**Pathfinder Visualizer**

**PROJECT SYNOPSIS**

**of**

**Project-I**

(BTCS 603-18)

BACHELOR OF TECHNOLOGY

Computer Science & Engineering



**SUBMITTED BY:** **GUIDED BY:**

Prince Chand (1906121) Er. Harsimranpreet Kaur

Priya (190124)

Bhanu Partap (1906069)

**Introduction**

Pathfinding 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. Pathfinding is closely related to the shortest path problem, within graph theory, which examines how to identify the path that best meets some criteria (shortest, cheapest, fastest, etc) between two points in a large network. At its core, a pathfinding method searches a graph by starting at one vertex and exploring adjacent nodes until the destination node is reached, generally with the intent of finding the cheapest route. Although graph searching methods such as a breadth-first search would find a route if given enough time, other methods, which "explore" the graph, would tend to reach the destination sooner. An analogy would be a person walking across a room; rather than examining every possible route in advance, the person would generally walk in the direction of the destination and only deviate from the path to avoid an obstruction and make deviations as minor as possible.

Algorithm: -

Edsger Dijkstra is Dutch. He is one of the big names in computer science. He is known for his handwriting and quotes such as:

• Simplicity is prerequisite for reliability.

• The question of whether machines can think is about as relevant as the question of whether submarines can swim.

Here we use the Front End web development for the creation of path finder visualizer. That’s are: -

Front-end web development, or client-side development, refers to working with HTML, CSS and JavaScript for a website or web application that allows users to see and interact with them directly.

Let us understand how and why these coding languages are used in front-end development.

**HTML**

The backbone of any website development process, web pages can not exist without Hyper Text Markup Language (HTML). Hypertext means that the text has links, called hyperlinks, embedded in it. So when a user clicks on a word or a phrase that has a hyperlink, it leads him/ her to another web-page. A markup language also means that the text can be converted into images, tables, links and other representations. The overall framework of how the site will look is written in HTML code. Developed by Tim Berners-Lee, HTML’s latest version is called HTML5 which was published back in 2014. This version boasts new and efficient ways of handling elements such as video and audio files.

**CSS**

A cornerstone technology of the World Wide Web (along with HTML and JavaScript), Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. So it deals with the presentation aspect of the site and gives your site its own unique look. This is done by maintaining style sheets which sit on top of other style rules and are activated on the basis of other inputs like device screen size and resolution.

**JavaScript**

A multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles. It is used to transform a static HTML page into a dynamic interface. JavaScript code can use the DOM to manipulate a web page in response to events, such as user input. In fact, JavaScript code can also actively retrieve content from the web as well react to server-side events using a technique called AJAX. An essential part of web applications, most websites use it for client-side page behavior and all major web browsers have a dedicated JavaScript engine to execute it.

**ReactJS**

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It’s ‘V’ in MVC. ReactJS is an open-source, component-based front-end library responsible only for the view layer of the application. It is maintained by Facebook.

React uses a declarative paradigm that makes it easier to reason about your application and aims to be both efficient and flexible. It designs simple views for each state in your application, and React will efficiently update and render just the right component when your data changes. The declarative view makes your code more predictable and easier to debug.

**Literature Survey**

Literature Review is required to take the matter into considerations that can’t be cleared in the past researches. Many researchers try to interpret various kind of conclusions and to improve those past results literature review is needed. The present literature serves many varied interesting features, which forms the vital background for the study and conducted a consideration. An important field of mathematical theory is the mathematical study of the structure of abstract relationships between objects by means of graphs (networks). Although investigating of these constructions can be purely theoretical, they can be used to model pair wise relationships in many real world systems. One of most widely using applications is determination of shortest paths in many practical applications as: maps; robot navigation; texture mapping; typesetting in TeX; urban traffic planning; optimal pipelining of VLSI chips; subroutines in advanced algorithms; telemarketer operator scheduling; routing of telecommunications messages; approximating piecewise linear functions; network routing protocols (OSPF, BGP, RIP); exploiting arbitrage opportunities in currency exchange; optimal truck routing through given traffic congestion pattern.

**DATA STRUCTURES**

In practice, graphs are usually represented by one of two standard data structures: adjacency lists and adjacency matrices. At a high level, both data structures are arrays indexed by vertices; this requires that each vertex has a unique integer identifier between 1 and V. In a formal sense, these integers are the vertices.

**ADJACENCY LISTS**

By far the most common data structure for storing graphs is the adjacency list. An adjacency list is an array of lists, each containing the neighbors of one of the vertices (or the out-neighbors if the graph is directed). For undirected graphs, each edge uv is stored twice, once in u’s neighbor list and once in v’s neighbor list; for directed graphs, each edge uv is stored only once, in the neighbor list of the tail u. For both types of graphs, the overall space required for an adjacency list is O(V + E).

There are several dierent ways to represent these neighbor lists, but the standard implementation uses a simple singly-linked list. The resulting data structure allows us to list the (out-)neighbors of a node v in O(1 + deg(v)) time; just scan v’s neighbor list. Similarly, we can determine whether uv is an edge in O(1 + deg(u)) time scanning the neighbor list of u. For undirected graphs, we can improve the time to O(1 + min{deg(u), deg(v)}) by simultaneously scanning the neighbor lists of both u and v, stopping either when we locate the edge or when we fall of the end of a list.

**ADJACENCY MATRICES**

The other standard data structure for graphs is the adjacency matrix, first proposed by Georges Brunel in . The adjacency matrix of a graph G is a V ⇥ V matrix of 0s and 1s, normally represented by a two-dimensional array A[1 .. V, 1 .. V], where each entry indicates whether a particular edge is present in G. Specifically, for all vertices u and v:

if the graph is undirected, then A[u, v] := 1 if and only if uv 2 E, and if the graph is directed, then A[u, v] := 1 if and only if uv 2 E.

For undirected graphs, the adjacency matrix is always symmetric, meaning A[u, v] = A[v, u] for all vertices u and v, because uv and vu are just dierent names for the same edge, and the diagonal entries A[u, u] are all zeros. For directed graphs, the adjacency matrix may or may not be symmetric, and the diagonal entries may or may not be zero.

Given an adjacency matrix, we can decide in ⇥ (1) time whether two vertices are connected by an edge just by looking in the appropriate slot in the matrix. We can also list all the neighbors of a vertex in ⇥ (V) time by scanning the corresponding row (or column). This running time is optimal in the worst case, but even if a vertex has few neighbors, we still have to scan the entire row to find them all. Similarly, adjacency matrices require ⇥ (V2) space, regardless of how many edges the graph actually has, so they are only space-evident for very dense graphs.

Diagram, engineering drawing

Description automatically generated

**Methodology/ Planning of work**

In this section, the overall working of the project has been described. How the project started and how the project works and how the various phases of project were carried out and the challenges faced at each level. What does the project do? At its core, a pathfinding algorithm seeks to find the shortest path between two points. This project visualizes various pathfinding algorithms in action, and more! All of the algorithms on this project are adapted for a 2D grid, where 90 degree turns have a "cost" of 1 and movements from a node to another have a "cost" of 1. Picking an Algorithm Choose an algorithm from the "Algorithms" drop-down menu. Note that some algorithms are unweighted, while others are weighted. Unweighted algorithms do not take turns or weight nodes into account, whereas weighted ones do.

Additionally, not all algorithms guarantee the shortest path. Meet the algorithms Dijkstra's Algorithm :The father of pathfinding algorithms; guarantees the shortest path. Greedy Best-first Search (weighted): A faster, more heuristic-heavy version of A\*; does not guarantee the shortest path. Breath-first Search (unweighted): A great algorithm; guarantees the shortest path. Depth-first Search (unweighted): A very bad algorithm for pathfinding; does not guarantee the shortest path. Adding walls Click on the grid to add a wall. Walls are impenetrable, meaning that a path cannot cross through them. Visualizing and more Use the navbar buttons to visualize algorithms and to do other stuff! You can clear the current path, clear walls and weights, clear the entire board, and adjust the visualization speed, all from the navbar. If you want to access this tutorial again, click on "Pathfinding Visualizer" in the top left corner of your screen. Objectives of the project:

● It can be used as a E learning tool to understand Algorithms.

● It is used in finding Shortest Path.

● It is used in the telephone network.

● It is used in IP routing to find Open shortest Path First.

● It is used in geographical Maps to find locations of Map which refers to vertices of graph.

● We can make a GPS system which will guide you to the locations.

● Search engine crawlers are used BFS to build index. Starting from source page, it finds all links in it to get new pages.

● In peer-to-peer network like bit-torrent, BFS is used to find all neighbor nodes.

● As users of wireless technology, people demand high data rates beyond GigaBytes per second for Voice, Video and other applications. There are many standards to achieve data rates beyond GB/s. One of the standards is MIMO(Multi input Multi output). MIMO employs K-best Algorithm(which is a Breadth-First Search algorithm) to find the shortest partial euclidian distances. Phases of the project. The development of the project has been carved out in Six Phases. These phases include all the steps of the project, beginning from data collection and processing to the output for the user. The Six phases are:

1. Building of Graph Matrix.

2. Added Walls and Event listeners.

3. Embed the Graph Algorithms.

4. Integrated the Path finding Functionality.

5. Improved the Design and UI.

6. Added the Timer Functionality

After all these phases the project is completely ready for the user to use. Each Phase has been discussed in detail from here on onwards, letting to a complete understanding of the project.

**Facilities required for proposed work**

1. Prince Chand (Project Manager + coder)
2. Priya (UI/UX designer)
3. Bhanu (Project Organizer + coder)
4. Windows 10 OS
5. Visual Studio code

**References**

1. <https://en.wikipedia.org/wiki/Web_development> [21/03/2022 09:12pm]
2. <https://www.google.com/url?sa=t&source=web&rct=j&url=https://www.ijmtst.com/volume6/issue12/93.IJMTST0612296.pdf&ved=2ahUKEwie8dbBuNb2AhWIwZQKHWbBCUsQFnoECDIQAQ&usg=AOvVaw1iz6e-L3kLN7Ercv2hRljX> [21/03/2022 10:06pm]
3. <https://www.youngwonks.com/blog/Introduction-to-Front-end-development> [04/04/2022 02:02pm]
4. <https://en.wikipedia.org/wiki/Pathfinding> [23/03/2022 03:30pm]
5. <https://www.w3schools.com/js/> [23/03/2022 03:35pm]
6. <https://reactjs.org/> [23/03/2022 09:21pm]