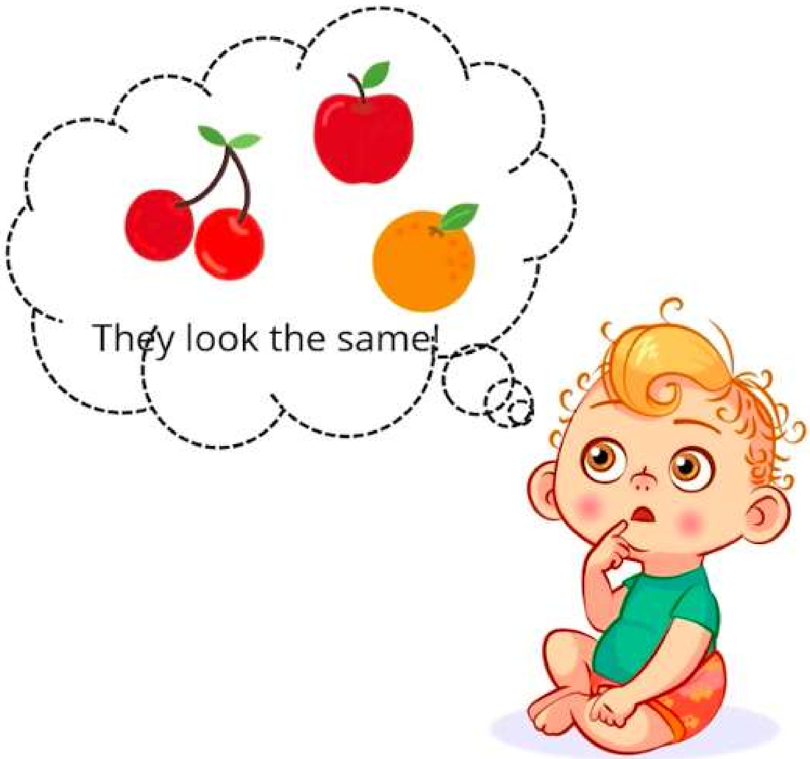
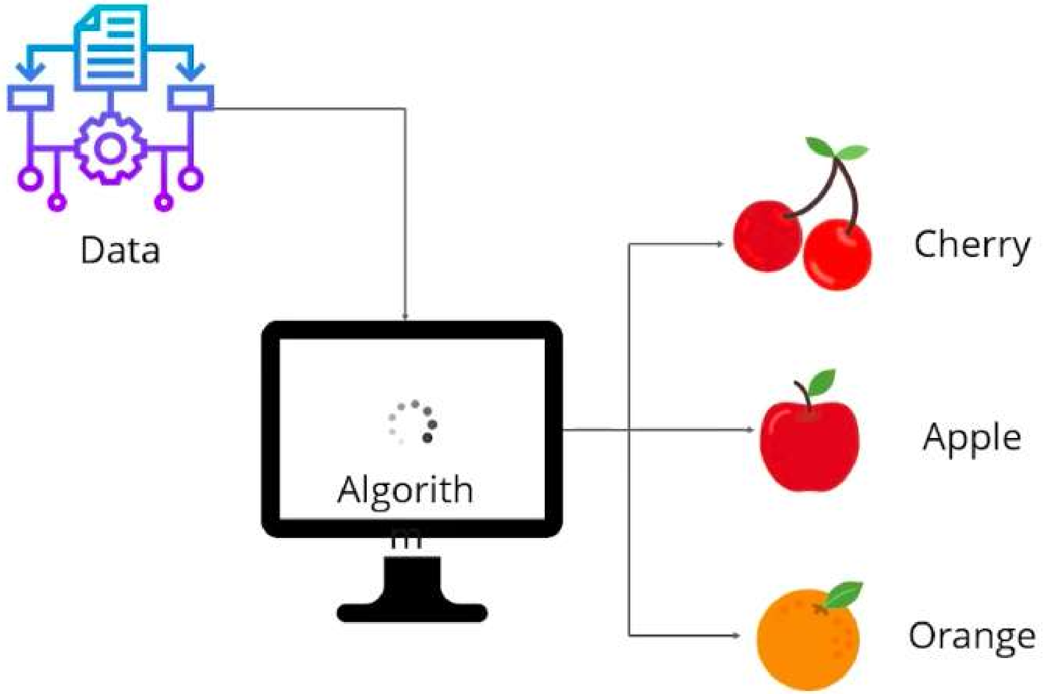
Agenda

1. Introduction To Machine Learning
2. What is Reinforcement Learning?
3. Reinforcement Learning with an Analogy
4. Reinforcement Learning Process
5. Reinforcement Learning – Counter strike example
6. Reinforcement Learning Definitions
7. Reinforcement Learning Concepts
8. Markov's Decision Process
9. Understanding Q-Learning
10. Hands-On

What Is Machine Learning?

Machine learning is a subset of artificial intelligence (AI) which provides machines the ability to learn automatically & improve from experience without being explicitly programmed.





