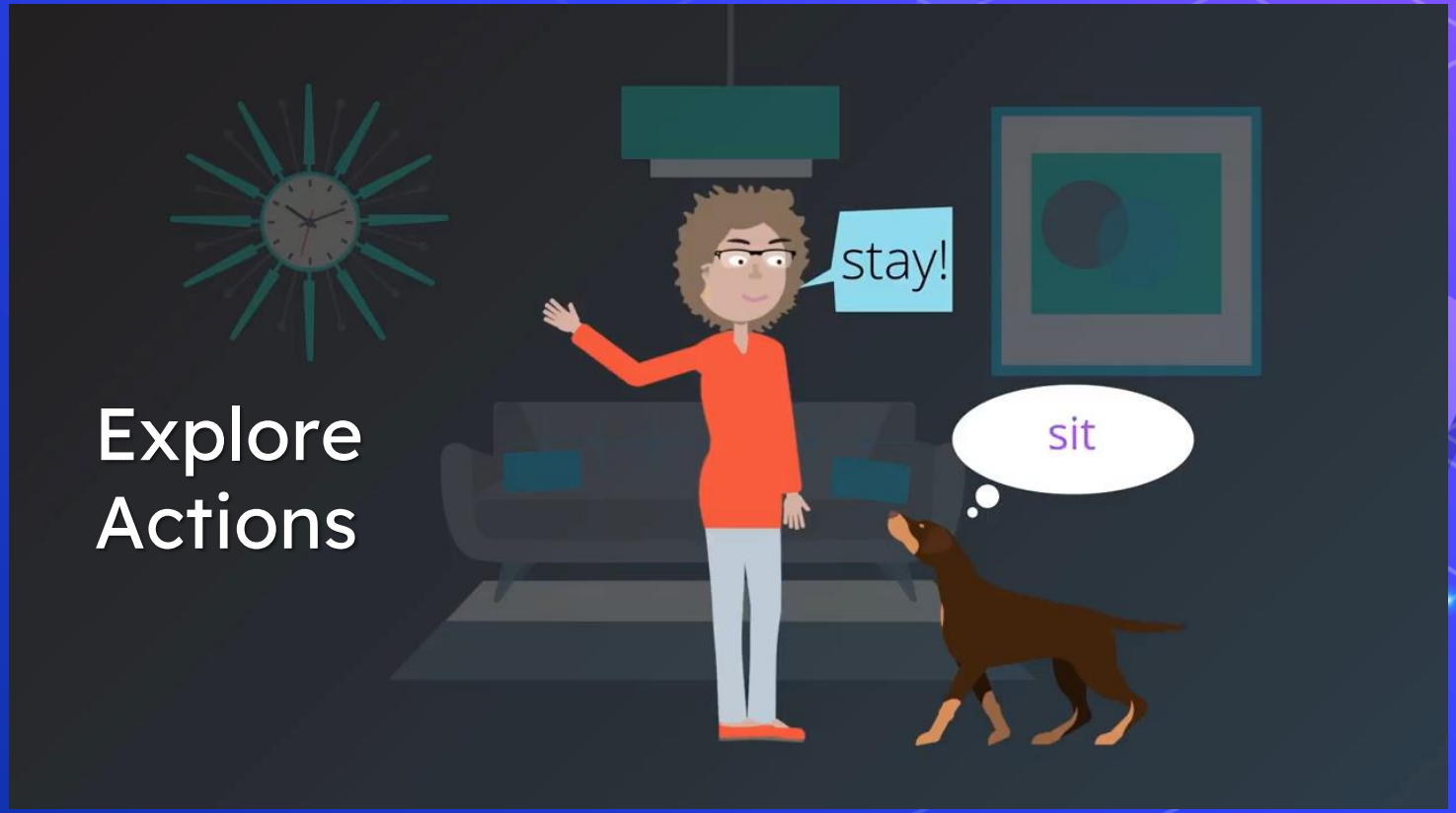
What is RL ?

What is RL ?

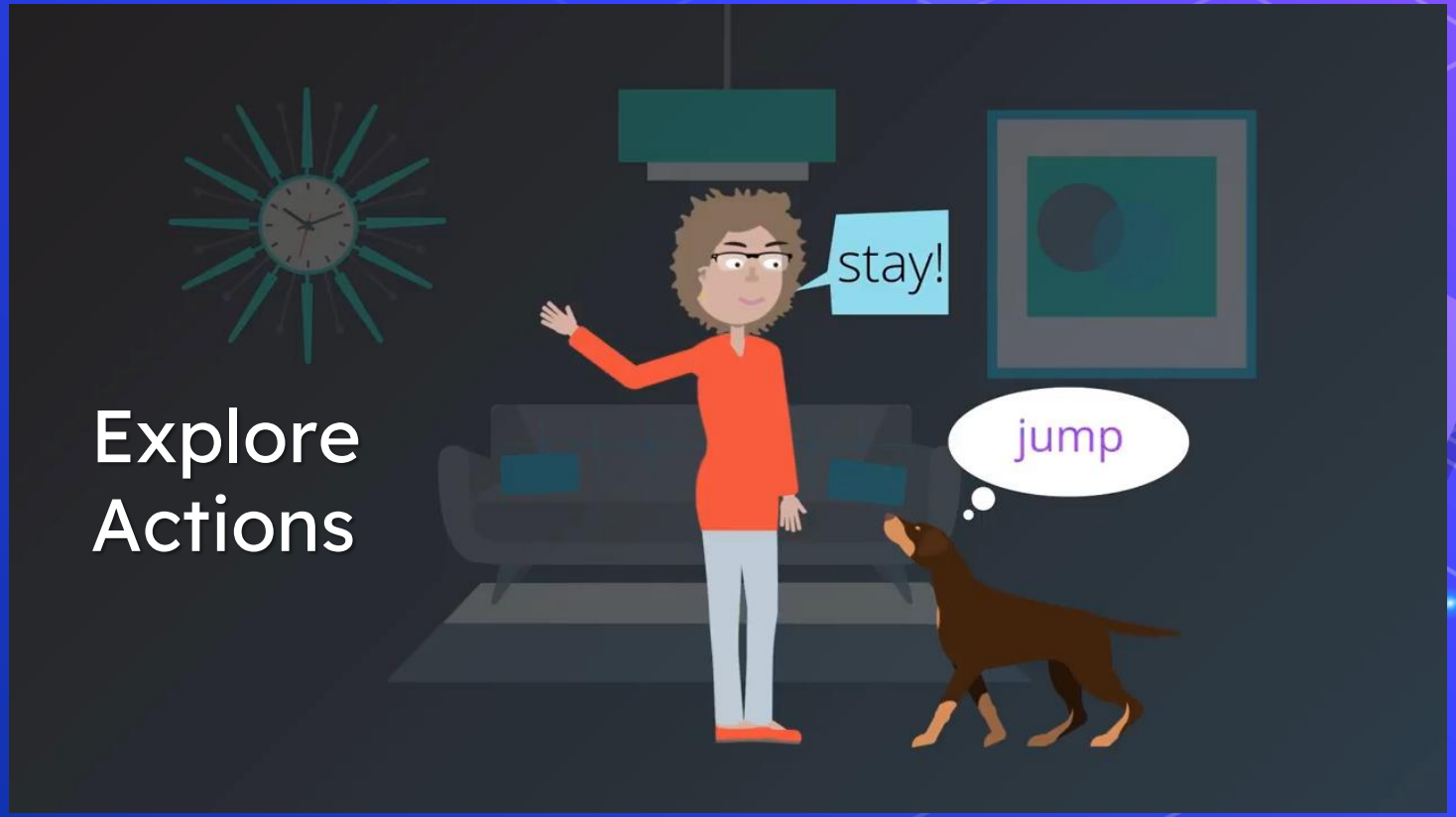
What is RL ?

What is RL ?

What is RL ?



What is RL ?



What is RL ?

What is RL ?

What is RL ?

What is RL ?

What is RL ?

What is RL ?

Again and again
for a lot of times
(Episodes)

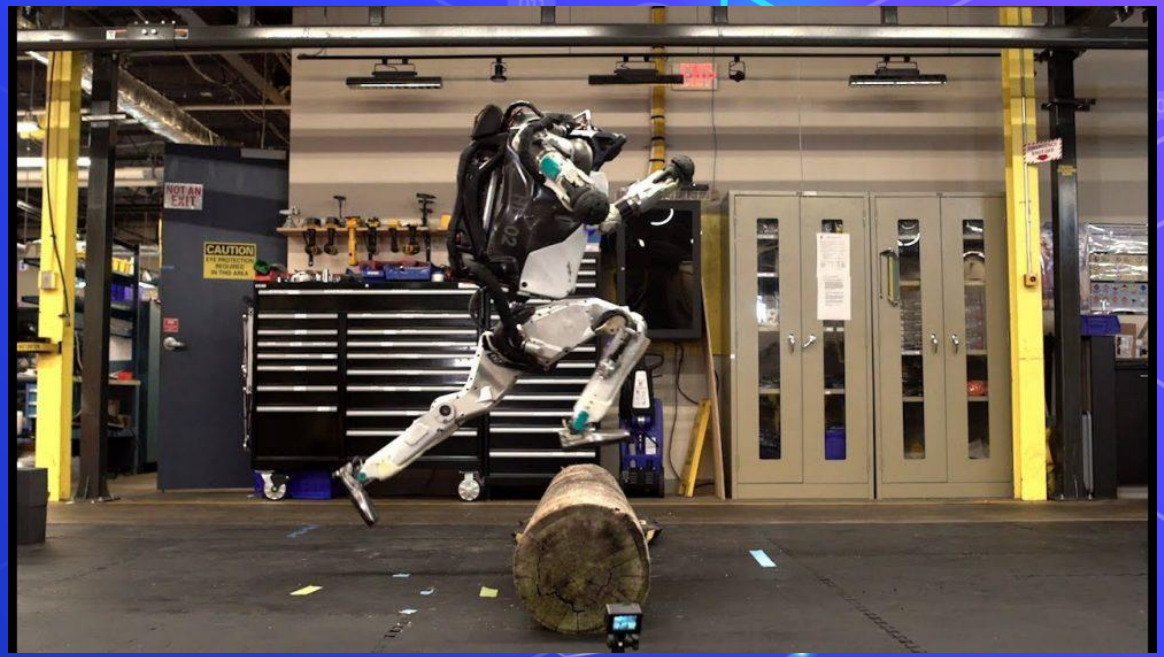
Contents

Reinforcement Learning Apps

- Robotics.
- Self Driving Cars.
- Game Playing.
- Finance.
- ...

Reinforcement Learning Apps

- **Robotics.**
- Self Driving Cars.
- Game Playing.
- Finance.
- ...



Reinforcement Learning Apps

- Robotics.
- **Self Driving Cars.**
- Game Playing.
- Finance.
- ...

Reinforcement Learning Apps

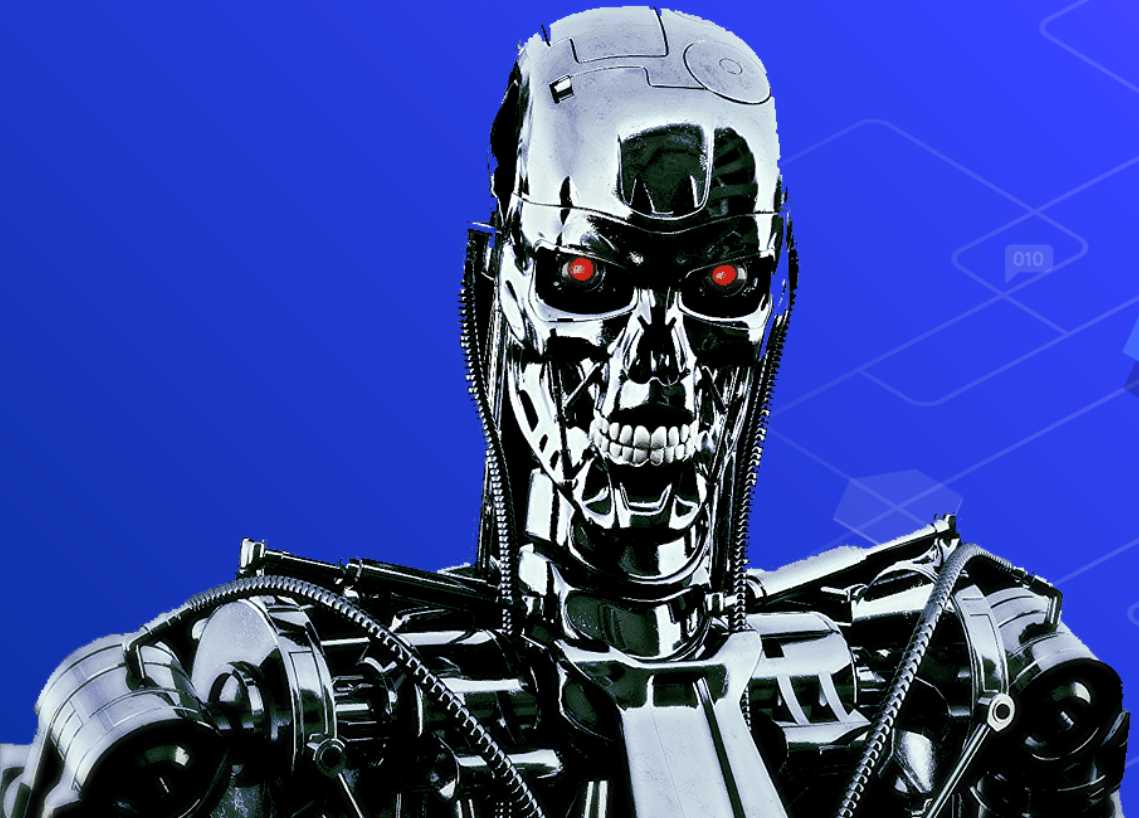
- Robotics.
- Self Driving Cars.
- **Game Playing.**
- Finance.
- ...

Reinforcement Learning Apps

- Robotics.
- Self Driving Cars.
- Game Playing.
- **Finance.**
- ...

Contents

Why RL is Creepy ?



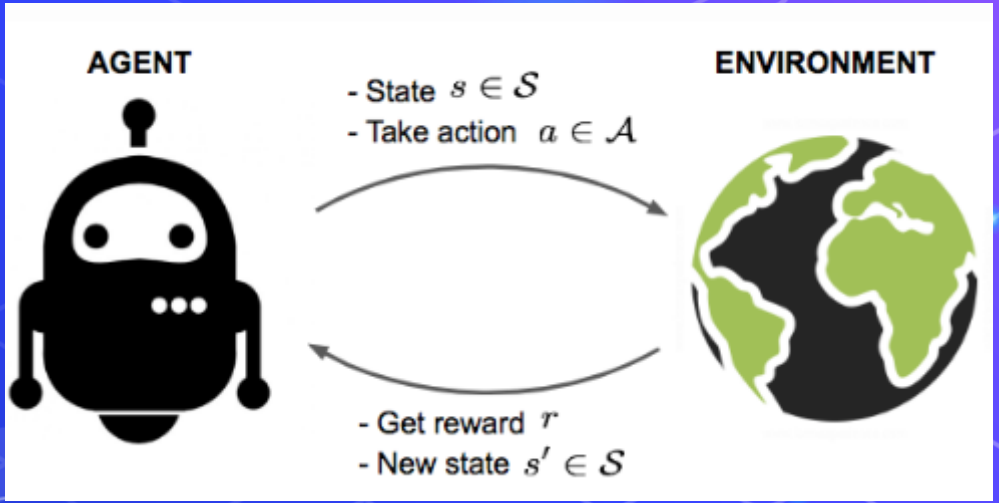
Why RL is Creepy ?

Why RL is Creepy ?

Contents

Markov Decision process (MDP)

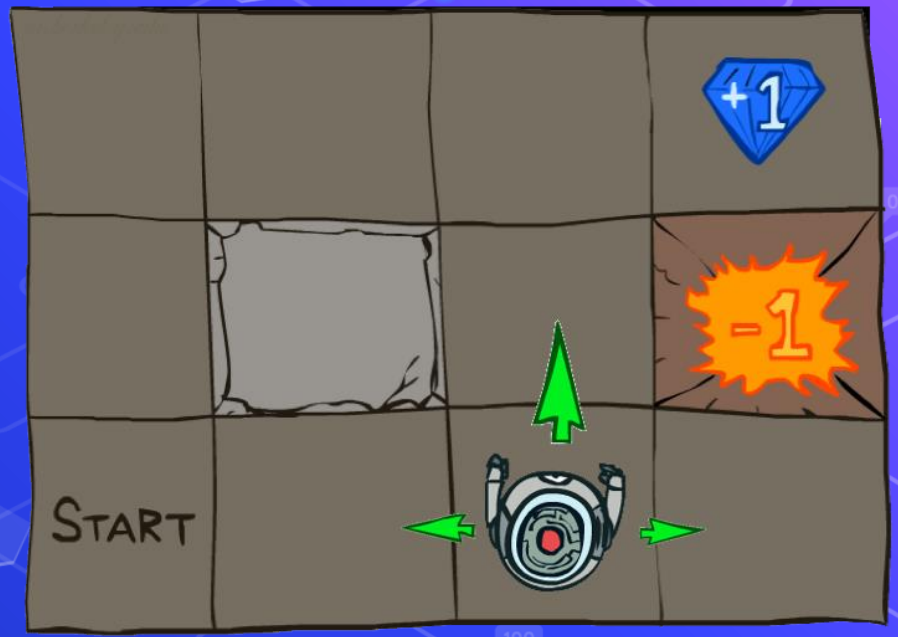
- Agent
- Environment
- Set of States [S]
- Set of Actions [A]
- Expected Rewards for taking an action in a state *[R(s, a)]*
- *Goal is to find the optimal action taken in any state to get the maximum rewards (Optimal Policy).*



Contents

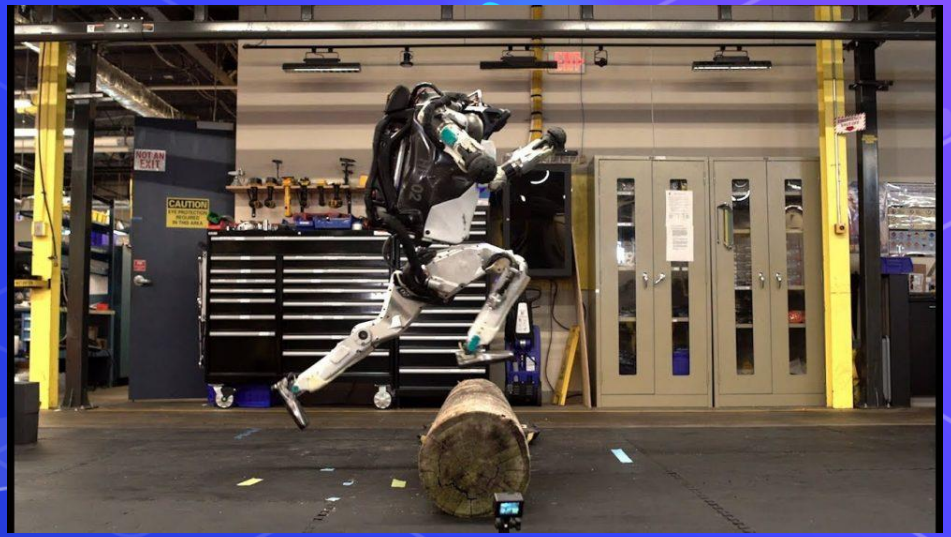
Markov Decision process (MDP) - Maze

- Set of States
  - Robot position in grid
- Set of Actions
  - Up, Down, Right, Left
- Rewards
  - (+1) if get the diamond.
  - (-1) if go to the fire.
  - ...



Markov Decision process (MDP) - Robot

- Set of States
  - Camera and Sensors.
- Set of Actions
  - Torque on the joints to walk or jump.
- Rewards
  - (+10) if not fall while jumping.
  - (-5) if apply high torque on joints.
  - ...



Markov Decision process (MDP) - Self driving car

- Set of States
  - Camera and Sensors.
- Set of Actions
  - Steer, Throttle, Break.
- Rewards
  - (-1000) if hit human.
  - (-500) if hit a car.
  - ...

Contents

MDP Solution.

- Policy.
- Exploration vs Exploitation.
- Reward Function.
- Discounted Reward Factor.
- Q-Values.

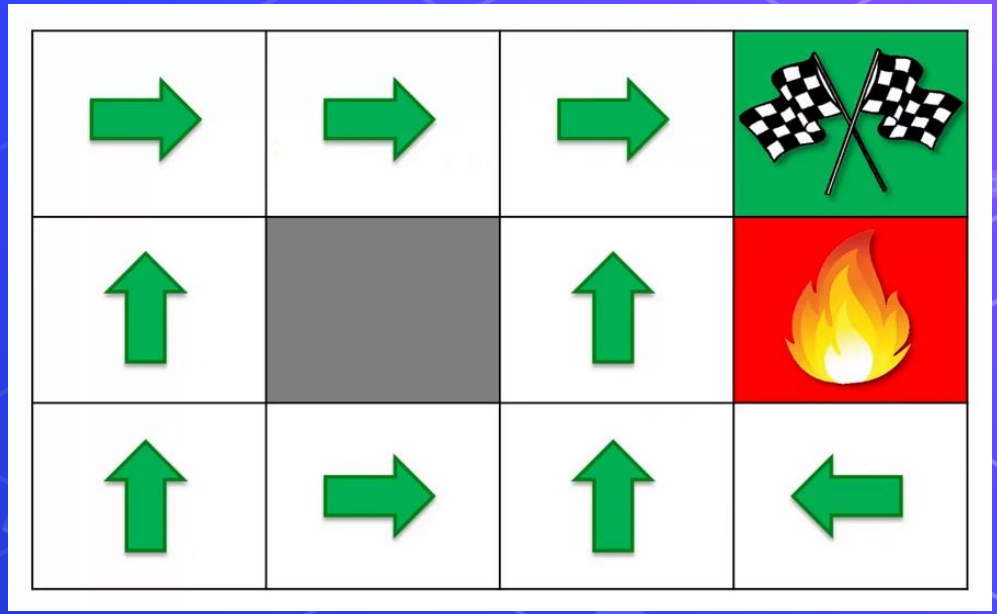
MDP Solution.

- **Policy.**
- Exploration vs Exploitation.
- Reward Function.
- Discounted Reward Factor.
- Q-Values

Policy

The set of actions for all the states that solves the MDP and get to the Goal.

The **Optimal** policy is the best actions led to the maximum reward for solving the goal.



MDP Solution.

- Policy.
- **Exploration vs Exploitation.**
- Reward Function.
- Discounted Reward Factor.
- Q-Values

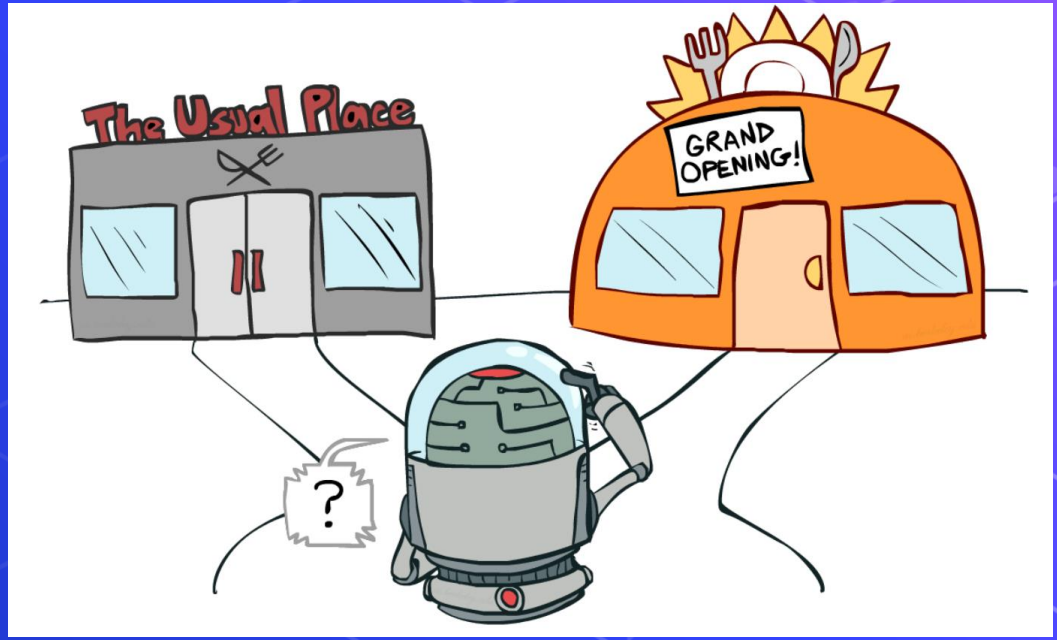
Exploration vs Exploitation

Exploration

Keep taking random actions to gain more knowledge about the environment.

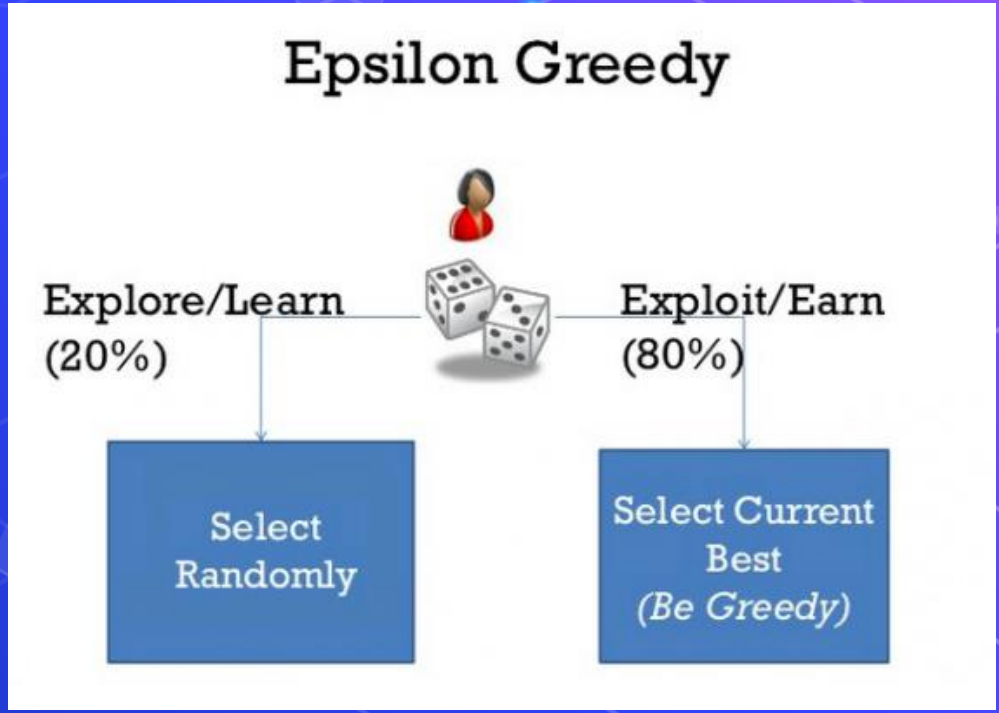
Exploitation

Use your gained knowledge to take actions based on your collected experience.



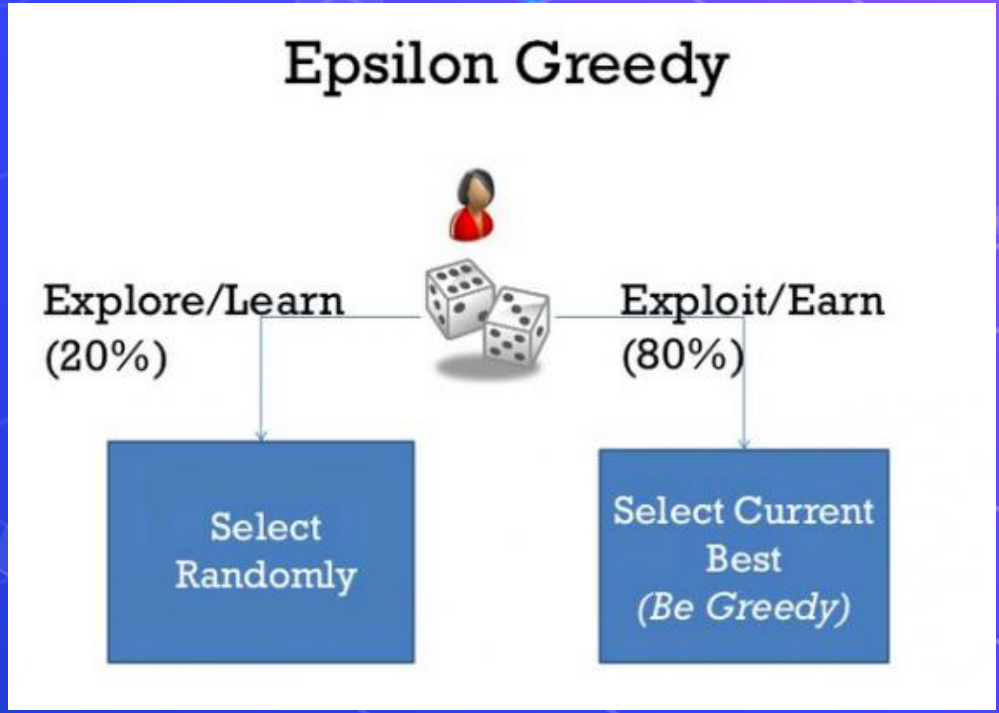
Exploration vs Exploitation (Epsilon-Greedy algorithm)

- Epsilon = 0.2 (20%) having random action.
- (1-Epsilon) = 0.8 (80%) the agent became ***greedy*** for having an action from collected experience.



Exploration vs Exploitation (Epsilon-Greedy algorithm)

- At first we set epsilon to 1 (100%) so at first the agent will explore the environment (taking random actions).
- Then gradually decaying the epsilon value (start explore then gradually exploit while collecting knowledge).
- At the end epsilon will be ~ 0 so that the agent now has a great knowledge about the environment and no need to take random actions any more.



Exploration vs Exploitation (Epsilon-Greedy algorithm)

Implementation

1- Generate a random number between 0 & 1.

2- If the number is > current epsilon value then exploit the knowledge.

