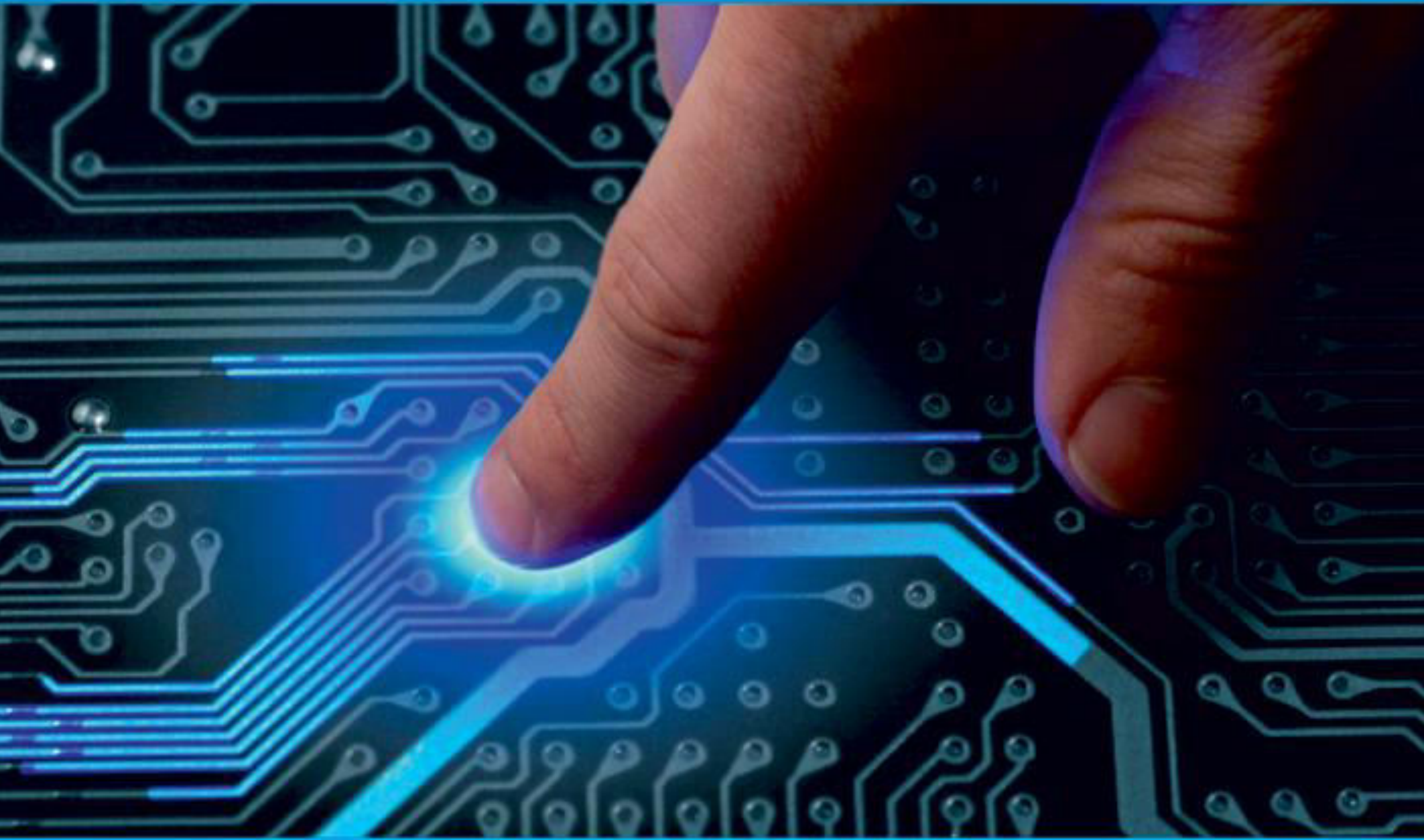
IJIRCCE

e-ISSN: 2320-9801 | p-ISSN: 2320-9798



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

Volume 12, Issue 4, April 2024

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.379

International Journal of Innovative Research in Computer and Communication Engineering

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 4, April 2024 ||

| DOI: 10.15680/IJIRCCE.2024.1204129 |

Path Finding Visualizer

Prafulla Walunj[1], Omkar Gore[1], Harshad Lole[1], Dnyaneshwar More1, Prof. Vishakha Chilpipre[2], Prof. Trupti Khose[2]

Student, Department of Information Technology, Dhole Patil College of Engineering Pune, Maharashtra, India[1]

Professor, Department of Information Technology, Dhole Patil College of Engineering Pune, Maharashtra, India[2]

ABSTRACT: Visualization is an efficient way of learning any concept faster than conventional methods. Modern technology allows the creation of e-Learning tools that also help in improving computer science education very much. The goal of this project is to create a web-based e-Learning tool, 'Path-finding Visualizer', which can be used to visualize shortest-path algorithms. The conceptual application of the project is illustrated by the implementation of algorithms like Dijkstra's, A*, and DFS. This project aims to complete all these tasks with some knowledge of HTML, CSS, JavaScript, and React Framework. The end product is a web application so that any user can easily see and learn the working of the algorithms. The user-friendliness of the project provides users with easy instructions on how to operate it. The initial results of using the application show promises of the benefits of this e-Learning tool towards students getting a good understanding of shortest paths algorithms.

I. INTRODUCTION

The Pathfinding Visualizer project is a software application that aims to provide a visual representation of various pathfinding algorithms. Pathfinding algorithms are used to find the shortest or most optimal path between two points in a graph or grid.The main objective of the Pathfinding Visualizer project is to create an interactive and educational tool that allows users to understand and compare different pathfinding algorithms. The project provides a graphical user interface where users can create a grid-like environment, set start and end points, and visualize how different algorithms navigate through the grid to find the optimal path.The project offers several key features to enhance the user experience. Users can choose from a variety of popular pathfinding algorithms such as A* algorithm, Depth-First Search, Breadth-First Search (BFS), and more. They can observe the step-by-step progress of the algorithm or run it in real-time to see how it explores the grid, evaluates node distances, and makes decisions to reach the destination.Furthermore, the Pathfinding Visualizer project allows users to customize the grid by adding obstacles or walls that the algorithms must navigate around. This feature provides insights into how obstacles affect the pathfinding process and helps users understand the importance of choosing the right algorithm for different scenarios.

II. IMPLEMENTATION

1.Set up the HTML structure:
- Create a <div> container to hold the grid and controls.
- Inside the container, create a grid using nested <div> elements to represent individual cells.

2.Style the grid using CSS:
- Apply CSS styles to the container, grid cells, and any additional elements.
- Define different classes for start node, end node, obstacles, and the path.

3.Implement the JavaScript functionality:
- Write JavaScript functions to handle user interactions and implement the pathfinding algorithms.
- Add event listeners to detect mouse clicks and movements on the grid.
- Handle the logic to set the start node, end node, and obstacles based on user clicks.
- Implement the selected pathfinding algorithm (e.g., Dijkstra's algorithm, A* algorithm, BFS, DFS) using appropriate data structures (e.g., priority queue, queue, stack).
- Write helper functions for grid manipulation, node evaluation, and path generation.
- Update the grid dynamically to reflect the algorithm's progress and the final path.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 4, April 2024 ||

| DOI: 10.15680/IJIRCCE.2024.1204129 |

4.Visualize the algorithm:

Use CSS classes to change the color or styling of nodes to indicate their state (e.g., visited, evaluated, in the path).

Implement animations or delays between algorithm steps to visualize the process gradually.

III.LITERATURE SURVEY

Pathfinding algorithms and visualization techniques play a crucial role in various domains such as robotics, video games, and route planning applications. This literature survey aims to explore the existing research on pathfinding algorithms and visualization techniques, with a focus on their applications and advancements.

❖ Pathfinding Algorithms:

1. A* Algorithm - Hart, P. E., Nilsson, N. J., & Raphael, B. (1968).
2. Depth-First Search Algorithm - Cormen,T. H., Leiserson, C. E., Rivest, R. L., & Stein, C(2009).
3. Breadth-First Search Algorithm - Leiserson,C. E.,Rivest, R. L., Stein,C., & Cormen,T. H. (2009).

eq. (3)

IV.ALGORITHM

A* Algorithm :-

The A* algorithm follows these steps to find the shortest path between a start node and a goal node in a graph:

```
1. Initialize the open list and the closed set. The open list contains nodes that are to be explored, and the closed set contains nodes that have already been visited.
2. Create the start node and calculate its heuristic cost (`h`) from the start node to the goal node. Set the cost from the start node to the current node (`g`) as 0.
3. Add the start node to the open list.
4. While the open list is not empty:
    - Select the node with the lowest total cost (`f`) from the open list. This node will be the current node.
    - Check if the current node is the goal node. If so, the shortest path has been found.
    - Otherwise, move the current node from the open list to the closed set.
    - Generate the neighboring nodes of the current node.
    - For each neighboring node:
        - Calculate the cost from the start node to the neighboring node (`g`).
        - Calculate the heuristic cost from the neighboring node to the goal node (`h`).
        - Calculate the total cost (`f = g + h`).
        - If the neighboring node is already in the closed set and the new cost is higher, skip to the next neighboring node.
        - If the neighboring node is already in the open list and the new cost is lower, update its cost and parent.
        - If the neighboring node is not in the open list, add it to the open list with the calculated costs and parent.
5. If the open list becomes empty and the goal node has not been reached, there is no path from the start node to the goal node.
```

The A* algorithm utilizes a heuristic function (such as Euclidean distance or Manhattan distance) to estimate the remaining cost (`h`) from each node to the goal node. This heuristic helps guide the algorithm towards the goal, making it more efficient than Dijkstra's algorithm in many cases.

By using the A* algorithm, you can find the shortest path from the start node to the goal node in various applications, such as pathfinding in games, route planning, and navigation systems.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 4, April 2024 ||

| DOI: 10.15680/IJIRCCE.2024.1204129 |

Deapth-First Search Algorithm :-

The Depth-First Search (DFS) algorithm explores a graph by traversing as far as possible along each branch before backtracking. Here are the steps of the DFS algorithm:

1. Choose a starting node as the current node and mark it as visited.
2. Explore the current node by visiting one of its unvisited neighbors.
3. If there are unvisited neighbors, choose one and go to step 4. Otherwise, backtrack to the previous node.
4. Set the chosen neighbor as the new current node, mark it as visited, and repeat steps 2-4.
5. If there are no unvisited neighbors and all nodes have been visited, the algorithm is complete.
6. If there are unvisited nodes remaining, choose one as the new starting node and go to step 1.

The DFS algorithm explores the graph deeply before visiting the neighboring nodes. It uses a stack to keep track of the nodes to visit. When backtracking, it removes nodes from the stack until it finds a node with unvisited neighbors.

DFS is often implemented using recursion or an explicit stack data structure. It is used for various applications, including graph traversal, cycle detection, connected component analysis, and solving puzzles like maze exploration.

It's important to note that DFS does not necessarily find the shortest path between two nodes. If a goal node is the objective, other algorithms like Breadth-First Search or Dijkstra's algorithm might be more suitable.

Breadth-First Search Algorithm :-

The Breadth-First Search (BFS) algorithm explores a graph by systematically traversing all the vertices at the current depth level before moving to the next depth level. Here are the steps of the BFS algorithm:

1. Choose a starting node as the current node and enqueue it into a queue.
2. Mark the current node as visited.
3. While the queue is not empty, repeat steps 4-7.
4. Dequeue a node from the front of the queue and set it as the current node.
5. Explore the current node by visiting all its unvisited neighbors.
6. Mark each unvisited neighbor as visited and enqueue it into the queue.
7. If there are more nodes in the queue, go to step 4. Otherwise, the algorithm is complete.

The BFS algorithm ensures that all the nodes at a particular depth level are visited before moving to the next level. This guarantees that the algorithm explores the graph in breadth-first fashion, moving from the starting node to its neighbors, then to their neighbors, and so on.

BFS uses a queue data structure to keep track of the nodes to visit. The queue follows the First-In-First-Out (FIFO) principle, ensuring that the nodes are visited in the order they were inserted.

BFS is commonly used to find the shortest path between two nodes in an unweighted graph. It can also be used for other applications such as connected component analysis and puzzle solving.

It's important to note that BFS may not be efficient for large graphs or graphs with complex edge weights. In such cases, other algorithms like Depth-First Search or Dijkstra's algorithm might be more suitable.

V. RESULTS

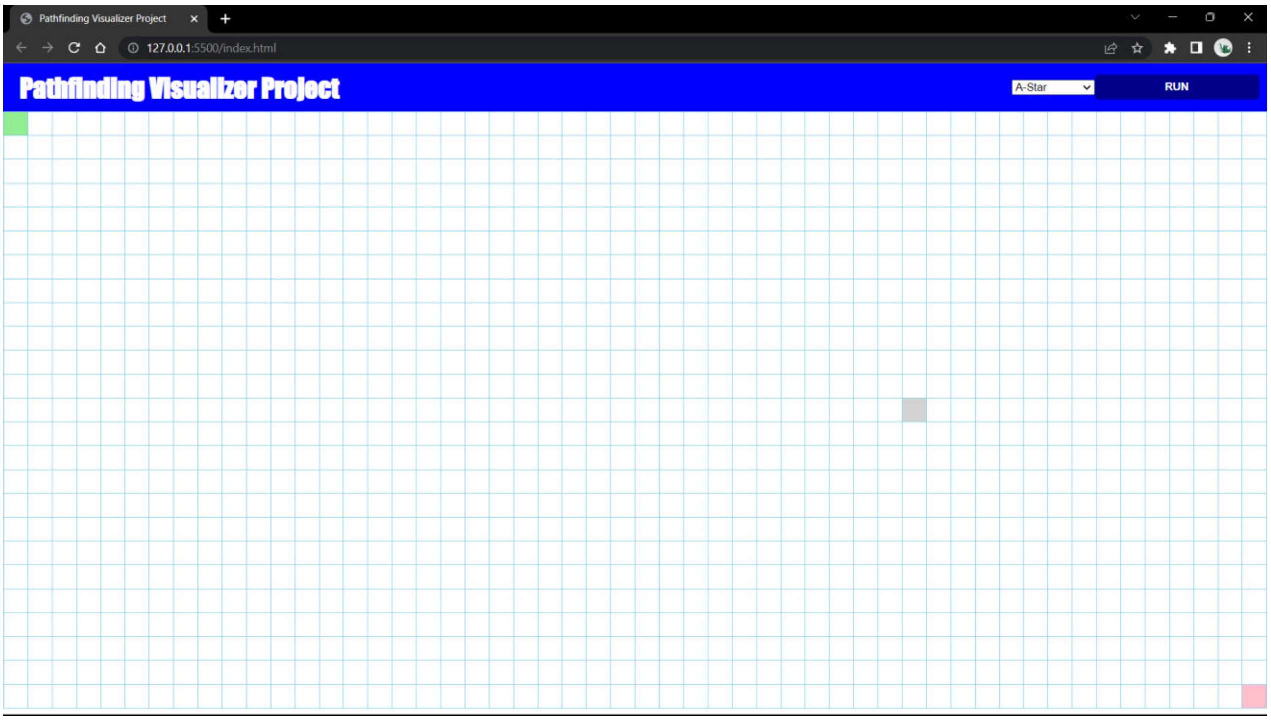

Pathfinding Visualizer Project
127.0.0.1:5500/index.html
Pathfinding Visualizer Project
A-Star
RUN

Fig No 1: Homepage

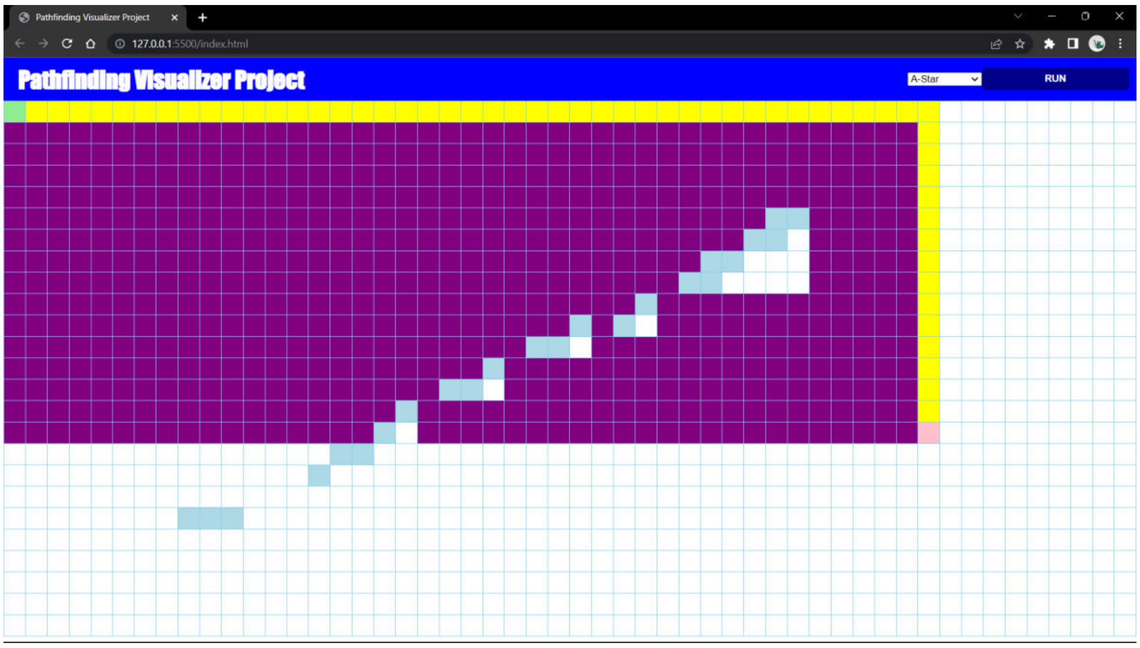

Pathfinding Visualizer Project
127.0.0.1:5500/index.html
Pathfinding Visualizer Project
A-Star
RUN

Fig No. 02 : A* Visual

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 4, April 2024 ||

| DOI: 10.15680/IJIRCCE.2024.1204129 |

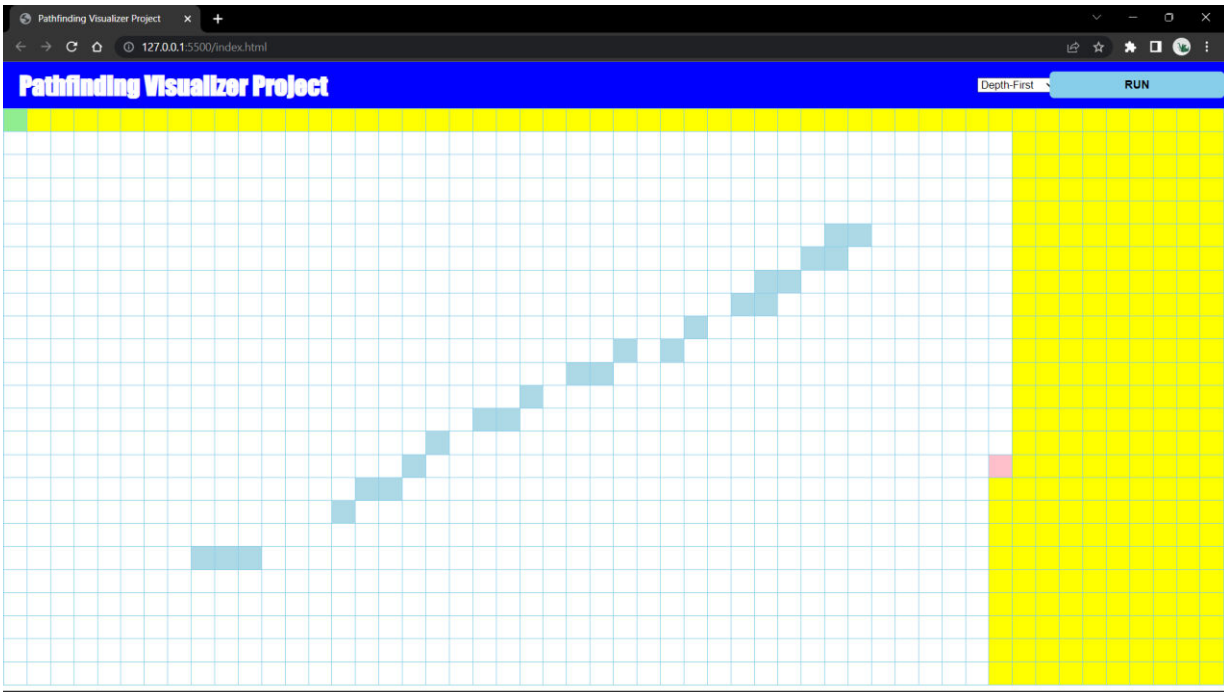

Pathfinding Visualizer Project
127.0.0.1:5500/index.html
Pathfinding Visualizer Project
Depth-First
RUN

Fig No. 3 : Depth-First Search Visual

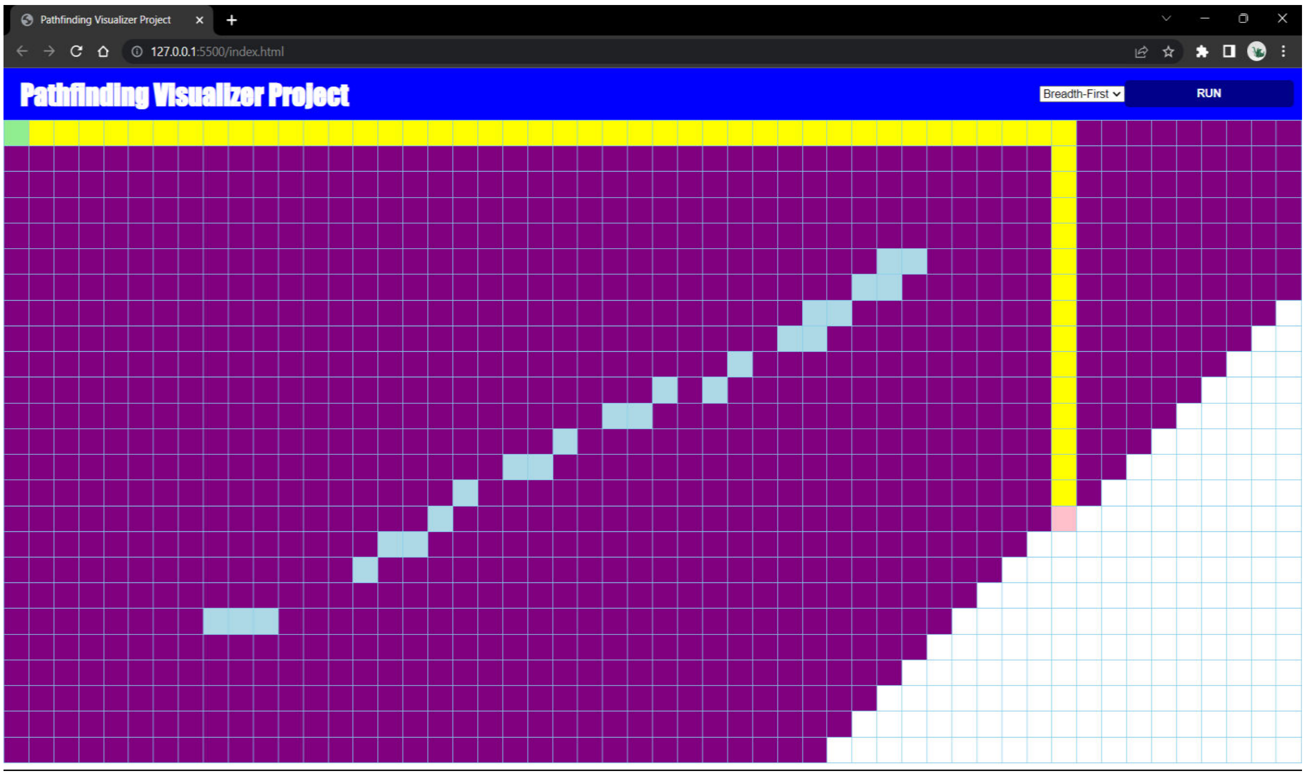

Pathfinding Visualizer Project
127.0.0.1:5500/index.html
Pathfinding Visualizer Project
Breadth-First
RUN

Fig No. 04 : Breadth-First Search Visual

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 4, April 2024 ||

| DOI: 10.15680/IJIRCCE.2024.1204129 |

VI. CONCLUSION

By implementing popular algorithms like A*, BFS and DFS, the project demonstrates the versatility and effectiveness of different approaches to pathfinding. The visualization aspect greatly enhances the learning experience, allowing users to witness the step-by-step process of algorithm execution and understand how decisions are made to find the optimal path.

The project's modular design and extensible nature provide ample opportunities for future enhancements and additions. Additional algorithms, maze generation techniques, customization options, and performance metrics can be incorporated to further expand the functionality and cater to diverse user requirements. The introduction of weighted graphs, multi-end point support, and real-time updates would provide even more realistic and versatile pathfinding scenarios.

Overall, the Pathfinding Visualizer project serves as a valuable learning resource and practical tool for anyone interested in pathfinding algorithms. It empowers users to explore, experiment, and gain a deeper understanding of these algorithms' principles and applications in a visually engaging and intuitive manner.

REFERENCES

1. Pathfinder Visualizer using JavaScript and HTML/CSS:
   - Tutorial: https://levelup.gitconnected.com/pathfinding-visualizer-in-javascript-8d09d0b8afa3
2. A* Algorithm:
   - Wikipedia: https://en.wikipedia.org/wiki/A*_search_algorithm
   - GeeksforGeeks: https://www.geeksforgeeks.org/a-search-algorithm/
   - Brilliant: https://brilliant.org/wiki/a-star-search/
3. HTML Canvas:
   - MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial
4. CSS Grid- MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout
5. JavaScript Event Handling:
   - MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/Guide/Events/Overview_of_Events_and_Handlers
6. Visualization and Animation Techniques:
   - CSS-Tricks: https://css-tricks.com/almanac/properties/a/animation/
   - CSS Animation Examples: https://www.creativebloq.com/inspiration/css-animation-examples

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi crossref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING