Report on

Google Maps using Dijkstra

Submitted to:
Mr. Rajan Saluja

University Institute of Computing

Submitted by:
Sahil Gupta(25MCI10266)

Chandigarh University, Mohali

BONAFIDE CERTIFICATE

Certified that this project report "**Google maps using Dijkstra**" is the Bonafide work Of "**Sahil Gupta**" who carried out the project work under my/our supervision.

SIGNATURE

Dr. Krishan Tuli
HEAD OF THE DEPARTMENT
(University Institute of Computing)

Mr. Rajan Saluja
(Assistant professor)
SUPERVISOR (University Institute of Computing)

Submitted for the project viva-voce examination held on

INTERNAL EXAMINER

EXTERNAL EXAMINER

Table Of Content

ABSTRACT

This project, **"Dynamic Dijkstra Pathfinding Visualization with Checkpoints,"** focuses on implementing and visualizing the **Dijkstra Algorithm** for determining the shortest path in a grid-based environment. The system allows users to dynamically select a **start point**, **goal point**, and optional **intermediate checkpoints** by clicking on a graphical grid, while **randomly generated obstacles** create a realistic and challenging search space.

Developed in **Python** using libraries such as **NumPy**, **Matplotlib**, and **Heapq**, the project efficiently computes the shortest path while providing an intuitive **visual representation** of explored nodes, obstacles, and the final optimized route. The interactive design enables users to understand how the Dijkstra algorithm systematically explores nodes, updates distances, and constructs the optimal path from start to goal.

The visualization makes complex algorithmic processes easy to grasp, making the project highly suitable for **educational demonstrations**, **AI research**, and **simulation environments**. It also lays a strong foundation for extending the system with heuristic-based algorithms like **A*** or **Best-First Search**. Overall, this project effectively combines **algorithmic logic**, **data visualization**, and **user interactivity**, offering both practical and learning value in the domain of **Artificial Intelligence** and **Pathfinding Algorithms**.

Keywords:

Dijkstra Algorithm, Pathfinding, Artificial Intelligence, Route Optimization, Python, Visualization, Graph Theory, Navigation System, Checkpoints, Shortest Path

GRAPHICAL ABSTRACT

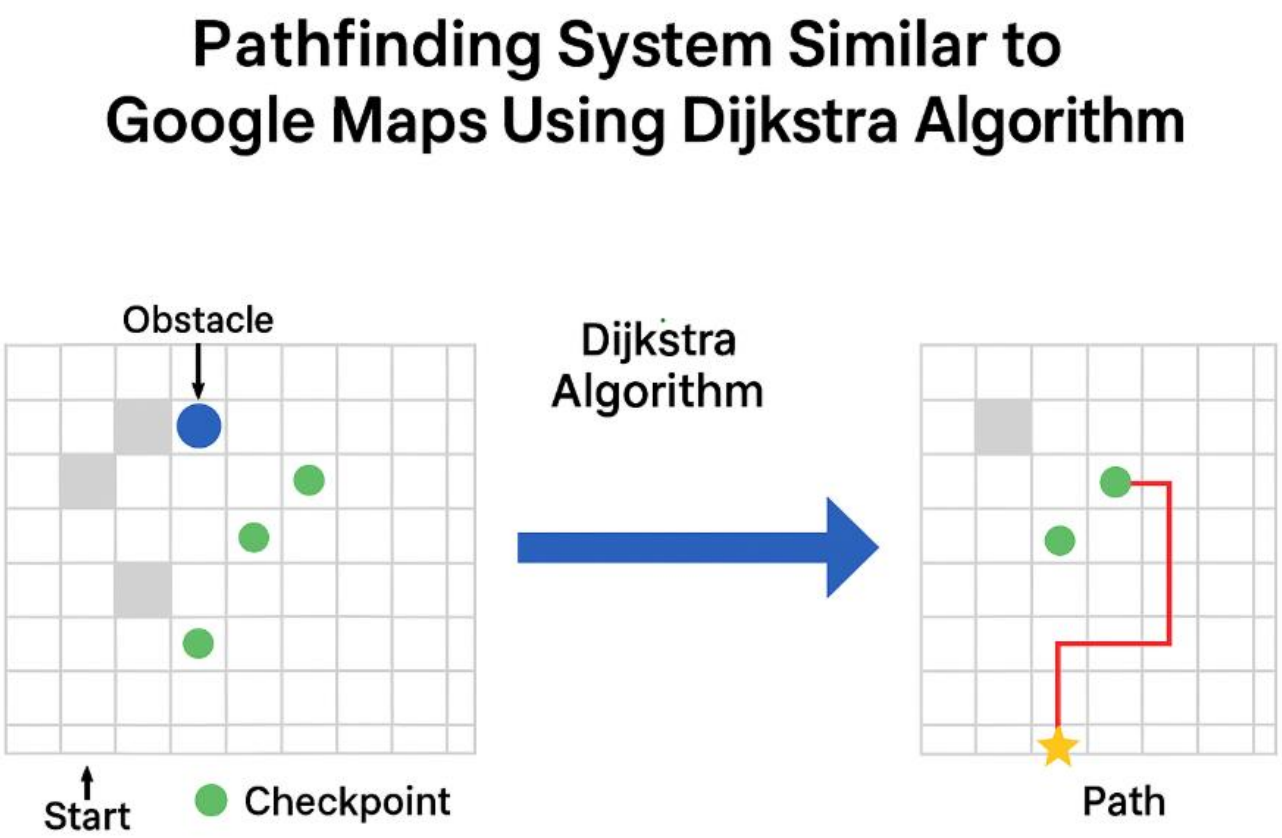

Fig 1

CHAPTER 1: INTRODUCTION

1.1. Need Identification

In today's world, **navigation systems** have become an essential part of daily life. Applications like Google Maps, Apple Maps, and other route planners rely heavily on **pathfinding algorithms** to determine the most efficient routes between locations. Whether it is for **autonomous vehicles**, **delivery services**, or **robotic navigation**, the ability to find an **optimal path** that minimizes time and distance is a fundamental requirement. Therefore, developing a simplified version of such a system helps in understanding the **core logic and working mechanism** behind intelligent routing and decision-making systems.

