ISSN: 2366-1313

PATHFINDING VISUALIZER

[1]Ms. C.Archana, [2]Mohd. ObaidUllah Ansari,[3]Shaik Aftabuddin,[4]Mohd Khan

[1]Assistant Professor&HOD, Dept.of CSE, Teegala Krishna Reddy Engineering College, Meerpet, Hyderabad,

[2]BTech student, Dept.of CSE, Teegala Krishna Reddy Engineering College, Meerpet, Hyderabad, obaidansary4u@gmail.com

[3]BTech student, Dept.of CSE, Teegala Krishna Reddy Engineering College, Meerpet, Hyderabad,aftab.shaik13@gmail.com

[4]BTech student, Dept.of CSE, Teegala Krishna Reddy Engineering College, Meerpet, Hyderabad,mohdkhan150866@gmail.com

Abstract: A Pathfinding visualizer is a type of front-end visualization for the user to help them understand the different types of Pathfinding algorithms. This algorithm is used to find the shortest path of the algorithm between two different points using different techniques. A visual Pathfinding Program helps you to create your own mazes and obstacles and then run different algorithms on them. We can create our own mazes and our own type of puzzle and generate different types of outcomes with it. Most of the students often try to study these concepts from the provided notes from the college or online notes. But to do this, a lot of time is utilized and it's not time-saving. due to the computing power of the recent hardware, even very entangled visualization involving 3D could be successfully implemented using interpreted graphic script languages like JavaScript that are available to every web user without any installation. The Languages used in this project are HTML, CSS, and JavaScript. The Different types of algorithms used in this representation are Dijkstra's Algorithm (weighted), Greedy Best-firstSearch (weighted), SwarmAlgorithm(weighted), BidirectionalSwarmAlgorithm (weighted), Depth-first Search (unweighted).

Keywords: visualization, animation, algorithm, Invariant,Pathfinding visualizer,SwarmAlgorithm,Dijkstra's Algorithm.

I. INTRODUCTION

Algorithm visualization (often called algorithm animation) uses dynamic graphics to visualize computation of a given algorithm. First attempts to animate algorithms date to mid-80's (Brown, 1988; Brown and Sedgewick, 1985), and the golden age of algorithm visualization was around the year 2000, when magnificent software tools for an energetic algorithm visualization (e.g., the language Java and its graphic libraries) and plenty of

ISSN: 2366-1313

powerful hardware were already available. It was expected that algorithm visualization would completely change the way algorithms are taught. Many algorithm animations had appeared, mostly for simple problems like primary tree data structures and sorting. There were even attempts to robotize development of animated algorithms and algorithm visualization. Another guidance was to develop tools that would allow learners to prepare their own animations comfortably. Instead of giving appropriate references to algorithm animation papers, the reader is directed to a super-reference (Algoviz,) that brings a list of more than 650 authors/creator, some of them even with 29 references in algorithm animation and visualization. Understanding Data Structures and Algorithms (DSA), which includes the arrangement of algorithm theory, is a very challenging task in the computer science field. DSA is one of the most important subjects, but due to its abstract character, it is also one of the most difficult to master. The difficulty is generally due to the algorithm, which is derived from dynamic step-by-step procedures. Programming classes are included in both computer science and information science curriculum. Their objective is to educate students on fundamental programming principles such as control, methods of aggregation, sorting algorithms, and structures, etc. pupils, to support them in their learning. The objective of novice programmers is to implement the abstract process of algorithm execution in a way or language that will be understandable to the computer.However, if a programmer wants to 'explain' something to the computer, he or she first has to understand it perfectly. Namely, it is very difficult to understand and learn complex algorithms such as iterations, recursions or sorting algorithms only by watching code lines or a flow chart. On the contrary, if students can control their own data sets and see the whole process of algorithm execution, they are able to draw conclusions from the resulting data or in course of the algorithm visualization.

We strongly understand that the reason is relative basic: An algorithm operates on some data (the input data, working variables, and the output data). Usually, in any particular scope of Computer Science, there is a fundamental way of visualization of data - graphs and trees are drawn as circles linked by line segments, number chain could be visualized as collections of vertical bars, there are fundamental ways of drawing matrices, vectors, real functions, etc. An algorithm animation is usually enforced by running the algorithm slowly or in steps, and simply reorganize

ISSN: 2366-1313

the visual portrayal of the data in the screen. A person who knows and understands the algorithm in question can see how the algorithm progresses, but a learner user just sees visual objects moving and changing their shapes and colours, but finding out why the movie runs in that way is usually too difficult for him or her.

II. LITERATUE SURVEY

To design effective software, every software engineer needs to have a thorough understanding of DSA. Visualizers have a proven track record of giving useful information to the user's comprehension.

The various Algorithm Visualizers developed so far over the past few years are:An Artificial intelligence search Algorithm visualizer has been developed in the past by Abu Naser in 2008.The tool was deployed to solve the water jug problem with strategic three operational modes. A platform named JHAVEPOP was made by Furcy David in 2009 Languages like C++ or Java having Data Structures as Linked List can be visualized using this platform. The platform creates step by step visualization of the code written by the scholar in C++ or Java.

A platform VisuAlgo was designed by Dr Steven Halim in 2011 to productively explain the foundation of distinct algorithms to their students. Many Advanced algorithms were present in this platform as introduced in the book 'Competitive Programming which was co-authored with his brother Dr Felix Halim. The platform was limited to running only on small devices. A platform for Algorithm visualization was designed by Shaffer C. et Al, in 2011 which was deployed on the belief of group learning, unlike other platforms that provided barely a limited set of algorithms to be visualized. Group deployed learning through forums, discussions were promoted by the platform. The idea of the operation for expanding algorithm visualization tools by the programmers perhaps upgrade the

future of algorithm visualizers was proposed by Cooper Mathew et al, in his publications in 2014. A platform named VizAlgo was initiated by Simonak Slavomair [15] in 2015.

Visualization of sorting algorithms was the principal focus of the platform. It is comprised of mainly two interlinked components: the main component and apair of individualistic parts. There were many classes for helping in the execution of code in the main segment while the code for visualization was present in the independent component. For the Visualization of shortest path algorithms, an e-learning software was developed by

ISSN: 2366-1313

Borissova D. in 2015. The platform replenishes step by step visualization of the shortest path algorithm execution by conceiving, visualizing, and improving different graph structures. Algorithm visualizations were discussed and also an aspect was introduced to choose the particular algorithm deployed on the performance and virtue of a certain problem by Jonathan F. C. et al. in 2016. Algorithm visualization on the mobile platform was initiated by Supli A. A. et al. in 2017. It throws light on

two main characteristics: the layout of the User interface (UI) and its interactivity with the user. The designing of an algorithm visualizer was executed by Romanowska K. et al. in 2018. Moreover,

discussions were made regarding visualizer traits deployed on training and reliability goals. A recursion tree visualizer was generated by Bruno Papa et al. in 2020. The platform was meant to generate a visualized recursion tree from a certain recursion code. Reingold -Tilford's algorithm was set up to place the nodes of trees in a productive way. An integrated platform named AlgoAssist has been developed by Aniket B. Ghadge et al., in 2021. The tool contained a lab integration feature which makes it more realistic for both students and teachers.

III. PROPOSED WORK

Pathfinding is that the best tool to know the operating of any path finding rule or it's conjointly useful as academic tool to know the operating of the rule. There square measure several downsides which may be resolved by this type of tool. Pathfinding is wide employed in virtual environments, like laptop games. Most pathfinding varieties involve shortest pathfinding, that explores the quickest path, however military science ways may be sought for victimization varied properties. This paper provides a way for locating safe ways that maintain a balance between path length and risks from hostile components, likewise as a way to cut back computation time employing a hierarchic search strategy to reinforce operational potency. Safe pathfinding uses the A* rule, relating the influence map, that addresses the degree of risk within the tract. The searched path represents its attributes concerning total length and accumulative risk.

Pathfinding topic is widely used topic in finding the best path or we can say shortest path. .Many algorithm animations had appeared, mostly for simple problems like basic tree data structures and sorting when you start the program the shortest path from the start node represented in black to the end node represented in red. Another

ISSN: 2366-1313

direction was to develop tools that would allow students to prepare their own animations easily. In this paper we get so many. This application supports many algorithms, with the help of that algorithms finding the shortest path will be easy. For that we use Dijkstra, A* search, BFS etc.

The Implementation steps for our system are as follows:

Step 1: Reading the displayed tutorial shown by the webpage.

Step 2: Choose an algorithm from the list of algorithms.

Step 3: Adding a bomb is optional

Step 4: Choose a maze from the 5 of the given maze options given.

Step 5: Adjust the starting and ending point in the maze.

Step 6: Click on "Visualize" and then the given output will show inside the webpage.

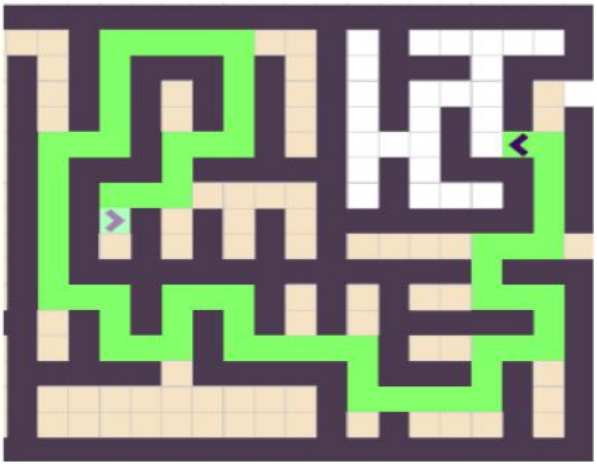
Pathfinding Algorithms

In Pathfinding Algorithms, the user can choose from a variety of mazes available under the Generate Maze button or the user can build its own maze through the interactive platform. Then, the user can select from various Pathfinding Algorithms and then visualization of the selected pathfinding algorithm is shown.

Dijkstra 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 at 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. All nodes that are neighbors to the current node are examined and their distance D from the initial node is calculated through the current node is calculated. If this new distance D is less than the previously recorded distance D for that node, the new distance value replaces the old distance value for that node. The neighbor node with the lowest distance value is marked as the new current node and the process repeats until the target is marked as visited or all nodes are marked as visited without the target being found.

Figure 1.1: Dijkstra's Algorithm B. A*



ISSN: 2366-1313

Fig.1 Dijkstra's Algorithm

B. A* Algorithm

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 nodebeing examined to the target. The value of g(x) is the distance from the initial node to the current node through all previous nodes traversed to get to that point. Therefore, if A* is examining node C as a possible next step in the path after traversing node B, then the g(x) value of node C is equal to the distance from the origin A to node B, plus the distance from node B to node C. This makes the g(x) value for a given node equal to the actual distance required to travel from the origin to that node, through all preceding nodes. h(x) is an estimate of the distance from the current node to the node located at the target. f(x) is the sum of g(x) and h(x).

A* also maintains an "open list," which is a list of all unvisited nodes, and a "closed list," or a list of visited nodes. At the beginning of the search, all nodes are on the open list, and the initial node is marked as current. The values of g(x), h(x), and f(x) are calculated for each of its neighbours. If the new f(x) value of a node being examined is less than the previous f(x) value for that node, the new f value replaces the old f value. The current node is moved from the open list to the closed list, the neighbour node with the lowest f(x) value is marked as the new current node and the process repeats until the target is added to the closed list, or there are no more nodes on the open list.

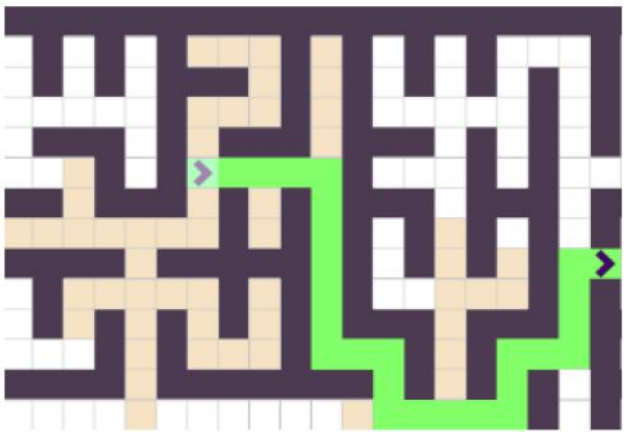


Fig.2 A* Search Algorithm

Breadth-First Search

The breadth-first search was discovered by Moore in the context of finding paths through mazes. BFS is a graph traversal algorithm to explore a tree or a graph efficiently. The algorithm starts with an initial node (root node) and then proceeds to explore all the nodes adjacent to it, in a breadth-first fashion, as opposed to depth-first, which goes down a particular branch till all the nodes in that branch are visited.

ISSN: 2366-1313

Put simply, it traverses the graph level-wise, not moving down a level till all the nodes in that level are visited and marked. It operates on the first-in-first-out (FIFO) principle and is implemented using a queue data structure. Once a node is visited, it is inserted into a queue. Then it is recorded and all its children's nodes are inserted into the queue. This process goes on till all the nodes in the graph are visited and recorded.

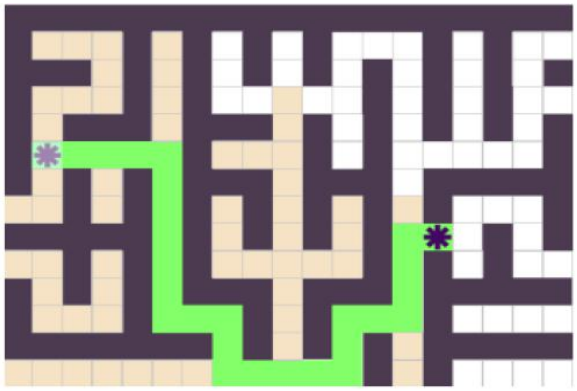


Fig.3 Breadth First Search Algorithm

Depth First Search

A version of the depth-first search was investigated in the 19th century by French mathematician Charles Pierre Trémaux as a strategy for solving mazes. The DFS search begins starting from the first node and goes deeper and deeperexploring down until the targeted node is found. If the targeted key is not found, the search path is changed to the path that was stopped exploring during the initial search, and the same procedure is repeated for that branch. The spanning tree is produced from the result of this search. The total number of nodes in the stack data structure is used to implement DFS traversal.

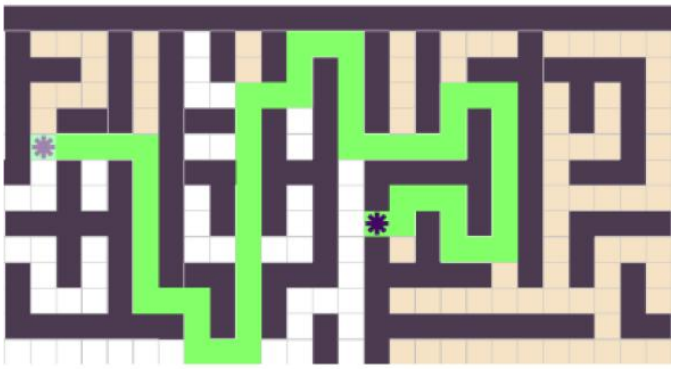


Fig.4 Depth First Search

Swarm Algorithm

The algorithm is essentially a mixture of Dijkstra's Algorithm and A* Search; more precisely, while it converges to the target node like A*, it still explores quite a few neighboring nodes surrounding the start node like Dijkstra's. The algorithm differentiates itself from A* through its use of heuristics: it continually updates nodes' distance from the start node while taking into account their estimated distance from the target node.

This effectively "balances" the difference in total distance between nodes closer to the start node and nodes closer to the target node, which results in the triangle-like shape of the Swarm Algorithm. We named the algorithm "Swarm" because one of its potential applications could be seen in a

ISSN: 2366-1313

video-game where a character must keep track of a boss with high priority (the target node), all the while keeping tracking of neighboring enemies that might be swarming nearby.

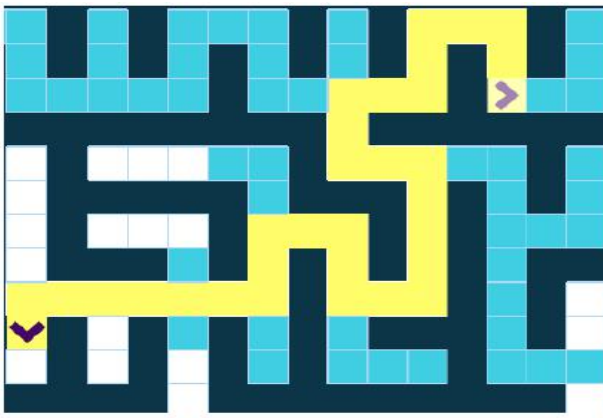


Fig.5 Swarm Algorithm

Bi-Directional Algorithm

Bidirectional search is a graph search algorithm that finds a 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 backward from the goal, stopping when the two meet. The reason for this approach is that in many cases it is faster: for instance, in a simplified model of search problem complexity in which both searches expand a tree with branching factor b, and the distance from start to goal is d, each of the two searches has complexity O(bd/2) (in Big O notation), and the sum of these two search times is much less than the O(bd) complexity that would result from a single search from the beginning to the goal.

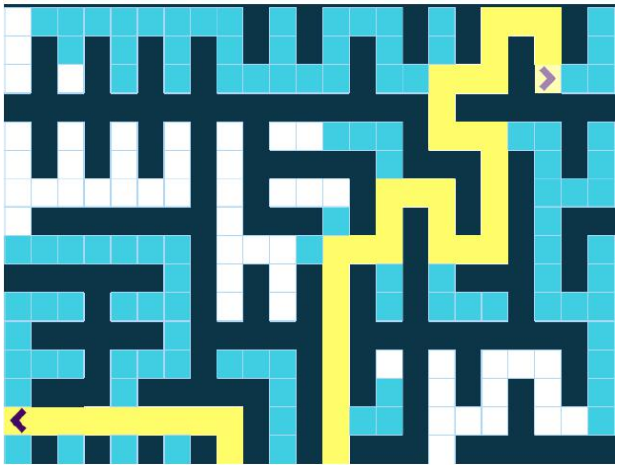


Fig.6 Bi-Directional Algorithm

Analysis for Sorting Algorithms

Table I depicts the comparison between various sorting algorithms that we have implemented in our web-based visualization tool. The algorithms are analysed with 8 input values with average runtime in seconds. From the table below it is clear that Selection Sort consumes less time as compared to other Sorting Algorithms. Among all the algorithms Bubble Sort will be the most time-consuming algorithm as each adjacent element are compared and are swapped as per the requirements. The entire process will be repeated for each traversal. Hence, the time complexity of Bubble Sort is worse than the others.

Table.1 Analysis of Algorithms

ISSN: 2366-1313

No. of Inputs	Time taken by Bubble Sort in secs	Time taken by Selection Sort in secs	Time taken by Insertion Sort in secs	Time taken by Merge Sort in secs	Time taken by Quick Sort in secs
5	4.6	2.9	4.5	4.0	2.4
6	5.3	3.8	4.7	4.3	2.6
7	6.3	4.9	4.9	4.9	2.9
8	7.4	6.1	5.0	5.2	3.4

Analysis for Pathfinding Algorithms

An efficient algorithm is one that calculates the shortest path with the fewest number of node visitations. Among all the Pathfinding Algorithms, Dijkstra's Algorithms is least efficient as it has no method of cutting down on search space and it made far more node visitations during each path calculation phase. Depending, on the situation A* and D* are the most efficient pathfinding Algorithms.

IV. RESULTS

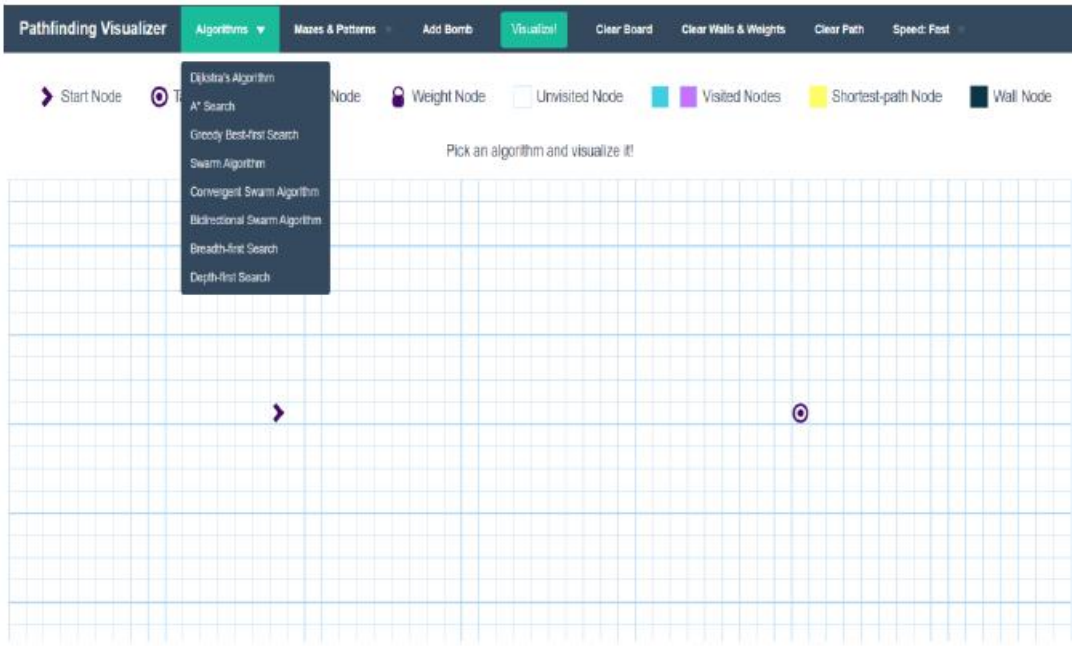


Fig.7 Visualizer Interface

After selecting the algorithm, enter "Visualize Dijkstra's Algorithm" This is how it visualize using the Dijkstra's Algorithm. Given below is the screenshot representing it.

ISSN: 2366-1313

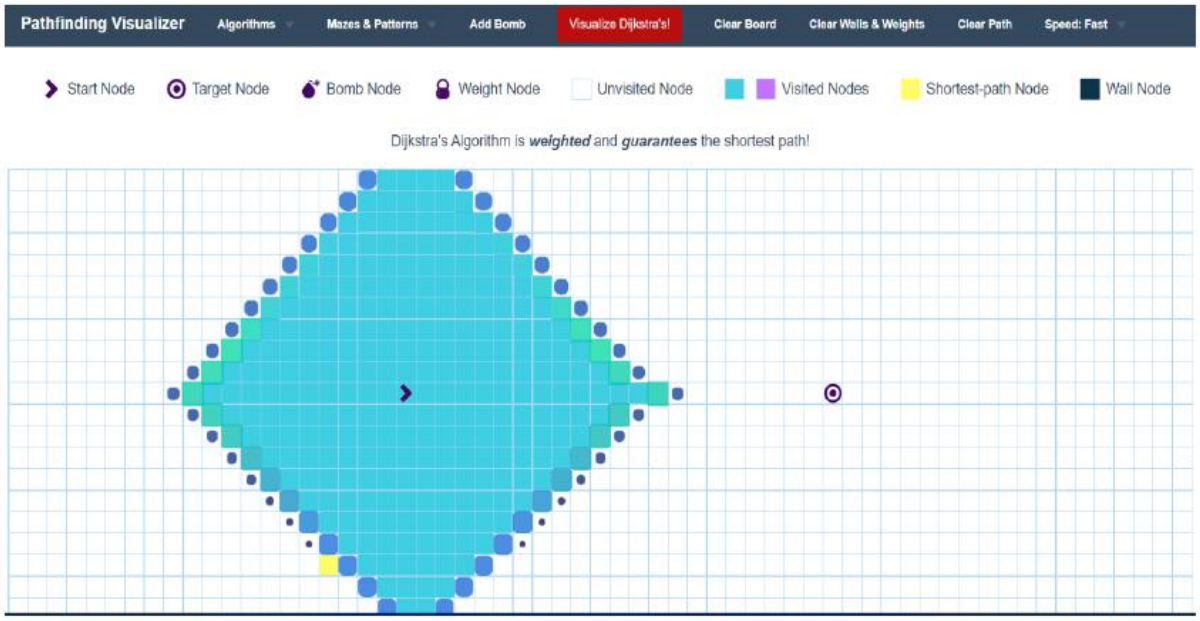


Fig.8 Visualization of Dijkstra's

Here, the visualizer finds the shortest path between two points. We can see it visualizes the starting point and the ending point.

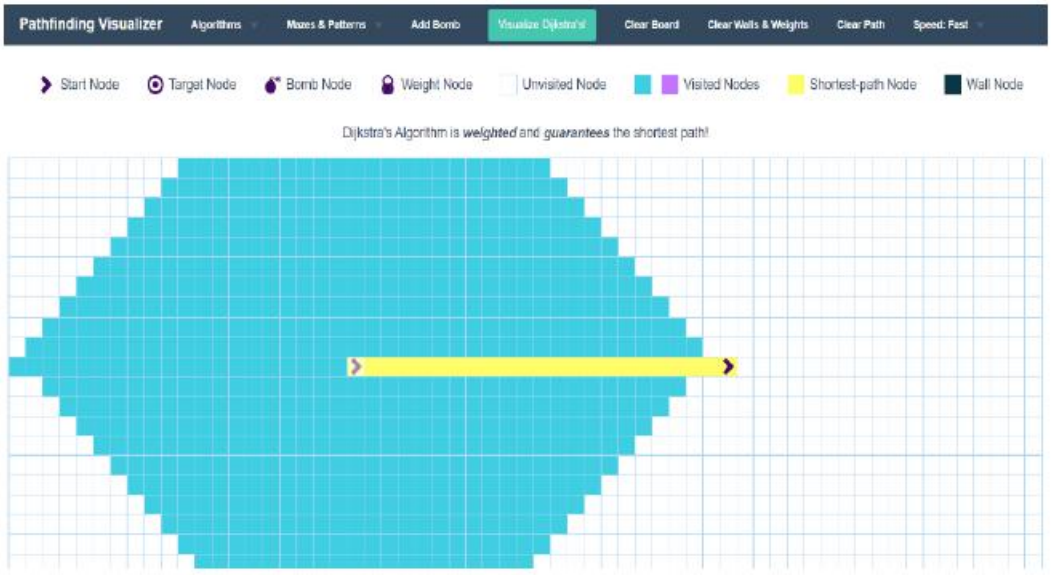


Fig.9 Pathfinder visualization

Here, we choose a type of maze called the "Recursive Division Method" It automatically creates a maze shown in the above figure.

ISSN: 2366-1313

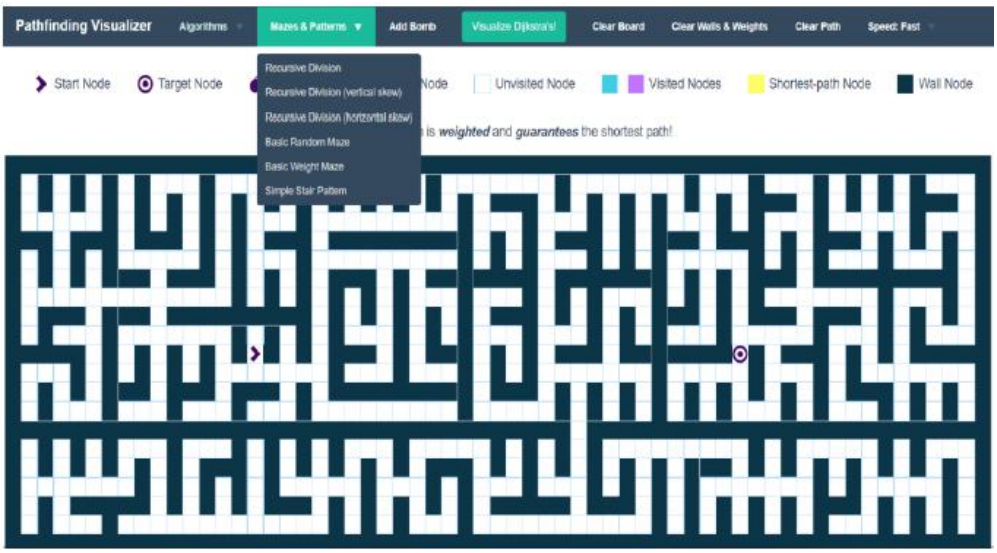


Fig.10 Maze Selection

We can even add a bomb to add a new starting point in the visualizer. So, by using the bomb we can have a new stopping point before reaching the endpoint.

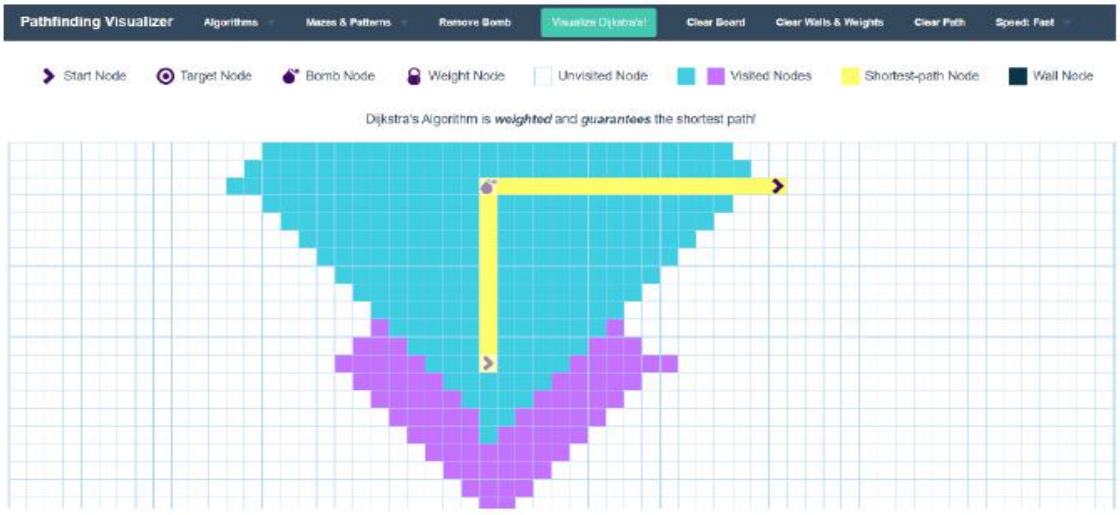


Fig.11 Adding Bomb

V. CONCLUSION

Different algorithms and different mazes are used in this visualization in order to visualize the pathfinding visualizations. We can even add 6 types of mazes in this visualizer to test different types of outcomes. There are 5 different types of Algorithms in need of visualization. Here, the design and implementation of the visualization in the algorithms is more efficient and easier to understand. Just like how we use it in maps and trees etc. These algorithms can also be implemented in

ISSN: 2366-1313

gaming devices to find the shortest path to go from one place to another.

REFERENCES

1. JavaScript reference and documentation retrieved from https://devdocs.io/javascript/

2. A. Issa, M.O.A. Aqel, M. Khdair, M. Abubaker, M. ElHabbash and M. Massoud, "Intelligent Maze Solving Robot Based On Image Processing and Graph Theory," 2017 International Conference on Promising Electronic Technologies (ICPET), page 49-53, October 2017

3. Nawaf Hazim Barnouti, Sinan Sameer Mahmood Al-Dabbagh and Mustafa Abdul Sahib Naser, "Pathfinding in Strategy Games and Maze Solving Using A Search Algorithm," Journal of Computer and Communications Vol.04 No.11(2016), Article ID:70460,11 pages

4. P Prasadu Peddi (2019), "Data Pull out and facts unearthing in biological Databases", International Journal of Techno-Engineering, Vol. 11, issue 1, pp: 25-32.

5. N. Garg, "Lecture - 25 Data Structures for Graphs [Video file]", September 24,2008. Retrieved from https://www.youtube.com/watch?v=hk5rQs7TQ7E

6. M. Chandra Wijaya, S. Tjiharjadi, and E. Setiawan, "Optimization Maze Robot Using A and Flood Fill Algorithm," International Journal of Mechanical Engineering and Robotics Research Vol. 6, No. 5, September 2017

7. C.E. Leiserson, T.H. Coremen, R.L. Rivest and C. Stein, "Introduction to Algorithms", Third edition, Prentice Hall of India, page 587-748, 2009

8. Silvester Dian Handy Permana, Ketut Bayu Yogha Bintoro, Budi Arifitama and Ade Syahputra, "Comparative Analysis of Pathfinding Algorithms A , Dijkstra, and BFS on Maze Runner Game" International Journal Of Information System & Technology Vol. 1, No. 2, (2018), pp. 1-8

9. Xiao Cui and Hao Shi, "A-based Pathfinding in Modern Computer Games" VOL.11 No.1, January 2011

10. Bi Yu chen, Chaoyang Shi and Shujin Xiang, "Most reliable path-finding algorithm for maximizing on-time arrival probability" Pages 248-264 | Received 01 Jul 2015, accepted 21 Mar 2016, Published online: 13 Apr 2016

11. N Srivani, Prasadu Peddi (2021), Face Assessment Learned From Existing Images In Order To Classify The Gender Of The Images Based On Improved Face Recognition, (TURCOMAT), Vol 12, issue 6, pp: 5724-5735

ISSN: 2366-1313

[12] J. Kennedy and R.C. Eberhart (1997), “A discrete binary version of the particle swarm algorithm” IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation.

[13] Prasadu Peddi (2019), "Data Pull out and facts unearthing in biological Databases", International Journal of Techno-Engineering, Vol. 11, issue 1, pp: 25-32.