Project Report

Path Finding Algorithm analysis using Maze game

Abstract

The “Path Finding Algorithm analysis using Maze game" project is an interactive tool designed to demonstrate the behavior of various pathfinding algorithms, specifically A* (A-star), Breadth-First Search (BFS), and Depth-First Search (DFS), within the context of a randomly generated maze. The primary objective of the project is to allow users to visualize and compare how these algorithms find a path from a starting point to a destination while avoiding obstacles. The maze is dynamically created using a recursive backtracking algorithm, ensuring diverse and challenging scenarios for the pathfinding algorithms to solve.

The methods used in the project include implementing the three pathfinding algorithms, visualizing their exploration process step-by-step in real-time, and rendering the maze using Python's Tkinter library. The project also provides an interactive user interface where users can select the start and end points of the maze, choose one of the algorithms, and watch the algorithm in action as it solves the maze. The visualization displays the algorithm's pathfinding steps, highlighting the paths explored and the final optimal (or viable) solution.

The results of the project show that A* is the most efficient, finding the shortest path in the least amount of time, especially in larger mazes. BFS guarantees the shortest path but can be less efficient in terms of time and space, as it explores all possible paths level by level. DFS, on the other hand, may not always find the shortest path but demonstrates a simpler, deeper exploration of the maze.

In conclusion, the project effectively demonstrates the strengths and limitations of each algorithm through real-time visualization. It serves as an educational tool for understanding pathfinding algorithms, graph traversal, and optimization. By allowing users to interact with the maze and observe algorithmic behavior, the project enhances the understanding of computational concepts and provides a practical platform for experimenting with pathfinding methods.

1. Introduction

Background

Pathfinding is a fundamental problem in Artificial Intelligence (AI), which plays a crucial role in various real-world applications, including robotics, video games, autonomous vehicles, network routing, and geographic information systems (GIS).

Several algorithms exist to tackle pathfinding problems, with **A*** (A-star), **Breadth-First Search (BFS)**, and **Depth-First Search (DFS)** being some of the most popular. *A (A-star)** is a widely used heuristic-based algorithm that combines the benefits of **graph traversal** and **heuristic search** to efficiently find the shortest path. **BFS**, on the other hand, guarantees the shortest path in an unweighted grid but can be computationally expensive, especially in large mazes. **DFS** explores a path deeply, but unlike BFS, it doesn't guarantee the shortest path, and its performance can degrade depending on the maze structure. Understanding how these algorithms work, their strengths, and their limitations is essential for selecting the most appropriate method in real-world applications, from games to robotics.

This project aims to visualize and compare these three pathfinding algorithms, allowing users to explore their behavior in navigating dynamically generated mazes. By providing an interactive tool to compare A*, BFS, and DFS, this project serves as both a learning resource and a demonstration of fundamental concepts in AI, such as graph traversal, optimization, and decision-making.

Objectives

The main goals of this project are:

1. **Visualize Pathfinding Algorithms**: To create an interactive platform that visually demonstrates the workings of A*, BFS, and DFS in solving mazes.
2. **Compare Algorithm Performance**: To compare the performance of A*, BFS, and DFS in terms of pathfinding efficiency, computational complexity, and the quality of paths found.
3. **Provide Interactive Learning**: To offer an engaging user interface that allows users to set the maze's starting and ending points, select the algorithm, and observe how the algorithm solves the maze in real-time.
4. **Enhance Understanding of Pathfinding**: To serve as an educational tool for students and enthusiasts to learn about the inner workings of pathfinding algorithms and their practical applications in AI.

Problem Statement

The project addresses the challenge of solving pathfinding problems in dynamic, maze-like environments. Specifically, it aims to compare the performance of three well-known algorithms—A*, BFS, and DFS—in finding the shortest or most viable path through a maze.

2. Literature Review

Related Work:

A] Maze Generation Algorithms:

Maze generation has been extensively studied, with algorithms such as **Prim's Algorithm**, **Kruskal's Algorithm**, and **Recursive Backtracking** being widely utilized. The Recursive Backtracking method, which forms the foundation of this project, is noted for its simplicity and efficiency in generating perfect mazes—structures with one unique path between any two points. The algorithm systematically explores the maze space using depth-first search (DFS), backtracking when no further progress can be made.