Types Of Machine Learning

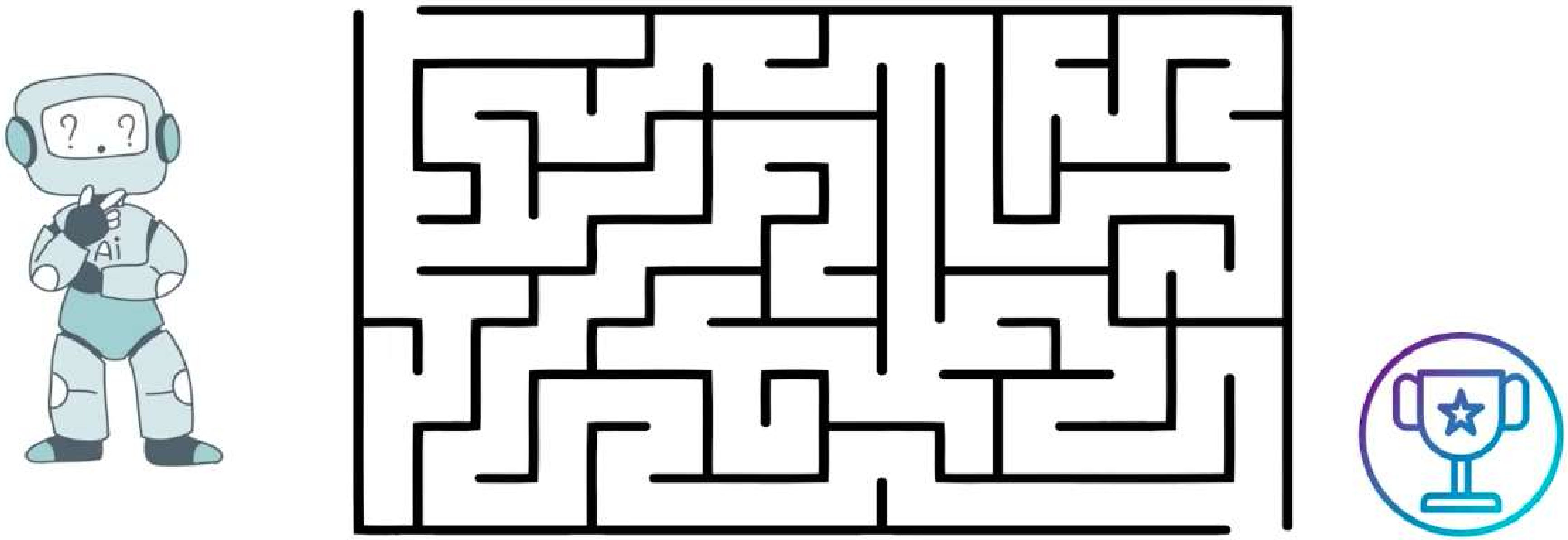
Supervised Learning

Unsupervised Learning

Reinforcement Learning

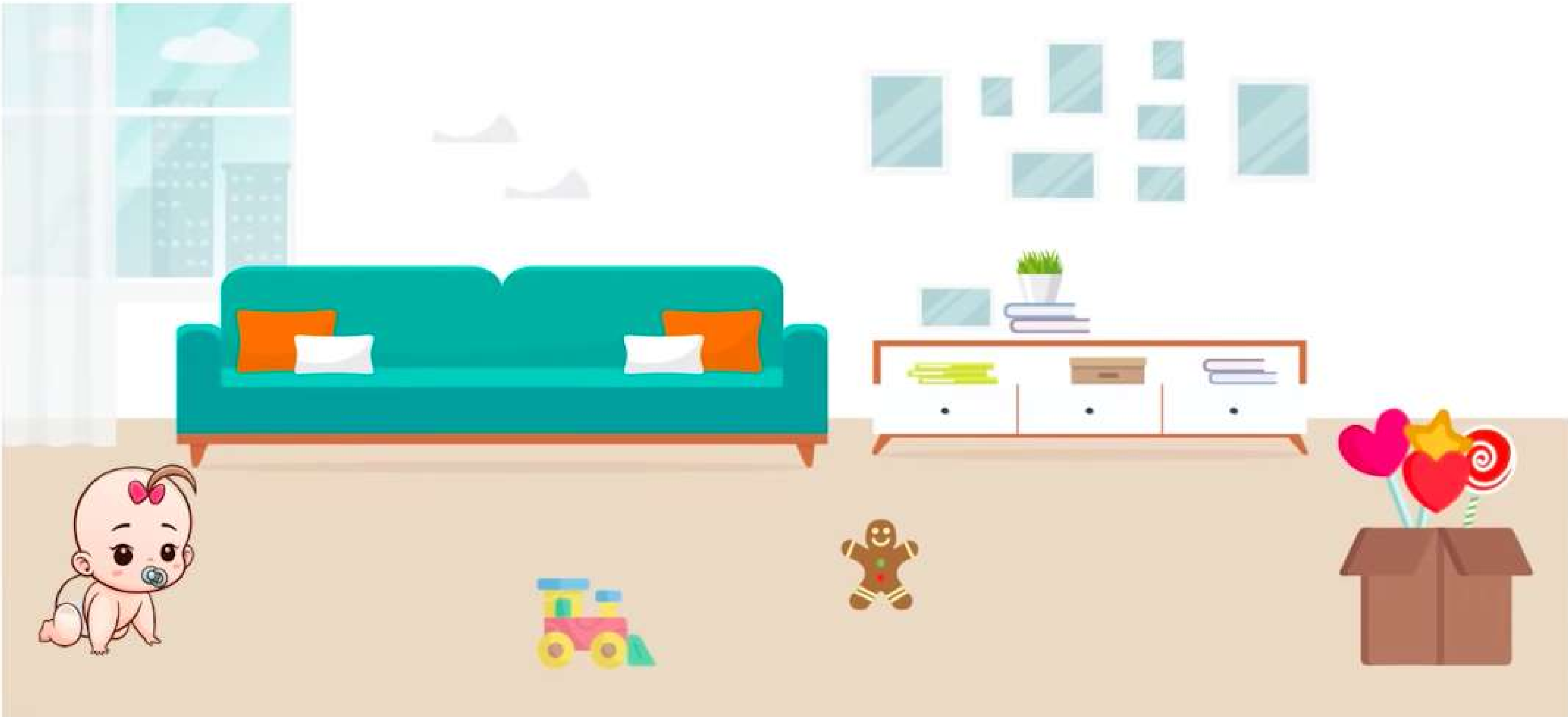
What Is Reinforcement Learning?

Reinforcement learning is a type of Machine Learning where an agent learns to behave in a environment by performing actions and seeing the results



