ALGORITHM VISUALIZER

A

Minor Project

Submitted in the partial fulfilment for the award of

Bachelor of Technology
In
Computer Science & Engineering

Submitted to

RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA

BHOPAL (M.P.)

Submitted by:

Abhishek Kumar (0133CS181006)
Anuj Kumar Sah (0133CS181028)
Dev Purohit (0133CS181050)
Gaurav Dubey (0133CS181060)

Under the Guidance of
Prof. Mohit Tomar

Department of Computer Science & Engineering

Sagar Institute of Research & Technology, Bhopal

MAY- 2021

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the minor project, entitled " **Algorithm Visualizer**" Submitted in the Department of Computer Science & Engineering, Sagar Institute of Research & Technology, Bhopal is an authentic record of my own work carried out during the period from January 2021 - June 2021, under the guidance of Mr. Mohit Tomar , Department of Computer Science & Engineering, Sagar Institute of Research & Technology, Bhopal.

Date : 10/05/2021

Place : Bhopal

Abhishek Kumar (0133CS181006)
Anuj Kumar Sah (0133CS181028)
Dev Purohit (0133CS181050)
Gaurav Dubey(0133CS181060)

CERTIFICATE

This is to certify that the minor project entitled **"Algorithm Visualizer"** submitted to Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.) by **Abhishek Kumar (0133CS181006), Anuj Kumar Sah (0133CS181028), Dev Purohit (0133CS181050) and Gaurav Dubey (0133CS181060)** is a partial fulfilment of the requirement for the award of degree of **Bachelor of Engineering** in **Computer Science & Engineering**. The matter embodied is the actual work done by **Abhishek Kumar (0133CS181006), Anuj Kumar Sah (0133CS181028), Dev Purohit (0133CS181050) and Gaurav Dubey (0133CS181060)** and is a record of bonafide work done by her under my supervision.

Guided By
Prof. Mohit Tomar
Computer Science & Engg.
SIRT, Bhopal

Approved By
Dr. Ritu Shrivastava
Head of Department
Computer Science & Engg. SIRT, Bhopal

SAGAR INSTITUTE OF RESEARCH & TECHNOLOGY
Ayodhya Bypass Road, Bhopal (M.P)

APPROVAL CERTIFICATE

This Minor Project work entitled "**ALGORITHM VISUALIZER**" submitted by **Abhishek Kumar (0133CS181006), Anuj Kumar Sah (0133CS181028), Dev Purohit (0133CS181050) and Gaurav Dubey (0133CS181060)** is approved for the award of degree of Bachelor of Engineering in Computer Science and Engineering.

Project Incharge

Prof. Mohit Tomar

Date: 10/05/2021

DECLARATION

I hereby declare that the Minor Project titled **"Algorithm Visualizer"** is my own work conducted under the supervision of **Prof. Mohit Tomar**, Professor, Department of Computer Science & Engineering, **Sagar Institute of Research & Technology, Bhopal, M.P.**

Abhishek Kumar (0133CS181006)
Anuj Kumar Sah (0133CS181028)
Dev Purohit (0133CS181050)
Gaurav Dubey (0133CS181060)

ACKNOWLEDGEMENT

I express my deep sense of gratitude to (Guide Name) **Prof. Mohit Tomar** , Head Department of Computer Science & Engineering, at Jai Narain College of Technology, Bhopal. Whose kindness, valuable guidance and timely help encouraged me to complete this work; He / She exchanged her interesting ideas, thoughts and made this research easy and accurate.

I express my thanks to **Dr. Rajiv Srivastava,** Director of SIRT Bhopal for extending his support.

I express my thanks to **Dr. Ritu Shrivastava** HOD Computer Science & Engineering dept. **SIRT** Bhopal for kind support.

I express my thanks to the authors whose works have been consulted by me during the research.

I would like to thank the management for providing me lab facilities. Also I would like to thank the faculty **Prof. Arun Jhapate , Prof. Anupriya Suman, Prof. Komal Tahiliani and Prof. Amit Shrivastava**, without whom this research would have been a distant reality.

Abhishek Kumar (0133CS181006)
Anuj Kumar Sah (0133CS181028)
Dev Purohit (0133CS181050)
Gaurav Dubey (0133CS181060)

ABSTRACT

Algorithm visualization illustrates how algorithms work in a graphical way. It mainly aims to simplify and deepen the understanding of algorithms operation. Within the paper we discuss the possibility of enriching the standard methods of teaching algorithms, with the algorithm visualizations.

As a step in this direction, we introduce the **Algorithm Visualizer,** a platform that will provide an active visualization of algorithms by using web technologies and help in imparting the knowledge of algorithms to the individuals who are encountering challenges in interpreting fundamental algorithms concepts. By seeing how the algorithm works step by step, one can often understand it with just a glance, while going through the pseudo code and other written explanation can be time-consuming and frustrating

We have presented our practical experiences and described possible future directions, based on our experiences and exploration performed by means of a simple questionnaire.

CONTENTS

LIST OF FIGURES

Page No.

OBJECTIVES

The main objective of this project is to help beginners to be able to visualize the basic algorithms and get a better understanding of the underlying operations.

We believe our web application, **Algorithm Visualizer** will serve as a teaching and learning tool with three major components :-

Learning mode, providing step-by-step, graph-based visualization of sorting algorithms, such as bubble sort, quick sort, selection sort, insertion sort, heap sort etc.

Visual interface for manipulating data and data structures to duplicate the steps performed by the specific algorithm, an excellent way for users to test their understanding.

Statistics, providing an overview of the number and frequency of each type of mistake made and the step in the algorithm at which they occurred. Such statistics can be very useful to an instructor, who can use them to identify problematic areas and work on providing better explanation.

We also believe that our project will help many technical students to test and improve their knowledge of algorithms.

SCOPE

- Provides an Algorithm Visualization tool that is practical, reliable and helps in eliminating the complexity of already complex algorithms
- Present a system that can help in unlocking the hidden patterns within multiple algorithm

METHODOLOGY

- **Working Principle:**

The modern web applications are mostly based on client-server model. This project also follows the same model. All the codes and other misc. file like images and audios files are deployed to our server. The user just needs to click to our provided link to get into the frontend part of our Application. As soon as the user clicks the provided link, it will direct him/ her to the home page of our application.

In the main page, a dropdown list will be provided from which user have to select an algorithm which he/she wants to visualize.

Every algorithm's code is defined as a JS function, the event of selecting an item from the dropdown list is like loading the function's name which is supposed to be called,

The selected function will be called after hitting the START button, and then the animation will begin.

- **Technologies Used:**

The project, Algorithm Visualizer is developed using very basic web technologies:

HTML (Hypertext Markup Language) is the code that is **used** to structure a web page and its content. For example, content could be structured within a set of paragraphs, a list of bulleted points, or using images and data tables

CSS is the acronym of "Cascading Style Sheets". **CSS** is a computer language for laying out and structuring web pages (HTML or XML). This language contains coding elements and is composed of these "cascading style sheets" which are equally called **CSS** files.

JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat.

Bootstrap is a powerful toolkit - a collection of HTML, CSS, and JavaScript tools for creating and building web pages and web applications. It is a free and open source project, hosted on GitHub, and originally created by (and for) Twitter.

LITERATURE SURVEY

In addition to numerous hundreds of individual Algorithms Visualizer created over the years, there have also been many systems created to support AV development. A significant amount of research has been devoted to understanding the effectiveness of AVs. We examine some of the literature in this section.

Algorithm Visualization Development Several AV development systems have become well known within the CS educational community. We can divide their history into two parts: what came before the rise of Sun's Java programming language and widespread uptake of content delivered via the Internet, and what came after. Since the widespread use of Java, development seems to have moved away from AV authoring toolkits, and towards series of "canned" AVs. Pre-Java systems often came packaged with pre-generated AVs but allowed educators and even students the freedom to implement other AVs in a scripting language or by annotating a real program. Today's systems are frequently distributed as collections of AVs not tied to any operating system or environment. Some of these collections come as part of AV development systems and some do not.

Brown Algorithm Simulator and Animator (BALSA) introduced the concept of using program-generated animations to teach fundamental computer science concepts. Besides undergraduate education, the system was also used for debugging and research in algorithm analysis. Later systems developed by Brown include Zeus and JCAT

XTANGO is the most recent version of the TANGO visualization system. Developed by John Stasko at Georgia Tech, its goal is to allow students who are not experts in graphical systems to create algorithm animations by implementing the algorithm in C and then adding special calls to the XTANGO graphical library. XTANGO is still maintained and distributed (making it one of the few non-Java AV development systems still available).

