Path Finding Visualizer

A PROJECT REPORT

Submitted by

Tushar Sharma(201500746)
Jaideep Gautam (201500310)
Srashti Gautam(2115990020)
Deepak Singh(201500209)

in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

Computer Engineering and Application

GLA University, Mathura

APRIL 2023

BONAFIDE CERTIFICATE

Certified that this project report **"Path Finding Visualizer"**is the bonafide work of "**Tushar Sharma, Jaideep Gautam, Srashti Gautam, Deepkar Singh"**who carried out the project work under my/our
supervision.

SIGNATURE

SIGNATURE

SUPERVISOR

HEAD OF THE DEPARTMENT

Submitted for the project viva-voce examination held on

INTERNAL EXAMINER **EXTERNAL EXAMINER**

TABLE OF CONTENTS

ABSTRACT

Algorithm visualization has been high topic in Computer science education for years, but it did not make its way to schools/collages lecture halls as the main educational tool. The present paper identifies two key circumstances that an algorithm visualization must fulfil to be successful: general availability of used software, and visualization of why an algorithm solves the problem rather than what it is doing. One possible method of "why" algorithm visualization is using algorithm unvarying rather than showing the data conversion only.

Theory and Software authentication and many researchers believe that knowledge of invariants is essentially correspondent to understanding the algorithm. Algorithm stable visualizing leads to codes that are computationally very commanding, and powerful software tools require downloading/installing compilers and/or runtime machines, which restrict the opportunity of users. One our important finding is that, due to computing power of the recent hardware, even very entangle visualization involving 3D animation could be successfully implemented using interpreted graphic script languages like JavaScript that are available to every web user without any installation. The use of images to deliver some useful information about algorithms.

CHAPTER 1.

INTRODUCTION

1.1. Client Identification/Need Identification/Identification of relevant Contemporary issue

- To offer a platform that can be used to understand the working of the algorithm,
- it's a problem that needs resolution.
- \To provide a greater user experience while using the website

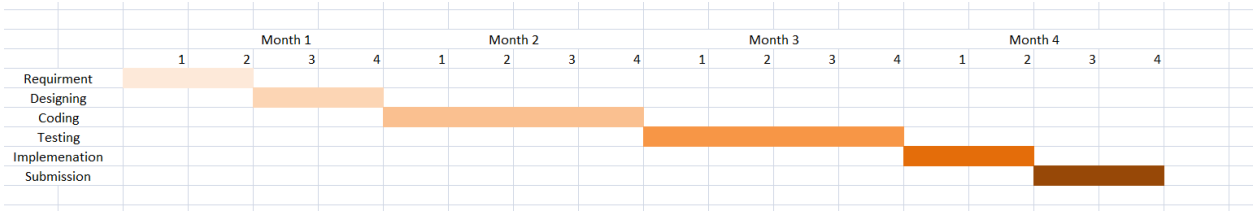
1.2. Identification of Problem

Learning by visualization has been shown to help improve learning ability. By providing a visual representation of what the destination node algorithms look like, applications aim to make them easier to understand.

1.3. Identification of Tasks

Define and differentiate the tasks required to build the visualize that can fulfil the purpose.

1.4. Timeline (Gantt chart)



CHAPTER 2.

LITERATURE REVIEW/BACKGROUND STUDY

2.1. Timeline of the reported problem

Cloning of the real website/application and offering the same level of user experience.

2.2. Proposed solutions

A fully functional project in which we can visualize various shortest path finding algorithm.

2.3. Bibliometric analysis

Analysis based on (key features, effectiveness and drawback)

2.4. Review Summary

Link findings with the project at hand.

2.5. Problem Definition

Define the problem at hand including what is to be done, how it is to be done and what not to be done

2.6. Goals/Objectives

Statements setting the milestones during the course of project work.
Keeping in mind

- Narrow, specific statements about what is to be learned and performed
- Precise intentions

CHAPTER 3.

DESIGN FLOW/PROCESS

3.1. Evaluation & Selection of Specifications/Features

Anumber of features that will allow users to see how path finding algorithm works actually.

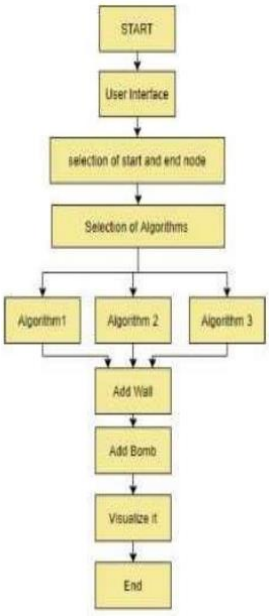
3.2.Analysis and Feature finalization subject to constraints

The project comprise of a start pointer and a end pointer that can be used to depict the desired path between them.

3.3. Design selection

The design consists of cell that is used to represent the nodes and path between them.

3.4. Implementation plan/methodology



CHAPTER 4.

RESULTS ANALYSIS AND VALIDATION

4.1. Implementation of solution

Use modern tools in:

- analysis,
- report preparation,
- project management, and communication,
- Testing/characterization/interpretation/data validation.

CHAPTER 5.

CONCLUSION AND FUTURE WORK

5.1. Conclusion

The pathfinding visualize is beneficial for both educations and to find the shortest path among all the path. It is best to understand the algorithm.

5.2. Future work

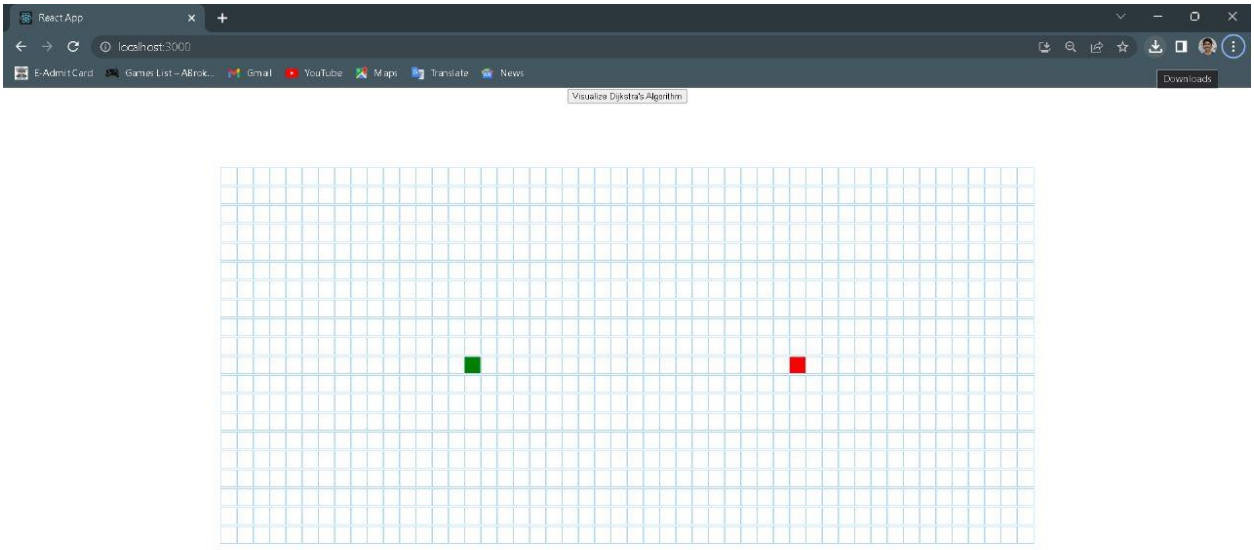
Future work of the pathfinding algorithm that we can create various puzzle games or we can also add all sorting algorithm to add various features in path finding visualzer.

CHAPTER 6.