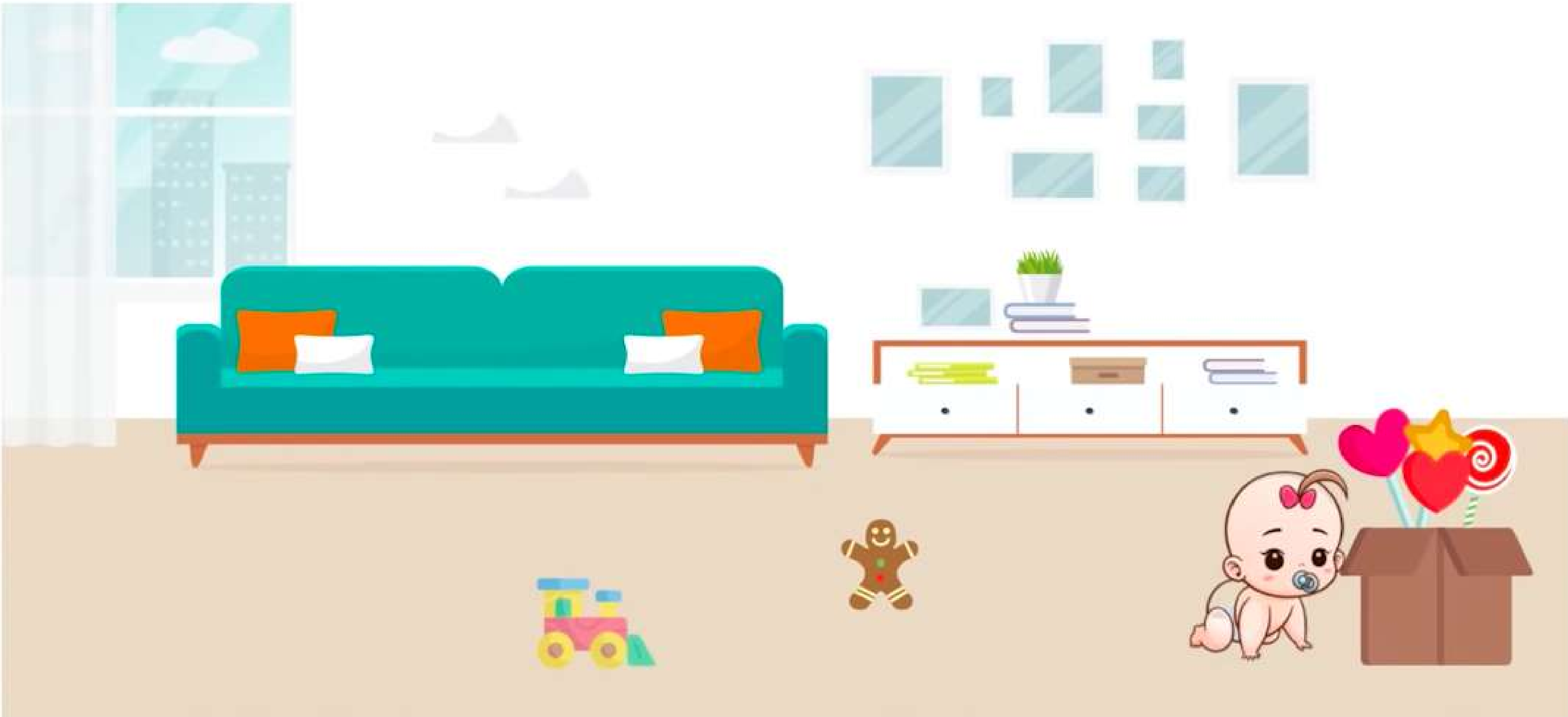
Reinforcement Learning With An Analogy

Scenario 1: Baby starts crawling and makes it to the candy



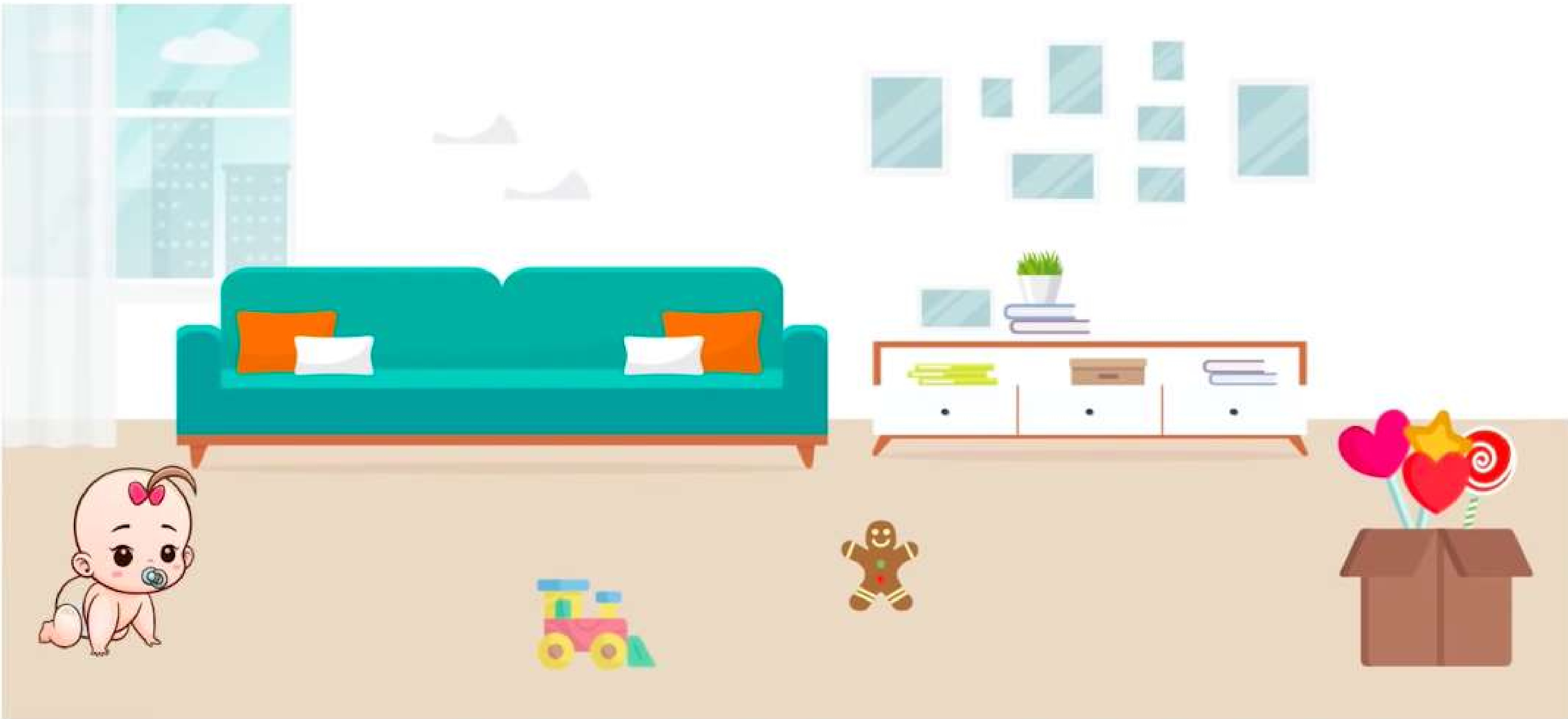
Reinforcement Learning With An Analogy

Scenario 1: Baby starts crawling and makes it to the candy



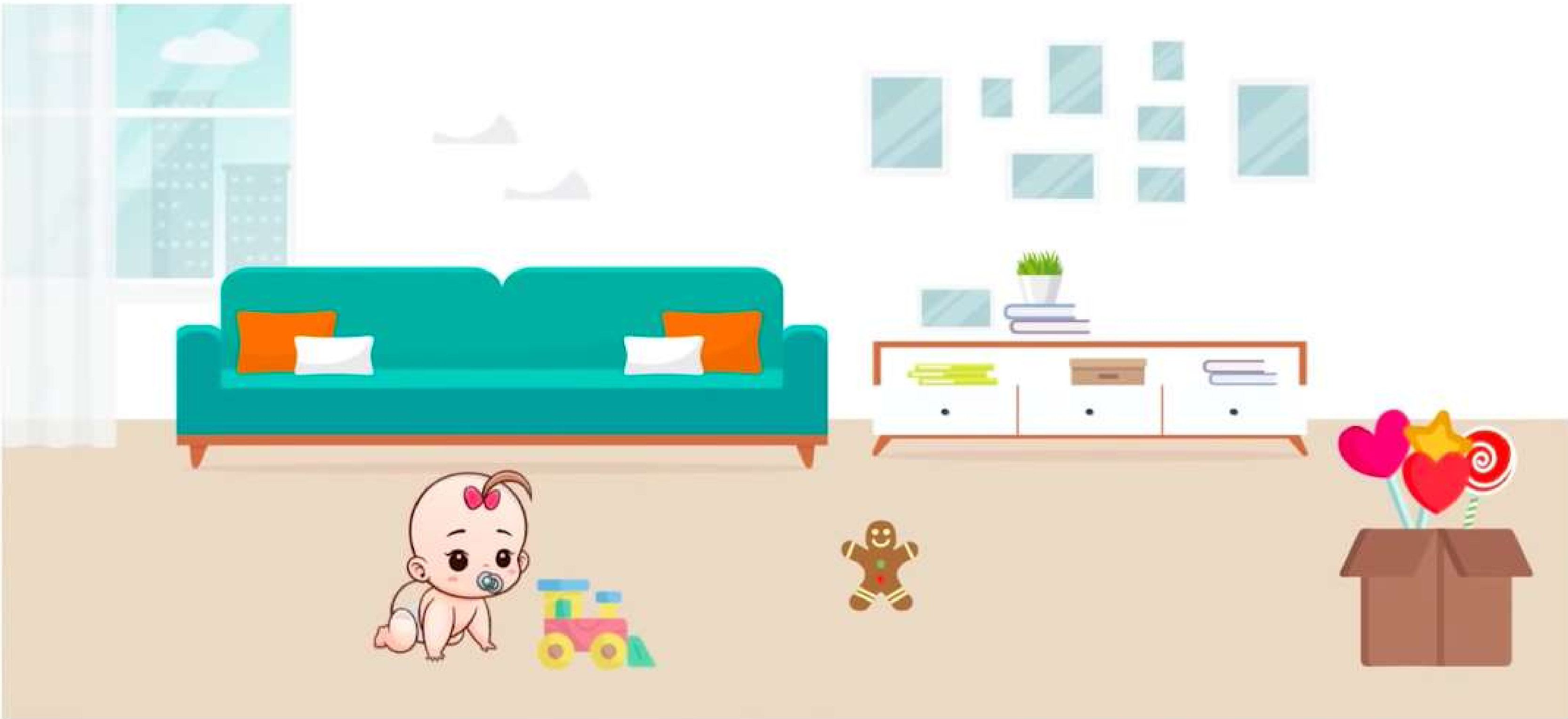
Reinforcement Learning With An Analogy

Scenario 2: Baby starts crawling but falls due to some hurdle in between



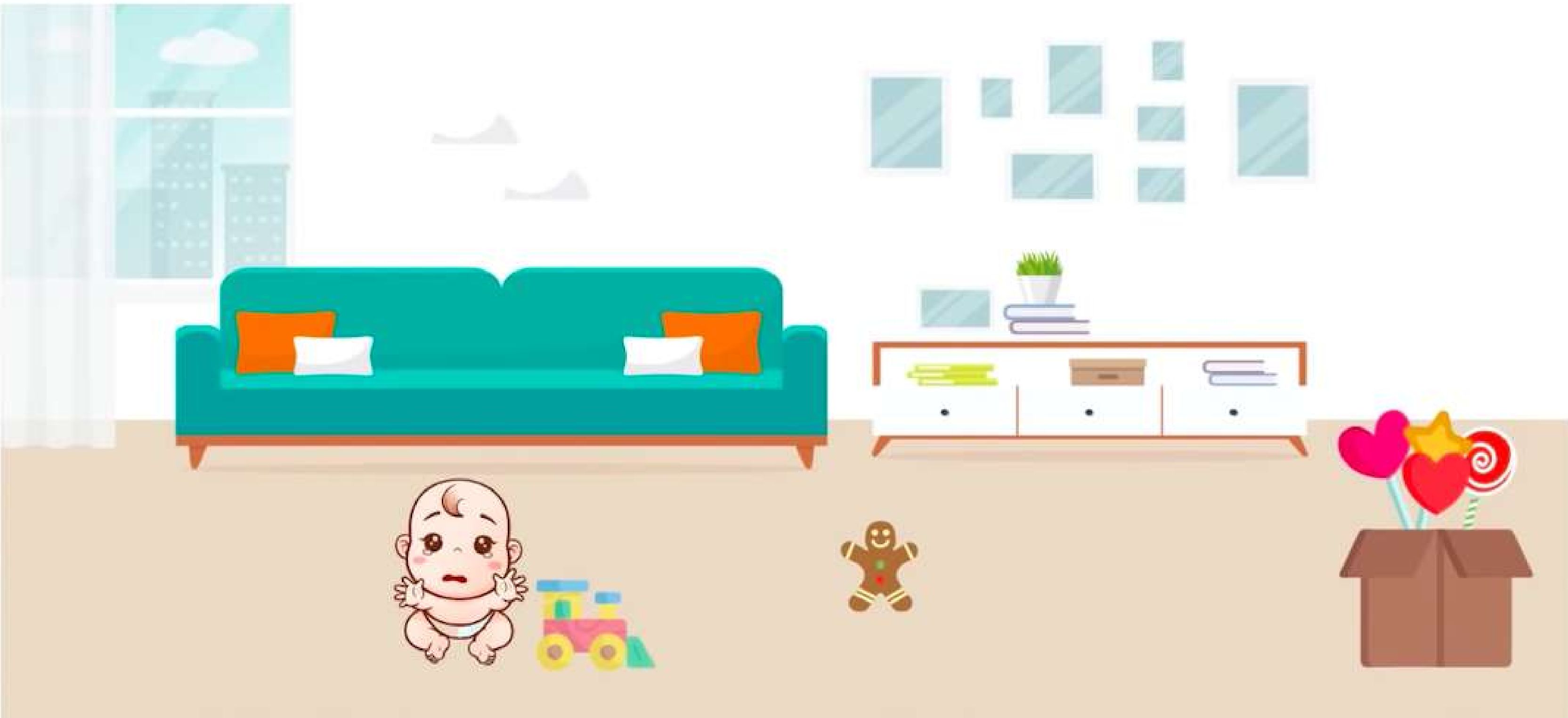
Reinforcement Learning With An Analogy

Scenario 2: Baby starts crawling but falls due to some hurdle in between



Reinforcement Learning With An Analogy

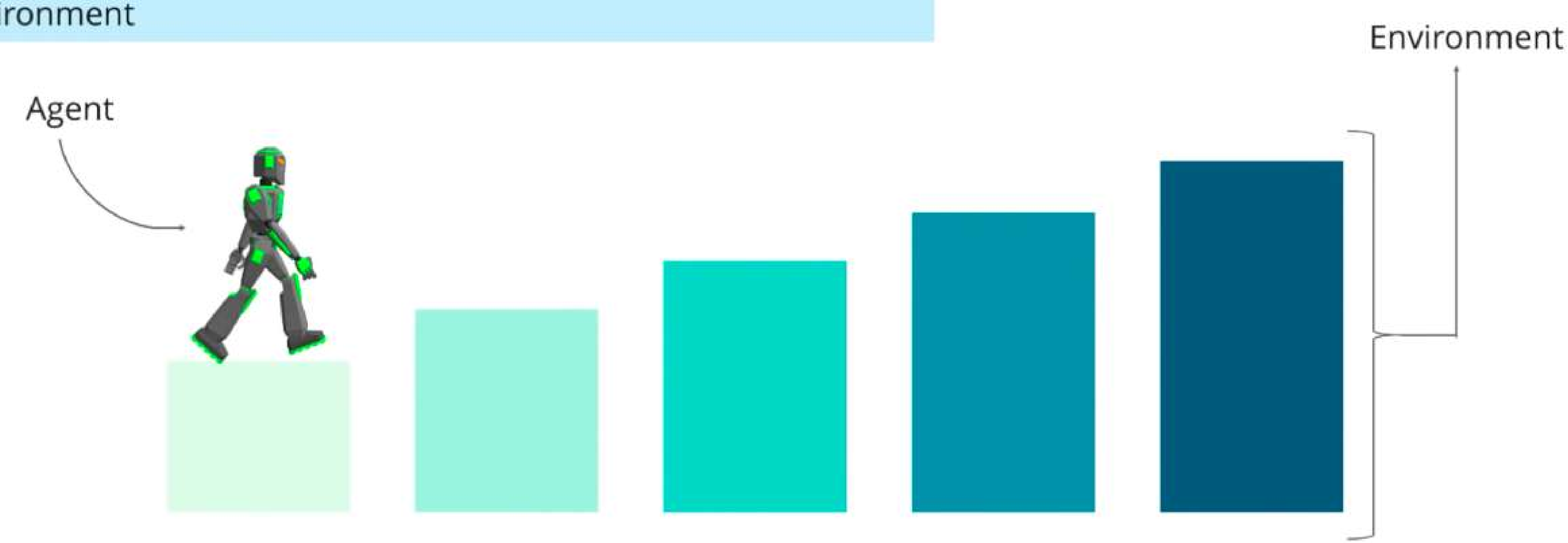
Scenario 2: Baby starts crawling but falls due to some hurdle in between



Reinforcement Learning Process

Reinforcement Learning system is comprised of two main components:

- Agent
- Environment



Reinforcement Learning Process

Reinforcement Learning system is comprised of two main components:

- Agent
- Environment

Reinforcement Learning Process

Reinforcement Learning system is comprised of two main components:

- Agent
- Environment

Reinforcement Learning Process

Reinforcement Learning system is comprised of two main components:

- Agent
- Environment

Reinforcement Learning Process

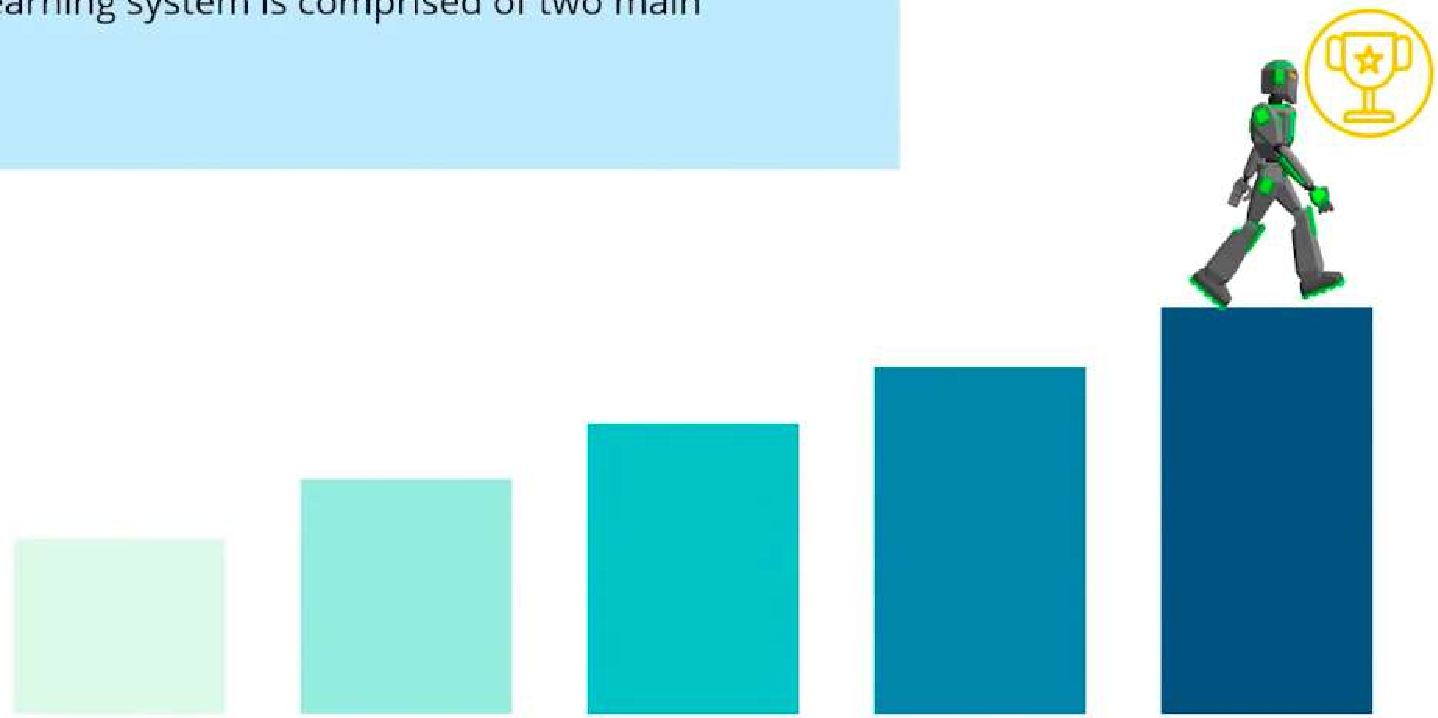
Reinforcement Learning system is comprised of two main components:

- Agent
- Environment

Reinforcement Learning Process

Reinforcement Learning system is comprised of two main components:

- Agent
- Environment



Counter Strike Example

1. The RL Agent (Player1) collects state S^0 from the environment
2. Based on the state S^0, the RL agent takes an action A^0, initially the action is random
3. The environment is now in a new state S^1
4. RL agent now gets a reward R^1 from the environment
5. The RL loop goes on until the RL agent is dead or reaches the destination