B] Pathfinding Algorithms

Pathfinding is a critical area of study, especially in game development and robotics. Popular algorithms include:

- ***A* Algorithm:*** A heuristic-based search algorithm combining Dijkstra's shortest path approach with a heuristic to prioritize exploration. It is valued for its optimality and efficiency when an admissible heuristic is used.
- **Breadth-First Search (BFS):** An uninformed search technique that guarantees the shortest path in an unweighted grid. Its simplicity makes it a foundational approach in pathfinding studies.
- **Depth-First Search (DFS):** A less optimal but straightforward method for exploring paths, often used as a teaching tool for recursion and stack-based traversal.

C] Relevance to Applications

Research has highlighted the importance of pathfinding in robotics, particularly for autonomous navigation in structured and unstructured environments. The use of grid-based representations for the environment and the application of algorithms like A* for obstacle avoidance have been extensively documented. Similarly, maze generation and traversal algorithms are integral in procedural content generation for video games and simulations.

Research Papers and Studies:

- Zeng, W.; Church, R. L. (2009). "Finding shortest paths on real road networks: the case for A*". International Journal of Geographical Information Science. Link
- IFN Application in BFS of Artificial Intelligence Link
  Published in: 2008 International Conference on MultiMedia and Information Technology
  Date of Conference: 30-31 December 2008
  Date Added to IEEE Xplore: 19 June 2009 Publisher: IEEE
  Print ISBN:978-0-7695-3556-2 Conference Location :Three Gorges, China
  DOI: 10.1109/MMIT.2008.53

3. Methodology

Approach:

In this project, we implement maze generation and pathfinding algorithms, with a focus on providing an interactive visualization. The maze is generated using the Recursive Backtracking algorithm, which ensures a perfect maze with a single solution between any two points. For pathfinding, three algorithms are utilized: A*, BFS (Breadth-First Search), and DFS (Depth-First Search). A* combines Dijkstra's algorithm with a heuristic for efficient and optimal pathfinding. BFS guarantees the shortest path in an unweighted grid by exploring all neighbors level by level, while DFS explores paths by going as deep as possible before backtracking. The system allows users to visualize the maze generation and the traversal process of each algorithm in real-time, providing an engaging and educational tool for understanding these fundamental concepts.

Algorithms and Techniques:

In this project, we implement fundamental search algorithms used in AI for maze traversal. The algorithms include:

1. **Recursive Backtracking (Maze Generation)**: This algorithm generates mazes by employing Depth-First Search (DFS) with backtracking. Starting from a random cell, it marks the cell as visited, chooses a neighboring unvisited cell at random, and moves to it, removing the wall between the two cells. This process continues until it reaches a dead end, at which point the algorithm backtracks to the previous cell with unvisited neighbors and resumes exploration. This method ensures the maze is a "perfect maze," meaning it has exactly one solution with no loops or isolated areas, making it ideal for maze generation.
2. ***A* Algorithm (Pathfinding)****: A* is a heuristic-driven search algorithm that combines the strengths of Dijkstra's algorithm and greedy best-first search. It calculates the cost of reaching a node by combining the actual cost from the start node (g(n)) and an estimated cost to the goal (h(n)), where h(n) is a heuristic function. This allows it to prioritize paths that are more promising while still guaranteeing the shortest path if the heuristic is admissible (never overestimates). A* is highly efficient for weighted grids and enables informed and optimal pathfinding, making it suitable for scenarios like game AI and navigation.
3. **Breadth-First Search (BFS)**: BFS is an uninformed search algorithm that explores all possible paths level by level from the starting node. It uses a queue to ensure that nodes closer to the start are explored first, guaranteeing the shortest path in an unweighted maze. BFS is particularly useful for problems requiring all possible solutions or shortest path discovery in simple grids, but its memory usage can grow significantly in large or complex mazes as it needs to store all nodes at the current frontier.
4. **Depth-First Search (DFS)**: DFS explores as far as possible along one branch before backtracking to explore alternative paths. It uses a stack (either explicitly or through recursion) to track visited nodes and is efficient in terms of memory since it doesn't need to store all nodes at a level like BFS. However, DFS is not optimal for finding the shortest

path, as it may prioritize deeper paths that are longer. It is simple and often used in maze traversal to explore all possibilities or understand recursion concepts in search algorithms.

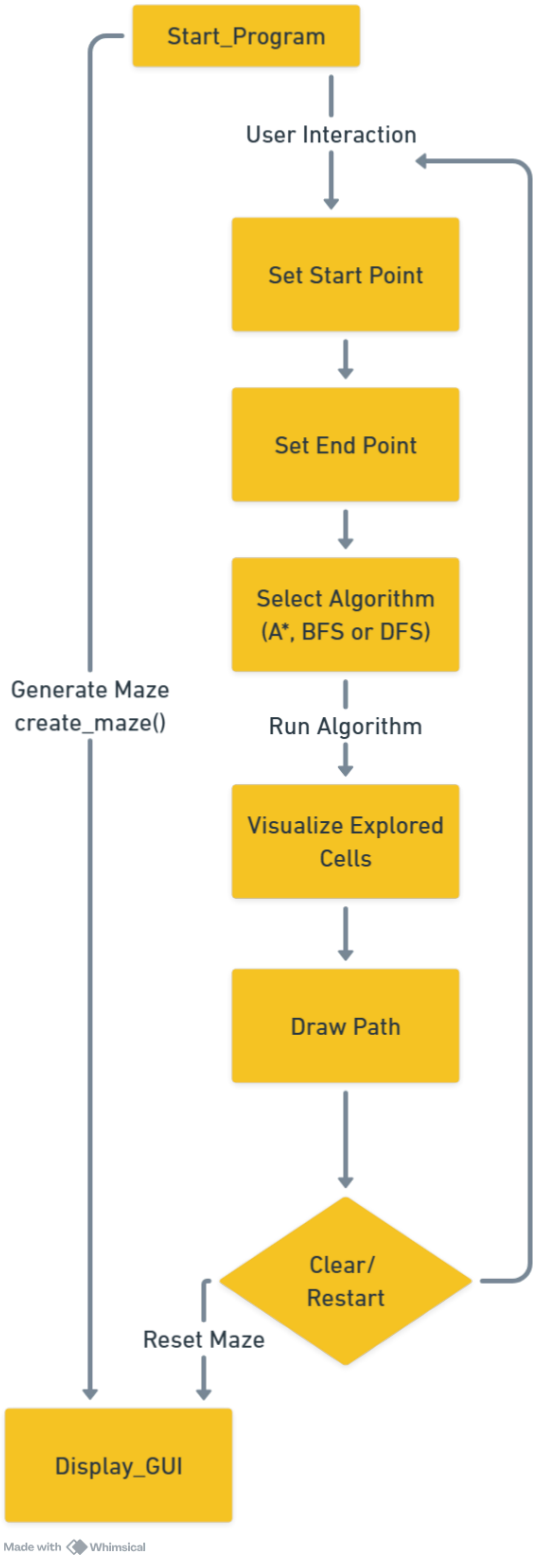
These algorithms are implemented to provide both maze generation and pathfinding capabilities, helping visualize their behavior in real-time and compare their performance in terms of efficiency and pathfinding effectiveness.

Tools and Technologies:

1) **Programming Language**: Python
2) **GUI Framework**: Tkinter (for creating the graphical user interface and visualizing the maze and pathfinding process)
3) **Data Structures**: Python's built-in list, queue, and dictionary for managing the maze grid, exploring nodes, and storing pathfinding data
4) **Algorithm Implementations**: Custom implementations of A*, BFS (Breadth-First Search), and DFS (Depth-First Search) for pathfinding
5) **Visualization**: Tkinter's Canvas widget for drawing the maze and pathfinding steps
6) **Randomization**: Python's random module for generating random maze paths
7) **Time Delay**: Python's time module to create delays in the visualization of the pathfinding process.

4. Implementation

Process:



Code Snippets:
Include key parts of the code (if relevant) and explain them.

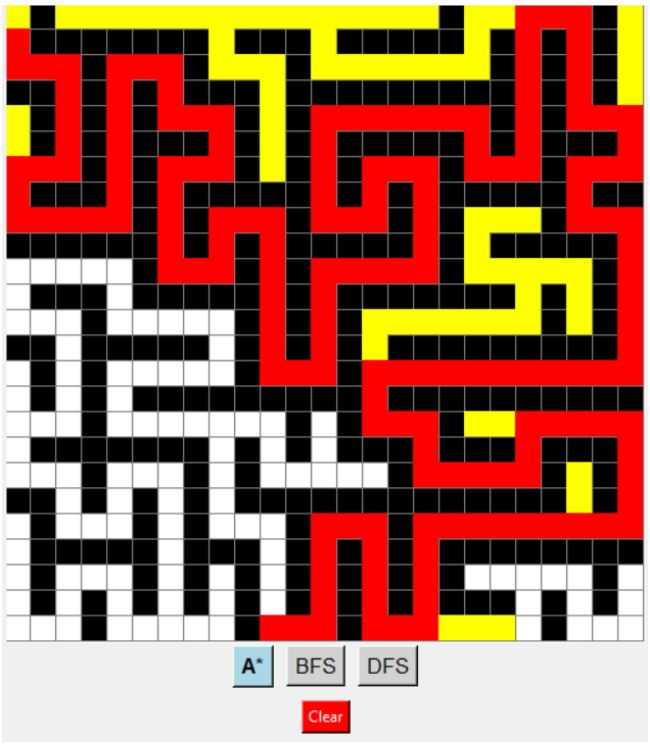
```python
def dfs(maze, start, end, canvas, cell_size):
    stack = [start]
    came_from = {start: None}
    explored = []

    while stack:
        current = stack.pop()
        if current == end:
            visualize(canvas, explored, cell_size, color="magenta")
            path = []
            while current:
                path.append(current)
                current = came_from[current]
            path.reverse()
            return path

        x, y = current
        explored.append(current)

        for dx, dy in [[1, 0], [-1, 0], [0, 1], [0, -1]]:
            neighbor = (x + dx, y + dy)
            if maze.is_wall(*neighbor) or neighbor in came_from:
                continue
            stack.append(neighbor)
            came_from[neighbor] = current
    return None
```

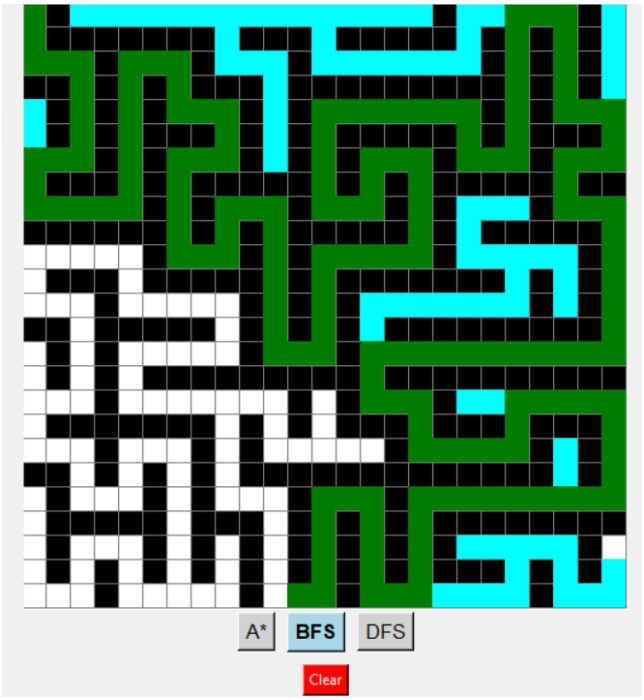
```python
def bfs(maze, start, end, canvas, cell_size):
    queue = [start]
    came_from = {start: None}
    explored = []

    while queue:
        current = queue.pop(0)
        if current == end:
            visualize(canvas, explored, cell_size, color="cyan")
            path = []
            while current:
                path.append(current)
                current = came_from[current]
            path.reverse()
            return path

        x, y = current
        explored.append(current)

        for dx, dy in [[1, 0], [-1, 0], [0, 1], [0, -1]]:
            neighbor = (x + dx, y + dy)
            if maze.is_wall(*neighbor) or neighbor in came_from:
                continue
            queue.append(neighbor)
            came_from[neighbor] = current
    return None
```

