Implementing a Simple Q-Learning Model

Q-Learning is a **model-free reinforcement learning algorithm** used to learn the value of actions in a given environment. The agent updates a **Q-table** to estimate the expected cumulative reward for each state-action pair.

Key Components of Q-Learning

1. **Q-Table:** Stores values of state-action pairs.
2. **Learning Rate (α):** Determines how much new information overrides old.
3. **Discount Factor (γ):** Balances immediate and future rewards.
4. **Exploration Rate (ε):** Probability of taking random actions to explore the environment.

Algorithm Steps

1. Initialize Q-table with zeros.
2. For each episode:
    - Observe initial state.
    - Select an action (ε-greedy: explore or exploit).
    - Execute action, observe reward and next state.
    - Update Q-value using the formula:

$Q(s,a)=Q(s,a)+\alpha[r+\gamma\max_{a'}Q(s',a')-Q(s,a)]$Q(s,a) = Q(s,a) + \alpha [r + \gamma \max_{a’} Q(s’,a’) – Q(s,a)]

    - Repeat until reaching terminal state.

Python Example Using OpenAI Gym (FrozenLake Environment):

```
import gym

import numpy as np


# Initialize environment

env = gym.make('FrozenLake-v1', is_slippery=False)
```

```
n_states = env.observation_space.n
n_actions = env.action_space.n
q_table = np.zeros((n_states, n_actions))

# Hyperparameters
alpha = 0.8      # Learning rate
gamma = 0.95     # Discount factor
epsilon = 0.1    # Exploration rate
episodes = 1000

# Q-Learning algorithm
for episode in range(episodes):
   state = env.reset()
   done = False
   while not done:
      # Choose action (ε-greedy)
      if np.random.rand() < epsilon:
         action = env.action_space.sample()  # Explore
      else:
         action = np.argmax(q_table[state])  # Exploit

      # Take action
      next_state, reward, done, _ = env.step(action)

      # Update Q-value
      q_table[state, action] = q_table[state, action] + alpha * (
         reward + gamma * np.max(q_table[next_state]) - q_table[state, action]
      )
```

```
        state = next_state

# Evaluate Q-table
print("Trained Q-Table:\n", q_table)

# Test the trained agent
state = env.reset()
done = False
while not done:
    action = np.argmax(q_table[state])
    state, reward, done, _ = env.step(action)
    env.render()
print("Reward:", reward)
```

Notes

- Q-Learning works best in **discrete environments** with a finite number of states and actions.
- For large or continuous state spaces, **Deep Q-Networks (DQN)** are used instead of Q-tables.
- Hyperparameters like learning rate, discount factor, and exploration rate significantly affect learning efficiency.

This simple Q-learning implementation provides a **foundation for reinforcement learning**, demonstrating how an agent can learn optimal actions by trial and error.