PATH FINDING VISUALIZER

A PROJECT REPORT
for
Mini Project-I (K24MCA18P)
Session (2024-25)

Submitted by

Nikita Agarwal
(202410116100133)
Poornima
(202410116100142)
Priyanshi
(202410116100153)

Submitted in partial fulfillment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Ms. Divya Singhal
Assistant Professor

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206

(DECEMBER- 2024)

CERTIFICATE

Certified that **Nikita Agarwal 202410116100133, Poornima 202410116100142, Priyanshi 202410116100153** have carried out the project work having "**Path Finding Visualizer**" (**Mini Project-I, K24MCA18P**) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Ms. Divya Singhal
Assistant Professor
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Dr. Arun Kr. Tripathi
Dean
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Path Finder Visualizer

Nikita Agarwal
Poornima
Priyanshi

ABSTRACT

The Pathfinder Visualizer is an interactive tool that showcases the functionality and efficiency of various pathfinding algorithms through real-time visual demonstrations. Pathfinding algorithms are crucial in computer science, robotics, and game development, making this project a valuable resource for understanding their behavior and applications. The visualizer allows users to explore popular algorithms like Dijkstra's Algorithm, A* Search, and Greedy Best-First Search. By providing a customizable grid interface, users can define start and end points, place obstacles, and observe how each algorithm navigates the search space to find the shortest path. The tool offers adjustable parameters and performance metrics, such as execution time and path cost, enabling users to compare the strengths and limitations of each algorithm.This project prioritizes usability and engagement, making it suitable for students, educators, and developers seeking a deeper understanding of graph traversal techniques. It serves as both an educational platform and a testing environment, encouraging experimentation and practical learning.

Keywords:Pathfinding Algorithms ,Educational Tool ,Shortest Path ,Real-time Visualization

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, Ms. Divya Singhal for her guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to Dr. Arun Kumar Tripathi, Professor and Dean, Department of Computer Applications, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Nikita Agarwal

Poornima

Priyanshi

TABLE OF CONTENTS

Chapter 1

Introduction

1.1 Project description

Plotting is known as pathfinding or pathing. A computer program, particularly brief path connecting two locations. Being more a useful maze-solving variation. This branch of study is primarily centered on Dijkstra's formula for determining the on a weighted graph, the shortest path that best meets some criteria.

A pathfinding approach fundamentally searches. Beginning at one vertex in a graph, looking into nearby nodes until the reaches the destination node, typically with the goal of locating the least expensive path. Although graph search techniques like if a route is needed, a breadth-first search would be used. If given enough time, alternative techniques “Explore" the graph, which would aim to the finish line earlier. An example would rather than a person moving across a space than considering all alternatives, The person would typically walk forward just traveling in the destination's direction veer off the path to avoid an impediment, and downplay deviations the maximum area (shortest, cheapest, fastest, etc.) between two points in a large network.

Through visual representations and animations, it helps users understand the inner workings of algorithms like Dijkstra's, A*, and more, making it an invaluable educational and problem-solving resource in various domains.

Pathfinding algorithms systematically explore nodes, starting at the source and evaluating nearby nodes, until reaching the destination. The goal is to minimize the cost, which could represent time, distance, or another factor. Techniques like A* prioritize paths that seem to lead toward the destination, while Dijkstra’s algorithm evaluates all paths exhaustively.

1.2 Project Scope

The Pathfinding Visualization Project aims to create an interactive tool showcasing the behavior and efficiency of various pathfinding algorithms. It supports Dijkstra's, A*, BFS, DFS, Greedy Best-First Search, and allows users to input start and end points on a grid-based map. Real-time visualization demonstrates algorithm exploration and shortest path determination. Grid cells represent obstacles, open and closed nodes, and the final path. The user interface enables algorithm selection, visualization speed adjustment, maze generation, grid reset, and customization.

Algorithms can be compared for path efficiency, time taken, and nodes explored. Key performance metrics include path length, nodes evaluated, and algorithm run time. Maze generation techniques include recursive division, Prim's Algorithm, randomized DFS, and basic random maze. The tool is responsive and user-friendly.

The project ensures compatibility across devices and screen sizes. Comprehensive documentation explains how to use the tool, pathfinding algorithms, and more. Real-world maps, advanced 3D rendering, and commercial deployment are not supported.

1.3 Hardware / Software used in Project

1.3.1 Hardware Requirement for development

- **Memory**: 4 GB RAM (minimum)
- **Processor**: Intel Core 2 Duo or higher
- **Graphics Card**: Not mandatory
- **Storage**: 3 GB free disk space
- **Operating System**: Windows 7 or above and MacOS

1.3.2 Software Requirement for development

- **Programming Language:** Python 3.7.6
- **Libraries/Modules:** Pygame Module that is used for creating the graphical interface and animations.
- **Code Editor:** Visual Studio Code (VS Code) or any other compatible code editor.
- **Development Environment:** Windows OS and MacOS
- **Additional Tools:** Python Package Installer (pip) to install dependencies.

1.4 FUNCTIONAL REQUIREMENTS

The functional requirements describe the core features and operations of the project:

1. **Algorithm Selection**:
    - The system must allow users to select various pathfinding algorithms such as:
        - **Dijkstra's Algorithm**
        - *A Search**
        - **Greedy Best-First Search**
2. **Grid Interaction**:
    - Users must be able to:
        - Define start and end points.
        - Place or remove obstacles on the grid.
        - Customize the grid size.
3. **Visualization**:
    - The system must visualize the step-by-step progress of the selected algorithm.
    - Real-time animations must display explored nodes, paths, and obstacles.
4. **Performance Metrics**:
    - The system must display metrics such as:
        - Path length
        - Number of nodes evaluated
        - Time taken to find the path
5. **Speed Control**:
    - The system must allow users to adjust the speed of the visualization:
        - Slow
        - Medium
        - Fast

6. **Reset and Clear**:
    - Users must be able to reset the grid and clear all obstacles or paths.
7. **User Interface Controls**:
    - Provide intuitive buttons or menu options for:
        - Starting, pausing, or resetting the visualization.
        - Selecting algorithms.