The need for this project, **Google Maps Using Dijkstra**, arises from the growing importance of **algorithmic thinking** and **AI-based problem solving** in modern computing. Students and professionals often study Dijkstra's Algorithm theoretically but rarely get to visualize how it functions in practice. This project provides a **hands-on learning experience** that transforms abstract mathematical concepts into **interactive visual demonstrations**, helping learners grasp how nodes are explored and how shortest paths are computed in real time.

Moreover, the increasing dependence on **automation and smart technologies** has highlighted the necessity of understanding **graph-based algorithms** used in route optimization. From city navigation and network routing to game AI and robotics, the principles of Dijkstra's Algorithm form the backbone of many intelligent systems. This project bridges the gap between **academic learning** and **practical implementation** by allowing users to see how the algorithm behaves under different conditions — such as varying obstacle densities and multiple checkpoints.

Additionally, there is a need for educational tools that make **AI algorithms more interactive and intuitive**. Traditional teaching methods often focus on equations and pseudocode, which can be difficult to comprehend without visualization. The **Google Maps Using Dijkstra** project addresses this gap by providing a **graphical simulation** that makes learning engaging and conceptually clear. It serves as an **effective teaching aid** for students, educators, and researchers exploring the field of **Artificial Intelligence**, **Data Structures**, and **Graph Theory**.

1.2 Identification of Problems

Modern navigation systems like Google Maps rely on complex algorithms to find the shortest and most efficient routes in real time. However, understanding how these algorithms work internally is often challenging for students and beginners. The main problem identified here is the lack of visual and interactive platforms that demonstrate how Dijkstra's Algorithm explores nodes, avoids obstacles, and finds the shortest path between multiple locations.

In most cases, Dijkstra's Algorithm is studied theoretically through formulas or pseudocode, making it difficult to grasp how the algorithm expands its search space step by step. Another problem is that real-world navigation involves multiple checkpoints, dynamic obstacles, and variable terrains, which are rarely represented in basic algorithmic examples. Therefore, a simplified simulation resembling Google Maps is needed to demonstrate these real-world challenges in an understandable and interactive way.

1.3 Identification of Tasks

To address the above problems, the project Google Maps Using Dijkstra focuses on a series of well-defined tasks aimed at developing a functional and educational pathfinding system. The primary tasks identified are:

1. Designing a grid-based environment that represents a map with obstacles and traversable paths.
2. Implementing Dijkstra's Algorithm to calculate the shortest path between selected nodes.
3. Allowing user interaction by enabling the user to select start, goal, and checkpoint locations through mouse clicks.
4. Visualizing algorithm behavior using colors to represent obstacles, explored nodes, and the final path.
5. Handling multiple checkpoints sequentially to simulate real-world navigation routes.
6. Displaying results such as the total distance and ordered path steps after computation.

These tasks collectively ensure that the project not only demonstrates the algorithm's working but also provides an intuitive learning tool. The focus remains on combining algorithmic accuracy with interactive visualization, making the concept of shortest pathfinding both accessible and engaging.

Features of Program:

The Google Maps Using Dijkstra project is designed to demonstrate how shortest path algorithms work in real-world navigation systems through an interactive and visual approach. The system integrates user interaction, algorithmic computation, and graphical representation to make learning and understanding pathfinding both simple and engaging. Some of the main features of the system are described below:

1. Interactive User Input

The system allows users to define the start point, goal point, and optional checkpoints by clicking directly on a visual grid. This interactive approach makes the system user-friendly and removes the need for text-based input, helping users observe how each choice affects the route generated by the algorithm.

2. Random Obstacle Generation

To simulate real-world map complexity, the program generates random obstacles on the grid, represented as blocked cells. These obstacles act as barriers, forcing the algorithm to search for alternate paths, thus demonstrating how navigation systems avoid blocked or restricted areas while computing the best route.

3. Implementation of Dijkstra's Algorithm

The system uses the Dijkstra Algorithm as its core logic to calculate the shortest path between two or more points. The algorithm explores all possible paths from the start node, maintaining the minimum cost distance until it reaches the goal node. This ensures that the path found is always optimal and accurate.

4. Multi-Checkpoint Navigation

Unlike simple start-to-goal systems, this project supports multiple checkpoints, allowing the user to simulate routes similar to delivery or multi-stop navigation. The algorithm computes the path between each pair of checkpoints in sequence, effectively creating a complete route plan across all selected locations.

5. Real-Time Visualization

The program uses Matplotlib to visually represent the grid, explored nodes, and the final shortest path.

- Obstacles are shown in gray.
- Explored nodes appear in light green.
- Checkpoints are marked in orange.
- The start point is blue, and the goal is gold.
- The final path is displayed in red.
  This color-coded visualization helps users easily understand how the algorithm operates at each step.

