iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 10 **Issue:** III **Month of publication:** March 2022

DOI: https://doi.org/10.22214/ijraset.2022.41117

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue III Mar 2022- Available at www.ijraset.com

Directional Search Algorithm Analysis

Keshav Sharma[1], Pradyum Shetty[2], Rajith Shetty[3], Prof. Savita Lade[4]
[1, 2, 3]*UG Student,* [4]*Assistant Professor, Department of Computer Engineering, Rajiv Gandhi Institute of Technology-University of Mumbai*

Abstract: ***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.***
Keywords: Navigation, Directional Algorithms, GUI, Shortest distance,

I. INTRODUCTION

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 5 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. By adding more obstacles users can truly see how different algorithms perform in different situations. One of the main reasons to add visualizations is that we, humans, are better visual learners. 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 even if there are obstacles in between. The system basically flows in a set of steps/stages as shown in Fig 1.

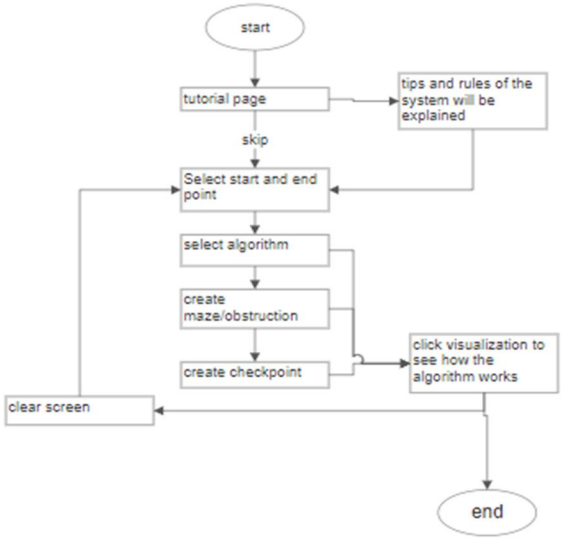


Fig 1: System Flow

Section II briefly explains the past work done in this particular discipline. Sections II and III delineate the methodology & algorithms used in the comparison, and the results obtained for each. Finally, Section V presents our conclusions and briefly illustrates the potential for further improvements in our methodology.

II. STATE OF ART

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 tried to implement their limitations in our project.

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 (value), this paper compares the three algorithms to draw a conclusion.

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. Existing Systems don't support the use of GUI and also don't support the analysis in real-world navigation.

III. METHODOLOGY

This section is subdivided into 6 parts. Our Project can be understood by the following 6 subdivisions,

1) Algorithms & pseudocode.
2) The GUI
3) Maze Creation
4) Map Tracing
5) Use case diagram
6) Data set

A. *Understanding the GUI*

1) *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 stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is 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 evaluation function for n' and place into 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.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue III Mar 2022- Available at www.ijraset.com

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 as 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 of the node with the minimal temporary distance as active. Mark its distance as permanent.

Step 7: Repeating of steps 4 to 7 until there aren't any nodes left with a permanent distance, which neighbours still have temporary distances.

3) 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 start node is not empty and endq is not empty

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

Step 3: perform next iteration of BFS for 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 parent array

4) 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 check if the node has been in either open or closed list. If the node has not been in both lists, then add it to the OPEN list.

Step 7: Return to Step 2.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue III Mar 2022- Available at www.ijraset.com

5) Breadth-First Search (BFS) Algorithm

Breadth-first search is a graph traversal algorithm that starts traversing the graph from 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 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

6) 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

Step 6: EXIT

B. 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. Here users can add two points, add obstacles as per need & run various. Algorithms to compare the faster one & see the most efficient algorithm amongst all. This current grid is basically a 2d array of objects. Here every object has properties such as a row, column, is_start, is_finish, is_visited, is_wall to form the grid, locate the points & traverse the grid from one point to another using the logic of the particular algorithm.

This grid represents the full features & working space for the algorithms. It is designed to be simple & direct.

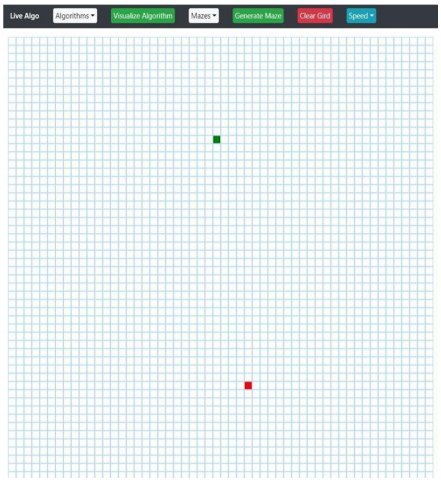


Fig. 2: The grid

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.

1) 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.

