**Task 1: Maze Generation**

**Overview:**

- **Generate a random maze with a size of 10x10 and a specified obstacle density.**
- **The maze includes a defined starting point (S) and a goal point (G).**

**Task 2: Depth-First Search (DFS)**

**Overview:**

- **Implement the Depth-First Search algorithm in Python.**
- **Find a path from the starting point to the goal point.**
- **Visualize the explored paths and the final solution.**

**Task 3: Breadth-First Search (BFS)**

**Overview:**

- **Implement the Breadth-First Search algorithm in Python.**
- **Find a path from the starting point to the goal point.**
- **Visualize the explored paths and the final solution.**

**Task 4: A* Algorithm**

**Overview:**

- **Implement the A* algorithm in Python with an appropriate heuristic function.**
- **Find an optimal path from the starting point to the goal point.**
- **Visualize the explored paths and the final solution.**

**Task 5: Performance Evaluation**

**Overview:**

- **Compare the performance of DFS, BFS, and A* algorithms.**
- **Evaluate in terms of Solution Path Length, Number of Nodes Expanded, and Time Execution.**

**Task 6: Visualization**

**Overview:**

- **Implement a visualization tool in Python using Pygame.**
- **Display the maze, explored paths, and final solution for each algorithm.**
- **Visualization should be interactive and highlight the progress of the search.**

**Maze Generation and Visualization Documentation**

**Overview**

This Python program generates a random maze, allowing visualization of the Depth-First Search (DFS), Breadth-First Search (BFS), and A* algorithms finding the optimal path from the starting point to the goal point. The visualization includes an interactive display of the maze, explored paths, and the final solution for each algorithm.

**Requirements**

- Python 3.x
- Pygame library (**pygame**)

Install the Pygame library using the following command:

bashCopy code

pip install pygame

**Components**

**1. Maze Generation**

- The **generate_maze** function creates a random maze with a size of 10x10 and a specified obstacle density. The maze includes a defined starting point (S) and a goal point (G).

pythonCopy code

maze = generate_maze()

**2. Depth-First Search (DFS)**

- The **depth_first_search** function implements the Depth-First Search algorithm to find a path from the starting point to the goal point.

pythonCopy code

path_dfs = depth_first_search(maze, start, goal)

**3. Breadth-First Search (BFS)**

- The **breadth_first_search** function implements the Breadth-First Search algorithm to find a path from the starting point to the goal point.

pythonCopy code

path_bfs = breadth_first_search(maze, start, goal)

**4. A* Algorithm**

- The **astar** function implements the A* algorithm with an appropriate heuristic function to find an optimal path from the starting point to the goal point.

pythonCopy code

path_astar = astar(maze, start, goal)

**5. Visualization**

- The **visualize_algorithm** function creates an interactive Pygame window to display the maze, explored paths, and the final solution for a specific algorithm.

pythonCopy code

screen_dfs = pygame.display.set_mode((len(maze[0]) * CELL_SIZE, len(maze) * CELL_SIZE))
visualize_algorithm(screen_dfs, depth_first_search, maze, start, goal, RED, BLUE)

**6. Main Program**

- The main program initializes the font, generates the maze, and creates separate Pygame windows for DFS, BFS, and A* visualization.

pythonCopy code

maze = generate_maze() initialize_font() screen_dfs = pygame.display.set_mode((len(maze[0]) * CELL_SIZE, len(maze) * CELL_SIZE)) visualize_algorithm(screen_dfs, depth_first_search, maze, start, goal, RED, BLUE) screen_bfs = pygame.display.set_mode((len(maze[0]) * CELL_SIZE, len(maze) * CELL_SIZE)) visualize_algorithm(screen_bfs, breadth_first_search, maze, start, goal, RED, BLUE) screen_astar = pygame.display.set_mode((len(maze[0]) * CELL_SIZE, len(maze) * CELL_SIZE)) visualize_algorithm(screen_astar, astar, maze, start, goal, RED, BLUE)

**Running the Program**

1. Run the main Python script:

bashCopy code

python maze_visualization.py

2. Pygame windows will open for DFS, BFS, and A* visualization, each displaying the maze, explored paths, and the final solution.

**Conclusion**

This program demonstrates the generation of random mazes and the visualization of different pathfinding algorithms. Users can interactively observe how DFS, BFS, and A* navigate through the maze to find the optimal path from the starting point to the goal.

DFS

| Run | Algorithm | Solution Path Length | Nodes Expanded | Execution Time | Status |
|---|---|---|---|---|---|
| 1 | DFS | 13 | 13 | 4.100799560546875 Seconds | Pass |
| 2 | DFS | 26 | 27 | 6.508827209472656 Seconds | Pass |
| 3 | DFS | 4 | 51 | 6.723403930664062 Seconds | Pass |
| 4 | DFS | 22 | 22 | 2.9087066650390625 Seconds | Pass |
| 5 | DFS | N/A | N/A | N/A | Fail |
| 6 | DFS | 50 | 56 | 6.008148193359375 Seconds | Pass |
| 7 | DFS | 20 | 25 | 3.0994415283203125 Seconds | Pass |
| 8 | DFS | 14 | 16 | 2.5272369384765625 Seconds | Pass |
| 9 | DFS | N/A | N/A | N/A | Fail |
| 10 | DFS | 31 | 47 | 5.078315734863281 Seconds | Pass |

BFS

| Run | Algorithm | Solution Path Length | Nodes Expanded | Execution Time | Status |
|---|---|---|---|---|---|
| 1 | BFS | 7 | 38 | 0.00015974044799804688 Seconds | Pass |
| 2 | BFS | 14 | 71 | 6.794929504394531 Seconds | Pass |
| 3 | BFS | 2 | 4 | 0.0001010894775390625 Seconds | Pass |
| 4 | BFS | 6 | 21 | 3.1948089599609375 Seconds | Pass |
| 5 | BFS | N/A | N/A | N/A | Fail |
| 6 | BFS | 8 | 52 | 5.1975250244140625 Seconds | Pass |
| 7 | BFS | 6 | 33 | 3.409385681152344 Seconds | Pass |
| 8 | BFS | 14 | 41 | 4.1961669921875 Seconds | Pass |
| 9 | BFS | N/A | N/A | N/A | Fail |
| 10 | BFS | 5 | 26 | 3.409385681152344 Seconds | Pass |

A*

| Run | Algorithm | Solution Path Length | Nodes Expanded | Execution Time | Status |
|---|---|---|---|---|---|
| 1 | A* | 7 | 11 | 3.0994415283203125 Seconds | Pass |
| 2 | A* | 14 | 45 | 7.700920104980469 Seconds | Pass |
| 3 | A* | 2 | 2 | 2.09808349609375 Seconds | Pass |
| 4 | A* | 6 | 7 | 2.6702880859375 Seconds | Pass |
| 5 | A* | N/A | N/A | N/A | Fail |
| 6 | A* | 8 | 15 | 3.814697265625 Seconds | Pass |
| 7 | A* | 6 | 11 | 2.6941299438476562 Seconds | Pass |
| 8 | A* | 14 | 48 | 5.412101745605469 Seconds | Pass |
| 9 | A* | N/A | N/A | N/A | Fail |
| 10 | A* | 5 | 8 | 7.486343383789062 Seconds | Pass |