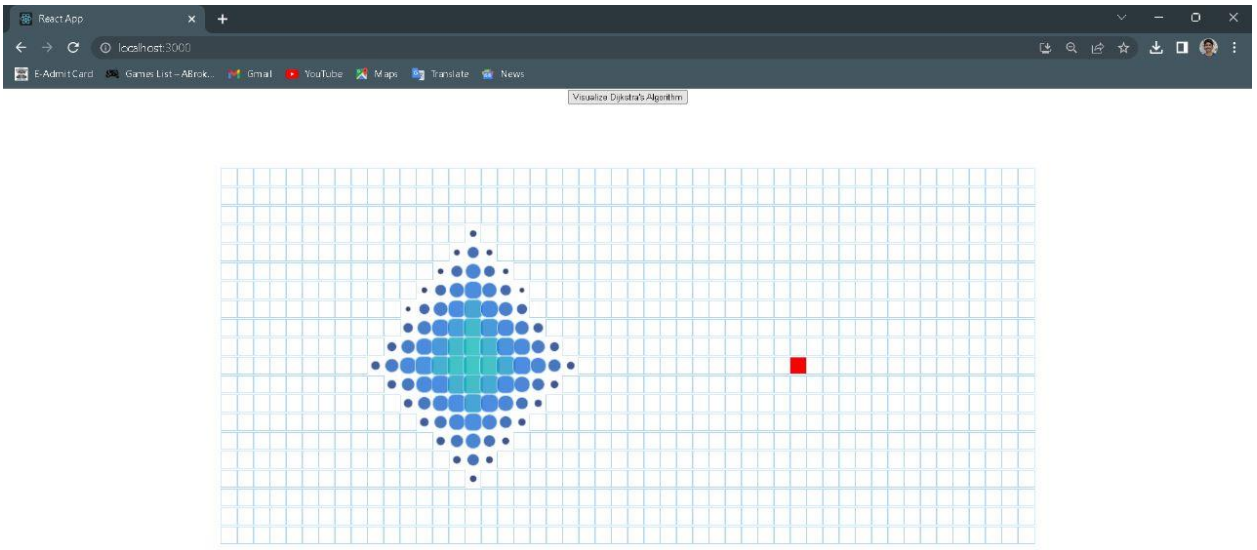
REFERENCES

1. Javatpoint
2. The Shortest-Path Problem: Analysis and Comparison of Methods by Hector Ortega-Arranz , Diego R. Llanos , Arturo Gonzalez-Escribano
3. ResearchGate- "Path Finding Algorithm Visualization"
4. GeeksforGeeks
5. HTML and CSS- w3schools

