FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE

EXPERIMENT 2

MAAZ MALIK

TE AIML B

2022 6000 31

AIM

To formulate and implement an Intelligent Agent that autonomously navigates an environment by perceiving stimuli through sensors, interpreting percepts, and executing optimal actions via actuators, guided by a defined set of rules.

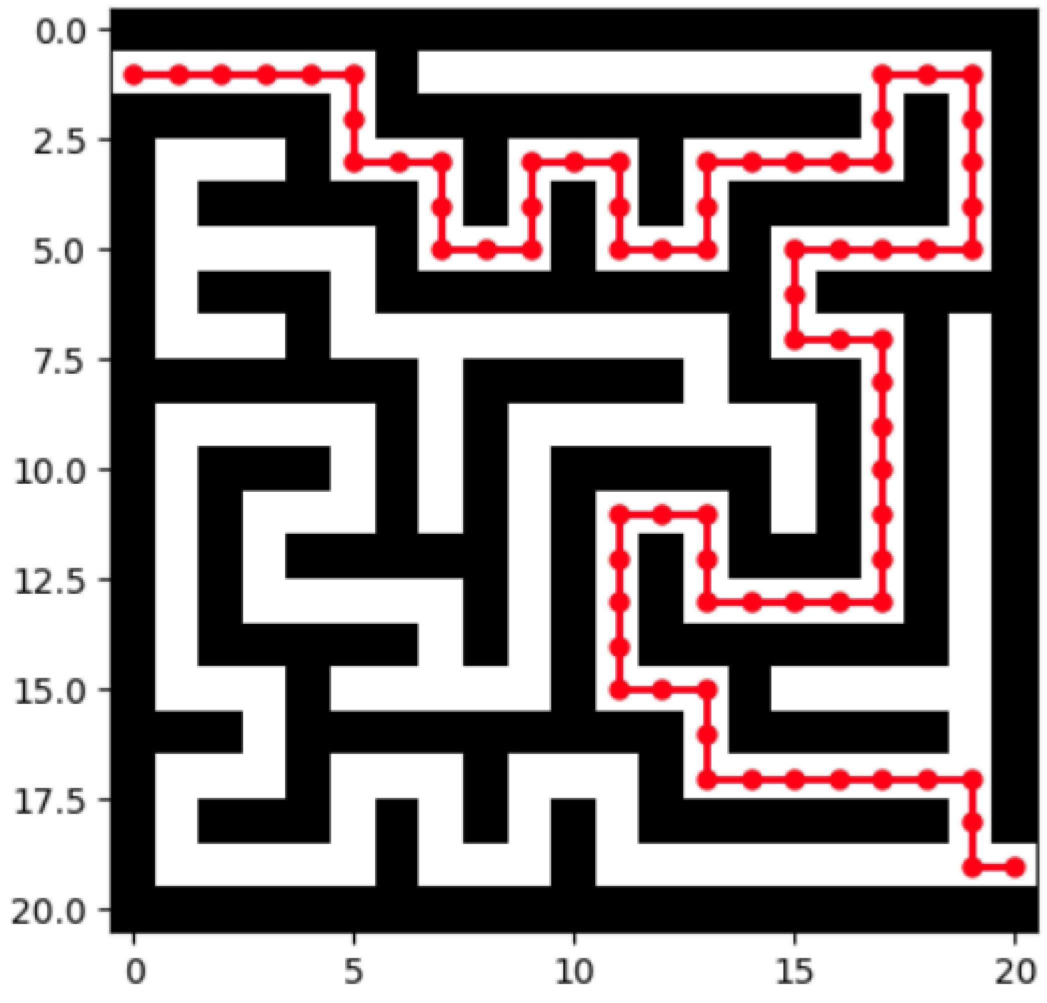
THEORY

An Intelligent Agent is a system designed to interact with its environment by sensing stimuli through sensors, processing the information into percepts, and making decisions based on a set of rules. The agent then performs actions using actuators to achieve specific goals. The effectiveness of an agent depends on its ability to interpret percepts accurately and execute actions that optimize its performance within the environment. This process models autonomous behavior in complex and dynamic settings, allowing the agent to adapt and respond to varying conditions.

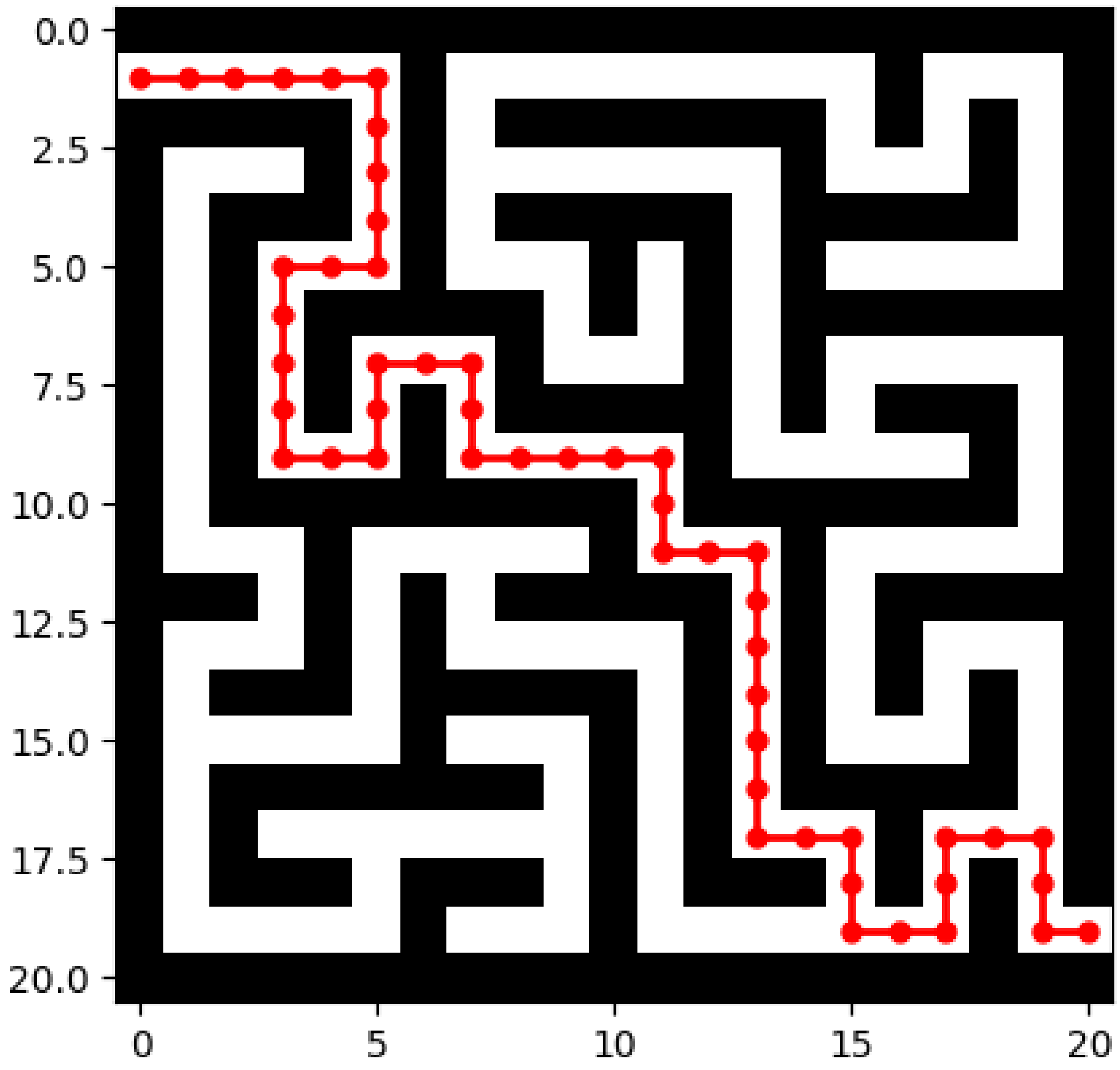
This code implements a maze-solving algorithm using recursion to find and visualize a valid path in a generated maze. The `` `Environment` `` class creates a maze of specified dimensions using a randomized depth-first search algorithm. It initializes entry and exit points and provides methods to check cell accessibility and display the maze. The `` `Agent` `` class, utilizing recursive backtracking, navigates from the entry point to the exit by exploring adjacent cells and maintaining the current path. Upon discovering a valid path, it visualizes the maze with the path highlighted. The approach ensures that only the first valid path is displayed, if found.

CODE

```python
import numpy as np
import matplotlib.pyplot as plt
import random

class Environment:
    def __init__(self, width, height):
        self.width = width
        self.height = height
        self.maze = self.generate_maze()
        self.entry_point = (1, 0)
        self.exit_point = (self.height - 2, self.width - 1)

    def generate_maze(self):
        maze = np.ones((self.height, self.width), dtype=int)
        stack = [(1, 1)]
        maze[1, 1] = 0

        while stack:
            y, x = stack[-1]
            directions = [(0, 2), (0, -2), (2, 0), (-2, 0)]
            random.shuffle(directions)
            moved = False

            for dy, dx in directions:
                ny, nx = y + dy, x + dx
                if 1 <= ny < self.height - 1 and 1 <= nx < self.width - 1
and maze[ny, nx] == 1:
                    maze[ny, nx] = 0
                    maze[y + dy // 2, x + dx // 2] = 0
                    stack.append((ny, nx))
                    moved = True
                    break

            if not moved:
                stack.pop()

        maze[1, 0] = 0
        maze[self.height - 2, self.width - 1] = 0
        return maze

    def get_maze(self):
        return self.maze

    def is_free(self, y, x):
        return self.maze[y, x] == 0

    def display(self, path=None):
        plt.imshow(self.maze, cmap='gray_r')
        if path:
            path_y, path_x = zip(*path)
            plt.plot(path_x, path_y, marker='o', color='red',
markersize=5, linestyle='-', linewidth=2)
        plt.show()

class Agent:
    def __init__(self, env):
        self.env = env
        self.entry_point = env.entry_point
        self.exit_point = env.exit_point

    def solve(self, y, x, path):
        if (y, x) == self.exit_point:
            path.append((y, x))
            self.env.display(path)
            return True

        if not (0 <= y < self.env.height and 0 <= x < self.env.width) or
not self.env.is_free(y, x):
            return False

        if (y, x) in path:
            return False

        path.append((y, x))

        for dy, dx in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            ny, nx = y + dy, x + dx
            if self.solve(ny, nx, path):
                return True

        path.pop()
        return False

env = Environment(21, 21)
agent = Agent(env)
path = []
if not agent.solve(*env.entry_point, path):
    print("No valid path found.")

print("Path taken by the agent:", path)
```



```
Path taken by the agent: [(1, 0), (1, 1), (1, 2), (1, 3), (1, 4),
(1, 5), (2, 5), (3, 5), (3, 6), (3, 7), (4, 7), (5, 7), (5, 8), (5,
9), (4, 9), (3, 9), (3, 10), (3, 11), (4, 11), (5, 11), (5, 12),
(5, 13), (4, 13), (3, 13), (3, 14), (3, 15), (3, 16), (3, 17), (2,
17), (1, 17), (1, 18), (1, 19), (2, 19), (3, 19), (4, 19), (5, 19),
(5, 18), (5, 17), (5, 16), (5, 15), (6, 15), (7, 15), (7, 16), (7,
17), (8, 17), (9, 17), (10, 17), (11, 17), (12, 17), (13, 17), (13,
16), (13, 15), (13, 14), (13, 13), (12, 13), (11, 13), (11, 12),
(11, 11), (12, 11), (13, 11), (14, 11), (15, 11), (15, 12), (15,
13), (16, 13), (17, 13), (17, 14), (17, 15), (17, 16), (17, 17),
(17, 18), (17, 19), (18, 19), (19, 19), (19, 20)]
```



Path taken by the agent: [(1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5), (5, 4), (5, 3), (6, 3), (7, 3), (8, 3), (9, 3), (9, 4), (9, 5), (8, 5), (7, 5), (7, 6), (7, 7), (8, 7), (9, 7), (9, 8), (9, 9), (9, 10), (9, 11), (10, 11), (11, 11), (11, 12), (11, 13), (12, 13), (13, 13), (14, 13), (15, 13), (16, 13), (17, 13), (17, 14), (17, 15), (18, 15), (19, 15), (19, 16), (19, 17), (18, 17), (17, 17), (17, 18), (17, 19), (18, 19), (19, 19), (19, 20)]

Conclusion

In this experiment, we focused on the fundamentals of intelligent agents by implementing a maze-solving agent in Python. The environment was set up as a maze, and the agent was designed to navigate from the entry point to the exit by intelligently exploring possible paths. Through the use of recursive depth-first search, the agent was able to identify a valid path to the goal, demonstrating its capability to make decisions and adapt to the environment. This experiment effectively showcased the principles of intelligent agents, emphasizing their ability to perceive, reason, and act within a given environment.