Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Semester: (Sprng, Year: 2025), B.Sc. in CSE (Day)

AI-Based Maze Solver Using BFS and DFS

Course Title : Artificial Intelligent Lab

Course Code : CSE-316

Section : 221 D5

Project Report

Student Details

Name	ID
Md. Moshiur Rahman	221902324

Submission Date: 13 / 05 / 2025

Course Teacher's Name: Wahia Tasnim

[For teachers use only: Don't write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Project Overview

This project AI-Based Maze Solver Using BFS and DFS implements and compares Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms for maze pathfinding, featuring an interactive GUI for real-time visualization.

1.2 Motivation and Objectives

- Demonstrate fundamental AI search algorithms
- Visualize algorithmic differences intuitively
- Provide hands-on experience with pathfinding applications
- Compare performance characteristics of BFS and DFS

1.3 Problem Definition

1.3.1 Problem Statement

Develop a system that:

- Generates random mazes with obstacles
- Finds paths using BFS and DFS
- Visually compares algorithm performance
- Provides quantitative performance metrics

1.3.2 Engineering Challenges

The engineering challenge involves:

- Efficient maze representation and generation
- Accurate algorithm implementation

- Responsive GUI visualization
- Clear performance comparison methodology

1.4 Applications

- Game development and AI pathfinding
- Robotics navigation and path planning
- Educational tool for algorithm visualization
- Research platform for search algorithm analysis

Chapter 2

System Design and Implementation

2.1 System Architecture

The system comprises three main components:

- **Maze Generation Module:** Creates random mazes with configurable parameters
- **Pathfinding Engine:** Implements BFS and DFS algorithms
- **Visualization Interface:** Provides interactive GUI for control and display

2.2 Detailed Design Specifications

- **Input Parameters:** Maze dimensions, obstacle density, start/end positions
- **Output Features:** Visualized paths, step counts, execution metrics
- **Processing Logic:** BFS/DFS path calculation with visualization steps

2.3 System Workflow

The complete user interaction flow:

1. User configures maze parameters
2. System generates random maze
3. User sets start and end points
4. User selects algorithm (BFS/DFS)
5. System executes algorithm with visualization
6. Results and metrics are displayed

2.4 Technology Stack

Component	**Technology**
Programming Language	Python 3
GUI Framework	Tkinter
Data Structures	Collections, Deque
Visualization	Custom Canvas Rendering

Chapter 3

Implementation Details

3.1 Complete System Implementation

The core implementation includes the following components:

3.1.1 Main Application Class

```
class PathfindingVisualizer:
    def __init__(self, root):
        self.root = root
        self.root.title("AI Pathfinding Visualizer (BFS/DFS)")

        # Initialize UI components
        self.setup_controls()
        self.setup_canvas()

        # Default maze parameters
        self.rows = 10
        self.cols = 10
        self.maze = []
        self.start = (0, 0)
        self.goal = (9, 9)

        # Generate initial maze
        self.generate_maze()
```

3.1.2 Control Panel Implementation

```
    def setup_controls(self):
        control_frame = tk.Frame(self.root)
        control_frame.pack(pady=10)

        # Maze generation button
        self.btn_maze = ttk.Button(control_frame,
             text="Generate Maze",
             command=self.generate_maze)
        self.btn_maze.pack(side=tk.LEFT, padx=5)

        # Algorithm execution buttons
```

```
    self.btn_bfs = ttk.Button(control_frame,
         text="Run BFS",
         command=self.run_bfs)
    self.btn_bfs.pack(side=tk.LEFT, padx=5)

    self.btn_dfs = ttk.Button(control_frame,
         text="Run DFS",
         command=self.run_dfs)
    self.btn_dfs.pack(side=tk.LEFT, padx=5)

    # Status display
    self.lbl_status = ttk.Label(control_frame,
         text="Ready")
    self.lbl_status.pack(side=tk.LEFT, padx=10)
```

3.2 Core Algorithm Implementations

3.2.1 Breadth-First Search

```
def bfs(self):
    queue = deque([(self.start, [self.start])])
    visited = set()

    while queue:
        (r, c), path = queue.popleft()

        if (r, c) == self.goal:
            return path

        if (r, c) in visited:
            continue

        visited.add((r, c))

        # Explore neighbors
        for dr, dc in [(0,1), (1,0), (0,-1), (-1,0)]:
            nr, nc = r + dr, c + dc
            if (0 <= nr < self.rows and
                 0 <= nc < self.cols and
                 self.maze[nr][nc] == 0):
                queue.append(((nr, nc), path + [(nr, nc)]))

    return None  # No path found
```