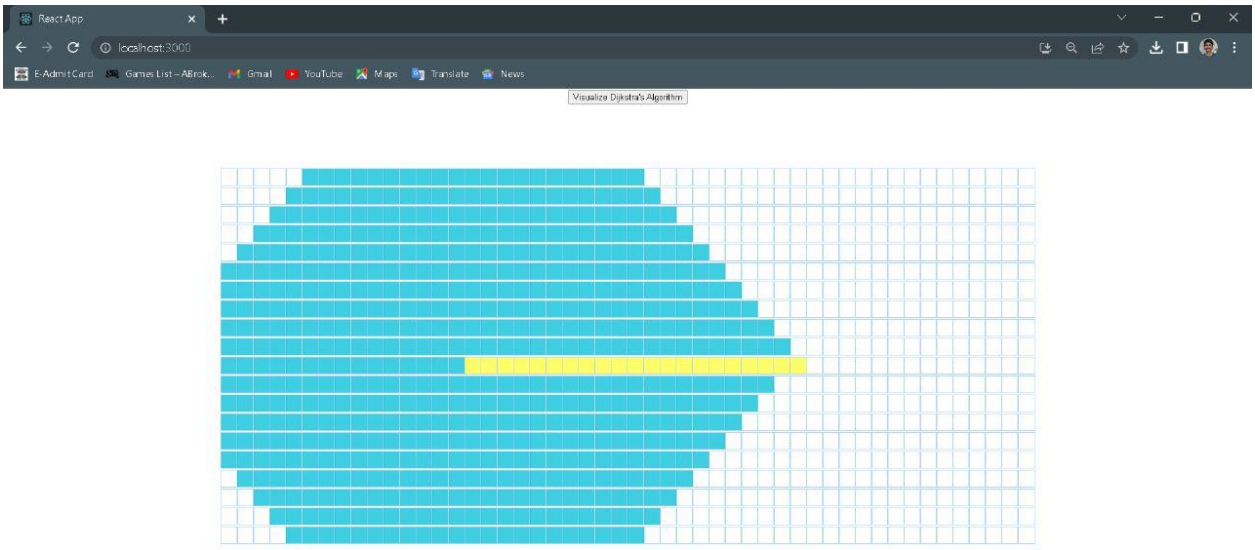
USER MANUAL



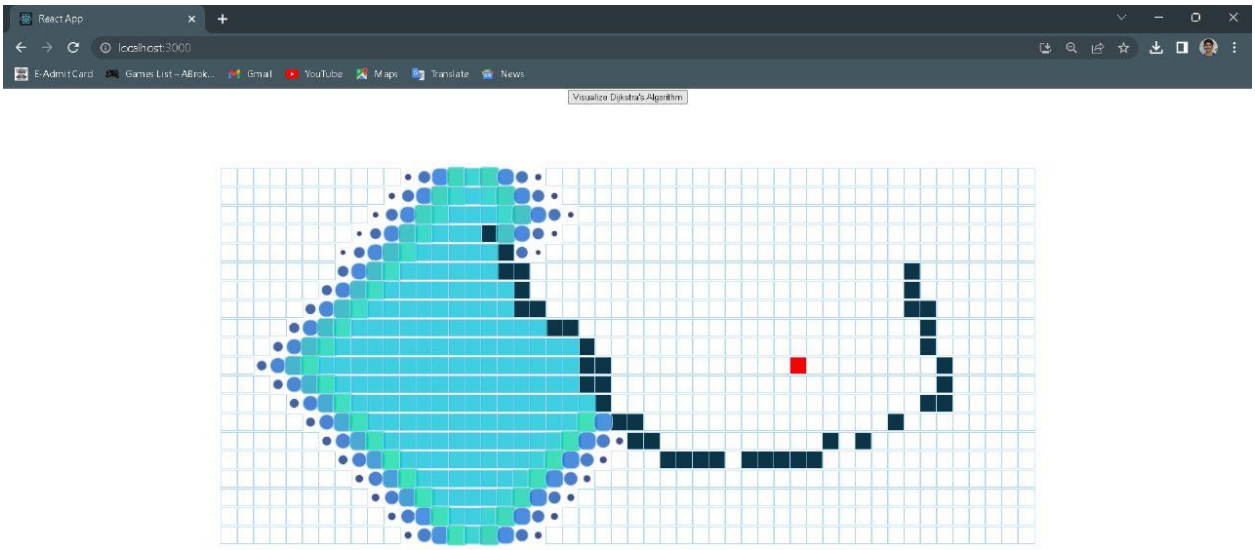
There a two nodes starting node(Green colour) and the ending node (red colour)



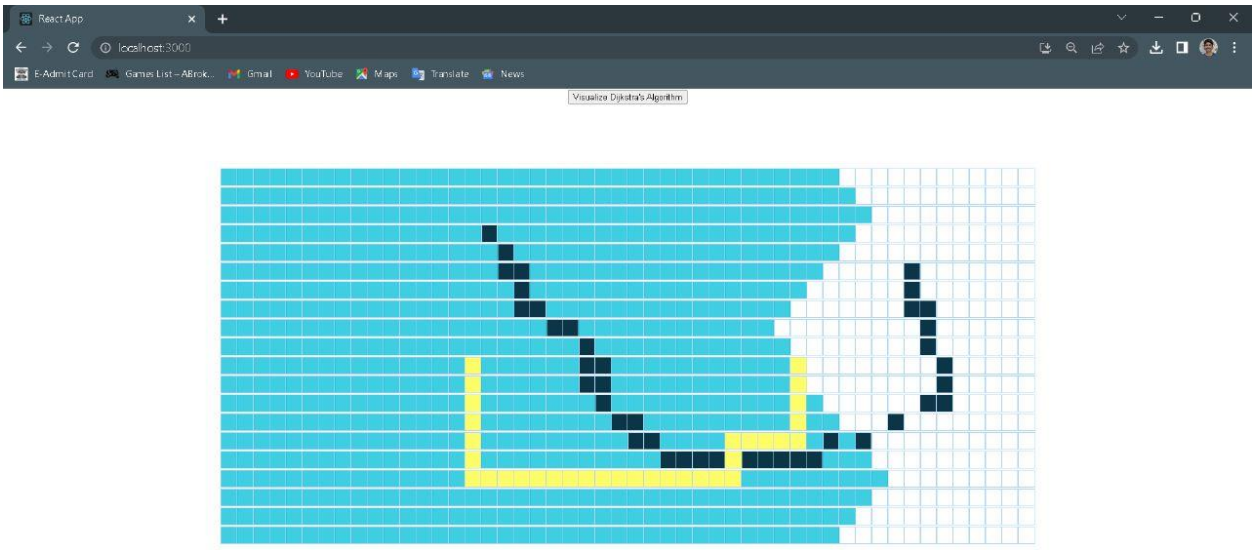
Finding path between the two nodes



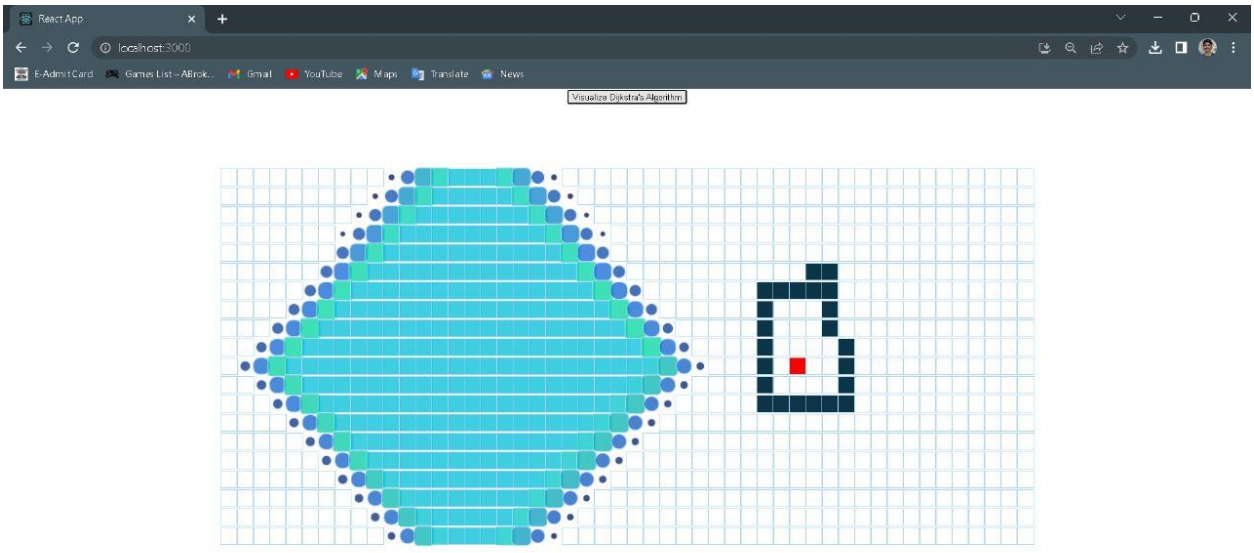
Path found (shown in yellow colour) between two nodes.

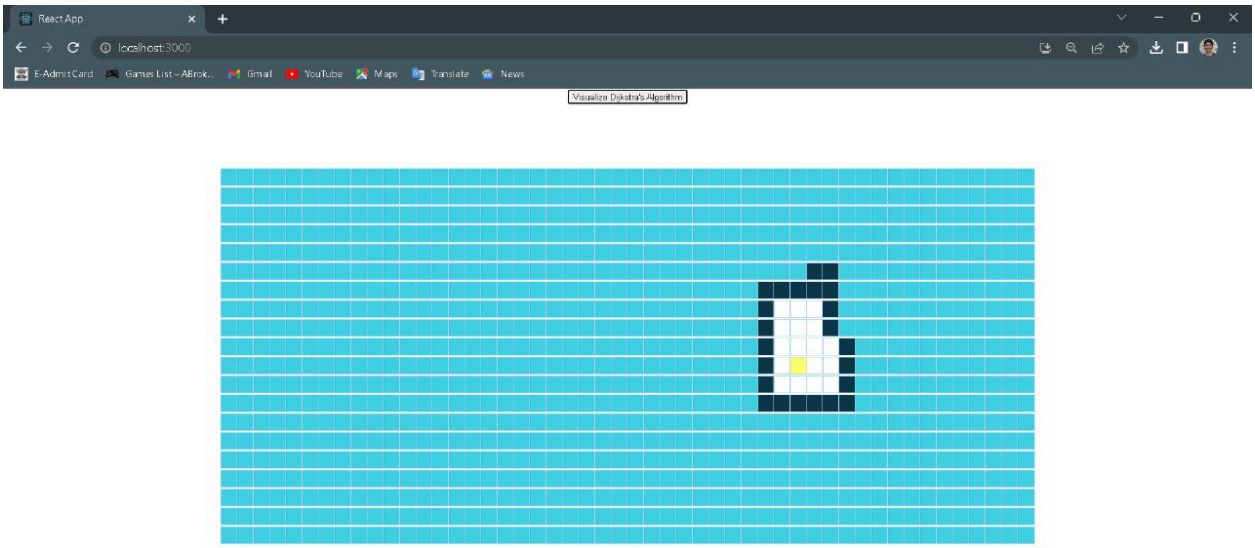


There are blocks between the starting node and the ending node.



Shortest path found between the nodes .





There is no path possible as there is no way possible to reach the ending node.

Source Code

1. DIJKSTRA'S ALGORITHM

```
export function dijkstra(grid, startNode,
finishNode)
{
const visitedNodesInOrder = [];
startNode.distance = 0;
 const unvisitedNodes = getAllNodes(grid);

  while (!!unvisitedNodes.length)
{
sortNodesByDistance(unvisitedNodes);
    const closestNode = unvisitedNodes.shift();

 if (closestNode.isWall) continue;

if (closestNode.distance === Infinity)
return visitedNodesInOrder;

closestNode.isVisited = true;
```

```
visitedNodesInOrder.push(closestNode);
      if (closestNode === finishNode) return
   visitedNodesInOrder;
updateUnvisitedNeighbors(closestNode, grid);
 }
}

  function sortNodesByDistance(unvisitedNodes)
{
unvisitedNodes.sort((nodeA, nodeB) => nodeA.distance
- nodeB.distance);
 }

  function updateUnvisitedNeighbors(node, grid)
 {
const unvisitedNeighbors =
getUnvisitedNeighbors(node, grid);

for (const neighbor of unvisitedNeighbors)
{
 neighbor.distance = node.distance + 1;
 neighbor.previousNode = node;

}
}
```

```
 function getUnvisitedNeighbors(node, grid)
{
const neighbors = [];
const {col, row} = node;

if (row > 0) neighbors.push(grid[row - 1][col]);
if (row < grid.length - 1) neighbors.push(grid[row +
1][col]);

if (col > 0) neighbors.push(grid[row][col - 1]);
 if (col < grid[0].length - 1)
neighbors.push(grid[row][col + 1]);

return neighbors.filter(neighbor =>
!neighbor.isVisited);
}

  function getAllNodes(grid)
{
 const nodes = [];

 for (const row of grid)
{
for (const node of row)
```

```
 {

nodes.push(node);
 }
}
   return nodes;
  }

 export function
getNodesInShortestPathOrder(finishNode)
{
const nodesInShortestPathOrder = [];
let currentNode = finishNode;

 while (currentNode !== null)
 {
nodesInShortestPathOrder.unshift(currentNode);
  currentNode = currentNode.previousNode;
  }
 return nodesInShortestPathOrder;
  }
```

2. PATH FINDING VISUALIZER

```
import React, {Component} from 'react';

import Node from './Node/Node';

import {dijkstra, getNodesInShortestPathOrder} from
'../algorithms/dijkstra';

import './PathfindingVisualizer.css';

const START_NODE_ROW = Math.floor(Math.random()*20);
const START_NODE_COL = Math.floor(Math.random()*30);
const FINISH_NODE_ROW = Math.floor(Math.random()*20);
const FINISH_NODE_COL = Math.floor(Math.random()*30);

export default class PathfindingVisualizer extends
Component
{

  constructor()
 {
```

```
    super();
    this.state =
{
      grid: [],
      mouseIsPressed: false,
    };
  }

  componentDidMount()
{
    const grid = getInitialGrid();
    this.setState({grid});
  }

  handleMouseDown(row, col)
 {
    const newGrid =
getNewGridWithWallToggled(this.state.grid, row, col);
    this.setState({grid: newGrid, mouseIsPressed:
true});
  }
```

```
 handleMouseEnter(row, col)
 {
    if (!this.state.mouseIsPressed)
return;

    const newGrid =
getNewGridWithWallToggled(this.state.grid, row, col);
    this.setState({grid: newGrid});
  }

  handleMouseUp()
{
    this.setState({mouseIsPressed: false});
  }

  animateDijkstra(visitedNodesInOrder,
nodesInShortestPathOrder)
 {
    for (let i = 0; i <= visitedNodesInOrder.length;
i++)

 {
      if (i === visitedNodesInOrder.length)
```

```
{
        setTimeout(() => {

this.animateShortestPath(nodesInShortestPathOrder);
        }, 10 * i);
        return;
      }
      setTimeout(() => {
        const node = visitedNodesInOrder[i];
        document.getElementById(`node-${node.row}-
${node.col}`).className =
          'node node-visited';
      }, 10 * i);
    }
  }

  animateShortestPath(nodesInShortestPathOrder)
{
    for (let i = 0; i <
nodesInShortestPathOrder.length; i++)
 {
      setTimeout(() => {
        const node = nodesInShortestPathOrder[i];
```

```
        document.getElementById(`node-${node.row}-
${node.col}`).className =
          'node node-shortest-path';
      }, 50 * i);
    }
  }

  visualizeDijkstra()
{
    const {grid} = this.state;
    const startNode =
grid[START_NODE_ROW][START_NODE_COL];
    const finishNode =
grid[FINISH_NODE_ROW][FINISH_NODE_COL];
    const visitedNodesInOrder = dijkstra(grid,
startNode, finishNode);
    const nodesInShortestPathOrder =
getNodesInShortestPathOrder(finishNode);
    this.animateDijkstra(visitedNodesInOrder,
nodesInShortestPathOrder);
  }

  render()
{
```

```
 const {grid, mouseIsPressed} = this.state;


    return (
      <>
        <button onClick={() =>
this.visualizeDijkstra()}>
          Visualize Dijkstra's Algorithm
        </button>
        <div className="grid">
          {
            grid.map((row, rowIdx) => {
            return (
              <div key={rowIdx}>
                {row.map((node, nodeIdx) => {
                  const {row, col, isFinish, isStart,
isWall} = node;
                  return (
                    <Node
                      key={nodeIdx}
                      col={col}
                      isFinish={isFinish}
                      isStart={isStart}
                      isWall={isWall}
```

```
 mouseIsPressed={mouseIsPressed}

                    onMouseDown={(row, col) =>
this.handleMouseDown(row, col)}
                    onMouseEnter={(row, col) =>
                    this.handleMouseEnter(row, col)
                      }
                      onMouseUp={() =>
this.handleMouseUp()}
                      row={row}>
                     </Node>
                  );
                })}
              </div>
            );
          })}
        </div>
      </>
    );
  }
}
```

```
const getInitialGrid = () => {
  const grid = [];

  for (let row = 0; row < 20; row++)
{
    const currentRow = [];
    for (let col = 0; col < 50; col++)
{
      currentRow.push(createNode(col, row));
}
    grid.push(currentRow);
  }
  return grid;
};

const createNode = (col, row) => {
  return
 {
    col,
    row,
```

```
 isStart: row === START_NODE_ROW && col === START_NODE_COL,
    isFinish: row === FINISH_NODE_ROW && col === FINISH_NODE_COL,
    distance: Infinity,
    isVisited: false,
    isWall: false,
    previousNode: null,
  };
};

const getNewGridWithWallToggled = (grid, row, col) => {
  const newGrid = grid.slice();
  const node = newGrid[row][col];
  const newNode = {
    ...node,
    isWall: !node.isWall,
  };
  newGrid[row][col] = newNode;
  return newGrid;
};
```

3.ANIMATION IN NODE

```
.node
{
    width: 25px;
    height: 25px;
    outline: 1px solid rgb(175, 216, 248);
    display: inline-block;
  }

  .node-finish
{
    background-color: red;
  }

  .node-start
{
    background-color: green;
  }

  .node-visited
{
    animation-name: visitedAnimation;
```

```
    animation-duration: 1.5s;
    animation-timing-function: ease-out;
    animation-delay: 0;
    animation-direction: alternate;
    animation-iteration-count: 1;
    animation-fill-mode: forwards;
    animation-play-state: running;
  }

  @keyframes visitedAnimation
 {
    0%
{
      transform: scale(0.3);
      background-color: rgba(0, 0, 66, 0.75);
      border-radius: 100%;
    }

    50%
{
      background-color: rgba(17, 104, 217, 0.75);
    }
```

```
    75%
{
      transform: scale(1.2);
      background-color: rgba(0, 217, 159, 0.75);
    }


    100%
 {
      transform: scale(1);
      background-color: rgba(0, 190, 218, 0.75);
    }
  }


  .node-wall
{
    background-color: rgb(12, 53, 71);
  }

  .node-shortest-path
{
```

```
    animation-name: shortestPath;
    animation-duration: 1.5s;
    animation-timing-function: ease-out;
    animation-delay: 0;
    animation-direction: alternate;
    animation-iteration-count: 1;
    animation-fill-mode: forwards;
    animation-play-state: running;

  }


 @keyframes shortestPath
{
    0%
{
      transform: scale(0.6);
      background-color: rgb(255, 254, 106);
    }

    50%
{
      transform: scale(1.2);
```

```
    background-color: rgb(255, 254, 106);
  }

  100%
{
    transform: scale(1);
    background-color: rgb(255, 254, 106);
  }
```

4.NODE.JS

```
import React, {Component} from 'react';

import './Node.css';

export default class Node extends Component
{
  render()
 {
    const
{
      col,
      isFinish,
      isStart,
      isWall,
      onMouseDown,
      onMouseEnter,
      onMouseUp,
      row,

} = this.props;
    const extraClassName = isFinish
```

```
    ? 'node-finish'
    : isStart
    ? 'node-start'
    : isWall
    ? 'node-wall'
    : '';

  return (
    <div
      id={`node-${row}-${col}`}
      className={`node ${extraClassName}`}
      onMouseDown={() => onMouseDown(row, col)}
      onMouseEnter={() => onMouseEnter(row, col)}
      onMouseUp={() => onMouseUp()}></div>
  );
}
```

5 .APP.CSS

```
.App
 {
    text-align: center;
  }

  .App-logo
{
    height: 40vmin;
  }

  .App-header
{
    background-color: #282c34;
    min-height: 100vh;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
```

```
font-size: calc(10px + 2vmin);
color: white;
  }

  .App-link
{
    color: #09d3ac;
  }
```

6. APP.JS

```
import React from 'react';

import './App.css';

import PathfindingVisualizer from
'./PathfindingVisualizer/PathfindingVisualizer';

function App()
 {
  return (
    <div className="App">
      <PathfindingVisualizer>
</PathfindingVisualizer>
    </div>
  );
}

export default App;
```

7. INDEX.CSS

```
body
{
    margin: 0;
    font-family: -apple-system, BlinkMacSystemFont,
"Segoe UI", "Roboto", "Oxygen",
      "Ubuntu", "Cantarell", "Fira Sans", "Droid
Sans", "Helvetica Neue",
      sans-serif;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
  }


  code
{
    font-family: source-code-pro, Menlo, Monaco,
Consolas, "Courier New",
      monospace;
  }
```

8. INDEX.JS

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(<App />,
document.getElementById('root'));

serviceWorker.unregister();
```

9. SERVICEWORKER.JS

```
const isLocalhost = Boolean(
    window.location.hostname === 'localhost' ||
      window.location.hostname === '[::1]' ||
      window.location.hostname.match(
        /^127(?:\.(?:25[0-5]|2[0-4][0-9]|[01]?[0-
9][0-9]?)){3}$/
      )
  );
  export function register(config)
{
    if (process.env.NODE_ENV === 'production' &&
'serviceWorker' in navigator)
{
      const publicUrl = new
URL(process.env.PUBLIC_URL, window.location.href);

      if (publicUrl.origin !==
window.location.origin)
{
        return;
      }
```

```
      window.addEventListener('load', () => {
        const swUrl =
`${process.env.PUBLIC_URL}/service-worker.js`;

        if (isLocalhost)
 {
          checkValidServiceWorker(swUrl, config);

          navigator.serviceWorker.ready.then(() =>
{
            console.log('This web app is being
served cache-first by a service ' +'worker.  );
          });
        }
else
{
     registerValidSW(swUrl, config);
        }
      });
    }
  }

  function registerValidSW(swUrl, config)
{
```

```
navigator.serviceWorker
      .register(swUrl)
      .then(registration => {
        registration.onupdatefound = () => {
          const installingWorker =
registration.installing;

          if (installingWorker == null)
 {
            return;
          }
          installingWorker.onstatechange = () => {
        if (installingWorker.state === 'installed')
{
   if (navigator.serviceWorker.controller)
 {
console.log('New content is available and will be
used when all ' +'tabs for this page are closed.');

if (config && config.onUpdate)
{
   config.onUpdate(registration);
                }
```

```
              }
 else
{
 console.log('Content is cached for offline use.');
                if (config && config.onSuccess)
{
                  config.onSuccess(registration);
                }
              }
            }
          };
        };
      })
  .catch(error => {
        console.error('Error during service worker
registration:', error);
      });
  }

  function checkValidServiceWorker(swUrl, config)
 {
      fetch(swUrl)
      .then(response => {
```

```
  const contentType = response.headers.get('content-
  type');


if(response.status === 404 || (contentType != null &&
contentType.indexOf('javascript') === -1))
  {

  navigator.serviceWorker.ready.then(registration =>
  {
              registration.unregister().then(()
  => {
                window.location.reload();
              });
            });
      }
   else
  {
   normal.
            registerValidSW(swUrl, config);
          }
        })
        .catch(() => {
          console.log(
```

```
          'No internet connection found. App is
running in offline mode.'
        );
      });
  }

  export function unregister()
{
    if ('serviceWorker' in navigator)
{

navigator.serviceWorker.ready.then(registration =>
{
        registration.unregister();
      });
    }
  }
```