POLKA (Parallel program-focused Object-Oriented Low Key Animation) [Stasko and Kraemer 1992], another Georgia Tech project, is described as "a general-purpose animation system that is particularly well-suited to building algorithm and program animations" [Stasko 2001]. Like XTANGO before it, POLKA allows nonexperts to author AVs without deep knowledge of graphics programming.

Swan [Shaffer et al. 1996], developed at Virginia Tech, provides visualizations of C and C++ programs annotated with calls to SAIL, the Swan Annotation Interface Library. Unlike most program annotation systems, SWAN supports the ability to provide significant user control over the behavior of the annotated program.

ANIMAL (A New Interactive Modeler for Animations in Lectures) [Rossling ¨ et al. 2000] incorporates lessons learned from pedagogical research . Developed at the University of Siegen in Germany, ANIMAL's developers believe that visualization systems should supplement (instead of supplanting) traditional approaches to CS education. Animal AVs tend to have limited user interaction, which is not crucial when used as a lecture aid

JAWAA (Java and Web-based Algorithm Animation) [Pierson and Rodger 1998] was developed at Duke University by Willard Pierson and Susan Rodger. It is a Java applet-based system that runs on script files generated by any programming language. It is similar to Animal in that the AVs it generates provide little user interaction. JAWAA assists the animation developer by providing primitives in a range of granularities, from basic graphics (circles, squares) up to data structures (arrays, trees). Generally, only a few lines of JAWAA script are required to animate an algorithm: one to create and display the structure, the others to perform operations on it.

Data Structure Visualization (DSV) was created at the University of San Francisco by David Galles. DSV aspires to be a comprehensive suite of AVs. DSV provides animations for a wide variety of algorithms and data structures, including several sorts, multiple varieties of trees, and several graph algorithms.

The Algorithms in Action project is directed by Linda Stern of the University of Melbourne. The original version was made available in 2000 and was quite sophisticated for its time in its use of multiple windows in a Java applet to track pseudocode, the visualization, and supporting explanatory text. An extensive collection of additional applets was released in early 2010 after classroom testing.

Effectiveness of Algorithm Visualizers

AVs can provide a compelling alternative to other types of instruction, particularly written presentations such as pseudocode and real code.

In lecture situations, students routinely report liking AVs [Gurka and Citrin 1996; Stasko et al. 2001], and faculty appear to strongly support them in principle [Naps et al. 2002]. The literature, however, has not demonstrated that AVs are always effective in practice. Results form a continuum from "no significant difference" [Gurka and Citrin 1996; Hundhausen and Douglas 2000; Jarc et al. 2000] to demonstrations that AVs can indeed improve understanding of data structures and algorithms [Shaffer et al. 2007; Lawrence et al. 1994; Hansen et al. 2000; Hundhausen et al. 2002]. The existing body of research helps to illuminate some of the features of AVs that make them effective.

Gurka and Citrin [1996] found little to no evidence that visualizations are effective teaching tools in a survey of previous experiments. They point out that while many fields are concerned with false positives in evaluation, the visualization community may need to concern itself with false negatives. They suggest repeating some "failed" experiments from previous projects with tighter controls on issues such as usability, algorithm difficulty, animation quality, and experimental design. They caution, however, that the community must be prepared to accept an ultimate finding of "no significant difference."

Hundhausen and Douglas [2000] found that students who construct their visualizations might be distracted by the creation process. Their experiment was based on the finding that learner involvement increases effectiveness, but technology does not necessarily improve the result. They found that student-created low-fidelity visualizations made of art supplies had no less impact on educational outcomes than high-quality computer-mediated AVs also created by the students

From the above reports, it is clear that. While more fundamental research on how to develop and use AVs is necessary, we also simply need more implementers to produce more quality AVs.

HARDWARE & SOFTWARE REQUIREMENTS

- **Hardware Requirements**

This project is primarily a Web Application; hence it can be executed on any device which have a stable net connection, and a web browser installed. Although it is recommended to use a device with a wide screen so that the visualization can be observed by the user clearly.

Components	Recommended
System Requirement	Intel Core i3-2100 2nd Generation,2GB RAM

- **Software requirements**

Any web browser which supports modern web technologies like HTML5, CSS, JS and Bootstrap can run this web application with its full potential.

Chrome Browser or Mozilla Firefox is recommended for better performance.

Browser Settings

- Java Script Enabled
- Cookies Enabled
- All Pop-Up Blockers **MUST** be disabled

DESIGN & FRAMEWORK

5.1 Software Process Model

Our main hurdle will be to find a streamlined visual interface which provides all necessary information, allows the user to modify the data and data structure involved in the algorithm, while being easy and intuitive to use. We will also build the system with expandability in mind, so that it can be extended with additional algorithms without too much difficulty.

The project has four stages: *Requirement* Gathering, Design, Infrastructure Implementation, and User Interface.

Stage1 - Requirement Gathering Stage.

Before commencing any project, it is important to clearly define, objectives, users, requirements, a realistic timeline and ways to fairly and efficiently divide work within the team.

Types of users: A wide variety of users can use our app, for example students, teachers and people who are just curious about algorithms.

User interaction modes:

Sorting Visualizer mode: All the sorting visualization operation will be performed on a random array of generated bars which will be sorted in real time allowing the learners to better understand how various sorting works

Path Visualizer mode: Upon selecting the preferred path algorithm from a set of algorithms (sorting visualization, Dijkstra etc) the path visualizer will find the shortest path from start node to end node

Real World Scenario 1

Suppose student has been taught about various sorting algorithm in school along with the code. But still the student is unable to understand the bigger picture of how sorting works and how some sorting algorithm are faster than other algorithms. Now this student can use the proposed sorting algorithm visualizer through which he/she will be able to visualize

How Algorithm depends on size of sorting data?

How Different algorithm internally work to give a sorted data?

How some algorithms are faster than others?

Real World Scenario 2

Leaning directly through code in case of algorithms is pretty overwhelming. The beauty of algorithm comes out when we visualize it. Curiosity is a fuel. It propels us to try to figure things out and, ultimately, to learn. Teachers can use our algorithm visualizer to spark the curiosity of students learning the algorithm for the first time.

Project Time-line and Division of Labour

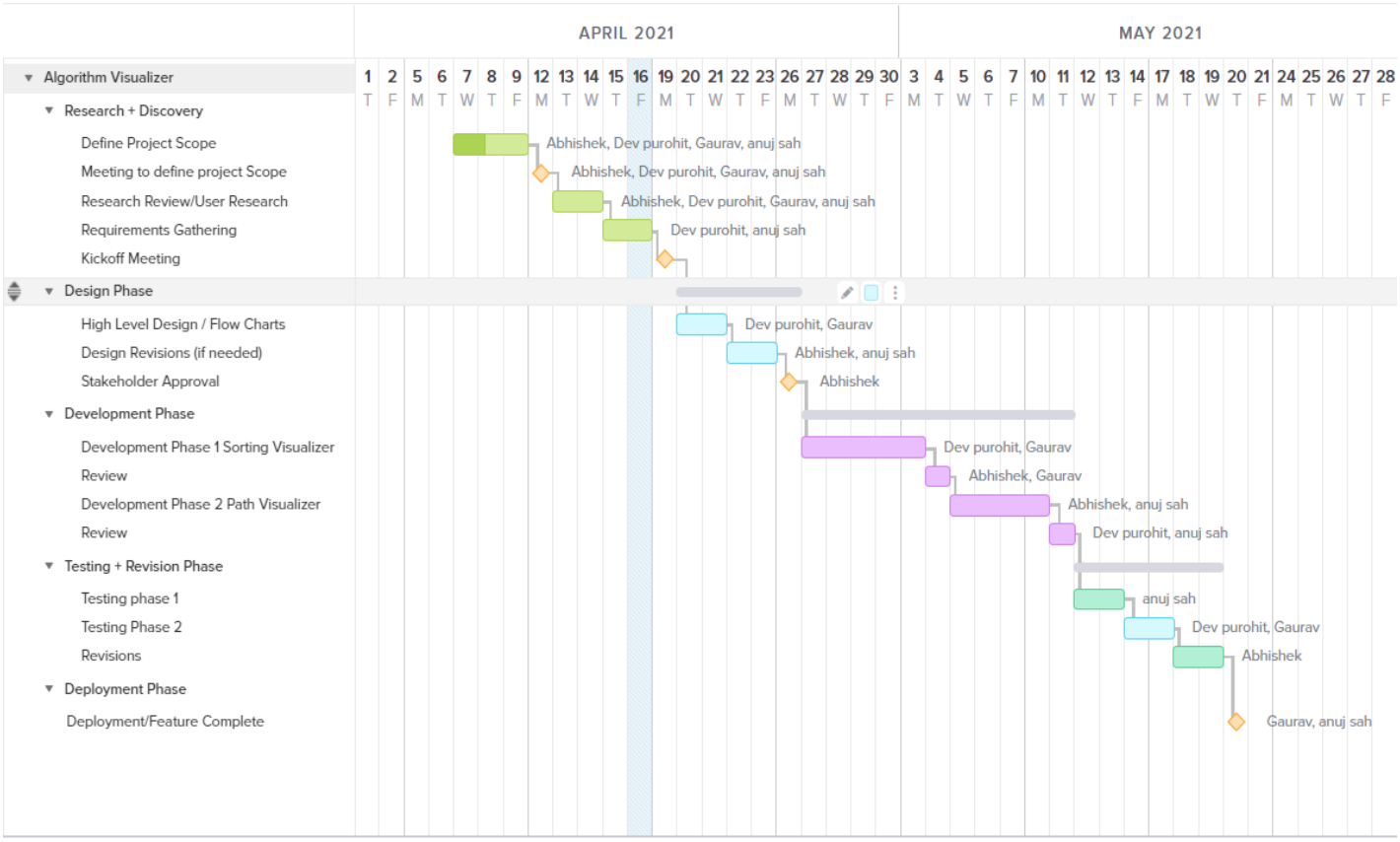

Figure 1: Project Timeline

Stage2 - Design Stage

Project Description

When the user opens the web page, he is presented with a choice to choose between the Path Visualizer and Sorting Visualizer

Suppose if the user chooses sorting visualizer then a new screen will be opened.

At the start the app will automatically generate a random array

This random array is presented on the screen as bars in unsorted fashion

Then user is prompted to select a sorting algorithm. User can select from a set of sorting algorithms mainly -

- Bubble Sort
- Insertion Sort
- Selection Sort
- Heap Sort
- Quick Sort

After which the program waits for user to press the sort button. Now as the sort button is pressed the program starts to sort the vertical bars in increasing order of height according to the visual speed selected using a scrollable slider

Similarly, if the user chooses the **path visualizer** on the start screen, a 2d grid is generated with starting and ending point. User now can add path blocks by tapping on the grid nodes. Similar to the sort visualizer here also visual speed can be set.

User can select from a set of path visualization algorithms including-

- Dijkstra's
- A*
- Depth First Search
- Breadth First Search

After this the user presses the start button, this will cause the application to find a shortest or any path from start node to the end node. Finally, we will see the path as set of connected green nodes.

Use Case Diagram

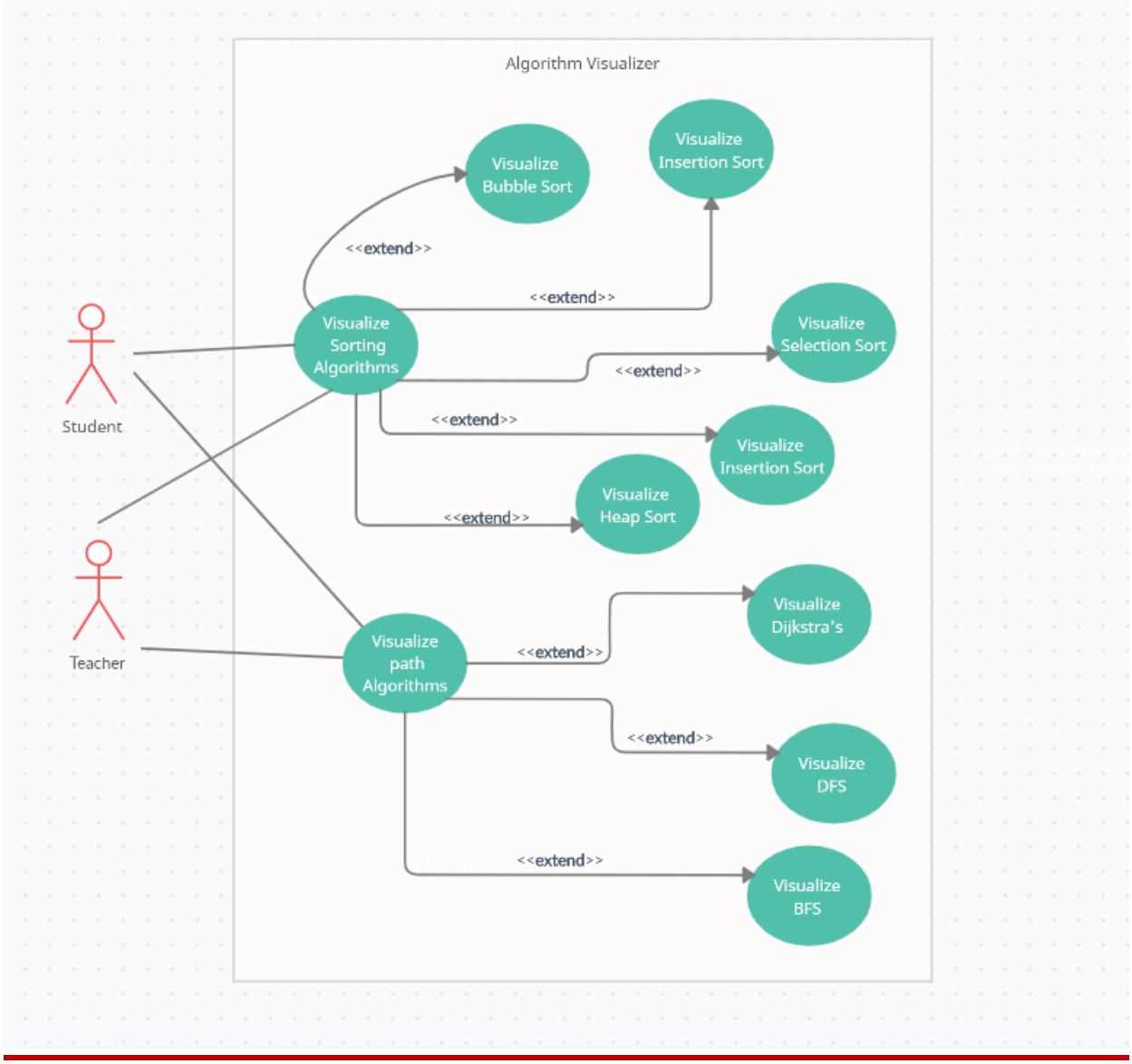

Figure 2 : Use case diagram of the algorithm Visualizer

Data Flow Diagram

- **Level 0 DFD**

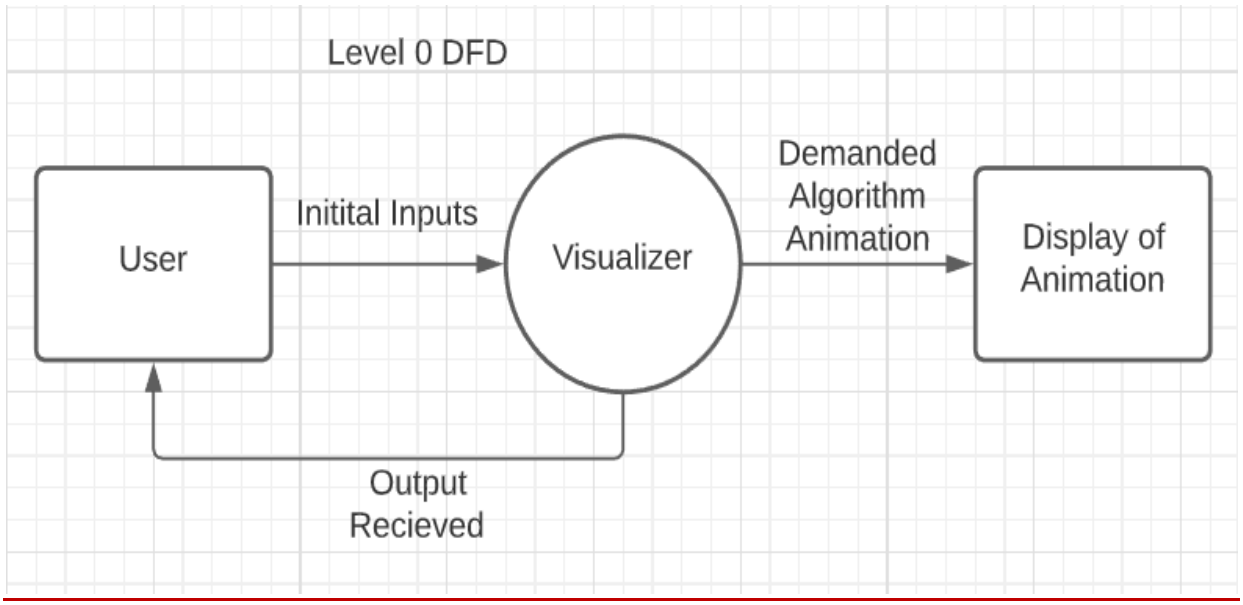

Figure 3 : Level 0 DFD of algorithm visualizer

- **Level 1 DFD**

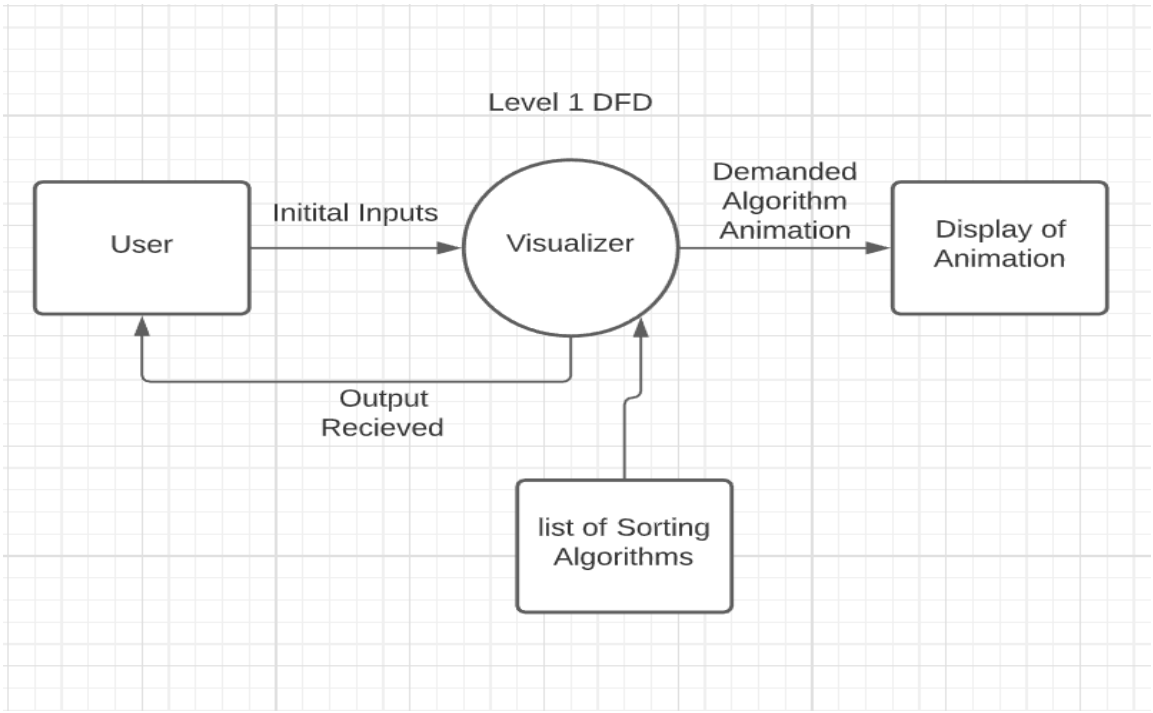

Figure 4 : Level 1 DFD of algorithm visualizer

- **Level 2 DFD**

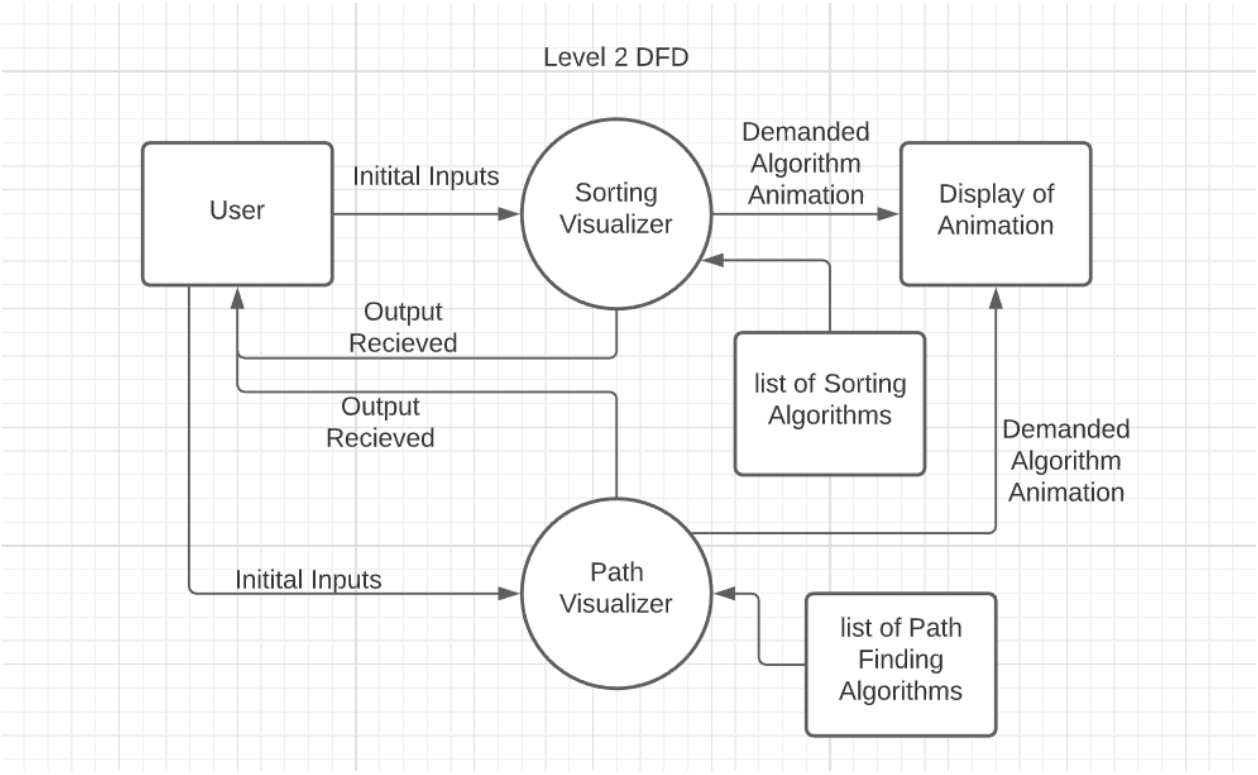

Figure 5 : Level 2 DFD of algorithm visualizer

Flow Diagram

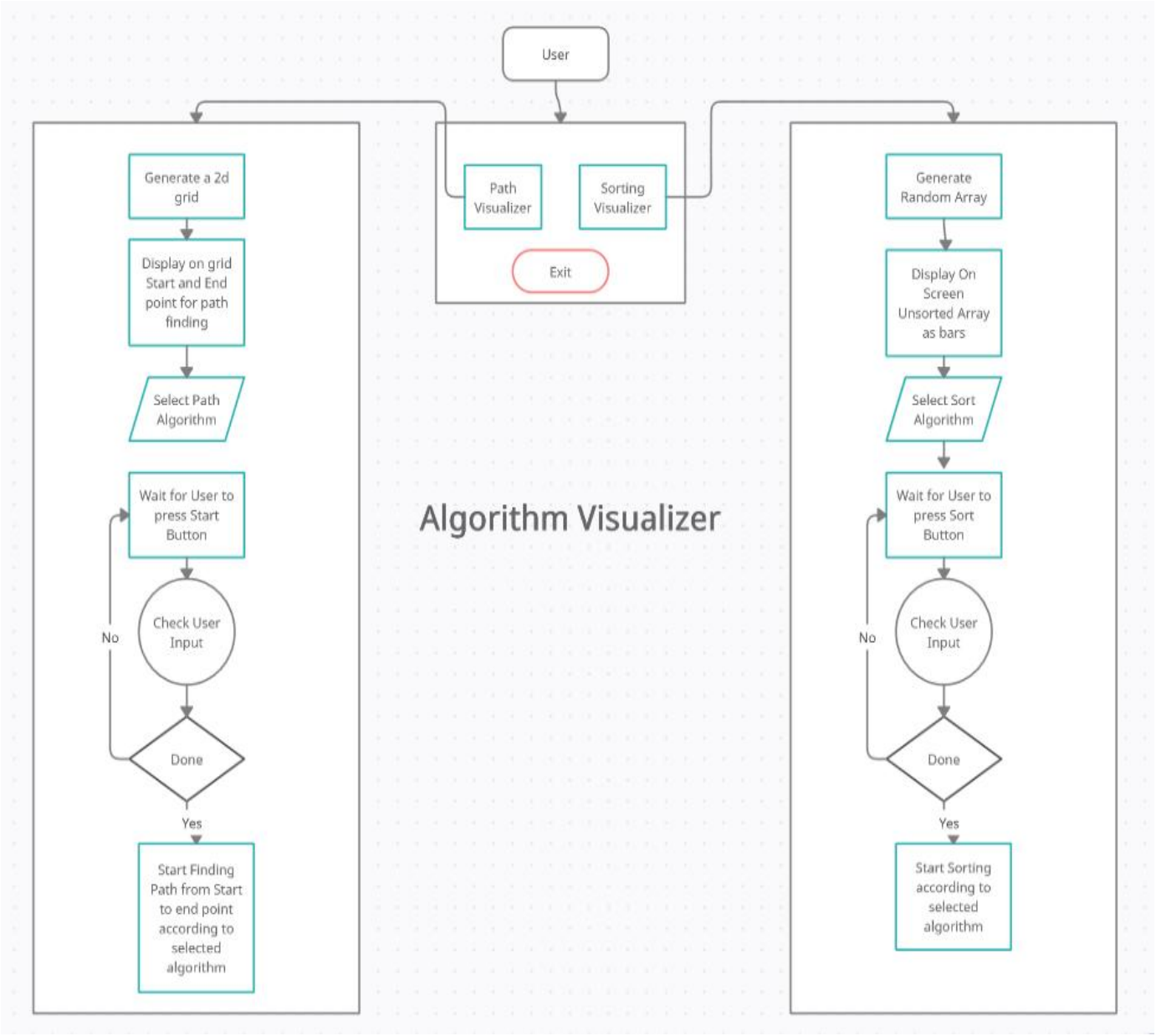

Figure 6 : Control Flow Diagram of algorithm visualizer

Architecture and Implementation

There are two building components in Algorithm Visualizer a) the user interface part, b) the animation engine.

The user interface is actually a simple web page written in HTML5 and CSS3. The animation engine uses JavaScript and Bootstrap for animations.

The user interface consists of a browser window divided into two main components:

- The animation area (down centre pane) where the animation of the algorithm is displayed, i.e., the constant changes of the graphical representations associated with the operation of the current algorithm command. The critical algorithmic parameters are highlighted or visualized by smooth transitions of objects.

- The control panel (down centre pane) includes operation buttons that support student-visualization interaction. There are five main control operations, e.g., start, pause, restart, sorting algorithm selector. There are also two sliders present. The middle slider is used to control the size of the array and the top-left slider aims to control the animation speed

The Animation Engine uses vanilla JavaScript functions like set Timeout to perform smooth animations which results in beautiful animations.CSS3 is also used to perform set the visual layout and perfect the animation using multitude of animation properties.

Key Features

- Visualization features: Our Algorithm Visualizer application represents an array by a series of contiguous vertical divisions that can move during execution of the algorithm. This helps student to build mental model and build a solid understanding on how the algorithms actually works.

- Web features: The Algorithm Visualizer project is designed as a fully web-based system written in HTML5, CSS3 and JavaScript. Therefore, by simply visiting the hosting URL, it runs in any platform, operating system, browser or device. Educators and students do not need to install any additional software or download any special-purpose package. It is also fully client-based so there is no overload due to the communication with any server.

- It provides customized visualizations that depend on the nature of the given algorithm.

- It allows user to change input data in order to test algorithm behaviour in specific cases.

- It is easy to use and offers full platform compatibility due to the web-based architecture.

CODING

Code for generating random array:

```
function GenerateArr() {
    //create a container for the array
    ArrayBox.innerHTML = "";
    for (var i = 0; i < n; i++) {
        //generate random numbers in array
        arr[i] = 1 + (Math.floor(Math.random() * 500) + 101) % 95;
        //create two divs one of them work as a margin!!! the other represent a number
//n the array
        divs[2 * i] = document.createElement("div");
        divs[2 * i + 1] = document.createElement("div");
        ArrayBox.appendChild(divs[2 * i]);
        ArrayBox.appendChild(divs[2 * i + 1]);
        //style the two divs
        divs[2 * i].style = "width: " + width + "%;";
        divs[2 * i + 1].style =
            " background: #808080; width:" + width + "%;" +
            "box-shadow: 1px -1px 2px #5F5F5F, 1px -1px 2px white; " +
            " height:" + arr[i] + "%;";
    }
}
```

Code for Visualizing BubbleSort algorithm:

```
function bubble(divs, height) {
    c_delay = 0;
    // delay_time = 40 - 2*(speed - 1);
    delay_time = 10000 / (Math.floor(n / 30) * speed);
    //console.log(c_delay, delay_time);
    for (var i = 0; i < n - 1; i++) {
        //let div one to be red
        div_update(divs[1], height[j], "red"); //Color update
        var j;
        //bubble sort normal algorithm
        for (j = 1; j < n - i; j++) {
            if (height[j] < height[j - 1]) {
                var temp = height[j];
                height[j] = height[j - 1];
                height[j - 1] = temp;
            }
```

```
            //swap colors
            div_update(divs[2 * (j - 1) + 1], height[j - 1], "#808080"); //Color update
            div_update(

                divs[2 * j + 1], height[j], "red"); //Color update
        }
        //turn the sorted number to be green
        div_update(divs[2 * (j - 1) + 1], height[j - 1], "green"); //Color update
    }

    div_update(divs[1], height[0], "green"); //Color update

    // enable_buttons();
}
```

Code for creating graph Board:

```
 function createBoard() {

    var grid = document.querySelector('.container');

    grid.innerHTML = '';

    for (var row = 0; row < rowsize; row++) {

        for (var col = 0; col < colsize; col++) {

            let weight = Math.round(getRandomArbitrary(5));

            let temp = createNode(row, col, weight);

            grid.appendChild(temp);

        }

    }

}


function createEmptyBoard() {




   var grid = document.querySelector('.container');

    grid.innerHTML = '';
```

```
    for (var row = 0; row < rowsize; row++) {
        for (var col = 0; col < colsize; col++) {
            let weight = 0;
            let temp = createEmptyNode(row, col);
            grid.appendChild(temp);
        }
    }
}
```

Code for creating and updating graph node:

```
// Create a Node
function createNode(row, col, weight) {
    var node = document.createElement('div');
    node.setAttribute('class', 'before_start');
    node.setAttribute('row', row);
    node.setAttribute('col', col);
    node.setAttribute('wall', 0);
    node.setAttribute('cost', Number.POSITIVE_INFINITY);
    node.setAttribute('parent', null);
    node.setAttribute('weight', weight);
    node.innerText = weight.toString();
    return node;
} /
function updateNode(node, row, col, weight, wall) {

    node.setAttribute('row', row);
    node.setAttribute('col', col);
    node.setAttribute('cost', Number.POSITIVE_INFINITY);
```

```
    node.setAttribute('parent', null);
    node.setAttribute('weight', weight);
    node.setAttribute('class', 'before_start');
    node.innerText = weight.toString();
    if (wall == 1) {
        node.setAttribute('wall', 1);
        node.className+=' wall';
    }
    else node.setAttribute('wall', 0);
    return node;
}

function createEmptyNode(row, col) {
    var node = document.createElement('div');
    node.setAttribute('class', 'before_start');
    node.setAttribute('row', row);
    node.setAttribute('col', col);
    node.setAttribute('wall', 0);
    node.setAttribute('cost', Number.POSITIVE_INFINITY);
    node.setAttribute('parent', null);
    node.setAttribute('border', '1px solid black');
    return node;
}

function updateEmptyNode(node, row, col, wall) {

    node.setAttribute('row', row);
    node.setAttribute('col', col);
    node.setAttribute('cost', Number.POSITIVE_INFINITY);
```

```
    node.setAttribute('parent', null);

    node.setAttribute('border', '1px solid black');

    node.setAttribute('class', 'before_start');

    node.innerText = '';

    if (wall == 1) {

        node.setAttribute('wall', 1);

        node.className+=" wall";

    }

    else node.setAttribute('wall', 0);

    return node;

}
```

Code for Visualizing DFS:

```
function dfs(x1 = 0, y1 = 0, x2 = rowsize - 1, y2 = colsize - 1) {
    time = slider.value;
    time = 40 + (time - 1) * (-2);
    container.removeEventListener('mousedown', setWallAttribute);
    container.removeEventListener('mouseover', setWallAttribute);
    var startNode = document.querySelector(`div[row='${x1}'][col='${y1}']`);
    var endNode = document.querySelector(`div[row='${x2}'][col='${y2}']`);
    // Hide button
    var btn = document.querySelector('.start');
    var refreshBtn = document.querySelector('.refresh');
    btn.style.visibility = 'hidden';
    refreshBtn.style.visibility = 'hidden';

    /* ################################### Algo here ############################3*/

    var seen = [];
    let counter = 1;
    fl=false;
    traverse(startNode, seen, counter, endNode);

    // Draw out best route
    setTimeout(() => {
        startNode.setAttribute('class', 'ends')
        while (endNode.getAttribute('parent') != 'null') {
```

```
            endNode.setAttribute('class', 'Path_green')
            var coor = endNode.getAttribute('parent').split('|');
            var prow = parseInt(coor[0]);
            var pcol = parseInt(coor[1]);
            endNode = document.querySelector(`div[row="${prow}"][col="${pcol}"]`);
        }
        endNode = document.querySelector(`div[row="${x2}"][col="${y2}"]`);
        endNode.setAttribute('class', 'ends');
    }, counter * time + 100);
    // Show refresh button again
    setTimeout(() => {
        refreshBtn.style.visibility = 'visible';
    }, counter * time + 100);
} // End start
```

SNAPSHOTS

Home Page of Algorithm Visualizer

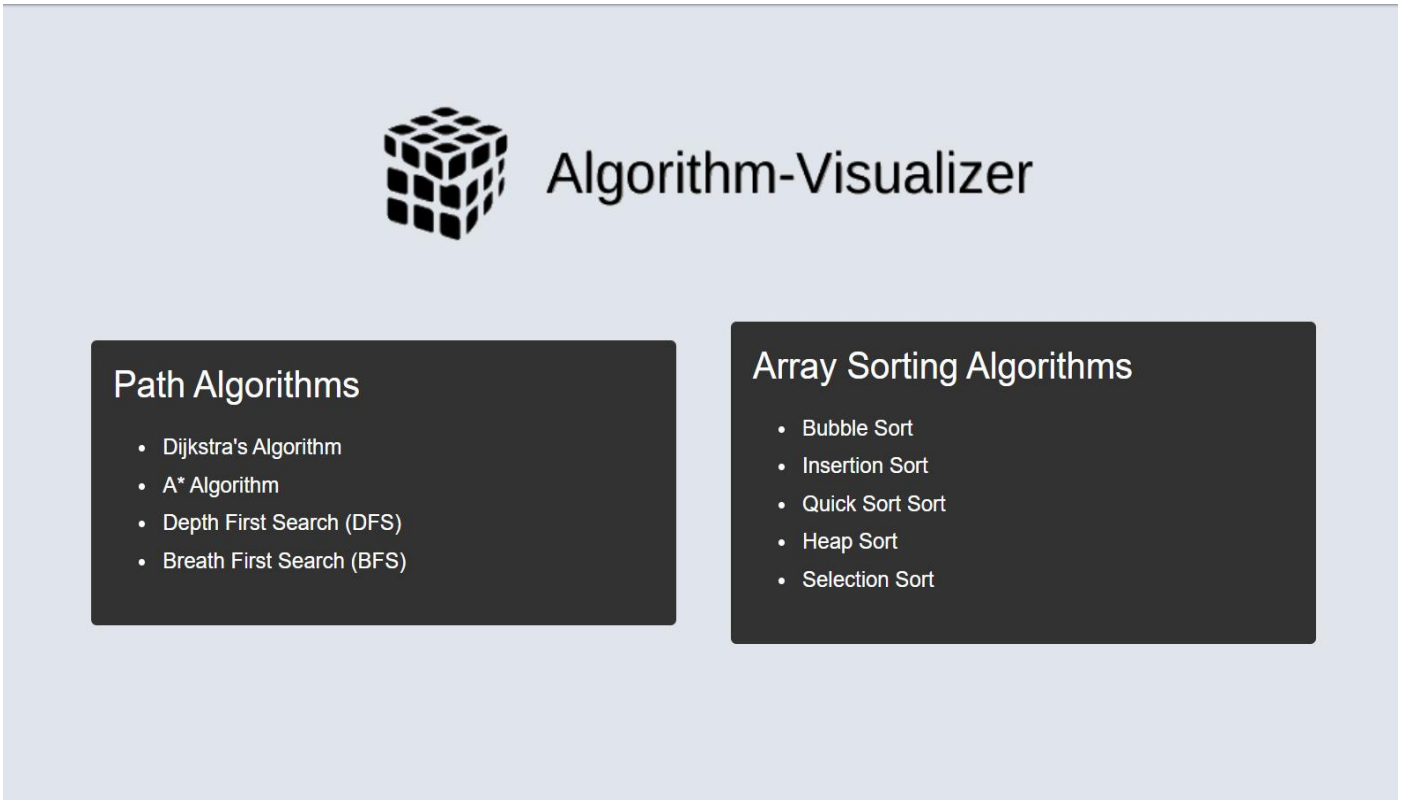

Figure 7 : Home page of the application

Sorting Visualizer in start phase

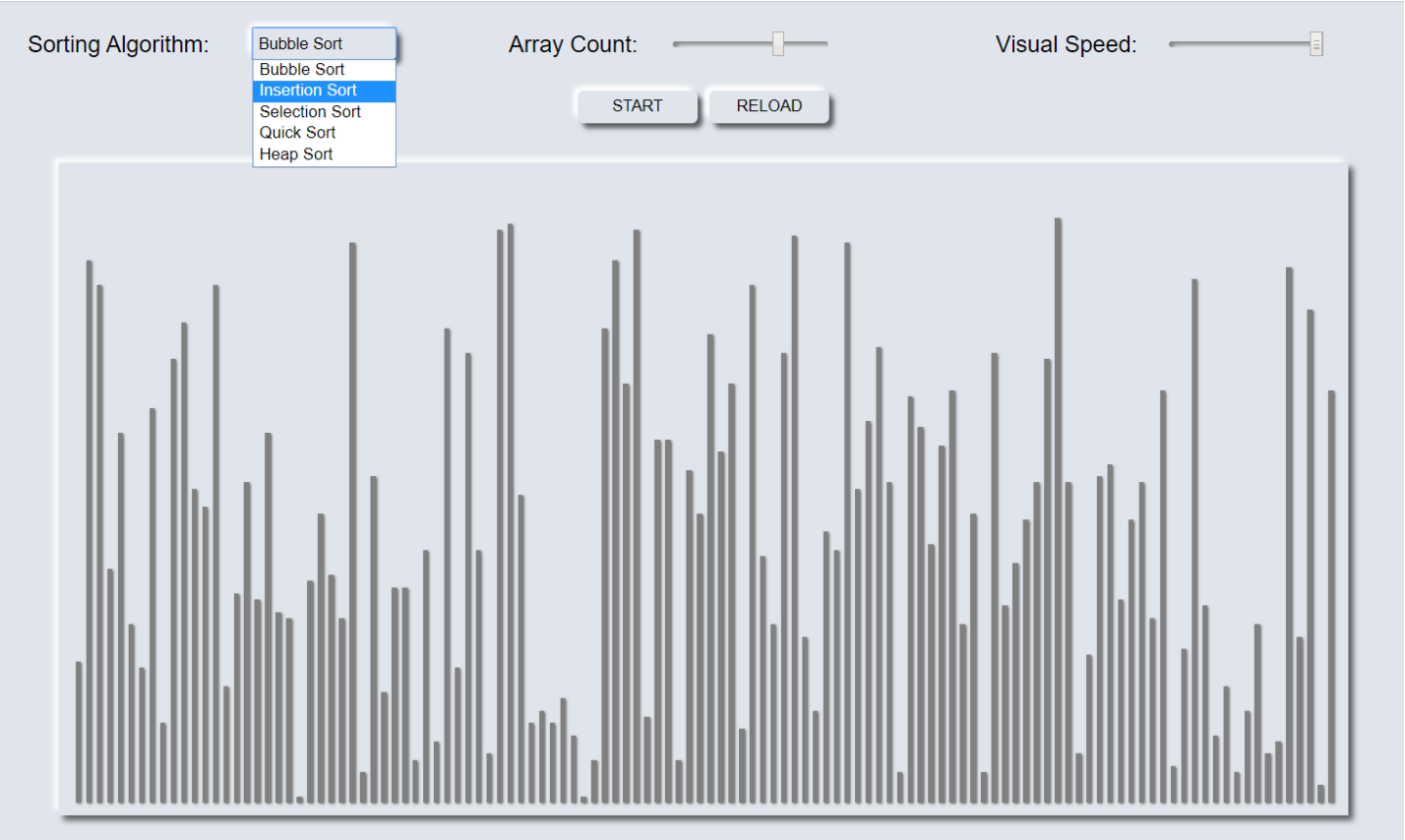

Figure 8 : Sorting visualizer page (with a randomly generated, unsorted Array depicted as bar graph)

Sorting Visualizer in Running phase

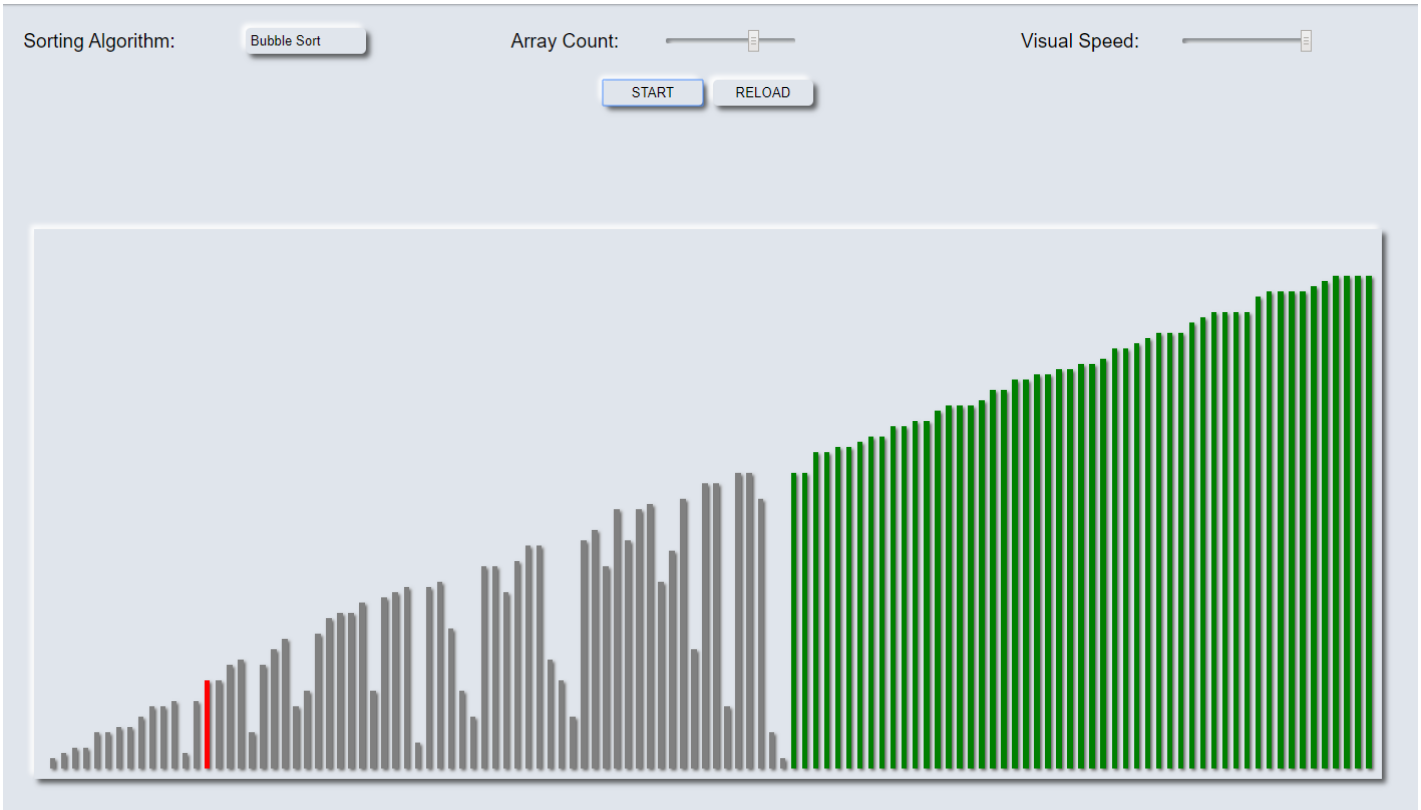

Figure 9 : A running visualization of Bubble sort

Path Visualizer in start phase

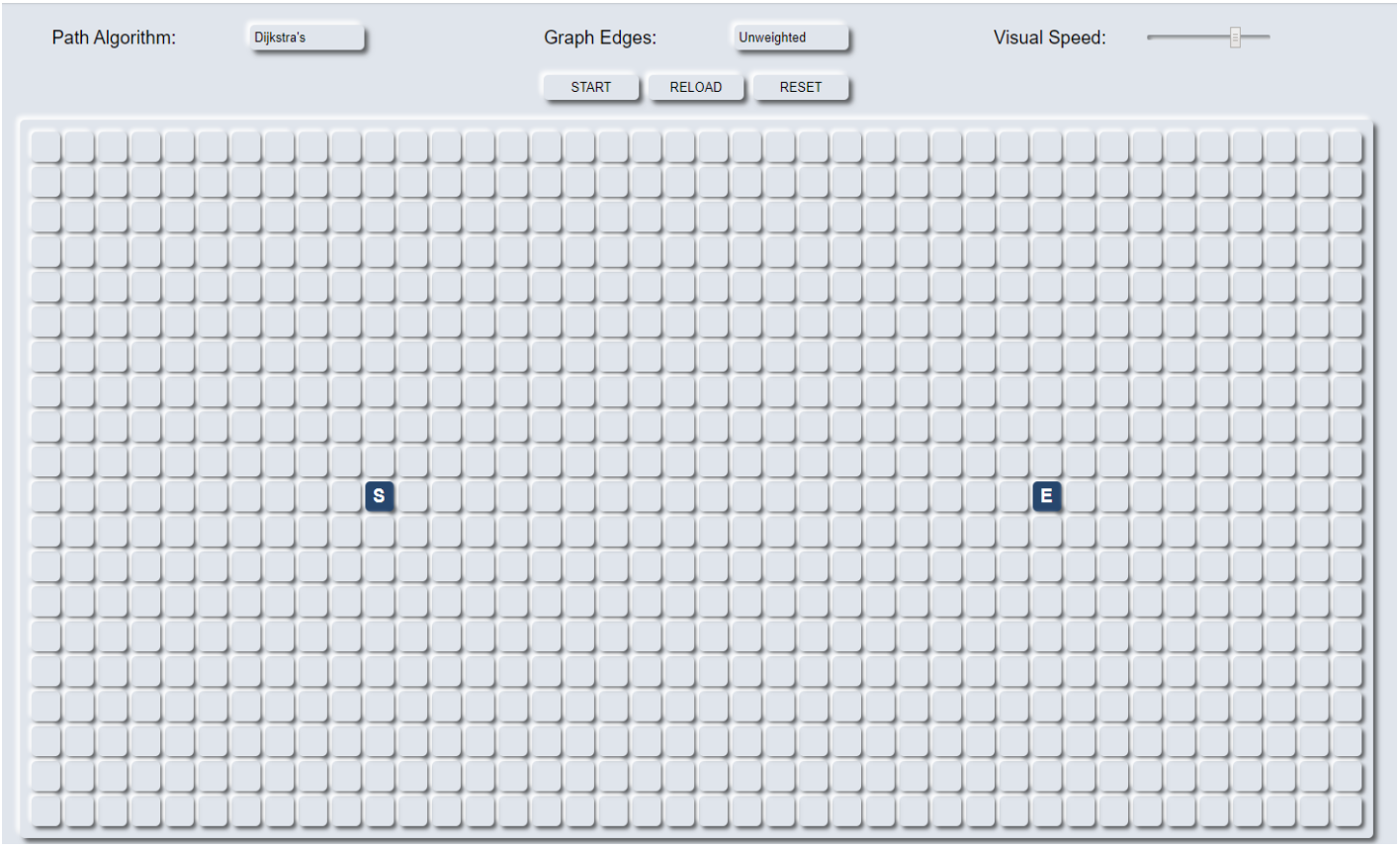

Figure 10 : Path Visualizer page (Path S to E is supposed to be traced)

Path Visualizer in Running phase

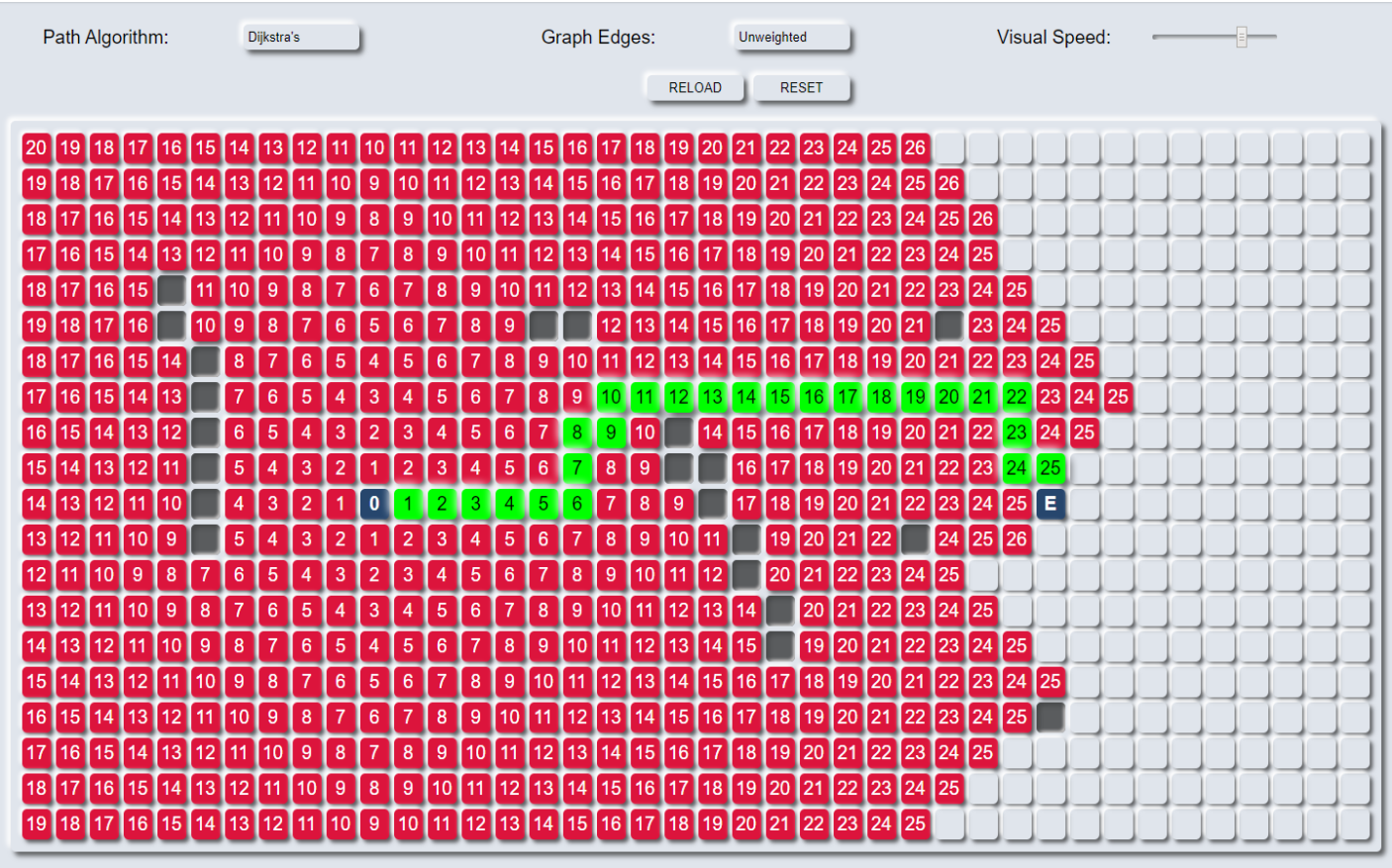

Figure 11 : A running visualization of Path Tracing using Dijkstra's Algo

APPLICATIONS

There are two principal applications of algorithm visualization:-

1.Research

2.Education

Potential benefits for researchers are based on expectations that algorithm visualization may help uncover some unknown features of algorithms. For example, one researcher used a visualization of the recursive Tower of Hanoi algorithm in which odd- and even-numbered disks were coloured in two different colours. He noticed that two disks of the same colour never came in direct contact during the algorithm's execution. This observation helped him in developing a better non-recursive version of the classic algorithm.

The application of algorithm visualization to education seeks to help students learning algorithms. The available evidence of its effectiveness is decisively mixed. Although some experiments did register positive learning outcomes, others failed to do so. The increasing body of evidence indicates that creating sophisticated software systems is not going to be enough. In fact, it appears that the level of student involvement with visualization might be more important than specific features of visualization software.

In some experiments, low-tech visualizations prepared by students were more effective than passive exposure to sophisticated software systems.

The application of algorithm visualization to education seeks to help students learning algorithms. The available evidence of its effectiveness is decisively mixed. Although some experiments did register positive learning outcomes, others failed to do so.

The increasing body of evidence indicates that creating sophisticated software systems is not going to be enough. In fact, it appears that the level of student involvement with visualization might be more important than specific features of visualization software. In some experiments, low-tech visualizations prepared by students were more effective than passive exposure to sophisticated software systems.

LIMITATIONS

Although this web application can be very helpful but obviously it cannot be treated as a magic wand which will solve every problem i.e. certain limitations are also there ,

i)This application works on random values which is auto generated from a section of code, i.e. there is no such option where the user can perform algorithm visualization on the certain desired values inserted by user itself.

ii)The application provides a fixed set of algorithms, one may not find a particular algo which he/ she wants to visualize.

iii) The speed controlling slide can be sometime hard to handle.

iv) The ability to manipulate algorithm code is not available.

FUTURE ENHANCEMENT

While many good algorithm visualizations are available, the need for more and higher quality visualizations continues. There are many topics for which no satisfactory visualizations are available.

Yet, there seems to be less activity in terms of creating new visualizations now than at any time within the past ten years.

On the other hand, the theoretical foundations for creating effective visualizations appear to be steadily improving. More papers are appearing regarding effective use of visualization in courses, and a body of work has begun to form regarding how to develop effective visualizations in the first place.

There are a lot of features and functionalities that can be integrated in the proposed system but the project scope has been limited to diligently resolve the problems as identified in the problem areas above. The project objective has to be achieved pertaining to the Time Constraint and Monetary constraint applied in accordance with the defined functionality of the system. However, features that are not included in the system can be considered as future enhancements. The limiting areas of the project contributing for enhancement thus are as follows, namely,

Presently the system aims to incorporate the feature of on the go algorithm visualization but does not allow user to change the algorithm parameters which would have given it a more surreal feel. The enhancement shall be to create a separate compiler which can allow algorithm change and show it instantly on the animation window.

Next, due to availability of infinitely many algorithms the project can be enhanced by adding more algorithms to the project.

Collectively, the community is learning how to improve. While more fundamental research on how to develop and use algorithm visualizations is necessary, we also simply need more implementors to produce more quality visualizations. And we need to encourage them to provide visualizations on under-represented topics.

REFERENCES

LGOVIZ WIKI. 2010. Data structures and algorithm visualization wiki.

http://web-cat.cs.vt.edu/AlgovizWiki.

ALGOVIZ.ORG. 2010. Annotated bibliography of the AV research literature.

http://algoviz.org/biblio.

BAECKER, R. AND SHERMAN, D. 1981. Sorting out sorting. Video.

BRABEC, F. AND SAMET, H. 2003. Maryland spatial index demos.

http://donar.umiacs.umd.edu/quadtree/.

BROWN, M., NAJORK, M., AND RAISAMO, R. 1997. A java-based implementation of collaborative algorithm visualizer

Bubble Sort Visualizer - https://www.geeksforgeeks.org/bubble-sort-visualization-using-javascript/?ref=rp

Sorting Algorithms - https://www.geeksforgeeks.org/bubble-sort-visualization-using-javascript/?ref=rp

Path Algorithms- https://www.geeksforgeeks.org/a-search-algorithm/