Reinforcement Learning Definitions

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

Reinforcement Learning Definitions

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

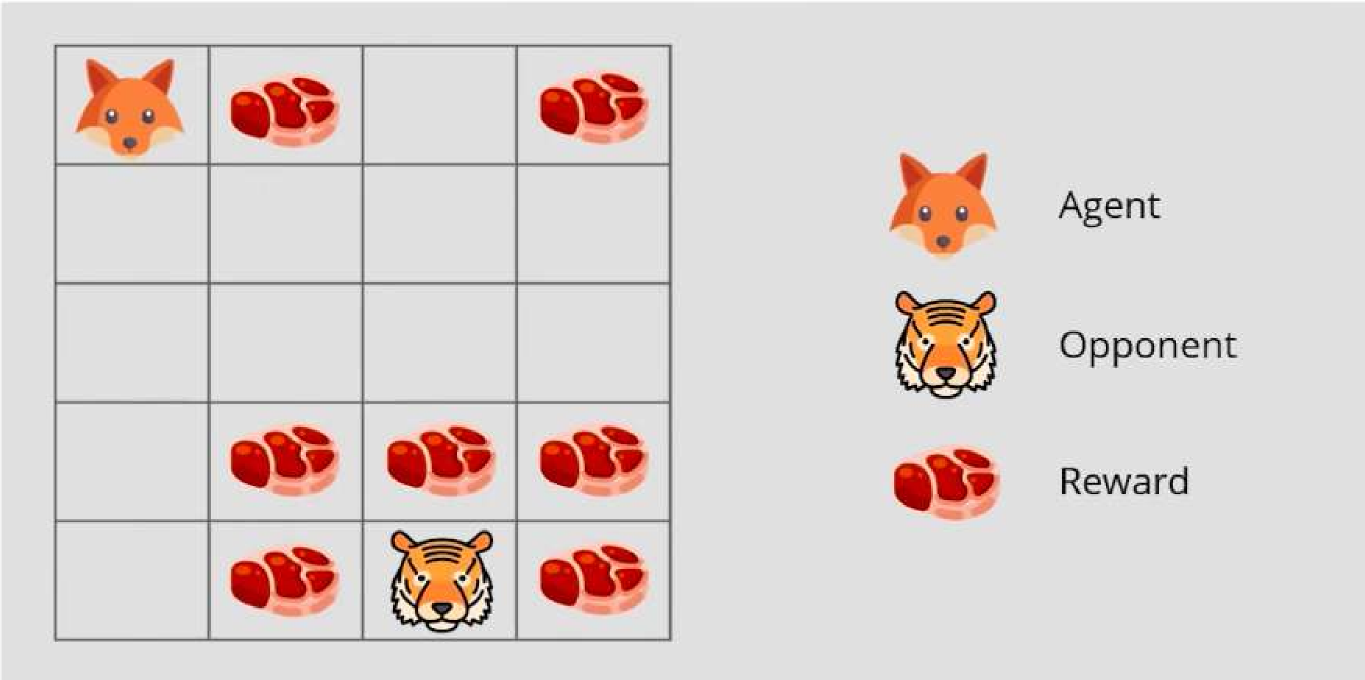
Policy (π): The approach that the agent uses to determine the next action based on the current state

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

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

Reward Maximization

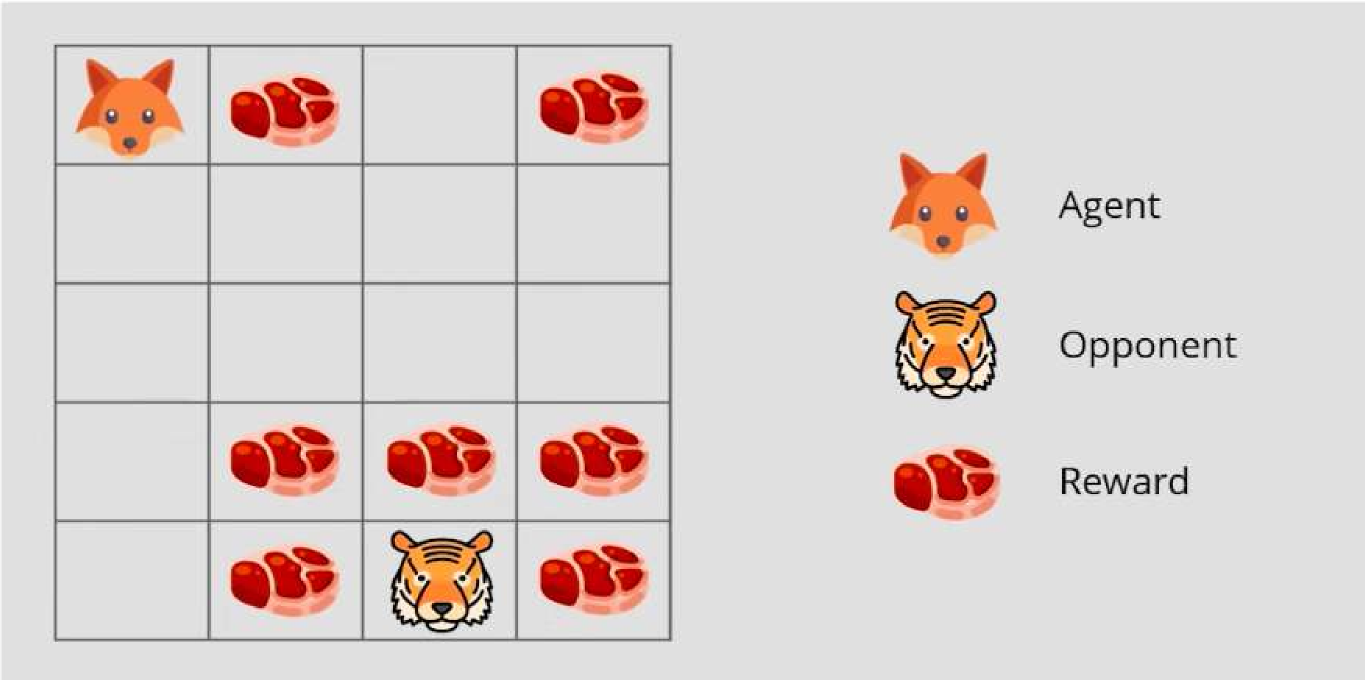
Reward maximization theory states that, *a RL agent must be trained in such a way that, he takes the best action so that the reward is maximum.*



Exploration & Exploitation

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

Exploration is about exploring and capturing more information about an environment

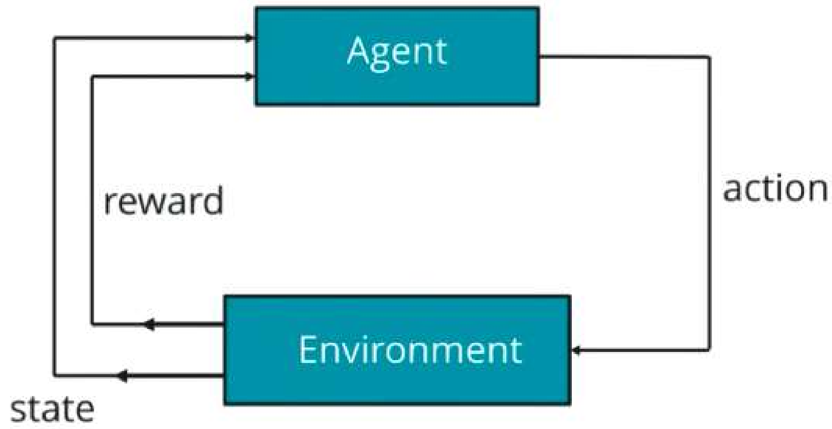


Markov Decision Process

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

The following parameters are used to attain a solution:

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

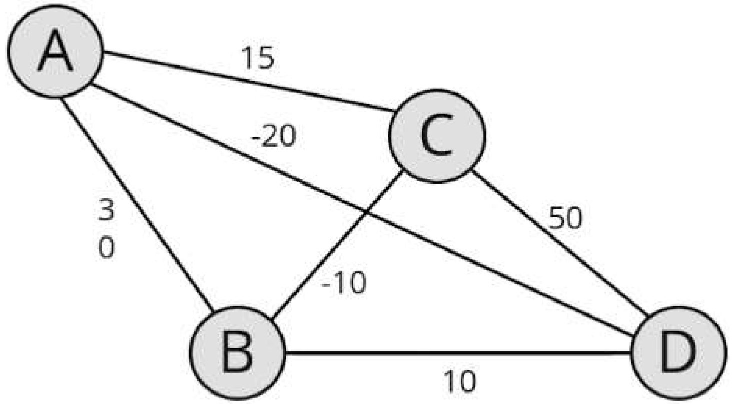


Markov Decision Process – Shortest Path Problem

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

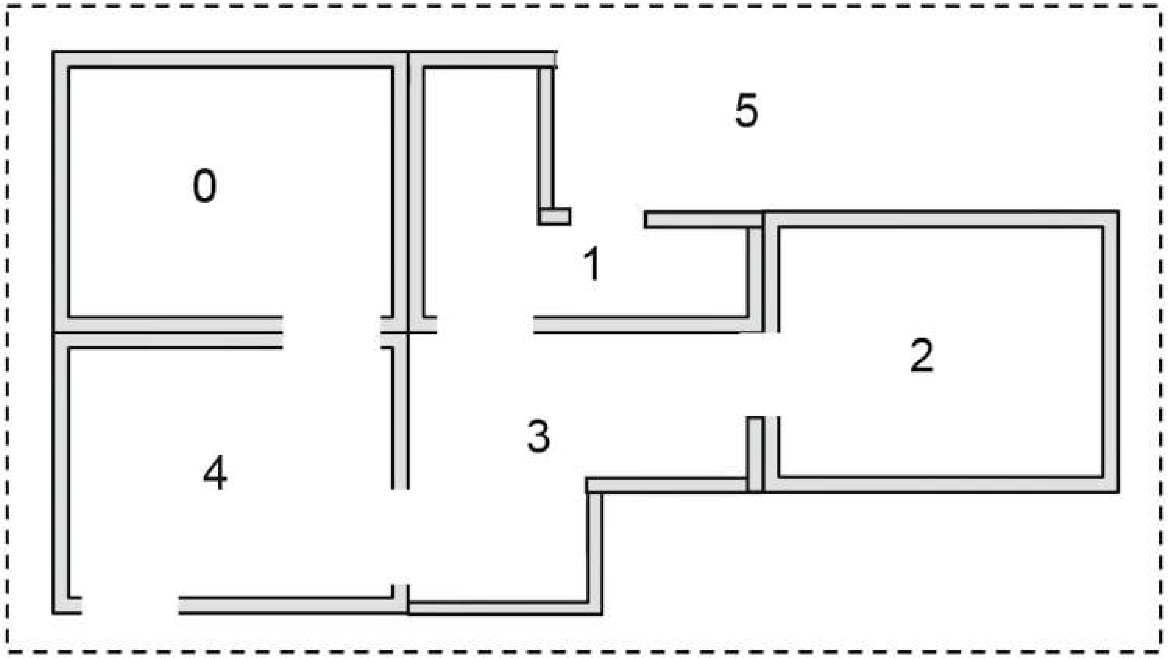
In this problem,

- Set of states are denoted by nodes i.e. {A, B, C, D}
- Action is to traverse from one node 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}



Understanding Q-Learning With An Example

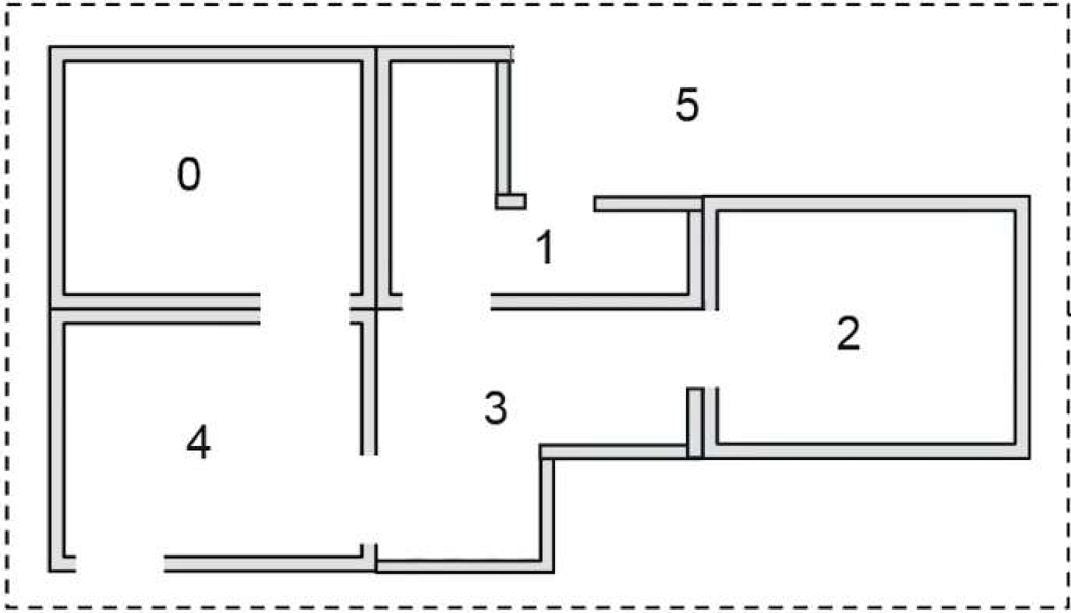
Place an agent in any one of the rooms (0,1,2,3,4) and the goal is to reach outside the building (room 5)

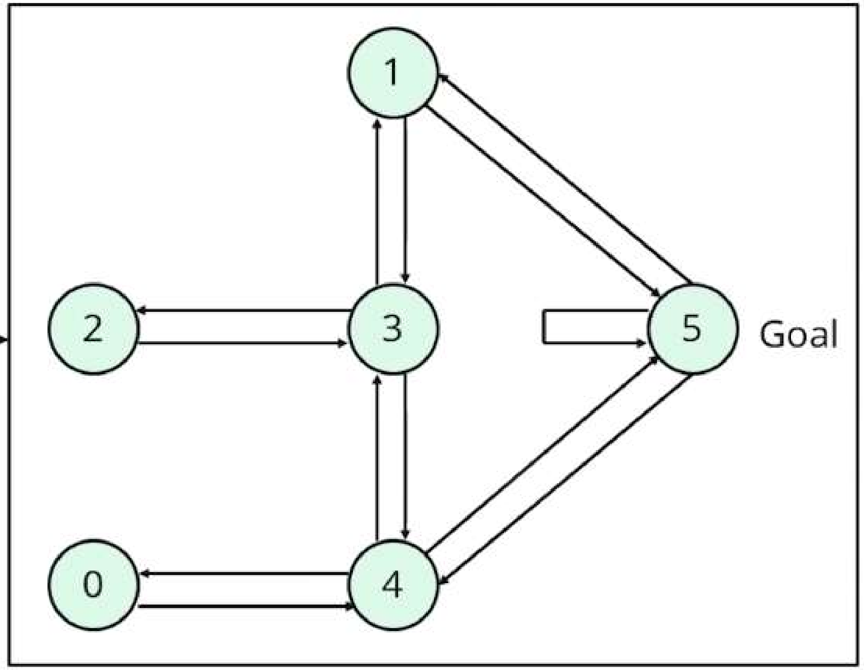


- 5 rooms in a building connected by doors
- each room is numbered 0 through 4
- The outside of the building can be thought of as one big room (5)
- Doors 1 and 4 lead into the building from room 5 (outside)

Understanding Q-Learning With An Example

Let's represent the rooms on a graph, each room as a node, and each door as a link

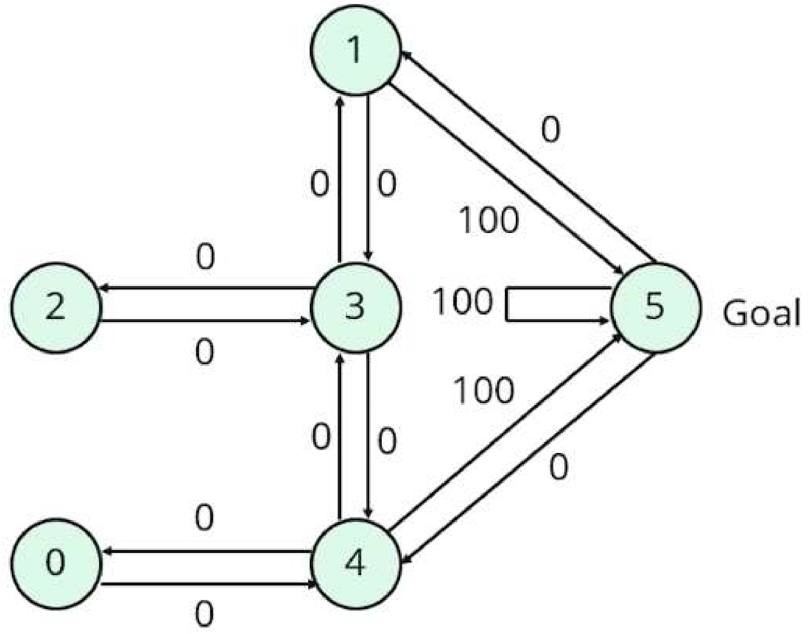




Understanding Q-Learning With An Example

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

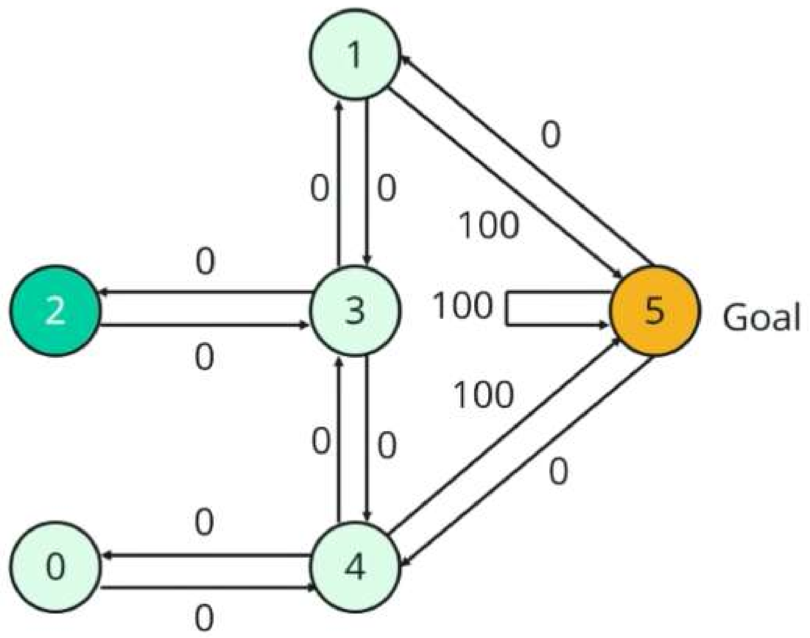
- doors that lead directly to the goal have a reward of 100
- Doors not directly connected to the target room have zero reward
- Because doors are two-way, two arrows are assigned to each room
- Each arrow contains an instant reward value



Understanding Q-Learning With An Example

The terminology in Q-Learning includes the terms state and action:

- Room (including room 5) represents a state
- agent's movement from one room to another represents an action
- In the figure, a state is depicted as a node, while "action" is represented by the arrows

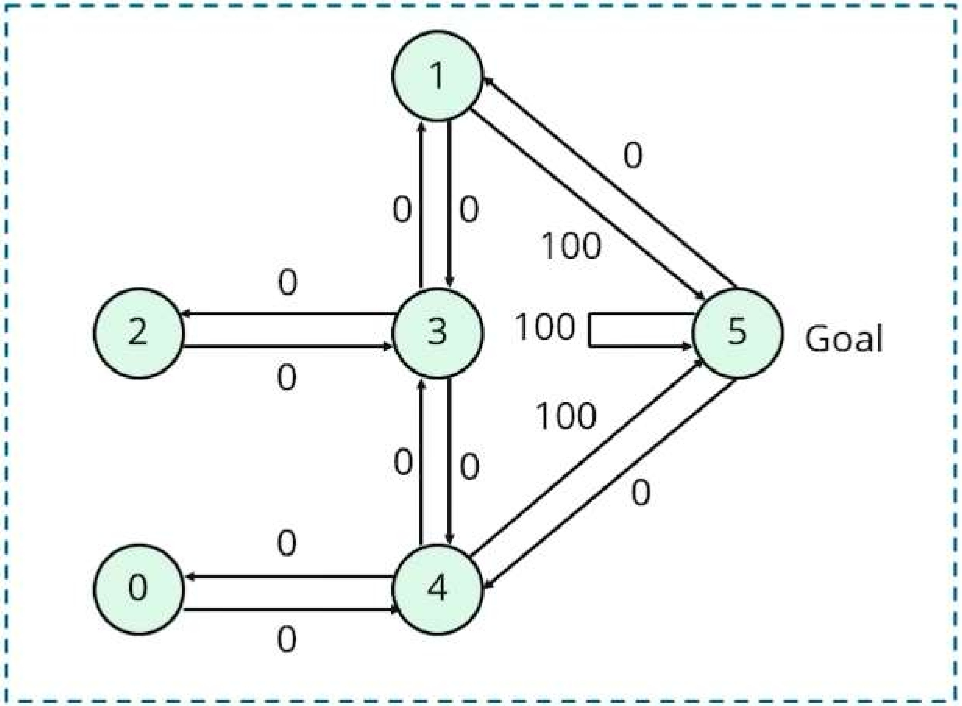


Example (Agent traverse from room 2 to room5):

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

Understanding Q-Learning With An Example

We can put the state diagram and the instant reward values into a reward table, matrix R.



$$
R = \begin{array}{c|cccccc}
 & \multicolumn{6}{c}{\text{Action}} \\
\text{State} & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & & & & & & \\
1 & & & & & & \\
2 & & & & & & \\
3 & & & & & & \\
4 & & & & & & \\
5 & & & & & &
\end{array}
\left[\begin{array}{cccccc}
-1 & -1 & -1 & -1 & 0 & -1 \\
-1 & -1 & -1 & 0 & -1 & 100 \\
-1 & -1 & -1 & 0 & -1 & -1 \\
-1 & 0 & 0 & -1 & 0 & -1 \\
0 & -1 & -1 & 0 & -1 & 100 \\
-1 & 0 & -1 & -1 & 0 & 100
\end{array}\right]
$$

The -1's in the table represent null values

Understanding Q-Learning With An Example

Add another matrix Q, representing 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
- Formula to calculate the Q matrix:

*Q(state, action) = R(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.
- If Gamma is closer to one, the agent will consider future rewards with greater weight

Q – Learning Algorithm

1. Set the gamma parameter, and environment rewards in matrix R
2. Initialize matrix Q to zero
3. Select a random initial 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. Repeat above steps until current state = goal state

Q – Learning Example

First step is to set the value of the learning parameter Gamma = 0.8, and the initial state as Room 1.

Next, initialize matrix Q as a zero matrix:

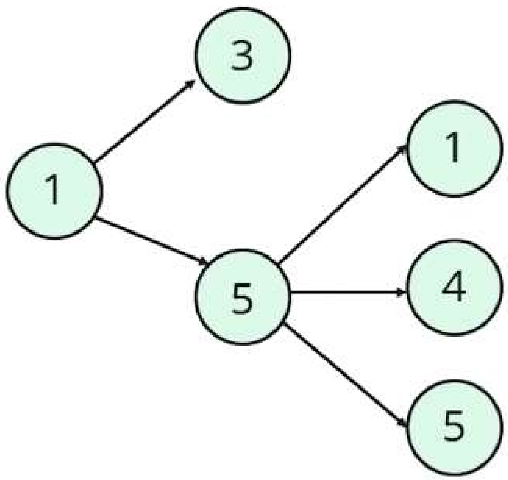
- From room 1 you can either go to room 3 or 5, let's select room 5.
- From room 5, calculate maximum Q value for this next state based on all possible actions:

*Q(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

$$Q = \begin{array}{c|c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$R = \begin{array}{c|c} \text{State} & \begin{array}{cccccc} & & & \text{Action} & & \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$



Q – Learning Example

For the next episode, 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:

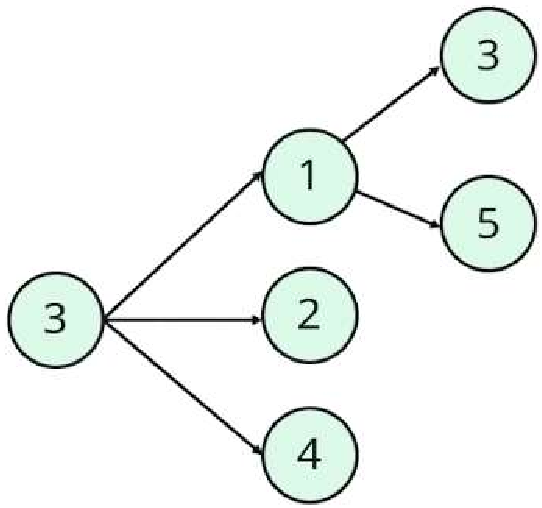
*Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]*

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

$$Q = \begin{array}{cc} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$R = \begin{array}{cc} & \text{Action} \\ \text{State} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$



Q – Learning Example

For the next episode, the next state, 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)]*

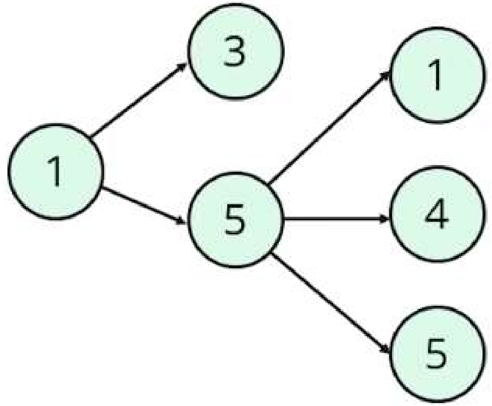
Q(1,5) = R(1,5) + 0.8 * Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8 * 0 = 100

The matrix Q remains the same since, Q(1,5) is already fed to the agent

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$\begin{array}{c} \\ \text{State} \end{array} \quad \begin{array}{c} \text{Action} \end{array}$$

$$R = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$



```python
import numpy as np

# R matrix
R = np.matrix([[-1, -1, -1, -1, 0, -1],
               [-1, -1, -1, 0, -1, 100],
               [-1, -1, -1, 0, -1, -1],
               [-1, 0, 0, -1, 0, -1],
               [-1, 0, 0, -1, -1, 100],
               [-1, 0, -1, -1, 0, 100]])

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

# Gamma (learning parameter).
gamma = 0.8

# Initial state. (Usually to be chosen at random)
initial_state = 1
```

Test [C:\Users\zulaikha\PycharmProjects\Test] - ...\demo.py [Test] - PyCharm

File Edit View Navigate Code Refactor Run Tools VCS Window Help

Test 〉 demo.py

demo.py

```python

# Initial state. (Usually to be chosen at random)
initial_state = 1


# This function returns all available actions in the state given as an argument
def available_actions(state):
    current_state_row = R[state,]
    av_act = np.where(current_state_row >= 0)[1]
    return av_act


# Get available actions in the current state
available_act = available_actions(initial_state)


# This function chooses at random which action to be performed within the range
# of all the available actions.
def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_act, 1))
```