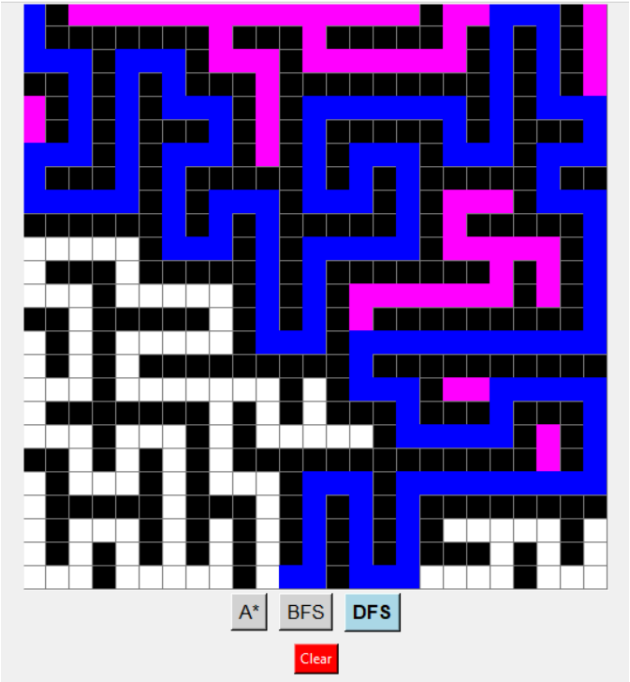
```python

def a_star(maze, start, end, canvas, cell_size):
    open_set = PriorityQueue()
    open_set.put((0, start))
    came_from = {}
    g_score = {start: 0}
    f_score = {start: heuristic(start, end)}
    explored = []

    while not open_set.empty():
        _, current = open_set.get()
        if current == end:
            visualize(canvas, explored, cell_size, color="yellow")
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.reverse()
            return path

        x, y = current
        explored.append(current)

        for dx, dy in [[1, 0], [-1, 0], [0, 1], [0, -1]]:
            neighbor = (x + dx, y + dy)
            if maze.is_wall(*neighbor):
                continue
            tentative_g_score = g_score[current] + 1
            if tentative_g_score < g_score.get(neighbor, float('inf')):
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = tentative_g_score + heuristic(neighbor, end)
                open_set.put((f_score[neighbor], neighbor))
    return None
```

5. Results and Discussion

Results:







Analysis:

The project successfully meets its objectives by providing real-time visualizations of A*, BFS, and DFS algorithms, enabling users to observe and compare their performance. It demonstrates each algorithm's strengths, weaknesses, and trade-offs in pathfinding efficiency and path quality. The interactive interface enhances learning through user-defined start/end points and algorithm selection. While effective as an educational tool, the project could improve by addressing scalability challenges and incorporating more advanced pathfinding algorithms.

6. Conclusion

Summary of Findings:

ASPECT	A*	BFS	DFS
Number of Cells Explored	Explores fewer cells than BFS if the heuristic is good, focusing on cells likely to lead to the goal.	Explores all cells at a given "depth" uniformly, often visiting more cells than A*.	Explores many unnecessary cells, especially in dead-end paths, as it goes deep without prioritizing.
Path Optimality:- (If the maze contains more than one path from start to end)	Guarantees the shortest path (if heuristic is admissible).	Guarantees the shortest path in unweighted mazes.	Does not guarantee the shortest path; may find suboptimal solutions.
Memory Usage	High: Stores all visited cells and the frontier in memory.	High: Stores all visited cells and the current level of nodes in memory.	Low: Only keeps track of the current path and minimal backtracking information.

Accomplishments:-

By generating random mazes every time, the project ensures diverse test cases and helps us to understand the working of search algorithms (A*, bfs,dfs) and compare the tradeoffs between the three algorithms in an interactive way..

Limitations:

The project faces limitations such as reduced scalability in larger mazes, DFS's inability to guarantee the shortest path, reliance on static maze environments, limited algorithm selection (excluding advanced methods like Dijkstra's), and hardware dependency. These factors impact its performance and adaptability, offering scope for future improvements and enhancements.

Future Work:

This project could be further transformed into a timed maze game played by humans where the player has to collect coins while reaching the goal. To find the shortest path to the nearest coin, the player can take help of search algorithms (bfs, dfs or A*), if he/she has enough coins.

7. References:-

https://levelup.gitconnected.com/a-star-a-search-for-solving-a-maze-using-python-with-visualization-b0cae1c3ba92

https://www.youtube.com/watch?v=VMP1oQOxfM0&ab_channel=edureka%21

https://youtu.be/tvAh0JZF2YE?feature=shared\

https://youtu.be/pcKY4hjDrxk?feature=shared