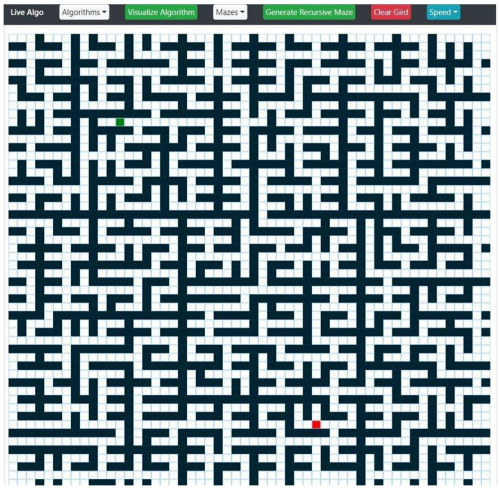


Fig 3: A maze created on the grid

2) 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.

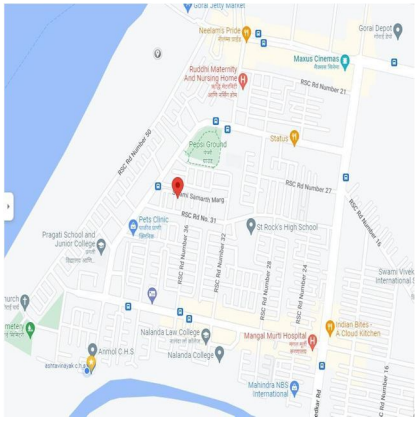


Fig 4: Image of a locality on Google map

Fig 5: Tracings of the same locality on the Grid

3) Use Case Diagram

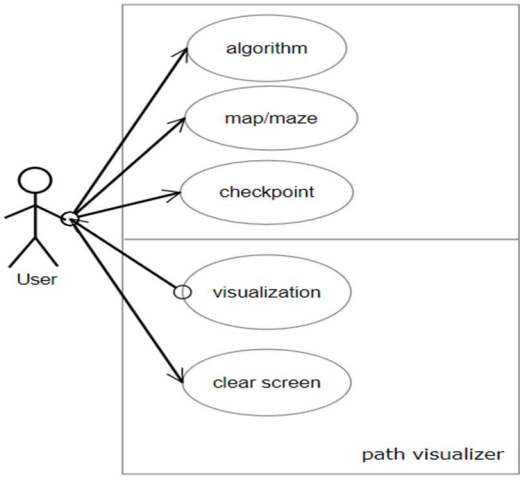


Fig 6: Prediction using AutoCorrect

4) 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.

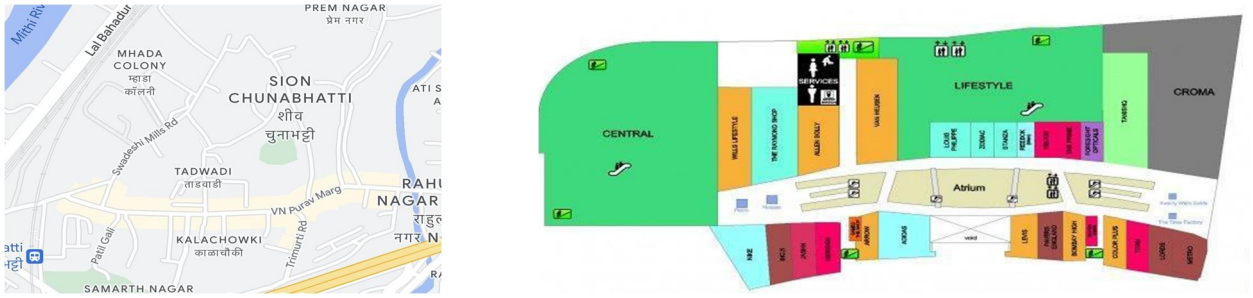


Fig 7: Outdoor & Indoor mappings

IV. ARCHITECTURE OF PROPOSED WORK

A. *System Architecture*

In this section, we study the usage of various mapping techniques & algorithms & how the system structure is set up. The structure consists of specific kinds of layers, with the primary layer being the input layer and the final layer being the output layer. The 2nd layer is known for the processing which consists of various algorithms & techniques used to execute the algorithm effectively.

The architecture of the system is as follows:

1) *Input Layer:* The input layer in this project consists of the GUI with a grid system built using arrays of objects with various properties to record & execute the algorithm based on pre-defined conditions. These objects consist of various parameters to store & retrieve the relevant information as per the program's need. This results in the formation of a grid-like system. This layer also includes the various features & overall UI for the system.
2) *Processing layer*: This layer is the constructing block of the entire system. Most of the computational work that is required to set the maze/plot the map, traverse the grid, find the shortest distance between the points & record the execution for various algorithms from the input is completed on this layer. The layer includes a set of rules for the creation of the outline (maze or map trace) & its traversing based on the fundamental principles of the algorithms.
3) *Output Layer:* The output layer is where the user can see the final executed grid with the traversed pathway followed by the algorithm & the resultant shortest route estimated by the particular algorithm based on the maze or map structure used.

V. RESULT

The results are promising. Every algorithm flows in a different way 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 8: 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.