1.5 NON- FUNCTIONAL REQUIREMENTS

The non-functional requirements define the system's quality attributes:

- **Performance**: Smooth visualizations with minimal lag, even for large grids, and efficient algorithms for real-time results.
- **Usability**: Intuitive, easy-to-navigate user interface suitable for non-technical users.
- **Scalability**: Handles varying grid sizes and dynamic obstacles without performance degradation.
- **Compatibility**: Runs on Windows 7 and above and supports major Python versions (3.7 and above).
- **Reliability**: Consistently produces accurate and reliable results for all supported algorithms.
- **Portability**: Lightweight and runs smoothly on basic hardware.
- **Customization**: Users can customize grid size, speed, and obstacle placement.
- **Maintainability**: Modular code for easy updates and enhancements.
- **Real-time feedback**: Visualization process shows algorithm progress step by step.
- **Accessibility**: Minimal training required for users.

Chapter 2

Feasibility Study

2.1 Technical feasibility-

The project is technically feasible due to the following factors:

- **Programming Language**: Python is a suitable choice for implementing the project as it provides powerful libraries like *pygame* for real-time visualization.
- **Frameworks and Libraries**:
  - Pygame: Used for graphical interface and animations.
  - Python Standard Libraries: Provide efficient handling of algorithms, file management, and data visualization.
- **Hardware Requirements**:
  - The project requires minimal hardware resources (4GB RAM, Intel Core 2 Duo).
  - No dedicated GPU is needed, ensuring compatibility with basic systems.
- **Software Requirements**:
  - Python 3.7+ is lightweight and easily available.
  - VS Code or any IDE is compatible for development and debugging.
- **Platform Compatibility**: The project runs on Windows OS (7 or above), ensuring broad accessibility.

2.2 Operational Feasibility-

Operational feasibility focuses on how well the project meets user needs:

- **User Needs**:
  - The project provides an interactive and educational platform for understanding pathfinding algorithms.
  - It is tailored for students, educators, and developers looking to learn and compare algorithms visually.
- **Ease of Use**:

    - The user interface is intuitive, offering options like grid customization, algorithm selection, speed adjustment, and reset functionality.
    - No advanced technical knowledge is required to operate the application.
- **Customization**:
    - Users can customize the grid, add/remove obstacles, and modify start and end points, making the tool versatile for experimentation.

The project meets its intended purpose effectively and aligns with educational goals.

2.3 Economic Feasibility-

The project is cost-effective due to the following:

- **Free Tools and Technologies**:
    - Python and Pygame are open-source, reducing software expenses.
    - Development tools like VS Code are free to use.
- **Minimal Hardware Costs**: The system does not require advanced hardware, making it affordable to implement and run on existing systems.
- **No Licensing Costs**: The project focuses on educational and non-commercial purposes, eliminating licensing fees.
- **Maintenance Costs**: The modular design ensures low maintenance and easy scalability.

Overall, the project can be developed with negligible costs, making it economically feasible.

Chapter 3

Project Objective

The objective of pathfinding visualization is to visually represent the process of finding a path between two points on a map or grid. This technique is commonly used in computer science, gaming, and various applications involving navigation and optimization.

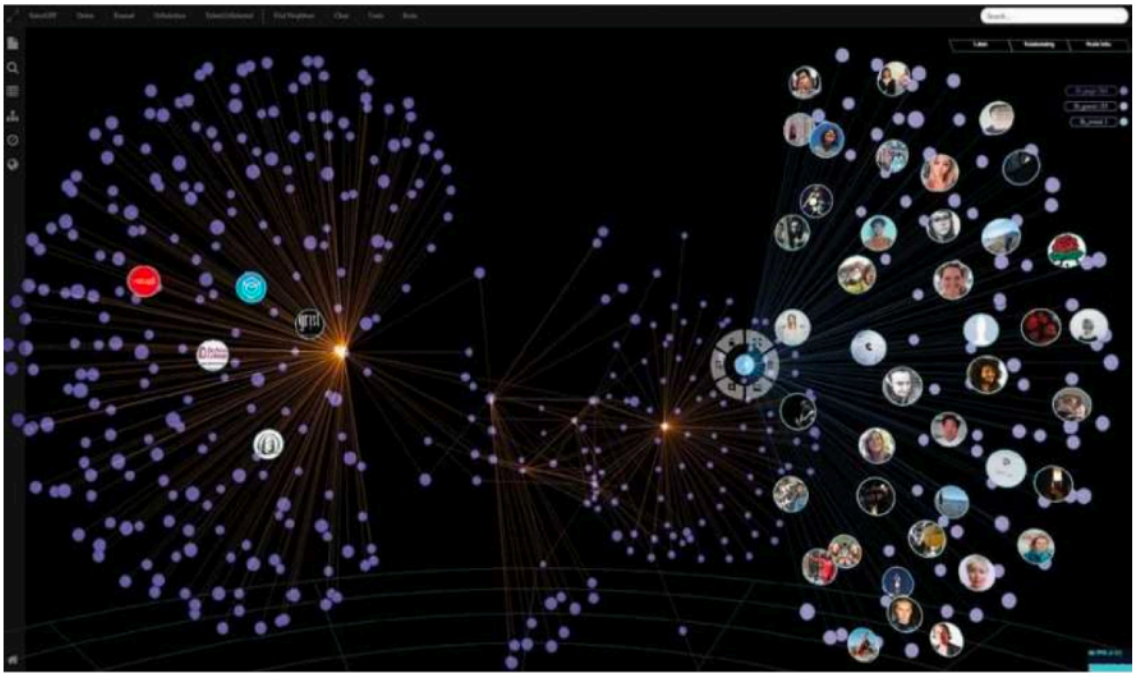

Fig 3.1: Network graph shows connections between entities, clustered into groups.

The main goals of pathfinding visualization include:

- **Educational Purpose**: Pathfinding visualization serves as a powerful tool to educate students, developers, and enthusiasts about different pathfinding algorithms. It helps them understand how these algorithms work, their strengths, weaknesses, and how they make decisions to find the shortest or most efficient path.

- **Algorithm Comparison:** Pathfinding visualization allows for a direct comparison between different pathfinding algorithms. Developers and researchers can see how algorithms like Dijkstra's, A* (A-star), Breadth-First Search, Depth-First Search, and others perform under various scenarios. This can help in choosing the right algorithm for a specific application.

- **Debugging and Testing:** Developers can use pathfinding visualization to test and debug their implementations of pathfinding algorithms. By observing how the algorithm explores and selects paths, they can identify errors or inefficiencies in the code and make improvements.

- **User Experience in Games and Applications:** In video games and applications that involve navigation, pathfinding visualization can enhance the user experience by showing characters or objects moving intelligently from one point to another. This can make the interaction more engaging and realistic.

- **Optimization:** Pathfinding visualization can highlight areas of the map where certain pathfinding algorithms struggle or take longer to find paths. This can lead to insights for optimizing the algorithm or designing the environment in a way that facilitates faster pathfinding.

- **Algorithm Development and Research:** For researchers working on new pathfinding algorithms or improvements to existing ones, visualization can aid in understanding algorithm behaviour and performance characteristics, helping refine and iterate on ideas.
- **Demonstration of Concepts:** Pathfinding visualization is used in tutorials, presentations, and documentation to explain the concepts of graph traversal, heuristic functions, open and closed sets, and other elements of pathfinding algorithms.

- **Problem Solving Strategies:** Observing pathfinding visualization can inspire problem-solving strategies in other areas. The techniques and decision-making processes employed by pathfinding algorithms can sometimes be adapted to solve different types of optimization or search problems. In essence, pathfinding visualization bridges the gap between abstract algorithmic concepts and real-world applications, allowing users to see, understand, and interact with the process of finding paths in a dynamic and visual manner.

Chapter 4

Project Flow

4.1 Workflow of the Pathfinding Visualization Process

1. **User Input**:
    - The project begins by allowing the user to interact with the **grid-based interface**.
    - Users can:
        - Select a **start point** and **end point** on the grid.
        - Add or remove obstacles.
        - Choose the desired pathfinding algorithm (e.g., Dijkstra's, A*, Greedy Best-First Search).
2. **Algorithm Selection**:
    - The user selects an algorithm from the menu.
    - Options include:
        - Dijkstra's Algorithm
        - A* (A-Star) Algorithm
        - Greedy Best-First Search
3. **Grid Customization**:
    - Users can modify the grid further by:
        - Adding walls/obstacles.
        - Resetting the grid.
        - Adjusting the speed of the visualization (e.g., Slow, Medium, Fast).
4. **Algorithm Execution**:
    - Upon selecting the **visualize** button, the chosen algorithm begins execution.
    - The algorithm explores the grid step-by-step and dynamically updates:
        - Visited nodes (in **blue**)
        - Final shortest path (in **yellow**)
5. **Real-Time Visualization**:
    - The algorithm's progress is visually displayed in real-time:
        - Start and end points are marked.
        - Nodes being evaluated are highlighted.

- Obstacles and walls block the algorithm's path.

6. **Performance Metrics**:
   - After execution, the project displays performance metrics such as:
     - Path length
     - Execution time
     - Number of nodes explore

7. **Reset or Repeat**:
   - The user can reset the grid, change configurations, or run another algorithm for comparison.

