

Path-finding Algorithms and Solving Mazes

Oliver Temple

-

Spring 2022

Contents

1	analysis	3
1.1	Project Description	3
1.2	Research	4
1.2.1	Overview	4
1.2.2	Research Log	4
1.2.2.1	Maze Generating Algorithms	4
1.2.2.2	Path Finding Algorithms	5
1.3	Project Background	6
1.3.1	Current Systems	6
1.3.1.1	Example 1	6
1.3.1.2	Example 2	7
1.3.1.3	Example 3	8
1.3.2	Proposed Solution	10
1.3.3	Prospective Users	10
1.3.3.1	Questions To Users	10
1.4	Objectives	11
1.4.1	Generate Mazes	11
1.4.2	Solve Mazes	11
1.4.3	Customisation	12
1.4.4	User written algorithms	12
1.4.5	Update Visualization	12
2	Documented Design	12
2.1	Visualization Structure	12
2.2	User Interaction Flow Chart	14
2.3	User Interface	15
2.4	Classes Breakdown	15
2.4.1	Genereator	15
2.4.2	Solver	15
2.4.3	Maze	16
2.4.4	Node	16
2.5	Algorithms	17
2.5.1	Prims	17
2.5.2	Recursive Backtracking	17
2.5.3	Dijkstra's	17
2.5.4	Depth First Search	18
2.5.5	Breadth First Search	18
2.5.6	Greedy Search	18
2.6	Subroutine Breakdown API	19
2.6.1	generator.py: recursive_backtracking	19
2.6.2	generator.py: recursive_backtracking_run	19
2.6.3	generator.py: prims	19
2.6.4	solver.py: get_adjacent_paths	19

2.6.5	solver.py: dijkstra	19
2.6.6	solver.py: dfs	19
2.6.7	solver.py: bfs	20
2.6.8	solver.py: manhattan	20
2.6.9	solver.py: euclidean	20
2.6.10	solver.py: greedy	20
2.6.11	grid.py: Node: __init__	20
2.6.12	grid.py: Node: load	20
2.6.13	grid.py: Node: serialize	21
2.6.14	grid.py: Grid: __init__	21
2.6.15	grid.py: Grid: load	21
2.6.16	grid.py: Grid: serialize	21
2.6.17	grid.py: Grid: generateGrid	21
2.6.18	grid.py: Grid: printGrid	21
2.7	Subroutine Breakdown react app	21
2.7.1	App.js: App: Constructor	21
2.7.2	App.js: App: componentDidMount	21
2.7.3	App.js: App: setHeuristic	21
2.7.4	App.js: App: setSpeed	22
2.7.5	App.js: App: setSize	22
2.7.6	App.js: App: setStart	22
2.7.7	App.js: App: setEnd	22
2.7.8	App.js: App: setAlgorithm	22
2.7.9	App.js: App: setSolve	22
2.7.10	App.js: App: async should_solve	22
2.7.11	App.js: App: async clear_node_index	23
2.7.12	App.js: App: async fetchGrid	23
2.7.13	App.js: App: async solveGrid	23
2.7.14	App.js: App: async clearGrid	23
2.7.15	App.js: App: render	23
2.7.16	DisplayGrid.jsx: DisplayGrid: handelDrop	23
2.7.17	DisplayGrid.jsx: DisplayGrid: setDragObject	23
2.7.18	DisplayGrid.jsx: DisplayGrid: renderTable	23
2.7.19	DisplayGrid.jsx: DisplayGrid: render	24
2.7.20	DisplayNode.jsx: DisplayNode: handelDragStart	24
2.7.21	DisplayNode.jsx: DisplayNode: handelDrop	24
2.7.22	DisplayNode.jsx: DisplayNode: handelDragOver	24
2.7.23	DisplayNode.jsx: DisplayNode: handelDragLeave	24
2.7.24	DisplayNode.jsx: DisplayNode: render	24
2.7.25	Settings.jsx: Settings: componentDidMount	24
2.7.26	Settings.jsx: Settings: componentWillUnmount	24
2.7.27	Settings.jsx: Settings: handelClickOutside	24
2.7.28	Settings.jsx: Settings: handelSizeChange	25
2.7.29	Settings.jsx: Settings: setHeuristic	25
2.7.30	Settings.jsx: Settings: renderSettings	25

3	Technical Solution	25
3.1	Summary Of Skills Used	25
3.2	The Full Code	26
3.2.1	Python API: lambda_function.py	26
3.2.2	Python API: grid.py	28
3.2.3	Python API: generator.py	30
3.2.4	Python API: solver.py	32
3.2.5	React App: App.js	36
3.2.6	React App: Menu.jsx	41
3.2.7	React App: MenuKey.jsx	42
3.2.8	React App: DisplayGrid.jsx	42
3.2.9	React App: DisplayNode.jsx	44
4	Testing	46
4.1	Test 8: Pass	50
4.2	Test 9: Pass	50
4.3	Test 10: Pass	50
5	Evidence of Completeness	50
5.1	Objectives Again	50
5.1.1	Generate Mazes	50
5.1.2	Solve Mazes	51
5.1.3	Customisation	51
5.1.4	User written algorithms	51
5.1.5	Update Visualization	51
5.2	Evaluation of Objectives	51
5.2.1	Generate Mazes	51
5.2.2	Solve Mazes	52
5.2.3	Customisation	52
5.2.4	User written algorithms	52
5.2.5	Update Visualization	52
6	Evaluation	52
6.1	Independent Feedback	52
6.2	Response to Independent Feedback	53
6.2.1	Gif	53
6.2.2	Speed Slider	53
6.3	Improvements that could be made	54
6.4	Evaluation	54

1 analysis

1.1 Project Description

Path finding algorithms are essential in many aspects of computer science, from computer games to solving complex real world problems, however they can be

difficult to visualize, especially when learning about them for the first time. This project's aim to create a path finding visualization tool that will generate and solve mazes, and to provide a general understanding of the algorithms used.

There are a number of different algorithms that can be used to solve and generate mazes, and this project will focus on the more common ones like Dijkstra's, Depth First Search, Prim's and more. I intend to include algorithms that are different from each other to show the advantages and disadvantages of each.

1.2 Research

1.2.1 Overview

To complete this project, I will need a strong understanding of maze generation and path finding algorithms, how they work and how to model them, knowledge of react and javascript to create a website for the visualization as well as user opinions on what features are needed.

1.2.2 Research Log

I was introduced to path finding algorithms in one of my lessons, where we learned what they were used for and some examples, as well as how they can be modeled. For example, modelling the maze as a graph with weighted nodes of 1 and 0, for the walls and space respectively. The algorithms we looked at were Dijkstra's and A*. We also looked at Prim's algorithm, recursive backtracking and Kruskal's algorithms for maze generation.

1.2.2.1 Maze Generating Algorithms

Prim's In class, we were taught about Prim's algorithm, and how it works. I used the information that our teacher gave us to write my own implementation of Prim's algorithm.

Recursive Backtracking When researching recursive backtracking I can across a website that said:

Here's the mile-high view of recursive backtracking:

- 1. Choose a starting point in the field.*
- 2. Randomly choose a wall at that point and carve a passage through to the adjacent cell, but only if the adjacent cell has not been visited yet. This becomes the new current cell.*
- 3. If all adjacent cells have been visited, back up to the last cell that has uncarved walls and repeat.*
- 4. The algorithm ends when the process has backed all they way up to the starting point.*

<https://weblog.jamisbuck.org/2010/12/27/maze-generation-recursive-backtracking>
I used this along with another visualization that I found to write my own recursive backtracking algorithm.

1.2.2.2 Path Finding Algorithms

Depth First Search For the depth first search, I looked at https://isaacomputerscience.org/concepts/dsa_pathfinding_dfs_bfs?examBoard=ocr&stage=a_level, a resource I often use for studying. I used their easy to understand description to write my own implementation of depth first search. Their website said:

A depth-first search begins at the start node and then searches as far as possible down a branch of the graph, moving forward until there are no more nodes along the current branch to be explored. If the target node is found along the way, the search can stop. Otherwise, it must backtrack and find another branch to explore.

This process uses a stack as a supporting data structure to keep track of the nodes that have not been fully explored. As each node is discovered, it is added to the stack.

Breadth First Search Isaac Computer Science also have a page on breadth first search, also containing a simple description that I used to implement my own breadth first search.

The algorithm starts searching at a designated start node and searches through all the adjacent nodes (neighbours) before moving on. You can think of this process as moving out in waves from a given point. Another simple way to visualise a breadth-first search algorithm is to imagine that you are making a cake one layer at a time. You can't add the next layer unless the previous one is complete.

This process uses a queue as a supporting data structure to keep track of the nodes that have not been fully explored. As each node is discovered, it is added to the queue.

Dijkstra's For dijkstra's, i found this website:<https://daemianmack.org/posts/2019/12/mazes-for-programmers-dijkstras-algorithm.html>, that had a very good description of Dijkstra's algorithm.

1. Determine the starting point of the grid.
2. Record the cost of reaching that cell: 0.
3. Find that cell's navigable neighbors.
4. For each neighbor, record the cost of reaching that neighbor: 1.
5. For each neighbor, repeat steps 3-5, taking care not to revisit already-visited cells.

Greedy Search Having researched the other path finding algorithms, and learnt about greedy algorithms in class, I used the knowledge I already had to write my own greedy search algorithm. A greedy algorithm is one that always chooses the best option at each step. This gives them the advantage of being faster, at the expense of accuracy, as greedy algorithms don't tend to guarantee the optimal solution.

1.3 Project Background

1.3.1 Current Systems

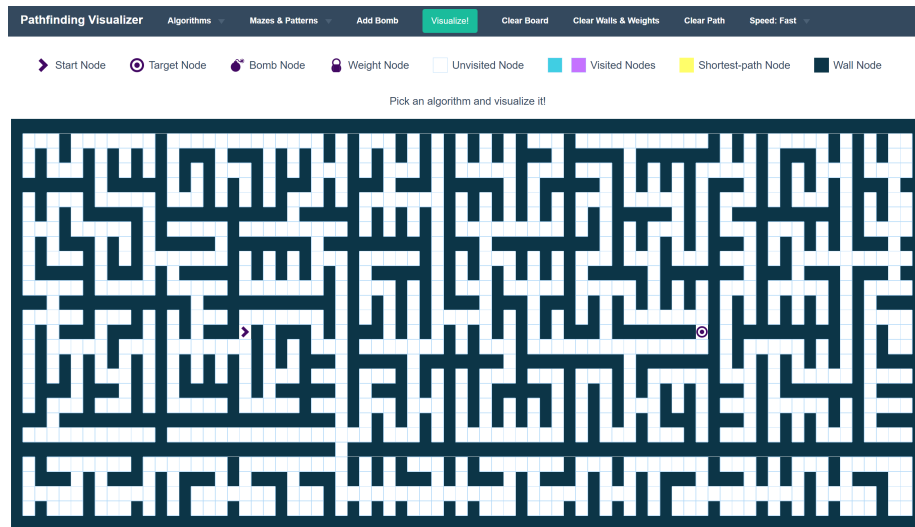
1.3.1.1 Example 1 <https://clementmihailescu.github.io/Pathfinding-Visualizer>

In this example, mazes can be generated with various algorithms, as well as being drawn by the user. The mazes can also be edited after they have been generated. The available maze generation algorithms are:

- Recursive Division
- Recursive Division (vertical skew)
- Recursive Division (horizontal skew)
- Basic Random Maze
- Basic Weight Maze
- Simple Stair Pattern

These mazes can then be solved with a number of different path finding algorithms. The available path finding algorithms are:

- Dijkstra's Algorithm
- A* Search
- Greedy Best-first Search
- Swarm Algorithm
- Convergent Swarm Algorithm
- Bidirectional Swarm Algorithm
- Breadth-first Search
- Depth-first Search



pros

- Many different algorithms to choose from.
- Start and end nodes can be moved.
- Maze can be altered.
- If nodes are moved after visualization has run, then the visualization will update.
- "Bomb" node, adds a via point that the path must go through.

cons

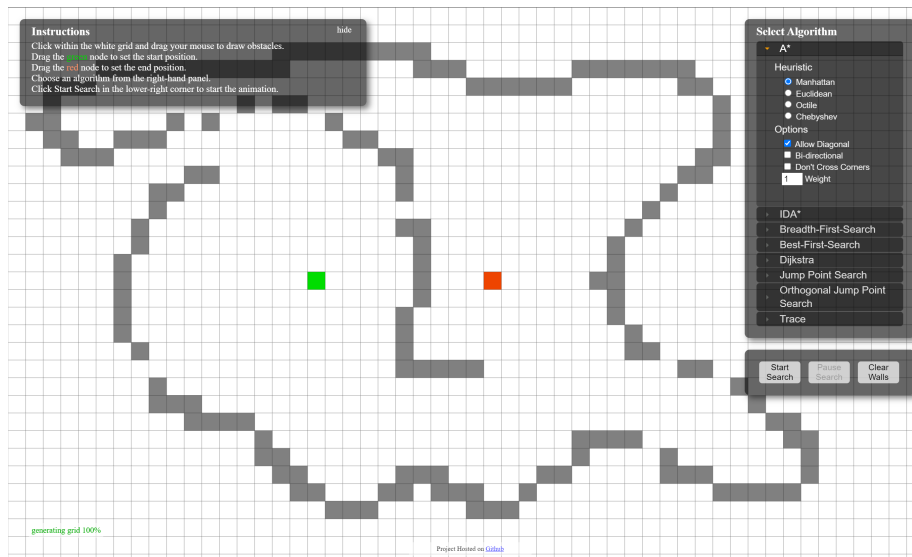
- The visualization is too slow.
- If maze is altered by user after visualization has run, then the visualization will not update.

1.3.1.2 Example 2 <https://qiao.github.io/PathFinding.js/visual/>

In this example, mazes have to be drawn by the user. The maze can then be solved with a number of different algorithms, however, these algorithms have more choice. For example, in the A* option, you can change the heuristic that is used. The available algorithms are:

- A*
- IDA*
- Breadth-First-Search

- Best-First-Search
- Dijkstra
- Jump Point Search
- Orthogonal Jump Point Search
- Trace



pros

- More options to choose from within each algorithm.

cons

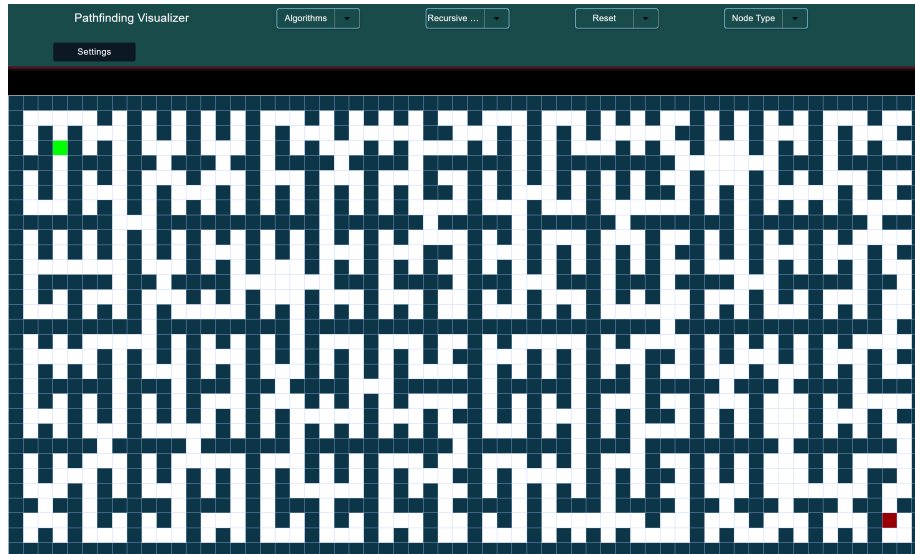
- No maze generation.
- Visualization does not update when maze or start/finish nodes are changed.

1.3.1.3 Example 3 <https://pathfindout.com/>

In this example, mazes can be generated or drawn, however, mazes can only be generated with the recursive division algorithm. There are fewer path finding algorithms to solve the mazes than the others. The available algorithms are:

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

- Depth First Search

**pros**

- Different weighted nodes available.
- Shows how many nodes visited.
- Shows final path length.
- Data structure for some algorithms can be changed.
- Weights of specific node types can be changed.
- Node size can be changed.

cons

- Sometimes generates mazes that cannot be solved.
- Cannot edit maze after visualization has run.
- Only one maze generation algorithm.
- Fewer path finding algorithms to solve the maze.

1.3.2 Proposed Solution

I will make a path-finding visualization that encompasses as many of the merits of the existing solutions as possible, while also tackling as many of the drawbacks. To do this, I will take the following steps:

- Research the different algorithms needed and write the corresponding algorithm.
- Create a mockup of the user interface.
- Create the user interface in react.
- Code the algorithms in python.
- Create an AWS lambda function in python that runs the algorithms as an API.
- Create the react website to visualize the algorithms.
- Add additional features suggested by end users.
- Test visualization and check that it meets all of the objectives.

1.3.3 Prospective Users

The users of this system will most likely consist of teachers and students who are learning about path finding algorithms as it will clearly show how the algorithms function.

1.3.3.1 Questions To Users I asked a student in my Computer Science class, who is a prospective user, some questions about what they would like to see in the visualizer.

Questions

- Q1. What algorithms would you like to see for maze generation?
- Q2. What algorithms would you like to see for solving the maze?
- Q3. What additional features would you like to see in the path finding visualization?
- Q4. Would it be useful to be able to write your own path finding algorithms that can be run in the visualization?

Answers

- A1. I would like to see recursive backtracking and Prim's algorithms used for maze generation, as well as Kruskal's (less important). This is because recursive backtracking and Prim's are similar but each has their own advantages and disadvantages and recursive backtracking is different and uses recursion.
- A2. I would like to see an algorithm (such as Dijkstra's) as well as a heuristic (such as A* or a greedy search), as this will show the difference between a heuristic and an algorithm.
- A3. The ability to change the speed of the visualization.
- A4. I would find it useful to be able to code my own algorithms. This would allow me to use the visualization with other, lesser known algorithms that I may want to visualize.

The answers to these questions confirm what is required from the visualization, which will be reflected in the objectives. The need for contrasting algorithms is something that I will consider when deciding what algorithms to use.

1.4 Objectives**1.4.1 Generate Mazes**

The website should be able to generate mazes using multiple algorithms, including, but not limited to:

- Prim's Algorithm
- Recursive Backtracking

There should also be a brief description of the algorithm that has been selected.

1.4.2 Solve Mazes

The website should be able to solve mazes using multiple algorithms, including, but not limited to:

- Greedy Search
- Dijkstra's Algorithm
- Depth-first Search
- Breadth-first Search

There should also be a brief description of the algorithm that has been selected.

1.4.3 Customisation

The user should be able to customize aspects of the visualization, including:

1. Size of the maze.
2. Speed of the animation.
3. The heuristic used in any heuristic algorithms.

1.4.4 User written algorithms

The website should be able to run algorithms written by the user for both maze generation and solving. To do this I will:

- Supply documentation on parameters needed for the algorithm.
- Supply documentation of the format the maze must be returned in.
- Supply documentation for any other information that the user may need to write an algorithm.

1.4.5 Update Visualization

If the start or end nodes are moved once the visualization has been run, then it should update without the user having to rerun the visualization.

2 Documented Design

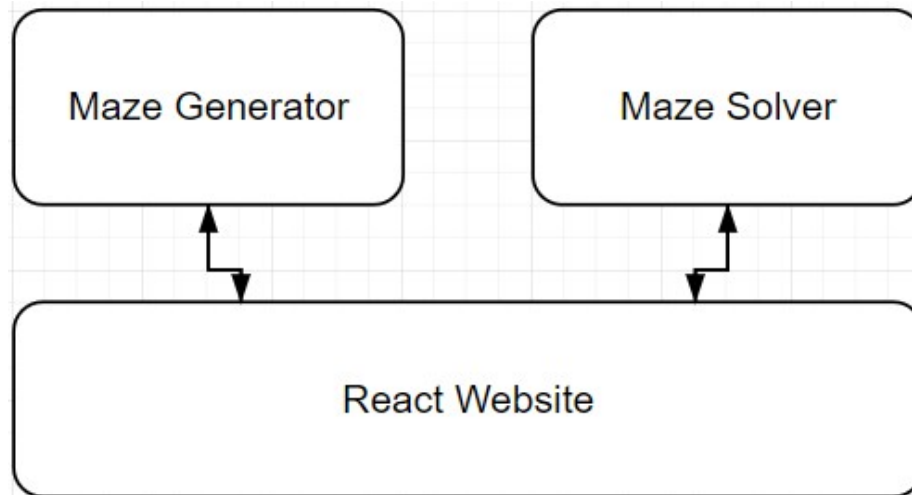
2.1 Visualization Structure

The project will be split into two main sections, the first being the visualization and the second being the python API. The python API will be for generating and solving the mazes, and the react website that will visualize the algorithms. The python API can be subdivided into two further sections, the first being the maze generation and the second being the maze solving. Different parameters will be passed to the API depending on the what the user has requested. These parameters will be:

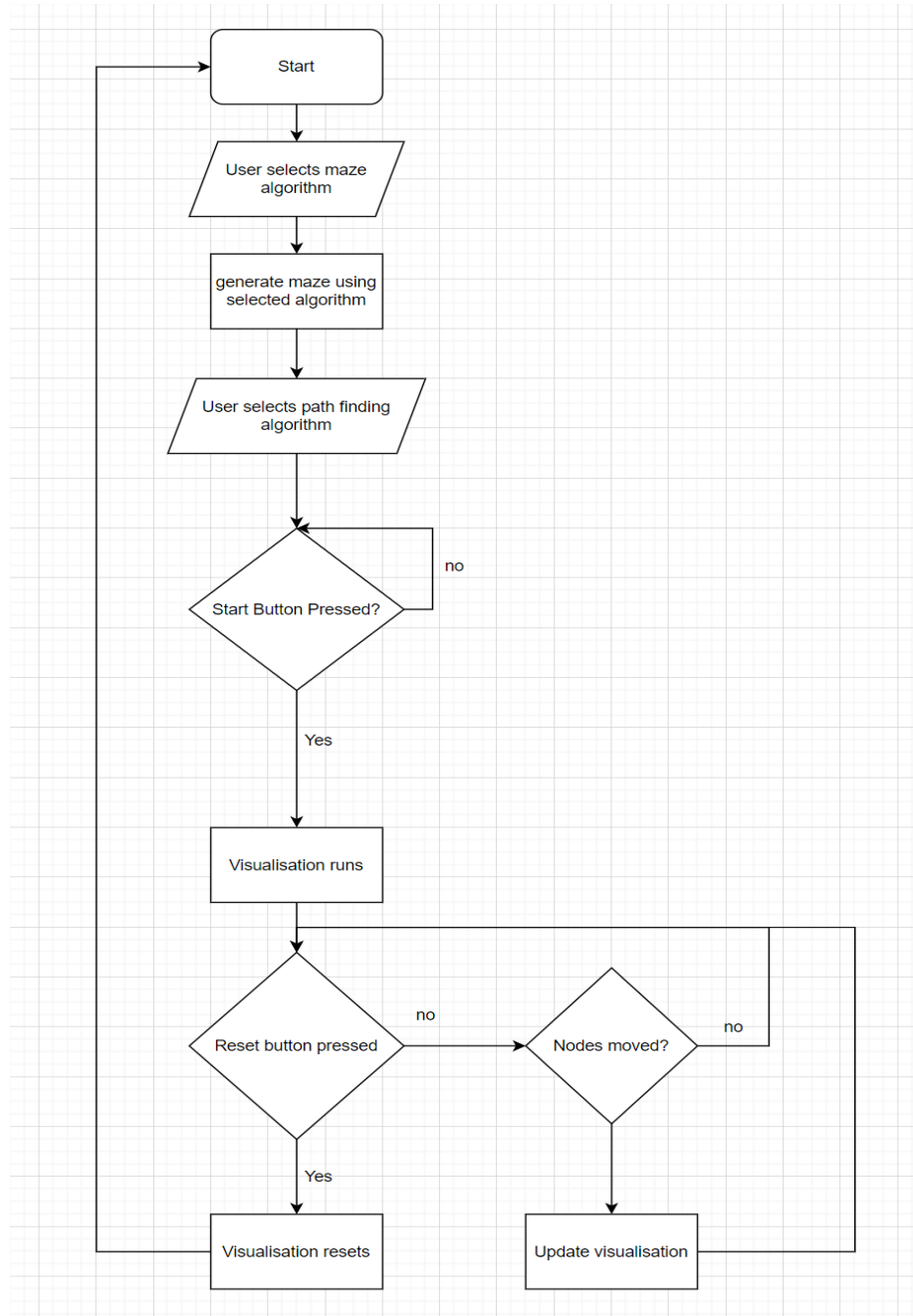
- width
- height
- type (solve/generate/empty maze)
- generate (algorithm for generation)
- solve (algorithm for solving)
- start node location

- end node location

When solving the maze, the maze will be sent to the API as part of the body of the request.

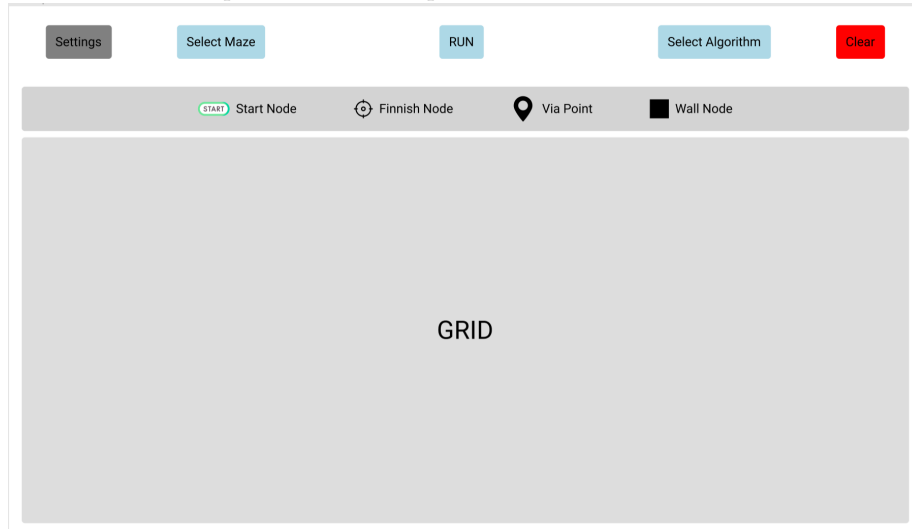


2.2 User Interaction Flow Chart



2.3 User Interface

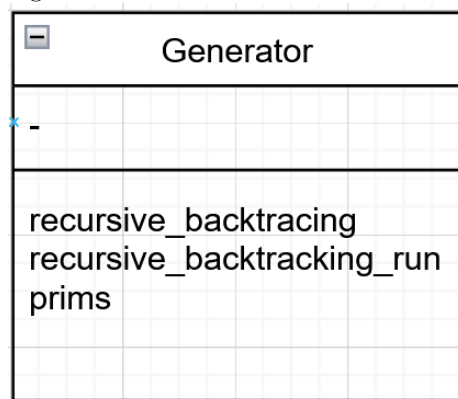
I have used Figma to create a design mockup of how the user interface will look. This allows me to plan what user inputs will be needed.



2.4 Classes Breakdown


2.4.1 Genereator

The generator class will be responsible for generating the maze with different algorithms.




2.4.2 Solver

The solver class will be responsible for solving the maze with different algorithms.

 Solver
-
get_adjacent_paths dijkstra dfs bfs manhattan euclidean greedy

2.4.3 Maze

The maze class will be responsible for storing the maze and any other information that is needed, as well as methods for loading a generated maze to be solved and a serialize method for sending the maze to the react app.

 <i>Grid</i>
height:int width:int grid:list
load generateGrid serialize printGrid (for development)

2.4.4 Node

The node class will be responsible for storing the nodes and any other information that is needed, as well as methods for loading the nodes from a generated maze and a serialize method for sending the maze to the react app.

Node
x:int y:int type:str index:int parent:Node visited:bool distance:str/int wallLeft:bool wallBottom:bool
serialize load

2.5 Algorithms

2.5.1 Prims

Prim's algorithm will have two lists, an inMaze list and a frontier, to store the nodes that are in the maze and in the frontier respectively. To start, the nodes adjacent to the start node will be added to the frontier. Then a random wall will be taken from the frontier, and the wall will be removed from the maze. The node that was connected by removing the wall will then be added to the inMaze list. Any nodes that are adjacent to the new node and not in the maze will be added to the frontier. This will continue while there are nodes in the frontier.

2.5.2 Recursive Backtracking

Recursive backtracking will use a stack to store the previous nodes that have been visited. The stack will be a list of Nodes that are added to the stack as they are visited. If the algorithm reaches a node that has no unvisited adjacent nodes, then the algorithm will backtrack to the last node that was visited by popping off of the stack. This will continue until the algorithm backtracks to the start node.

2.5.3 Dijkstra's

Dijkstra's algorithm works by keeping a priority queue of the nodes that have not been visited, however, all connections have the same weights, as it is solving a maze. The queue will be a list of Nodes, with their distance initially set to "infinity". The algorithm will then search outwards from the start node, adding

the distance from the start to each node as it is discovered, until the end node is reached. The algorithm will then backtrack from the end node to the start node, choosing the node with the lowest distance at each step.

2.5.4 Depth First Search

Depth first search(dfs) works by keeping a stack of the nodes that need to be visited, and adding to the stack as new nodes are discovered. When the nodes are added to the stack, the "parent" attribute of the nodes will be updated so that a path can be drawn. If there are no unvisited nodes connected to the current node, then the algorithm will backtrack by popping a node off of the stack. This will continue until the stack is empty or the end node is reached, at which point the algorithm will return and a path will be drawn.

2.5.5 Breadth First Search

Breadth first search(bfs) is similar to dfs, however a queue should be used instead of a stack. The queue will be a list of Nodes, with new Nodes being added to the end of the list as they are discovered. When the nodes are added to the queue, the "parent" attribute of the nodes will be updated so that a path can be drawn. The algorithm will keep searching until the queue is empty or the end node is reached, at which point the algorithm will return and a path will be drawn.

2.5.6 Greedy Search

Greedy search works by keeping a priority queue, with the heuristic distance from the node to the end node being the priority. The algorithm will start from the start node, searching outwards, adding nodes to the priority queue as they are discovered, with the "parent" attribute being updated so that the path can be drawn. The algorithm will keep searching until the queue is empty or the end node is reached, at which point the algorithm will return and a path will be drawn.

Heuristic Different heuristics can be used for the greedy search. The heuristic I will include are manhattan distance and euclidean distance, as these are some of the most common heuristics.

Manhattan Distance Manhattan is calculated by

$$h(n) = |x_{current\ node} - x_{end\ node}| + |y_{current\ node} - y_{end\ node}| \quad (1)$$

Euclidean Distance Euclidean is calculated by

$$h(n) = \sqrt{(x_{current\ node} - x_{end\ node})^2 + (y_{current\ node} - y_{end\ node})^2} \quad (2)$$

2.6 Subroutine Breakdown API

2.6.1 `generator.py: recursive_backtracking`

Parameters:

Grid—2D array of Nodes

Setup the variables for recursive backtracking by creating a list in unvisited nodes.

2.6.2 `generator.py: recursive_backtracking_run`

Parameters:

Grid— 2D array of Nodes

unvisited— list of Nodes

current— Node

previous— list of previous Nodes

The recursive backtracking algorithm.

2.6.3 `generator.py: prims`

Parameters:

Grid— 2D array of Nodes

Generate the maze using Prim's algorithm.

2.6.4 `solver.py: get_adjacent_paths`

Parameters:

Grid— 2D array of Nodes

node— Node to find adjacent paths

Calculate and return adjacent nodes that are not blocked by walls.

2.6.5 `solver.py: dijkstra`

Parameters:

Grid— 2D array of Nodes

start— Node to start from

end— Node to end at

Use Dijkstra's algorithm to find the shortest path from the start node to the end node and draw a path on the Grid.

2.6.6 `solver.py: dfs`

Parameters:

Grid— 2D array of Nodes

start— Node to start from

end— Node to end at

Use Depth First Search to find the path from the start node to the end node and draw a path on the Grid.

2.6.7 solver.py: bfs

Parameters:

Grid— 2D array of Nodes
start— Node to start from
end— Node to end at

Use Breadth First Search to find the path from the start node to the end node and draw a path on the Grid.

2.6.8 solver.py: manhattan

Parameters:

node1— Node to calculate distance from
node2— Node to calculate distance to

Calculate and return the manhattan distance between two nodes.

2.6.9 solver.py: euclidean

Parameters:

node1— Node to calculate distance from
node2— Node to calculate distance to

Calculate and return the euclidean distance between two nodes.

2.6.10 solver.py: greedy

Parameters:

Grid— 2D array of Nodes
start— Node to start from
end— Node to end at
heuristic— function to use for heuristic

Use Greedy Search to find the path from the start node to the end node and draw a path on the Grid.

2.6.11 grid.py: Node: __init__

Parameters:

x— x coordinate of node
y— y coordinate of node
type— type of node

Constructor for the Node class.

2.6.12 grid.py: Node: load

Parameters:

wallLeft— boolean if there is a wall to the left
wallRight— boolean if there is a wall to the right

Load the walls of the node. Used when loading the maze for solving.

2.6.13 grid.py: Node: serialize

Returns the Node as a dictionary to be sent to the react app.

2.6.14 grid.py: Grid: __init__

Parameters:

height— height of the maze

width— width of the maze

Constructor for the Grid class.

2.6.15 grid.py: Grid: load

Parameters:

grid— 2D array of Nodes

Load the maze into the Grid. Used when loading the maze for solving.

2.6.16 grid.py: Grid: serialize

Returns the Grid as a dictionary to be sent to the react app.

2.6.17 grid.py: Grid: generateGrid

Generate an empty grid.

2.6.18 grid.py: Grid: printGrid

Print the grid, used when developing maze generation and solving algorithms.

2.7 Subroutine Breakdown react app**2.7.1 App.js: App: Constructor**

Parameters:

props— props passed from react

Constructor for the App class.

2.7.2 App.js: App: componentDidMount

Method runs when the component mounts in the DOM, used to load an empty maze when the app loads.

2.7.3 App.js: App: setHeuristic

Parameters:

heuristic— heuristic to use

Callback function to change the heuristic used for the greedy search from the settings component.

2.7.4 App.js: App: setSpeed

Parameters:

speed— speed to use

Callback function to change the speed of the maze generation from the settings component.

2.7.5 App.js: App: setSize

Parameters:

size— size to use

Callback function to change the size of the maze from the settings component, the maze is reset when the size is changed.

2.7.6 App.js: App: setStart

Parameters:

start— start node to use

Callback function to change the start node of the maze when the start node is dragged and dropped to a new location.

2.7.7 App.js: App: setEnd

Parameters:

end— end node to use

Callback function to change the end node of the maze when the end node is dragged and dropped to a new location.

2.7.8 App.js: App: setAlgorithm

Parameters:

algorithm— algorithm to use for generating the maze

Callback function to change the algorithm used to generate the maze from the settings component.

2.7.9 App.js: App: setSolve

Parameters:

algorithm— algorithm to use for solving the maze

Callback function to change the algorithm used to solve the maze from the settings component.

2.7.10 App.js: App: async should_solve

Method to check if the maze should be resolved after start or end nodes have moved, and if it should be resolved, then resolve the maze. This method is asynchronous, as it changes the state of the app, which is an asynchronous operation and must be awaited before continuing.

2.7.11 App.js: App: async clear_node_index

Method to clear the index of the nodes so that the maze can be solved again. This method is asynchronous, as it changes the state of the app, which is an asynchronous operation and must be awaited before continuing, and it returns a promise, which is resolved once the state has been updated.

2.7.12 App.js: App: async fetchGrid

Method to fetch the maze from the server. This method is asynchronous, as it changes the state of the app and is making a call to the API over the internet, both of which are asynchronous operations and must be awaited before continuing.

2.7.13 App.js: App: async solveGrid

Method to send the grid to the API to be solved and receive a response with a solved maze. This method is asynchronous, as it changes the state of the app and is making a call to the API over the internet, both of which are asynchronous operations and must be awaited before continuing.

2.7.14 App.js: App: async clearGrid

Method to get an empty grid from the API. This method is asynchronous, as it changes the state of the app and is making a call to the API over the internet, both of which are asynchronous operations and must be awaited before continuing.

2.7.15 App.js: App: render

Method to render the react component.

2.7.16 DisplayGrid.jsx: DisplayGrid: handelDrop

Parameters:

pos— position of the node that is dropped

Callback function to handle the drop of a start or end node.

2.7.17 DisplayGrid.jsx: DisplayGrid: setDragObject

Parameters:

type— type of node that is being dragged

Callback function to set the type of node that is being dragged.

2.7.18 DisplayGrid.jsx: DisplayGrid: renderTable

Method to render the table of the grid.

2.7.19 DisplayGrid.jsx: DisplayGrid: render

Method to render the react component. It will return a table of the grid if there is a grid, else it will show a message saying that there is no grid.

2.7.20 DisplayNode.jsx: DisplayNode: handelDragStart

Callback function to handle the start of the node being dragged. It will set the type of node being dragged to the type of node that is being dragged in the parent component using DisplayGrid.jsx: DisplayGrid: setDragObject.

2.7.21 DisplayNode.jsx: DisplayNode: handelDrop

Callback function to handle the drop of the node.

2.7.22 DisplayNode.jsx: DisplayNode: handelDragOver

Callback function to run when the node is dragged over this node. It will change the colour of the node to indicate that it is being dragged over.

2.7.23 DisplayNode.jsx: DisplayNode: handelDragLeave

Callback function to run when the node is dragged off this node. It will change the colour of the node to indicate that it is no longer being dragged over.

2.7.24 DisplayNode.jsx: DisplayNode: render

Method to render the react component. Selection is used to examine the props and apply the correct attributes to the node.

2.7.25 Settings.jsx: Settings: componentDidMount

Method to run when the component mounts in the DOM. It will add an event listener to the document to detect if the user clicks outside the settings component.

2.7.26 Settings.jsx: Settings: componentWillUnmount

Method to run when the component is unmounted from the DOM. It will remove the event listener from the document.

2.7.27 Settings.jsx: Settings: handelClickOutsize

Parameters:

event— event that is triggered when the user clicks outside the settings component

Callback function to handle the click outside the settings component.

2.7.28 Settings.jsx: Settings: handelSizeChange

Parameters:

event— event that is triggered when the user changes the size of the maze

Callback function to the change of the size of the maze in the parent component using App.jsx: App: setSize.

2.7.29 Settings.jsx: Settings: setHeuristic

Parameters:

heuristic— selected heuristic

Callback function to change the heuristic used for the greedy search in the parent component using App.jsx: App: setHeuristic.

2.7.30 Settings.jsx: Settings: renderSettings

Method to render the settings component if the settings button is clicked.

3 Technical Solution

A running solution of the project is available at <https://nea.olivertemple.dev/>.

3.1 Summary Of Skills Used

These tables are a list of some of the skills I have used in this project, with the location of where they are demonstrated.

Skill	Where to find
Stacks	Solver.py: dfs
Queues	Solver.py: bfs, Solver.py: dijkstra
Priority Queue	Solver.py: greedy
Recursive Algorithms	Generator.py: recursive_backtracking_run
Complex OOP	grid.py: Grid, grid.py: Node, App.js, DisplayGrid.jsx, DisplayNode.jsx, Settings.jsx,
Dynamic generation of objects	Grid.py: Grid: generateGrid
Client-server model, parsing JSON	Called from App.js: 117, App.js: 131, App.js: 151, request handled in lambda_function.py
Dictionaries	lambda_function.py, grid.py: Grid: serialize, grid.py: Node: serialize
Multi-dimensional arrays	grid.py (used to store grid), generator.py: prims (inMaze, frontier)
Simple mathematical calculations	solver.py: manhattan, solver.py: euclidean
Complex Algorithms	solver.py, generator.py
GUI	App.js, DisplayGrid.jsx, DisplayNode.jsx, Footer.jsx, GeneratorInfo.jsx, Menu.jsx, MenuKey.jsx, Settings.jsx, SolverInfo.jsx
Graph traversals	solver.py, generator.py

3.2 The Full Code

3.2.1 Python API: lambda_function.py

```

1 import json
2 from generator import Generator
3 from grid import Grid
4 from solver import Solver
5 def lambda_handler(event, context):
6     #get the parameters from the url query
7     width = event["queryStringParameters"]["width"]
8     height = event["queryStringParameters"]["height"]
9
10    if event["queryStringParameters"]["type"] == "empty_maze":#
11        generate an empty grid
12        myGrid = Grid(int(width), int(height))
13
14    elif event["queryStringParameters"]["type"] == "generate":#
15        generate an empty grid, then create a maze
16        #get the maze generating algorithm
17        generate_algorithm = event["queryStringParameters"]["
18        generate"]
19
20        #create a new generator to generate the maze
21        myGenerator = Generator();
22
23        #create a new grid with the height and width algorithms
24        from the query

```

```

21     myGrid = Grid(int(width),int(height))
22
23     #generate the maze using the algorithm from the query
24     if generate_algorithm == "prims":
25         myGenerator.prims(myGrid)
26     elif generate_algorithm == "recursive_backtracking":
27         myGenerator.recursive_backtracking(myGrid)
28
29     elif event["queryStringParameters"]["type"] == "solve":#solve
30     the maze using requested algorithm
31     #get the grid from the body of the request
32     grid = json.loads(event["body"])
33
34     #generate an empty grid with the same size as the grid from
35     the body
36     myGrid = Grid(int(grid["width"]), int(grid["height"]))
37
38     #load the grid from the body into the empty grid by
39     iterating through the nodes and changing the attributes
40     myGrid.load(grid["grid"])
41
42     #get the solving algorithm from the request
43     solve_algorithm = event["queryStringParameters"]["solve"]
44
45     #get the start and node positions from the request
46     start = eval(event["queryStringParameters"]["start"])
47     end = eval(event["queryStringParameters"]["end"])
48
49     #get the start and end nodes in the grid
50     start_node = myGrid.grid[start[0]][start[1]]
51     end_node = myGrid.grid[end[0]][end[1]]
52
53     #create a new solver to solve the maze
54     mySolver = Solver()
55
56     #solve the maze with the requested algorithm
57     if solve_algorithm == "dijkstra":
58         mySolver.dijkstra(myGrid, start_node, end_node)
59     elif solve_algorithm == "dfs":
60         mySolver.dfs(myGrid, start_node, end_node)
61     elif solve_algorithm == "bfs":
62         mySolver.bfs(myGrid, start_node, end_node)
63     elif solve_algorithm == "greedy":
64         #get the heuristic for greedy from the request
65         heuristic = event["queryStringParameters"]["heuristic"]
66         mySolver.greedy(myGrid, start_node, end_node, heuristic)
67
68 )
69
70 #return the json of the grid
71 return {
72     'statusCode': 200,
73     'body': json.dumps(myGrid.serialize())
74 }
75
76 if __name__ == "__main__":
77     myGrid = Grid(15,15)
78     myGenerator = Generator()

```

```

74     myGenerator.prims(myGrid)
75     mySolver = Solver()
76     mySolver.dijkstra(myGrid, myGrid.grid[0][0], myGrid.grid
       [14][14])
77     myGrid.printGrid()

```

3.2.2 Python API: grid.py

```

1  #Node class for each node in the grid
2  class Node:
3      def __init__(self, x, y, type):
4          self.x = x
5          self.y = y
6          self.type = type
7
8          self.index = None
9          self.parent = None
10         self.visited = False
11
12         self.distance = "infinity"
13
14         self.wallLeft = True
15         self.wallBottom = True
16     def __str__(self):#for development purposes
17         return f"x:{self.x}, y:{self.y}, type:{self.type}, wallLeft
       :{self.wallLeft}, wallBottom:{self.wallBottom}, distance:{self.
       distance}"
18     def load(self, wallLeft, wallBottom):#when loading the grid,
       the walls must be set from the received grid
19         self.wallBottom = wallBottom
20         self.wallLeft = wallLeft
21     def serialize(self):#for returning the grid, the node must be
       serialized into a dictionary
22         return {
23             "x": self.x,
24             "y": self.y,
25             "wallLeft": self.wallLeft,
26             "wallBottom": self.wallBottom,
27             "type": self.type,
28             "index":self.index
29         }
30
31     class Grid:#Grid class for the grid
32     def __init__(self, height, width):
33         self.height = height
34         self.width = width
35         self.grid = self.generateGrid()
36     def load(self, grid):#load the grid from the grid received from
       the react app
37         self.grid = []
38         for row in grid:
39             row_inner = []
40             for item in row:
41                 node = Node(item["x"], item["y"], "space")
42                 node.load(item["wallLeft"], item["wallBottom"])
43                 row_inner.append(node)
44             self.grid.append(row_inner)

```

```

45     def serialize(self):#for returning the grid, the grid must be
46         serialized into a dictionary
47         obj = {
48             "height": self.height,
49             "width": self.width,
50             "grid": []
51         }
52         for row in self.grid:
53             row_inner = []
54             for item in row:
55                 row_inner.append(item.serialize())
56             obj["grid"].append(row_inner)
57
58         return obj
59     def generateGrid(self):#generate the grid
60         grid = []
61         for i in range(self.height):
62             row = []
63             for j in range(self.width):
64                 row.append(Node(j, i, "space"))
65             grid.append(row)
66
67         row = []
68         for j in range(self.width + 1):
69             row.append(Node(j, i, "space"))
70         grid.append(row)
71
72         return grid
73     def printGrid(self):#for development purposes
74         width = self.width
75         height = self.height
76
77         print(" _"*width)
78
79         for i in range(height):
80             for j in range(width):
81                 cell = ""
82                 if self.grid[i][j].wallLeft:
83                     cell += "|"
84                 else:
85                     cell += " "
86                 if self.grid[i][j].type == "path":
87                     cell += "XX"
88                 else:
89                     cell += " "
90             print(cell, end="")
91         print("|")
92
93         for j in range(width):
94             cell = ""
95             if self.grid[i][j].wallLeft:
96                 cell += "|"
97             else:
98                 cell += " "
99
100             if self.grid[i][j].wallBottom:

```

```

101         cell += "--"
102     else:
103         cell += " "
104     print(cell, end="")
105     print("|")

```

3.2.3 Python API: generator.py

```

1  import random
2  import sys
3
4  class Generator:
5      def __init__(self):
6          pass
7      #setup for recursive backtracking
8      def recursive_backtracking(self, Grid):
9          sys.setrecursionlimit(2000) #set the recursion limit to 2000
10         #create a list of all nodes in maze
11         unvisited = []
12         for row in Grid.grid:
13             for item in row:
14                 unvisited.append(item)
15
16         #pick a random start node
17         start = random.choice(unvisited)
18         unvisited.remove(start)
19
20         #start recursive backtracking
21         self.recursive_backtracking_run(Grid, unvisited, start, [])
22
23     #recursive backtracking algorithm
24     def recursive_backtracking_run(self, Grid, unvisited, current,
25     previous):
26         #work out which walls can be removed
27         orientation_options = []
28         if current.x > 0 and Grid.grid[current.y][current.x-1] in
29         unvisited:
30             orientation_options.append("left")
31         if current.y > 0 and Grid.grid[current.y-1][current.x] in
32         unvisited:
33             orientation_options.append("top")
34         if current.x < Grid.width - 1 and Grid.grid[current.y][
35         current.x+1] in unvisited:
36             orientation_options.append("right")
37         if current.y < Grid.height - 1 and Grid.grid[current.y+1][
38         current.x] in unvisited:
39             orientation_options.append("bottom")
40
41         #if there are no walls to remove, backtrack, else pick one
42         #and remove it
43         if len(orientation_options) > 0:
44             #pick a random wall to remove
45             orientation = random.choice(orientation_options)
46             #add the current node to the previous nodes list
47             previous.append(current)
48
49             #remove the wall depending on the orientation

```

```

44         if orientation == "left":
45             connecting_cell = Grid.grid[current.y][current.x -
1]
46                 current.wallLeft = False
47
48         elif orientation == "bottom":
49             connecting_cell = Grid.grid[current.y + 1][current.
x]
50                 current.wallBottom = False
51
52         elif orientation == "right":
53             connecting_cell = Grid.grid[current.y][current.x +
1]
54                 connecting_cell.wallLeft = False
55
56         elif orientation == "top":
57             connecting_cell = Grid.grid[current.y - 1][current.
x]
58                 connecting_cell.wallBottom = False
59
60         #remove the connecting node from the unvisited list
61         unvisited.remove(connecting_cell)
62
63         #recurse
64         self.recursive_backtracking_run(Grid, unvisited,
connecting_cell, previous)
65     else:
66         #if not back at the start, backtrack
67         if len(previous) > 0:
68             new = previous.pop()
69             self.recursive_backtracking_run(Grid, unvisited,
new, previous)
70
71     #generate a maze using prims algorithm
72     def prims(self, Grid):
73         #start at the top left node
74         inMaze = [[0, 0]]
75         #create a list of nodes connected to the start node
76         frontier = [[[0, 1], ["left"]], [[1, 0], ["top"]]]
77
78         #while there are still nodes to visit
79         while (len(frontier) > 0):
80             #pick a random node from the frontier
81             new = frontier.pop(random.randint(0, len(frontier)-1))
82
83             toAdd = new[0]
84
85             #pick a random wall from the available walls
86             wall = new[1][random.randint(0, len(new[1])-1)]
87
88             #add the wall to the inMaze list
89             inMaze.append(toAdd)
90
91             #remove the selected wall from the grid
92             if wall == "bottom":
93                 Grid.grid[toAdd[0]][toAdd[1]].wallBottom = False
94

```



```

95         if wall == "left":
96             Grid.grid[toAdd[0]][toAdd[1]].wallLeft = False
97
98         if wall == "top" and toAdd[0] > 0:
99             Grid.grid[toAdd[0]-1][toAdd[1]].wallBottom = False
100
101         if wall == "right" and toAdd[1] < Grid.width:
102             Grid.grid[toAdd[0]][toAdd[1]+1].wallLeft = False
103
104         #calculate the possible nodes to add to the frontier
105         possible = [[toAdd[0]-1, toAdd[1]], [toAdd[0]+1, toAdd
106                     [1]], [toAdd[0], toAdd[1]-1], [toAdd[0], toAdd[1]+1]]
107
108         #possible walls
109         walls = ["bottom", "top", "right", "left"]
110         #iterate through the possible nodes
111         for i in range(4):
112             p = possible[i]
113             wall = walls[i]
114             #check that the wall can be removed
115             if 0<=p[0]<Grid.height and 0<=p[1]<Grid.width:
116                 #check that the node is not already in the maze
117                 if p not in inMaze:
118                     found = False
119                     #check if the node is already in the
120                     frontier, if it is then add the wall to the possible walls for
121                     the node in the frontier
122                     for v in frontier:
123                         if v[0]==p:
124                             v[1].append(wall)
125                             found = True
126                             #if the node is not in the frontier, add it
127                             to the frontier
128                     if not found:
129                         frontier.append([p,[wall]])

```

3.2.4 Python API: solver.py

```

1  import math
2  class Solver:
3      def __init__(self):
4          pass
5
6      def get_adjacent_paths(self, Grid, node):#Get adjacent nodes in
7          the grid that are not blocked by a wall
8          paths = []
9          if node.x > 0 and not node.wallLeft: #check that the node
10             is not on the left edge and that it doesn't have a wall on the
11             left
12             paths.append(Grid.grid[node.y][node.x - 1])
13             if node.x < Grid.width - 1 and not Grid.grid[node.y][node.x
14                 + 1].wallLeft: #check that the node is not on the right edge,
15                 and that there is not a wall to the right of it
16                 paths.append(Grid.grid[node.y][node.x + 1])
17             if node.y > 0 and not Grid.grid[node.y - 1][node.x].
18                 wallBottom: #check that the node is not at the top and that
19                 there isn't a wall above it

```

```

13         paths.append(Grid.grid[node.y - 1][node.x])
14         if node.y < Grid.height - 1 and not node.wallBottom:#check
that the node is not at the bottom and that there is no wall
below it
15             paths.append(Grid.grid[node.y + 1][node.x])
16
17         #retrun a list of the available nodes
18         return paths
19
20     def dijkstra(self, Grid, start, end):#Dijkstra's algorithm for
solving the grid
21         #set the distance from the start node to the start node to
0
22         start.distance = 0
23         #set the start index as 0
24         #The index is a number that shows when that node was
visited by the solving algorithm, so that when the algorithm is
visualized, the react app can show in what order the nodes
were visited.
25         index = 0
26         #create a queue of unvisited nodes
27         unvisited = [start]
28         #create a flag
29         found = False
30         while not found:
31             #take the next node from the front of the queue
32             current = unvisited.pop(0)
33             #mark the node as visited
34             current.visited = True
35             #set the index on the node
36             current.index = index
37             #increment the index
38             index += 1
39             #iterate through the available adjacent nodes
40             for node in self.get_adjacent_paths(Grid, current):
41                 #if the node has not been visited already, set the
distance from the start node to be one more than the distance
of the current node and add it to the queue of unvisited nodes
42                 if not node.visited:
43                     node.distance = current.distance + 1
44                     unvisited.append(node)
45
46             #if the node is the end node, update the flag and
exit the algorithm
47             if node == end:
48                 found = True
49                 break
50
51         #backtrack from the end node to the start node, picking the
node with the smallest distance at every point
52         current = end
53         path = [end]
54         while current != start:
55             #get connecting cells
56             connecting = self.get_adjacent_paths(Grid, current)
57             #work out which node as the lowest distance
58             min = None

```

```

59         for node in connecting:
60             if node.distance != "infinity":
61                 if min == None or node.distance < min.distance:
62                     min = node
63             #append the node with the lowest distance to the path
64             path.append(min)
65             #update the current node
66             current = min
67
68         #draw the path
69         for node in path:
70             node.type = "path"
71
72     def dfs(self, Grid, start, end):
73         #create a stack of nodes to visit
74         stack = [start]
75         #set the start index as 0
76         #The index is a number that shows when that node was
77         #visited by the solving algorithm, so that when the algorithm is
78         #visualized, the react app can show in what order the nodes
79         #were visited.
80         start.index = 0
81         index = 1
82         #while the stack is not empty, keep searching
83         while len(stack) > 0:
84             #get the item at the top of the stack
85             current = stack[-1]
86             #mark the item as visited
87             current.visited = True
88             #check if the end has been found
89             if current == end:
90                 break
91
92             #find connecting nodes that have not been visited
93             possible = []
94             for node in self.get_adjacent_paths(Grid, current):
95                 if not node.visited:
96                     possible.append(node)
97
98             #if there are possible connections, choose the first
99             one
100             if len(possible) > 0:
101                 to_append = possible[0]
102                 #set the index
103                 to_append.index = index
104                 #set the parent node for drawing the path
105                 to_append.parent = current
106                 #increment the index
107                 index += 1
108                 #add the new node to the stack
109                 stack.append(to_append)
110             #If there are no possible connections, remove the
111             #current node from the stack
112             else:
113                 stack.pop()
114
115         #backtrack from the end to the start drawing the path

```

```

111         while current != start:
112             current.type = "path"
113             current = current.parent
114
115         start.type = "path"
116
117     def bfs(self, Grid, start, end):
118         #create a queue of nodes that need to be visited
119         queue = [start]
120         #set the start index as 0
121         #The index is a number that shows when that node was
        visited by the solving algorithm, so that when the algorithm is
        visualized, the react app can show in what order the nodes
        were visited.
122         start.index = 0
123         index = 1
124         #only run the algorithm while there are nodes to visit
125         while len(queue) > 0:
126             #get the first item in the queue and mark as visited
127             current = queue.pop(0)
128             current.visited = True
129
130             #check to see if the end node has been found
131             if current == end:
132                 break
133
134             #add unvisited adjacent nodes to the queue
135             for node in self.get_adjacent_paths(Grid, current):
136                 if not node.visited:
137                     #update nodes parent for drawing path
138                     node.parent = current
139                     #add index
140                     node.index = index
141                     #increment the index
142                     index += 1
143                     #append node to queue
144                     queue.append(node)
145
146             #backtrack from the end to the start and draw the path
147             while current != start:
148                 current.type = "path"
149                 current = current.parent
150
151             start.type = "path"
152
153     def manhattan(self, node1, node2):#calculate the manhattan
        distance between two nodes
154         return abs(node1.x - node2.x) + abs(node1.y - node2.y)
155
156     def euclidean(self, node1, node2):#calculate the euclidean
        distance between two nodes
157         return math.sqrt((node1.x - node2.x)**2 + (node1.y - node2.
        y)**2)
158
159     def greedy(self, Grid, start, end, heuristic):
160         #create a priority queue for nodes that need to be visited
161         queue = [start]

```

```

162     #set the start index as 0
163     #The index is a number that shows when that node was
    visited by the solving algorithm, so that when the algorithm is
    visualized, the react app can show in what order the nodes
    were visited.
164     start.index = 0
165     index = 1
166     #while there are nodes to visit
167     while len(queue) > 0:
168         #pop the node off of the front of the queue and mark as
    visited
169         current = queue.pop(0)
170         current.visited = True
171
172         #check if the end node has been found
173         if current == end:
174             break
175
176         #iterate through unvisited adjacent nodes
177         for node in self.get_adjacent_paths(Grid, current):
178             if not node.visited:
179                 #update the parent of the node
180                 node.parent = current
181                 #update the index
182                 node.index = index
183                 #increment the index
184                 index += 1
185                 #use the selected heuristic to update the
    distance
186
187                 if heuristic == "manhattan":
188                     node.distance = self.manhattan(node, end)
189                 elif heuristic == "euclidean":
190                     node.distance = self.euclidean(node, end)
191                 #insert node in the priority queue
192                 for item in queue:
193                     if item.distance > node.distance:
194                         queue.insert(queue.index(item), node)
195                         break
196                 queue.append(node)
197
198         #backtrack from the start and draw the path
199         while current != start:
200             current.type = "path"
201             current = current.parent
202
203         start.type = "path"

```

3.2.5 React App: App.js

```

1 import './App.css';
2 import { Component } from 'react';
3 import DisplayGrid from './components/DisplayGrid';
4 import Menu from './components/Menu';
5 import MenuKey from './components/MenuKey';
6 import Footer from './components/Footer';
7 class App extends Component {
8   constructor(props){

```

```

 9   super(props);
10   this.state = {
11     grid: null, //the grid of nodes
12     algorithm: null, //algorithm for generating the maze
13     solve: null, //algorithm for solving the maze
14     nodes: {
15       start: [0,0], //position of the start node
16       end: [null, null] //position of the end node
17     },
18     size: { //size of the maze
19       width: 15,
20       height: 15
21     },
22     heuristic: "euclidean",
23     speed: 0.1
24   }
25
26   this.solved = false;
27   this.maze = false;
28   this.api_endpoint = "https://jkrlv64tsl.execute-api.eu-west-2.
    amazonaws.com/default/NEA"
29   //bind the methods to the object so that the "this" keyword
    refers to the object no matter where the method is called from
30   this.fetchGrid = this.fetchGrid.bind(this);
31   this.setAlgorithm = this.setAlgorithm.bind(this);
32   this.clearGrid = this.clearGrid.bind(this);
33   this.setSolve = this.setSolve.bind(this);
34   this.solveGrid = this.solveGrid.bind(this);
35   this.setSize = this.setSize.bind(this);
36   this.setStart = this.setStart.bind(this);
37   this.setEnd = this.setEnd.bind(this);
38   this.should_solve = this.should_solve.bind(this);
39   this.setHeuristic = this.setHeuristic.bind(this);
40   this.setSpeed = this.setSpeed.bind(this);
41 }
42
43 componentDidMount(){
44   //generate a new maze empty when the page loads
45   this.clearGrid();
46 }
47
48 setHeuristic(heuristic){ //set the heuristic for the greedy
    algorithm
49   this.setState({heuristic: heuristic});
50 }
51 setSpeed(speed){ //set the speed of the animation
52   this.clearGrid();
53   this.setState({speed: speed});
54 }
55 setSize(size){ //set the size of the grid when changed in settings
56   if(size.width > 30 && size.height > 30){ //if the size is too
    large, alert the user
57     alert("The size of the maze must be less than 30.")
58     return;
59   }
60   this.setState({
61     size: size

```

```

62   }, () => { //setState is asynchronous, so we need to wait for it
        to finish before running the following code
63   if (size.width > 0 && size.height > 0 && size.width < 31 && size
        .height < 31){ //if the size is valid, generate a new maze
64       this.clearGrid();
65   }
66   })
67   }
68   setStart(node){ //set the start node
69       this.setState({
70           nodes:{
71               start: node,
72               end: this.state.nodes.end
73           }
74       }, () => { //setState is asynchronous, so we need to wait for it
        to finish before running the following code
75       this.should_solve() //solve the maze again if it is already
        solved
76       })
77   }
78   setEnd(node){ //set the end node
79       this.setState({
80           nodes:{
81               start: this.state.nodes.start,
82               end: node
83           }
84       }, () => { //setState is asynchronous, so we need to wait for it
        to finish before running the following code
85       this.should_solve() //solve the maze again if it is already
        solved
86       })
87   }
88
89   async should_solve(){ //if the maze is already solved, then solve
        again. Only run when the start or end nodes are changed
90   if (this.solved){
91       this.clear_node_index() //clear the index of the nodes, as the
        maze is being solved again
92       .then(() => {
93           this.solveGrid();
94       })
95   }
96   }
97
98   async clear_node_index(){ //clear the index of the nodes so that
        the maze can be solved again
99   return new Promise(resolve => { //since there are asynchronous
        calls, we need to wait for them to finish before running the
        code after the clear_node_index function, hence we use a
        promise that is resolved once this code is finished
100   let grid = this.state.grid.grid
101   //iterate through the grid and clear the index of the nodes
102   for (let i=0; i<this.state.size.height; i++){
103       for (let j=0; j<this.state.size.width; j++){
104           grid[i][j].index = null;
105       }
106   }

```

```

107 //update the state with the grid that has been cleared of the
    index's
108 this.setState({
109   grid: {
110     grid: grid,
111     height: this.state.grid.height,
112     width: this.state.grid.width
113   }
114 }, () => {
115   resolve(); // resolve the promise once the state has been
    updated
116 })
117 })
118
119 }
120 async fetchGrid(){//generate a new maze from the python API using
    the selected algorithm
121 if (this.state.algorithm){//check that there is an algorithm
    selected for generating the maze
122 let grid = await fetch(`${this.api_endpoint}?type=generate&width=
    ${this.state.size.width}&height=${this.state.size.height}&
    generate=${this.state.algorithm}`)//fetch the generated grid
    from the python API
123 grid = await grid.json();//convert the response to json
124 this.setState{//update the state with the new grid
125   grid:grid
126 }
127 this.solved = false;//the maze is no longer solved
128 this.maze = true;//set the maze to true as a maze has been
    generated
129 }else{//If there is no algorithm selected to generate the maze,
    alert the user
130 alert("Please select a maze generating algorithm")
131 }
132 }
133 async solveGrid(){//send the maze to the python API to be solved
    with the requested algorithm
134 await this.clear_node_index();//clear the index of the nodes, as
    the maze is being solved again
135
136 if (!this.maze || !this.state.solve){
137   if (!this.maze){
138     alert("Please generate a maze before solving it")
139   }else{
140     alert("Please select a maze solving algorithm")
141   }
142   return
143 }
144 let grid = await fetch(
145   `${this.api_endpoint}?type=solve&width=${this.state.size.width}&
    height=${this.state.size.height}&solve=${this.state.solve}&
    start=${this.state.nodes.start}&end=${this.state.nodes.end}&
    heuristic=${this.state.heuristic}`, {
146   method: "POST",
147   body: JSON.stringify(this.state.grid)//set the body of the
    request to the grid
148 })//send the maze to the python API to be solved, with the

```



```

        selected algorithm as a parameter
149   grid = await grid.json();//convert the response to json
150   this.setState({//update the state with the new grid
151     grid:grid
152   })
153   this.solved = true;//the maze is now solved
154
155 }
156 async clearGrid(){//generate an empty maze from the API
157   let grid = await fetch('https://jkr1v64tsl.execute-api.eu-west-2.
        amazonaws.com/default/NEA?type=empty_maze&width=${this.state.
        size.width}&height=${this.state.size.height}')//fetch the empty
        grid from the python API
158   grid = await grid.json();//convert the response to json
159   this.setState({//update the state
160     grid:grid,//update the state with the new grid
161     nodes:{//set the start and end nodes to default positions
162       start: [0,0],
163       end: [this.state.size.height - 1, this.state.size.width - 1]
164     }
165   })
166   this.maze = false;//there is no longer a maze to solve
167   this.solved = false;//the maze is no longer solved
168 }
169 setAlgorithm(algorithm){//set the maze generating algorithm
170   this.setState({
171     algorithm: algorithm
172   })
173 }
174 setSolve(algorithm){//set the maze solving algorithm
175   this.setState({
176     solve:algorithm
177   })
178 }
179 render(){
180   return (
181     <div className="App">
182       <Menu
183         setAlgorithm={this.setAlgorithm}//callback function to set the
        generation algorithm from the menu
184         setSolve={this.setSolve}//callback function to set the solving
        algorithm from the menu
185         generate={this.fetchGrid}//callback function to generate a new
        maze from the menu
186         clearGrid={this.clearGrid}//callback function to clear the
        maze from the menu
187         solve={this.solveGrid}//callback function to solve the maze
        from the menu
188         size={this.state.size}//the size of the maze
189         setSize={this.setSolve}//callback function to set the size of
        the maze from the menu
190         setHeuristic={this.setHeuristic}//callback function to set the
        heuristic from the menu
191         setSpeed={this.setSpeed}//callback function to set the speed
        from the menu
192         speed={this.state.speed}//the speed of the maze
193       />

```

```

194   <MenuKey />
195   <DisplayGrid
196     grid={this.state.grid}//the grid of the maze
197     nodes={this.state.nodes}//the start and end nodes
198     size={this.state.size}//the size of the maze
199     setStart={this.setStart}//callback function to set the start
node
200     setEnd={this.setEnd}//callback function to set the end node
201     generateAlgorithm={this.state.algorithm}//the algorithm used
to generate the maze
202     solveAlgorithm={this.state.solve}//the algorithm used to solve
the maze
203     heuristic={this.state.heuristic}//the heuristic used for the
greedy algorithm
204     speed={this.state.speed}//the speed of the animation
205   />
206   <Footer />
207 </div>
208 );
209 }
210 }
211 export default App;

```

3.2.6 React App: Menu.jsx

```

1  import React from 'react';
2  import Settings from './Settings';
3  export default function Menu(props) { //Menu bar for the app
4    return(
5      <div className="menu">
6        <Settings
7          size={props.size}//size of the maze
8          setSize={props.setSize}//callback to set the size of the
maze from the settings
9          setHeuristic={props.setHeuristic}//callback to set the
heuristic from the settings
10         setSpeed={props.setSpeed}//callback to set the speed from
the settings
11         speed={props.speed}//speed of the animation
12       />
13       <select className="algorithms" name="algorithms" id="
algorithms" onChange={(e) => {props.setAlgorithm(e.target.value
)}}>
14         <option value="select">Select Generating Algorithm</
option>
15         <option value="prims">Prims</option>
16         <option value="recursive_backtracking">recursive
backtracking</option>
17       </select>
18       <button className="button" onClick={props.generate}>
Generate</button>
19       <select className="algorithms" name="algorithms" id="
algorithms" onChange={(e) => {props.setSolve(e.target.value)}}>
20         <option value="select">Select Solving Algorithm</option>
21         <option value="dijkstra">Dijkstra</option>
22         <option value="dfs">Depth First Search</option>
23         <option value="bfs">Breadth First Search</option>

```

```

24         <option value="greedy">Greedy</option>
25     </select>
26     <button className="button" onClick={props.solve}>Solve</
button>
27     <button className="button clear" onClick={props.clearGrid}>
Reset</button>
28 </div>
29 )
30 }

```

3.2.7 React App: MenuKey.jsx

```

1 import React from "react";
2
3 export default function MenuKey(props){//key for showing the
different types of node
4     return(
5         <div className="key">
6             <div className="key_item">
7                 <div className="key_node_start"></div>
8                 <p>Start Node</p>
9             </div>
10            <div className="key_item">
11                <div className="key_node_end"></div>
12                <p>Finish Node</p>
13            </div>
14            <div className="key_item">
15                <div className="key_node_path"></div>
16                <p>Path Node</p>
17            </div>
18            <div className="key_item">
19                <div className="key_node_visited_node"></div>
20                <p>Visited Node</p>
21            </div>
22        </div>
23    )
24 }

```

3.2.8 React App: DisplayGrid.jsx

```

1 import React, { Component } from "react";
2 import DisplayNode from "./DisplayNode";
3 import GeneratorInfo from "./GeneratorInfo";
4 import SolverInfo from "./SolverInfo";
5 export default class DisplayGrid extends Component{
6     constructor(props){
7         super(props);
8         this.state = {
9             dragObject: ""
10        }
11        //bind the methods to the object so that the "this" keyword
refers to the object no matter where the method is called from
12        this.renderTable = this.renderTable.bind(this);
13        this.handelDrop = this.handelDrop.bind(this);
14        this.setDragObject = this.setDragObject.bind(this);
15    }

```

```

16  handelDrop(pos){//move the node that was being dragged to the new
    position
17  switch (this.state.dragObject){
18      case "start":
19          this.props.setStart(pos)
20          break;
21      case "end":
22          this.props.setEnd(pos)
23          break;
24      default:
25          break;
26  }
27  }
28  setDragObject(type){//set weather start or end node is being
    dragged
29  this.setState({
30      dragObject:type
31  })
32  }
33  renderTable(){//render the grid as a table
34      return(
35          <table>
36              <tbody className="column">
37                  {Array.from(Array(this.props.grid.height).keys()).map((_,
38                      i) => {//iterate through the rows of the grid
39                      return(
40                          <tr className={`row wall_right ${i === 0 ? "wall_top"
41                          : ""}` key={i}>
42                              {Array.from(Array(this.props.grid.width).keys()).
43                                  map((_, j) => {//iterate through the nodes in each row
44                                  return(
45                                  <DisplayNode
46                                  key={j}
47                                  wallLeft={this.props.grid.grid[i][j].wallLeft
48                                  } //bool: is there a wall to the left of this node
49                                  wallBottom={this.props.grid.grid[i][j].
50                                  wallBottom} //bool: is there a wall below this node
51                                  pos={[i, j]} //position of the node
52                                  start={this.props.nodes.start} //position of
53                                  the start node
54                                  end={this.props.nodes.end} //position of the
55                                  end node
56                                  handelDrop = {this.handelDrop} //callback
57                                  function to move the start or end node to a new position
58                                  setDragObject={this.setDragObject} //
59                                  callback function to set weather the start or end node is being
60                                  dragged
61                                  type={this.props.grid.grid[i][j].type} //type
62                                  of node
63                                  index={this.props.grid.grid[i][j].index} //
64                                  index of the node for visualization
65                                  speed={this.props.speed} //speed of the
66                                  animation
67                                  height={this.props.grid.height}
68                                  width={this.props.grid.width}
69                                  />
70                                  )
71                      }
72                  }
73              }
74          )

```

```

58         }}}
59       </tr>
60     )
61   }}
62 </tbody>
63 </table>
64 )
65
66 }
67 render(){
68   //If there is a grid, render it, else show a message
69   if (this.props.grid){
70     return(
71       <div className="grid" style={{padding:10}}>
72         <GeneratorInfo generator={this.props.generateAlgorithm}/>
73         <this.renderTable />
74         <SolverInfo solver={this.props.solveAlgorithm} heuristic
75         ={this.props.heuristic}/>
76       </div>
77     )
78   }
79   return(
80     <div className="grid message column">
81       <h1>No grid to display</h1>
82       <h2>Check your internet connection</h2>
83     </div>
84   )
85 }

```

3.2.9 React App: DisplayNode.jsx

```

1 import React from "react"
2 export default class DisplayNode extends React.Component{
3   constructor(props){
4     super(props)
5     this.state = {
6       style:{height:600/this.props.height, width:600/this.props.
7       width}
8     }
9     //bind the methods to the object so that the "this" keyword
10    refers to the object no matter where the method is called from
11    this.handelDragStart = this.handelDragStart.bind(this);
12    this.handelDragLeave = this.handelDragLeave.bind(this);
13    this.handelDragOver = this.handelDragOver.bind(this);
14    this.handelDrop = this.handelDrop.bind(this);
15  }
16  handelDragStart(){//set the type of node that is being dragged
17    this.props.setDragObject(this.start ? "start" : this.end ? "end
18    " : "")
19  }
20  handelDrop(){//move the node that was being dragged to the new
21    position
22    this.setState({style:{height:600/this.props.height, width:600/
23    this.props.width}});
24    this.props.handelDrop(this.props.pos)
25  }

```

```

21  handelDragOver(e){//when another node is dragged over this node,
    set the style of the node to be pink
22    e.preventDefault();
23    this.setState({
24      style:{
25        backgroundColor:"pink",
26        height:600/this.props.height,
27        width:600/this.props.width
28      }
29    })
30  }
31  handelDragLeave(){//remove the pink style when the node is no
    longer being dragged over
32    this.setState({
33      style:{height:600/this.props.height, width:600/this.props.
width}
34    })
35  }
36  render(){//render the node as a table cell
    //generate a list of css classes for this node
37    this.classList = ["node"];
38    //set default values for the node
39    this.draggable = false;
40    this.start = false;
41    this.end = false;
42    //add walls to the node classList
43    if (this.props.wallLeft){
44      this.classList.push("wall_left")
45    }
46    if (this.props.wallBottom){
47      this.classList.push("wall_bottom")
48    }
49    //add path to node classList
50    if (this.props.type === "path"){
51      this.classList.push("node_path")
52    }else{
53      //Remove the "node_path" item from the classList if it isn't
54      a path, as when maze is resolved the nodes would remain a path
      node if it was a path node before.
55      this.classList.filter(x => {return x !== "node_path"})
56    }
57    //add attributes for the start node or remove them if this node
    is no longer the start node
58    if (this.props.pos[0] === this.props.start[0] && this.props.pos
[1] === this.props.start[1]){
59      this.classList.push("node_start")
60      this.draggable = true
61      this.start = true
62    }else{
63      this.classList.filter(x => {return x !== "node_start"})
64      this.start = false
65    }
66    //add attributes for the end node or remove them if this node
    is no longer the end node
67    if (this.props.pos[0] === this.props.end[0] && this.props.pos
[1] === this.props.end[1]){
68      this.classList.push("node_end")

```

```

69     this.draggable = true
70     this.end=true
71   }else{
72     this.classList.filter(x => {return x !== "node_end"})
73     this.end=false
74   }
75
76   if(this.props.index){//Each node is given an index when it is
77   visited so the order of the visited nodes can be visualized
78     if (this.props.type !== "path"){//Add css animations for to
79     show the visited nodes
80       this.state.style = {
81         animation: 'visit_node 2s linear forwards',
82         animationDelay: `${this.props.index*this.props.speed}s`,
83         height:600/this.props.height,
84         width:600/this.props.width
85       }
86     }else{//Add css animations for to show the path nodes
87       this.state.style = {
88         animation: "visit_node_path 2s linear forwards",
89         animationDelay: `${this.props.index*this.props.speed}s`,
90         height:600/this.props.height,
91         width:600/this.props.width
92       }
93     }
94   }else{
95     //removes the colour if the node is no longer visited after
96     the maze is solved again
97     if (!this.state.style.backgroundColor){
98       this.state.style = {
99         height:600/this.props.height,
100         width:600/this.props.width
101       }
102     }
103   }
104   return(
105     <td style={this.state.style} className={this.classList.join("
106     ")} draggable={this.draggable} onDragStart={this.
107     handelDragStart} onDrop={this.handelDrop} onDragOver={this.
108     handelDragOver} onDragLeave={this.handelDragLeave}>
109   </td>
110   )
111 }
112 }

```

4 Testing

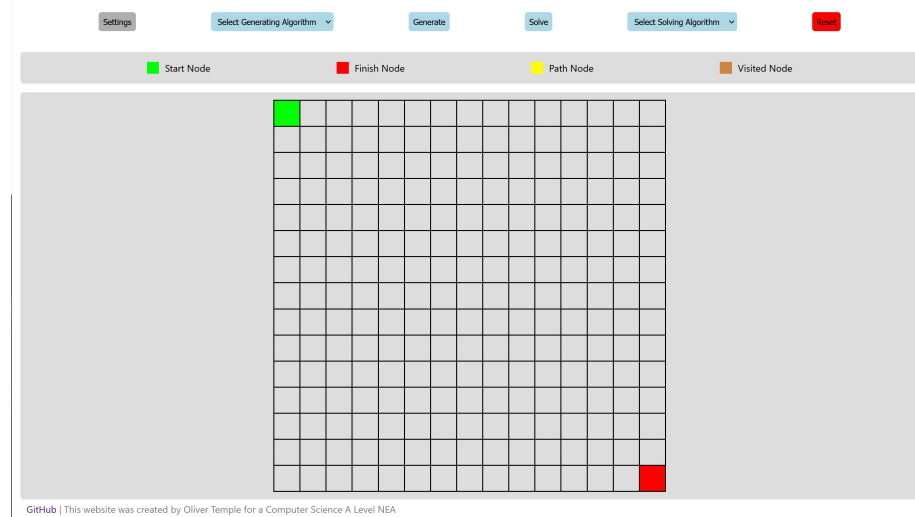
To test the project as a whole, I will check the following:

- Test1. Load the react app and check that an empty maze is generated correctly.
- Test2. Change the size of the maze and check that the size changes and doesn't go below 1 or above 30.
- Test3. Run each of the maze generation algorithms and check that the maze is generated correctly and is solvable.

- Test3.1. Run Prim's algorithm for a range of different sized mazes.
- Test3.2. Run recursive backtracking algorithm for a range of different sized mazes.
- Test4. Solve an assortment of mazes with assorted sizes using each of the algorithms (including heuristics for the greedy search) and check that they run correctly.
 - Test4.1. Solve a variety of mazes of various sizes that have been generated with each algorithm with Dijkstra's algorithm.
 - Test4.2. Solve a variety of mazes of various sizes that have been generated with each algorithm with depth first search algorithm.
 - Test4.3. Solve a variety of mazes of various sizes that have been generated with each algorithm with breadth first search algorithm.
 - Test4.4. Solve a variety of mazes of various sizes that have been generated with each algorithm with Greedy Manhattan algorithm.
 - Test4.5. Solve a variety of mazes of various sizes that have been generated with each algorithm with Greedy Euclidean algorithm.
- Test5. Use different speeds when solving the mazes and check that the speed changes correctly.
- Test6. Reset the maze and check that the maze is reset correctly.
- Test7. Drag and drop the start and finish nodes. Check that the maze resolves correctly.
- Test 8. Try to generate the maze without a generation algorithm selected and check that an error is thrown.
- Test 9. Try to solve the maze without a maze to solve and check that an error is thrown.
- Test 10. Try to solve the maze without a solver algorithm selected and check that an error is thrown.

Test1: Pass

Upon loading the react app, an empty maze is generated.



Test2: Pass

<https://www.youtube.com/watch?v=CAwgwo0V2gg> As seen in the video, the size of the maze can be changed, however it cannot go below 1 or above 30. If the size of the maze is set above 30, a message is displayed and the size is incremented, however the size of the maze does not change, and remains at 30 until the size is reduced to 30 or below.

Test3: Pass

Test 3.1: Pass

https://youtu.be/iKEupya0_jk As seen in the video, the maze is generated correctly using prim's algorithm for a range of different sized mazes. All of these mazes are solvable. The requirements for this test have been met.

Test 3.2: Pass

<https://youtu.be/JaqfMnUv05w> As seen in the video, the maze is generated correctly using recursive backtracking algorithm for a range of different sized mazes. All of these mazes are solvable. The requirements for this test have been met.

Test4

Test 4.1: Pass

<https://youtu.be/SU7yRI-M01E> As seen in the video, the maze is solved correctly using Dijkstra's algorithm when the maze is generated with both prims and recursive backtracking algorithms as well as with varying sizes of maze. The requirements for this test have been met.

Test 4.2: Pass

https://youtu.be/6t0w_TfPZv8 As seen in the video, the maze is solved correctly using depth first search algorithm when the maze is generated with both prims and recursive backtracking algorithms as well as with varying sizes of maze. The requirements for this test have been met.

Test 4.3: Pass

<https://youtu.be/wlYep7ihSd8> As seen in the video, the maze is solved correctly using breadth first search algorithm when the maze is generated with both prims and recursive backtracking algorithms as well as with varying sizes of maze. The requirements for this test have been met.

Test 4.4: Pass

<https://youtu.be/iRUjr1lZdyY> As seen in the video, the maze is solved correctly using Greedy algorithm with the manhattan heuristic when the maze is generated with both prims and recursive backtracking algorithms as well as with varying sizes of maze. The requirements for this test have been met.

Test 4.5: Pass

<https://youtu.be/VE-0nMJJ1DA> As seen in the video, the maze is solved correctly using Greedy algorithm with the euclidean heuristic when the maze is generated with both prims and recursive backtracking algorithms as well as with varying sizes of maze. The requirements for this test have been met.

Test5: Pass

https://youtu.be/NsGNPOV_UTQ As shown in the video, the slider can be used to change the speed that the visualization runs. A lower value of the slider will lower the delay between the nodes being visited, hence speeding it up. A higher value of the slider will increase the delay between the nodes being visited, hence slowing it down. The requirements for this test have been met.

Test6: Pass

<https://youtu.be/kffa8GjnVL4> When the reset button is pressed, an empty grid of the current size is generated.

Test7: Pass

<https://youtu.be/SOVFP3Kz01c> As shown in the video, when the start and end nodes are dragged and dropped, the maze is solved correctly from the new start and end nodes. If the maze has been solved and either node is moved, then the maze with solve again. The requirements for this test have been met.

4.1 Test 8: Pass

https://youtu.be/3qMWXevVs_s As seen in the video, when the generation algorithm is not selected, an error is thrown in the form of an alert from the website, informing the user that a maze generation algorithm must first be chosen. The requirements for this test have been met.

4.2 Test 9: Pass

https://youtu.be/_RUYGKPVKYk As seen in the video, when there is no maze to solve, an error is thrown in the form of an alert from the website, informing the user that a maze must be generated before a solving algorithm can be used. The requirements for this test have been met.

4.3 Test 10: Pass

<https://youtu.be/JKMkkur1sIo> As seen in the video, when no maze solving algorithm is selected, an error is thrown in the form of an alert from the website, informing the user that a maze solving algorithm must be selected before the maze can be solved. The requirements for this test have been met.

5 Evidence of Completeness

5.1 Objectives Again

5.1.1 Generate Mazes

The website should be able to generate mazes using multiple algorithms, including, but not limited to: Prim's algorithm and recursive backtracking. There should also be a brief description of the algorithm that has been selected.

5.1.2 Solve Mazes

The website should be able to solve mazes using multiple algorithms, including, but not limited to: greedy search, Dijkstra's algorithms, depth-first search and breadth-first search. There should also be a brief description of the algorithm that has been selected.

5.1.3 Customisation

The user should be able to customize aspects of the visualization, including:

- Size of the maze.
- Speed of the animation.
- The heuristic used in any heuristic algorithms.

5.1.4 User written algorithms

The website should be able to run algorithms written by the user for both maze generation and solving. This will be done by supplying documentation on what parameters need to be taken in and what will need to be returned from the function for the visualizer to work.

5.1.5 Update Visualization

If the start or end nodes are moved once the visualization has been run, then it should update without the user having to rerun the visualization.

5.2 Evaluation of Objectives

5.2.1 Generate Mazes

The project is able to generate mazes using multiple algorithms, as stated in the objective above. The available algorithms for generating mazes are:

- Prim's Algorithm
- Recursive Backtracking

These were the algorithms that were asked for by the prospective user, and that were in the objective above, as well as give a brief description of each algorithm. Unfortunately, I was unable to implement Kruskal's algorithm, as I did not have time, however, this was less important to the end user, so I feel that this does not effect the completeness of the project.

5.2.2 Solve Mazes

The project is able to solve mazes using multiple algorithms, as stated in the objective above, as well as give a brief description of each algorithm. The available algorithms for solving mazes are:

- Dijkstra's Algorithm
- Depth First Search
- Breadth First Search
- Greedy Search (Manhattan)
- Greedy Search (Euclidean)

These were the algorithms that were stated in the objective above, and include the request from the objective user, that there should be an algorithm as well as a heuristic. I feel that the breadth first search and depth first search make a good addition to the project, as they are algorithms that are studied in A Level Computer Science, and one of the purposes of this project is for it to be used in classrooms.

5.2.3 Customisation

All of the options for customization in the objective has been implemented into the project. The user can change the size of the maze, the speed of the animation, and the heuristic used in the greedy search algorithms.

5.2.4 User written algorithms

Unfortunately, the project is unable to run user written algorithms. This is because I have implemented a client server model, where the algorithms are run on the API, and it would be unsafe to run user written code on the API without first parsing and sanitizing the code, which is beyond the scope of this project.

5.2.5 Update Visualization

The project is able to update the visualization without the user having to re-run the visualization when the start or end nodes are moved, as stated in the objective above.

6 Evaluation

6.1 Independent Feedback

I wanted to go back to the end users (students in my class and computer science teachers at my school) with the finished visualization, to see if they had any feedback on the project or any ideas on further improvements that could be made. There feedback was:

- Item 1. "I thought that the menus were intuitive, and that the key was very useful to know what the different coloured squares meant."
- Item 2. "It could be helpful to download a gif animation of the maze being solved, particularly in the case of a teacher sharing it with the class."
- Item 3. "I think it would be nice if I could download a gif of the maze being solved."
- Item 4. "I think that the visualization was high tier."
- Item 5. "It would be nice if there were more maze generating algorithms, such as Kruskal's algorithm."
- Item 6. "It looks very nice, but the speed slider is counter intuitive, with a higher value being slower."
- Item 7. "I especially like that there is information on the algorithms, however, the information for breadth first search has a typo in it. It says "depth first search" instead of "breadth first search", which is what the description is describing."

6.2 Response to Independent Feedback

6.2.1 Gif

"It could be helpful to download a gif animation of the maze being solved, particularly in the case of a teacher sharing it with the class."

Although I think that being able to download a gif of the maze being solved would be a beneficial feature, it would require a lot of work on the back end, as there is no software for rendering the maze being solved on the backend. Because of this, I have put it into the "Improvements that could be made" section.

6.2.2 Speed Slider

"It looks very nice, but the speed slider is counter intuitive, with a higher value being slower."

Having contemplated this user feedback, I have decided that the speed slider should be changed. To do this, I have altered the code so that a lower value of the slider will result in a slower visualization. This change has been reflected in the live version of the project.

Before previously the value of the slider was used as the value of the speed. When the value of the slider was higher, the delay for the animation was higher and hence the visualization was slower.

```
1 <input type="range" defaultValue={this.props.speed} min={0.01} max
    =0.3} step={0.01} onChange={(e) => {this.props.setSpeed(e.
    target.value)}} />
```

After With the changes shown below, the current value of the slider is subtracted from the max value of the slider (0.3s). This is then used as the value of the animation, meaning that a lower value of the slider will result in a higher delay, and hence a slower visualization.

```
1 <input type="range" defaultValue={0.3 - this.props.speed} min
    =0.01} max={0.3} step={0.01} onChange={(e) => {this.props.
    setSpeed(0.3 - e.target.value)}} />
```

6.3 Improvements that could be made

There are a few improvements that I would make to this project if I had more time:

1. Add a "via point" node that the user can drag and drop that the path must go through.
2. Make the size of the maze squares automatically scale with the size of the maze, so that the user does not need to zoom out for larger mazes.
3. Allow for rectangular mazes.
4. User drawn mazes.
5. Allow the user to download a gif of the maze being solved.
6. Allow for the path finding visualization to be paused.

6.4 Evaluation

I feel that this project has been a success, as almost all of the objectives have been completed. The only objective that was not completed was the user written algorithms, however, this was not completed due to the security risks of running user written code on the API.