Project Title: AI Search Visualiser

Technical guide

Name	Student ID
James Farrelly	17396736
Emily Whyte	17405094

Project Supervisor	Charlie Daly

Date of document completion	07/05/2021

Abstract

This project, A.I search visualiser, is a web application developed to visualise popular search algorithms used within Artificial Intelligence.

Nodes and edges are drawn onto a canvas and stored together as elements. The next step is to select a start and end goal for the algorithm. The user can then select one of four currently available algorithms to visualise. There are playback controls which allow the user to step through the visualisation in whichever way they please. The visualisation consists of nodes changing colours to distinguish between what state they currently are in throughout the algorithm. Users may also wish to enable comparison between algorithms to understand how they differ from each other by visualising both simultaneously. There is also an option to save the current arrangement of elements in the graph, to then upload that save at a later time and reuse the tree. To increase customisation, users have the ability to edit the heuristic values and names of nodes and edges.

One goal was to make the web application as simple as possible to use. To achieve this the user interface is, with every option layed out in front of the user rather than being hidden in menus and other pages. If at any point the user is confused there is a helpful tutorial which will assist them in using the tool. This contributes to achieving the main goal of the project, assisting those studying A.I search algorithms by using custom visualisation and interaction which are tools often used to understand and learn complicated ideas.

Table of contents

1. Introduction

1.1. Motivation

The main source of motivation for this project comes from the CA318 module that we studied in 3rd year; Advanced Algorithms and A.I Search. This module was taught by our supervisor, Charlie Daly. Among other things, this module covered A.I search algorithms, such as Breadth-First Search, Depth-First Search and so on. We had various lab tasks and exam questions which required us to have a good understanding of the process of completing a search using one of these algorithms. Watching worked through examples of a search conducted on a graph felt like the most effective way for us to understand and remember the details of each different algorithm.

The aim of this web application was to provide students and programmers in general with an interactive tool to assist their learning of A.I search algorithms. Having studied these algorithms, we know first-hand how useful this tool can be in helping students grasp the fundamentals of A.I search.

The project was implemented in JavaScript, HTML, CSS. We learned to use HTML and CSS in previous years which allowed us to incorporate experience from previous web development projects into our final year project, combining it with JavaScript for increased functionality and new learning challenges. In particular, we used the React JavaScript library to create our interactive user interface. React is popular and flexible. JavaScript projects using React can also be tested using the Jest testing framework. For these reasons, we believed that using this combination of programming languages was the best approach to completing the project.

1.2. Glossary

Term	Definition
A.I	Abbreviation for Artificial Intelligence. Machines replicate the behaviour of the human brain.
A.I Search Algorithm	The process of getting from a start state to a goal state.
Current node	The node that is currently being checked by the algorithm to see if it matches the goal node.
Edge	Connects two nodes in a tree.
Heuristic	A score used to rank options in a search, using information that is currently available.
JSON file	A popular file type consisting of data in a simple structure which is readable by humans.
Node	Part of a tree. A node represents a value or condition.
To-do node	A node that is on a list, waiting to be checked against the goal at a later stage in the search, if the goal is not found before it is reached.
Search tree	A collection of nodes and edges. One node is the root of the tree. Nodes have at most one parent and can have any amount of children. A tree in which nodes have at most 2 children is called a binary search tree.

Visited node	A node that has already been checked to see if it matches the goal node at a previous stage in the search.
Web application	Software that is run on a server, unlike software that you would run locally, on your own machine. It is accessed through an active network connection using a web browser.
Object	A datatype used for storing unordered data in a key-value pair.
Element	Either a node or an edge

1.3. Research:

Using visualisation techniques in order to simplify teaching complex ideas is not a new idea and has been effectively used for over a thousand years. With this in mind it became about what was the best way to successfully communicate the concepts of each algorithm, without being too confusing as to create a boundary to those who were new to learning about them.

Many different types of algorithm visualization exist. There are those that visualize it using pathfinding i.e. navigating a maze, there are also those that sort graphs for sorting algorithms, and others that create unique visualizations such as the ones created by the javascript library D3.js. We decided as the aim of this project was the education, we would focus on using trees as they are what were used when we first started learning about the basics of searching algorithms as well as what most learning tutorials use online. In this sense it would be extremely helpful as students could follow their instructor along and eventually experiment with the tool themselves.

As we are not the first to create a web application focusing on algorithm visualisation we had the opportunity to visit these sites and draw from it elements that we thought suited our aims, and leave others that we didn't. For example popular site visualgo offers visualisations for a number of algorithms, including some A.I search algorithms. However, due to the lack of a focus on a particular type of algorithm, the web application can be confusing to navigate and understand so we decided it would be best if our web app was focused and easy to navigate, however we liked how you could draw your own tree. We worked with our supervisor, who provided critical feedback to our initial design of the visualisation and throughout its development.

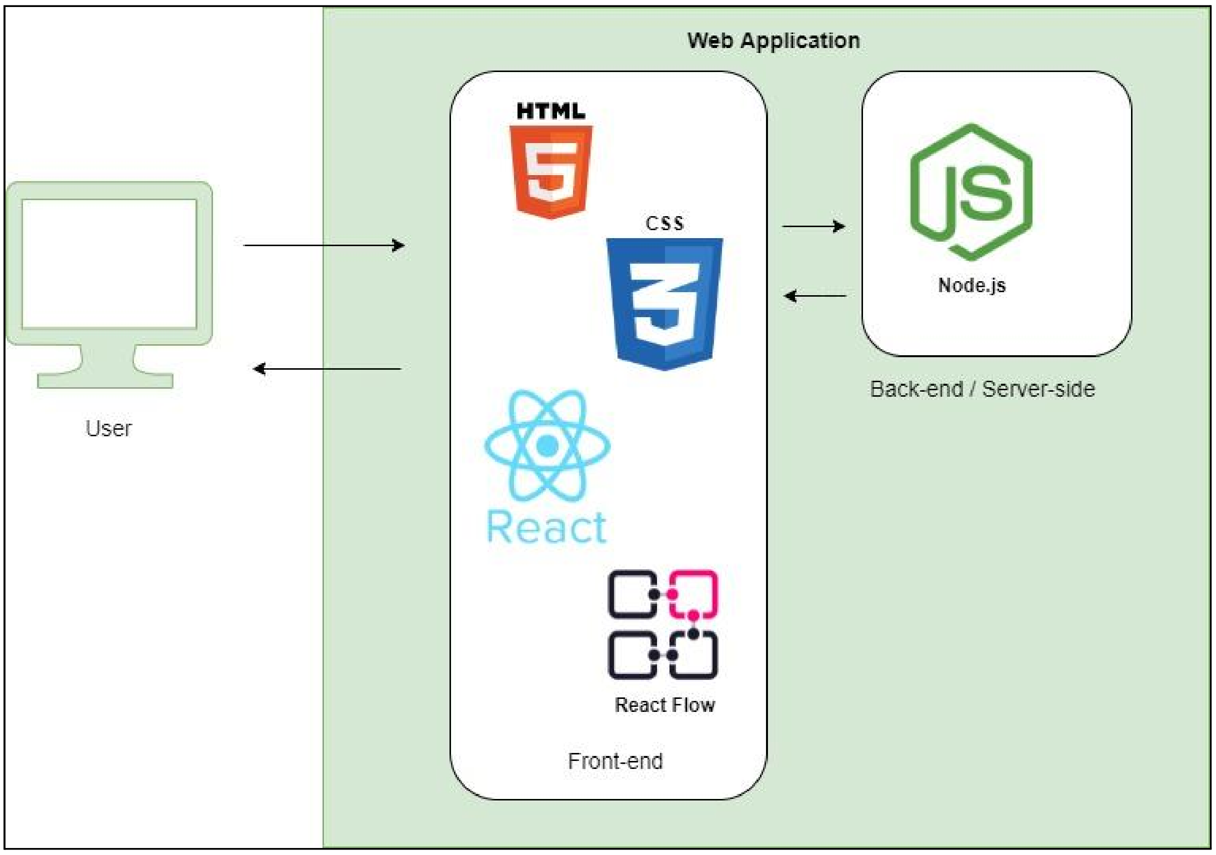
In order to implement the various algorithms in our system we first had to research and understand their workings. For this we used our knowledge gained from the Advanced Algorithms and A.I Search module we completed in third year of university. We also used a number of written and youtube tutorials. This really helped our understanding of each algorithm making it possible to implement them into our visualisation system as well as giving us insight into how people teach AI search algorithms and how they are learned.

To create an effective, useful visualisation tool we looked at methods used by various lecturers and blogs to explain A.I search algorithm examples. We also reviewed the notes we received in the Advanced Algorithms and A.I Search third year module.

2. Design

2.1. System Architecture

2.1.1. System Architecture Diagram



2.1.2. Description

AI-Search Visualiser was written primarily in Javascript ES6. The following libraries/dependencies visible in the diagram above were used to create the web application.

React - A popular front-end javascript library we used to develop most of the user interface for our web application.

Node.js - Another popular javascript library but unlike React, Node.js runs in the backend.

React Flow - An open source React library still in its infancy. Helped us make our application using nodes.

CSS - Cascading Style Sheets used for styling of react components and HTML

HTML - HyperText Markup Language used for the index

Additional libraries/dependencies used not mentioned in diagram:

Material UI - A React UI framework developed by Google. It was used to develop the front-end of our application.

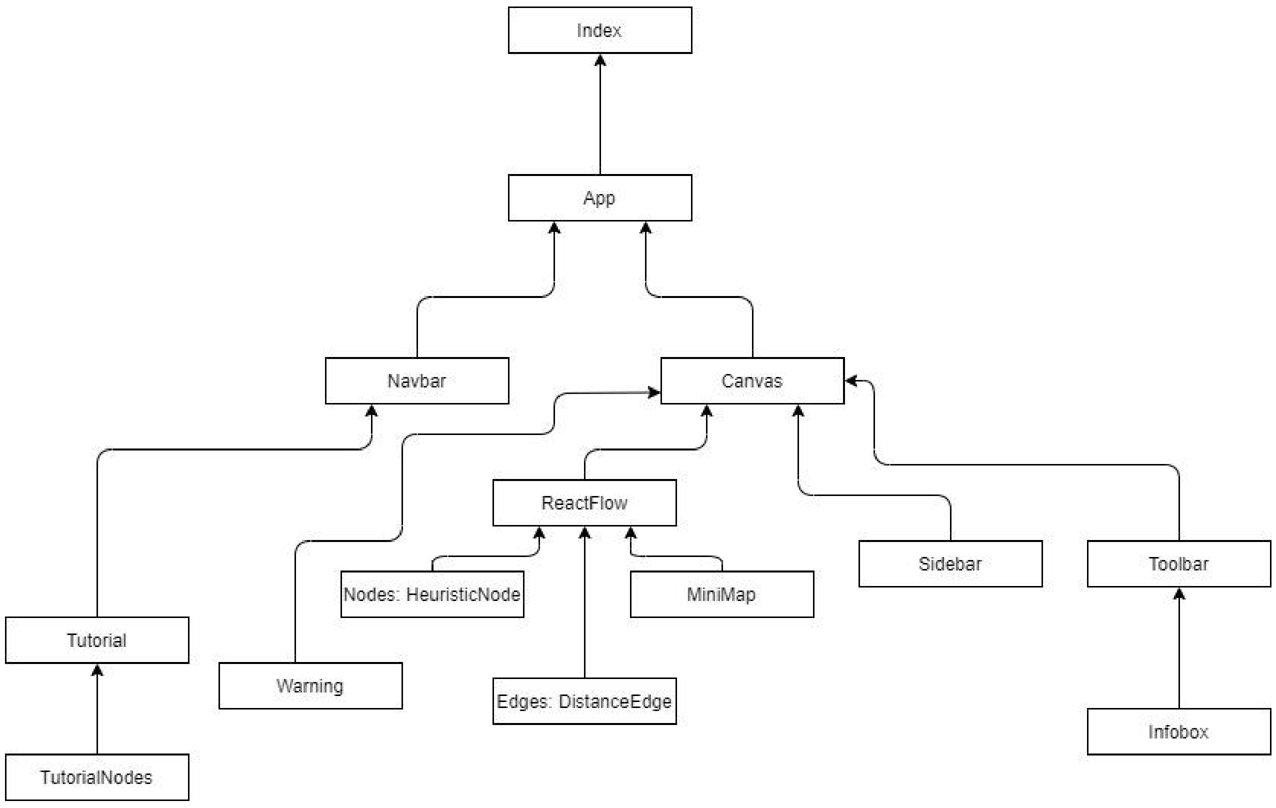
Jest - A javascript testing library we used for unit testing.

Lodash - Javascript utility library used for comparing objects.

Styled Components - Javascript library used to style in components in javascript while still using the CSS format.

Dagre - Javascript library used for auto layout

2.1.3. Component Hierarchy



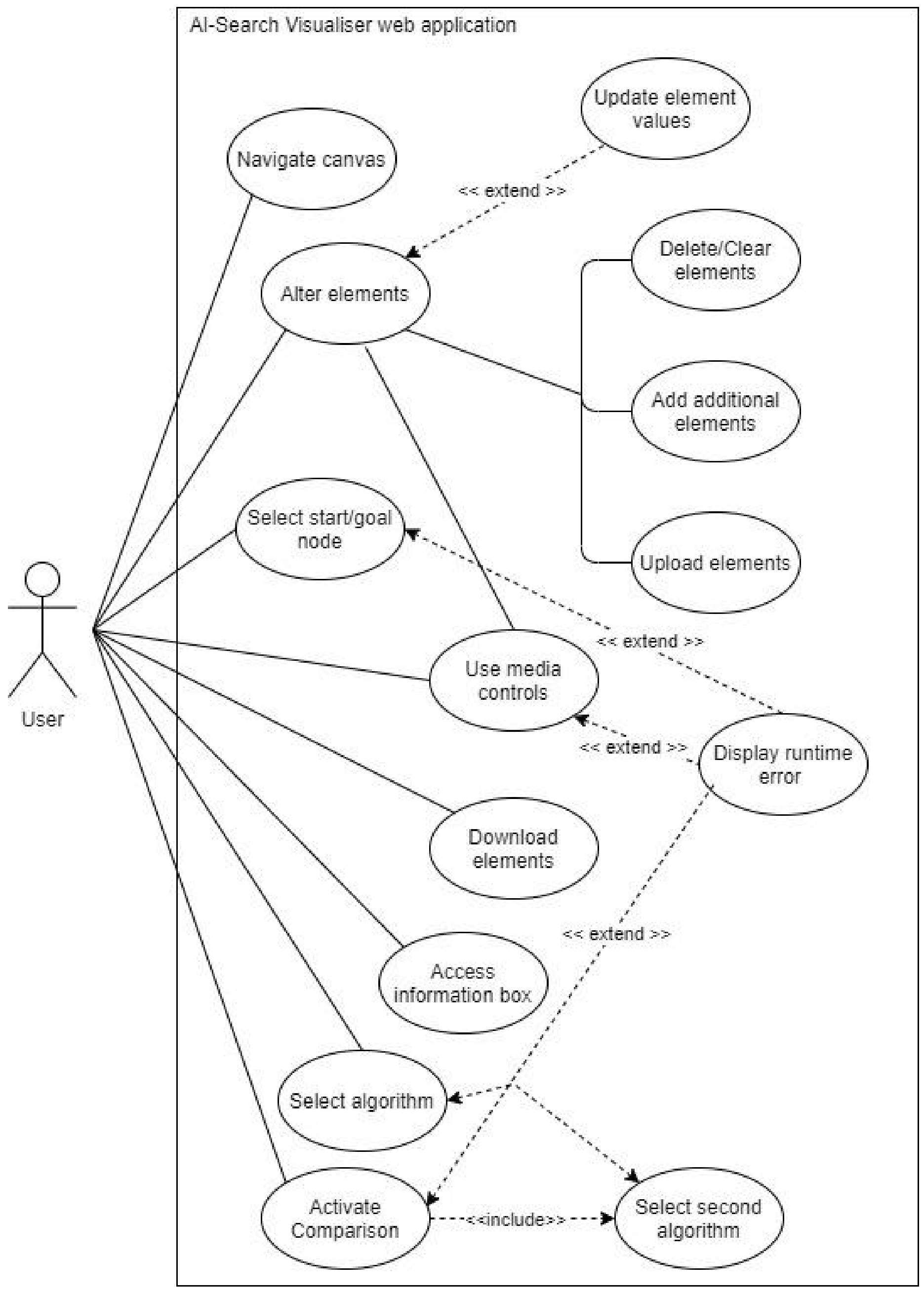
2.1.4. Component Hierarchy Description

The component hierarchy tree shows the relationships between each component. Each parent is a container for it's child node. At the top is the index which renders its child component, which is App. App renders the Navbar and the Canvas. This continues to the last child component to create a visualisation of how the UI for the system is generated.

3. High Level Design

3.1. Use cases

3.1.1. Use Case Diagram

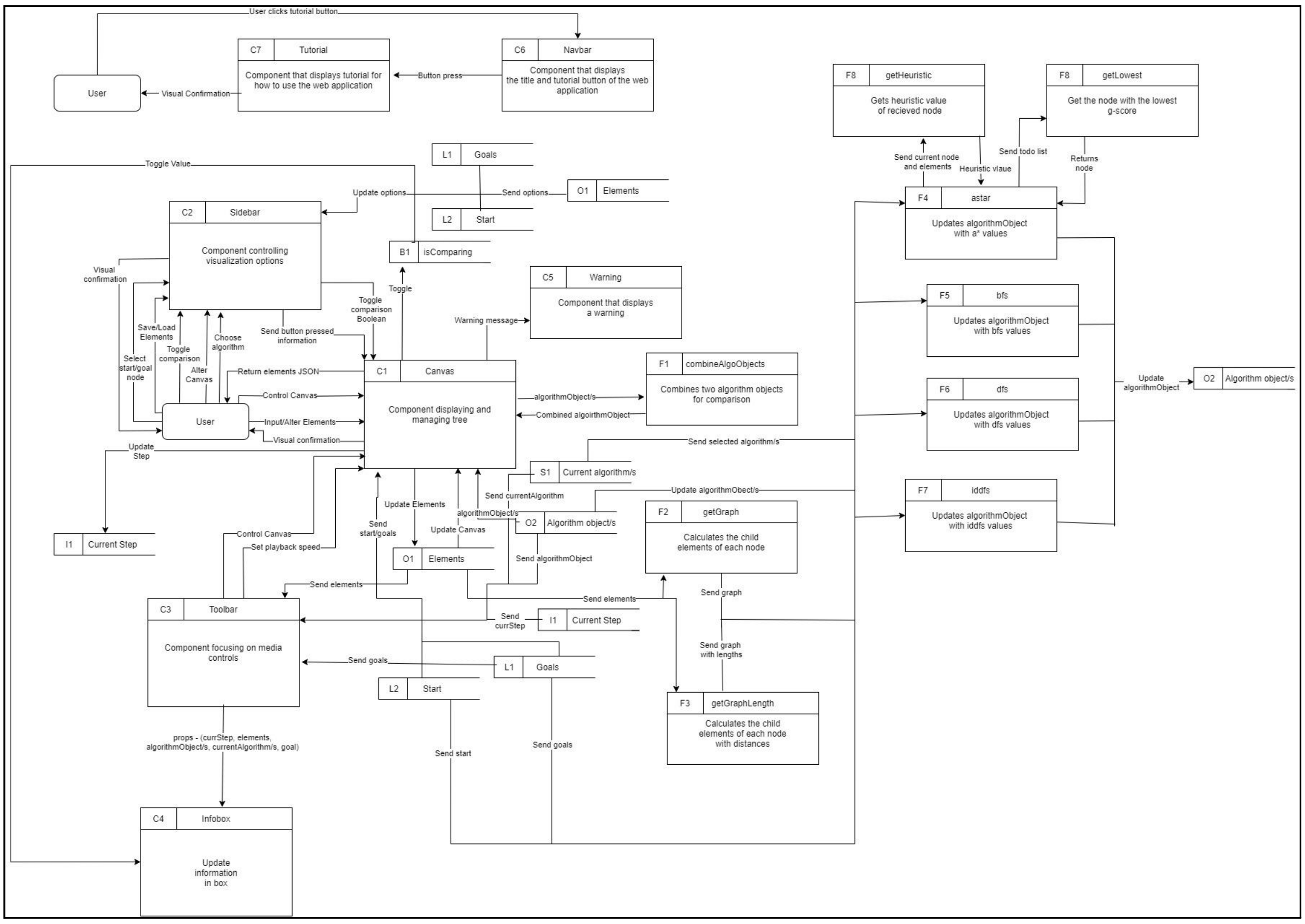


3.1.2 Use Case Description

The Use case diagram gives a visual representation of the specific interactions an actor, in this case, a user has with the system. The user can navigate the canvas by moving the view with their mouse. They can add or remove whatever nodes they wish and upload a JSON file to replace the set elements with those in the JSON. This use case is extended by the user's ability to alter the h(n) in a node or the distance value in an edge. The user can select a specific start and goal node out of those nodes available. They can also use one of many playback controls at their disposal to traverse through the tree. This traversal will alter the style of the elements in the tree, changing their colour to notify the user of the change depending on what algorithm is previously chosen. As the user can upload JSON files, they can also download their current tree as a JSON file to reupload later. There is an information box the user can open to provide additional information about their current step. The user can select an algorithm, if they activate a comparison they must choose a second algorithm before playback commences. If the user attempts to use the playback controls without an algorithm, start or end goal selected, they will be given a warning message which is shown by the extension.

3.2. Data Flow

3.2.1 Data Flow Diagram



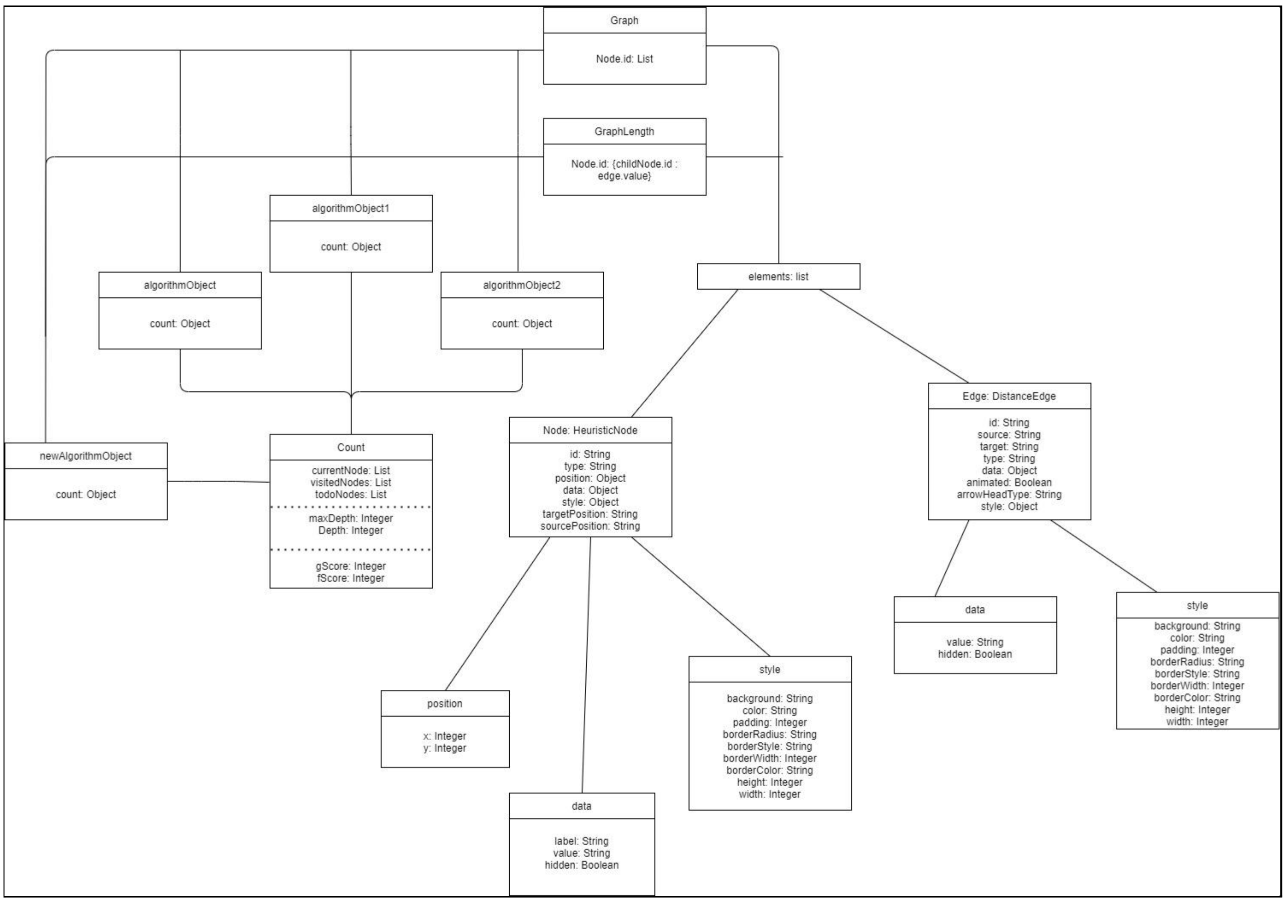
3.2.2 Data Flow Diagram Description

The following data flow diagram describes the flow of data between components and functions in the system. The diagram begins at the external entity “User” which is interacting with the system causing data to flow in the following path. Elements in this diagram have been given a key followed by a number which define what they are in the system as some elements are re-used.

Key	Type
C	Component
F	Function
O	Object
B	Boolean
I	Integer
L	List
S	String

3.3 Objects

3.3.1 Object Diagram



3.3.2 Object Diagram Description

The object diagram above denotes each of the objects in our system and their interactions. As React gradually tries to adopt the use of hooks instead of classes, we decided our implementation would take advantage of these tools. There are therefore no classes in our system which is why this object diagram appears without a class diagram. The diagram displays the key for each value in the object as well as the type of their values. If the values type is another object, it is presented as such. For example the HeuristicNode type has a child object of data which stores it's h(n) value used for A* search.

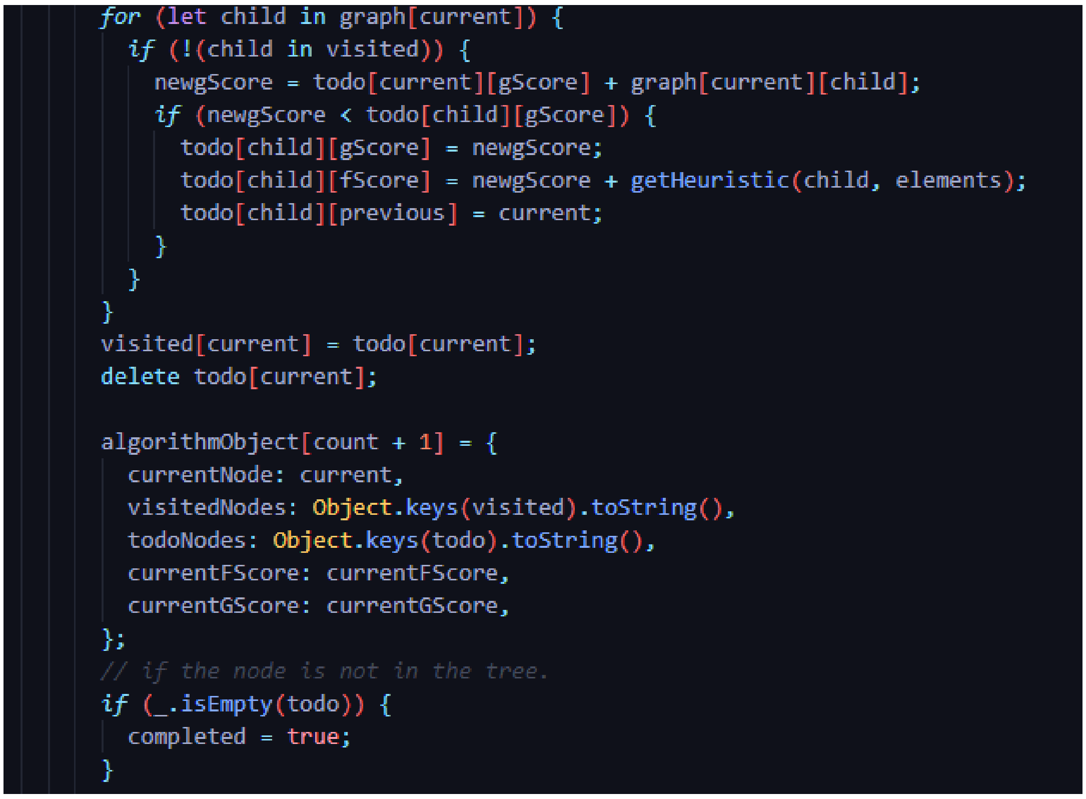
4. Implementation

This section explains how certain systems were implemented and includes code snippets to give a peek into the overall implementation. In order to initialize our react app we used the npm create-react-app. We used `npm run build` to deploy our web app.

4.1 Algorithms

When we began development of our project one of our primary goals for our implementation was to make integration of new algorithms into the visualisation component of our application as fluid as possible. In order to do this we created the algorithm functions with a generalisation in mind so they would all output the same object which is indexed by the current step within the visualisation. The object takes some additional information based on the algorithm but the main keys passed to the visualisation component remain the same.

The following is a snippet from our implementation of a* algorithm showing how at each step the current, visited and to-do nodes are saved along with the additional f and g score at the current node. This object is returned to the Canvas component and then passed to various other components such as Toolbar to update the information box.



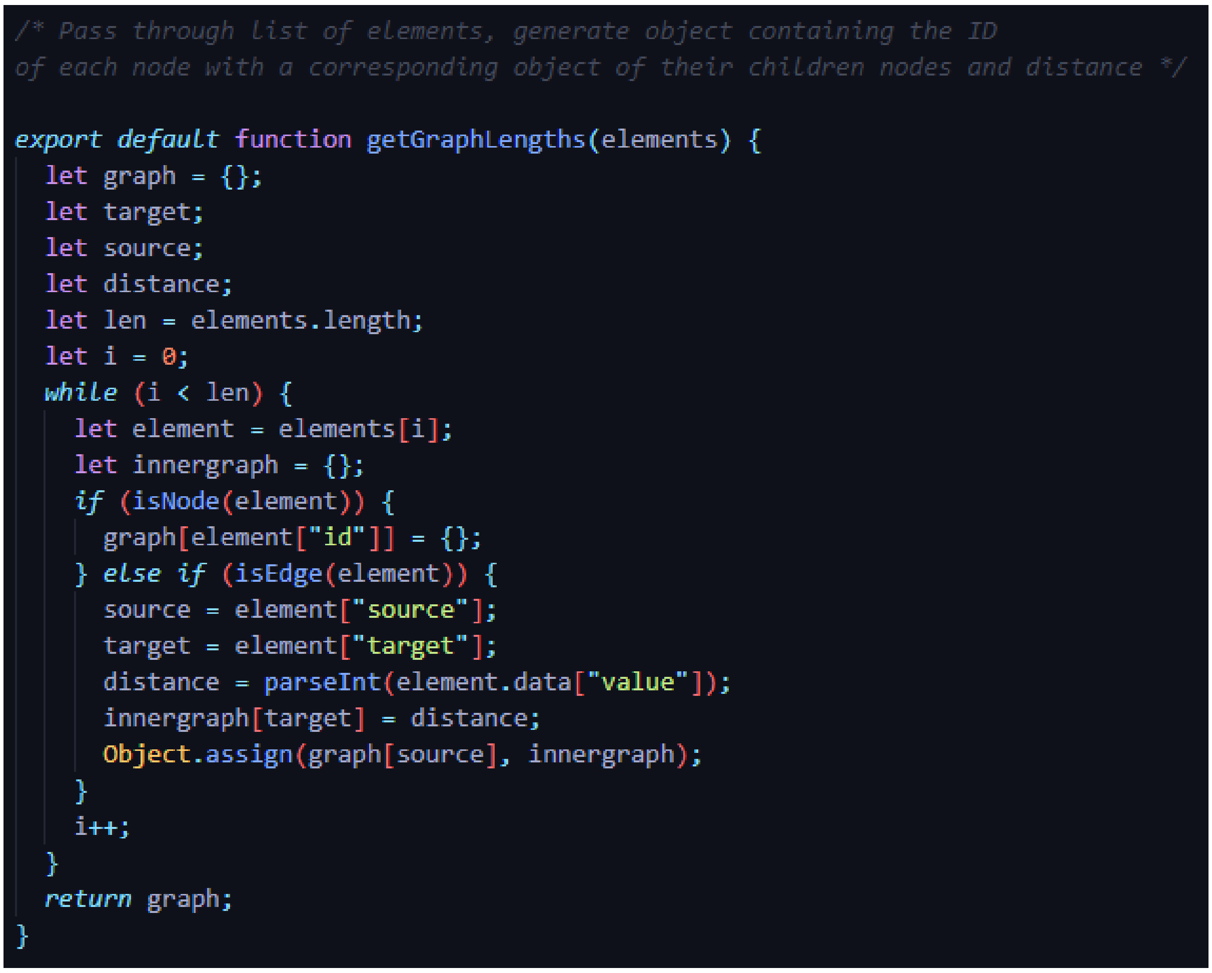
```
for (let child in graph[current]) {
  if (!(child in visited)) {
    newgScore = todo[current][gScore] + graph[current][child];
    if (newgScore < todo[child][gScore]) {
      todo[child][gScore] = newgScore;
      todo[child][fScore] = newgScore + getHeuristic(child, elements);
      todo[child][previous] = current;
    }
  }
}
visited[current] = todo[current];
delete todo[current];

algorithmObject[count + 1] = {
  currentNode: current,
  visitedNodes: Object.keys(visited).toString(),
  todoNodes: Object.keys(todo).toString(),
  currentFScore: currentFScore,
  currentGScore: currentGScore,
};
// if the node is not in the tree.
if (_.isEmpty(todo)) {
  completed = true;
}
```

Astar function snippet

All of our algorithm implementations made use of a graph. Our implementation supports the creation of nodes and edges on a canvas, which are all stored together in an elements array. We needed a way to quickly access and find out the parent child relationship between nodes for our implementation of every algorithm. For every algorithm other than A* we specifically only needed a graph of the child. In order to do this we created a graph array in which the key (the parent node) pointed to a value which was a list of child nodes. We saved this data using the elements id, which is unique to that element otherwise it would have been overwritten.

For A* search we needed the additional information of how far away the parent node is from the child. In order to implement this we changed the graph variable to an object which had an inner graph pointing to the children which were the key and their distance from the parent which was the value.

Both implementations took full advantage of the isNode & isElement functions provided by the React-Flow library to find what in the elements array was a node and what was an edge.



```
/* Pass through list of elements, generate object containing the ID
of each node with a corresponding object of their children nodes and distance */

export default function getGraphLengths(elements) {
  let graph = {};
  let target;
  let source;
  let distance;
  let len = elements.length;
  let i = 0;
  while (i < len) {
    let element = elements[i];
    let innergraph = {};
    if (isNode(element)) {
      graph[element["id"]] = {};
    } else if (isEdge(element)) {
      source = element["source"];
      target = element["target"];
      distance = parseInt(element.data["value"]);
      innergraph[target] = distance;
      Object.assign(graph[source], innergraph);
    }
    i++;
  }
  return graph;
}
```

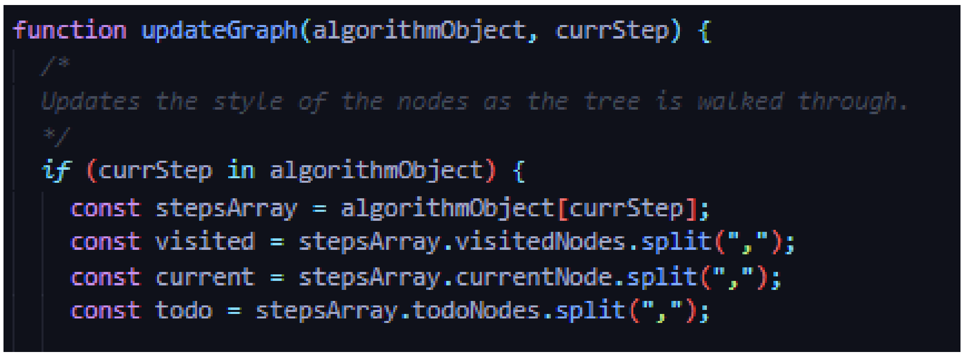
getGraphLengths function snippet

4.2 Visualisation

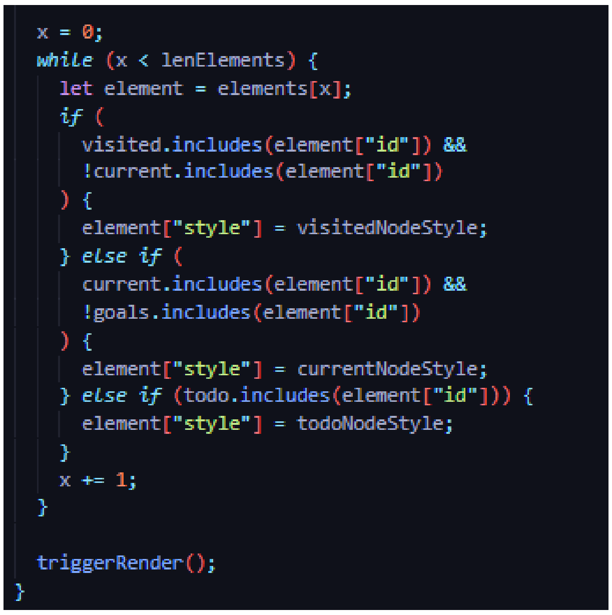
The visualisation side of the project was the part that was worked one for most of the development process. We decided in order to display nodes and edges on a Canvas we would make use of the

React Flow library which still gave us the ability to heavily customize the nodes, canvas and edges. The library enabled us to allow the users to use a number of controls such as zooming in and out, fitting graphs to screen, the minimap and element creation and deletion by including helpful functions.

We then expanded upon the library by adding our own features for visualisation like more playback options, which make use of the algorithm object mentioned in the previous section. The function snippet below shows how the tree is updated based on the current step in the playback. At each stage the algorithm object is being looked into to see the current, visited, to-do and goal lists which are compared to the current element. If the element id matches an id in the algorithm object, it will be subsequently updated to the designated style. This function is reused often for example when the play button is clicked the steps will increment and graph will update until the goal has been reached, or it is paused.



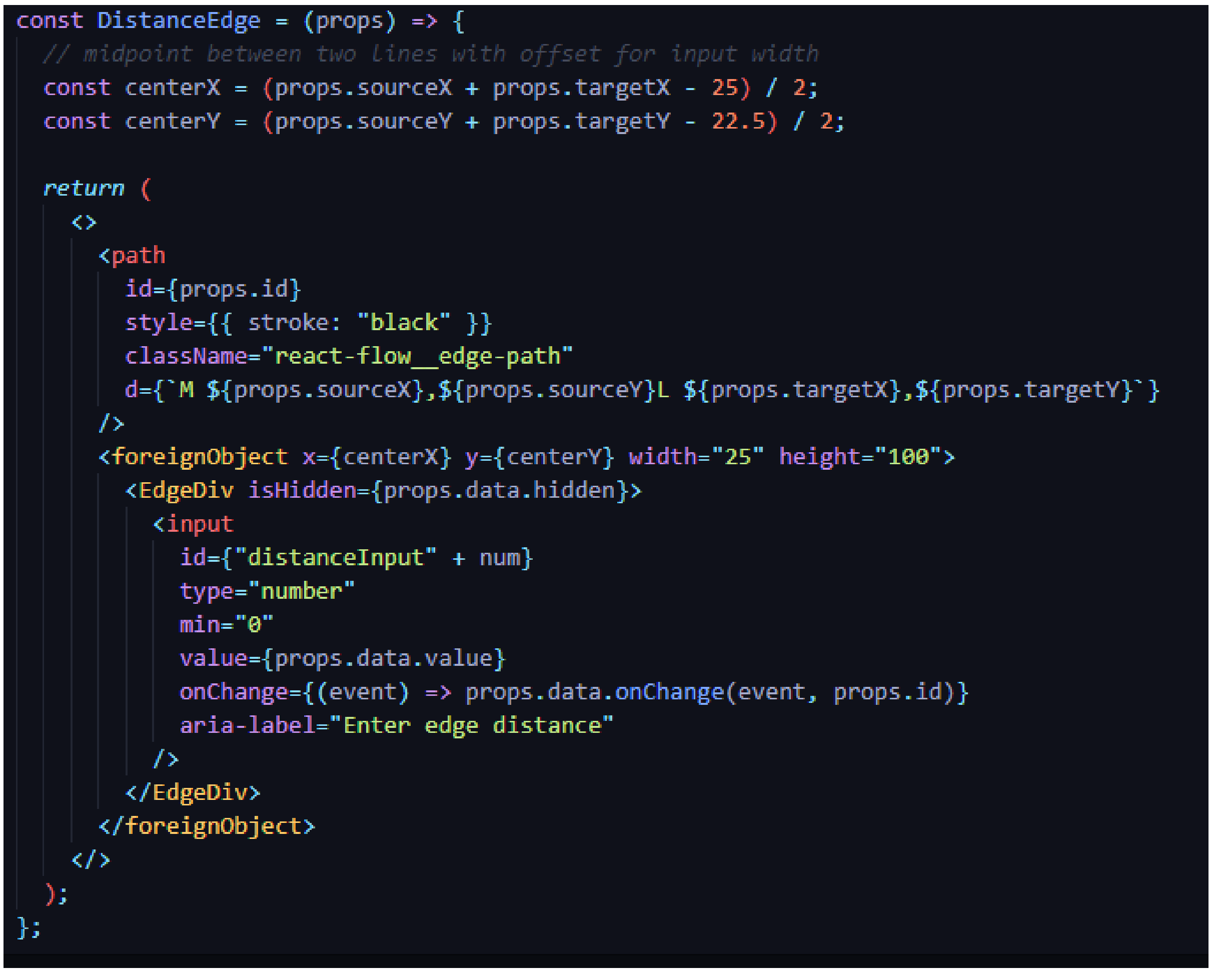
```
function updateGraph(algorithmObject, currStep) {
  /*
  Updates the style of the nodes as the tree is walked through.
  */
  if (currStep in algorithmObject) {
    const stepsArray = algorithmObject[currStep];
    const visited = stepsArray.visitedNodes.split(",");
    const current = stepsArray.currentNode.split(",");
    const todo = stepsArray.todoNodes.split(",");
```



```
  x = 0;
  while (x < lenElements) {
    let element = elements[x];
    if (
      visited.includes(element["id"]) &&
      !current.includes(element["id"])
    ) {
      element["style"] = visitedNodeStyle;
    } else if (
      current.includes(element["id"]) &&
      !goals.includes(element["id"])
    ) {
      element["style"] = currentNodeStyle;
    } else if (todo.includes(element["id"])) {
      element["style"] = todoNodeStyle;
    }
    x += 1;
  }

  triggerRender();
}
```

updateGraph function snippet

It became necessary for us to create our own custom edge and node components to replace the default React Flow ones as we needed more adaptive elements that could store values. As users needed to be able to alter the values of the distance between nodes and the h(n) value within the node we needed to expand the functionality of the nodes with input. Fortunately, React Flow allowed for this. We wanted our edge lines to be straight and the value input to be displayed mid line, we found the center point but needed an offset as the foreignObject did not begin in the middle. We set the input so it could only be an integer greater than 0, and a value isHidden which hid this input when it wasn't needed such as when breadth-first search is selected.



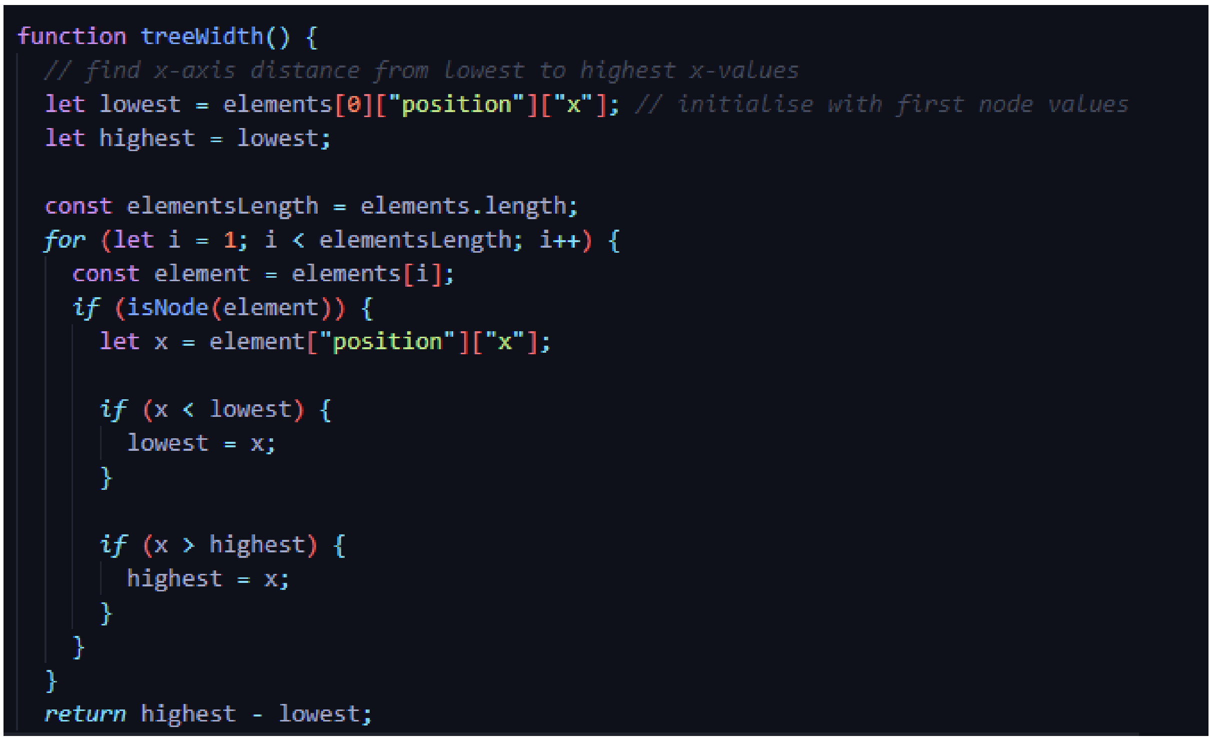
```
const DistanceEdge = (props) => {
  // midpoint between two lines with offset for input width
  const centerX = (props.sourceX + props.targetX - 25) / 2;
  const centerY = (props.sourceY + props.targetY - 22.5) / 2;

  return (
    <>
      <path
        id={props.id}
        style={{ stroke: "black" }}
        className="react-flow__edge-path"
        d={`M ${props.sourceX},${props.sourceY}L ${props.targetX},${props.targetY}`}
      />
      <foreignObject x={centerX} y={centerY} width="25" height="100">
        <EdgeDiv isHidden={props.data.hidden}>
          <input
            id={"distanceInput" + num}
            type="number"
            min="0"
            value={props.data.value}
            onChange={(event) => props.data.onChange(event, props.id)}
            aria-label="Enter edge distance"
          />
        </EdgeDiv>
      </foreignObject>
    </>
  );
};
```

DistanceEdge code snippet

4.3 Comparison

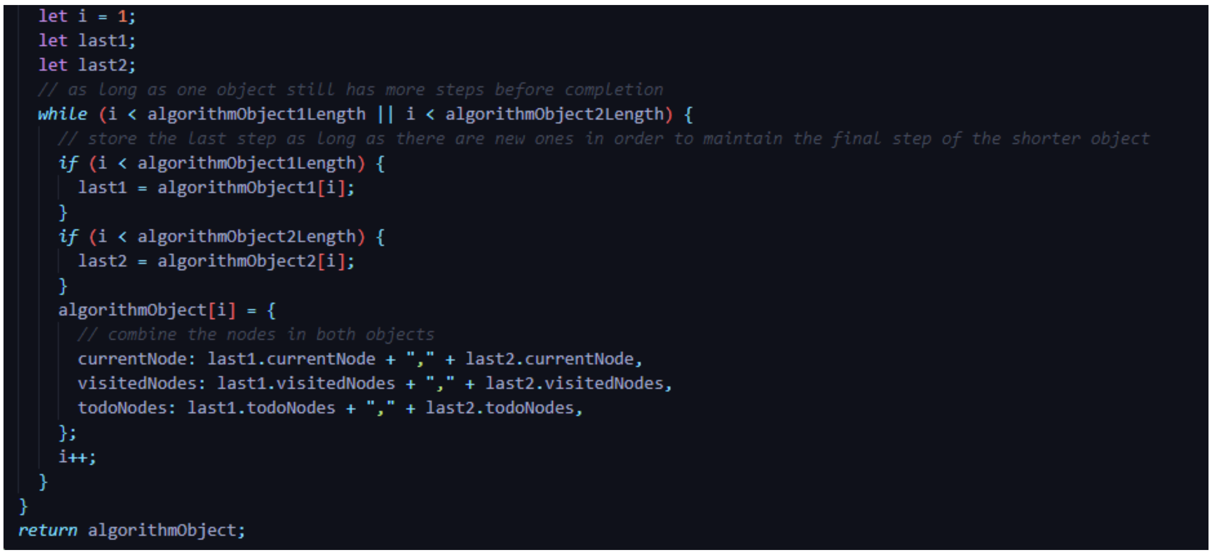
In our original design and proposal we had planned an algorithm comparison feature that used two separate canvases and controls. The more we discussed the idea the worse it sounded as we wanted our web application to remain easy to use in comparison to other algorithm visualisation tools and having two separate playback controls would be confusing. We decided that the algorithm comparison feature would create a replication graph of itself when comparing is active and when it was closed again the original graph would return.

To apply this we needed a way to make sure when the duplication graph was rendered, it wasn't colliding with the original graph. We created a treeWidth() function to find locations of the current elements in the graph, using the x value saved in the position of the elements object. A gap of x1.5 was then made to ensure a large enough gap while not being too large to not be able to compare different algorithms without zooming out too much.



```javascript
function treeWidth() {
  // find x-axis distance from lowest to highest x-values
  let lowest = elements[0]["position"]["x"]; // initialise with first node values
  let highest = lowest;

  const elementsLength = elements.length;
  for (let i = 1; i < elementsLength; i++) {
    const element = elements[i];
    if (isNode(element)) {
      let x = element["position"]["x"];

      if (x < lowest) {
        lowest = x;
      }

      if (x > highest) {
        highest = x;
      }
    }
  }
  return highest - lowest;
```

As in comparison there are two algorithms selected there are also two algorithm objects. We decided to combine both these algorithms so we could reuse the updateGraph and stepping functions. The function combines the values (visited nodes, current node, to-do nodes, etc) within each algorithm object and merges them into one at each step so that at each step the visual representation of what is occurring will still happen for both graphs.

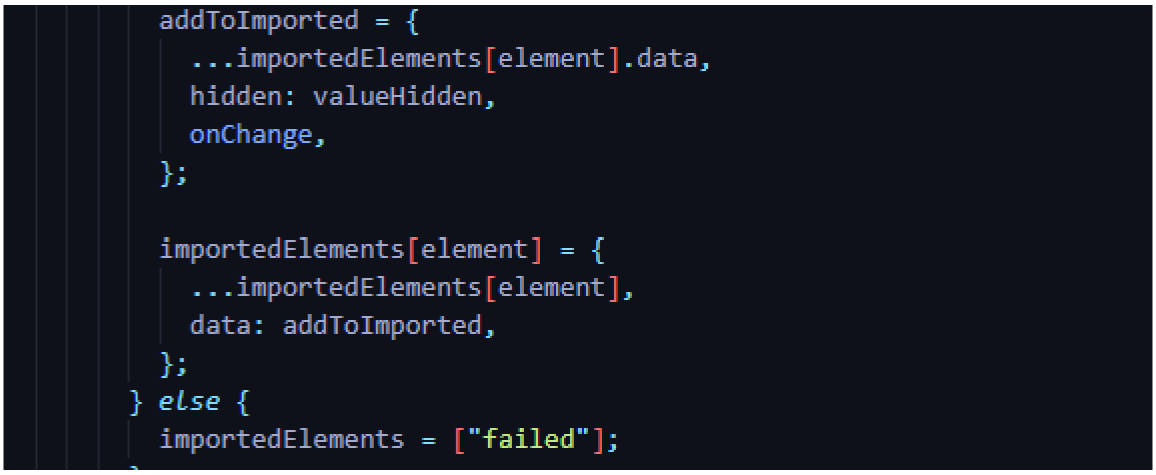


```javascript
  let i = 1;
  let last1;
  let last2;
  // as long as one object still has more steps before completion
  while (i < algorithmObject1Length || i < algorithmObject2Length) {
    // store the last step as long as there are new ones in order to maintain the final step of the shorter object
    if (i < algorithmObject1Length) {
      last1 = algorithmObject1[i];
    }
    if (i < algorithmObject2Length) {
      last2 = algorithmObject2[i];
    }
    algorithmObject[i] = {
      // combine the nodes in both objects
      currentNode: last1.currentNode + "," + last2.currentNode,
      visitedNodes: last1.visitedNodes + "," + last2.visitedNodes,
      todoNodes: last1.todoNodes + "," + last2.todoNodes,
    };
    i++;
  }
}
return algorithmObject;
```

5. Problems solved

5.1. Reusable trees

Originally we had planned to implement a screenshot feature so that at any point during use the user would be able to capture an image of the tree in its current state. Ultimately however this feature would have been somewhat useless as most operating systems currently offer a first party screen capturing tool. We still liked the idea of the user having the ability to save their tree's for reuse. We discussed using a database so the user could create an account and save their trees but this implementation went against the idea of our tool simple and quick to use. Our solution to this issue was to integrate a JSON reader and writer for our elements array. Our users could now save and load their trees. However, the issues didn't cease there, as our elements contain input for the h(n) value and edge distance value, they have a function that handles change within those inputs. After some manual testing we noticed this issue as after loading in a JSON file, changing these values would cause the site to crash. We created a function that initializes these elements, it first makes sure the elements in the JSON are the correct and then adds the onChange function to each element's value, solving this issue.



```
        addToImported = {
          ...importedElements[element].data,
          hidden: valueHidden,
          onChange,
        };

        importedElements[element] = {
          ...importedElements[element],
          data: addToImported,
        };
      } else {
        importedElements = ["failed"];
```

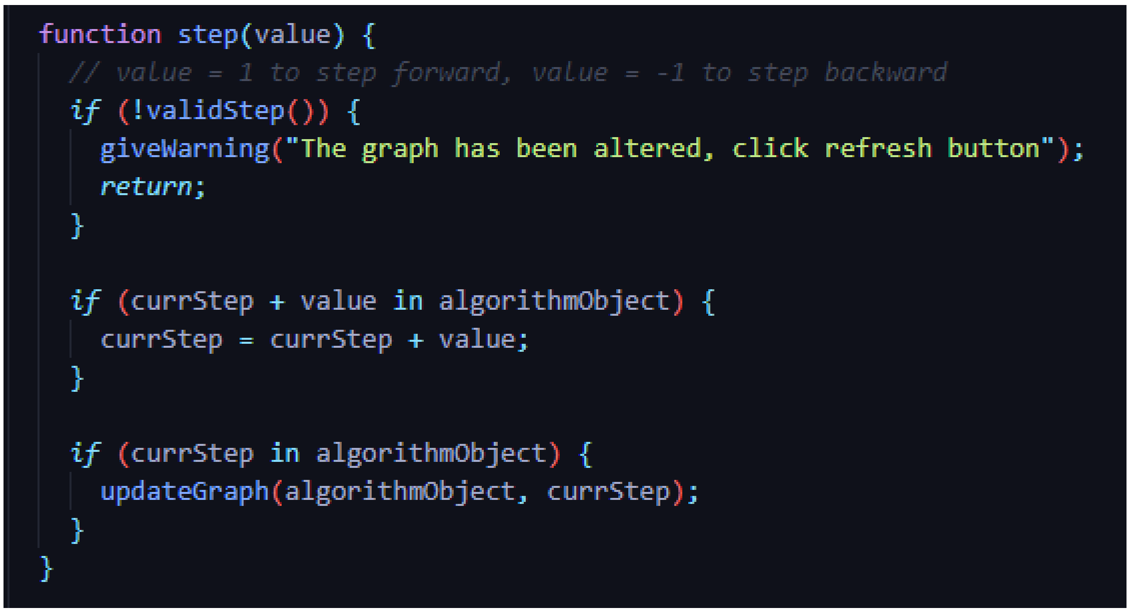
initializeElements function snippet

5.2. Unique IDs for elements

We ran into a number of issues regarding clashing IDs. Up until very late in the development process IDs were generated based on how many elements were in the elements array and before that it was just incremental. We first started running into issues when we enabled deletion because if a node in the middle of the elements array was deleted, the rest of the elements after that deleted node would not decrement. This meant that as soon as a new element was added to the canvas a ID would be overwritten causing an element in the tree to be deleted. We tested checking each element in the arrays ID and incrementing off the final element ID however as we wanted to let the user upload their elements as letter too so we decided to randomly generate the IDs and have a check to see if that ID is already present. This also lets us implement a limit on the number of nodes, currently set to 100.

5.3. Playback Controls

When developing our playback controls we ran into many problems due to the number of separate functions we had for playback such as stepping forward, pausing, rewinding, etc. As the play/pause button called an asynchronous function which called itself based on the amount of time that had passed we ran into issues where if other playback control buttons were pressed there would be overlaps and the visualisation would not work as intended sometimes even crashing the web application. We took a step back and discussed how we might go about redesigning these features. Our new implementation was heavily improved and while actually being a lot simpler. We implemented a global step counter as well as interrupts which if called would pause playback and/or give you a warning. Each playback button called the step function would receive a value to go either forward or backward in the visualisation. This made implementing the rest of the visualisation a lot easier as we could call these interrupts whenever we necessary. Having this step counter also allowed us to use it for indexing our object that stores what happens in the algorithm at each step.



```
function step(value) {
  // value = 1 to step forward, value = -1 to step backward
  if (!validStep()) {
    giveWarning("The graph has been altered, click refresh button");
    return;
  }

  if (currStep + value in algorithmObject) {
    currStep = currStep + value;
  }

  if (currStep in algorithmObject) {
    updateGraph(algorithmObject, currStep);
  }
}
```

Step function snippet

5.4. Remote work

Due to the current circumstances with Covid-19, we have had to attend college and ultimately complete our project remotely. This was a very different experience for us as we had completed our project last year while meeting on the college campus and having in-person meetings with each other and our previous supervisor.

To overcome this difficulty, we regularly had Zoom meetings and discussions with our supervisor about the project and current tasks. Keeping in regular contact was important in staying on schedule throughout the project. The use of GitLab helped us to work remotely as we were able to work separately, at home and combine or merge our work easily and safely.

6. Results:

6.1. Testing

Throughout the duration of the project, we used a combination of multiple testing techniques to ensure we produce software that is of high quality. To start, we began by finding the best testing strategies to adopt that fit our project. We did this by identifying the key aspects of our project and the most suitable methods to test them. In this section, we will discuss how we approached testing and the results found, as well as any choices we made in response to such results.

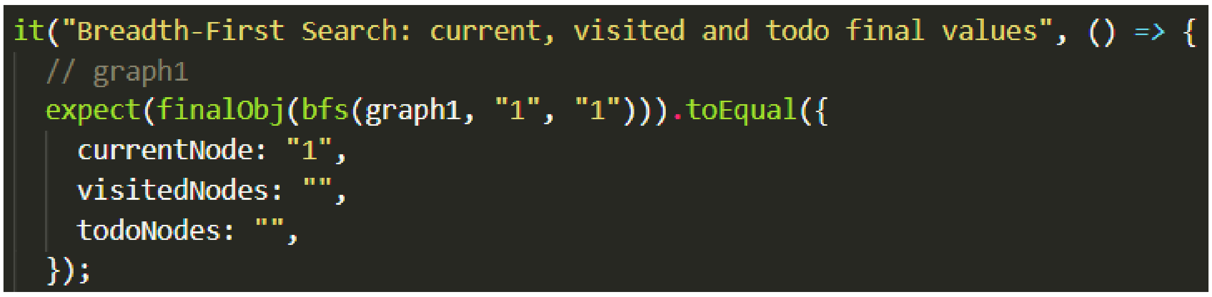
For unit testing, manual functional testing and accessibility testing, we recorded results in Google Sheets, as it allowed us to store information in tables easily, with different tabs assigned to different testing strategies. This is a link to view our test results. Recording test results has enabled us to take a structured approach to testing, making it easier to identify issues and communicate between each other about how to fix them or make improvements.

6.1.1 Unit testing

We used unit testing to verify that code we have written throughout the project is behaving as expected. The main elements of the project that we performed unit testing on were the JavaScript functions used throughout the application as well as the React components that are rendered.

To do this, we made use of the Jest testing framework. Jest is a Node-based JavaScript framework which allows you to test web applications that make use of Node.js and React, among many other things. As we used Create React App to set up our application, we believed it was the best choice because Jest is used as its test runner. We used this to test the logic within our application and functions. To test some code, we specify the result or outcome that we expect to happen as a result of executing it. If the actual result does not match the expected result, the test is a fail. Only when the two results match completely, is the test considered successful.

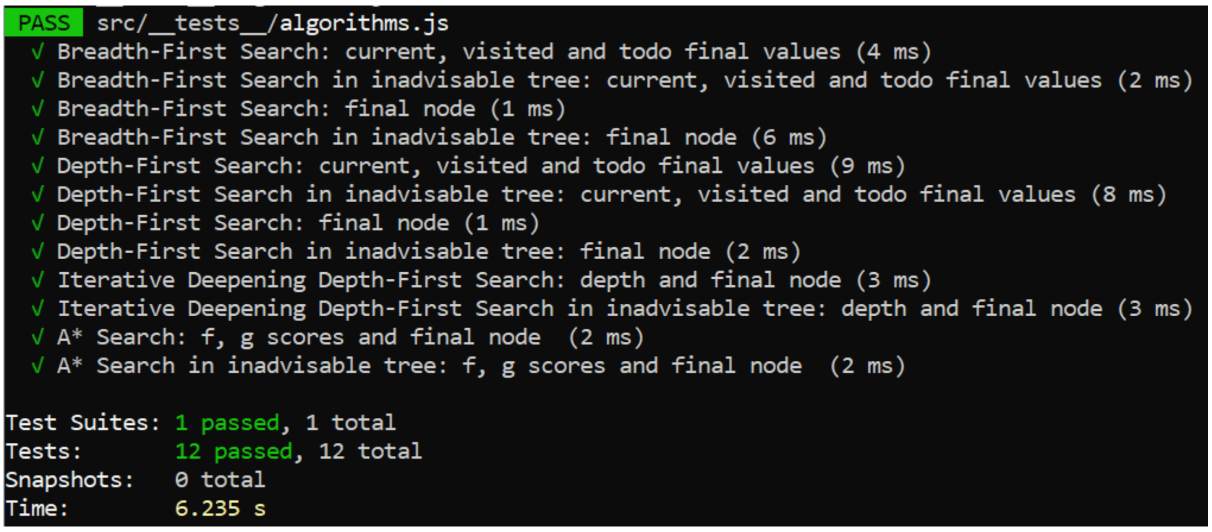
The code snippet below shows one segment of a larger test to check the final values for current, visited and to-do nodes that are returned after conducting a Breadth-First Search on a graph. The final object within the algorithm object, or the information from the final step of the search is retrieved as the actual result. The expected result is defined after we work out what the result should be if the search is conducted correctly.



```
it("Breadth-First Search: current, visited and todo final values", () => {
  // graph1
  expect(finalObj(bfs(graph1, "1", "1"))).toEqual({
    currentNode: "1",
    visitedNodes: "",
    todoNodes: "",
  });
```

Breadth-First Search final values test snippet

To run our unit tests, we used the command `npm test` inside the project directory. This command provided clear, efficient output for all tests, with the ability to pick and choose which specific tests will be running. With this command running, the tests would run again any time a test file was changed and saved, generating results immediately. This helped us to test our code easily and quickly. Below is an example of the output produced by using the test command to run the tests within `src/__tests__/algorithms.js`, which is specifically used to test our algorithm implementations. As you can see, the ability to create custom names for each test makes it easy for us to identify the cause of a fail within a test suite.



```
 PASS  src/__tests__/algorithms.js
  √ Breadth-First Search: current, visited and todo final values (4 ms)
  √ Breadth-First Search in inadvisable tree: current, visited and todo final values (2 ms)
  √ Breadth-First Search: final node (1 ms)
  √ Breadth-First Search in inadvisable tree: final node (6 ms)
  √ Depth-First Search: current, visited and todo final values (9 ms)
  √ Depth-First Search in inadvisable tree: current, visited and todo final values (8 ms)
  √ Depth-First Search: final node (1 ms)
  √ Depth-First Search in inadvisable tree: final node (2 ms)
  √ Iterative Deepening Depth-First Search: depth and final node (3 ms)
  √ Iterative Deepening Depth-First Search in inadvisable tree: depth and final node (3 ms)
  √ A* Search: f, g scores and final node  (2 ms)
  √ A* Search in inadvisable tree: f, g scores and final node  (2 ms)

Test Suites: 1 passed, 1 total
Tests:       12 passed, 12 total
Snapshots:   0 total
Time:        6.235 s
```

Output from algorithms unit tests

To run unit tests on the rendering of components, we used a combination of the Jest framework with the React Testing Library. The React Testing Library provides light-weight functions to use with the React DOM. React components are tested with real DOM nodes, which allowed us to write unit tests that simulate the expected behavior of our users. These tests helped us to make sure that the web application responded to user input and actions correctly. This is important in guaranteeing user satisfaction.

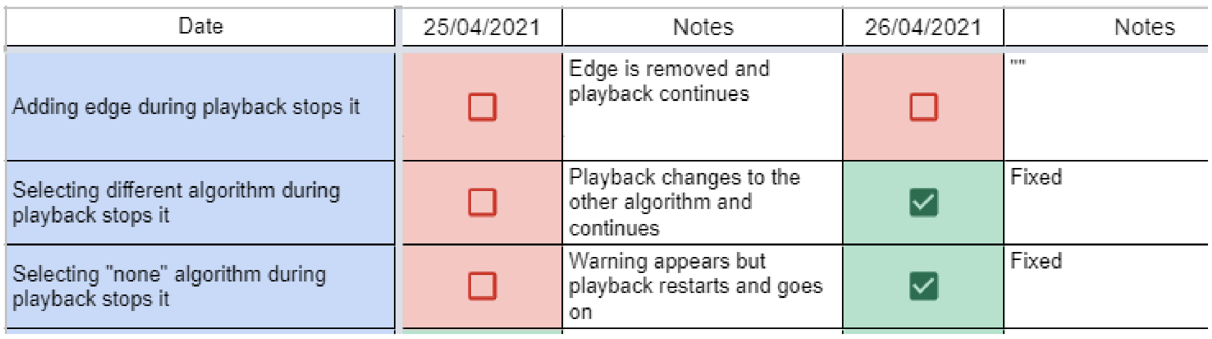
To record our unit tests, we wrote the number of tests that were run and the pass rate. We also made note of any changes since the tests were last run, such as the names of new tests that were added or issues that may have arisen in the meantime. This was particularly useful in identifying possible regression over the course of the development of the project.

Unit testing has improved our quality of code. We have been able to identify bugs early and remove them, continually improving our code. This was crucial as bugs must be identified before full release to users. We also found that unit testing helped us to identify inconsistencies in our design, allowing us to make changes and increase its quality.

6.1.2 Manual functional testing

Functional testing of software is required to validate whether or not the application is functioning as it should. These tests can pass or fail because the application either does what it should or it does not. We use functional testing to make sure the requirements of the project are met, which we set out and adapted from our initial proposal. We also used functional testing to measure how fit our application was to be deployed for use.

Ad hoc testing is a popular method used by developers during development to test software while building it. Typically, this form of testing is unstructured, unplanned and undocumented. At the start of our development process, we would write some code, compile it and see if it works the way we had expected. Over time, we saw the value in conducting these tests regularly, not only on the code we changed but the whole project, as with changes can come regression. To combine ad hoc testing with a more formal approach, we made a list of actions to manually carry out on the web application, with the expected outcome. Utilising Google Sheets' features, we added check boxes beside each different action. As features got added, the list of actions grew. See below for a segment of the manual test sheet at an earlier stage in the project.



Date	25/04/2021	Notes	26/04/2021	Notes
Adding edge during playback stops it	☐	Edge is removed and playback continues	☐	""""
Selecting different algorithm during playback stops it	☐	Playback changes to the other algorithm and continues	☑	Fixed
Selecting "none" algorithm during playback stops it	☐	Warning appears but playback restarts and goes on	☑	Fixed

Selection of manual functional tests conducted on two different dates

We regularly carried out manual functional tests using this checklist approach. We would launch the web application and go through the list, completing each action and comparing the real result to the expected outcome to decide if the test was successful. If a test was successful and the application behaved as expected, the box would be checked. If not, the box would be left blank, indicating that the result was not what it should have been. Beside this, we also wrote information about the inconsistencies found or what caused them so we could reproduce the errors easily when debugging.

This testing proved to be extremely helpful throughout the course of the project. It is easy when implementing or editing a feature to only test that feature alone. It can also happen that with so many features implemented, you forget what exactly it is you need to test. Using this approach made us go through all functionality, making it easy to identify problems, solve them and move on. This approach also helped us to find certain conditions that cause the application to behave in a non-ideal way, allowing us to make changes to code and rectify the issue.

6.1.3 Accessibility testing

Accessibility is very important as it provides all people with equal access to the web application. We carried out some accessibility tests using tools online to find accessibility issues with the application.

The first tool was the Lighthouse tool by Google. The tool is automated and aims to improve the quality of web pages. Using the `lighthouse https://student.computing.dcu.ie/~whytee6/` command generated a report in the current directory. This report included audits on the web page's performance, accessibility, best practices and SEO (search engine optimisation). The results were presented in an easy to understand format and helped us to identify where we could make improvements to the web application. When Lighthouse pointed out an issue, we changed or removed it and ran the test again to see if the results would improve.

Another tool we used was the Web Accessibility Evaluation Tool (WAVE). Using this tool, we could enter our web page address and run an accessibility evaluation. This pointed out some issues that were not mentioned in the Lighthouse report, such as flagging numerous buttons which were missing an accessible aria-label to allow people using screen readers to know what the purpose of a button is.

We made changes to our code to increase accessibility throughout development as this would allow more people to make use of our application. Accounting for accessibility also improves the quality of our code as accessibility is now a standard and in some cases a requirement. This means incorporating accessibility into our web application has helped us follow best practices.

6.1.4 User testing

To evaluate the usability of our web application, we asked a number of people to take part in end-user testing. To conduct this testing, we asked those taking part to use the web application for a short while before answering questions in a Google Forms survey. An example of just some of the questions asked:

- How would you describe your general computer skills?
- Do you have a prior understanding of A.I Search Algorithms?
- Do you have any recommendations for features to be added to this site that could improve your experience while using it?

We received some very useful comments from participants, which we acknowledged and used to improve the web application's usability.

Multiple participants pointed out that it would help their understanding of the web page if the sidebar and tutorial were open when first accessing the page as it was not clear what they needed to do. To address this, we changed both the tutorial and the sidebar to be open by default when someone accesses the web application.

The majority of participants in the survey did not fit our original end-user description of people with prior experience in learning A.I search algorithms. This meant we received multiple answers from people stating that they would like an explanation of the different algorithms to aid their understanding of the visualisations. This feedback was helpful because it helped us identify issues to address to make the web application more accessible to people in general.

Participants also provided positive feedback, such as mentioning the "efficient and easy to use" interface and the "very helpful" red reminders about what you need to do when setting up a visualisation. The positive feedback helped us find which parts of the web application we should maintain as they are, without making big changes unless we are enhancing the best parts of it.

Participants also provided recommendations for additional features that we could implement in the future to improve the usability of the web application. Two excellent suggestions were a static node colour key on the canvas and a small description of the application under the title in the navigation bar.

6.2 Conclusion

Overall, we are happy with the A.I Search Visualiser. We believe that with little experience in using JavaScript, React and other technologies used in this project, we both learned a lot and put our new skills to great use. The application has achieved the majority of our initial goals as set out in the proposal phase of the project. The application does what we had set out to do, that being visualising algorithms used in A.I search. The web app can assist those learning algorithms for the first time and those just looking to refresh their memory.

The user interface is extremely clean and easy to navigate, with many components being reactive so that the user can focus on the information they want. Through user testing, we received positive feedback about the UI. The application has a helpful tutorial, containing a key for the graph as well as instructions on how to use the website. The user has the ability to draw their own graph as well as save their creation locally, which they can reuse at a later time if they decide to upload it.

We are also confident in the quality of our final product because of the various types of testing we have used. We implemented code and tested it throughout the project, ensuring high quality.

There are 4 different algorithms currently available for visualisation with the ability to compare them to each other on command using the numerous playback controls available to users.

7. Future work

There is plenty of room for this project to continue growing, and we have plenty of idea’s however due to time constraints we weren’t able to implement them on time. The next feature on our list to add is graphs, currently the system only supports directional edges however we would like to implement undirected edges so that the elements are a graph rather than a tree. The most obvious of additional features would be to provide the user with a greater choice of algorithms such as Greedy Search which would be extremely easy to implement now that h(n) value input is accepted. Including A* in comparison would also be a priority.

A feature which we think would have been very useful, especially when we were studying algorithms in CA318 was visualising solving 8-puzzles. We would have let the user input the original 8-puzzle in the sidebar and the visualization would automatically generate each of the needed to solve the puzzle based on what algorithm was selected. This would have been a great resource to visualise the algorithms in a unique way that would have been extremely useful as a lot of universities use 8-puzzles to teach certain algorithms. If we had decided to do this we would have had to focus on implementing this from the start as it could be its own project.

Additional features could be added to the information box such as the pseudo code for what is currently happening in the algorithm as well as possibly letting the user know the current cost so they could compare each tree. Animation could be added to the visualisation to give even more visual information to the user as well as improve the user interface.

We acknowledge that in future development, we could improve the accessibility of the web application even further. While we have attempted to use a colour blindness friendly palette for nodes and general UI, we can see using online filters that it can be hard for some users to tell the difference between the different coloured nodes. One way we can fix this is by implementing a “colour blind mode” which will use patterns on nodes, which are easy to differentiate between, despite a lack of significant colour contrast. Another accessibility issue we would like to address is the possibility of the web application being difficult to use for those with poor fine motor skills, such as the action of clicking on small circles to create edges between nodes. The method to combat this would be to provide an alternative way to create edges that is easier for people with this issue. We would also like to implement keyboard shortcuts for people who are inclined to not use a mouse when visiting a webpage.

In terms of applying the knowledge gained from this project in other projects in the future, there are a number of possibilities. JavaScript and React are popular technologies used in web development and A.I algorithms are becoming increasingly popular as time goes on.