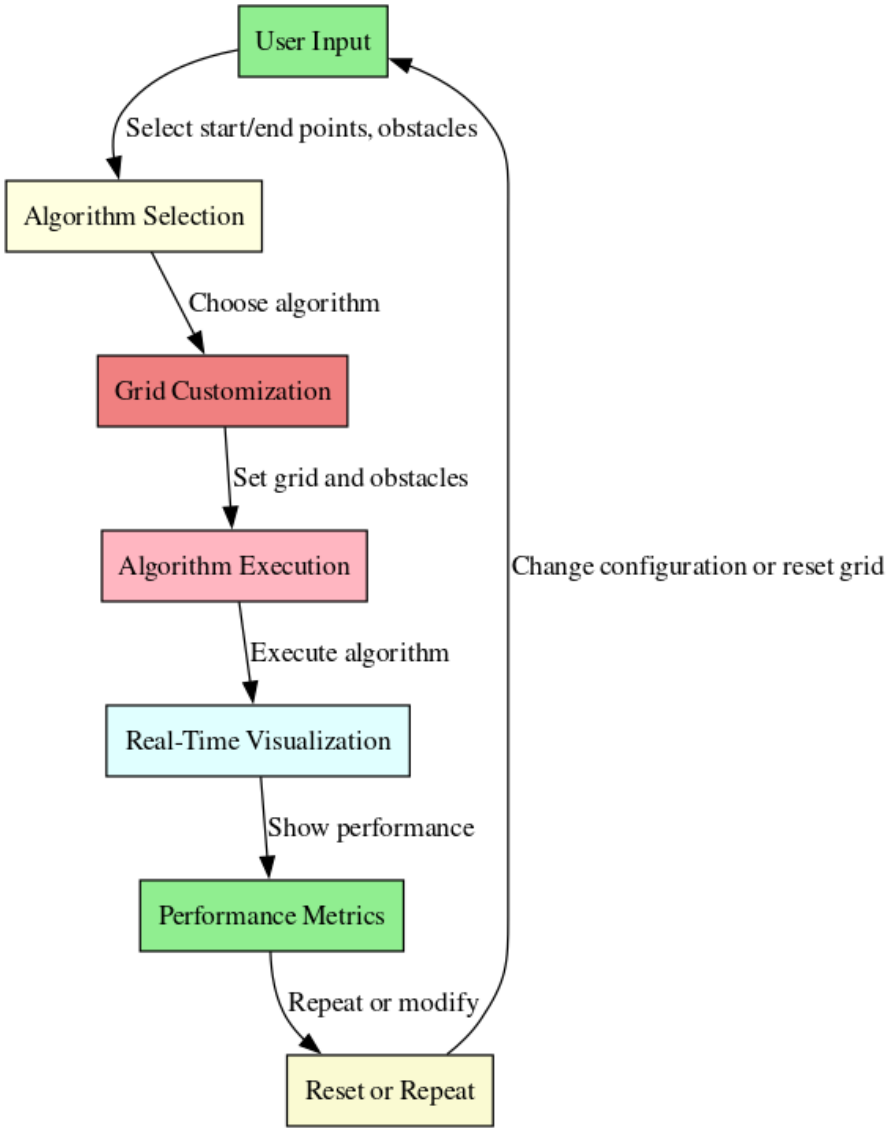

Fig 4.1: Workflow of PathFinder Visualizer

4.2 ER Diagram

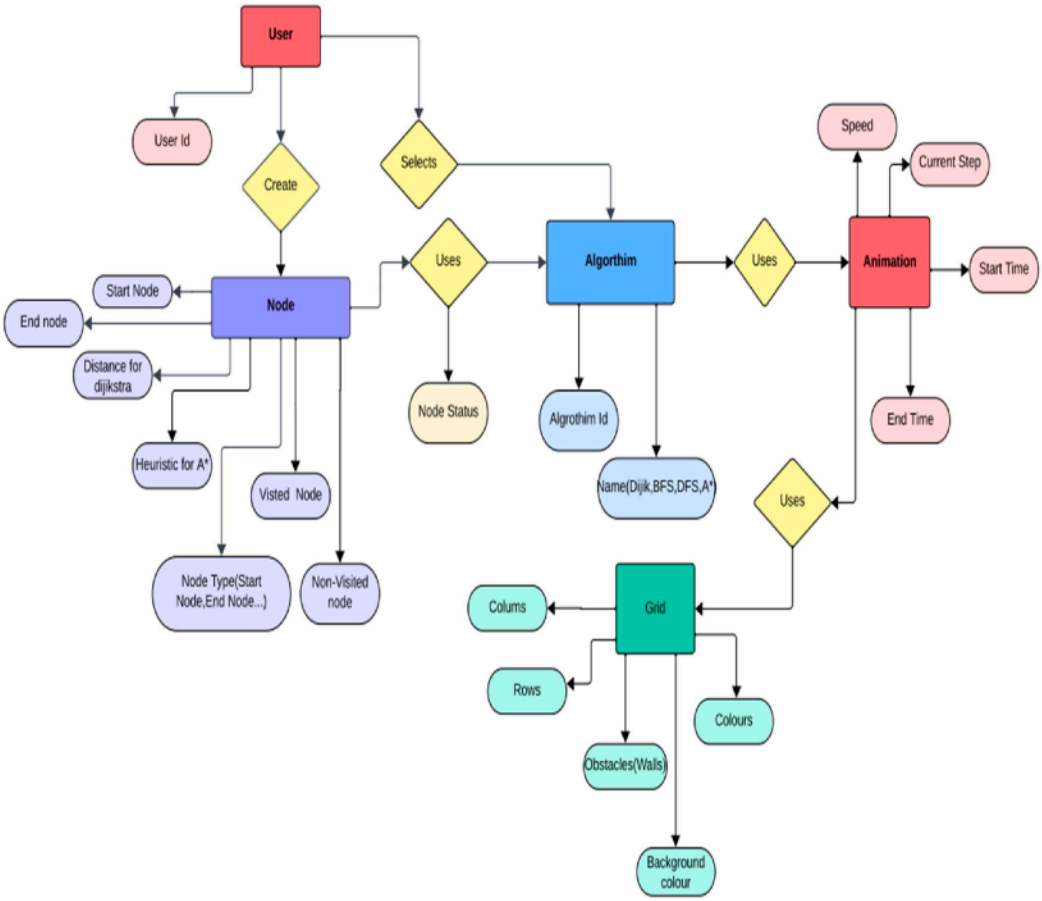

Fig 4.2: ER diagram of Path Finder Visualizer

An Entity-Relationship (ER) diagram is a visual representation used to model the structure and relationships of data within a database or system. In the context of a pathfinding visualization project, an ER diagram helps organize and clarify the various entities and their relationships that are crucial for the project's functionality.

Key elements in an ER diagram for a pathfinding visualization project may include:

- **Nodes and Edges**: Entities representing nodes and edges are fundamental. Nodes may have attributes such as coordinates, names, or types, while edges represent connections between nodes and could have attributes like distance or traversal cost.

- **Graph**: The central entity could be a "Graph" or "Map" entity that encompasses all the nodes and edges. This entity aids in managing and visualizing the entire network.

- **Algorithms**: Depending on the project's complexity, you might have entities for different pathfinding algorithms like Dijkstra's, A*, or others, each with its parameters and settings.

- **Users**: If the project involves user interaction, you may have entities for "Users" with attributes such as username, password, or access level.

- **History/Logs**: To record user activities or maintain a history of pathfinding requests, you may include entities for "History" or "Logs."

- **Visualization Settings**: If the visualization aspect of the project is extensive, you could have entities to store settings like zoom level, color schemes, or display preferences.

- **Permissions:** If multiple users are involved, entities for "Permissions" might be necessary to manage who can access and modify various parts of the system.

- **Reports and Statistics**: For analytics purposes, you may include entities to store generated reports, statistics on pathfinding requests, or performance metrics.

In the ER diagram, relationships between these entities should be clearly defined. For instance, nodes are connected by edges, users may have permissions to access specific parts of the system, and algorithms may utilize the graph/map data.

The ER diagram serves as a valuable blueprint for designing the database schema. It provides a visual guide for the development team to understand the data structure and relationships within the pathfinding visualization project. This diagram aids in maintaining data integrity, improving system performance, and ensuring efficient data retrieval and manipulation for the project's core functionalities.

4.3 Use Case

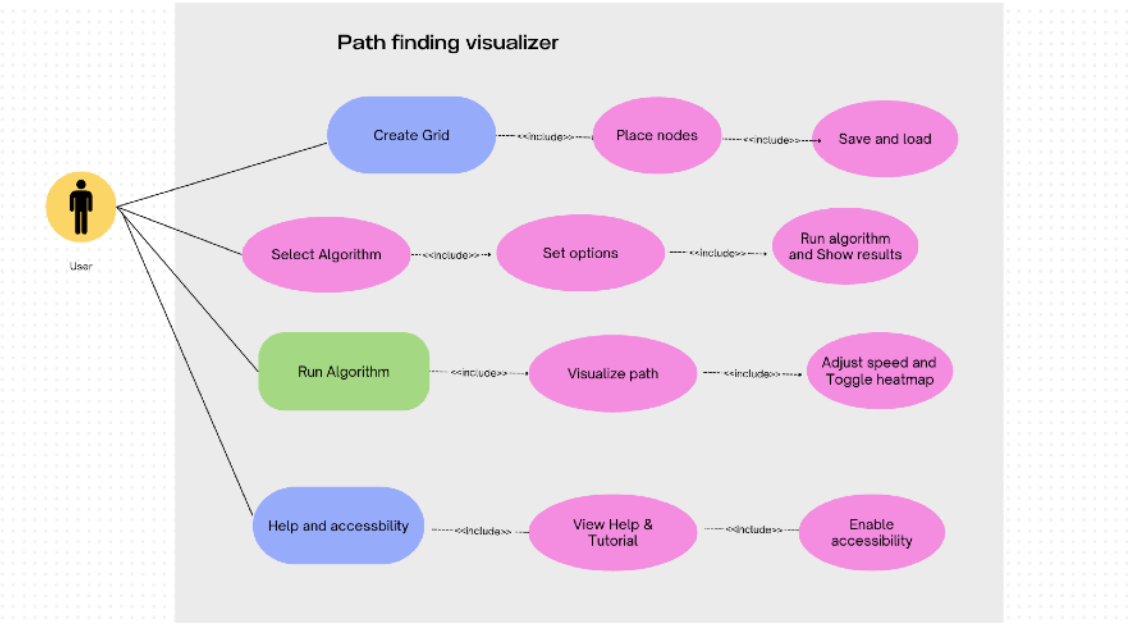

Fig 4.3: Use Case Diagram for Path Finder Visualizer

User Interactions:

- **Create Grid:** Generates a grid to represent the pathfinding environment.
- **Place Nodes:** Manually adds start and end points, as well as obstacles, to the grid.
- **Select Algorithm:** Chooses from various pathfinding algorithms (e.g., A*, Dijkstra's, Breadth-First Search).
- **Set Options:** Configures algorithm-specific settings (e.g., heuristic weight for A*).
- **Run Algorithm:** Executes the selected algorithm to find the shortest path between the start and end points.
- **Visualize Path:** Displays the calculated path on the grid.
- **Adjust Speed:** Controls the visualization speed.
- **Toggle Heatmap:** Enables/disables a heatmap showing the algorithm's exploration process.

- **View Help & Tutorial:** Accesses information and guidance on using the visualizer.
- **Enable Accessibility:** Activates accessibility features for users with disabilities.
- **Save and Load:** Saves and loads grid configurations and algorithm settings.

Overall Purpose:

The Path Finding Visualizer is a tool for understanding and experimenting with different pathfinding algorithms in a visual and interactive way. It helps users gain insights into how these algorithms work and compare their performance under various conditions.

4.4 Data Flow Diagram

4.4.1 Level 0 Data Flow Diagram:

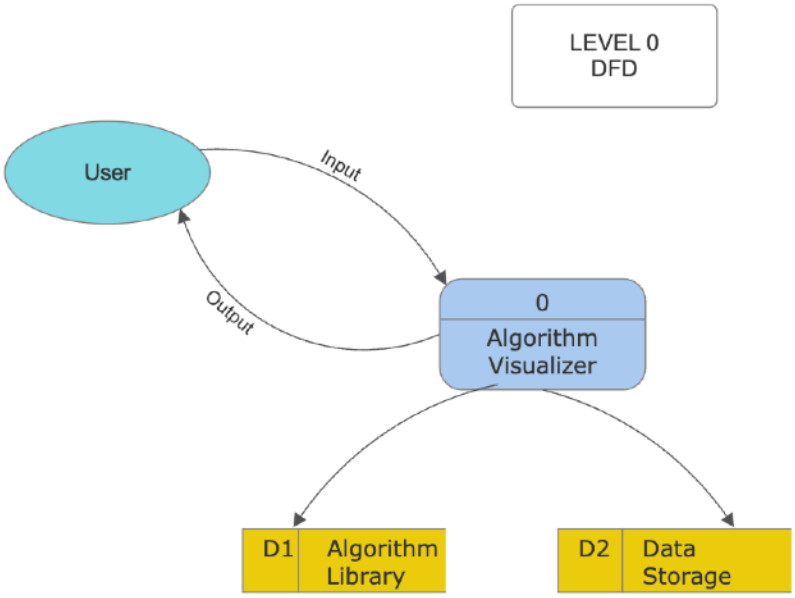

Fig 4.4: Level 0 Data Flow Diagram

The DFD represents a system for visualizing algorithms. It consists of:

- **External Entity:** The "User" is the external entity interacting with the system.
- **Process:** The "Algorithm Visualizer" is the central process that handles the visualization.
- **Data Stores:** The "Algorithm Library" and "Data Storage" are repositories for algorithms and data.

Data Flows:

The arrows indicate the flow of data between the components:

- **Input:** The user provides input to the Algorithm Visualizer.

- **Output:** The Algorithm Visualizer generates output that is displayed to the user.
- **Algorithm Library:** The Algorithm Visualizer accesses algorithms from the Algorithm Library.
- **Data Storage:** The Algorithm Visualizer may access data from the Data Storage.

User input processed by Algorithm Visualizer, accessing Algorithm Library and Data Storage as needed, to generate visualized output displayed to the user.

This Level 0 Data Flow Diagram illustrates the interaction between a User and the Algorithm Visualizer system. The user provides input, which is processed by the system using data retrieved from two sources: the Algorithm Library (D1) and Data Storage (D2). The system processes the input and sends the resulting output back to the user. This diagram provides a high-level overview of the system's data flow, showing how input, processing, and storage interact.

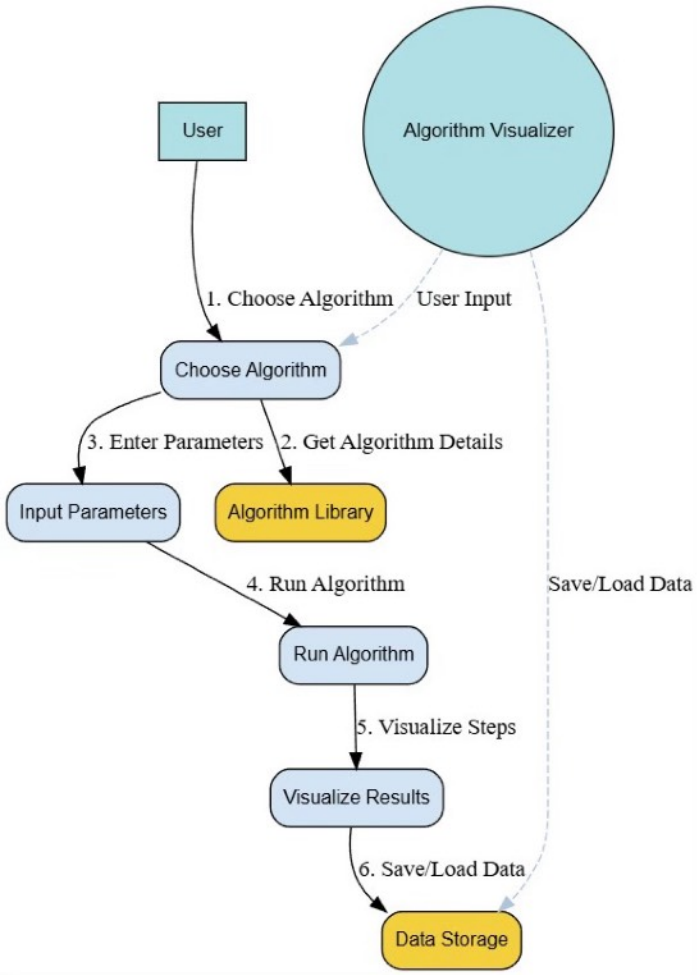

Fig 4.5: Level 1 Data Flow Diagram

4.4.2 Level 1 Data Flow Diagram:

This DFD provides a more detailed view of the algorithm visualization process compared to the previous one. Here's a breakdown:

1. Choose Algorithm: The user selects an algorithm from the Algorithm Library.

2. Get Algorithm Details: The Algorithm Visualizer retrieves information about the selected algorithm from the Library, including its parameters and execution steps.