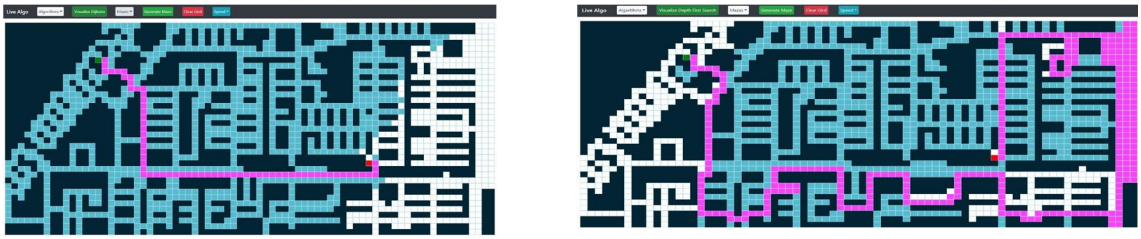


Fig 9: Input and Output of Algorithm comparison

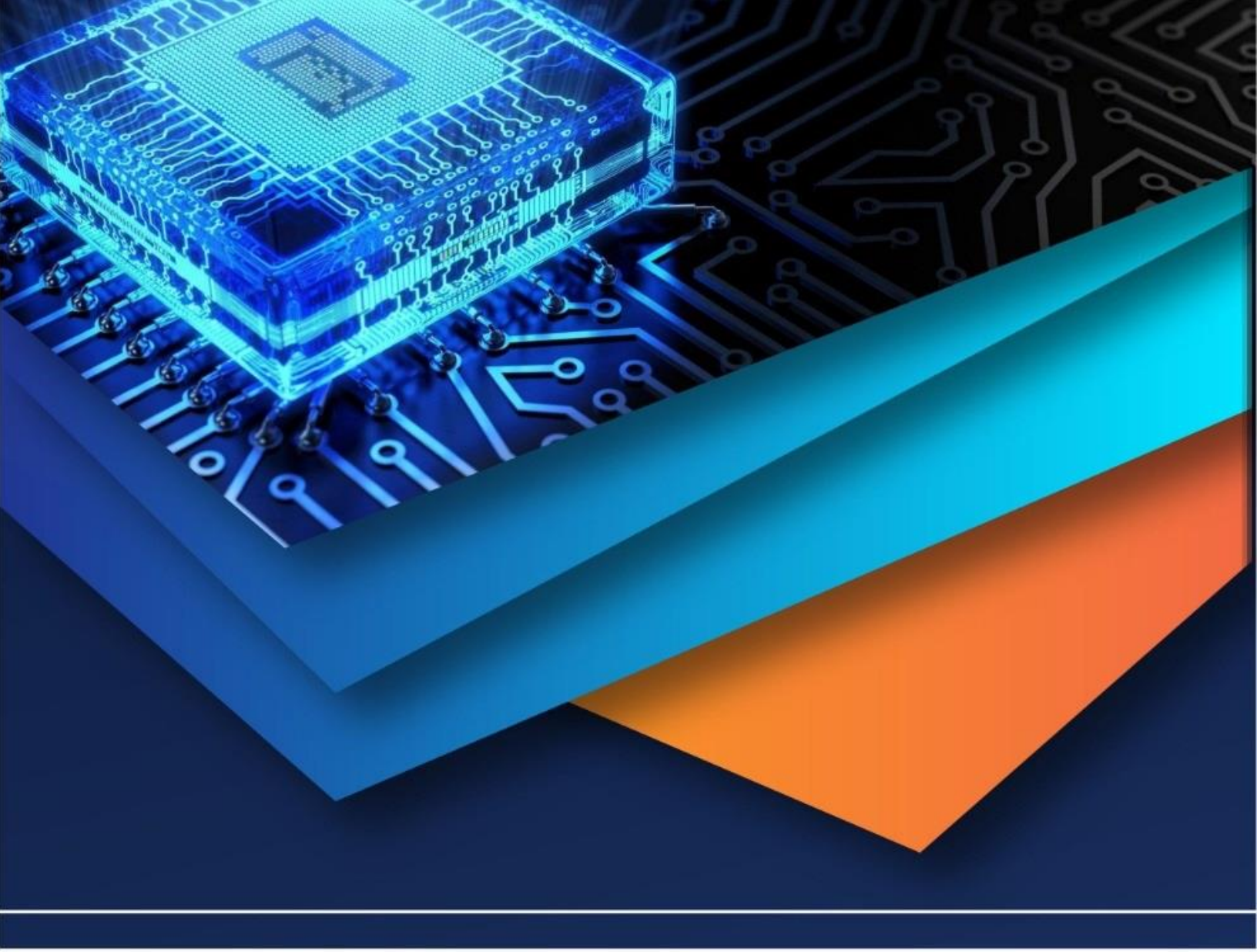
VI. CONCLUSION AND FUTURE SCOPE

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 it 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



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY