Maze Solving Algorithm Simulation

How to Run the Program

1. Extract the ZIP file containing the project.
2. Navigate to the extracted folder in your terminal.
3. Run the following command to execute the desired algorithm:

```
python3 DFS.py
```

Replace DFS.py with the corresponding script for other algorithms (e.g., Breadth_first_search.py, Best_First_search.py, or A_Star.py).

Assignment Description

In this assignment, I have implemented **four different algorithms** to simulate and solve maze navigation problems. These algorithms are:

1. **Breadth-First Search (BFS)**
2. **Depth-First Search (DFS)**
3. **Best-First Search**
4. **A* Search**

Project Overview

1. **Usage of pyMaze**
    - The project leverages the `pyMaze` package for maze generation and simulation.
    - This package supports random maze creation, obstacle customization, and GUI-based visualization using `tkinter`.
    - It also allows saving and loading specific maze configurations for comparison across algorithms.
2. **Key Features**
    - Randomly generate mazes of any size.
    - Adjust obstacle density to control maze complexity.
    - Save maze configurations for consistent testing across different algorithms.

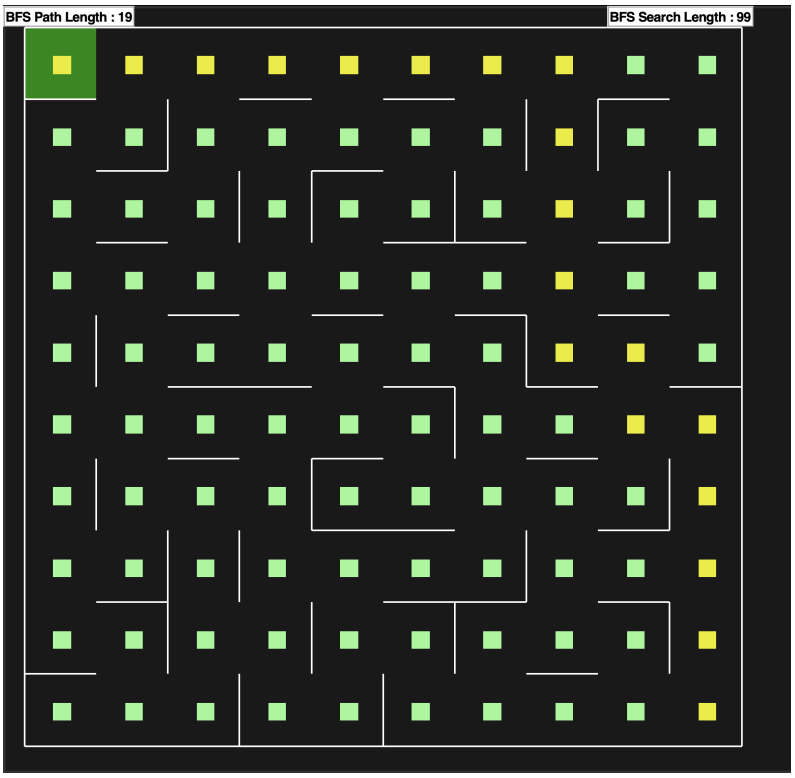
Algorithm Details

Breadth-First Search (BFS)

Description: Explores all neighbors of a cell before moving deeper. Guarantees the shortest path in the mazes.

Performance Table (BFS):

Maze Size	loopPercent (%)	Path Length	Nodes Expanded	Time (s)
10, 10	100	19	99	0.000203847885 13183594 sec
10, 10	50	19	94	0.000183820724 4873047 sec
100, 100	100	199	9999	1.326729774475 0977 sec
100, 100	50	213	9999	1.155523061752 3193 sec



BFS(10, 10) - Image 1

BFS Performance Analysis

The table shows the performance of the Breadth-First Search (BFS) algorithm with varying maze sizes and obstacle densities:

1. Maze Size:

   As the maze size increases from 10x10 to 100x100, both path length and nodes expanded increase. In larger mazes, BFS explores many more nodes, leading to longer computation times.

2. Loop Percent:

   Higher loopPercent (100%) means more obstacles. For smaller mazes (10x10), reducing loopPercent (50%) slightly improves performance (less time, fewer nodes expanded). For larger mazes (100x100), performance improvement is less noticeable due to the maze size.

3. Time Complexity:

   BFS takes very little time on smaller mazes (e.g., 0.0002s for 10x10). However, for larger mazes (100x100), the time grows significantly (e.g., 1.303s for 100% obstacles), highlighting the algorithm's inefficiency as maze size and obstacles increase.

Conclusion:

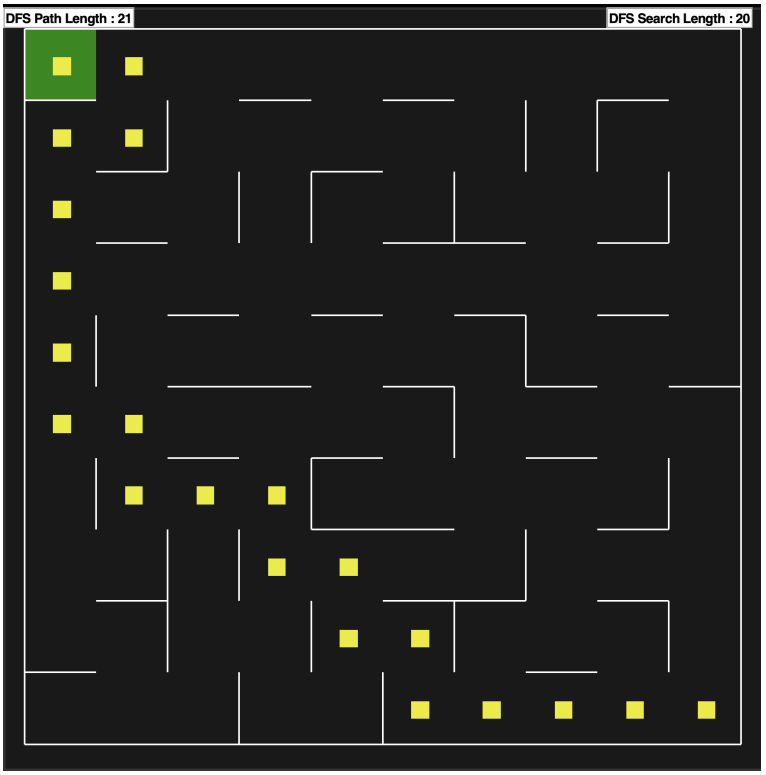
BFS is efficient on smaller mazes but becomes slower and less efficient as maze size and obstacle density increase.

Depth-First Search (DFS)

Description: Explores as far as possible along a branch before backtracking. Does not guarantee the shortest path.

Performance Table (DFS):

Maze Size	Obstacle (%)	Path Length	Nodes Expanded	Time (s)
10, 10	100	21	20	3.504753112792969e-05 sec
10, 10	50	37	43	3.0994415283203125e-05 sec
100, 100	100	235	236	0.0014231204986572266 sec
100, 100	50	299	316	0.0018990039825439453 sec



DFS(10, 10) - Image 1

DFS Performance Analysis

The table presents the performance of the Depth-First Search (DFS) algorithm with varying maze sizes and obstacle densities:

1. **Maze Size:**
   As the maze size increases from 10x10 to 100x100, both path length and nodes expanded grow. DFS explores more nodes in larger mazes, which increases the computational time.
2. **Obstacle Percent:**
   In smaller mazes (10x10), changing the loopPercent (from 50% to 100%) results in an increase in path length and nodes expanded, but the time remains quite low (around 0.00003s). In larger mazes (100x100), increasing obstacles leads to more exploration (higher nodes expanded), resulting in a slightly higher computation time (e.g., 0.0014s for 100% obstacles).
3. **Time Complexity:**
   DFS performs similarly to BFS in terms of time on small mazes, with times around 0.00003s for 10x10. However, on larger mazes (100x100), the time increases to 0.0014s or more, reflecting DFS's tendency to explore deeply into the maze, especially with higher obstacle densities.

Conclusion:

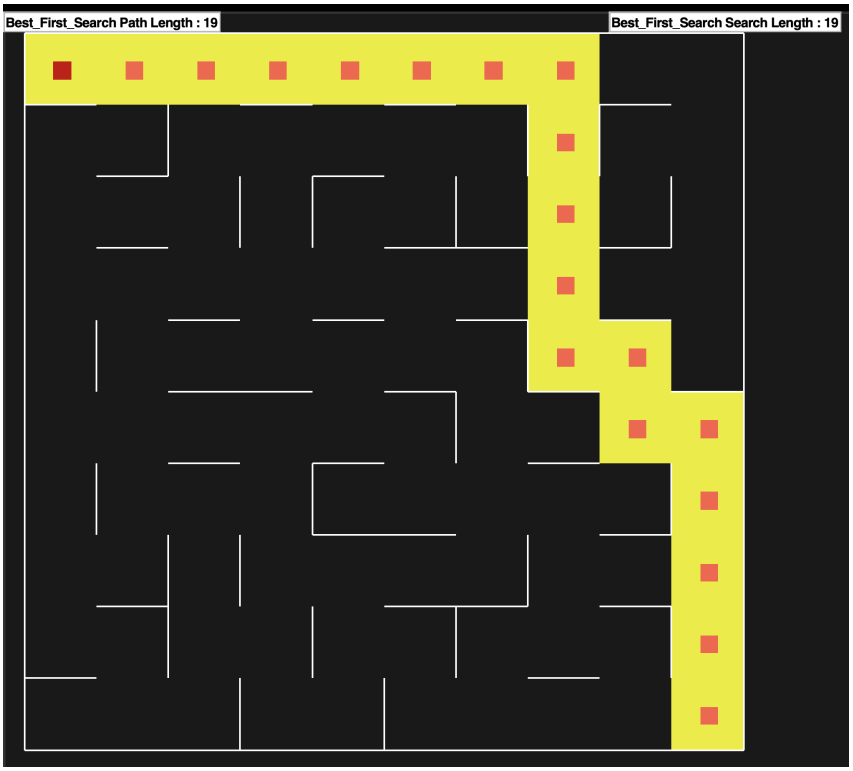
DFS is efficient for smaller mazes but faces slower performance on larger mazes with higher obstacle densities, as it explores more paths before finding the solution. The increase in path length and nodes expanded as maze size and obstacle density rise leads to longer computational times.

Best-First Search

Description: Uses a heuristic to prioritize exploration towards the goal. Does not guarantee the shortest path.

Performance Table (Best-First Search):

Maze Size	Obstacle (%)	Path Length	Nodes Expanded	Time (s)
10, 10	100	19	19	7.987022399902344e-05 sec
10, 10	50	21	23	9.012222290039062e-05 sec
100, 100	100	215	219	0.0012569427490234375 sec
100, 100	50	255	272	0.0015780925750732422 sec



Best-First Search(10, 10) - Image 1

Best-First Search Performance Analysis

The table shows the performance of the Best-First Search (BFS) algorithm with varying maze sizes and obstacle densities:

1. **Maze Size:**
   As the maze size increases from 10x10 to 100x100, both path length and nodes expanded increase. For instance, in the 100x100 maze, path length increases from 19 (10x10) to 215 (100x100), and nodes expanded increase from 19 to 219. This indicates that as the maze becomes larger, Best-First Search requires more steps and node exploration to reach the goal.
2. **Obstacle Percent:**
   As the loopPercent increases (from 50% to 100%), the path length and nodes expanded generally increase. This reflects how higher obstacle density creates more complex paths, forcing the algorithm to explore more nodes. For example, in the 10x10 maze, increasing obstacles from 50% to 100% causes a slight increase in both path length and nodes expanded.
3. **Time Complexity:**
   The time required for Best-First Search on smaller mazes (10x10) is very low, ranging from 0.00007s to 0.00009s. However, as the maze size grows (100x100), the time increases to 0.00125s for 100% obstacles and 0.00157s for 50% obstacles, showing that Best-First Search performance is sensitive to maze size and obstacle density. Larger mazes and denser obstacles increase the computational load.

Conclusion:

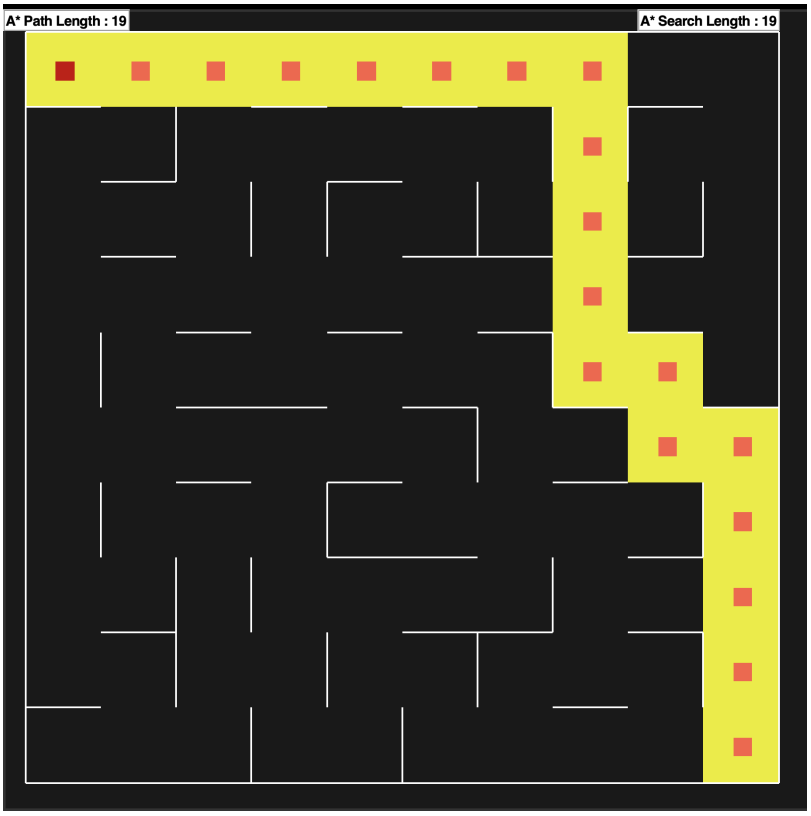
Best-First Search is efficient for small mazes, with minimal computational time. However, as maze size and obstacle density increase, the algorithm requires more exploration (nodes expanded), leading to a noticeable increase in both path length and computational time. Despite this, it still performs relatively well compared to other algorithms in larger mazes.

A* Search

Description: Combines BFS and heuristics to find the shortest path efficiently. Guarantees the shortest path in weighted mazes.

Performance Table (A* Search):

Maze Size	Obstacle (%)	Path Length	Nodes Expanded	Time (s)
10, 10	100	19	19	0.00012111663818359375 sec
10, 10	50	19	39	0.0001373291015625 sec
100, 100	100	199	1096	0.004474878311157226 6 sec
100, 100	50	213	5471	0.014770746231079102 sec



A* Search(10, 10) - Image 1

A* Search Performance Summary

1. **Maze Size:**

   As the maze size increases from 10x10 to 100x100, both path length and nodes expanded grow significantly, indicating higher computational cost.

2. **Obstacle Percent:**

   Increasing obstacle density leads to more nodes being expanded. For larger obstacles, nodes expanded and path length increase, especially in larger mazes.

3. **Time Complexity:**

   For small mazes, the time is minimal (0.00012s to 0.00014s). As maze size and obstacles increase, time also increases (up to 0.01477s), reflecting the algorithm's higher computational demand.

Conclusion:

A* Search performs efficiently on smaller mazes, but as maze size and obstacles increase, it requires more computation in terms of time, path length, and nodes expanded.

Comparison Between Algorithms

Maze Size	BFS	DFS	A*	Best-First
10, 10 - (100%)	0.00020384788513183594 sec	3.504753112792969e-05 sec	0.00012111663818359375 sec	7.987022399902344e-05 sec
10, 10 - (50%)	0.00018382072448730 47 sec	3.0994415283203125e-05 sec	0.00013732910156 25 sec	9.012222290039062e-05 sec
100, 100 - (100%)	1.3267297744750977 sec	0.0014231204986572266 sec	0.0044748783111572266 sec	0.0012569427490234375 sec
100, 100 - (50%)	1.1555230617523193 sec	0.0018990039825439453 sec	0.0147707462 31079102 sec	0.0015780925750732422 sec

Problem in macOS and Solution

When running the script on macOS, you may encounter an error similar to the following:

```
File "/Users/AI_Algorithm/DFS.py", line 1, in <module> from pyMaze
import maze, agent, COLOR, textLabel File
"/Users/AI_Algorithm/pyMaze.py", line 26, in <module> from tkinter
import * File
"/opt/homebrew/Cellar/python@3.13/3.13.0_1/Frameworks/Python.frame
work/Versions/3.13/lib/python3.13/tkinter/__init__.py", line 38,
in <module> import _tkinter # If this fails your Python may not be
configured for Tk ^^^^^^^^^^^^^^^
```

This error occurs because the `tkinter` module, required by `pyMaze`, is not properly configured in your Python installation.

Solution

Step 1: Verify tkinter Installation

Run the following command to check if `tkinter` is installed:

```
python3 -m tkinter
```

If a small GUI window opens, `tkinter` is installed and working. Otherwise, proceed to Step 2.

Step 2: Install tkinter for Python

If Python was installed using Homebrew, install `tkinter` with:

```
brew install python-tk
```

If the issue persists, reinstall Python with GUI support:

```
brew reinstall python
```

After resolving the issue, try running the program again.