NAAN MUDHALVAN

PHASE-3

ARTIFICIAL INTELLIGENCE & DATA SCIENCE
(2nd YEAR)

AUTONOMOUS VEHICLES AND ROBOTICS

AI-BASED ROUTE MEMORY AND SELF LEARNING

TEAM LEADER
Sri Ram R

TEAM MEMBERS
Nithin Deepak R
Santhosh S
Shanyu Starness P
Udhaya Kumar A

Phase 3: Implementation of Project

Title: AI-Based Route Memory and Self-Learning Navigation System

Objective

The goal of Phase 3 is to implement the core components of the AI-Based Route Memory and Self-Learning Navigation System based on the designs and strategies developed in Phase 2. This includes developing the reinforcement learning (RL) model, building the environment simulator, setting up a user interface, and incorporating basic data visualization tools to track AI learning.

1. AI Model Development

Overview

The primary function of this system is to allow a virtual agent to learn optimal routes in a grid-like environment using reinforcement learning. The model will "remember" paths and improve over time.

Implementation

- Algorithm: Q-Learning (tabular method).
- Environment: Grid-based world with static start and goal positions, plus optional obstacles.
- Reward Strategy:
    - Positive reward for reaching the goal.
    - Negative for hitting walls or obstacles.
    - Small negative per step to encourage efficiency.

Code – q_learning_agent.py

```python
import numpy as np
import random

class QLearningAgent:
    def __init__(self, grid_size=(5, 5), alpha=0.1, gamma=0.9, epsilon=0.2):
        self.grid_size = grid_size
        self.q_table = np.zeros(grid_size + (4,))  # 4 actions: up, down, left, right
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon

    def choose_action(self, state):
        if random.uniform(0, 1) < self.epsilon:
            return random.randint(0, 3)  # Explore
        return np.argmax(self.q_table[state])  # Exploit

    def update_q(self, state, action, reward, next_state):
        max_next = np.max(self.q_table[next_state])
        current = self.q_table[state][action]
        self.q_table[state][action] += self.alpha * (reward + self.gamma * max_next - current)

```

2. Environment Simulation

Overview

An environment where the agent learns movement, obstacles, and goals.

Implementation

- Grid Size: Configurable.
- Elements: Start, Goal, Obstacles.
- Display: Console-based with optional matplotlib visualization.

Code – grid_environment.py

```python
import numpy as np

class GridEnvironment:
    def __init__(self, size=(5, 5), start=(0, 0), goal=(4, 4), obstacles=[]):
        self.size = size
        self.start = start
        self.goal = goal
        self.obstacles = obstacles
        self.state = start

    def reset(self):
        self.state = self.start
        return self.state

    def step(self, action):
        x, y = self.state
        if action == 0: x -= 1  # Up
        elif action == 1: x += 1  # Down
        elif action == 2: y -= 1  # Left
        elif action == 3: y += 1  # Right

        # Boundary conditions
        x = max(0, min(x, self.size[0] - 1))
        y = max(0, min(y, self.size[1] - 1))
        next_state = (x, y)

        if next_state in self.obstacles:
            return self.state, -10, False
        elif next_state == self.goal:
            return next_state, 10, True
        else:
            self.state = next_state
            return next_state, -1, False
```

3. UI/Visualization (Optional for Phase 3)

Overview

A simple graphical representation of learning progress using matplotlib.

Code – visualize_learning.py

```python
import matplotlib.pyplot as plt

def plot_rewards(rewards):
    plt.plot(rewards)
    plt.xlabel("Episode")
    plt.ylabel("Total Reward")
    plt.title("Learning Progress")
    plt.grid(True)
    plt.show()

```

4. Data Security Implementation (Minimal)

Overview

While not handling personal data yet, we simulate securing learning data logs.

Implementation

- Save Q-table securely using serialization (Pickle).
- Optional: Encrypt using simple Fernet (symmetric encryption).

Code – save_q_table.py

```python
import pickle

def save_q_table(agent, filename='q_table.pkl'):
    with open(filename, 'wb') as f:
        pickle.dump(agent.q_table, f)

def load_q_table(agent, filename='q_table.pkl'):
    with open(filename, 'rb') as f:
        agent.q_table = pickle.load(f)

```

5. Testing and Feedback Collection

Overview

Initial test runs to observe route learning and performance over episodes.

Implementation

- Run for 100 episodes and observe convergence.
- Metrics: Total reward, steps to goal, success rate.

Code – train_and_test.py

```python
from grid_environment import GridEnvironment
from q_learning_agent import QLearningAgent
from visualize_learning import plot_rewards

env = GridEnvironment(size=(5, 5), start=(0, 0), goal=(4, 4), obstacles=[(1,1), (2,2)])
agent = QLearningAgent(grid_size=(5, 5))

rewards = []

for episode in range(100):
    state = env.reset()
    total_reward = 0
    done = False

    while not done:
        action = agent.choose_action(state)
        next_state, reward, done = env.step(action)
        agent.update_q(state, action, reward, next_state)
        state = next_state
        total_reward += reward

    rewards.append(total_reward)

plot_rewards(rewards)
```

Challenges and Solutions

CHALLENGES	SOLUTION
Model Convergence	Adjust learning rate, gamma, and epsilon over time.
Random Behaviour Early On	ε-greedy policy to balance explore/exploit.
Obstacle Handling	Penalize obstacle collisions to avoid dead zones.
Visual Debugging	Add heatmap or grid plots (future phases).

Outcomes of Phase 3

- By the end of Phase 3, you should have:
- Basic Q-Learning Agent trained in a grid.

- Simulation Environment with route learning.
- An optional UI or CLI interface to watch learning behaviour.
- Securely stored learning state (Q-table).
- Initial testing reports and reward visualizations.

Next Steps – Phase 4

- Add dynamic obstacles and path re-routing logic.
- Introduce multimodal environments (e.g., real maps).
- Extend to robot/IoT integration with real-world path testing.
- Use deep RL (DQN) to replace the Q-table for larger spaces.