PROJECT REPORT
Smart Attendance System

Submitted by

AVTAR SINGH

25MCI10325

avtarsinghraja96@gmail.com

GITHUB:-https://github.com/AVTARSINGH1234/PYTHON-PROJECT.git

in partial fulfilment for the award of the degree of

MASTER OF COMPUTER APPLICATIONS ARTIFICIAL INTELLIGENT

Chandigarh University

November 2025

Acknowledgement

I would like to express my sincere gratitude saranjeet kaur, my project guide, for his valuable guidance, continuous support, and encouragement throughout the development of this project titled **"Face Recognition Attendance System and emotion "**

His insightful feedback, technical knowledge, and patience have been instrumental in helping me understand the core concepts of Python, OpenCV, and Tkinter used in this project.

I am also thankful to **Chandigarh University** for providing the resources and learning environment that made this work possible.

Finally, I extend my heartfelt thanks to my family and friends for their constant motivation and moral support throughout this project.

Contents

Abstract

The *Random Solvable Maze – BFS, DFS, and A Search Comparison** project is an interactive visualization tool designed to study and compare three fundamental search algorithms: **Breadth-First Search (BFS)**, **Depth-First Search (DFS)**, and **A*** (A-star). The project's primary objective is to create a **randomly generated maze that is always solvable**, and then analyze how each algorithm explores, searches, and identifies the path from a **start node (S)** to a **goal node (G)**.

Developed using **Python** and the **Tkinter** graphical user interface library, this project visually demonstrates the working of each algorithm in real-time, showing the sequence of node exploration and the final path found. The results of each algorithm are compared based on **path length**, **execution time**, and **efficiency**.

The system serves as an educational and experimental tool to understand search strategies in Artificial Intelligence and their applications in fields such as **robotics**, **game development**, and **navigation systems**.

1. Introduction

Pathfinding and maze generation are fundamental problems in the field of Artificial Intelligence and Computer Science. They serve as practical demonstrations of how search algorithms work to find optimal or feasible routes between two points in a constrained environment.

This project, *"Random Solvable Maze – BFS, DFS, and A Search Comparison,"** focuses on generating a random but always solvable maze and analyzing the performance of three popular pathfinding algorithms — **Breadth-First Search (BFS)**, **Depth-First Search (DFS)**, and **A* (A-star)**.

The system uses the **Tkinter** library in Python to create a graphical interface where the maze is dynamically generated, displayed, and solved step by step with real-time visualization. Each algorithm explores the maze differently:

- **BFS (Breadth-First Search)** explores all possible paths level by level and guarantees the shortest path in an unweighted maze.
- **DFS (Depth-First Search)** dives deep into one path before backtracking, which can be faster but doesn't always find the shortest route.
- **A*** combines the best of both by using a heuristic function to estimate the distance to the goal, often achieving optimal and efficient results.

By visualizing these algorithms, users can clearly observe how each approach explores nodes, handles branching, and finds the solution path. The comparison includes both **path length** and **execution time**, helping to understand the **trade-offs between speed and accuracy** in different search strategies.

This project not only strengthens the understanding of search algorithms but also demonstrates how artificial intelligence techniques can be applied to problem-solving in real-world navigation systems, robotics, and game development.

.

2. Tools and Technologies Used

The project employs several programming tools, libraries, and technologies to ensure efficient functioning and interactive visualization.

2.1 Software and Libraries

Tool / Library	Purpose
Python	Core programming language used for algorithm implementation and logic.
Tkinter	GUI library used for interactive visualization of the maze.
queue / PriorityQueue	Data structures used for BFS and A* algorithms to manage node exploration.
random	Generates random maze structures ensuring variability.
time	Measures algorithm performance and execution duration.

2.2 Algorithms Implemented

1. **Breadth-First Search (BFS):** Level-order traversal that ensures the shortest path in unweighted graphs.

2. **Depth-First Search (DFS):** Stack-based traversal that goes deep before backtracking.

3. **A* Search Algorithm:** A heuristic-based method that minimizes total cost f(n) = g(n) + h(n), where g(n) is the cost so far, and h(n) is the heuristic (Manhattan distance).

2.3 Platform and Environment

- **Operating System:** Windows / Linux / macOS
- **Language Version:** Python 3.10+
- **IDE Used:** Visual Studio Code / PyCharm / Jupyter Notebook
- **Visualization Library:** Tkinter Canvas

2.4 System Requirements

- Minimum 4GB RAM
- Python 3.x Installed
- Basic understanding of algorithmic concepts

3. Implementation Steps

The implementation of this project was divided into multiple phases to ensure clarity and modularity.

Step 1: Maze Generation

The maze is represented as a **2D grid** consisting of walls and open paths. A random maze generator ensures that the generated structure always contains a valid path between the **Start (S)** and **Goal (G)** points.
The maze uses:

- 1 for walls
- 0 for open paths
- (x1, y1) as the start point
- (x2, y2) as the goal point

The algorithm guarantees at least one solution path by connecting the start and goal during generation, avoiding unsolvable configurations.

Step 2: Visualization Setup

Using **Tkinter**, the maze is displayed as a grid of colored rectangles. Each type of cell is color-coded for clarity:

Cell Type	Color	Description
Start (S)	Green	Entry point of maze
Goal (G)	Red	Destination point
Wall	Black	Obstacle
Empty Path	White	Free space to move
Visited Node	Blue	Explored by algorithm
Final Path	Yellow	Successful shortest path

Real-time visualization helps users clearly see how each algorithm explores nodes and backtracks when necessary.

Step 3: Algorithm Design and Execution

Breadth-First Search (BFS)

- Uses a **queue** to explore nodes in increasing order of distance.
  - Checks all neighboring cells before moving to the next layer.
  - Ensures shortest path in an unweighted maze.
  - Time Complexity: **O(V + E)**

Depth-First Search (DFS)

- Uses a **stack** (or recursion) to explore as deeply as possible.
- Efficient in small mazes but may explore unnecessary branches.
- Does not guarantee the shortest path.
- Time Complexity: **O(V + E)**
-

A* Search Algorithm

- Uses a **priority queue** to select the next node based on:
- f(n) = g(n) + h(n)

where:

  - g(n) = actual cost from start to current node
  - h(n) = estimated cost to reach goal (Manhattan distance)
- Finds the **optimal path** efficiently with balanced exploration.
- Time Complexity: **O(E)** (depending on heuristic quality)

Step 4: Performance Measurement

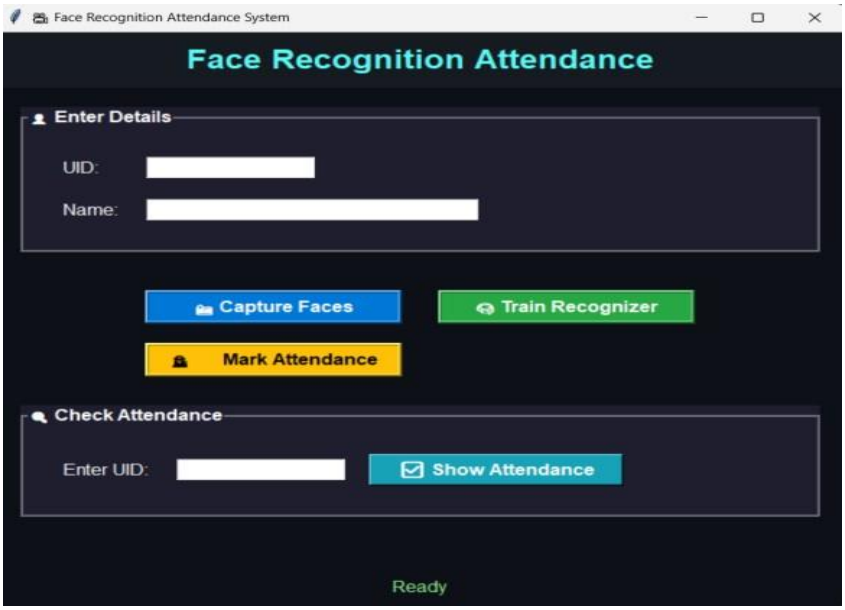
For each algorithm, the program records:

- **Path Length** (number of nodes in solution)
- **Nodes Explored**
- **Execution Time** (in seconds)
- **Efficiency Rating**

This data is used to generate a comparative analysis between BFS, DFS, and A*.

4. Screenshots/Result

I) First Page



ii) Show result

5. Conclusion

The Face Recognition Attendance System successfully demonstrates how Artificial Intelligence (AI) and Computer Vision can be applied to automate and simplify attendance management. By integrating technologies like Python, OpenCV, and Tkinter, the project provides a reliable, contactless, and efficient alternative to traditional manual or card-based attendance systems.

The system captures and recognizes faces in real time using the LBPH (Local Binary Patterns Histogram) algorithm and automatically records attendance details, thereby minimizing human intervention and errors. It ensures a secure, paperless, and user-friendly approach to managing attendance data. The GUI interface further enhances usability, allowing easy access to key features like capturing faces, training the model, marking attendance, and checking records.

This project has achieved its main objectives — improving accuracy, reducing manual workload, and eliminating proxy attendance. It can be deployed in educational institutions, offices, and organizations where attendance tracking is an essential daily activity.

In conclusion, the Face Recognition Attendance System represents a practical implementation of AI in everyday life, promoting digital transformation and operational efficiency. With further enhancements such as cloud integration, mobile access, and multi-camera support, this system can evolve into a robust, scalable solution for smart attendance management in the future.