A PROJECT REPORT

ON

Directional Navigation Algorithm Comparison

BY

1. Keshav Sharma

2. Pradyum Shetty

3. Rajith Shetty

Under the guidance of

Prof. Savita Lade.

MANJARA CHARITABLE TRUST

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

Department of Computer Engineering
University of Mumbai

May - 2022

MANJARA CHARITABLE TRUST

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

Juhu-Versova Link Road Versova, Andheri(W), Mumbai-53

C E R T I F I C A T E

Department of Computer Engineering

This is to certify that

1. Keshav Sharma
2. Pradyum Shetty
3. Rajith Shetty

Have satisfactorily completed this project entitled

Directional Navigation Algorithms Comparison

Towards the partial fulfilment of the

BACHELOR OF ENGINEERING
IN
(COMPUTER ENGINEERING)

as laid by the University of Mumbai.

Guide

Prof. Savita Lade

H.O.D.

Prof. S. P. Khachane

Principal
Dr Sanjay Bokade

Project Report Approval for B. E.

This project report entitled ”**Directional Navigation Algorithm Comparison**” by **Keshav Sharma, Pradyum Shetty and Rajith Shetty** is approved for the degree of Computer Engineering.

Examiners:

1----------

2.----------

Date:

Place: Mumbai

Declaration

We wish to state that the work embodied in this project titled **"Directional Navigation Algorithm Comparison"** forms our own contribution to the work carried out under the guidance of "**Prof Savita Lade**" at the Rajiv Gandhi Institute of Technology.

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Keshav Sharma (B-833) -

Pradyum Shetty (B-837) -

Rajith Shetty (B-838) -

Abstract

The directional navigation algorithm comparison project is a comparison of various algorithms to find the shortest path between two points, visually. This project has a lot of scope in the future to turn it into proper software and deploy it over various operating systems for users to use for indoor navigation. The start is to visually show the working of the various algorithms and expand the project from there.

Apps such as Google maps have inevitably become the go-to app for travelling & we are all heavily reliant on them. But as developers can the ones starting in this field grasp the concept of these algorithms easily? Being able to use such navigational tools outdoors has been very helpful but the same cannot be applied indoors everywhere. But understanding the core concepts & which algorithm will perform how in different situations can bring about a big change. In this project we plan to help the end-user (developers, students) visually study the search algorithms for the shortest distance between two points and how it is calculated by the various algorithms, thus being able to pick the best one possible. One of the main reasons to add visualizations is that we, humans, are better visual learners.

On top of the pathfinding algorithms, we will also be implementing a Recursive Division Maze Generation algorithm as a major part of the future scope of our project to test various algorithms with ease.

These are the parameters to compare the various algorithms,
1. Time taken

2. Structure of branching & grid mapping

Contents

‘

List Of Figures

List Of Tables

List Of Algorithms

CHAPTER 1

Introduction

1.1 Introduction Description

In today's world navigating and travelling from one place to another using an app for directions has become the de facto and with various technologies and algorithms available, new developers can often get confused about which algorithm is the best one possible among all that are available.

There are various trial & error-based approaches available but all of them lack the visuals of how an algorithm traces the path. In this project we plan to help the end-user visualise the shortest distance between two points and how it is calculated by the various algorithms, thus being able to pick the best one possible. By making it visual and dynamic i.e letting users be in control of where the obstacles are placed and which algorithm is being visualised, we make learning and understanding the logic behind the code easy & faster than just reading it.

There is a list of 6 algorithms, to begin with, the users select their choice of algorithm to visualise, they then create a start point & endpoint to run the algorithm for & add obstacles between the two points. We utilize the power of react to enable this on the backend. The main goal of this research is to create a GUI version that will be used to see various algorithms mapping the shortest route in real-time between two points.

1.2 Organization of report

- **Ch.1 Introduction:** This chapter consists of an overview and introduction to the overall project and what we are aiming to achieve with it.

- **Ch.2 Literature Review:** This chapter takes you through the various existing implementations research papers used and highlights the limitations in those and what problem we are aiming to solve.

- **Ch.3 Proposed System:** This details our proposed implementation of the solution.

- **Ch.4 Results & Discussion:** In this section, we will discuss the result such as the UI and the learnings from various algorithms and compare them with each other to understand and determine the relatively fastest/shortest path for an algorithm.

- **Ch.5 Conclusion & Future Work:** This section has the future roadmap for the project.

CHAPTER 2

Literature Review

A lot of research has been done in the field of algorithm comparisons & algorithm study. Many people have developed various systems & parameters to compare algorithms to determine the best amongst the list. We have studied some of the systems and some key points to refer to & realised that algorithm visualization has some major ideas that can be implemented with a better approach.

2.1 Survey the existing system

In the research paper by Lingling Zhao, Juan Zhao: Comparison Study of Three Shortest Path Algorithm 2019 [1] the algorithm procedure of three common shortest path algorithms has been introduced in detail, i.e. Dijkstra, Floyd, and Bellman-Ford. Through testing case diagrams, it describes the execution steps of the three algorithms. From spatial complexity, time complexity, application range and negative weight (negative weight value), this paper compares the three algorithms to conclude.

Foundations of Algorithms, Fifth Edition [3] offers a well-balanced presentation of algorithm design, complexity analysis of algorithms, and computational complexity. Each chapter presents an algorithm, a design technique, an application area, or a related topic. Yong Deng, Yuxin Chen and Yajuan Zhang in the paper Fuzzy Dijkstra algorithm for shortest path problem under uncertain environment [10], introduced a generalized Dijkstra algorithm to handle SPP in an uncertain environment. Two key issues need to be addressed in SPP with fuzzy parameters. One is how to determine the addition of two edges. The other is how to compare the distance between two different paths with their edge lengths represented by fuzzy numbers. To solve these problems, the graded mean integration representation of fuzzy numbers is adopted to improve the classical Dijkstra algorithm.

In 2015 Louis Boguchwal published a paper on Shortest path algorithms for functional environments [7] that generalises classic shortest path algorithms to network environments in which arc costs are governed by functions, rather than fixed weights. He makes use of algorithms based on monotonicity which improves the results obtained since Knuth in 1976, which requires several restrictive assumptions on the functions permitted in the network.

2.2 Limitations existing system or Research gap

We found the following limitations in the research papers & references,

- No visual explanation.
- Limited to particular algorithms.
- Inefficient comparisons i.e Comparisons based on just a few aspects of various algorithms.
- Limitations in terms of algorithms being compared.
- No interface input system & output is a simple matrix
- No dynamic comparison in traversing in different situations.

2.3 Problem Statement and Objectives

The purpose of this project is to take two points as inputs from the user, get the algorithm to be visualised by the user, find the shortest distance between the two points by visually traversing the grid using the logic & mapping techniques of the algorithm & decide the best algorithm for the given route. We are working with various algorithms & mapping techniques to accurately showcase the traversing of the grid.

2.3.1 Objectives

- To make learning more visual
- Aid end-users to see how different algorithms work on the backend
- Develop an end-to-end software application for external usage and contributions

2.4 Scope

This software application can be utilized for indoor navigation inside malls, and airports for easier travelling within such closed quarters.

Due to the visualising nature of the algorithms, students can learn & truly understand the functioning of an algorithm via this application & use the algorithms more accurately wherever needed.

CHAPTER 3

Proposed System

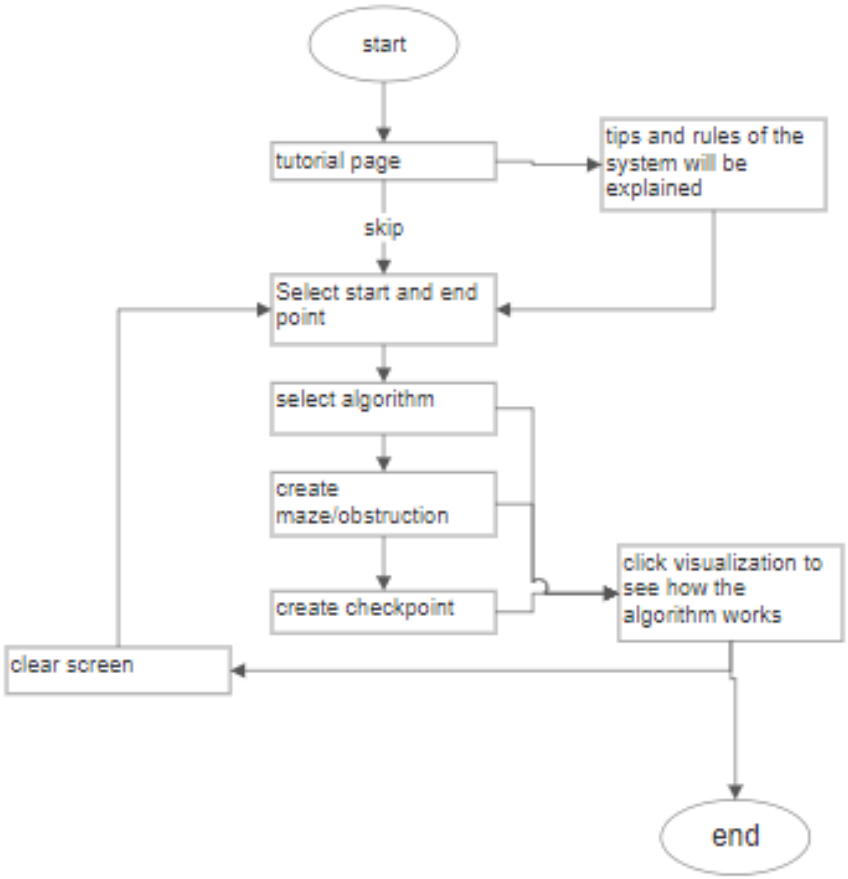

Fig. 3.1 System flow/ System Architecture

As we can see in Figure 3.1 the use system flow displays the general flow of the user using the system and its inner processing is displayed. Every single stage is mentioned where the components are contributing to each other the inputs and outputs. Every time the machine executes a task, this process is maintained and executed.

This system is proposed to enable students to be able to learn & understand algorithms in an easier & developer-friendly manner whilst keeping their needs as a top priority.

3.1 Algorithms

Here is a detailed explanation of the algorithms and frameworks we are going to be using in our project.

A search*

The A* search algorithm is generally regarded as the *de facto* standard in-game pathfinding. It was first described in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael.
For every node in the graph, A* maintains three values: *f(x)*, *g(x)*, and *h(x)*. *g(x)* is the distance, or cost, from the initial node to the node currently being examined. *h(x)* is an estimate or heuristic distance from the node being examined to the target.

In pseudocode:
Step1: Place the starting node in the OPEN list.
Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stop.
Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is the goal node then return success and stop, otherwise
Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute the evaluation function for n' and place it into the Open list.
Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.
Step 6: Return to Step 2.

Dijkstra's Algorithm

Named for its creator, Edsger Dijkstra, Dijkstra's algorithm was first proposed in 1959 and is the immediate precursor of A*.
The basic process is to assign each node a distance value, at first set to zero for the initial node and infinity for all other nodes. All nodes are marked as unvisited and the initial, node is marked as the current node.
In pseudocode:
Step 1: Initialization of all nodes with distance "infinite"; initialization of the starting node with 0
Step 2: Marking the distance of the starting node as permanent, all other distances as temporarily.
Step 3: Setting of starting node as active.
Step 4: Calculation of the temporary distances of all neighbour nodes of the active node by summing up its distance with the weights of the edges.
Step 5: If such a calculated distance of a node is smaller than the current one, update the distance and set the current node as an antecessor. This step is also called update and is Dijkstra's central idea.
Step 6: Setting the node with the minimal temporary distance as active. Mark its distance as permanent.
Step 7: Repeating steps 4 to 7 until there aren't any nodes left with a permanent distance, which neighbours still have temporary distances.

Bidirectional algorithm

Bidirectional search is a graph search algorithm that finds the shortest path from an initial vertex to a goal vertex in a directed graph. It runs two simultaneous searches: one forward from the initial state, and one backwards from the goal, stopping when the two meet.

In pseudocode:

parent=> Array where startparent[i] is parent of node i

visited=> Array where visited[i]=True if node i has been encountered

Step1: while the start node is not empty and the end is not empty

Step 2: perform the next iteration of BFS for the start node (also save the parent of children in the parent array)

Step 3: Perform the next iteration of BFS for the end node

Step 4: if we have encountered the intersection node

save the intersection node

Step 5: Using the intersection node, find the path using the parent array

Greedy BFS

In BFS and DFS, when we are at a node, we can consider any of the adjacent as the next node. So both BFS and DFS blindly explore paths without considering any cost function. The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. Best First Search falls under the category of Heuristic Search or Informed Search.

In pseudocode:

Step 1: Place the starting node into the OPEN list.

Step 2: If the OPEN list is empty, Stop and return failure.

Step 3: Remove the node n, from the OPEN list which has the lowest value of h(n), and place it in the CLOSED list.

Step 4: Expand the node n, and generate the successors of node n.

Step 5: Check each successor of node n, and find whether any node is a goal node or not. If any successor node is the goal node, then return success and terminate the search, else proceed to Step 6.

Step 6: For each successor node, the algorithm checks for evaluation function f(n), and then adds it to the OPEN list if it has not been in either open or close list.

Step 7: Return to Step 2.

Breadth-First Search (BFS) Algorithm

Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighbouring nodes. Then, it selects the nearest node and explores all the unexplored nodes. The algorithm follows the same process for each of the nearest nodes until it finds the goal.

In pseudocode:

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting for state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

Step 6: EXIT

Depth First Search (DFS) Algorithm

Depth-first search (DFS) algorithm starts with the initial node of the graph G and then goes deeper and deeper until we find the goal node or the node which has no children. The algorithm then backtracks from the dead-end towards the most recent node that is yet to be completely unexplored.

In pseudocode:

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting for state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

Step 6: EXIT

3.2 Details of hardware and software

3.2.1 Hardware requirements

No Hardware requirement

3.2.2 Software requirements

- React, Visual Studio Code, Java
- Operating System: Windows 7, Windows 8, Windows, Linux and Mac are compatible.
- Browser: internet explorer, chrome firefox and safari.

3.3 Design Details

3.3.1 Detailed Design

3.3.1 (a) Use - Case Diagram

Figure 3.2 shows the Use case diagram of the system highlighting the different options & actions the user can perform via the project.

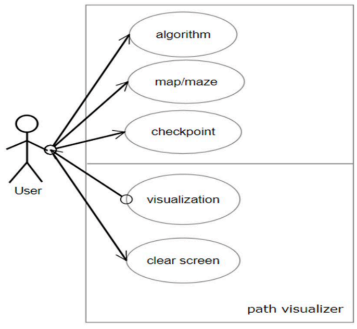

Fig 3.2: Use case diagram

3.3.1 (b) State Diagram

Figure 3.3 shows the state transition diagram of the system highlighting the different state transitions taking place.

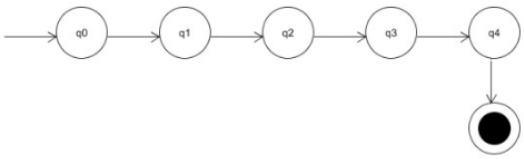

Fig 3.3: State Transition Diagram

q0= Selecting algorithm
q1= Selecting speed
q2= Selecting Maze creation algorithm
q3= Visualizing the algorithm search
q4= Showing the Shortest Path
q5= Close

3.3.1 (c) Sequence Diagram

Figure 3.4 shows the sequence diagram of the system highlighting the order of actions performed in the project.

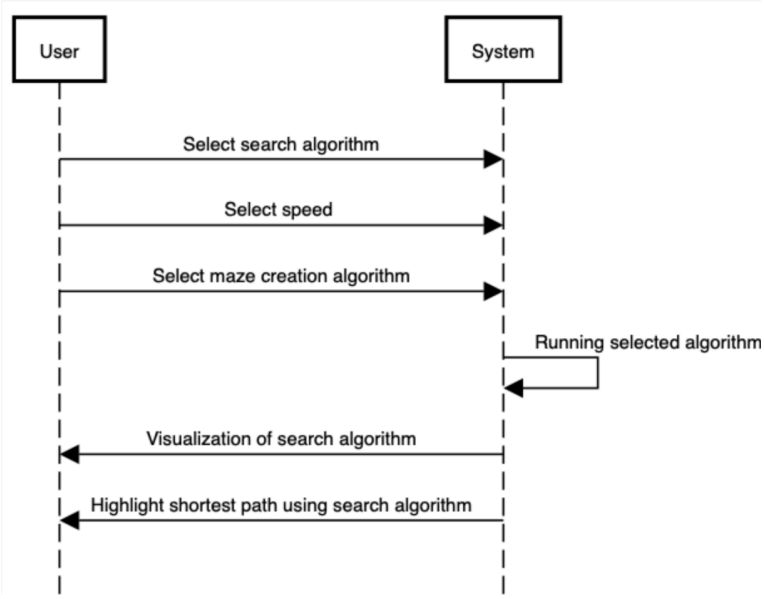

Fig 3.4: Sequence Diagram

3.4 Methodology/Procedure

Procedure

This section is subdivided into 5 parts. Our Project can be understood by the following 5 subdivisions,

1. The GUI
2. Maze Creation
3. Map Tracing
4. Use cases
5. Data set

1) Understanding the GUI

Machine The GUI consists of various elements such as the navbar highlighting the various feature list which users can use to input data & get the relevant results. This grid will, later on, be replaced with a map structure showcasing the indoor layout of an area.
Here users can add two points, add obstacles as per need & run various algorithms to compare the faster one & see the most efficient algorithm of all. This current grid is a 2d array of objects. Here every object has properties such as a row, column, is_start, is_finish, is_visited, and is_wall to form the grid, locate the points & traverse the grid from one point to another using the logic of the particular algorithm.

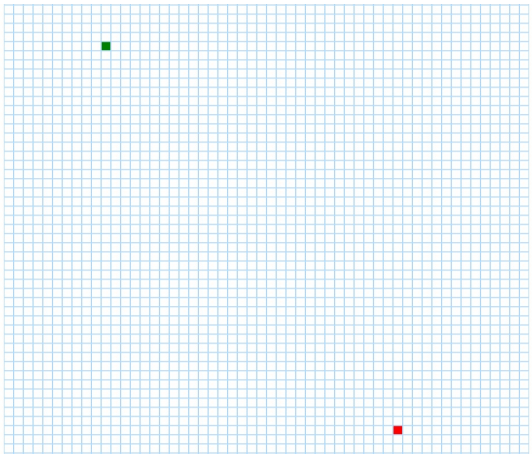

Fig. 3.5: The grid

This grid can be filled in 2 ways, randomised mazes or tracing real-world locations via google maps to find the shortest distance & comparing the algorithms. This grid is designed to be simple & direct.

2) Maze Creation

We have used various mapping techniques to generate random mazes to map the grid after an algorithm is set. There are three techniques used, which are as follows,

Vertical mapping - Here the grid is being traced in alternate columns from top to down except for random openings to facilitate the search algorithm traversing.

Horizontal mapping - Here the grid is being traced in alternate rows from left to right except for random openings to facilitate the search algorithm traversing.

Recursive mapping - In this mapping, we create a maze by recursively creating horizontal and vertical lines leaving a single space in between while traversing

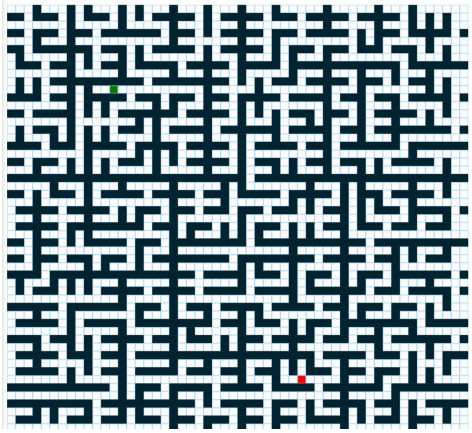

Fig 3.6: A maze created on the grid

3) Map Tracing Technique

We have used various screenshots of localities from google maps & using various image recognition & object detection tools we have converted these localities into a format recognisable via our code & formed the grid. This forms the basic layout for measuring & comparing the different algorithms based on the traverse path, time taken & the shortest route chosen by the particular algorithm.

