AML Lab Assignment-7:

Name: Gurvinder Kaur Matharu
PRN: 1032230432
Roll No.: PA14

AIM:

Implementation of Reinforcement Learning

THEORY:

Reinforcement Learning (RL) is a machine learning paradigm where an agent learns to make sequential decisions through interactions with an environment to maximize cumulative rewards. It involves the agent observing the environment, taking actions, receiving feedback (rewards or penalties), and adjusting its strategy to learn optimal decision-making policies.

The core idea in RL is to strike a balance between exploration (trying out different actions to learn more about the environment) and exploitation (leveraging learned knowledge to make optimal decisions). It employs the concept of Markov Decision Processes (MDPs) to model decision-making tasks involving states, actions, rewards, and transition probabilities.

The agent aims to learn a policy—a mapping of states to actions—that maximizes the long-term cumulative reward. To achieve this, RL algorithms use different strategies like value iteration, policy iteration, Q-learning, Deep Q Networks (DQN), and more recently, algorithms like Proximal Policy Optimization (PPO) and Deep Deterministic Policy Gradients (DDPG) that handle complex tasks and continuous action spaces.

RL finds applications in various domains like robotics, gaming, recommendation systems, finance, and healthcare. It's used for training agents to play games, control autonomous systems, optimize resource allocation, and solve complex decision-making problems where explicit instructions or labeled data are unavailable, making it a powerful approach for learning in dynamic, uncertain environments.

CODE EXECUTION & OUTPUT:

```
import numpy as np
```

```
# Define the environment as a 3x3 grid
environment = np.array([
    [0, 0, 0],
    [0, 1, 0],
    [0, 0, 2]
])
```

```
# Q-table initialization
state_space_size = 9  # 3x3 grid
action_space_size = 4  # up, down, left, right
q_table = np.zeros((state_space_size, action_space_size))
```

```
# Mapping from actions to (row, column) changes
action_mapping = {
    0: (-1, 0),  # Up
    1: (1, 0),   # Down
    2: (0, -1),  # Left
    3: (0, 1)    # Right
}
```

```
# Hyperparameters
learning_rate = 0.8
discount_factor = 0.95
num_episodes = 5000

# Exploration parameters
exploration_prob = 1.0
min_exploration_prob = 0.01
exploration_decay = 0.995
```

```
# Q-learning algorithm
for episode in range(num_episodes):
    state = 0  # Starting state

    while True:
        # Exploration-exploitation trade-off
        if np.random.rand() < exploration_prob:
            action = np.random.randint(action_space_size)  # Explore
        else:
            action = np.argmax(q_table[state, :])  # Exploit

        # Take the selected action
        row_change, col_change = action_mapping[action]
        new_row, new_col = np.clip([state // 3 + row_change, state % 3 + col_change], 0, 2)
        new_state = new_row * 3 + new_col

        # Reward function (1 for reaching the goal, 0 otherwise)
        reward = 1 if environment[new_row, new_col] == 2 else 0

        # Update the Q-table using the Q-learning formula
        q_table[state, action] = (1 - learning_rate) * q_table[state, action] + \
                                 learning_rate * (reward + discount_factor * np.max(q_table[new_state, :]))

        state = new_state

        # Check if the episode is done (agent reached the goal)
        if environment[new_row, new_col] == 2:
            break

    # Decay exploration probability
    exploration_prob = max(min_exploration_prob, exploration_prob * exploration_decay)
```

```
print("Learned Q-table:")
print(q_table)
```

```
 Learned Q-table:
 [[0.81450625 0.857375   0.81450625 0.857375  ]
  [0.857375   0.9025     0.81450625 0.9025    ]
  [0.9025     0.95       0.857375   0.9025    ]
  [0.81450625 0.9025     0.857375   0.9025    ]
  [0.857375   0.95       0.857375   0.95      ]
  [0.9025     1.         0.9025     0.95      ]
  [0.857375   0.9025     0.9025     0.95      ]
  [0.9025     0.95       0.9025     1.        ]
  [0.         0.         0.         0.        ]]
```