3- If the number is < current epsilon value then explore random action.

4- Decay the epsilon over time.


```python
if random_num > epsilon:
    # choose action via exploitation
else:
    # choose action via exploration
```

MDP Solution

- Policy.
- Exploration vs Exploitation.
- **Reward Function.**
- Discounted Reward Factor.
- Q-Values

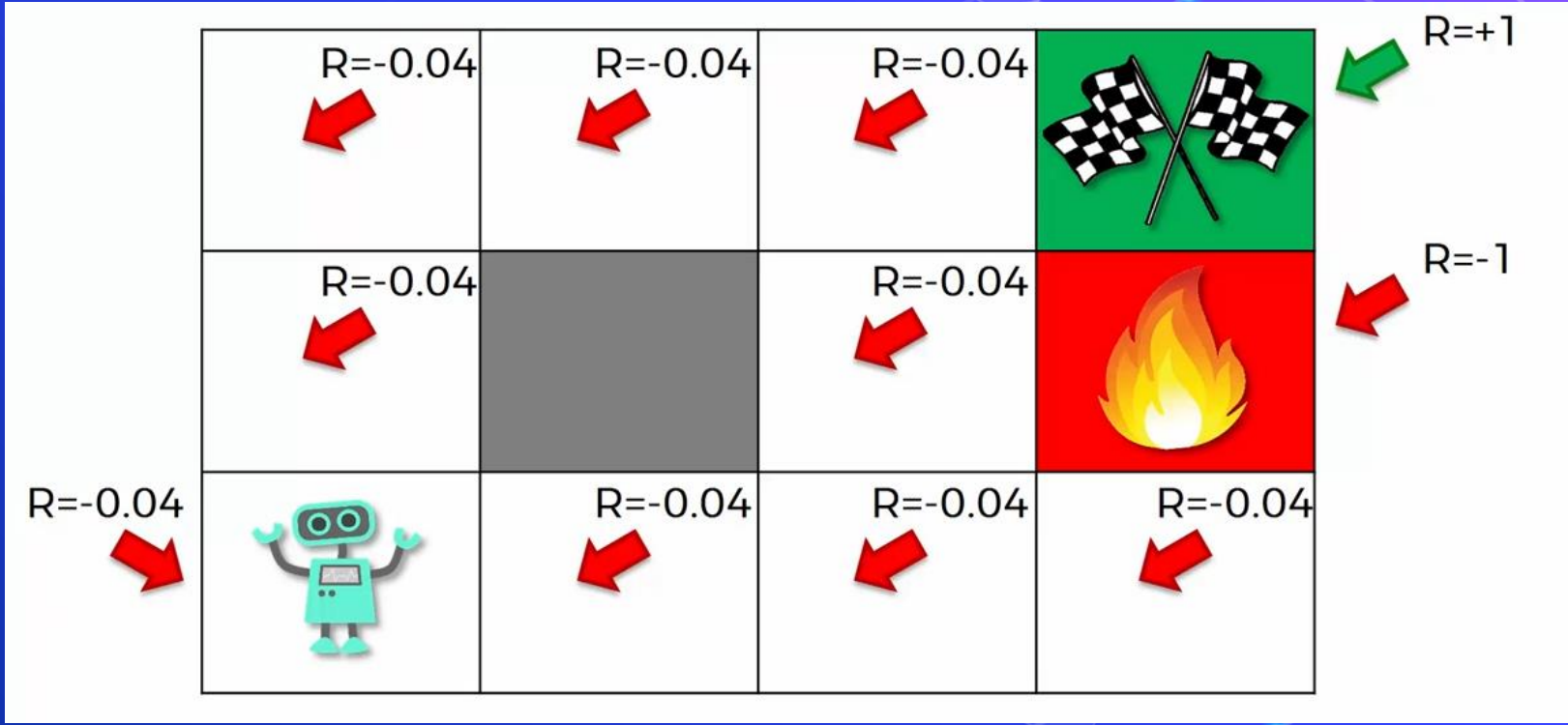
Reward Function

The goal of an agent in an MDP is to maximize its cumulative rewards. For this, we introduce the concept of the expected return of the rewards at a given time step.

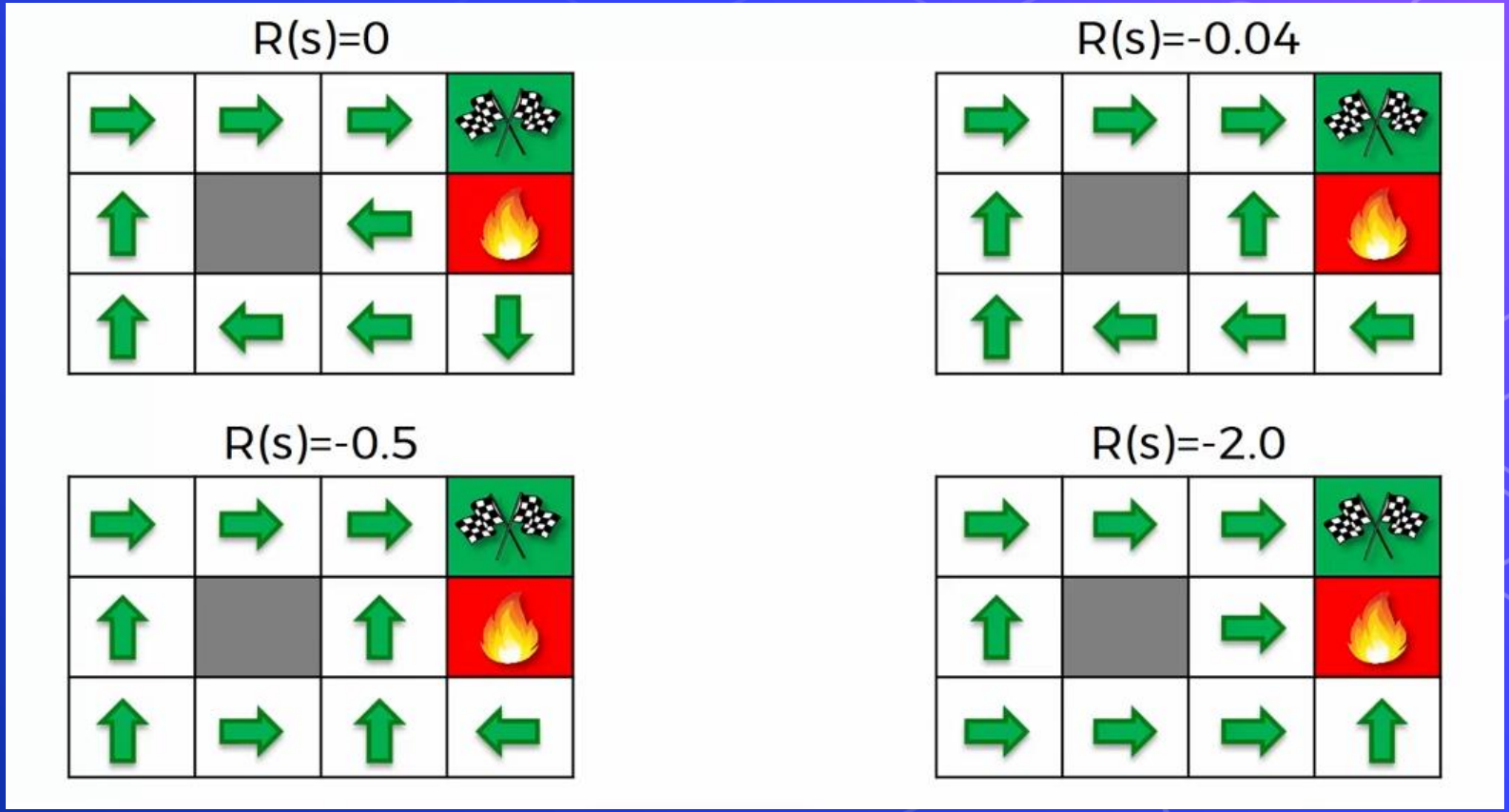
The expected return of the rewards is what's driving the agent to make the decisions it makes.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

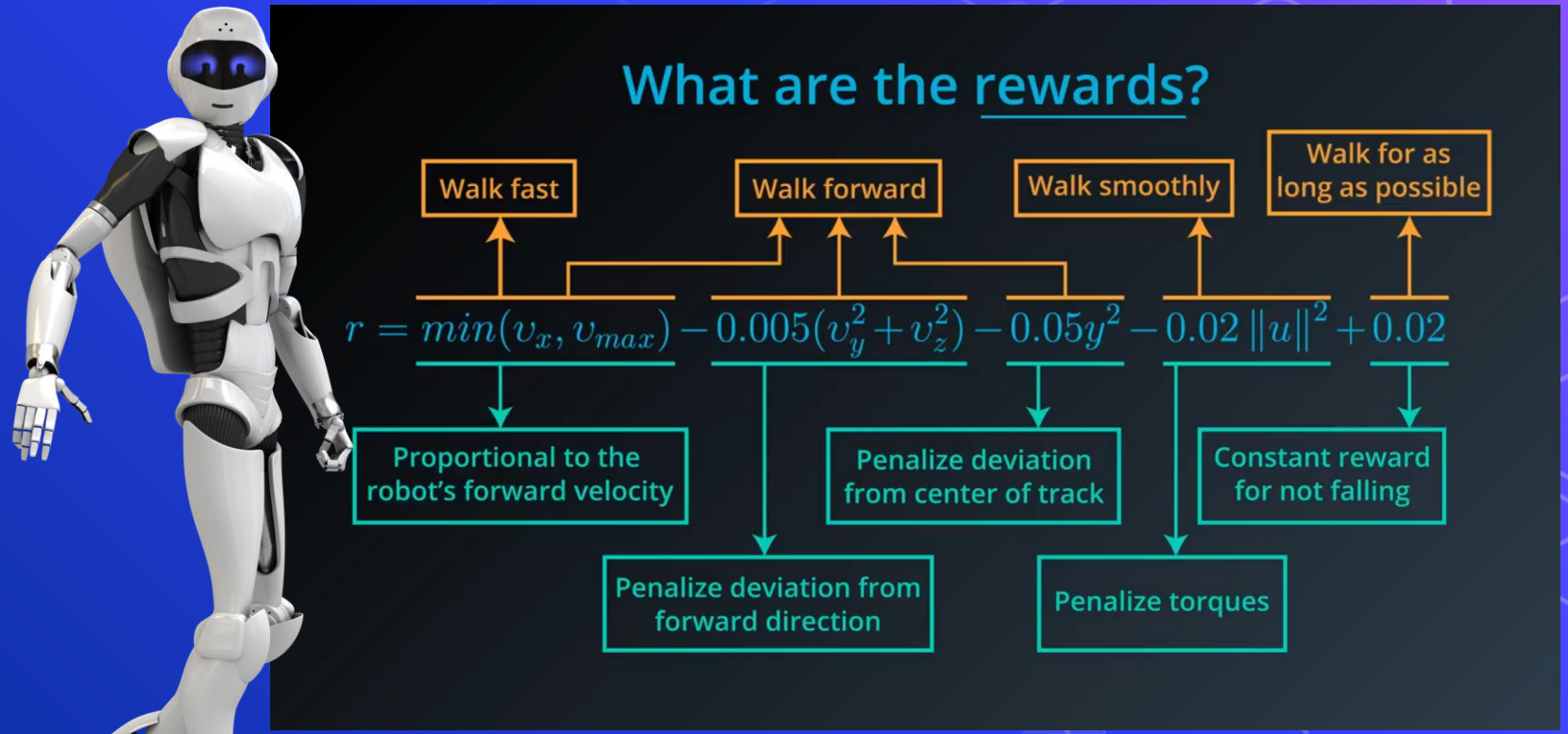
Reward Function



Reward Function



Reward Function



MDP Solution

- Policy.
- Exploration vs Exploitation.
- Reward Function.
- **Discounted Reward Factor.**
- Q-Values

Discounted Reward Factor

Do you prefer to have one Million Dollars now or tomorrow ?!

Discounted Reward Factor

For sure you will choose having the money now rather than tomorrow, because immediate rewards are the most valuable and more important than the next future rewards.

So to make the agent cares a lot of immediate rewards more than the future rewards, we added the discount factor (gamma = 0.9).

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\
&= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots \right) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

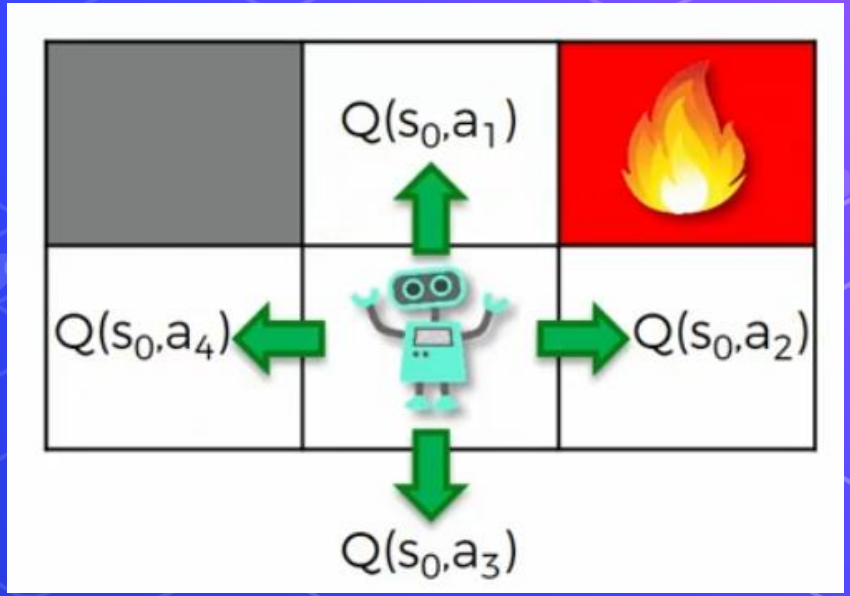
MDP Solution

- Policy.
- Exploration vs Exploitation.
- Reward Function.
- Discounted Reward Factor.
- Q-Values

Q-Values

Q-Values for a state are the probabilities of taking an action between set of actions in a given state.

For example: in this state the robot have 4 actions (up, down, right, left) for this state, we make 4 q-values one for each action, the highest q-value for an action is the optimal action to take.



Contents

Q Learning Algorithm

1. Initialize Q(s, a) by setting all of the Q-Values for each state equal to zero, epsilon = 1.
2. Observe the current state.
3. Based on the epsilon-greedy policy, choose an action (explore or exploit).
4. Take action A and observe the resulting reward, and the new state of the environment S'.
5. Update Q(s, a) based on the update rule.

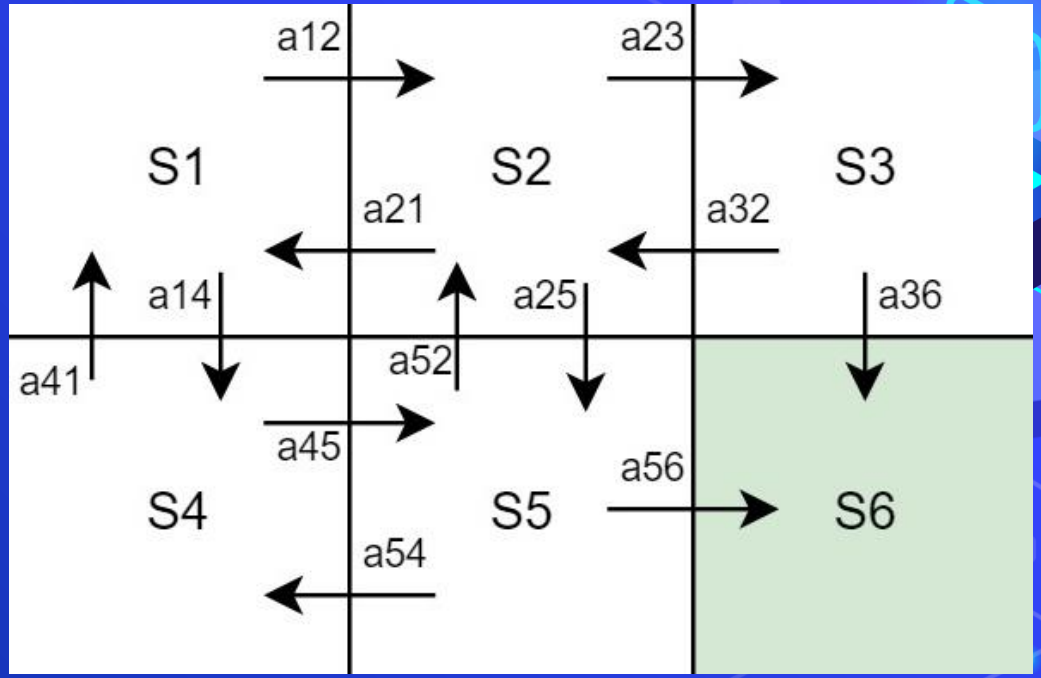
$$q^{new}(s,a) = (1-\alpha)\underbrace{q(s,a)}_{\text{old value}} + \alpha\overbrace{\left(R_{t+1} + \gamma \max_{a'} q(s',a')\right)}^{\text{learned value}}$$

Where R[t+1] is the reward, alpha is the learning rate and gamma is the discount rate = 0.9, Q(s', a') is the maximum q-value for an action at the next state.

6. S = S' & Decay epsilon then Repeat steps 2–6 until convergence.

Q Learning Algorithm (Grid World)

Consider a grid world environment that a robot wants to reach the flag at state 6.



Q Learning Algorithm (Grid World)

1. Initialize problem by this settings:

- All of the Q-Values for each state equal to zero.
- Epsilon = 1.
- Gamma = 0.5.
- Alpha = 1.
- The flag reward is +100 at state S6 otherwise zero.

When alpha = 1 for simplicity, the equation became

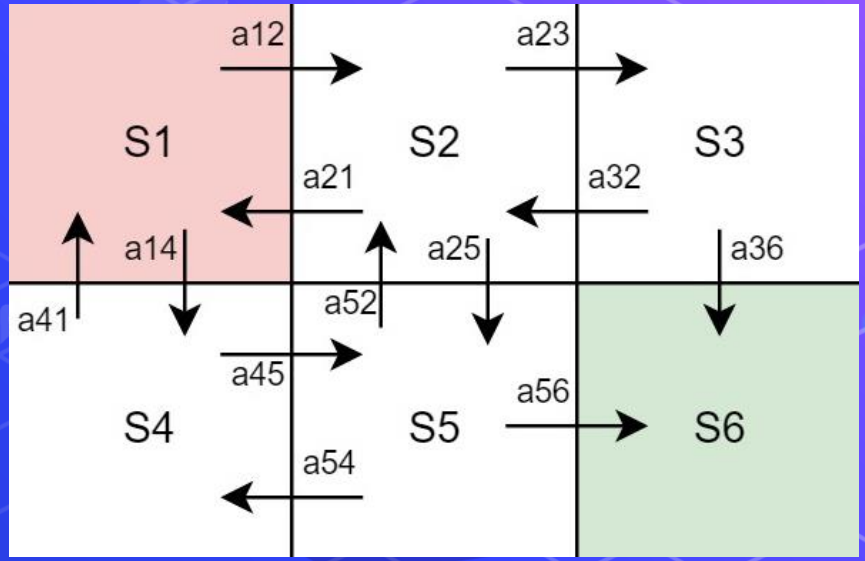
$$R_{t+1} + \gamma \max_{a'} q\left(s', a'\right)$$

Q(S, A)	Value
Q(S1, A12)	0
Q(S1, A14)	0
Q(S2, A21)	0
Q(S2, A23)	0
Q(S2, A25)	0
Q(S3, A32)	0
Q(S3, A36)	0
Q(S4, A41)	0
Q(S4, A45)	0
Q(S5, A54)	0
Q(S5, A52)	0
Q(S5, A56)	0

Q Learning Algorithm (Grid World)

2. Observe the current state S1 (the red state), available actions are (a12, a14) lets say we took action a12 so we became at state S' = S2.

3. Calculate q_new(S1, a12) with the equation using q-values from the table.



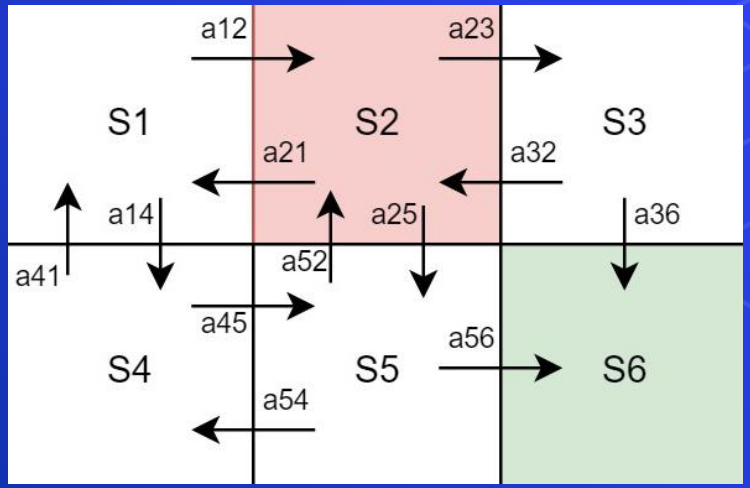
$$R_{t+1} + \gamma \max_{a'} q\left(s', a'\right)$$

```
q_new(S1, a12) = r + 0.5 * max[q(S2, a21), q(S2, a23), q(S2, a25)]
= 0 + 0.5 * 0 = 0
```

Q Learning Algorithm (Grid World)

4- Update the q table with new Q(S1, A12) value = 0.

5- New State became S2.

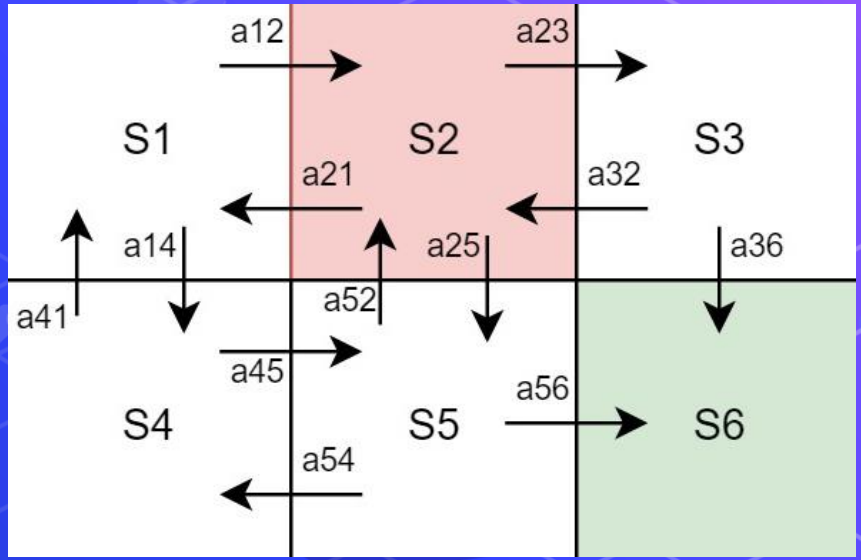


Q(S, A)	Value
Q(S1, A12)	0
Q(S1, A14)	0
Q(S2, A21)	0
Q(S2, A23)	0
Q(S2, A25)	0
Q(S3, A32)	0
Q(S3, A36)	0
Q(S4, A41)	0
Q(S4, A45)	0
Q(S5, A54)	0
Q(S5, A52)	0
Q(S5, A56)	0

Q Learning Algorithm (Grid World)

6. Observe the current state S2 (the red state), available actions are (a21, a23, a25) lets say we took action a23 so we became at state S' = S3.

7. Calculate q_new(S2, a23) with the equation using q-values from the table.



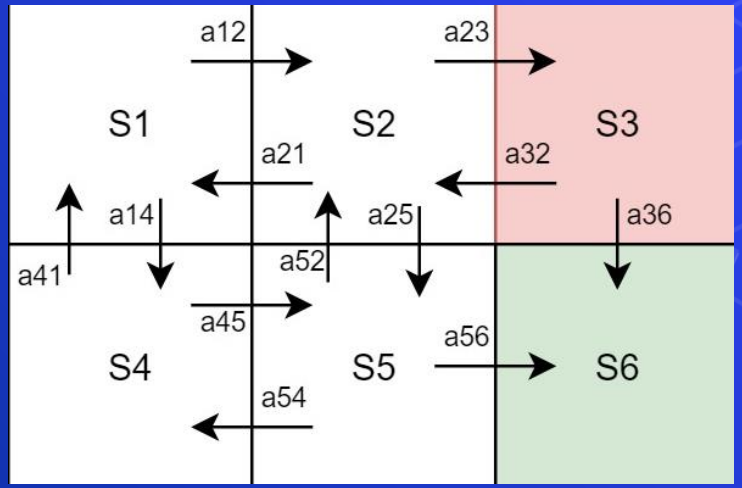
$$R_{t+1} + \gamma \max_{a'} q\left(s', a'\right)$$

```
q_new(S2, a23) = r + 0.5 * max[q(S3, a32), q(S3, a36)]
= 0 + 0.5 * 0 = 0
```

Q Learning Algorithm (Grid World)

8- Update the q table with new Q(S2, A23) value = 0.

9- New State became S3.



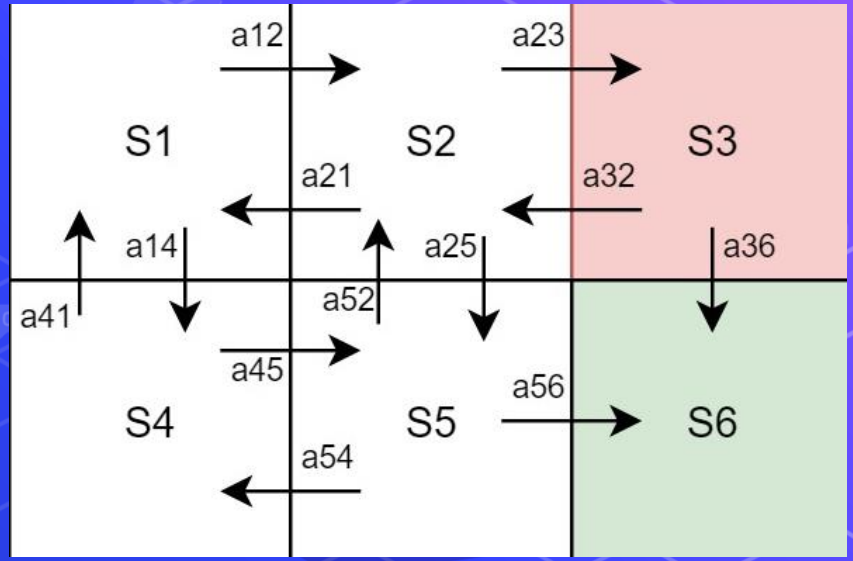
Q(S, A)	Value
Q(S1, A12)	0
Q(S1, A14)	0
Q(S2, A21)	0
Q(S2, A23)	0
Q(S2, A25)	0
Q(S3, A32)	0
Q(S3, A36)	0
Q(S4, A41)	0
Q(S4, A45)	0
Q(S5, A54)	0
Q(S5, A52)	0
Q(S5, A56)	0

Q Learning Algorithm (Grid World)

10. Observe the current state S3 (the red state), available actions are (a32, a36) lets say we took action a36 so we became at state S' = S6.

11. Calculate q_new(S3, a36) with the equation using q-values from the table.

$$R_{t+1} + \gamma \max_{a'} q\left(s', a'\right)$$

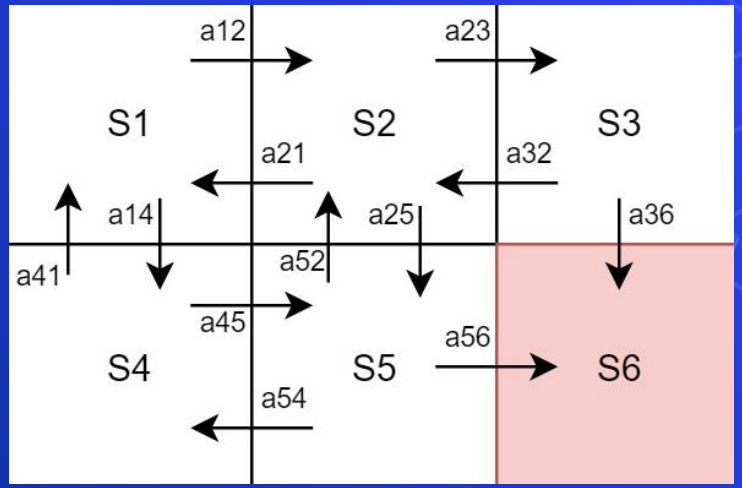


`q_new(S3, a36) = r = 100, because there is no actions in S' -> S6 (Goal)`

Q Learning Algorithm (Grid World)

12- Update the q table with new Q(S3, A36) value = 100.

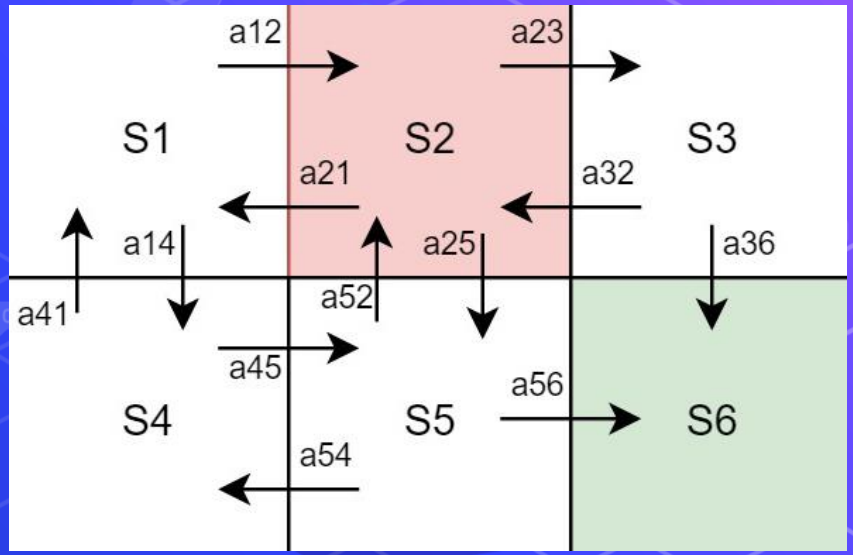
13- New State became S6 (Goal or Terminal State) and the episode ends.



Q(S, A)	Value
Q(S1, A12)	0
Q(S1, A14)	0
Q(S2, A21)	0
Q(S2, A23)	0
Q(S2, A25)	0
Q(S3, A32)	0
Q(S3, A36)	100
Q(S4, A41)	0
Q(S4, A45)	0
Q(S5, A54)	0
Q(S5, A52)	0
Q(S5, A56)	0

Q Learning Algorithm (Grid World)

14. After Episode 1 ends, Start Episode 2 with Random State for example S2 (the red state), available actions are (a21, a23, a25) lets say we took action a23 so we became at state S' = S3.



11. Calculate q_new(S2, a23) with the equation using q-values from the table.

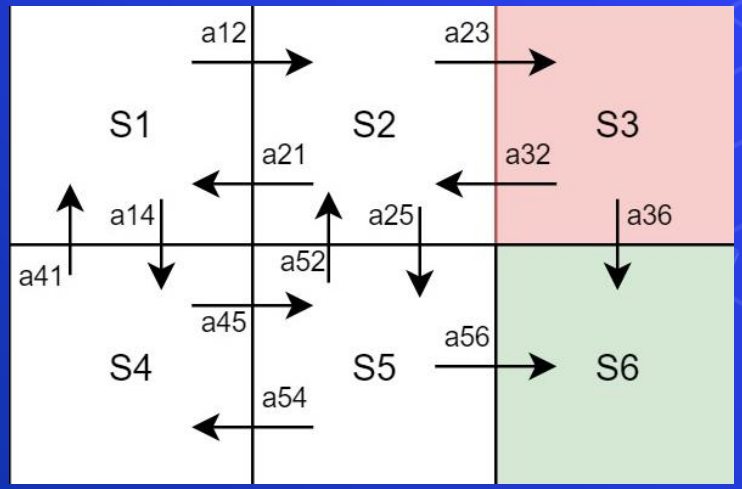
$$R_{t+1} + \gamma \max_{a'} q\left(s', a'\right)$$

```
q_new(S2, a23) = r + 0.5 * max[q(S3, a32), q(S3, a36)]
= 0 + 0.5 * 100 = 50
```

Q Learning Algorithm (Grid World)

12- Update the q table with new Q(S2, A23) value = 50.

13- New State became S3, and So on for N episodes.



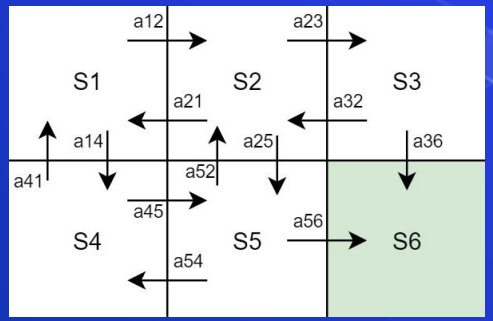
Q(S, A)	Value
Q(S1, A12)	0
Q(S1, A14)	0
Q(S2, A21)	0
Q(S2, A23)	50
Q(S2, A25)	0
Q(S3, A32)	0
Q(S3, A36)	100
Q(S4, A41)	0
Q(S4, A45)	0
Q(S5, A54)	0
Q(S5, A52)	0
Q(S5, A56)	0

Q Learning Algorithm (Grid World)

After a lot of episodes of updating the Q-Values Table,

Lets say after 200 episode we have these values on the left, so we finally know the policy to solve the grid world problem.

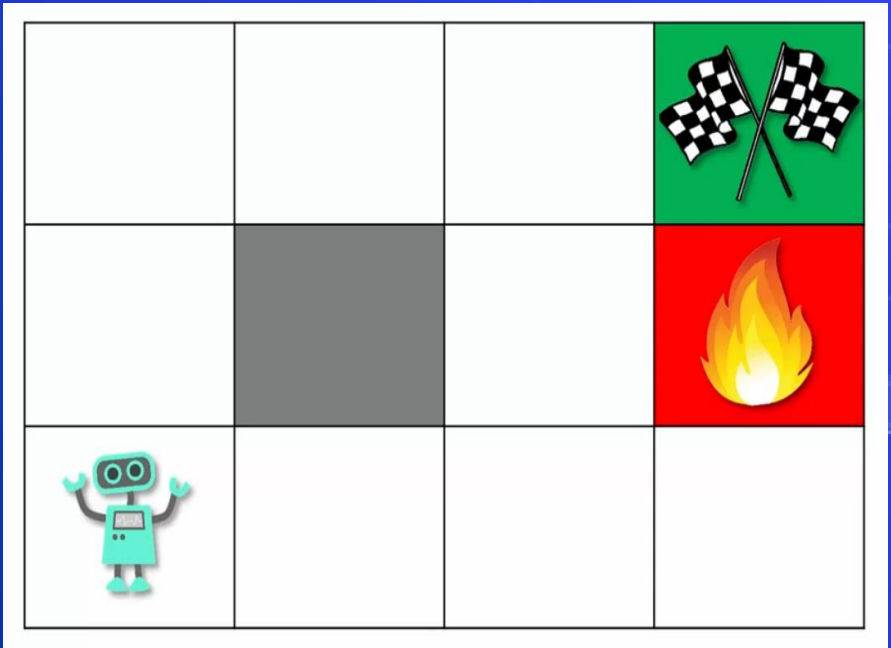
- At State 1 choose Action A12 or A14 = 25.
- At State 2 choose Action A23 = 50.
- At State 3 choose Action A36 = 100.
- At State 4 choose Action A45 = 50.
- At State 5 choose Action A56 = 100.

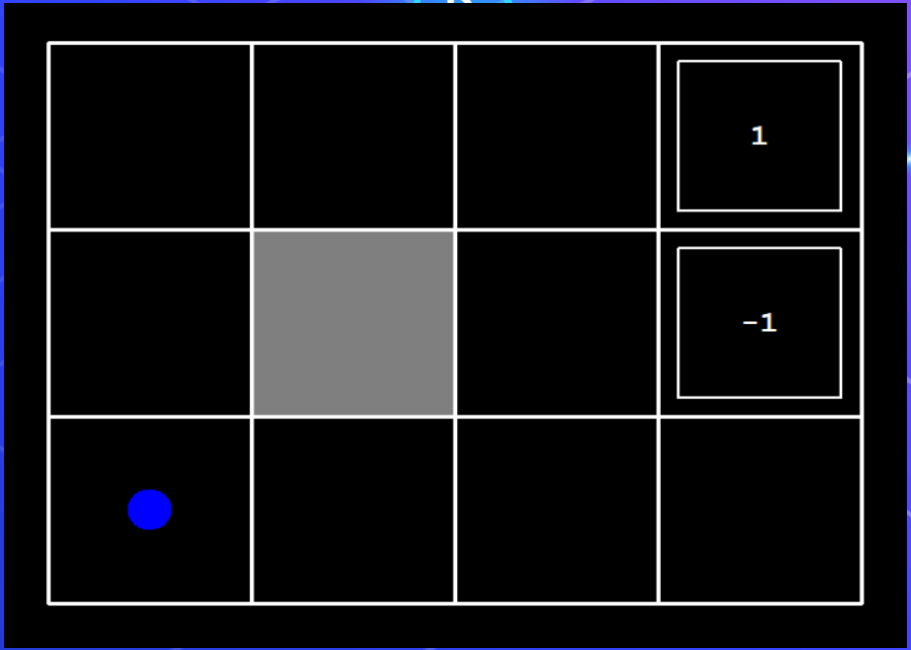


Q(S, A)	Value
Q(S1, A12)	25
Q(S1, A14)	25
Q(S2, A21)	12.5
Q(S2, A23)	50
Q(S2, A25)	25
Q(S3, A32)	25
Q(S3, A36)	100
Q(S4, A41)	12.5
Q(S4, A45)	50
Q(S5, A54)	25
Q(S5, A52)	25
Q(S5, A56)	100

Q Learning Algorithm (Another Grid World)

Consider a grid world environment that a robot wants to reach the flag and stay away from fire.





Q Learning Algorithm (Another Grid World)

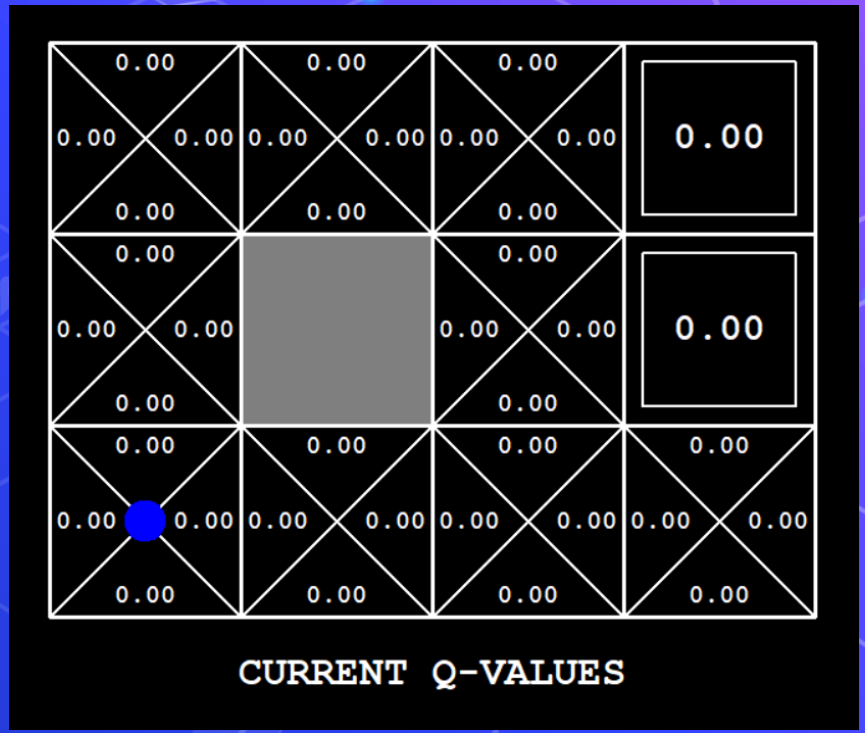
1. Initialize Q(s, a) by setting all of the Q-Values for each state equal to zero, epsilon = 1, gamma = 0.9, alpha = 0.7, the flag reward is +1 & fire penalty is -1 and every step has penalty 0, number of episodes = 400.

2. Observe the current state (the blue circle).

3. Based on the epsilon-greedy policy, choose an action, At first epsilon = 1, so the agent will take random actions for the first episodes.

4. after a while the agent gains some knowledge so it should be greedy and exploit its knowledge, after a lot of episodes epsilon became very small thus a little explored random actions is taken.

5. After taking an action, observe the new state and the resulting reward.



Q Learning Algorithm (Another Grid World)

6. The episode ends when the agent reaches the flag or the fire, lets say the agent is in state [0, 2] it choses action right then it reaches the flag and gains the reward 1, so it knows that the action (right) led to the flag, it's a good action so updates the q-value of the action with the equation below

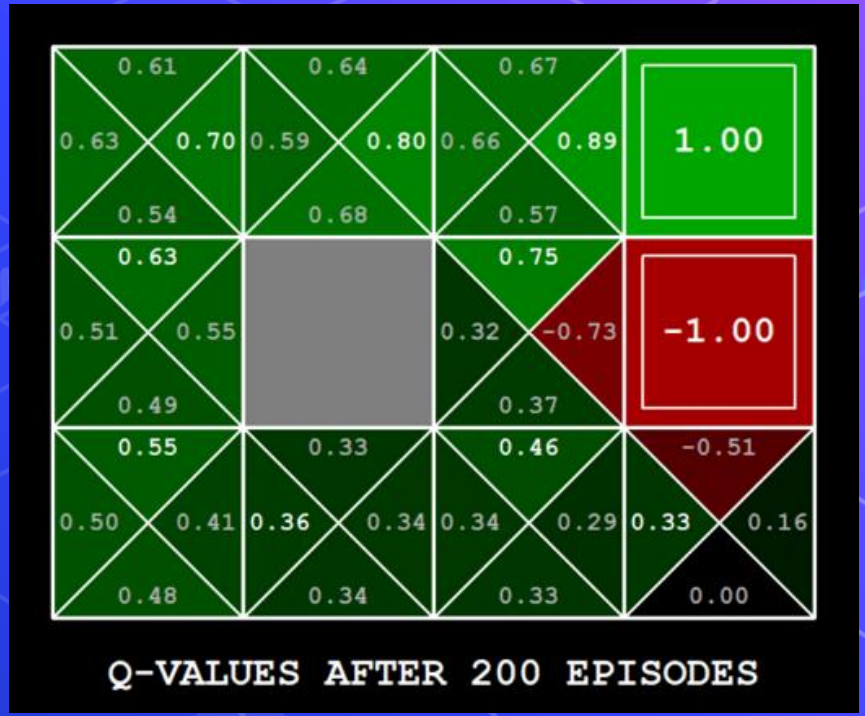
```
q_new([0, 2], Right) = (1-0.7) * 0 + 0.7 (1) = 0.7
```

$$q^{new}(s,a) = (1-\alpha)\underbrace{q(s,a)}_{\text{old value}} + \alpha\overbrace{\left(R_{t+1} + \gamma \max_{a'} q(s',a')\right)}^{\text{learned value}}$$

7. At a next step we ended for example at state [0, 1] and took action right, then the new q is

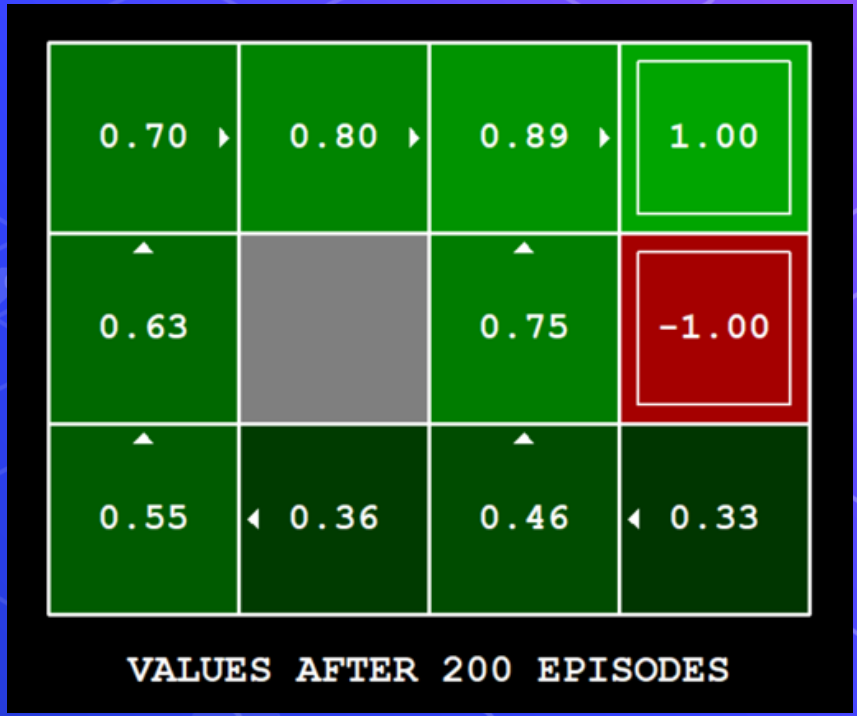
```
q_new([0, 1], Right) = (1-0.7) * 0 + 0.7 (0 + 0.9 * 0.7)
= 0.441
```

8. Repeat the steps for many episodes, and keep updating Q-values, after many episodes it will be like the image.



Q Learning Algorithm (Another Grid World)

So at the end we will have the optimal policy that can plays the game with its own to have the maximum reward.



Q Learning Algorithm (Another Grid World)

To run the code.

```
python gridworld.py -k 400 -a q -s 10 -r -0.3
```

Open AI Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.

https://gym.openai.com/

```
1 pip install gym
```

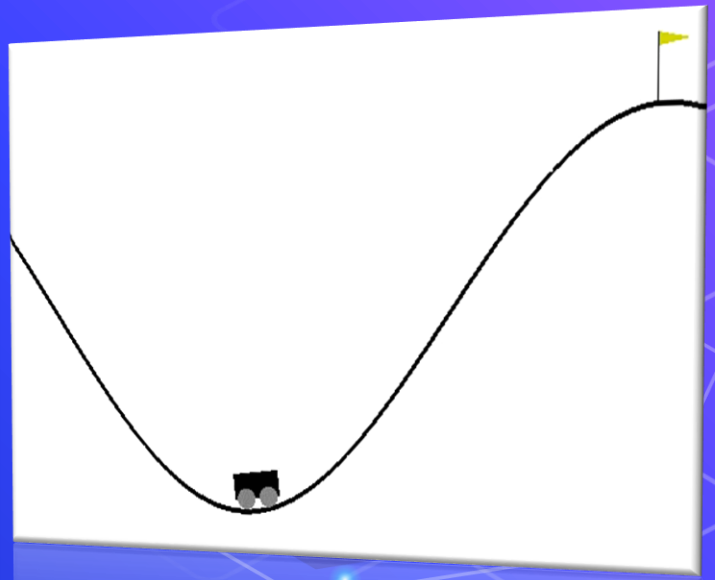
Open AI Gym – Mountain Car

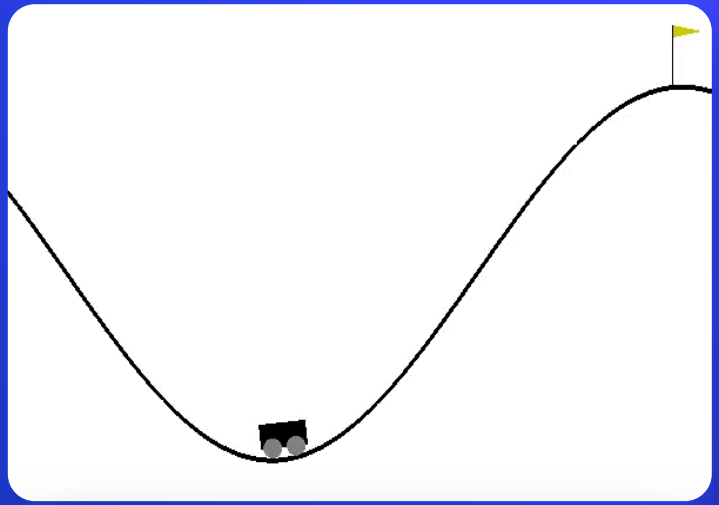
```python
import gym

env = gym.make('MountainCar-v0')

for i_episode in range(400):
    observation = env.reset() # get random state S0
    action = env.action_space.sample() # take random action
    observation, reward, done, info = env.step(action)
    env.render()
    if done:
        break

env.close()
```





```python
import gym
import numpy as np

env = gym.make("MountainCar-v0")

lr = 0.5
DISCOUNT = 0.95
EPISODES = 80
START_EPS_DECAY = 1
END_EPS_DECAY = EPISODES//2
epsilon = 1
eps_decay = epsilon/(END_EPS_DECAY - START_EPS_DECAY)

q_table = np.zeros(([10,10] + [env.action_space.n]))

```

```python
episode = 0
while episode < EPISODES:

    done = False
    discrete_state = calc_discrete_state(env.reset())

    while not done:

        # Exploit or explore
        if np.random.random() > epsilon:
            # Exploit from q-table
            action = np.argmax(q_table[discrete_state])
        else:
            # Explore - t
            action = np.random.randint(0, env.action_space.n)


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

        # Update q-table
        max_future_q = np.max(q_table[new_state_disc])
        current_q = q_table[discrete_state + (action,)]
        reward_fun = reward + DISCOUNT * max_future_q
        new_q = (1 - lr) * current_q + lr * reward_fun
        q_table[discrete_state + (action,)] = new_q

        discrete_state = calc_discrete_state(new_state_disc)

        env.render()

    if END_EPS_DECAY >= episode >= START_EPS_DECAY:
        epsilon -= epsilon_change

    episode += 1
```

Questions ?!

Thanks!

>_ Live long and prosper