# Graph Path Visualizer

**A PROJECT REPORT**

*Submitted by*

PRANAV CHAURASIA, 18BCE0216
MOHD UMAR, 18BCE0196
SHARMA TAVISH SANDEEP PRAKASH, 18BCE0867

CSE4019
Human Computer Interaction

Under the guidance of
**Dr. SHASHANK MOULI SATAPATHY**
**Associate Professor Grade 1, SCOPE,**
**VIT University, Vellore.**


**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**November 2020**

**Table of Contents**

**Abstract**

This project implements a website called Graph Path Visualizer. The website basically visualises various algorithms on a grid which has walls, weights, and bombs. The main motive was to make an application for educational purpose that can easily teach and show practically the complex graph algorithms. One of the most mainstream ways discovering calculation is known as the Dijkstra's shortest way first calculation. as a learner's progression to calculation and beginning to comprehend how they work we chose to make this visualizer that show in real life how the way discovering calculation works in real life. The comprehension of this calculation gives you the base thought of how route instruments are executed. In this visualizer, we first go through the tutorial provided, after which we land on the main page. Here, we make walls, mazes, patterns on the grid, finalize the initial and final nodes, select the graph algorithm of our choice, select the comfortable animation speed, and then visualize our setup. At the end of the visualization we discover how the different algorithm spread out in different ways and how the shortest path found for them is different. We also realize the amount of time taken by the different algorithms. Another feature of this website is that after the visualization is done, we can further work upon the same setup – change the initial and final positions, change the walls, etc, the visualization will get updated in real-time as we change it. We can also add bombs to our grid, which means the visualization will try to find out the shortest path which first reaches the bomb, and then to the final node. The website has been developed following most of the principles of UI design and looks appealing. Simple web tools like HTML, CSS, JS has been used to develop the project, and it can run on any computing device with a display and a pointing device. The website can also be further modified to suite various other needs like maze-solving, optimal flight-agenda for aeroplanes, map-navigation, etc.

## 2. Introduction

Simulated visualizations of algorithms have become a popular and effective learning tool. In the past, algorithms have been taught through exhaustive chalkboard drawings and pseudo-code guidelines. Visualization tools are therefore an attractive learning essential for instructors and students alike. Increased performance in languages such as Java along with robust graphical interfacing libraries such as Swing and JOGL make this a conquerable problem. In this paper, we also present a tool to visualize our smoothest path algorithm. Our goal is to simplify the algorithm into a series of visual steps that can be understood by elementary computer science students.

The usability of the e-learning tools is one of the main requirements for successful application of e-learning systems [1]. It is pointed out that such systems should be interactive and provide feedback, should be directed to specific goals and to motivate without allowing any factor of nuisance that interrupts the training stream. Visual aids are widely recognized as amplifiers of the learning capabilities of students facilitating the processing of complex information [2]. Various algorithms can be illustrated through static visualization but trend in computer science education is to make the learning process more individualized and interactive. Visualizations through interactive algorithms give the students more autonomy in their learning process.
Students learning DSA find it difficult to understand the visualizations of the different algorithms and their pattern of path-explorations. The website will help them visualize the paths easily. It will help them realise the difference between the different algorithms and their pros and cons.
Normal users can design their own pathways and find out the shortest path. Users can now complex maze puzzles using the different solutions using different algorithms The modern information and communication technologies provide means for easy presentation of information in a dynamic form according to the user preferences.

In our project we use different types mazes to draw the walls on the grid cells, using both free-hand and the predefined ones, and then select a path finding algorithm among the available ones, select the animation speed, and view the animation.
The various algorithm used are:

**Breadth First Search**

Breadth First Search explores equally in all directions. This is an incredibly useful algorithm, not only for regular traversal, but also for procedural map generation, flow field pathfinding, distance maps, and other types of map analysis. This may be the algorithm of choice to identify nearby places of interest in GPS. BFS guarantees the shortest path.

**Depth First Search**
It traverses by exploring as far as possible down each path before backtracking.
As useful as the BFS: DFS can be used to generate a topological ordering, to generate mazes, to traverse trees, to build decision trees, to discover a solution path with hierarchical choices… DFS does not guarantee the shortest path.

**Dijkstra**

Dijkstra's Algorithm lets us prioritize which paths to explore. Instead of exploring all possible paths equally, it favors lower cost paths.
We can assign lower cost to encourage moving on roads while assigning high cost on highway to avoid them.
It is the algorithm of choice for finding the shortest path paths with multiple destinations.

**A* (A-Star)**
A* is a modification of Dijkstra's Algorithm that is optimized for a single destination.
Dijkstra's Algorithm can find paths to all locations; A* finds paths to one location. It prioritizes paths that seem to be leading closer to a goal.
In a game, we could set costs to be attracted or discouraged in going near some objects : how useful it is for an AI.
It is more or less the golden ticket or industry standard algorithm for all applications finding directions between two locations.

**Breadth First Search (BFS)**
Breadth First Search (BFS) is one of the two most fundamental graph traversal algorithms. First published in 1959 by Edward F. Moore to find the shortest path out of a maze, it is now used in a daily basis not only for regular traversal, but also for networks analysis, GPS, Search Engines, scheduling and other types of graph analysis.
In short : this is an incredibly useful algorithm that guarantees the shortest path between two nodes (when connections have the same cost).
Breadth First Search explores equally in all directions until the goal is reached. In other words, it starts from a chosen node and examine all its neighbors, then it examines all the neighbors of the neighbors, and so on… If we consider a tree (which is a simplified graph), the BFS will proceed as follows:

## 5. Background / Related Work

Many researchers have attempted to design and develop individualized learning environments based on learning styles [3,4]. Visualizations of work of nets are an effective pedagogical approach. The first significant attempt to assess the pedagogical effects of algorithm visualization was conducted by Hundhausen [5]. A user requirements study revealed that personal learning environments are not seen as persistent environments, but they should evolve according to the learner's objectives and achievements [6]. Good algorithm visualization brings algorithms to life by graphically representing their various states and animating the transitions between those states. They illustrate data structures in natural, abstract ways instead of focusing on memory addresses and function calls [7]

## 6. Real-life Applicability

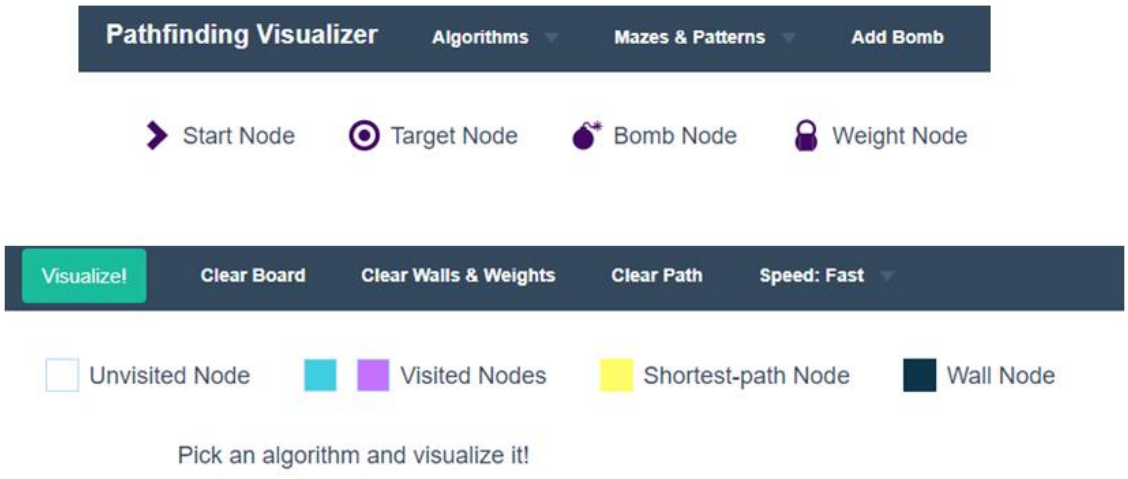
The project was developed with the aim of making topics of graph algorithms easier to learn for the students. The website developed helps us to show how the algorithm spreads among different paths – in what manner the grid cells are visited. The difference in the time taken to reach the final destination, and ofcourse the path taken.

However, we did some more research and we found out that there could be more uses of this :

- In Network routing protocols.
- In maze solving
- In digital mapping services such as Google Maps.
- In Biology by finding the network model in spreading of infectious diseases.
- In finding the optimal agenda of flights.
- In path planning for mobile robot transportation…

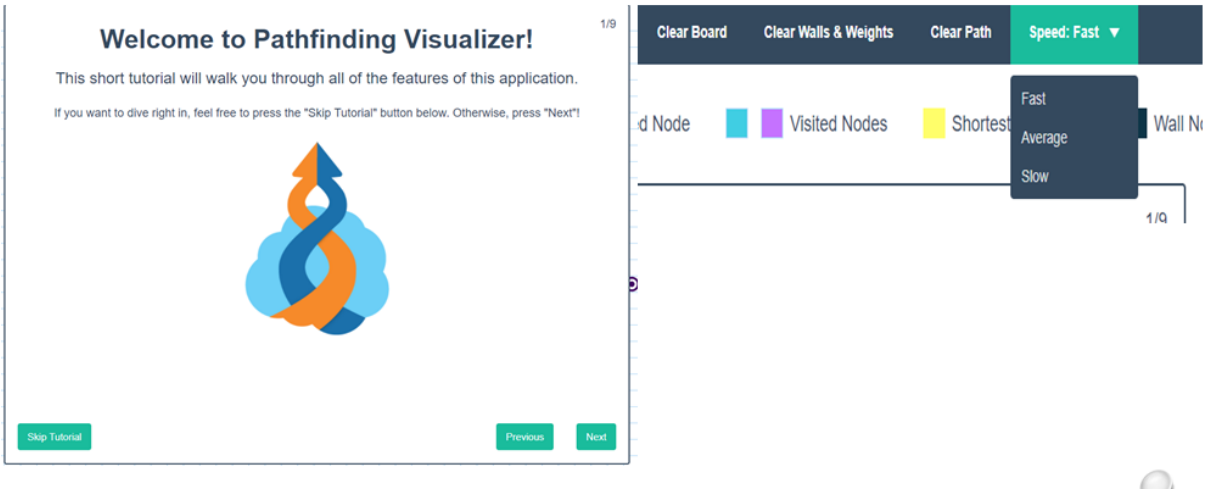
## 7. Individual Contribution

Module 1:
Catchy Design Phase
Nav bar and menu made using hci principles

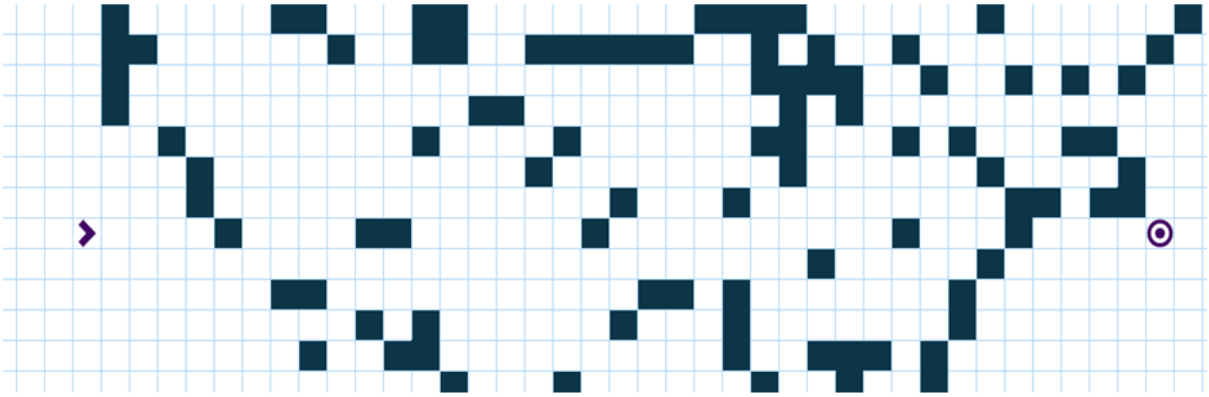



-By Tavish

Module 2:Tutorial and controllers
-Incorporating the golden rules in the UI/UX design




Module 3:Grid Development
-By Pranav
-Developing grid using HCI principles (more user friendly and easy to use



Module 4: Incorporating Js modules(Testing Grid)
-By  Tavish and Umar

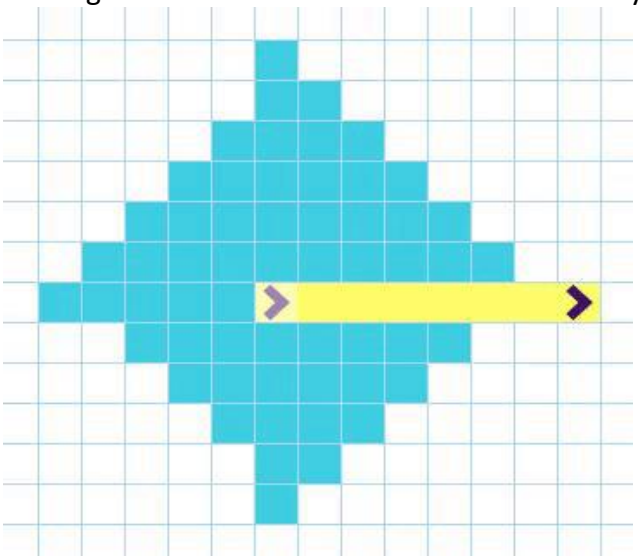
-Here both of them tested the grid, improvised and
Made sure about the usability testing, and easy of
User and further improvised the devised grid.
-The main ideology behind the following test was to develop
A grid which could easily encapsulate the mnemonics of the
Vivid imaginative intuition which we want to provide learners with this web application

Module 5:Random maze Algorithm
-By Umar
-this algorithm generates a random maze but with such precision
That in any two open spaces can be connected, I.e there always will be a possible solution

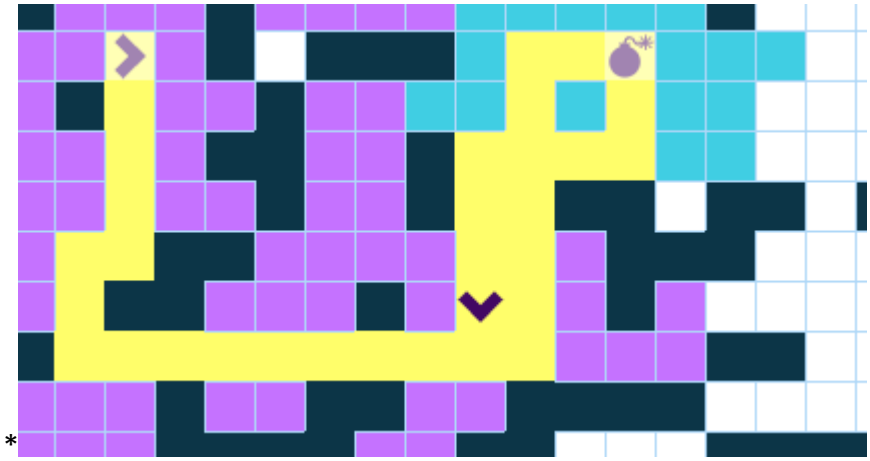
Module 6:Custom maze Algorithm
-By Tavish
-These algorithms were taken from research papers and then coded in javascript for the use for this project

That in any two open spaces can be connected, I.e there always will be a possible solution

Module 7: Algorithm Testing
-By Pranav
- Here all the coded algorithms were tested by us
In the grid and all the interactions were tested by us



Module 8:Color Code Maze
-By Tavish and Umar
-Here we tried incorporating different colors for different paths
-and incorporated various colors for different nodes



*

**8. Tools and Technologies used / Hardware – Software Requirements**

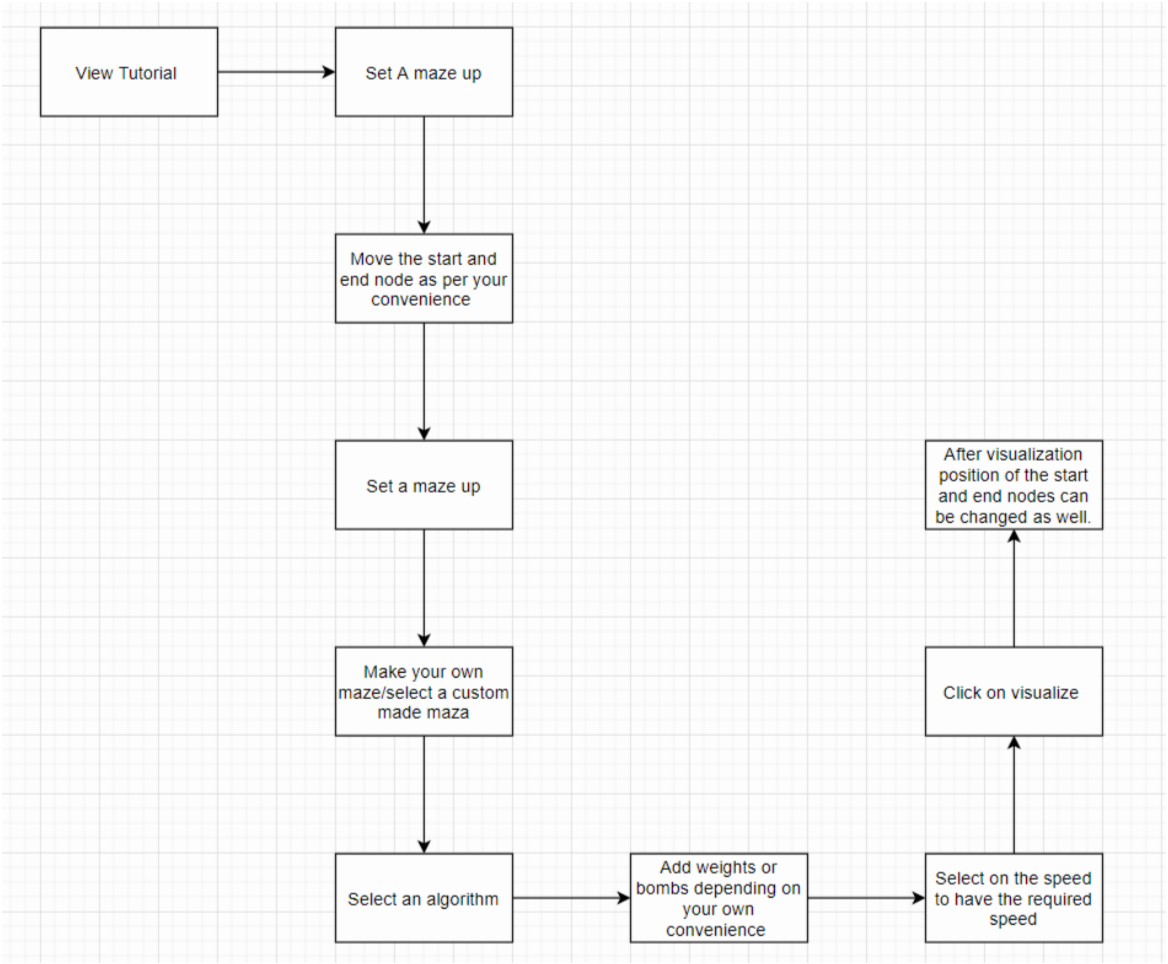
Web Tools used for software development : HTML, CSS, JavaScript

Hardware Requirements: A working computer of any configuration with a pointing device and a display

Software Requirements: An updated Web Browser

## 9. Proposed System Process Flow (Pictorial Representation)

The self-explanatory diagram shown below shows the run-through of our entire project



## 10. Working Methodology

The Grid:

In this grid, each box represents a node. A connection between two nodes is represented by a path between two boxes. This information is also displayed textually (top left box in the image). Here, our mouse is on the first node (0,0) and displays connections with the nodes (0,1) and (1,0).
We choose to describe our grid (or graph) using an adjacency lists data structure. The idea is pretty simple : an array of nodes with node storing the list of their neighbors. Easy to create, easy to manipulate, here is how the data could be represented in JSON :

```
[
  {
    "x":0, "y":0,
    "neighbors":[1,3]
  }, {
    "x":0, "y":1,
    "neighbors":[0]
  }
  ...
]
```

The node id is given by its array index, this id is used to refer to the neighbors.
It includes all the information we need to go through and to visualize the maze.

The most basic output that could be expected is the path found between two nodes. To represent this, an ordered (by step order) array is quite sufficient:

```
[0, 3, 19, 9, ...]
```

Starting from the node 0, we go to node 3, then to node 19 etc. until the goal.

*Note: A grid can use a non-grid pathfinding graph, or vice versa.*
*Here, grids are used for explanations as it is easier to understand, work and visualize.*

Below, we look into a rough working of some of the algorithms implemented in this project.

**Breadth First Search (BFS)**

Steps

- Add the start node in the queue and mark as visited.
- While there is a node waiting in the queue:
- 1. **Take the node** at the front of the line (queue).

- 2. **Add to the queue all available neighbors, note the parent and mark as visited**
- Done: backtrack from goal to start using parent link in order to get the shortest path.

**Depth First Search (DFS)**

Steps

- Add the start node in the stack and mark as visited.
- While there is a node waiting in the stack:
- 1. **Take the node** at the top of the stack.
- 2. **Add on the stack all available neighbors in order, note the parent and mark as visited** (we chose for the visualization North, East, South, West as order)
- Done: backtrack from goal to start using parent link in order to get the path.

**Dijkstra**

Steps

Let's take a **root node** and note **D the distance from a node to the root**. Dijkstra's algorithm will assign some initial distance values and improve them step by step.

- 1. Set all node distance D to infinity except for the root (current) node which distance is 0.
- 2. Mark current node as visited and **update D** for all of its unvisited neighbors **with the shortest distance**.
  Note: we can stop the algorithm here if we have reached the target node.
- 3. Add to the queue the unvisited neighbours and go back to step 2.

**A* (A-Star)**

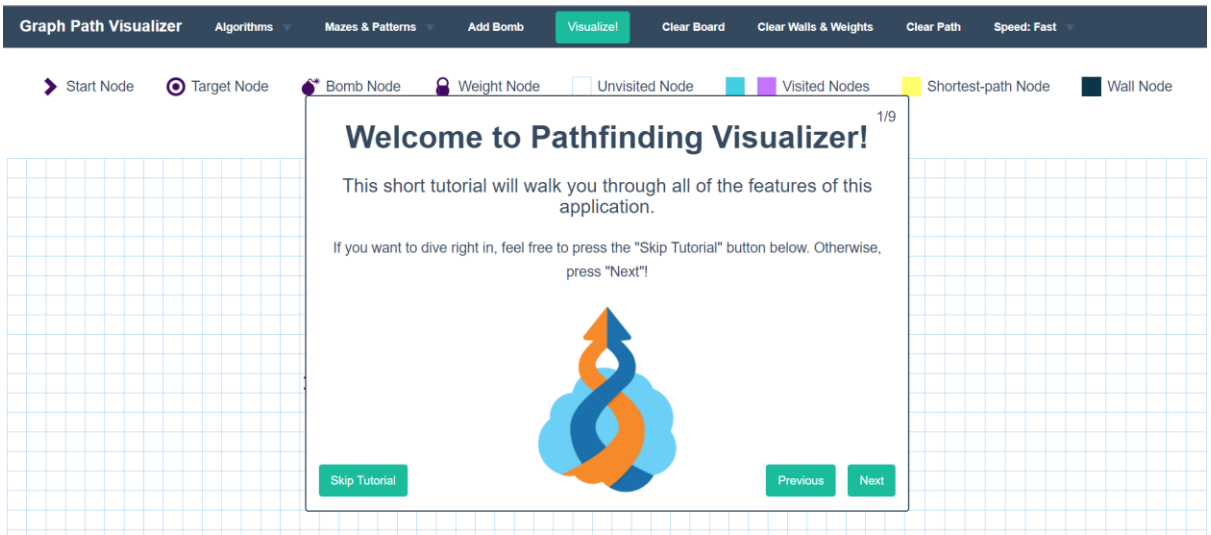
Steps

First we will assign some initial distance values and improve them step by step.

- 1. Set all node distance D to infinity except for the root (current) node which distance is 0.
- f(n)=g(n)+h(n), where:
  — **g(n)** is the cost of the path from the starting point to node n (**Dijkstra distance**).
  — **h(n)** is the estimated cost of the path from node n to the destination node, as computed by the **Manhattan distance** in our case.
- 2. Mark current node as visited and **update D** for all of its unvisited neighbours **with the shortest distance**.
  Note: we can stop the algorithm here if we have reached the target node.
- 3. Add to the queue the unvisited neighbours and go back to step 2.
- We minimise the manhattan distance : $h(n)=|goal.x-root.x|+|goal.y-root.y|$

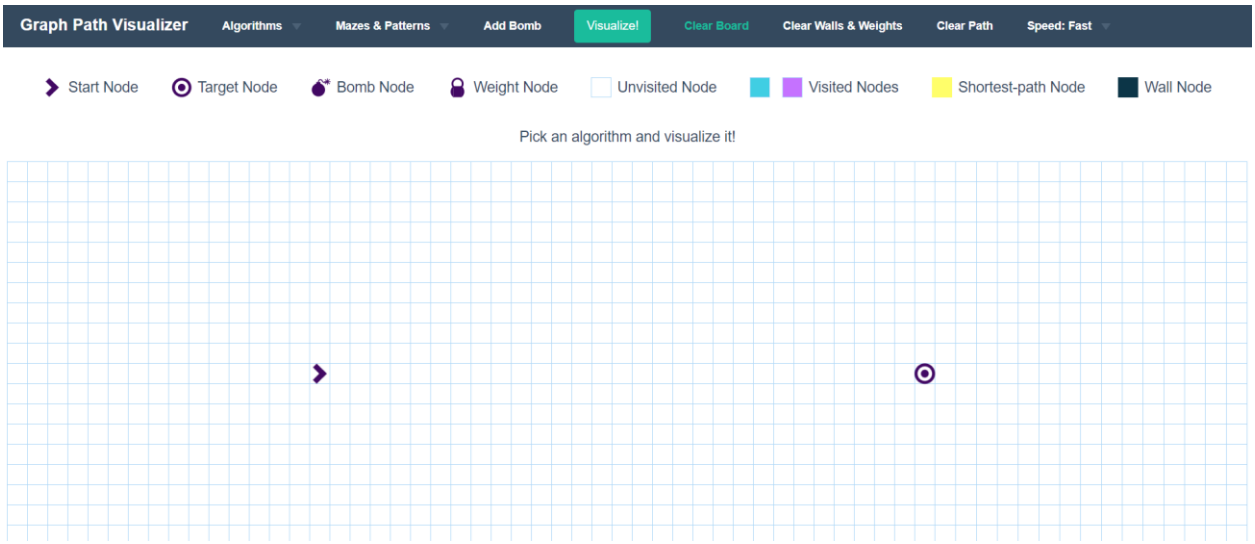
## 11. Implementation Results and User Interfaces (Explain each result and user interface)

On landing on the website go through the tutorial shown:

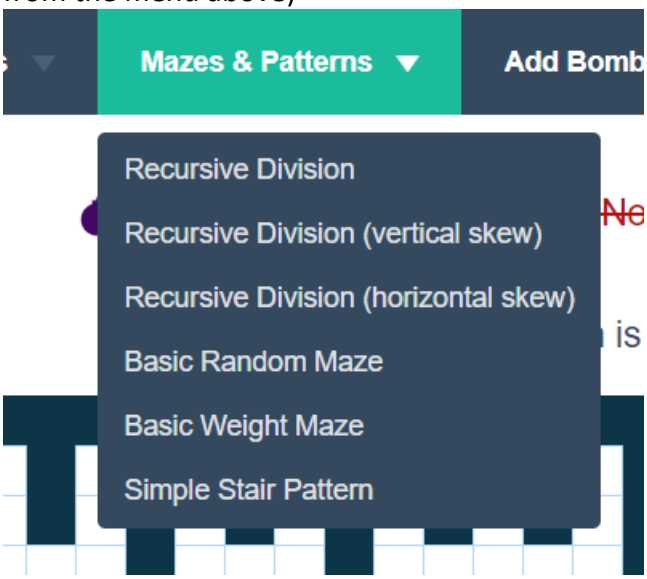


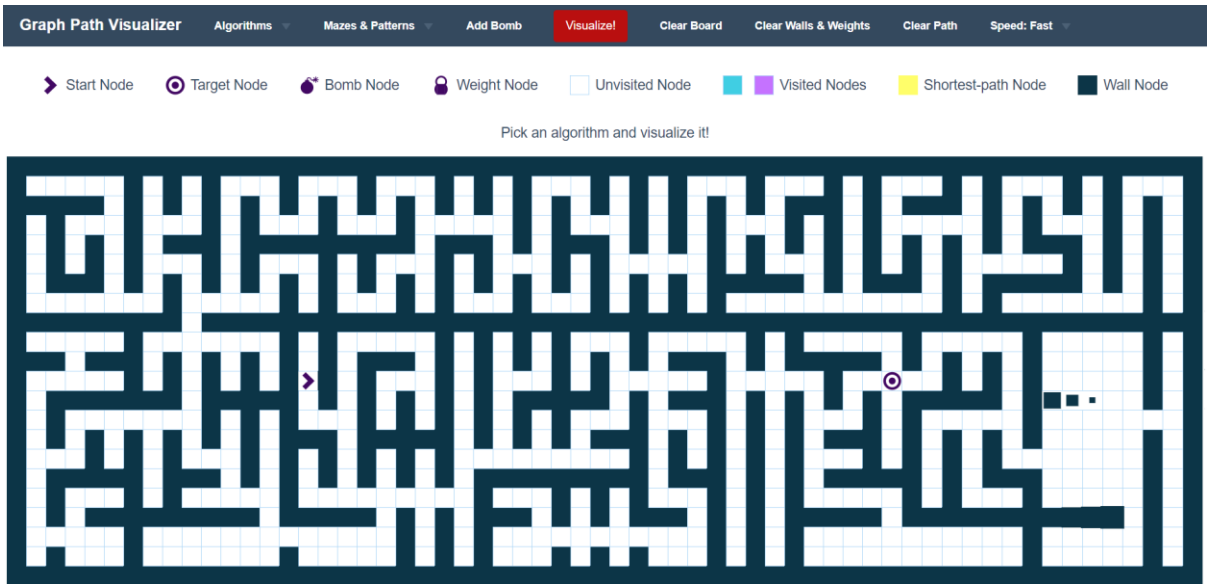

We can press “Previous”, “Next” to go back and forth can skip.

Landing Page:



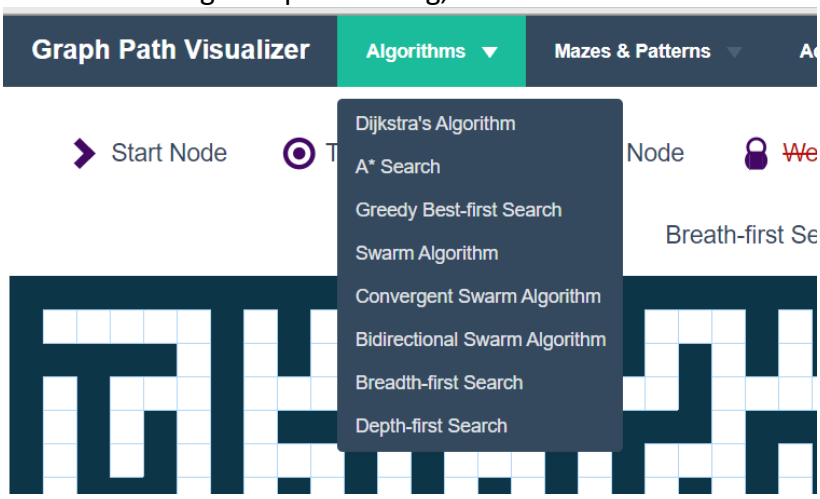

Then we select the maze & pattern (either draw free hand on the grid cells or select predefined mazes from the menu above)



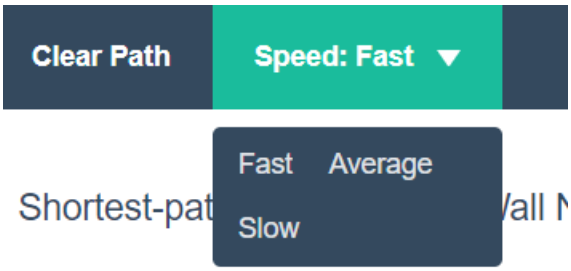



Here we select the Recursive Division Maze. The start and final nodes are already there, they can be shifted if required.

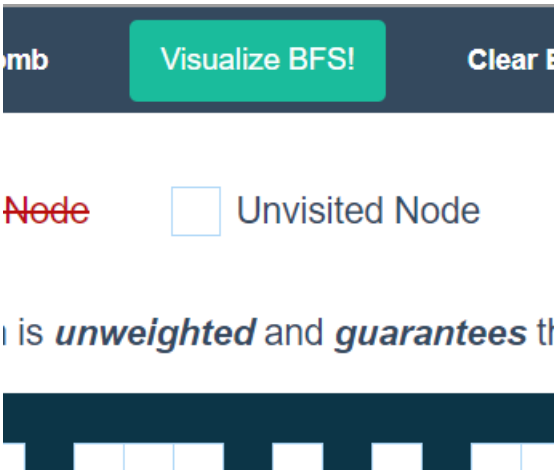
We select an algo for path finding, here BFS



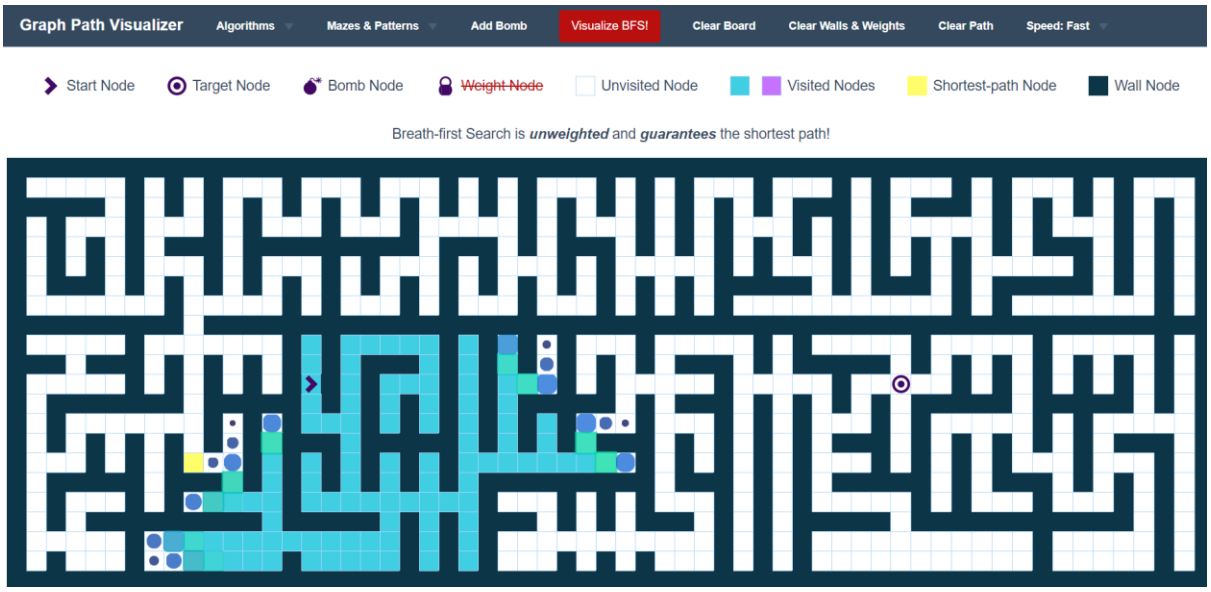

We set the speed of visualization, here fast.



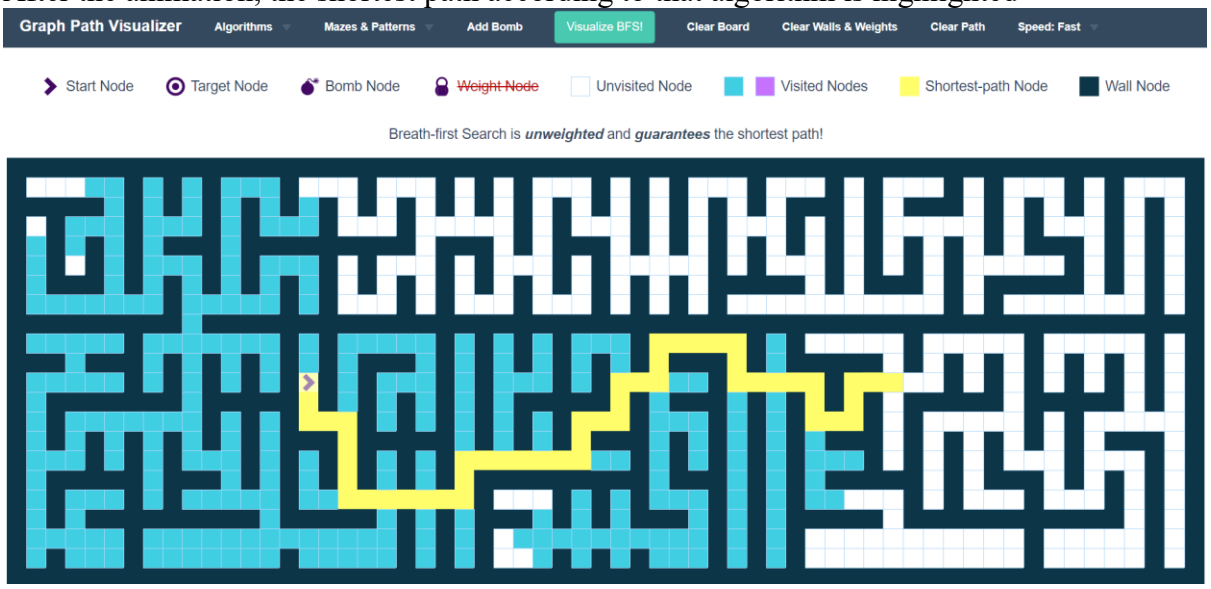

Click on the Visulaize button



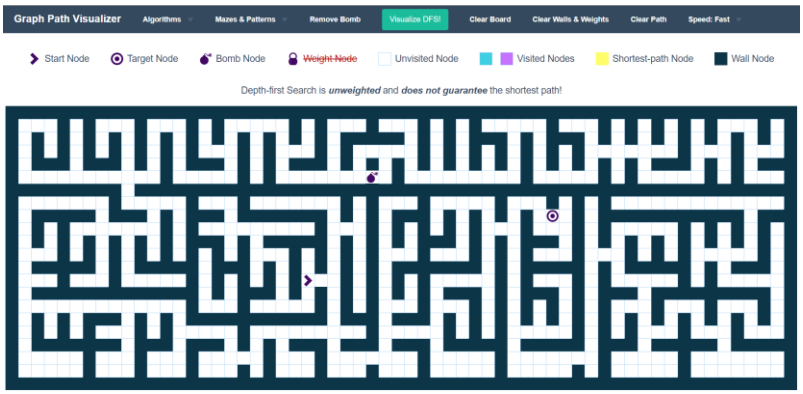

Now, we see the visualization in progress:



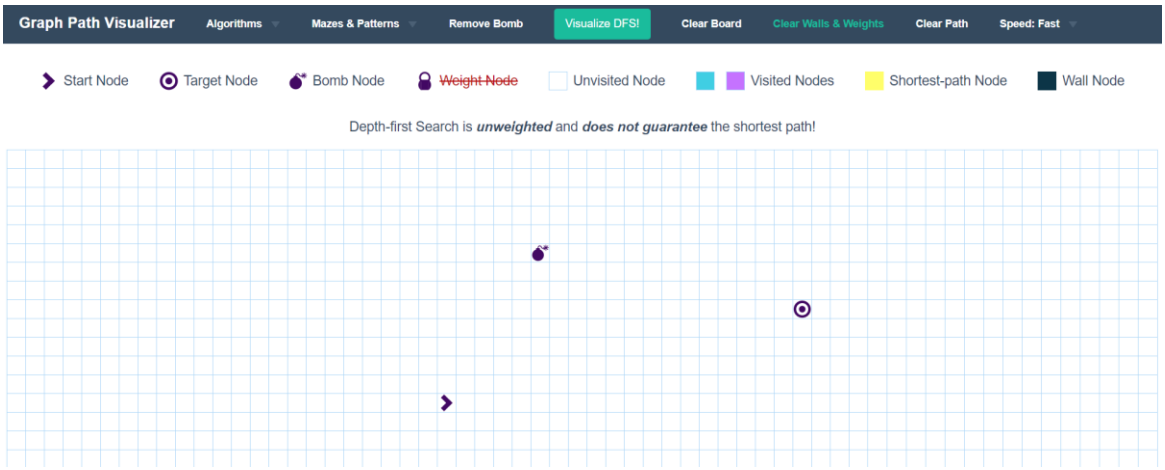

After the animation, the shortest path according to that algorithm is highlighted



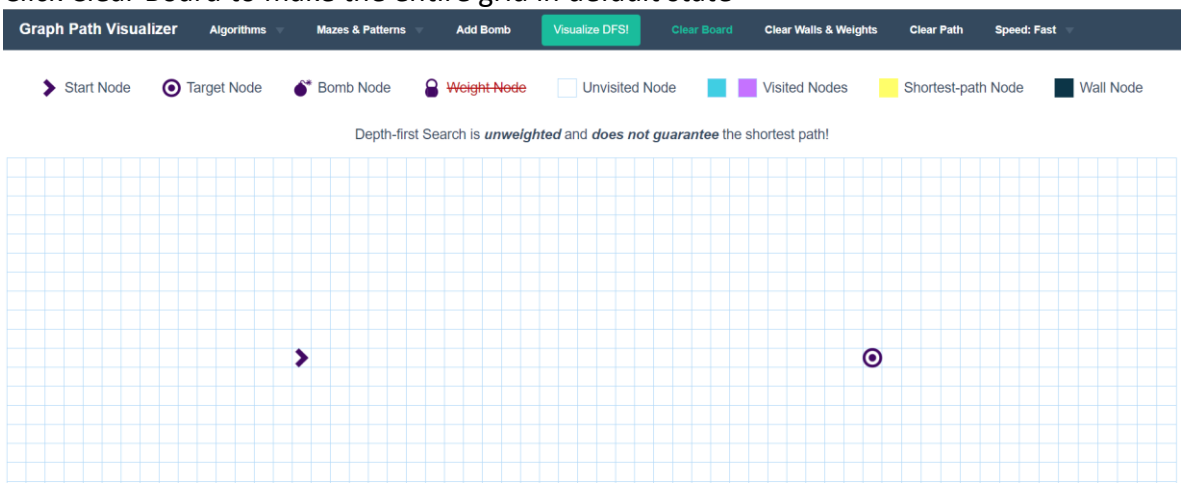

Click Clear Path to remove the visualizations



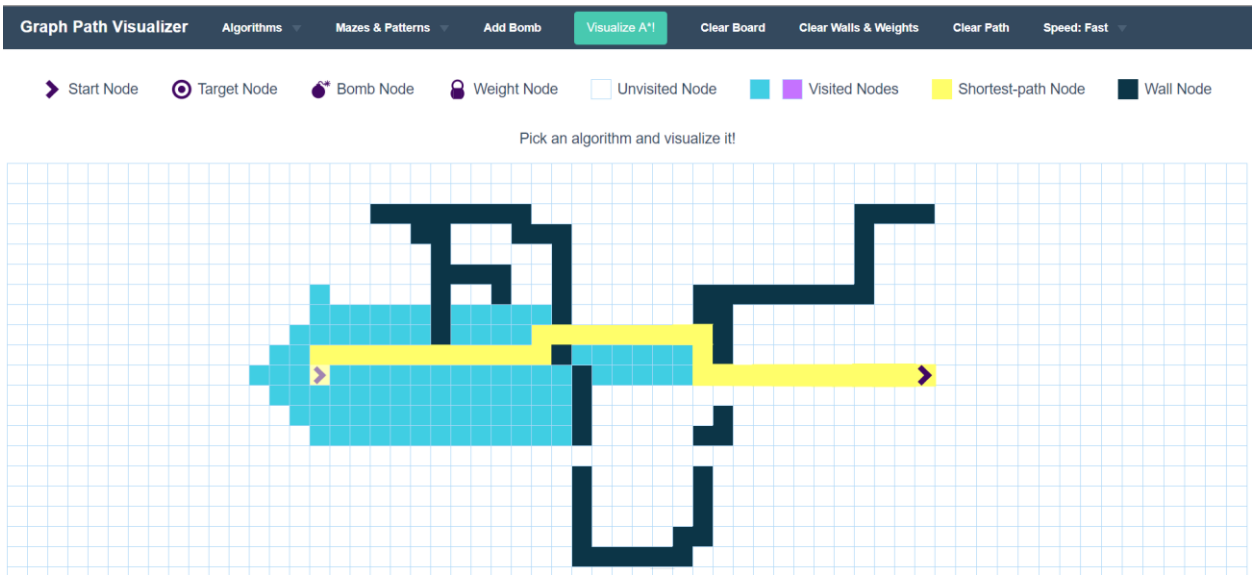

Click Clear Walls and weights to remove the walls also weights, but bombs and initial final position remains same.



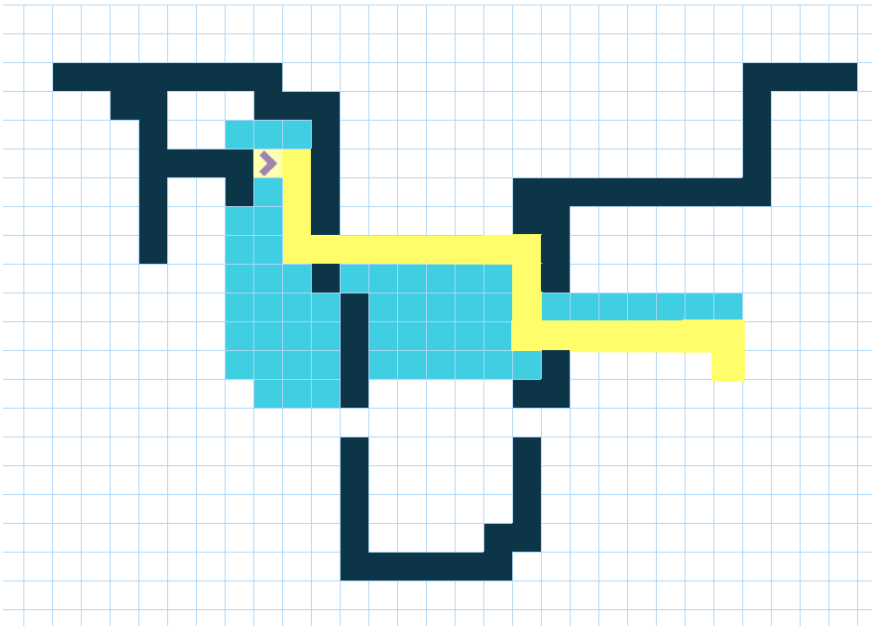

Click Clear Board to make the entire grid in default state



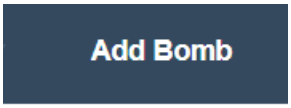

Now, suppose we make a visualization:



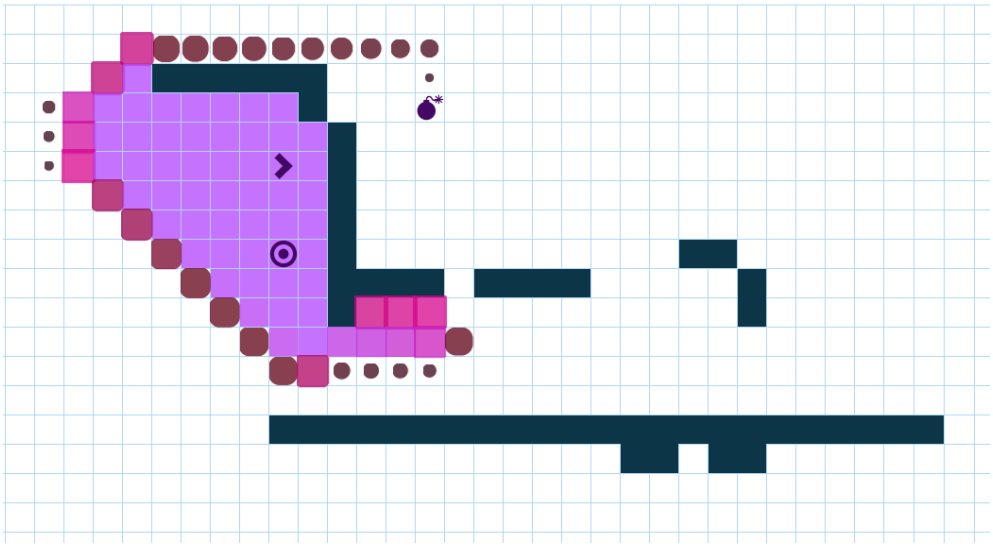

We can also shift the initial and final points by dragging; the visualization will update in real time.



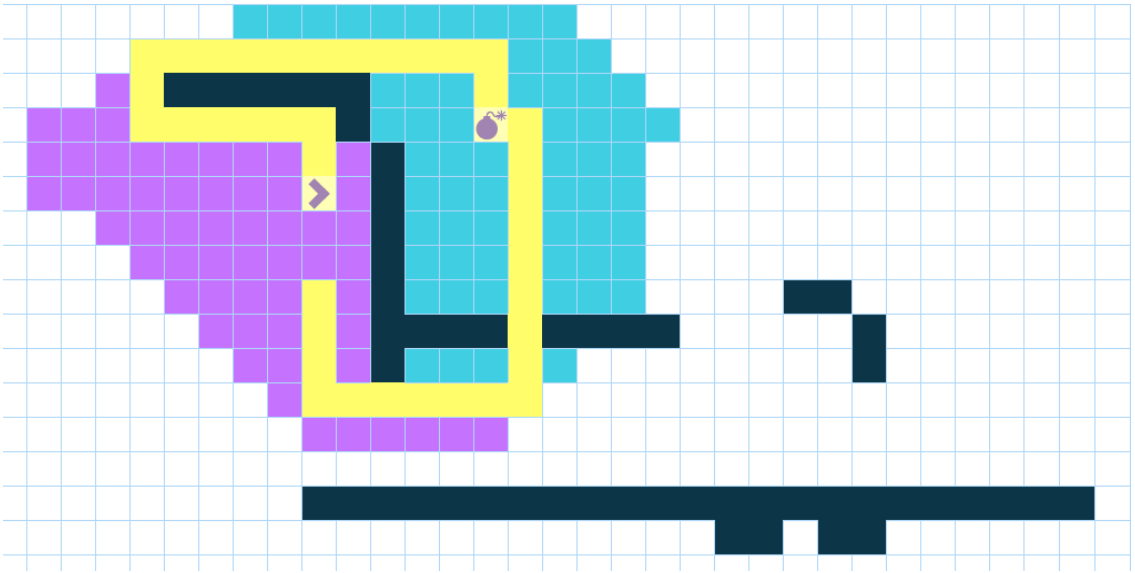
We can also add a bomb somewhere on the grid cells




Suppose we set up the walls, start, end nodes as follows and start the visualization(any algo chosen)



Now, the visualization will start from start node and reach the final node quickly, but since we have a bomb, a path will be searched where the bomb is arrived at first, and then the final cell.



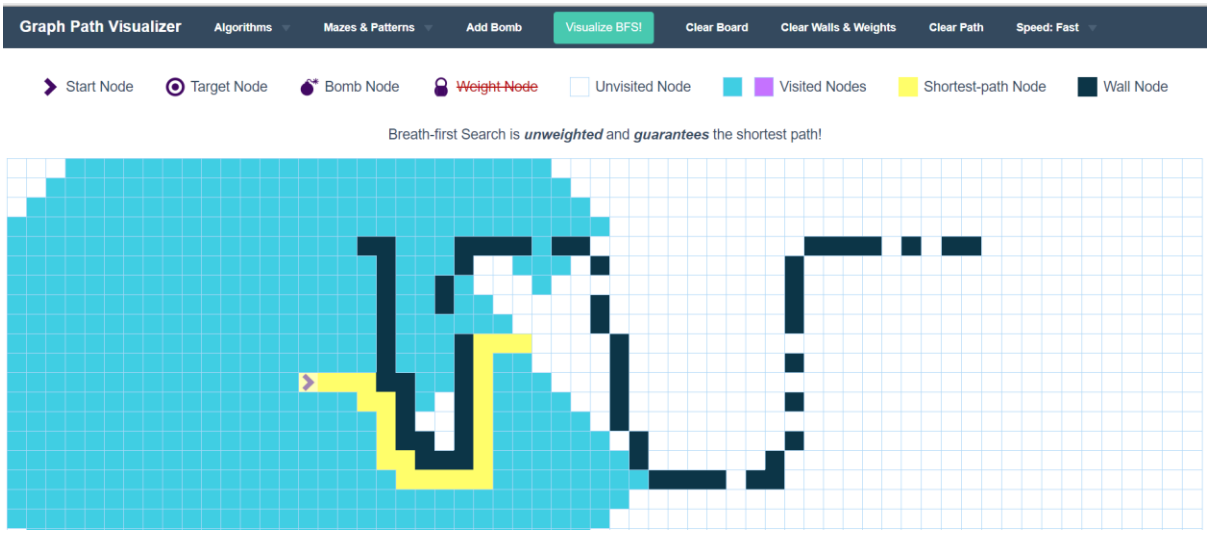
## 12. Interfaces Validation with Nielsen's 10 point heuristics

**Nielsen's ten-point heuristics**

1. **Visibility of the system**

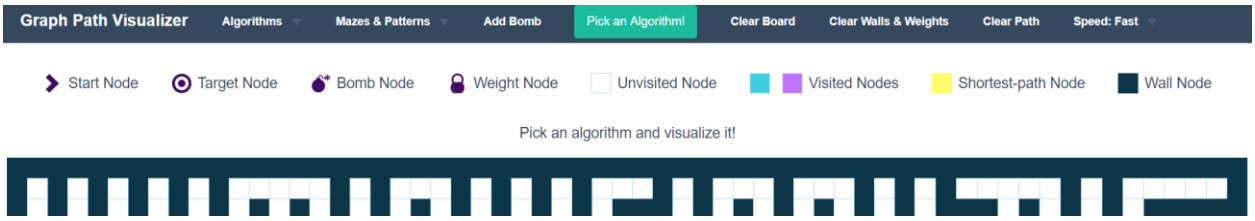
Since, our project a single webpage project, there is not much scope of the user **not** knowing what is happening, how its happening. The icons are are well defined and labelled.

The visalization proceeds at the specified speed with colours of grid-cells showing the progression, while the final path being highlighted in yellow.

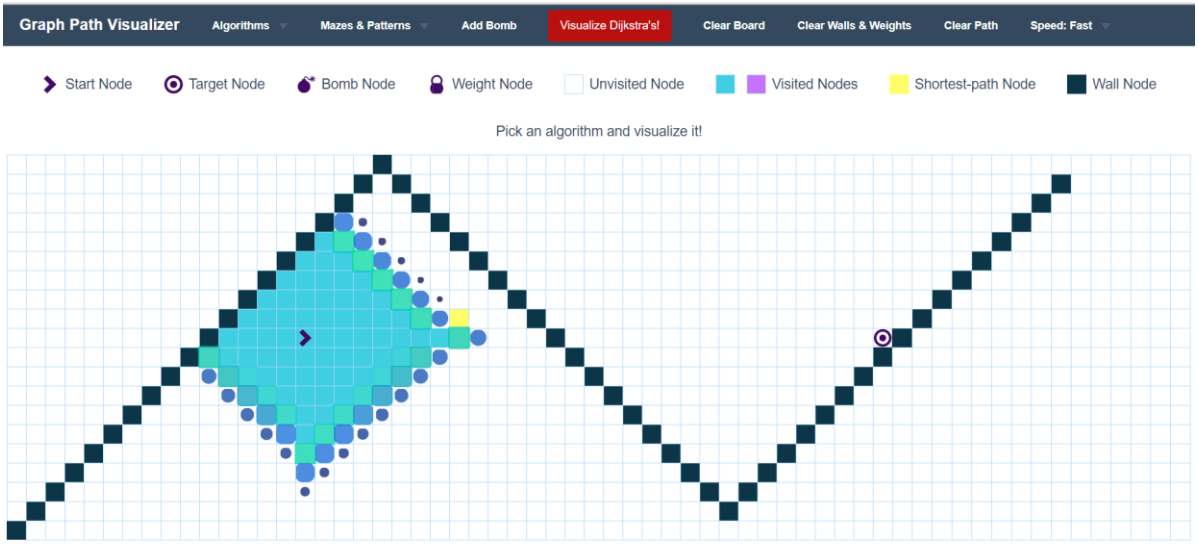



Well the state of a system is conveyed to its user always.

On this website, the interfaces informs the ‘user‘ in terms of learning & feedback which tells them that they are proceeding in the right direction.
For Ex. If the user makes/chooses the maze and clicks on the the “Visualize” button without selecting the algorithm, the visualize button will show “Pick an algorithm”

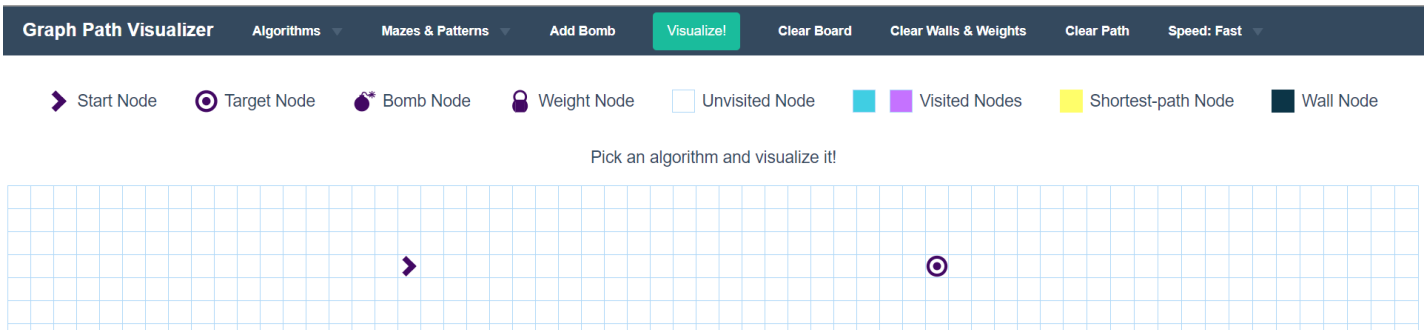



Also, the progress of the algorithm through animation step-wise gives feedback as to how the different paths are chosen.




2. **Match between system and real world**

The icons like that of the starting node, target node, bomb has been designed intuitively, so that they look like what the users expect them to look like (real worldly)
Besides, all the icons are labelled.

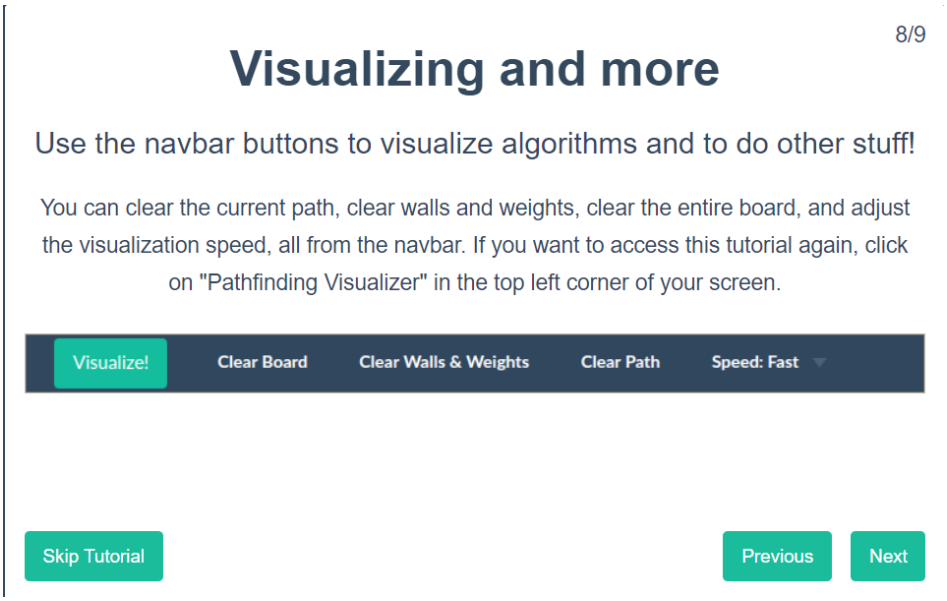



All the icons are well designed, labelled and intuitive.

For ex. The arrow symbol for the start cell feels like a “Go” symbol.

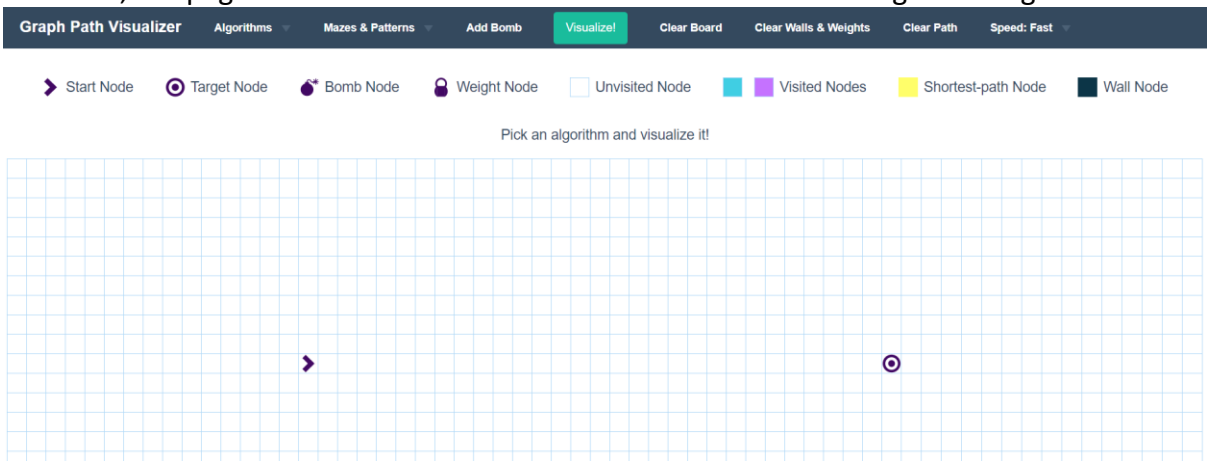
3. **User control and freedom**

We make sure the user feels in control of the website and confident in how to accomplish their tasks.
Since, the website consists of a single webpage with all the options that can be set in any order, the user seem to have full control of the operation. Task-activation is well marked when we choose the various options and then visualize.
Since, the website has single webpage, the only navigation required is in case of the tutorial which easily permits reversal of action.




### 4. Consistency and Standards

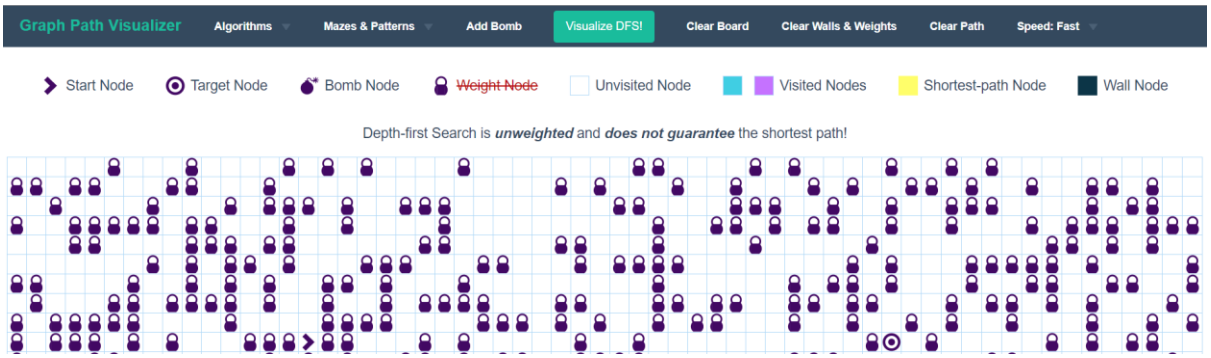
Our website Graph Path Visualizer has a single working page. So, no question of maintaining similar page structure across several pages.
However, the page maintains the usual structure which most drawing-visulizing websites carry.




For ex. The website logo is on the top-left.
All the operations that can be done are placed in menus and buttons, all placed in a single bar at the top of the page, evenly spaced out.
The main button ‘Visualiuze’, is highlighted and placed at the center
The working or the grid area, occupies the area beneath the the top bars, as we see in most other websites of this category

### 5. Error Prevention

The interface of the website has been designed in such a way that it is extremely difficult for the user to make mistakes. The menus and buttons on the website turn red or disabled when they shouldn’t be clicked.

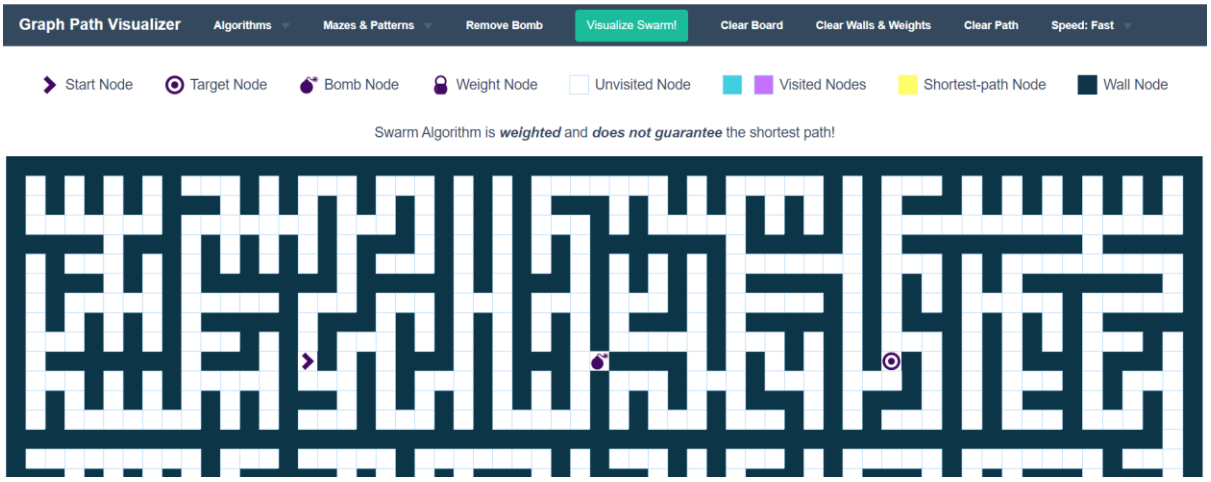



In some cases, we selected a weighted maze, and then if the algorithm chosen is unweighted in nature (like DFS has got nothing to do with weighted graph since it doesn't guarantee the shortest path to the target), then the board will automatically be cleared.

Violation
However, no dialog box telling the same is displayed after the board is automatically cleared, which needs to be done.
Also, no alert message is shown when the disabled buttons are clicked, just the colour changes.

6. **Recognition rather than recall**

the attributes required to an algo's visualization consists of just 4 factors: Maze Selection, Algo Selection, Add Bomb(if required) and animation speed.
There is no sequence of the events that needs to be memorised. Options can be selected in any order before we visualize.




The interface shows what option has been selected after we select them.
Here, we see "Swarm Algorithm is weighted and does not guarantee the shortest path!" and "Visualize Swarm!" denoting we have selected swarm algo type.

The speed shows, "Speed: Fast" showing the status.

Also, all the visualization icons & grid colours used are well suggestive and also labelled at the top the page like, start node, target, bomb node/ unvisited visited nodes, shortest path and wall node.
So, there is no need to recall anything.

### 7. Flexibility and Efficiency to use

Although, the website is made to help people learning DSA, it can be operated by people of age groups. The simplistic universal design strives to cater to as wide a range of human users of different characteristics (age, culture, educational level). However, it is to be noted that this website is of no use for the blind people. Also, the website does not have a voice-command-operation which might be helpful to people with extreme physical disability.
The website can be operated by users from novice to experienced, although people having prior knowledge of graphs will get the full understanding.
The symbols used are pretty clear to understand and are well labelled.

### 8. Aesthetic and minimalist design

The simplistic universal design strives to cater to as wide a range of human users of different characteristics (age, culture, educational level). However, it is to be noted that this website is of no use for the blind people. Also, the website does not have a voice-command-operation which might be helpful to people with extreme physical disability.
The colours used in the animation are soothing and not glaring. The symbols used are pretty clear to understand and are well labelled.

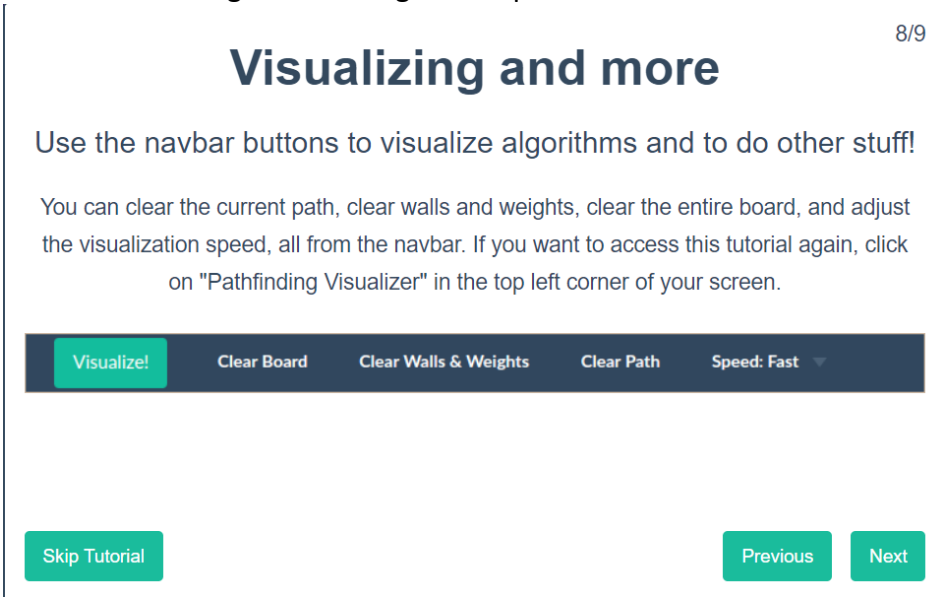
### 9. Help users recognize, diagnose and recover from errors.

The interface of the website has been designed in such a way that it is extremely difficult for the user to make mistakes. The menus and buttons on the website turn red or disabled when they shouldn't be clicked.
In some cases, we selected a weighted maze, and then if the algorithm chosen is un-weighted in nature (like DFS has got nothing to do with weighted graph since it doesn't guarantee the shortest path to the target), then the board will automatically be cleared.

Violation
However, no dialog box telling the same is displayed after the board is automatically cleared, which needs to be done.
Also, no alert message is shown when the disabled buttons are clicked, just the colour changes

### 10. Help and Documentation

The users will be guided through a complete tutorial of the website when they first log in.



## 13. Comparative Analysis with other existing technologies discussed in point 5.

- Existing models doesn't have as much variety of algorithms available as our project has (8 algos)
- Unlike other visualizers, the project also includes comprehensive tutorials explaining all the features.
- Existing models doesn't incorporate all HCI principles. We have taken great care to ensure that the UI is appealing and easy to use
- Unlike existing visualizers, the initial and final points can be shifted even after the visualization, with the shortest path being updated in real-time.
- Existing path visualizers don't have the variety of predefined mazes and patterns like ours

## 14. Conclusion & Future Scope

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.
Hence we developed this application with regard to fulfill the needs of slow learners by making this more exciting and exhilarating.

Future Scope for this project is immense:

-Project could be made more comprehensive to understand the algorithms, if 2 visualizations with different algorithms can be viewed side-by-side.
-More algorithms (ex. Greedy, mapping )could be added to the website
- The possibility of numbering the cells as the visualizations progresses, along the different paths can be explored
-A feature where users upload their own algorithms to visualize them can be added
-This website which visualizes the algorithms (to find the shortest path) could be modified

- In Biology by finding the network model in spreading of infectious diseases.
- In finding the optimal agenda of flights.
- In path planning for mobile robot transportation…

-

## 15. References

[1] Ardito C., De Marsico, M., Lanzilotti, R., Levialdi, S., Roselli, T., Rossano,V. & Tersigni, M. Usability of E-Learning Tools
[2] Akoumianakis D. (2011). Learning as 'Knowing': Towards Retaining and Visualizing Use in Virtual Settings. Educational Technology & Society, 14 (3), 55-68.
[3] Ozyurt O., Ozyurt H., Baki A., Guven B. & Karal H. (2012). Evaluation of an adaptive and intelligent educational hypermedia for enhanced individual learning of mathematics: A qualitative study. Expert Systems with Applications, 39(15), 12092-12104.
[4] Nguyen V.A. & Yamamoto A. (2012). Learning from graph data by putting graphs on the lattice. Expert Systems with Applications, 39(12), 11172-11182.
[5] Hundhausen C. D., Douglas S. A. & Stasko J. T. (2002). A metastudy of algorithm visualization effectiveness. Journal of Visual Languages and Computing, 13(3), 259-290.
[6] Gillet D., Law E.L.C. & Chatterjee A. (2010). Personal Learning Environments in a Global Higher. Engineering Education Web 2.0 Realm. In Proc. of IEEE EDUCON 2010 Conference. pp. 897- 906.
[7] Fouh E., Akbar M. & Shaffer C. A. (2012). The Role of Visualization in Computer Science Education. Computers in the Schools, 29(1-2), 95-117.

## 16. Appendix:

a. https://drive.google.com/drive/folders/1CCdLR-NNhMefqBAWt5UJW3C9unzZ3pHJ?usp=sharing

b. https://drive.google.com/file/d/1zVkJ-1Y5QDNclcMBvHWQBDDjHxlbrFzR/view?usp=drivesdk

c. https://drive.google.com/drive/folders/1THfK7mx4WhGwJi3bqZR_I2E8-sVhLFie?usp=sharing