3. Enter Parameters: The user provides input values for the algorithm's parameters.

4. Run Algorithm: The Algorithm Visualizer executes the selected algorithm using the provided parameters.

5. Visualize Steps: The Algorithm Visualizer displays the step-by-step execution of the algorithm, highlighting the current step.

6. Visualize Results: The Algorithm Visualizer displays the final output of the algorithm.

7. Save/Load Data: The user can save or load data related to the algorithm, its parameters, and its execution.

DFD shows optional flows, implied data flow, and granular algorithm visualization process steps. The Level 1 Data Flow Diagram provides a detailed view of the processes within the Algorithm Visualizer system. The interaction begins with the user choosing an algorithm, which triggers the system to fetch the algorithm details from the Algorithm Library. Next, the user enters the required parameters, allowing the system to proceed with running the algorithm. Once the algorithm is executed, the system visualizes its steps and outputs the results to the user. Additionally, the system interacts with the Data Storage to save or load data as needed.

Chapter 5

Project Overview

This project seeks to develop an application that visualizes pathfinding algorithms, including Dijkstra's Algorithm, A*, and Breadth-First Search (BFS), in real-time. Users can input starting and ending points on a graph or map, and the application will compute and display the shortest or least-cost path between these locations.

Key Features:

•**Interactive Visualization:** Users can interact with the graphical interface to set start and end points, view real-time algorithm execution, and observe how the algorithm explores the graph.

•**Multiple Algorithm Support:** The application will support various pathfinding algorithms, each with unique characteristics, such as Dijkstra's, A*, and BFS.

•**Dynamic Updates:** The visualization will update dynamically, showing the progression of the algorithm as it searches for the optimal path.

• **Graph Customization:** Users can create and modify graphs by adding nodes and edges, adjusting weights, and configuring obstacles to test different scenarios.

• **Educational Tool:** The application will serve as an educational resource for learning and understanding pathfinding algorithms, making complex concepts accessible through visual representation.

Menu Options:

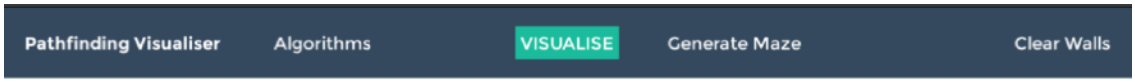

Fig 5.1: Shows menu options in the app

• **Algorithms:** Precise step-by-step instructions for solving problems. We can select different search algorithm from the given list like: A* Search, Dijkstra's Search, Greedy Best First Search.

• **Visualize:** Represent concepts or data in a graphical format for better understanding. It is used to start the search on the below maze according to the selected searching algorithm.

• **Generate Maze:** Create a maze structure with walls and paths.It gives option like Basic Random Maze in which it will create the basic maze.

• **Clear Wall:** Remove a wall in a maze to create a new path. This options helps you to clear maze created using generate maze options, once your searching is done using a particular search technique use clean wall option it will clear the grid area for a new maze.

Algorithm Options:

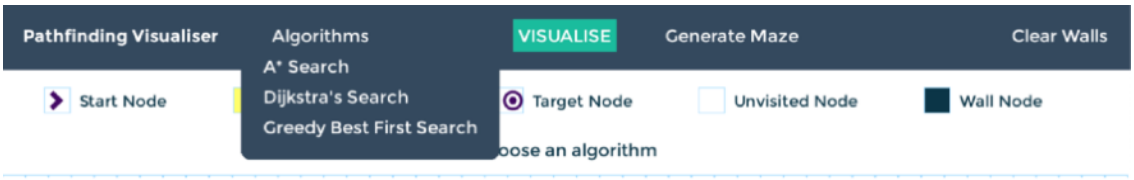

Fig 5.2: Show type of algorithms in the app

- **A* Search:** An informed search algorithm that uses a heuristic function to efficiently find the shortest path between two nodes in a graph.

- **Dijkstra's Search:** A greedy algorithm that finds the shortest path from a source node to all other nodes in a weighted graph.

- **Greedy Best First Search**: A search algorithm that always expands the node that appears to be closest to the goal, based on a heuristic function.

Generate Maze Options:

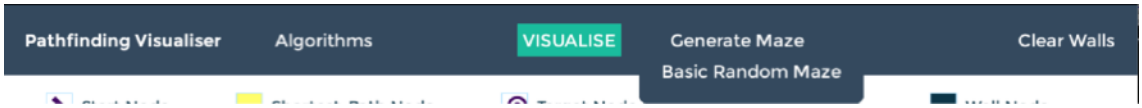

Fig 5.3: Shows the option of Basic Random Maze

Basic Random Maze:

Basic Random Maze generation involves randomly placing walls and passages to create a simple, unpredictable maze layout.

A simple algorithm that starts from a random cell and carves a path to a random neighbor, repeating until all cells are visited, creating a maze with a single, winding path.