Run: demo

4: Run 6: TODO Terminal Python Console Event Log

1 char 18:18 CRLF UTF-8 4 spaces

Test [C:\Users\zulaikha\PycharmProjects\Test] - ...\demo.py [Test] - PyCharm

File Edit View Navigate Code Refactor Run Tools VCS Window Help

Test 〉 demo.py

demo.py

```python
# Get available actions in the current state
available_act = available_actions(initial_state)


# This function chooses at random which action to be performed within the range
# of all the available actions.
def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_act, 1))
    return next_action


# Sample next action to be performed
action = sample_next_action(available_act)


# This function updates the Q matrix according to 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]
```

sample_next_action()

Run: demo

4: Run 6: TODO Terminal Python Console Event Log

35:49 CRLF UTF-8 4 spaces

SUBSCRIBE

Test [C:\Users\zulaikha\PycharmProjects\Test] - ...\demo.py [Test] - PyCharm

File Edit View Navigate Code Refactor Run Tools VCS Window Help

Test 〉 demo.py

demo.py

```python


# Sample next action to be performed
action = sample_next_action(available_act)


# This function updates the Q matrix according to 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
    Q[current_state, action] = R[current_state, action] + gamma * max_value
```

update()

Run: demo

4: Run 6: TODO Terminal Python Console Event Log

55:11 CRLF UTF-8 4 spaces

Test [C:\Users\zulaikha\PycharmProjects\Test] - ...\demo.py [Test] - PyCharm

File Edit View Navigate Code Refactor Run Tools VCS Window Help

Test demo.py

demo.py

```python
            max_index = int(max_index)
        max_value = Q[action, max_index]

        # Q learning formula
        Q[current_state, action] = R[current_state, action] + gamma * max_value


# Update Q matrix
update(initial_state, action, gamma)

# -------------------------------------------------------------------------------
# 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 = sample_next_action(available_act)
    update(current_state, action, gamma)

```

for i in range(10000)

Run: demo

4: Run 6: TODO Terminal Python Console Event Log

65:23 CRLF UTF-8 4 spaces

SUBS CRIBE

Test [C:\Users\zulaikha\PycharmProjects\Test] - ...\demo.py [Test] - PyCharm

File Edit View Navigate Code Refactor Run Tools VCS Window Help

Test / demo.py

demo.py

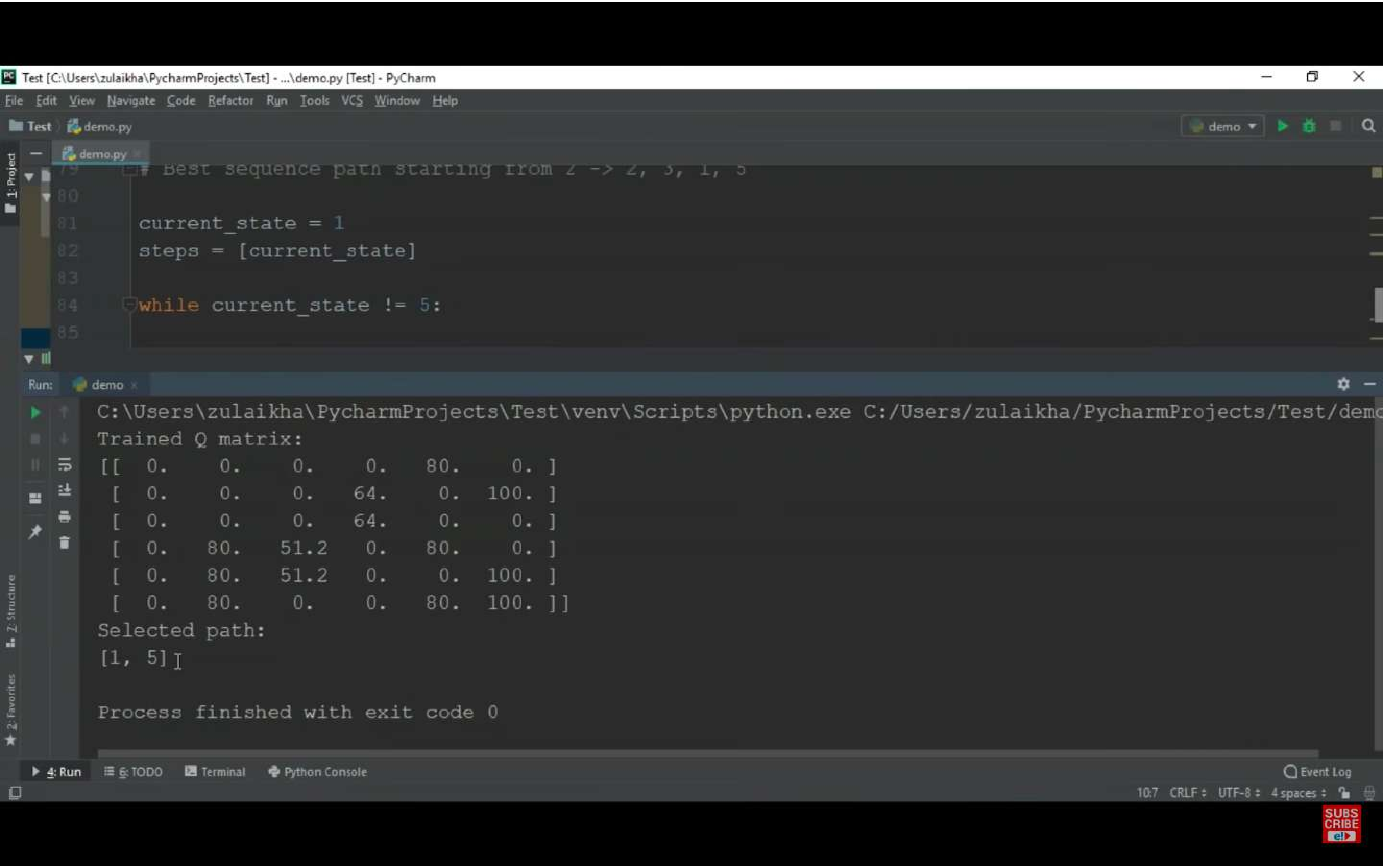
```python
    update(current_state, action, gamma)

# Normalize the "trained" Q matrix
print("Trained Q matrix:")
print(Q / np.max(Q) * 100)

# -------------------------------------------------------------------------------
# Testing

# Goal state = 5
# Best sequence path starting from 2 -> 2, 3, 1, 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:
```

Run: demo

4: Run   6: TODO   Terminal   Python Console   Event Log

81:18 CRLF UTF-8 4 spaces

```python
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 selected sequence of steps
print("Selected path:")
print(steps)
```


Test [C:\Users\zulaikha\PycharmProjects\Test] - ...\demo.py [Test] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Test demo.py
demo
demo.py
# Best sequence path starting from 2 -> 2, 3, 1, 5
current_state = 1
steps = [current_state]
while current_state != 5:
Run: demo
C:\Users\zulaikha\PycharmProjects\Test\venv\Scripts\python.exe C:/Users/zulaikha/PycharmProjects/Test/demo
Trained 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.]
[0. 80. 51.2 0. 0. 100.]
[0. 80. 0. 0. 80. 100.]]
Selected path:
[1, 5]
Process finished with exit code 0
4: Run
6: TODO
Terminal
Python Console
Event Log
10:7 CRLF UTF-8 4 spaces
SUBSCRIBE

Test [C:\Users\zulaikha\PycharmProjects\Test] - ...\demo.py [Test] - PyCharm

File Edit View Navigate Code Refactor Run Tools VCS Window Help

Test 〉 demo.py

demo.py

```python
# Best sequence path starting from 2 -> 2, 3, 1, 5

current_state = 2
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 selected sequence of steps
```

while current_state != 5

Run: demo

```
[  0.    80.    51.2   0.    80.     0. ]
[  0.    80.    51.2   0.     0.   100. ]
[  0.    80.     0.    0.    80.   100. ]]
```

4: Run 6: TODO Terminal Python Console Event Log

86:34 CRLF UTF-8 4 spaces

SUBS CRIBE

Test [C:\Users\zulaikha\PycharmProjects\Test] - ...\demo.py [Test] - PyCharm

File Edit View Navigate Code Refactor Run Tools VCS Window Help

Test 〉 demo.py

demo.py

```python
# Best sequence path starting from 2 -> 2, 3, 1, 5

current_state = 2
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))
```

while current_state != 5

Run: demo

```
Trained 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. ]
 [  0.   80.   51.2   0.    0.  100. ]
 [  0.   80.    0.    0.   80.  100. ]]
Selected path:
[2, 3, 4, 5]
```

4: Run 6: TODO Terminal Python Console Event Log

10:13 CRLF ÷ UTF-8 ÷ 4 spaces ÷

SUBS CRIBE