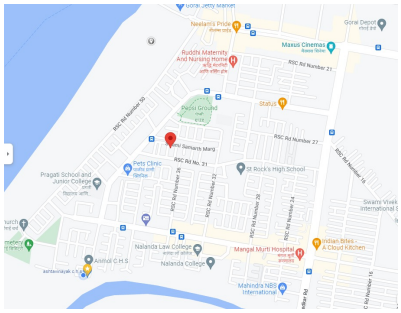

Fig 3.7: Image of a locality on Google map

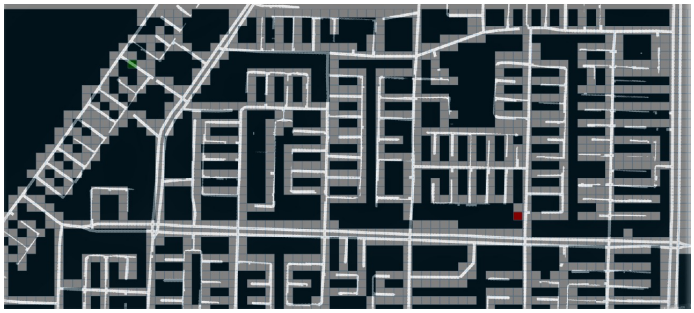

Fig 3.8: Tracings of the same locality on the Grid

4) Use cases

Figure 3.9 shows the Use case diagram of the system highlighting the different options & actions the user can perform via the project.

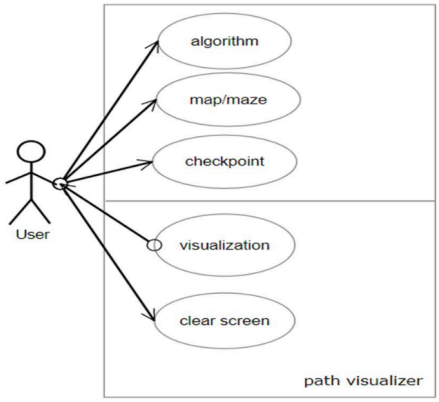

Fig 3.9: Use cases

5) Data set

By using google maps to locate nearby localities & tracing those areas using the map tracing technique we can create a vast database of maps to run these algorithms on to compare their efficiency. Apart from the outdoor maps, using various repositories & publicly available information, indoor maps can also be generated for such use cases.

Fig 3.10: Outdoor & Indoor mappings

CHAPTER 4

Results and Discussion

The results are promising. Every algorithm flows differently as opposed to our current understanding based on the conversation with a few developers. Adding obstacles makes the flow much more interesting to see the shortest ro

4.1 Implemented pseudo code

```
function isClass(type) {
  // React.Component subclasses have this flag

  return (
    Boolean(type.prototype) &&
    Boolean(type.prototype.isReactComponent)
  );
}

// This function only handles elements with a composite type.
// For example, it handles <App /> and <Button />, but not a
<div />.

function mountComposite(element) {
  var type = element.type;
  var props = element.props;

  var renderedElement;
  if (isClass(type)) {

    // Component class
    var publicInstance = new type(props);

    // Set the props
    publicInstance.props = props;
    // Call the lifecycle if necessary

    if (publicInstance.componentWillMount) {
      publicInstance.componentWillMount();
    }
    renderedElement = publicInstance.render();
  } else if (typeof type === 'function') {
    // Component function
```

```
    renderedElement = type(props);
  }

  // This is recursive but we'll eventually reach the bottom of
recursion when
  // the element is host (e.g. <div />) rather than composite (e.g.
<App />):

  return mount(renderedElement);
}

// This function only handles elements with a host type.
// For example, it handles <div /> and <p /> but not an <App />.

function mountHost(element) {
  var type = element.type;
  var props = element.props;
  var children = props.children || [];
  if (!Array.isArray(children)) {
    children = [children];
  }
  children = children.filter(Boolean);

  // This block of code shouldn't be in the reconciler.
  // Different renderers might initialize nodes differently.
  // For example, React Native would create iOS or Android views.

  var node = document.createElement(type);
  Object.keys(props).forEach(propName => {
    if (propName !== 'children') {
      node.setAttribute(propName, props[propName]);
    }
  });

  // Mount the children
  children.forEach(childElement => {

    // Children may be host (e.g. <div />) or composite (e.g.
<Button />).
    // We will also mount them recursively:
    var childNode = mount(childElement);

    // This line of code is also renderer-specific.
    // It would be different depending on the renderer:
    node.appendChild(childNode);
  });
```

```
  // Return the DOM node as mount result.
  // This is where the recursion ends.
  return node;
}

function mount(element) {
  var type = element.type;
  if (typeof type === 'function') {
    // User-defined components
    return mountComposite(element);
  } else if (typeof type === 'string') {
    // Platform-specific components
    return mountHost(element);
  }
}

var rootEl = document.getElementById('root');
var node = mount(<App />);
rootEl.appendChild(node);
```

4.2 Results

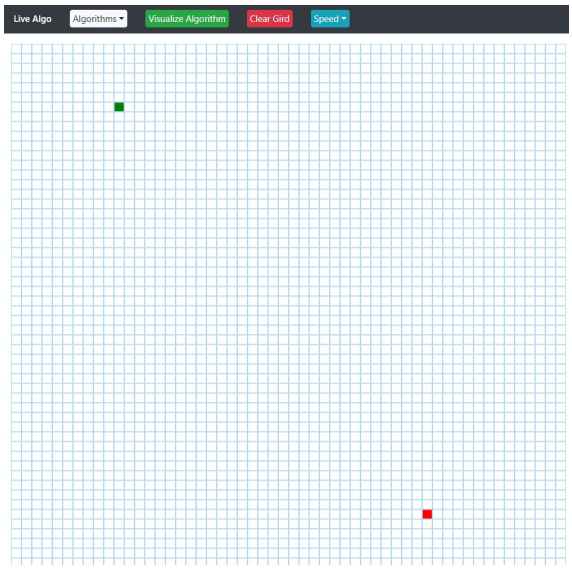

Fig. 4.1 The resulting grid

The results are promising. Every algorithm flows differently as opposed to our current understanding based on the conversation with a few developers. Adding obstacles makes the flow much more interesting to see the shortest route between two points. Using the traced area from google maps, we compared Dijktra's & Dept first search algorithms for the shortest route from Point A to Point B.

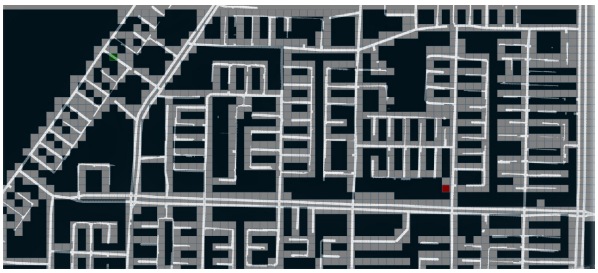

Fig 4.2: 2D Mapping of a nearby locality

We found the results to be accurate by cross-checking the theoretical values of these algorithms with our obtained results. The accuracy obtained from the application makes it suitable for most practical purposes, Although the accuracy can be improved.

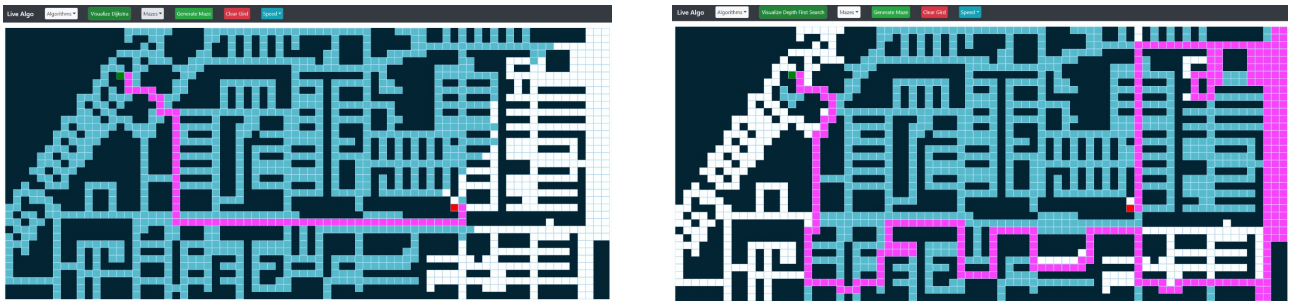

Fig 4.3: Input and Output of Algorithm comparison

4.3 Discussions

Here we discuss the comparative theoretical study of these algorithms.

Algorithm	Time Complexity	Space complexity
A* Search	O(|E|+|V|)	O(bd)
Dijkstra's Algorithm	O(ELogV)	o(v)
Bi-directional Search	O(bd/2)	O(bd/2)
Greedy Best-first Search	O(n*logn)	O (b m).
Breadth-first Search	$O(V^2)$	O (b d)
Depth-first Search	O(E log(V)) to O(V+E)	O (V + E) O(V +E)

Table 4.1 Comparative Study of all algorithms

CHAPTER 5

CONCLUSION

Thus, our project will take starting, ending point with the prefered choice of algorithm & a maze for the grid(optional) as an input, process the algorithm based on its' fundamental operating principles & methodology, then the grid will showcase the pattern on the screen via blocks of the grid being covered highlighting how the algorithm flows. In this project, we have presented the process of production and integration of software engineering & visualisation of various algorithms. This algorithm at its very primitive level will help users understand the flow of an algorithm much better & thus will help in the creation of an in-door navigational system.

Further work will include increasing the number of algorithms, making the map tracing method automated and opening it up for other developers to contribute and students to learn from. We will also be adding more complex maze/pattern generation algorithms, a feature to map any indoor area for quick navigation that can be used by travellers in-store, at airports and so on.

REFERENCES

[1] Lingling Zhao, Juan Zhao, “Comparison Study of Three Shortest Path Algorithm”, IEEE, August 2019.

[2] Xie Luyun, Algorithm. 4th ed. People’s posts and telecommunications press: Publishing house, 2012.

[3] Richard E.Neapolitan, “Foundations of Algorithmics. 5th edition”, People's posts and telecommunications press, Jones and Bartlett Publishers, 2016.

[4] Huilin Yuan, Jianlu Hu, Yufan Song, “A new exact algorithm for the shortest path problem: An optimized shortest distance matrix”, Science Direct, May 2021.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, “Introduction to Algorithms. 3rd edition”, Mechanical Industry Press: MIT Press, 2013.

[6] Pedro Maristany, Antonio Sedeño-Noda, Ralf Borndörfer, “An Improved Multi-objective Shortest Path Algorithm”, Science Direct, June 2021.

[7] Louis Boguchwal,” Shortest path algorithms for functional environments”, Science Direct, November 2015.

[8] Mark Allen Weiss, “Data Structure and Algorithm Analysis C Language Description 1st edition”, Mechanical Industry Press: Pearson, 2004.

[9] Anany Levitin, “Introduction to Design and Analysis of Algorithm. 3rd Edition”, Villanova University Press: Pearson, 2015.

[10] Yong Deng, Yuxin Chen, Yajuan Zhang, “Fuzzy Dijkstra algorithm for shortest path problem under uncertain environment”, Science Direct, November 2011.

[11] Basics of Greedy Algorithms Tutorials & Notes | Algorithms

Publication by Students

Paper entitled “Directional Navigational Algorithm” is presented at International Journal for Research in Applied Science & Engineering Technology (IJRASET) in March 2022.

Annexure

Acknowledgement

We wish to express our sincere gratitude to **Dr Sanjay U. Bokade, Principal** and Prof. S.P.Khachane, H.O.D of Department of Computer Engineering of Rajiv Gandhi Institute of Technology for providing us with an opportunity to do our project work on "Directional Navigational Algorithm Comparison".

This project bears on the imprint of many peoples. We sincerely thank our project guide, Prof. Savita Lade, for his guidance and encouragement in carrying out this synopsis work.

Finally, we would like to thank our colleagues and friends who helped us in completing the project work successfully

1. Keshav Sharma
2. Pradyum Shetty
3. Rajith Shetty