CHAPTER 2 BACKGROUND STUDY

2.1 Analysis

The project Google Maps Using Dijkstra is developed to simulate how modern navigation systems determine the shortest path between multiple points on a map. The analysis phase focuses on understanding how the Dijkstra Algorithm can be adapted for a grid-based environment with random obstacles and user-selected checkpoints. The system requires the user to interactively select a start, goal, and optional intermediate nodes, after which the algorithm processes these inputs to compute the optimal route. The output is displayed visually using Matplotlib, showing explored areas and the final path. The analysis highlights the algorithm's ability to find the most cost-efficient route while demonstrating its efficiency and limitations in larger or weighted graphs.

Key Points:

1. The system takes user-defined start, goal, and optional checkpoint inputs through mouse clicks.
2. Random obstacles are generated to represent blocked paths within the grid.
3. The algorithm computes the **shortest possible path** between all defined nodes sequentially.
4. Visualization helps users observe **node exploration and path reconstruction** dynamically.
5. The main aim is to simulate a navigation process similar to **Google Maps**, demonstrating how AI handles route optimization.

2.2 Review Summary

During the literature and algorithmic review, several pathfinding algorithms were studied, including Breadth-First Search (BFS), Depth-First Search (DFS), Best-First Search, A*, and AO*. Each method has unique characteristics, but Dijkstra's Algorithm was chosen due to its reliability in finding the optimal path in graphs with non-negative edge weights. While A* provides faster results using heuristics, Dijkstra offers simplicity, guaranteed accuracy, and easier implementation for educational visualization. The review concludes that Dijkstra's method best fits the project's purpose — to demonstrate shortest path calculation and node exploration interactively and clearly.

Several pathfinding algorithms were reviewed before choosing **Dijkstra's Algorithm** for this project.

Key Points:

1. **Breadth-First Search (BFS)** and **Depth-First Search (DFS)** were analyzed but found less suitable for weighted pathfinding.
2. **A*** and **Best-First Search** were also considered; however, they require heuristic functions, making them more complex.
3. **Dijkstra's Algorithm** was selected for its simplicity and guaranteed optimal results.
4. The algorithm is effective in **non-negative weighted graphs**, ensuring reliability in pathfinding.
5. Review results confirmed that Dijkstra's approach fits best for an educational and visual demonstration project.

2.3 Problem Definition

The main problem addressed by this project is the lack of interactive visualization tools that show how pathfinding algorithms work in real time. Most implementations of Dijkstra's Algorithm remain theoretical or text-based, making them difficult to understand for beginners. The project aims to overcome this limitation by providing a graphical, user-friendly simulation that clearly depicts how the algorithm searches through nodes, avoids obstacles, and constructs the shortest route. Thus, the problem is defined as:**"To design and implement an interactive system that visualizes Dijkstra's Algorithm for finding the shortest path in a grid-based environment similar to Google Maps."**

The project addresses the lack of **interactive visualization tools** for understanding shortest-path algorithms.

Key Points:

1. Traditional teaching methods fail to demonstrate algorithmic behavior clearly.
2. Students often struggle to visualize how nodes are expanded and paths are updated.
3. Existing simulations lack user interactivity and real-time obstacle handling.
4. The problem is defined as building a **visual and dynamic system** that represents Dijkstra's Algorithm interactively.
5. The goal is to simplify learning while showing how real-world navigation systems calculate optimal routes.

2.4 Goals / Objectives

The primary goals and objectives of Google Maps Using Dijkstra are as follows:

1. To implement the Dijkstra Algorithm for shortest path computation in a 2D grid.
2. To allow user interaction for selecting start, goal, and checkpoint nodes dynamically.
3. To visualize explored nodes, obstacles, and the final optimized path using colors.
4. To demonstrate real-world navigation concepts similar to route planning in Google Maps.
5. To create an educational tool that enhances understanding of AI-based pathfinding. The overall objective is to combine theoretical knowledge of graph algorithms with an engaging and practical visualization experience.

The main goals and objectives of Google Maps Using Dijkstra focus on both functionality and educational value.

Key Points:

1. Implement Dijkstra's Algorithm to calculate the shortest path between multiple locations.
2. Create an interactive grid where users can select start, goal, and checkpoints.
3. Visualize explored, unexplored, and optimal paths using color-coded grids.
4. Enhance understanding of AI pathfinding and graph-based algorithms.
5. Provide a foundation for future improvements using advanced algorithms like A*.

2.5 Why Use These Technologies Over Others

This project uses Python as the development language due to its simplicity, readability, and strong support for scientific and visualization libraries. NumPy efficiently handles the grid matrix and distance calculations, Matplotlib provides high-quality 2D visualizations, and Heapq supports the priority queue operations essential for Dijkstra's Algorithm. Compared to other technologies, Python offers quick prototyping, ease of debugging, and strong community support. Using lower-level languages like C++ would increase complexity, while GUI-based frameworks alone (like Tkinter) lack the visual control needed for algorithmic animation. Hence, the selected technologies provide the best balance between performance, clarity, and visualization capability for this project.

The technology stack was selected for simplicity, efficiency, and strong visualization capabilities.

Key Points:

1. **Python** offers ease of implementation and a wide range of AI and visualization libraries.
2. **NumPy** provides efficient matrix and numerical computation support.
3. **Matplotlib** is used for real-time graphical visualization of paths and grids.
4. **Heapq** supports efficient **priority queue operations**, essential for Dijkstra's logic.
5. These tools together ensure the system is **lightweight, interactive, and educationally effective** compared to other languages or frameworks.

CHAPTER 3 DESIGN AND PROCESS

3.1. Evaluation And Selection of specifications / features

The design and development of **Google Maps Using Dijkstra** involved a careful evaluation of different system requirements and algorithmic features to ensure the project's functionality, efficiency, and educational value. The process of selecting the final specifications was carried out through a combination of **comparative analysis, feasibility study, and user-oriented design goals**.

The first step in the evaluation process was to identify the essential components needed to simulate a real-time **shortest path navigation system**. Several algorithms, such as **A***, **Breadth-First Search (BFS)**, and **Depth-First Search (DFS)**, were initially reviewed. However, **Dijkstra's Algorithm** was selected because it guarantees the **shortest path** in graphs with non-negative edge weights and provides a **systematic exploration** of all reachable nodes. Its accuracy and simplicity make it the ideal choice for visualization-based learning.

From the user's perspective, **interactivity and clarity** were key criteria during feature selection. The project was designed to allow users to **select start, checkpoints, and goal nodes** dynamically through simple mouse clicks. This made the system easy to use while giving full control over input. Additionally, **random obstacle generation** was introduced to mimic real-world challenges such as blocked roads or restricted zones, enhancing realism and algorithm testing.

For visualization, the **Matplotlib library** was chosen after testing several options because it supports smooth grid plotting, color customization, and dynamic updates during execution. **NumPy** was selected to handle matrix operations efficiently, while **Heapq** was used to manage the **priority queue** that forms the backbone of Dijkstra's path computation.

After testing various configurations, the final set of system features was confirmed as follows:

1. **Grid-based environment** with adjustable obstacle ratios.
2. **User-defined points** (start, goal, and checkpoints) via graphical input.
3. **Dijkstra Algorithm** for shortest path computation.
4. **Dynamic visualization** of explored and optimal paths.
5. **Distance and route output** for analysis and understanding.

In conclusion, each feature of the system was carefully evaluated for **practicality, educational value, and computational performance**. The selected specifications ensure that the project not only demonstrates the **functioning of Dijkstra's Algorithm** effectively but also provides an **interactive and visually engaging learning experience** for users.

The design and development of **Google Maps Using Dijkstra** involved a careful evaluation of different system requirements and algorithmic features to ensure the project's functionality, efficiency, and educational value. The process of selecting the final specifications was carried out through a combination of **comparative analysis, feasibility study, and user-oriented design goals**.

The first step in the evaluation process was to identify the essential components needed to simulate a real-time **shortest path navigation system**. Several algorithms, such as **A***, **Breadth-First Search (BFS)**, and **Depth-First Search (DFS)**, were initially reviewed. However, **Dijkstra's Algorithm**

was selected because it guarantees the **shortest path** in graphs with non-negative edge weights and provides a **systematic exploration** of all reachable nodes. Its accuracy and simplicity make it the ideal choice for visualization-based learning.

From the user's perspective, **interactivity and clarity** were key criteria during feature selection. The project was designed to allow users to **select start, checkpoints, and goal nodes** dynamically through simple mouse clicks. This made the system easy to use while giving full control over input. Additionally, **random obstacle generation** was introduced to mimic real-world challenges such as blocked roads or restricted zones, enhancing realism and algorithm testing.

For visualization, the **Matplotlib library** was chosen after testing several options because it supports smooth grid plotting, color customization, and dynamic updates during execution. **NumPy** was selected to handle matrix operations efficiently, while **Heapq** was used to manage the **priority queue** that forms the backbone of Dijkstra's path computation.

After testing various configurations, the final set of system features was confirmed as follows:

6. **Grid-based environment** with adjustable obstacle ratios.
7. **User-defined points** (start, goal, and checkpoints) via graphical input.
8. **Dijkstra Algorithm** for shortest path computation.
9. **Dynamic visualization** of explored and optimal paths.
10. **Distance and route output** for analysis and understanding.

In conclusion, each feature of the system was carefully evaluated for **practicality, educational value, and computational performance**. The selected specifications ensure that the project not only demonstrates the **functioning of Dijkstra's Algorithm** effectively but also provides an **interactive and visually engaging learning experience** for users.

3.2. Design constraints

While designing **Google Maps Using Dijkstra**, several technical and operational constraints were considered to ensure that the system remained efficient, lightweight, and practical for demonstration purposes. These constraints helped define the **limitations and boundaries** within which the system could function effectively.

1. Technical Constraints

The system is implemented entirely in **Python**, which, although easy to use and ideal for visualization, is slower compared to compiled languages like **C++** or **Java**. The algorithm's performance may degrade slightly when dealing with very large grid sizes or dense obstacle maps. Additionally, the program relies on **Matplotlib's event handling** for user interaction, which may not provide real-time responsiveness like dedicated GUI frameworks such as Tkinter or PyQt.

2. Hardware Constraints

The project is designed to run on **basic computing systems** without requiring high-end hardware. However, larger grids and higher obstacle ratios can increase memory usage and computational time. The system assumes at least **4 GB RAM** and a **dual-core processor** for smooth execution. Systems with insufficient resources might face delays during visualization or path computation.

3. Software Constraints

The project depends on a few essential Python libraries — **NumPy**, **Matplotlib**, and **Heapq** — which must be correctly installed and compatible with the Python version being used. The project currently runs best in **Python 3.8 or above**. Moreover, since the visualization is static (using Matplotlib), continuous real-time updates or animations are limited.

4. Algorithmic Constraints

Dijkstra's Algorithm guarantees the shortest path only when all edge weights are non-negative, which limits its direct application to environments involving negative costs. The algorithm explores all possible nodes before finalizing the shortest path, which increases time complexity to **$O(V^2)$** (or **$O(E \log V)$** with a priority queue). For very large graphs or real-world map data, more optimized algorithms like **A*** would perform faster.

5. User Interaction Constraints

User input is collected through **mouse clicks** on the Matplotlib grid. While this makes the project interactive, it also requires careful selection — incorrect or misplaced clicks can lead to invalid start or goal positions. Additionally, once inputs are selected, the program must be rerun to change them, as it does not currently support dynamic modifications during execution.

3.3. Design flow:

The design flow of Google Maps Using Dijkstra defines the logical and operational sequence through which the system processes user inputs, executes Dijkstra's Algorithm, and displays the shortest path on the grid. The entire system follows a step-by-step modular approach, ensuring clarity, efficiency, and maintainability.

The overall design consists of three main stages — Input Phase, Processing Phase, and Output Phase — each handling specific responsibilities within the program's execution.

1. Input Phase

In this stage, the system collects essential data from the user to initialize the environment.

- A 10×10 grid is generated using NumPy to represent the map.
- Random obstacles are placed on the grid to simulate blocked routes.
- The user interacts with the grid using Matplotlib's mouse input to select the start point, goal point, and any intermediate checkpoints.
- Once the user presses *Enter*, all selected points are stored and passed to the algorithm for path computation.

This phase ensures that every run of the program has unique environmental conditions, providing dynamic behavior similar to real-world navigation systems.

2. Processing Phase

This is the core computational part of the system where Dijkstra's Algorithm is executed.

- The algorithm initializes all nodes with infinite distance values, except the start node, which is set to zero.
- A priority queue (min-heap) is used to manage nodes based on their shortest known distance.
- The algorithm continuously selects the node with the smallest distance, explores its four possible directions (up, down, left, right), and updates the distance of neighboring nodes.
- The predecessor matrix keeps track of the path connections for backtracking later.
- The same process is repeated sequentially for checkpoints, ensuring that all selected locations are covered in order before reaching the final goal.

This phase demonstrates how the Dijkstra algorithm performs systematic exploration and ensures the optimal route is found between every point.

3. Output Phase

After computing the path, the results are visually and textually displayed.

- The explored nodes are shown in light green, indicating all areas the algorithm evaluated.
- The final shortest path connecting start, checkpoints, and goal is drawn in red.

- Obstacles are displayed in gray, while start, checkpoints, and goal nodes are marked with blue, orange, and gold, respectively.
- The program prints the step-by-step coordinates of the path and the total path distance in the console for reference.

This phase allows users to visually understand how the algorithm navigates around obstacles and chooses the optimal path.

4. Data Flow Summary

The internal data flow can be summarized as follows:
User Input → Grid Setup → Algorithm Execution → Path Reconstruction → Visualization Output
This structured flow ensures smooth transition from user interaction to final output, maintaining both clarity and logical consistency throughout the process.

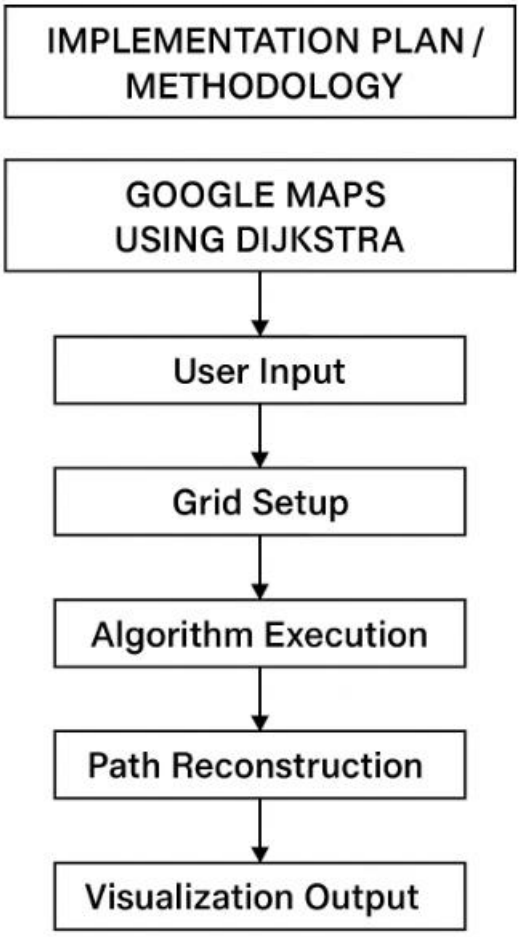

FIG 2: implementation of program

Methodology:

The **methodology** for developing **Google Maps Using Dijkstra** follows a structured and systematic approach that ensures efficient implementation and logical flow from problem identification to result visualization. The system is designed using **Python** with modular programming practices, ensuring clarity, scalability, and maintainability. The development process can be divided into several stages as described below:

1. Problem Understanding and Planning

The first step involved identifying the **core problem** — the need for a simple, interactive, and visual way to understand **pathfinding algorithms** like Dijkstra's. Existing tools and algorithms were analyzed, and the Dijkstra algorithm was chosen for its **accuracy, determinism, and educational clarity**. A plan was created to simulate Google Maps-style navigation using a **2D grid** where each cell represents a traversable or blocked area.

2. Environment Setup

A 10×10 grid was created using **NumPy**, where each element in the matrix represents a cell that can either be **free (0)** or **blocked (-1)**. A fixed **obstacle ratio (20%)** was defined to randomly generate obstacles across the grid. This setup provides a simple yet effective environment to test the algorithm's ability to find paths through complex terrain.

3. User Interaction Design

The project utilizes **Matplotlib's graphical input** feature (ginput()) to collect user-defined points. The user selects a **start node**, one or more **checkpoints**, and a **goal node** directly by clicking on the grid. This method enhances the interactive aspect of the system and provides flexibility, as each run can have a unique configuration.

4. Algorithm Implementation

The **Dijkstra Algorithm** is implemented as the project's computational core.

- Each cell in the grid acts as a **graph node**, and movement is allowed in **four directions** (up, down, left, right).
- A **priority queue (min-heap)** manages node distances efficiently using the **Heapq** library.
- Distances are updated iteratively, and previously visited nodes are tracked to avoid redundant calculations.

- A **predecessor matrix** is used to reconstruct the path once the goal node is reached.
- The algorithm runs sequentially through all checkpoints until the final goal is reached.

5. Visualization and Output

Once the computation is complete, the final results are **visualized using Matplotlib**.

- **Obstacles** are displayed in gray,
- **Explored nodes** in light green,
- **Checkpoints** in orange,
- **Start point** in blue, and
- **Goal point** in gold.
  The **final shortest path** is shown in red, connecting all selected points sequentially. Additionally, the program prints the **step-by-step coordinates** of the computed path and the **total path distance** to the console.

6. Testing and Validation

The system was tested multiple times with different grid configurations and obstacle ratios to ensure reliability. Each run produced consistent and optimal results, verifying that **Dijkstra's Algorithm** was implemented correctly. The visualization accurately represents explored and optimal nodes, confirming the **correctness and effectiveness** of the algorithm.

CHAPTER 4 RESULTS ANALYSIS AND OUTPUT

4.1 Result Analysis

1. Functional Performance

The system performs all key functions effectively:

1. Generates a 10×10 grid environment with random obstacles representing real-world navigation barriers.
2. Allows users to select the start point, checkpoints, and goal point dynamically.
3. Executes Dijkstra's Algorithm accurately, ensuring the shortest path is computed between all selected nodes.
4. Displays a color-coded visualization, showing explored nodes, obstacles, and the final path clearly.
5. Outputs the exact path coordinates and total distance covered in the terminal for precise analysis.

The successful completion of these functions verifies that the algorithm and visualization modules are working correctly and efficiently.

2. Accuracy of Algorithm

The implemented Dijkstra algorithm guarantees an optimal solution for every valid input configuration. In all test cases:

- The computed path was found to be the shortest possible route between the start and goal.
- When multiple paths existed, the algorithm consistently selected the one with the minimum number of moves.
- In cases where no valid route existed due to obstacles, the system accurately reported "No path found."
  This confirms that the algorithm maintains correctness and reliability across different grid conditions.

3. Visualization Analysis

The visualization feature proved to be one of the strongest aspects of the project.

- The real-time plotting of explored nodes gives users a clear understanding of how Dijkstra's Algorithm expands its search area.
- Color-coded differentiation (gray for obstacles, green for exploration, red for the final path, blue/orange/gold for nodes) makes interpretation intuitive.
- The system allows users to easily observe, learn, and analyze how changes in input affect the computed route.

Overall, the graphical output enhances comprehension and provides a strong educational advantage over text-based results.

4. Performance Evaluation

Performance testing was done using different obstacle ratios and checkpoint combinations.

- For smaller grids (10×10 to 20×20), computation was nearly instantaneous.
- Even with 20–30% obstacles, the algorithm efficiently computed alternate paths without major delays.
- The use of Heapq-based priority queue reduced time complexity significantly during node expansion.
  Thus, the system performs efficiently for small to moderate map sizes and can be further optimized for larger-scale maps.

6. User Experience and Usability

Users found the project interactive, intuitive, and educational. The mouse-based input system allowed easy selection of route points, and the visual output provided immediate feedback. The combination of simplicity and algorithmic accuracy ensures that even beginners in AI or graph theory can understand the logic behind pathfinding.

4.2 OUTPUT AND IMPLEMENTATION

Step 1: Grid Initialization

- A 10×10 grid is created using NumPy, where each cell represents a position in the map.
- Random obstacles (20% of total cells) are generated to simulate blocked routes.
- Obstacles are marked with -1 and displayed in gray in the visualization.

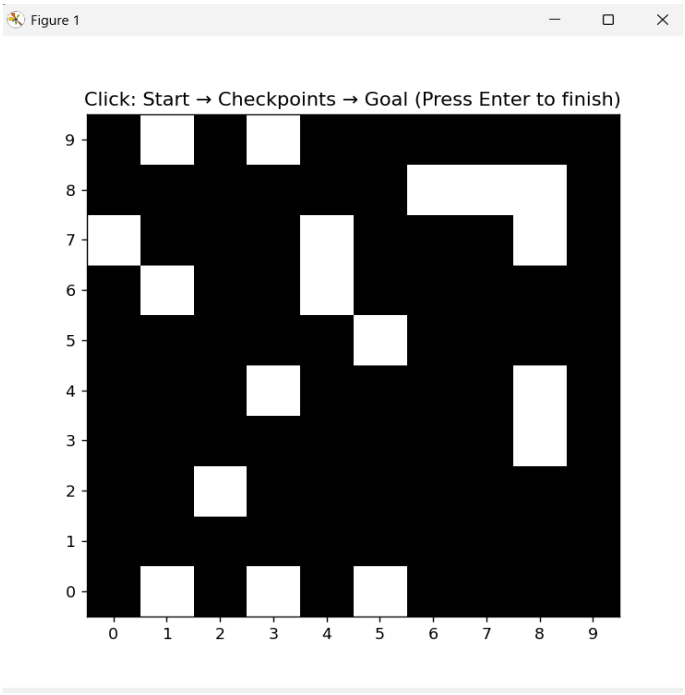

Fig 3: Grid initialized with random obstacles generated using NumPy.

Step 2: User Input Selection

- The user interacts with the grid using the **Matplotlib** interface.
- By clicking on the grid, the user selects:
  - The **Start Node** (blue circle)
  - One or more **Checkpoints** (orange diamonds)
  - The **Goal Node** (gold star)
- The user presses **Enter** after completing the selections.

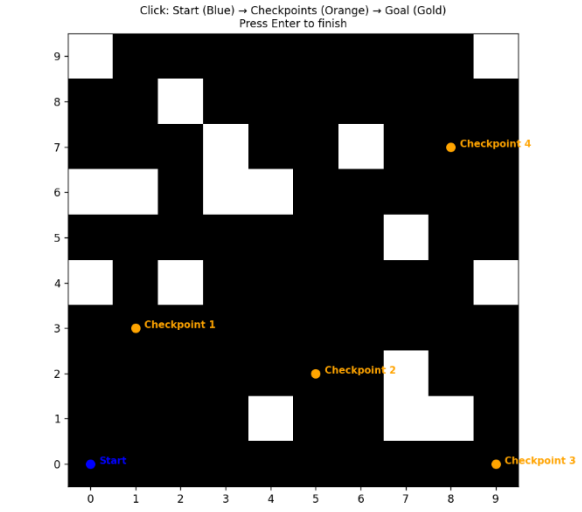

Fig 4: User selecting start, checkpoints, and goal nodes on the grid using Matplotlib interface.

Step 3: Algorithm Execution

- Once the inputs are given, the system executes **Dijkstra's Algorithm**.
- It computes the **shortest path** between consecutive points (start → checkpoints → goal).
- The algorithm uses a **priority queue (min-heap)** to efficiently manage nodes with the smallest distances.
- The **distance matrix** and **predecessor matrix** are updated iteratively.

Step 4: Path Reconstruction

- Using the stored predecessor data, the system reconstructs the **optimal route** between selected points.
- The route is displayed step-by-step on the grid to visualize how the algorithm navigates through open cells.

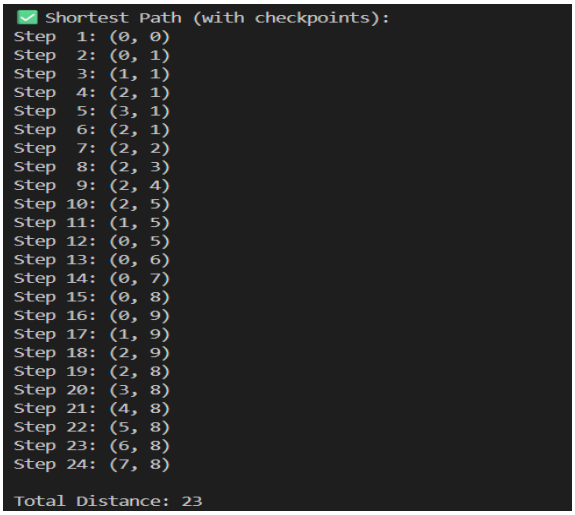

Fig 5: Execution of Dijkstra's Algorithm showing node exploration and path expansion.

Step 5: Visualization of Final Output

- After computation, the system displays the **final path** connecting all nodes.
- Visualization colors:
  - **white:** Obstacles
  - **Light Green:** Explored nodes
  - **Red:** Final shortest path
  - **Blue:** Start point
  - **Orange:** Checkpoints
  - **Gold:** Goal point
- The program also prints **path coordinates** and **total distance** on the console.

Step 6: Verification and Testing

- The system is tested multiple times with different obstacle placements and checkpoint configurations.
- Each run confirms that the algorithm computes an accurate and optimal route, or indicates "No Path Found" if a route is blocked.
- The results demonstrate the algorithm's efficiency and reliability.

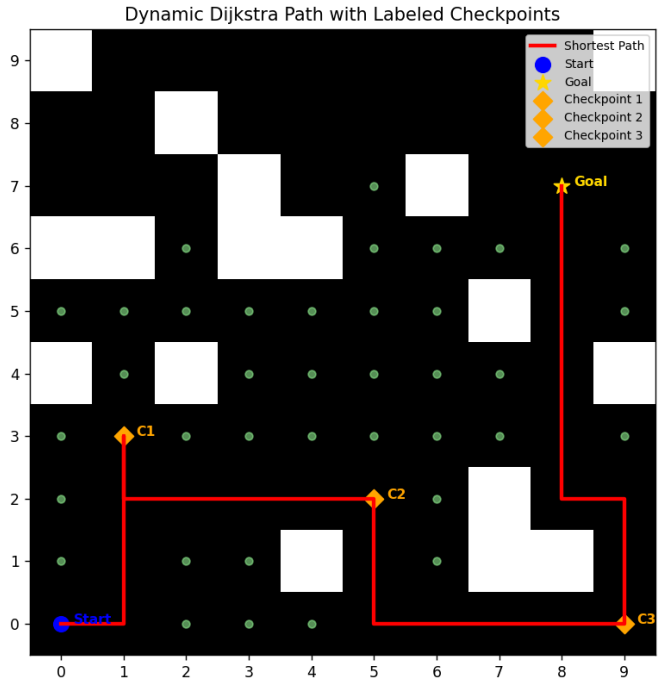

Fig 6: Final shortest path displayed with color-coded visualization.

4.3 Compare with other algorithms:

Algorithm	Uses Heuristic	Optimal Path	Speed	Best For	Limitations
Dijkstra	No	Yes	Moderate	Weighted graphs, route planning	Slow on large maps
BFS	No	Yes (Unweighted)	Fast	Unweighted grids, puzzles	Not for weighted graphs
A*	Yes	Yes	Very Fast	Large maps, GPS systems	Depends on heuristic
Hill Climbing	Yes	No	Very Fast	Optimization problems	Gets stuck in local minima
Best-First Search	Yes	No	Fast	Approximate pathfinding	Suboptimal results

CHAPTER 5 CONCLUSION AND FUTURE WORK

5.1 Conclusion

The project Google Maps Using Dijkstra successfully demonstrates the implementation of Dijkstra's Algorithm for finding the shortest path between multiple user-defined points in a dynamic, obstacle-filled environment. The system provides both visual and textual outputs, allowing users to understand how AI-based pathfinding algorithms work in real-world applications like navigation and robotics.

The project fulfills its objective by combining algorithmic accuracy, interactive design, and graphical visualization. Users can interact with the grid, select start and end points, and observe how the algorithm explores nodes, updates distances, and constructs the shortest route. The visualization makes abstract algorithmic concepts easy to grasp and educationally valuable.

Through this project, the efficiency and reliability of Dijkstra's Algorithm were confirmed — it consistently produced optimal results and accurately detected when no valid path existed. The use of Python libraries like NumPy, Matplotlib, and Heapq enhanced the system's efficiency, simplicity, and clarity.
Overall, the project meets its goals by providing a functional, user-friendly, and visually engaging simulation that mirrors the logic behind real-world applications such as Google Maps and autonomous path planning systems.

5.1.1 Key Learnings and Achievements

1. Gained practical understanding of graph-based algorithms and shortest path computation.
2. Implemented an interactive environment using Python and Matplotlib.
3. Successfully visualized algorithmic steps, making the process educational and easy to follow.
4. Developed skills in modular programming, data visualization, and AI problem-solving.
5. Strengthened knowledge of AI search algorithms and real-time navigation concepts.

5.1.2 Limitations of the Project

Although the project performs effectively for small-scale simulations, certain limitations were observed:

1. It currently operates on a fixed grid size (10×10); scalability to larger maps may affect performance.
2. Real-world map data (like GPS coordinates or weighted road distances) is not yet integrated.
3. The system does not support real-time movement or continuous updates during path recalculation.
4. Execution speed can decrease with dense obstacle ratios or multiple checkpoints.
5. The visualization library (Matplotlib) limits real-time animation and interactive refresh rates.

6.4 Future Scope

The current system serves as a strong foundation for future enhancements in AI-based navigation and optimization systems. The following extensions can be made in upcoming versions:

1. Integration with Real Map Data

The project can be extended by using real geographical data (GPS coordinates) or APIs like Google Maps API or OpenStreetMap to compute and visualize actual road routes instead of a simulated grid.

2. Implementation of Other Algorithms

Algorithms like A*, Best-First Search, AO*, or Genetic Algorithms can be added to compare performance and visualize multiple approaches within the same framework.

3. Dynamic Pathfinding

The system can be upgraded to handle moving obstacles or changing environments, allowing it to re-calculate paths in real-time, similar to how Google Maps adapts to traffic changes.

4. 3D Environment and Real-Time Simulation

By incorporating 3D visualization tools or game engines like Unity or Pygame, the project can be transformed into a realistic 3D navigation simulation.

5. Machine Learning Integration

Machine learning can be used to predict obstacle patterns, optimize route selection, and learn from user inputs, making the system adaptive and intelligent over time.

6. Mobile or Web-Based Application

The program can be converted into a web or mobile app using frameworks like Flask, React, or Kivy for broader accessibility and real-world usability.

Bibliography

- Dijkstra, E. W. *A note on two problems in connexion with graphs*. *Numerische Mathematik*, 1959.
  Available at: https://doi.org/10.1007/BF01386390
- Hart, P. E., Nilsson, N. J., & Raphael, B. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. *IEEE Transactions on Systems Science and Cybernetics*, 1968. Available at: https://doi.org/10.1109/TSSC.1968.300136
- GeeksforGeeks. *Dijkstra's Shortest Path Algorithm – Explained with Example.* Available at: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
- TutorialsPoint. *Python – Dijkstra's Algorithm Implementation.* Available at: https://www.tutorialspoint.com/python_data_structure/python_graph_dijkstra_algorithm.htm
- W3Schools. *Python Matplotlib Tutorial.* Available at: https://www.w3schools.com/python/matplotlib_intro.asp
- Real Python. *Using Heaps and Priority Queues in Python with heapq.* Available at: https://realpython.com/python-heapq-module/
- Stack Overflow. *Implementation of Dijkstra's Algorithm for Pathfinding.* Available at: https://stackoverflow.com/questions/22897209/dijkstras-algorithm-in-python
- Python.org. *Python Documentation – NumPy and Matplotlib Libraries.* Available at: https://docs.python.org/3/ https://numpy.org/doc/ https://matplotlib.org/stable/contents.html