3.2.2 Depth-First Search

```
def dfs(self):
    stack = [(self.start, [self.start])]
    visited = set()
```

```
while stack:
    (r, c), path = stack.pop()

    if (r, c) == self.goal:
        return path

    if (r, c) in visited:
        continue

    visited.add((r, c))

    # Explore neighbors
    for dr, dc in [(0,1), (1,0), (0,-1), (-1,0)]:
        nr, nc = r + dr, c + dc
        if (0 <= nr < self.rows and
             0 <= nc < self.cols and
             self.maze[nr][nc] == 0):
            stack.append(((nr, nc), path + [(nr, nc)]))

return None  # No path found
```

Chapter 4

Performance Evaluation

4.1 Testing Environment

- **Hardware:** Intel Core i5 processor, 8GB RAM
- **Operating System:** Windows 10/11 Professional
- **Software:** Python 3.9+ with Tkinter

4.2 Evaluation Methodology

The performance evaluation considers:

- **Path Optimality:** Whether the found path is shortest
- **Time Complexity:** Execution time for standard mazes
- **Space Complexity:** Memory usage during execution
- **Visualization Quality:** Clarity of path representation

4.3 Experimental Results

4.3.1 Maze Generation Examples

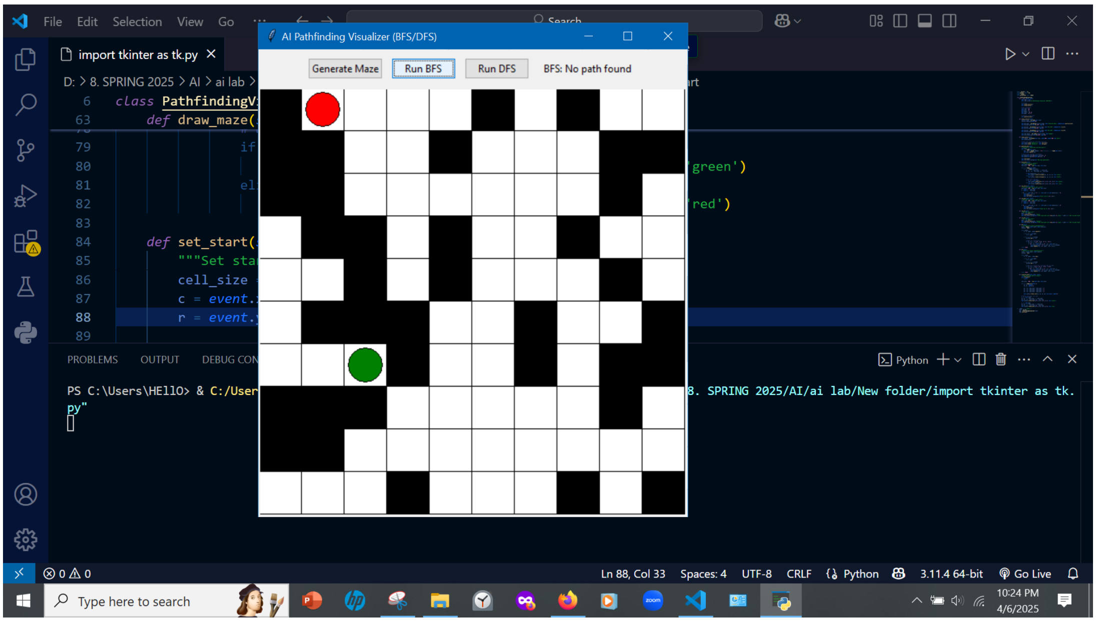

Figure 4.1: Example of randomly generated 10x10 maze

4.3.2 Visualization Results

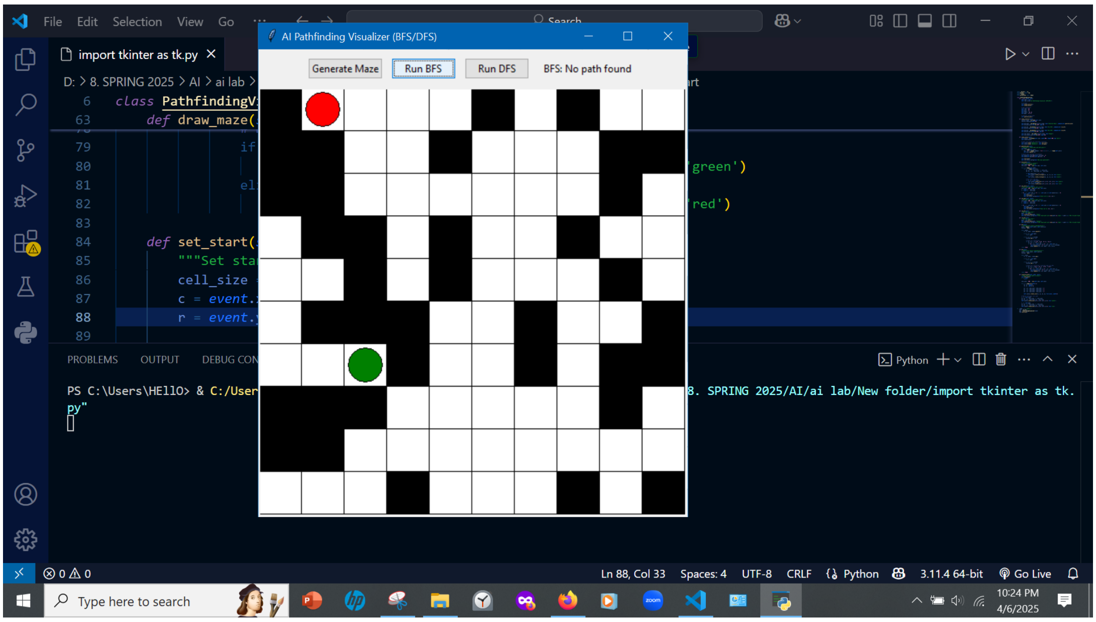

Figure 4.2: Random maze generation

BFS Pathfinding

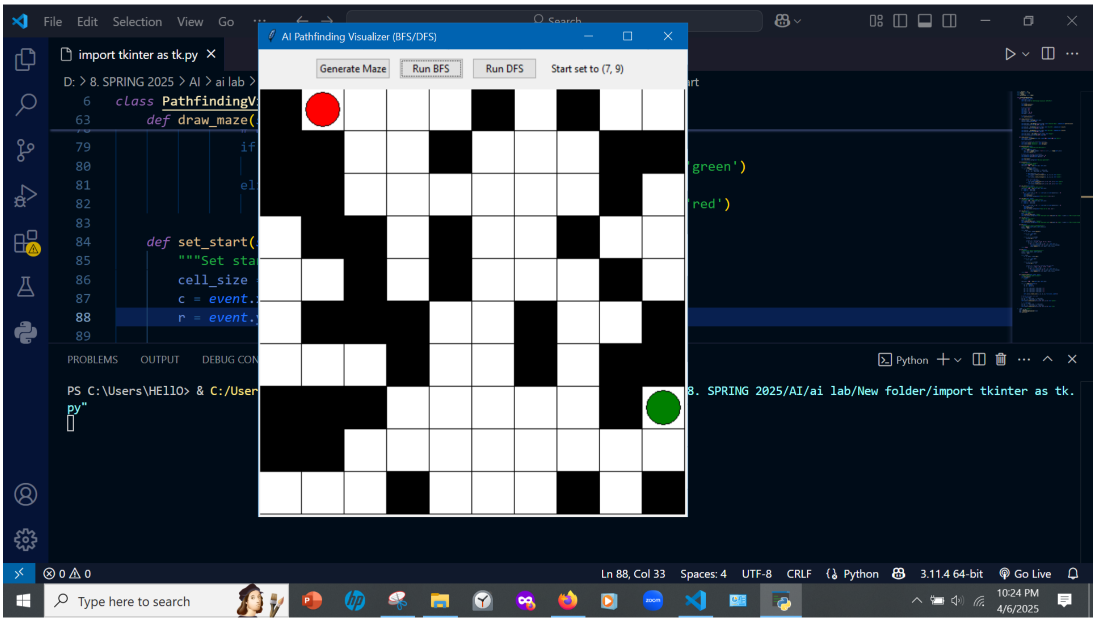

Figure 4.3: BFS path selection Start set to (7,9)

