PRACTICAL 2

Topic: Grid Game nXn

Solution: I have created a grid game solver ai agent based on BFS with a feature of creating the structure of the grid and placing the obstacles ourselves

Code:

from collections import deque

class GridSolver:

    def \_\_init\_\_(self, n, obstacles=None):

        """Initialize the grid, start, goal, and obstacles."""

        self.n = n

        self.start = (0, 0)

        self.goal = (n - 1, n - 1)

        self.grid = [[0 for \_ in range(n)] for \_ in range(n)]

        self.obstacles = obstacles or []

        self.place\_obstacles()

    def place\_obstacles(self):

        """Place obstacles on the grid."""

        for x, y in self.obstacles:

            if (x, y) not in [self.start, self.goal]:

                self.grid[x][y] = 1  # Mark as obstacle

    def display\_grid(self, path=None):

        """Display the grid with start, goal, obstacles, and optional path."""

        for i in range(self.n):

            for j in range(self.n):

                if (i, j) == self.start:

                    print("S", end=" ")  # Start

                elif (i, j) == self.goal:

                    print("G", end=" ")  # Goal

                elif path and (i, j) in path:

                    print("o", end=" ")  # Path

                elif self.grid[i][j] == 1:

                    print("X", end=" ")  # Obstacle

                else:

                    print(".", end=" ")  # Empty cell

            print()

        print("-" \* (self.n \* 2))

    def bfs(self):

        """Find the shortest path using BFS."""

        # Directions for moving up, down, left, and right

        directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

        queue = deque([(self.start, [self.start])])  # Queue holds (current position, path to position)

        visited = set()  # Track visited nodes

        while queue:

            current, path = queue.popleft()

            if current == self.goal:

                return path  # Return the path when goal is reached

            if current in visited:

                continue

            visited.add(current)

            # Explore neighbors

            for dx, dy in directions:

                x, y = current[0] + dx, current[1] + dy

                if 0 <= x < self.n and 0 <= y < self.n and self.grid[x][y] == 0 and (x, y) not in visited:

                    queue.append(((x, y), path + [(x, y)]))

        return None  # Return None if no path is found

    def solve(self):

        """Solve the grid and display the solution."""

        print("Initial Grid:")

        self.display\_grid()

        path = self.bfs()

        if path:

            print(f"Shortest Path Found (length {len(path)}): {path}")

            print("Solution Grid:")

            self.display\_grid(path)

        else:

            print("No path found!")

# Example Usage

grid\_size = 5

obstacles = [(1, 1), (2, 2), (3, 3)]  # Example obstacles

solver = GridSolver(grid\_size, obstacles)

solver.solve()

Output:

