IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Path Finding Visualizer: An Experimental Survey On Various Path Finding Algorithms

[1]Shikha Bhardwaj, [2]Shefali Dhingra

[1]Assistant Professor, [2]Assistant Professor

[1]Department of ECE,

[1]UIET Kurukshetra University Kurukshetra, India

Abstract: Plotting is known as path finding or pathing. A computer program, is used for connecting the path between two locations. Being more of a useful maze-solving variation, Path finding or pathing is the plotting, by a computer application, of the shortest route between two points. It is a more practical variant on solving mazes. This field of research is based heavily on Dijkstra's algorithm for finding the shortest path on a weighted graph. A path finding approach fundamentally searches, beginning at one vertex in a graph, looking into nearby nodes until they reach the destination node, typically with the goal of locating the least expensive path. Although graph search techniques can also be used, viz. a breadth-first search, Depth-based, swarm based and other algorithms can also be used.

Index Terms **- Dijkstra's, Path finding algorithm, Greedy, Swarm, Breadth, Depth.**

Introduction

Path finding Visualizer is simply a tool that visualizes how path finding algorithms work. A path finding visualizer allows the user to create their own obstacles or mazes and then run different path finding algorithms on it. For example, a person moving across space considering all alternatives. The person would typically walk forward just traveling in the destination's direction veer off the path to avoid an impediment, and downplay deviations the maximum area (shortest, cheapest, fastest, etc.) between two points in a large network. Path-finding algorithms build on top of graph search algorithms and explore routes between nodes, starting at one node and traversing through relationships until the destination has been reached. These algorithms find the cheapest path in terms of the number of hops or weight [1]. Weights can be anything measured, such as time, distance, capacity, or cost. There are many path finding visualizer algorithms like:

1. Dijkstra's Algorithm
2. A* Search
3. Greedy Best-first Search
4. Swarm Algorithm
5. Convergent Swarm Algorithm
6. Bidirectional Swarm Algorithm
7. Breadth-First Search
8. Depth-First Search

Similarly, many types of mazes and patterns are also available like:

1. Recursive Division
2. Recursive Division(vertical skew)
3. Recursive Division(horizontal skew)
4. Basic Random Maze
5. Basic Weight Maze

6. Simple Star Pattern.

I. PATHFINDING VISUALIZER USING DIJKSTRA'S ALGORITHM

Dijkstra's algorithm is used to find the shortest path from a single source vertex to all other vertices in a given graph. This is a teaching tool that is used for easy visualization of Dijkstra's algorithm [2] implemented using the JS library for graph drawing. The screenshot for this algorithm has been shown in Fig. 1. And with vertical skew maze is shown in fig. 2.

```
function dijkstra(G, S)
    for each vertex V in G
        distance[V] <- infinite
        previous[V] <- NULL
        If V != S, add V to Priority Queue Q
    distance[S] <- 0

    while Q IS NOT EMPTY
        U <- Extract MIN from Q
        for each unvisited neighbour V of U
            tempDistance <- distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
                distance[V] <- tempDistance
                previous[V] <- U
    return distance[], previous[]
```

Fig. 1 Pseudocode for Dijkstra's Algorithm

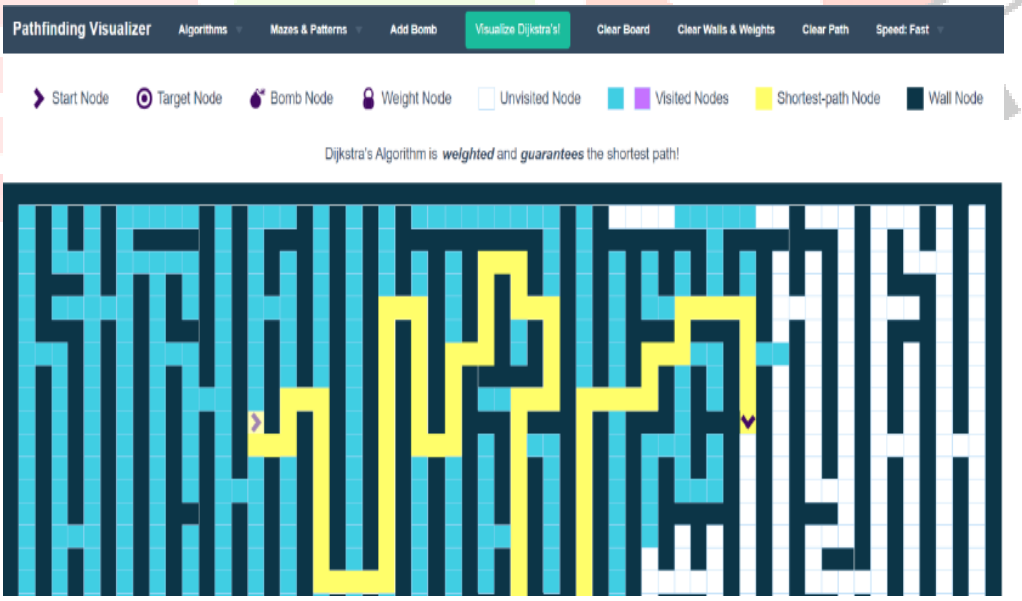

Fig. 2 Dijkstra's Algorithm with vertical skew maze

II. PATHFINDING VISUALIZER USING A* ALGORITHM:

A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals. It is a searching algorithm that is used to find the shortest path between an initial and a final point. It is a handy algorithm that is often used for map traversal to find the shortest path to be taken. A* was initially designed as a graph traversal problem, to help build a robot that can find its own course. It still remains a widely popular algorithm for graph traversal. It searches for shorter paths first, thus making it an optimal and complete algorithm [2]. An optimal algorithm will find the least cost outcome for a problem, while a complete algorithm finds all the possible outcomes of a problem. Another aspect that makes A* so powerful is the use of weighted graphs in its implementation. A weighted graph uses numbers to represent the cost of taking each path or course of action. This means that the algorithms can take the path with the least cost, and find the best route in terms of distance and time.It is arguably the best pathfinding algorithm [3], uses heuristics to guarantee the shortest path much faster than Dijkstra's Algorithm. With maze it is shown in Fig. 3.

Fig. 3 A search Algorithm with random maze

III. A SEARCH ALGORITHM WITH RANDOM MAZE

Greedy Best-First Search is an AI search algorithm that attempts to find the most promising path from a given starting point to a goal. It prioritizes paths that appear to be the most promising, regardless of whether or not they are actually the shortest path. The algorithm works by evaluating the cost of each possible path and then expanding the path with the lowest cost. This process is repeated until the goal is reached. The algorithm works by using a heuristic function to determine which path is the most promising. The heuristic function takes into account the cost of the current path and the estimated cost of the remaining paths [4]. If the cost of the current path is lower than the estimated cost of the remaining paths, then the current path is chosen. This process is repeated until the goal is reached.

Advantages of Greedy Best-First Search:

- Simple and Easy to Implement: Greedy Best-First Search is a relatively straightforward algorithm, making it easy to implement.
- Fast and Efficient: Greedy Best-First Search is a very fast algorithm, making it ideal for applications where speed is essential.
- Low Memory Requirements: Greedy Best-First Search requires only a small amount of memory, making it suitable for applications with limited memory.
- Flexible: Greedy Best-First Search can be adapted to different types of problems and can be easily extended to more complex problems.
- Efficiency: If the heuristic function used in Greedy Best-First Search is good to estimate, how close a node is to the solution, this algorithm can be a very efficient and find a solution quickly, even in large search spaces.

Disadvantages of Greedy Best-First Search:

- Inaccurate Results: Greedy Best-First Search is not always guaranteed to find the optimal solution, as it is only concerned with finding the most promising path.

- Local Optima: Greedy Best-First Search can get stuck in local optima, meaning that the path chosen may not be the best possible path.
- Heuristic Function: Greedy Best-First Search requires a heuristic function in order to work, which adds complexity to the algorithm.
- Lack of Completeness: Greedy Best-First Search is not a complete algorithm, meaning it may not always find a solution if one is exists. This can happen if the algorithm gets stuck in a cycle or if the search space is a too much complex..

IV. PATHFINDING VISUALIZER USING SWARM'S ALGORITHM:

Swarm's Algorithm is a mixture of Dijkstra's Algorithm and A* and also does not guarantee the shortest-path. It is a searching algorithm used for traversing a weighted tree or graph. This algorithm comes into play when a different cost is available for each edge. The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost [4]. Uniform-cost search expands nodes according to their path costs form the root node. It can be used to solve any graph/tree where the optimal cost is in demand. A uniform-cost search algorithm is implemented by the priority queue. It gives maximum priority to the lowest cumulative cost. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

V. PATHFINDING VISUALIZER USING BREADTH-FIRST SEARCH ALGORITHM:

Path-finding using BFS i.e. Breadth-First Search BFS is a traversal algorithm in which you should begin at a chosen node (source or starting node) and move layer-by-layer through the network to explore your neighbours' nodes (nodes that are directly connected to the source node). After that, you must head toward the nodes that are next in level [5]. 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. While using BFS for traversal, any node in the graph can be considered as the root node.

There are many ways to traverse the graph, but among them, BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.

VI. PATHFINDING VISUALIZER USING DEPTH-FIRST SEARCH ALGORITHM:

It is a recursive algorithm to search all the vertices of a tree data structure or a graph [6]. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children. Because of the recursive nature, stack data structure can be used to implement the DFS algorithm. The process of implementing the DFS is similar to the BFS algorithm.

The step-by-step process to implement the DFS traversal is given as follows –

1. First, create a stack with the total number of vertices in the graph.
2. Now, choose any vertex as the starting point of traversal, and push that vertex into the stack.
3. After that, push a non-visited vertex (adjacent to the vertex on the top of the stack) to the top of the stack.
4. Now, repeat steps 3 and 4 until no vertices are left to visit from the vertex on the stack's top.
5. If no vertex is left, go back and pop a vertex from the stack.
6. Repeat steps 2, 3, and 4 until the stack is empty.

VII. PATHFINDING VISUALIZER USING BIDIRECTIONAL SWARM ALGORITHM:

Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and other from goal node called as backward-search, to find the goal node. Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex. The search stops when these two graphs intersect each other.

Bidirectional search can use search techniques such as BFS, DFS, DLS, etc [6].

VIII. CONCLUSION

In this paper, review of various path finding algorithms have been successfully done. Various Path-finding algorithms have been studied in this paper and different mazes have been created for each of the algorithm. Paths can be judged and ranked holistically, taking topology, attributes, and grouping structure into account. Visualizers are algorithms used to find optimal path between two locations. These algorithms are widely used in map applications like Google Maps, for example. A code has been written for each algorithm and different mazes have been tried for each.

REFERENCES

[1] Geekforgeeks: https://www.geeksforgeeks.org/selection-sort-visualizer-in-javascript/
[2] Open Sans Font (Google Fonts): https://fonts.google.com/specimen/Ope...
[3] The 'iconify' dependency: https://www.npmjs.com/package/@iconif...
[4] Dotenv' dependency: https://www.npmjs.com/package/dotenv
[5] Github
[6] ReactJS code snippets extension: https://marketplace.visualstudio.com/...