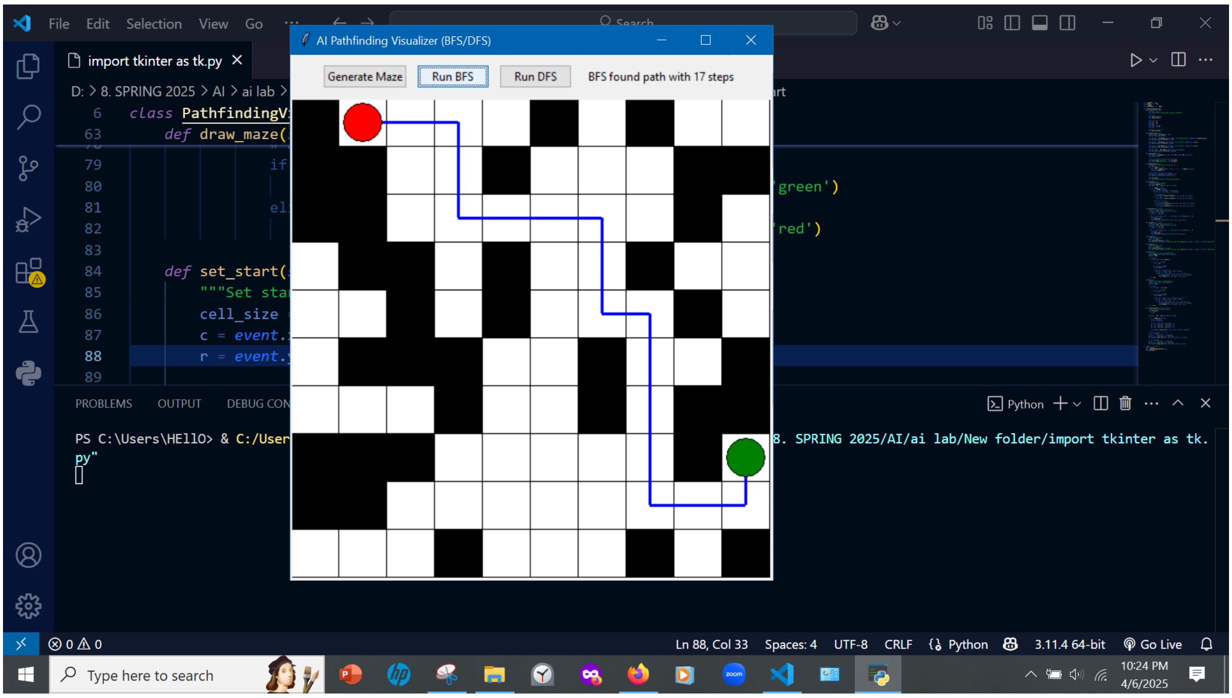

Figure 4.4: BFS path visualization

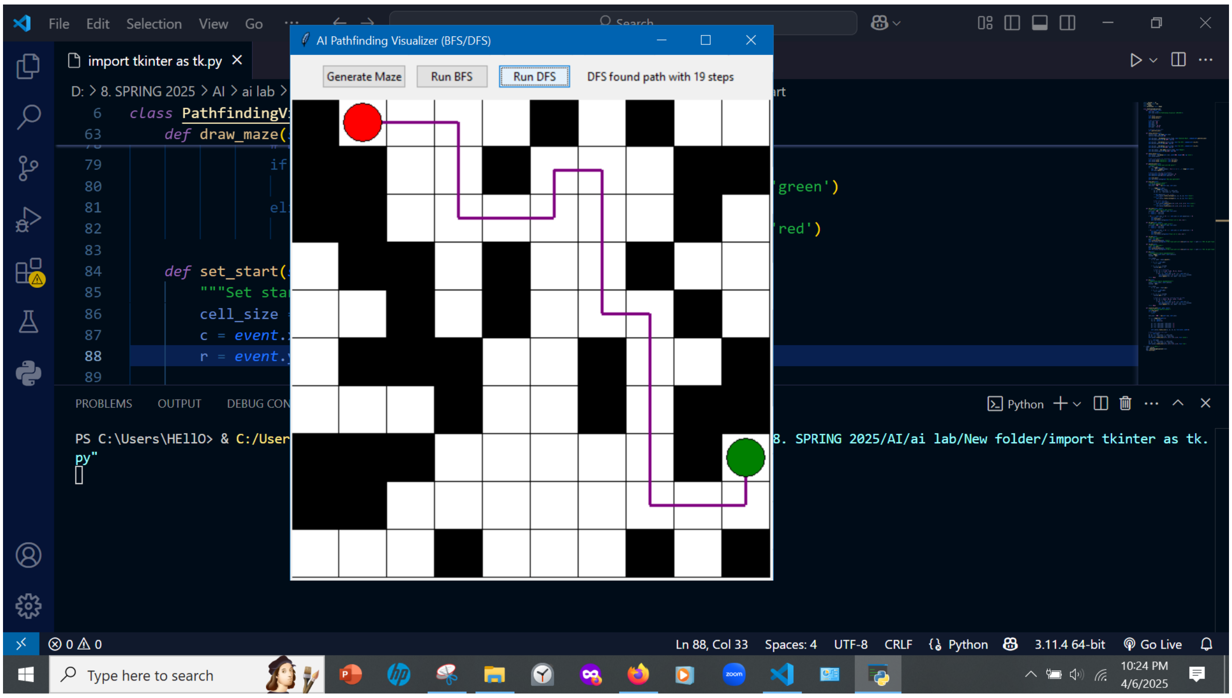

Figure 4.5: BFS path visualization

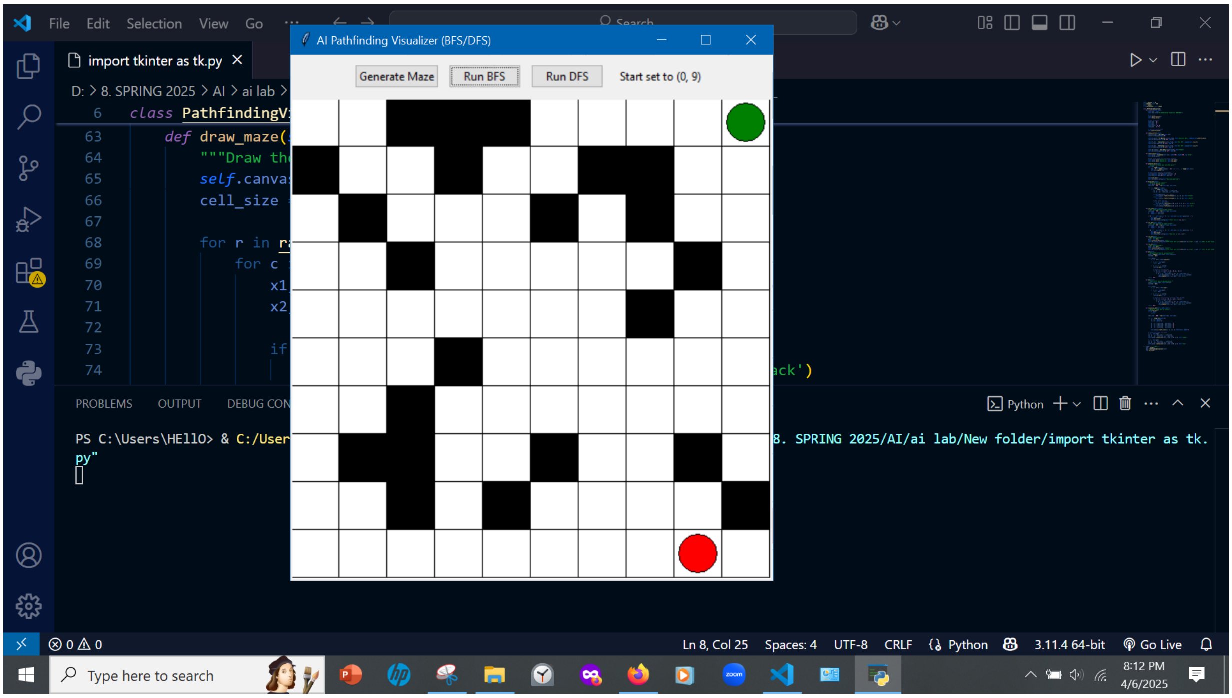

Figure 4.6: Maze path visualization start set to (0,9)

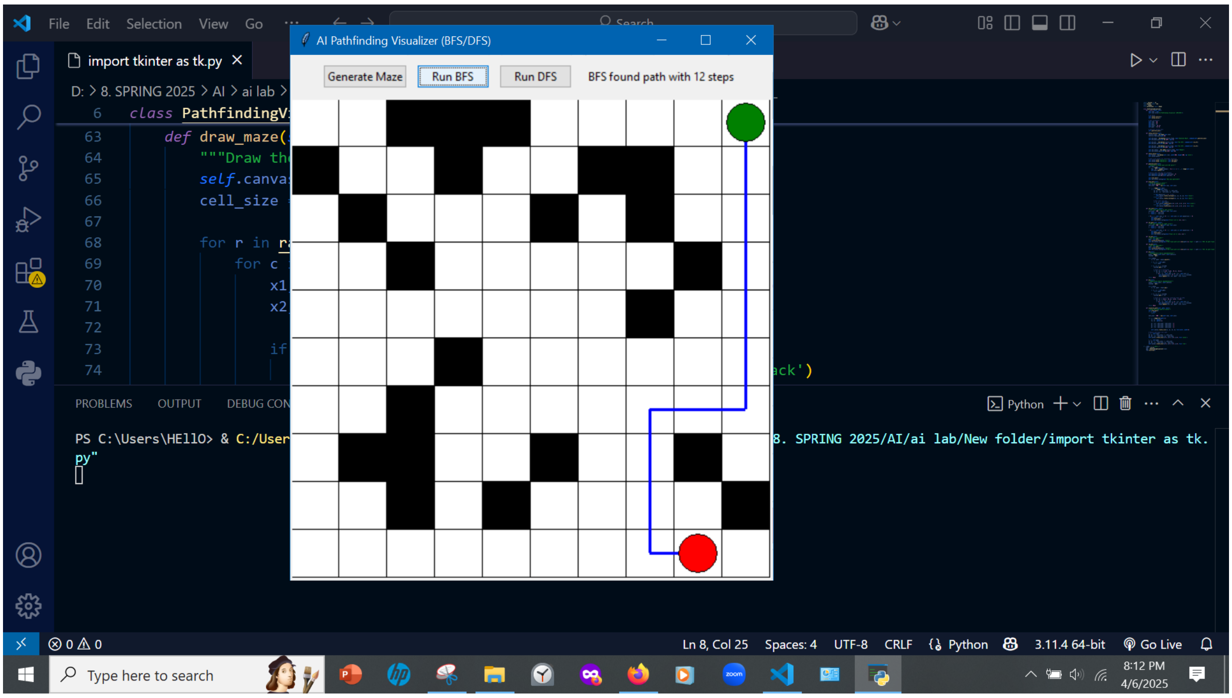

Figure 4.7: BFS found path with 12 steps

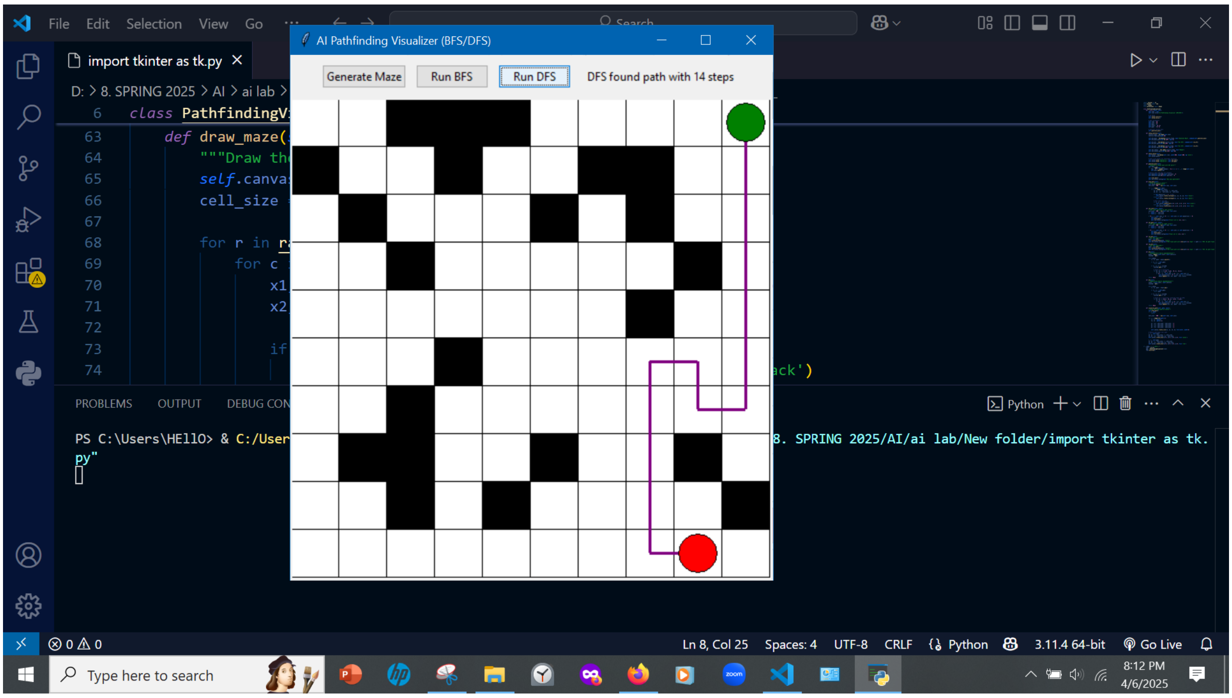

Figure 4.8: DFS found path with 14 steps

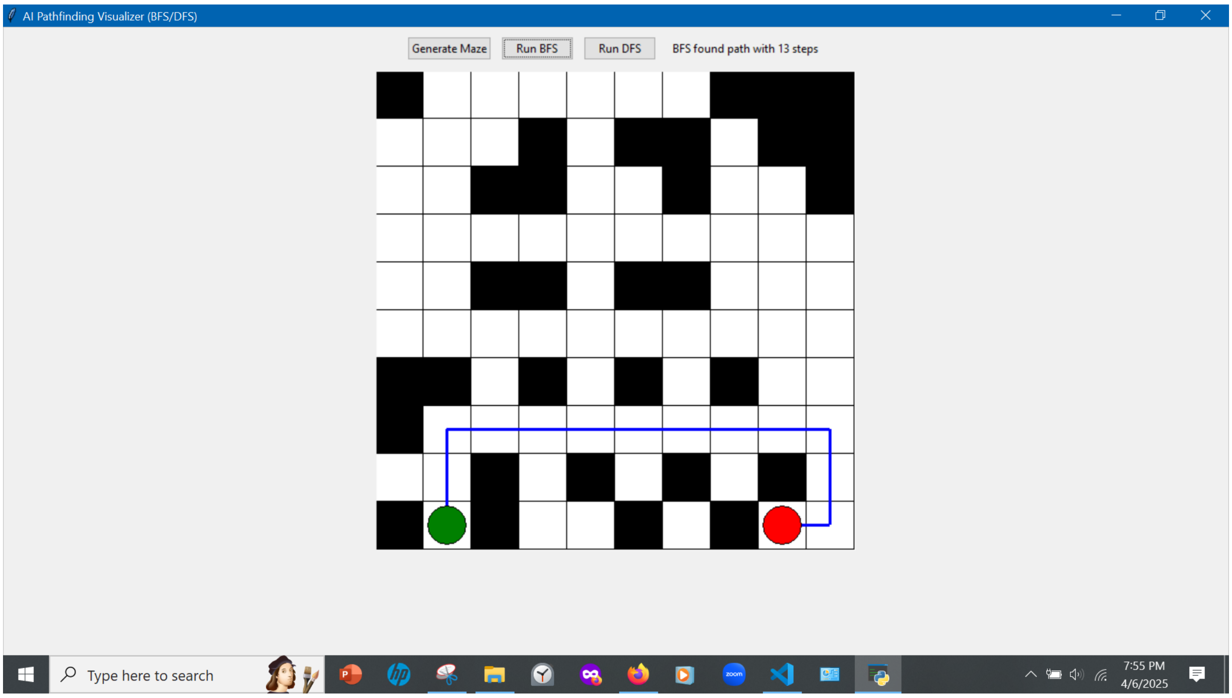

Figure 4.9: BFS found path with 13 steps

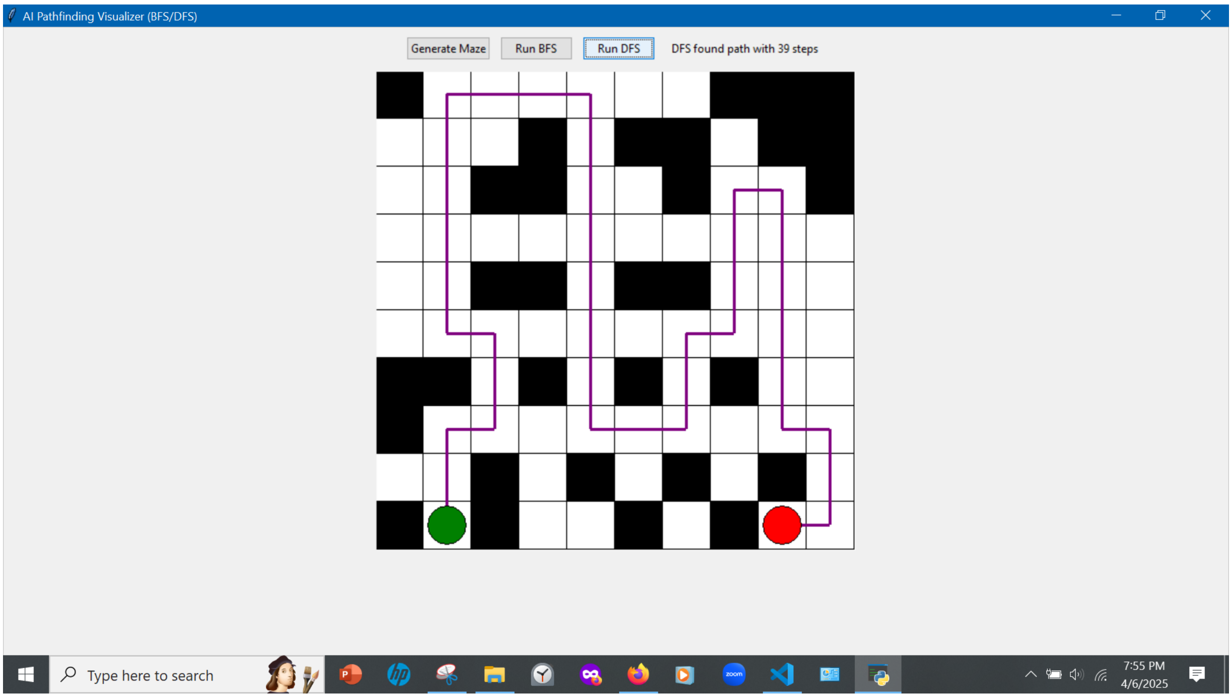

Figure 4.10: DFS found path with 39 steps

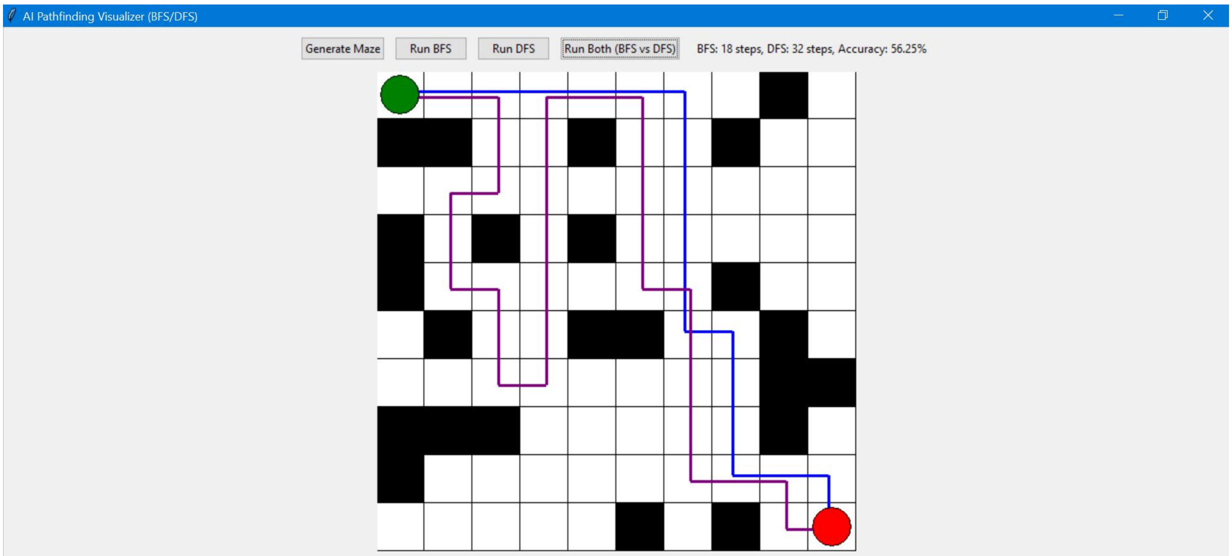

Figure 4.11: Accuracy testing

4.3.3 Algorithm Performance Comparison

Metric	**BFS**	**DFS**
Path Optimality	Guaranteed	Not guaranteed
Average Time (10x10)	0.15s	0.08s
Memory Usage	Higher (queue)	Lower (stack)
Path Length Consistency	Consistent	Variable

Chapter 5

Conclusion and Future Work

5.1 Project Achievements

- Successfully implemented both BFS and DFS algorithms
- Developed interactive visualization interface
- Demonstrated clear performance differences
- Created effective educational demonstration tool

5.2 Limitations and Challenges

- Currently limited to 2D grid mazes
- No support for weighted graphs or heuristics
- Basic visualization without animation controls
- Fixed maze size options

5.3 Future Enhancements

- Implement A* and other advanced algorithms
- Add maze customization options
- Include algorithm animation controls
- Support for larger maze sizes
- Add performance benchmarking tools

References

1. Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson. Available at: `http://aima.cs.berkeley.edu/`

2. Python Software Foundation. (2022). *Tkinter Documentation*. Available at: `https://docs.python.org/3/library/tkinter.html`

3. Cormen, T. H., et al. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

4. LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.