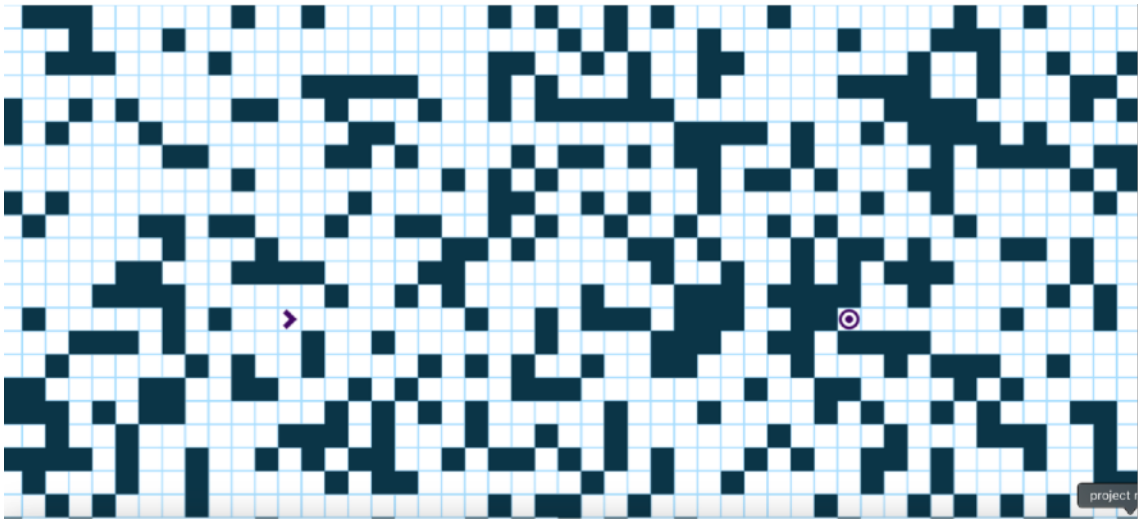

Fig 5.4: Basic Random Maze is created

Elements :

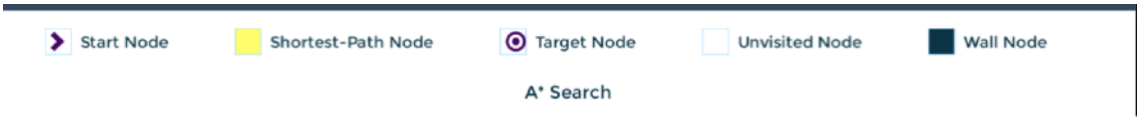

Fig 5.5: Shows the elements in app

- **Start Node:** The node where the search begins. The starting node is the initial point from which a search algorithm begins exploring a graph or maze. It's the first node the algorithm examines as a potential solution or goal reach.

- **Shortest-Path Node:** A node that is currently part of the shortest path found so far. The shortest path found by the algorithm, highlighted to indicate the route taken.

- **Target Node:** The goal node that the search is trying to reach.The goal node, the desired destination, is the final node a search algorithm seeks to find while traversing a graph or maze. Once identified, the algorithm can trace back the path to provide the solution.

- **Unvisited Node:** A node that has not yet been explored by the algorithm. An unvisited node refers to a location or point in a graph that has not yet been explored during the search for the optimal path.

- **Wall Node:** A node that is an obstacle and cannot be traversed. Blocked or impassable areas within the grid that the pathfinding algorithm must navigate around.

- **Algorithm Controls:** User interface elements to control the visualization, such as starting the search, pausing, resetting, and selecting different algorithms.

- **Heuristics (for some algorithms):** Estimates or guiding principles used to prioritize nodes in the search, commonly seen in algorithms like A*.

- **Cost or Distance Values:** Numeric values associated with nodes or edges, representing the cost or distance from one node to another.

- **Animation:** Visual representation of the algorithm's movement, often animated to illustrate how it explores nodes and progresses toward the goal.

- **Algorithm Selection:** Options to choose different pathfinding algorithms, allowing users to compare their behaviours and efficiency.
- These elements collectively help users understand and interact with how pathfinding algorithms work, providing insights into their decision-making processes and showcasing the paths they generate to reach the destination.

Working:

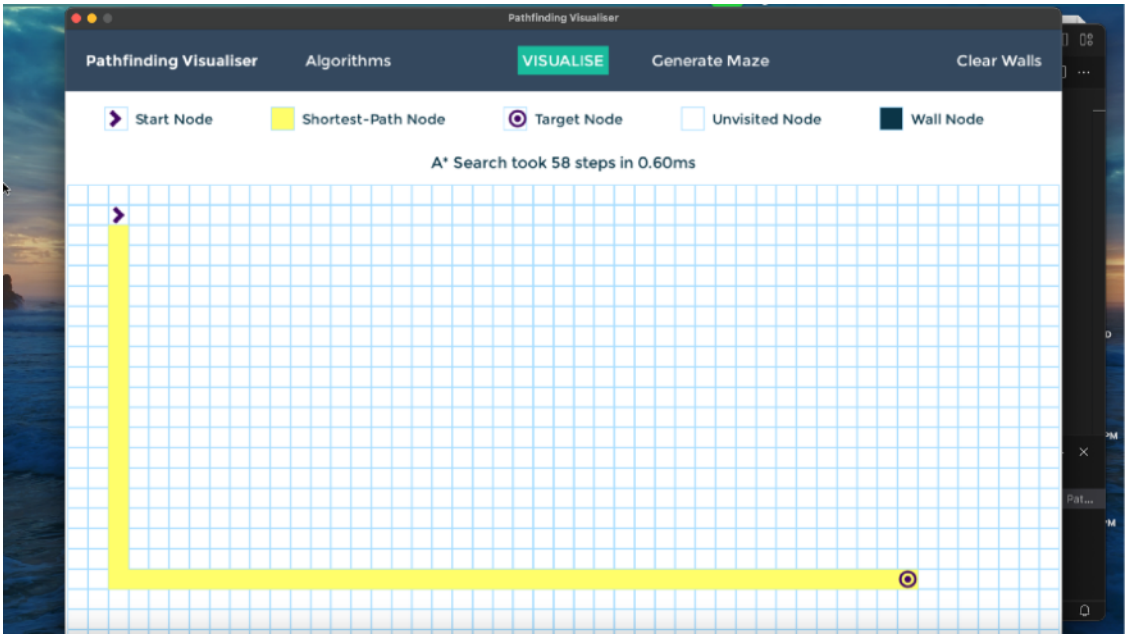

Fig 5.6: Explain working of A* in the app

The visualization displays a grid-based maze with various colored nodes representing states:

Start Node: Yellow arrow pointing to the start.

Shortest-Path Node: Yellow nodes forming the path.

Target Node: Orange circle.

Unvisited Node: White.

Wall Node: Dark blue.

The top left corner indicates the A* search algorithm, which found the shortest path in

0.60 milliseconds using 58 steps.

A* is a popular pathfinding algorithm that combines a heuristic function and path cost to find the shortest path between two nodes. It maintains open and closed lists, selecting the lowest combined cost node at each step until the target is reached or the open list is empty.

The yellow path highlights the shortest path, while colored nodes provide visual feedback on the search process. The text displays the algorithm, steps, and execution time.

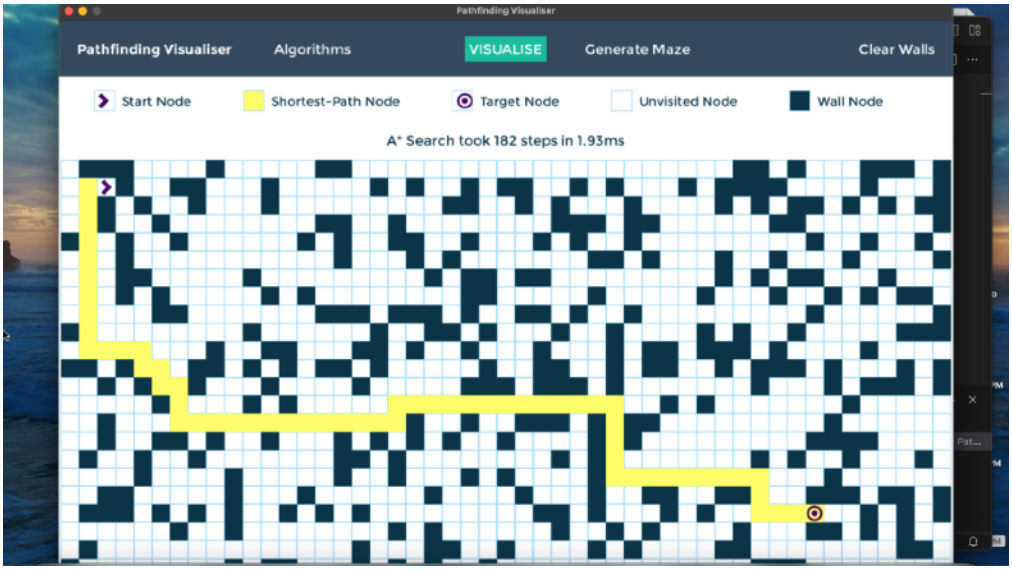

Fig 5.7 : Explains the working of the A* in the basic random maze

- **A Search in Action:*** The A* algorithm is a powerful pathfinding technique that intelligently explores the maze, guided by a heuristic function. This function estimates the distance to the goal, helping the algorithm prioritize the most promising paths.
- **Visualizing the Path:** The yellow path you see is the result of the A* algorithm's careful calculations. It represents the shortest and most efficient route from the start point to the goal.
- **Step-by-Step Exploration:** The algorithm systematically explores the maze, considering each node and its potential connections. It evaluates the cost of reaching each node and the estimated cost to the goal, making informed decisions about which path to follow.
- **Obstacle Avoidance:** The algorithm cleverly navigates around the obstacles (represented by the dark blue squares), ensuring that the chosen path is clear and unobstructed.
- **Efficient Pathfinding:** The A* algorithm's efficiency is evident in the relatively short number of steps taken to find the optimal path. This demonstrates its effectiveness in solving complex pathfinding problems.

Chapter 6

Project Outcome

The Pathfinding Visualization Project aims to achieve the following outcomes:

- **Educational Value:** The project offers an interactive and visually appealing platform for students, educators, and developers to grasp and compare pathfinding algorithms. It simplifies complex algorithmic concepts through real-time animations, making them more accessible and understandable.

- **Algorithm Insights:** The project provides a comprehensive demonstration of the working principles of various algorithms, including Dijkstra's, A*, and Greedy Best-First Search. It highlights the differences in efficiency, pathfinding strategies, and performance metrics, enabling users to gain a deeper understanding of these algorithms.

- **User Interaction:** The project empowers users to interact with a grid-based environment, placing obstacles and customizing start and end points. This interactive feature allows users to test and experiment with algorithm behavior in diverse scenarios, fostering a hands-on learning experience.

- **Visualization Features:** The project offers a range of visualization features, including step-by-step progress tracking, visual marking of explored nodes, shortest paths, and obstacles. Real-time feedback and adjustable visualization speed further enhance the user experience.

- **Performance Analysis:** The project provides metrics such as path length, nodes evaluated, and execution time for each algorithm run. This data allows users to compare algorithms and gain insights into their strengths and weaknesses, promoting a deeper understanding of their capabilities.

- **Practical Applications:** The project showcases real-world applications of pathfinding algorithms in various fields, including navigation, robotics, game development, and optimization problems. By demonstrating practical scenarios, the project enhances users' ability to apply these algorithms in real-world contexts.

- **Ease of Use:** The project features an intuitive and user-friendly interface, making it accessible to learners and non-technical users alike. It provides options to reset and rerun algorithms, encouraging users to explore different configurations and algorithms.

- **Skill Development:** The project enhances users' problem-solving and analytical skills through experimentation with grid configurations and algorithm selection. By providing a hands-on learning experience, the project empowers users to develop essential skills in pathfinding and related fields.

Chapter 7

Bibliography

1. Online Articles and Tutorials:

- Red Blob Games. (n.d.). A* Pathfinding for Beginners. Retrieved from https://www.redblobgames.com/pathfinding/a-star/introduction.html
- Clement, D. (2017). Pathfinding Algorithms in Python. RealPython. Retrieved from https://realpython.com/python-pathfinding-algorithms/
- Baranwal, V. (2019). Introduction to Graph Algorithms:Breadth-First Search and Depth-First Search. Towards Data Science. Retrieved from https://towardsdatascience.com/introduction-to-graph-algorithms-breadth-first-search.
- Pygame Documentation. (n.d.). Retrieved from https://www.pygame.org/docs/.

2. Books:

- LaValle, S. M. (2006). Planning Algorithms. Cambridge University Press.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). The MIT Press.
- Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. Numerische Mathematik, 1, 269-271.

3.GitHub Repositories:

- Python Pathfinding. (2023). GitHub Repository. Retrieved from https://github.com/brean/python-pathfinding.
- TechWithTim. (2023). Pathfinding Visualizer with Python and Pygame. GitHub Repository. Retrieved from https://github.com/techwithtim/Pathfinding-Visualizer.

4. Educational Platforms:

- Udemy. (n.d.). Python A* Pathfinding Visualizer. Retrieved from https://www.udemy.com/course/python-a-pathfinding-visualizer.
- YouTube - Code with Harry YouTube channel for concepts learning and python programming.
- Coursera. (n.d.). Algorithms Specialization. Retrieved from https://www.coursera.org/specializations/algorithms.