Path-finding Algorithms and Solving Mazes

Oliver Temple

November 23, 2021

Contents

1 analysis

1.1 Project Description

Path finding algorithms are essential in many aspects computer science, especially in games and simulations. However, they can be complex things that are hard to visualize, especially when being taught about them. Path finding algorithm visualizers do exist, however, I feel that none of them are perfect.

Because of this, I am going to make a path finding visualizer that ticks all of the boxes. It will be simple and easy to use, have advanced customization and have the ability to run a path finding algorithms given by the user.

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

Class 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, depth fist search and Kruskal's algorithms for maze generation.

1.3 Project Background

1.3.1 Current Systems

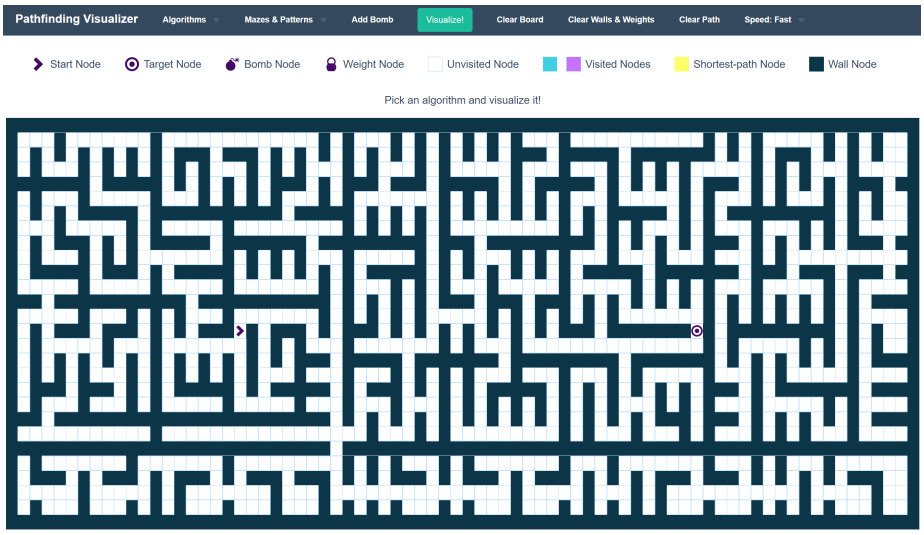
1.3.1.1 Example 1 https://clementmihailescu.github.io/Pathfinding-Visualizer

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:

- Djikstra'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.

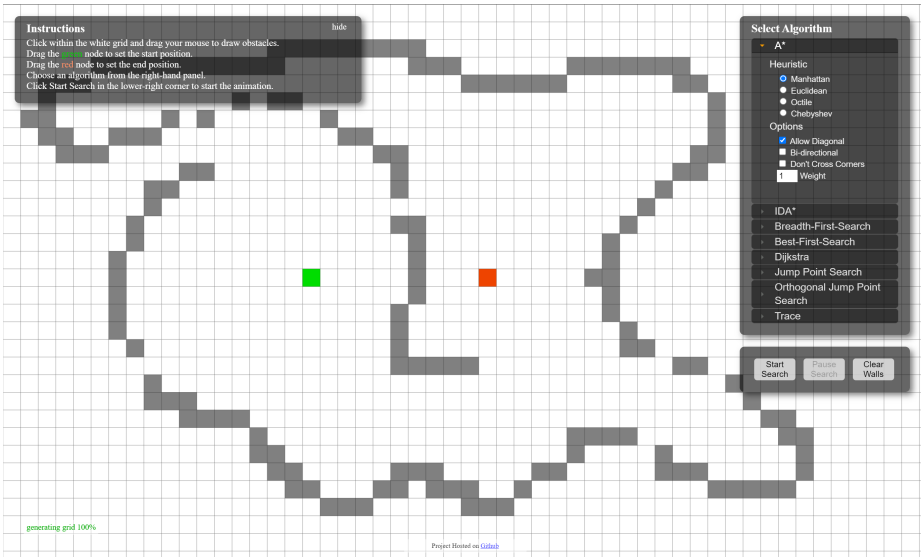
cons

- 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.

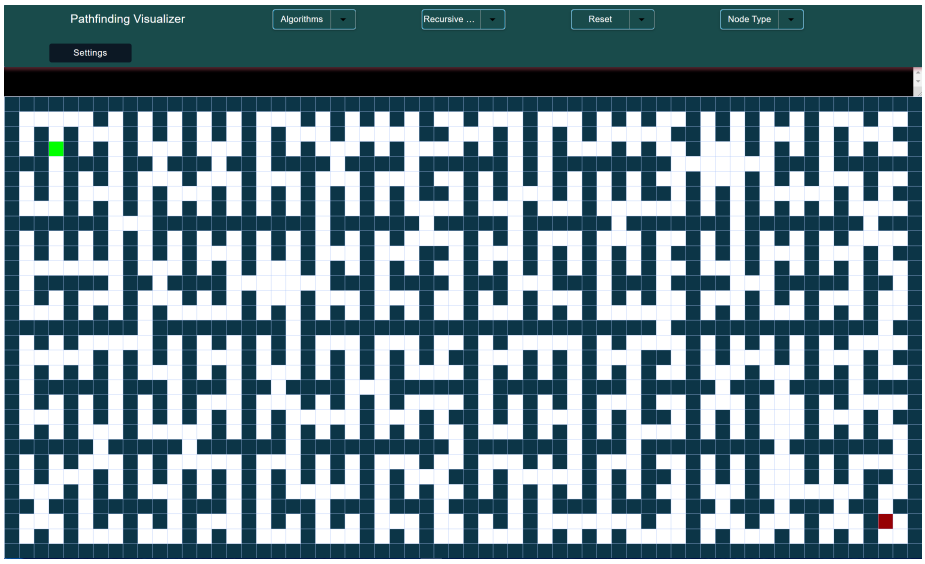
cons

- No maze generation.
- Visualization does not update when maze or start/finnish 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.

- Research the different algorithms needed and write the corresponding pseudoscode.

- 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 some prospective users 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 Kruskal's and Prims are similar but each has their own advantages and disadvantages and recursive backtracking is different and uses recursion.

A2. I would like to see Dijkstra's algorithm and A* search, as these are very popular algorithms and one is a heuristic, allowing the visualization to show the difference between an algorithm and a heuristic.

A3. Option to add your own png image for the drawing of final path - "a sussy imposter running around the maze".

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 need.

The answers to these questions confirms 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 and random generation. There should also be a brief description of the algorithm that has been selected and its advantages and disadvantages.

1.4.2 User Drawn Mazes

The user should be able to draw mazes and obstacles easily on the grid using the mouse.

1.4.3 Solve Mazes

The website should be able to solve mazes using multiple algorithms, including, but not limited to: A* 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 and its advantages and disadvantages.

1.4.4 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.

1.4.5 Special Nodes

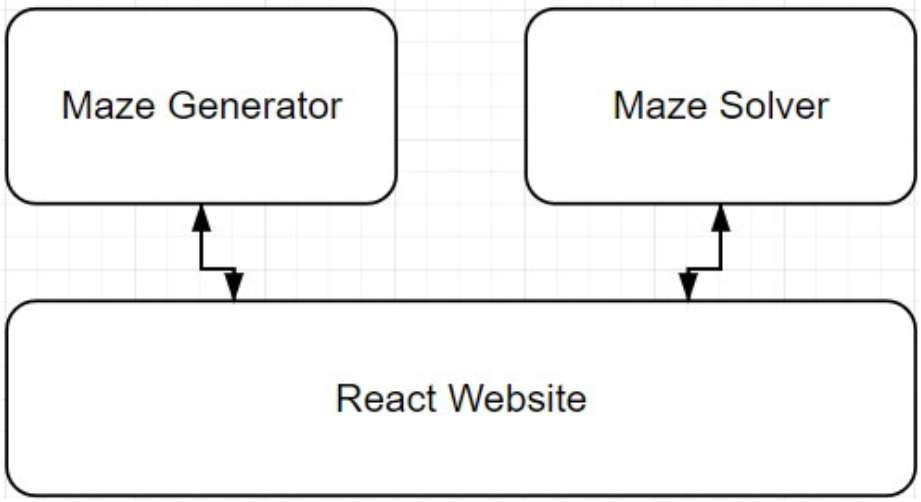
There should be "special" nodes that are different from walls or space. For example, nodes with different weights or a "via point" node that the path must go through.

2 Documented Design

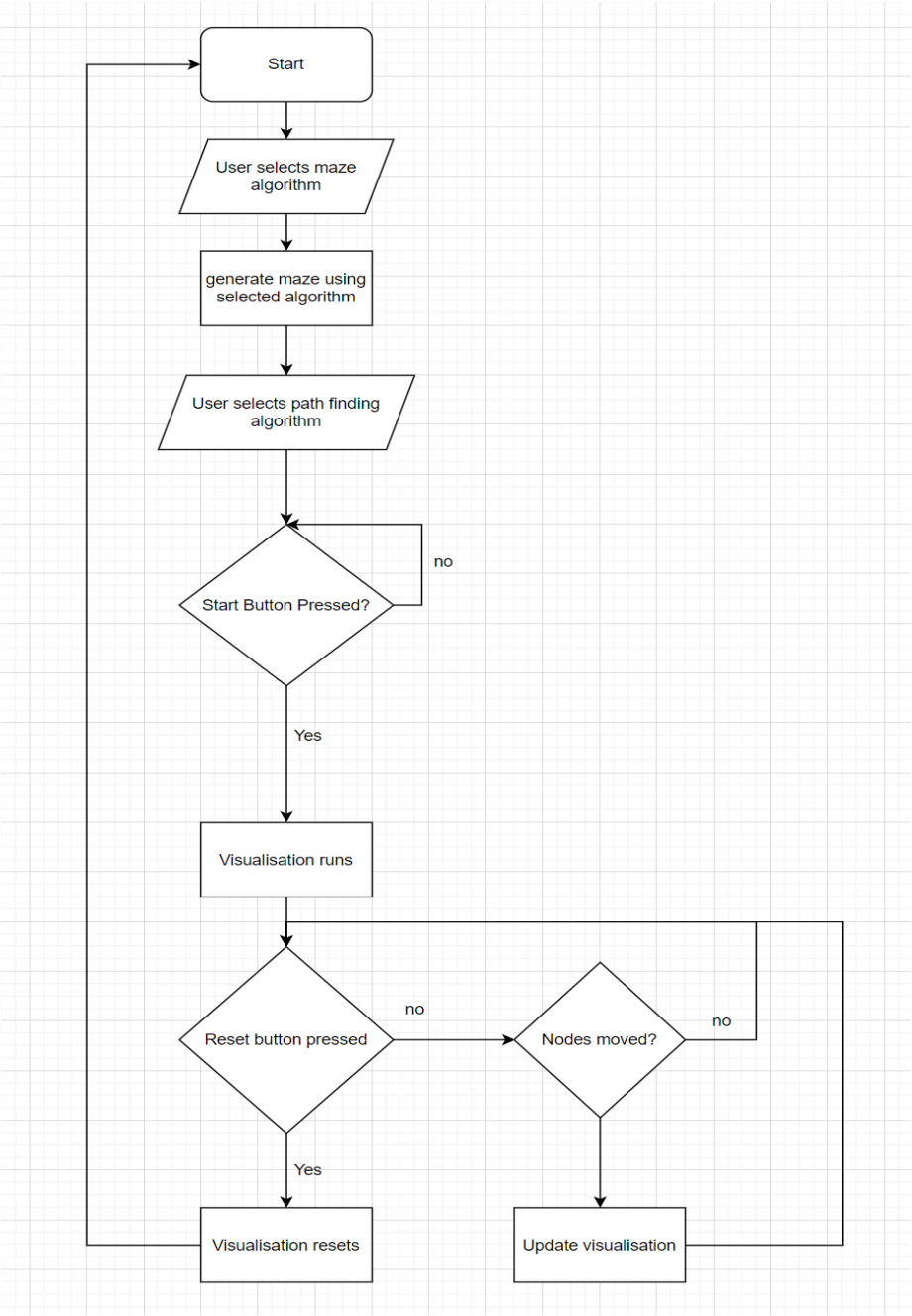
2.1 Visualization Structure

There will be a python API for generating and solving the mazes, and a react website that will visualize the algorithms. There will be two API endpoints,

on for generating the mazes and on for solving the mazes, hosted using AWS Lambda.

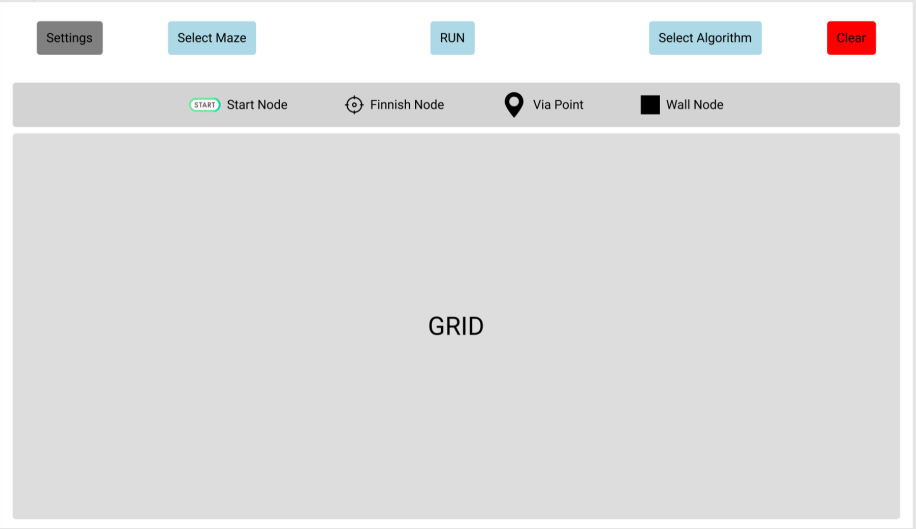


2.2 Flow Chart

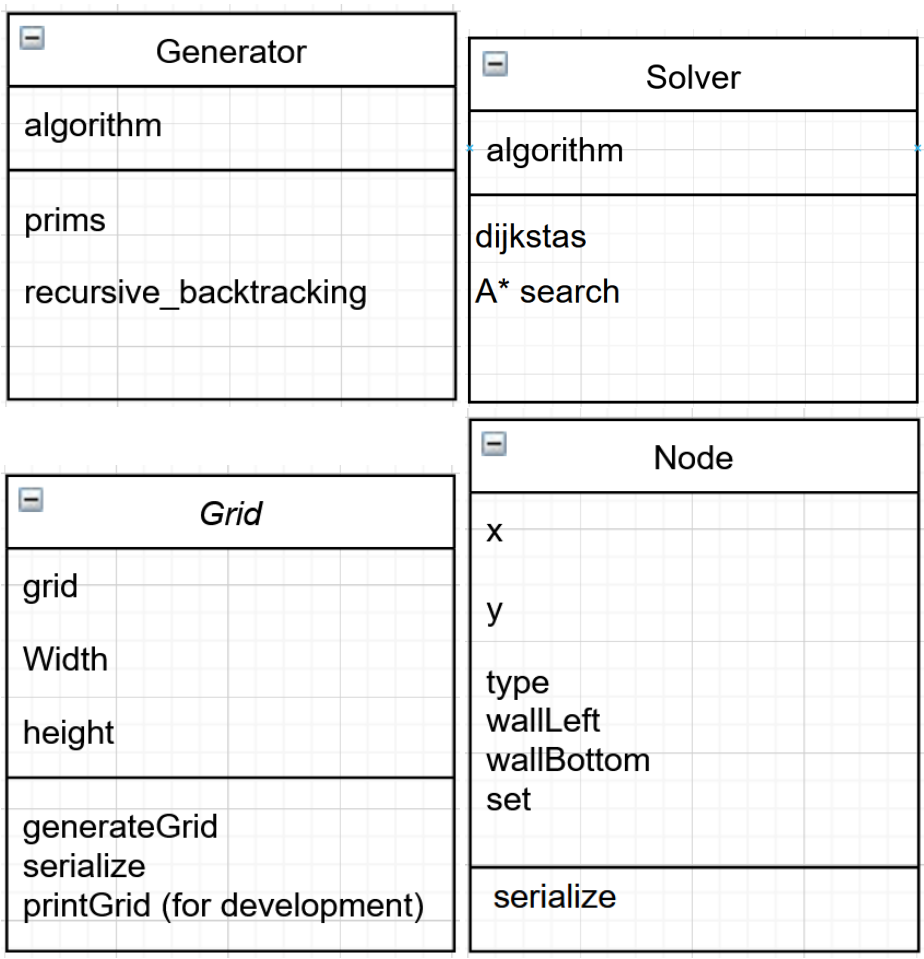


2.3 User Interface

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 Class Diagrams



2.5 Data Structures

2.5.1 Grid

The Grid for the maze will be stored in a 2D array with each item being a Node object. For example, a 2x2 grid would look like: [[Node, Node], [Node, Node]]

2.5.2 Prims

For prim's algorithm maze generation, multi-dimensional arrays will be used for storing the nodes that are in the maze and the "frontier" - a list of nodes that are adjacent to the current node and have not been visited. The frontier will be a 3D array, for example, [[[x, y], [available walls]], [[x, y], [available walls]], ...].

2.6 Algorithms

2.6.1 Maze generating algorithms

Pseudoscode for the maze generation algorithms.

2.6.1.1 prims

```
function prims(Grid)
    let inMaze = [[0,0]]
    let frontier = [
        [[0,1],["left"]],
        [[1,0],["top"]]
    ]

    while frontier.length > 0
        new <- frontier.pop(rand(0, frontier.length - 1))
        toAdd <- new[0]
        wall <- new[1][rand(0, new[1].length - 1)]
        inMaze.push(toAdd)

        if wall == "bottom"
            Grid.grid[toAdd[0]][toAdd[1]].wallBottom = false
        if wall == "left"
            Grid.grid[toAdd[0]][toAdd[1]].wallLeft = false
        if wall == "top" and toAdd[0] > 0
            Grid.grid[toAdd[0] - 1][toAdd[1]].wallBottom = false
        if wall == "right" and toAdd[1] < Grid.width - 1
            Grid.grid[toAdd[0]][toAdd[1] + 1].wallLeft = false

        possible <- [
            [toAdd[0]-1, toAdd[1]],
            [toAdd[0]+1,toAdd[1]],
            [toAdd[0],toAdd[1]-1],
            [toAdd[0], toAdd[1]+1]
        ]

        walls <- ["bottom", "top", "right", "left"]

        for i in range 4
            p <- possible[i]
            wall <- walls[i]
            if p[0] >= 0 and p[0] < Grid.height and p[1] >= 0 and p[1] < Grid.width
                if p not in inMaze
                    found <- false
                    for v in frontier
                        if v[0] == p
```

```
                            v[1].push(wall)
                            found = true
                    end for
                    if not found
                        frontier.push([p, [wall]])
            end for
        end while
    end function
```

2.6.1.2 Recusive backtracking

```
    function recursiveBacktracking(Grid)
        unvisited <- []
        for row in Grid.grid
            for items in row
                unvisited.push(items)
            end for
        end for

        start <- unvisited.pop(rand(0, unvisited.length - 1))
        recursive_backtracking_run(Grid, unvisited, start, [])
    end function

    function recursive_backtracking_run(Grid, unvisited, current, previous)
        orientation_options <- []
        if current.x > 0 and Grid.grid[current.y][current.x - 1] in unvisited
            orientation_options.push("left")
        end if

        if current.y > 0 and Grid.grid[current.y - 1][current.x] in unvisited
            orientation_options.push("top")
        end if

        if current.x < Grid.width - 1 and Grid.grid[current.y][current.x + 1] in unvisited
            orientation_options.push("right")
        end if

        if current.y < Grid.height - 1 and Grid.grid[current.y + 1][current.x] in unvisited
            orientation_options.push("bottom")
        end if

        if orientation_options.length > 0
            orientation <- orientation_options[rand(0, orientation_options.length - 1)]
            previous.push(current)

            if orientation == "left"
```

```
                connecting_cell <- Grid.grid[current.y][current.x - 1]
                cell.wallLeft = false

            else if orientation == "bottom"
                connecting_cell <- Grid.grid[current.y + 1][current.x]
                cell.wallBottom = false

            else if orientation == "right"
                connecting_cell <- Grid.grid[current.y][current.x + 1]
                connecting_cell.wallLeft = false

            else if orientation == "top"
                connecting_cell <- Grid.grid[current.y - 1][current.x]
                connecting_cell.wallBottom = false
            end if

            unvisited.remove(connecting_cell)

            recursive_backtracking_run(Grid, unvisited, connecting_cell, previous)
        else
            if previous.length > 0
                new <- previous.pop()
                recursive_backtracking_run(Grid, unvisited, new, previous)
            end if
        end if
    end function
```

3 Evidence of Completeness

4 Technical Solution

```
import json
from generator import Generator
from grid import Grid
from solver import Solver
from grid import Node
def lambda_handler(event, context):
    #get the parameters from the url query
    width = event["queryStringParameters"]["width"]
    height = event["queryStringParameters"]["height"]
    algorithm = event["queryStringParameters"]["algorithm"]

    #create a new generator to generate the maze
    myGenerator = Generator();

    #create a new grid with the height and width algorithms from
    the query
    myGrid = Grid(int(width),int(height))
```

```

    #generate the maze using the algorithm from the query
    if algorithm == "prims":
        myGenerator.prims(myGrid)
    elif algorithm == "recursive_backtracking":
        myGenerator.recursive_backtracking(myGrid)

    #return the json of the grid
    return {
        'statusCode': 200,
        'body': json.dumps(myGrid.serialize())
    }

#for development purposes
if __name__ == "__main__":
    myGenerator = Generator()
    myGrid = Grid(22,22)
    myGenerator.prims(myGrid)
    myGrid.printGrid()
    mySolver = Solver()
    mySolver.astar(myGrid,Node(0,0, ""),Node(21,21, ""))
    myGrid.printGrid()
```

```
import random

class Generator:
    def __init__(self):
        pass

    #setup for recursive backtracking
    def recursive_backtracking(self, Grid):
        #create a list of all nodes in maze
        unvisited = []
        for row in Grid.grid:
            for item in row:
                unvisited.append(item)

        #pick a random start node
        start = random.choice(unvisited)
        unvisited.remove(start)

        #start recursive backtracking
        self.recursive_backtracking_run(Grid, unvisited, start, [])

    #recursive backtracking algorithm
    def recursive_backtracking_run(self, Grid, unvisited, current,
    previous):
        #work out which walls can be removed
        orientation_options = []
        if current.x > 0 and Grid.grid[current.y][current.x-1] in
    unvisited:
            orientation_options.append("left")
        if current.y > 0 and Grid.grid[current.y-1][current.x] in
    unvisited:
            orientation_options.append("top")
        if current.x < Grid.width - 1 and Grid.grid[current.y][
    current.x+1] in unvisited:
            orientation_options.append("right")
```

```
32          if current.y < Grid.height - 1 and Grid.grid[current.y+1][
    current.x] in unvisited:
33              orientation_options.append("bottom")
34
35          #if there are no walls to remove, backtrack, else pick one
    and remove it
36          if len(orientation_options) > 0:
37              #pick a random wall to remove
38              orientation = random.choice(orientation_options)
39              #add the current node to the previous nodes list
40              previous.append(current)
41
42              #remove the wall depending on the orientation
43              if orientation == "left":
44                  connecting_cell = Grid.grid[current.y][current.x -
    1]
45                  current.wallLeft = False
46
47              elif orientation == "bottom":
48                  connecting_cell = Grid.grid[current.y + 1][current.
    x]
49                  current.wallBottom = False
50
51              elif orientation == "right":
52                  connecting_cell = Grid.grid[current.y][current.x +
    1]
53                  connecting_cell.wallLeft = False
54
55              elif orientation == "top":
56                  connecting_cell = Grid.grid[current.y - 1][current.
    x]
57                  connecting_cell.wallBottom = False
58
59              #remove the connecting node from the unvisited list
60              unvisited.remove(connecting_cell)
61
62              #recurse
63              self.recursive_backtracking_run(Grid, unvisited,
    connecting_cell, previous)
64          else:
65              #if not back at the start, backtrack
66              if len(previous) > 0:
67                  new = previous.pop()
68                  self.recursive_backtracking_run(Grid, unvisited,
    new, previous)
69
70      #generate a maze using prims algorithm
71      def prims(self, Grid):
72          #start at the top left node
73          inMaze = [[0, 0]]
74          #create a list of nodes connected to the start node
75          frontier = [[[0, 1],["left"]],[[1, 0],["top"]]]
76
77          #while there are still nodes to visit
78          while (len(frontier) > 0):
79              #pick a random node from the frontier
80              new = frontier.pop(random.randint(0, len(frontier)-1))
```

```
81
82              toAdd = new[0]
83
84              #pick a random wall from the available walls
85              wall = new[1][random.randint(0, len(new[1])-1)]
86
87              #add the wall to the inMaze list
88              inMaze.append(toAdd)
89
90              #remove the selected wall from the grid
91              if wall == "bottom":
92                  Grid.grid[toAdd[0]][toAdd[1]].wallBottom = False
93
94              if wall == "left":
95                  Grid.grid[toAdd[0]][toAdd[1]].wallLeft = False
96
97              if wall == "top" and toAdd[0] > 0:
98                  Grid.grid[toAdd[0]-1][toAdd[1]].wallBottom = False
99
100             if wall == "right" and toAdd[1] < Grid.width:
101                 Grid.grid[toAdd[0]][toAdd[1]+1].wallLeft = False
102
103             #calculate the possible nodes to add to the frontier
104             possible = [[toAdd[0]-1, toAdd[1]],[toAdd[0]+1,toAdd
    [1]],[toAdd[0],toAdd[1]-1],[toAdd[0], toAdd[1]+1]]
105
106             #possible walls
107             walls = ["bottom", "top", "right", "left"]
108             #iterate through the possible nodes
109             for i in range(4):
110                 p = possible[i]
111                 wall = walls[i]
112                 #check that the wall can be removed
113                 if 0<=p[0]<Grid.height and 0<=p[1]<Grid.width:
114                     #check that the node is not already in the maze
115                     if p not in inMaze:
116                         found = False
117                         #check if the node is already in the
    frontier, if it is then add the wall to the possible walls for
    the node in the frontier
118                         for v in frontier:
119                             if v[0]==p:
120                                 v[1].append(wall)
121                                 found = True
122                         #if the node is not in the frontier, add it
     to the frontier
123                         if not found:
124                             frontier.append([p,[wall]])
```

```
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.set = None
9
```

```
10          self.wallLeft = True
11          self.wallBottom = True
12
13      #for development purposes
14      def __str__(self):
15          return f"x:{self.x}, y:{self.y}, type:{self.type}, set:{
        self.set}, wallLeft:{self.wallLeft}, wallBottom:{self.
        wallBottom}"
16
17      #for returning the grid, the node must be serialized into a
        dictionary
18      def serialize(self):
19          return {
20              "x": self.x,
21              "y": self.y,
22              "type": self.type,
23              "wallLeft": self.wallLeft,
24              "wallBottom": self.wallBottom
25          }
26
27  #Grid class for the grid
28  class Grid:
29      def __init__(self, height, width):
30          self.height = height
31          self.width = width
32
33          self.grid = self.generateGrid()
34
35      #for returning the grid, the grid must be serialized into a
        dictionary
36      def serialize(self):
37          obj = {
38              "height": self.height,
39              "width": self.width,
40              "grid": []
41          }
42          for row in self.grid:
43              row_inner = []
44              for item in row:
45                  row_inner.append(item.serialize())
46              obj["grid"].append(row_inner)
47
48          return obj
49
50      #generate the grid
51      def generateGrid(self):
52          grid = []
53          for i in range(self.height):
54              row = []
55              for j in range(self.width):
56                  row.append(Node(j, i, "space"))
57              row.append(Node(j, i, "space"))
58              grid.append(row)
59
60          row = []
61          for j in range(self.width + 1):
62              row.append(Node(j, i, "space"))
```

```
        grid.append(row)

        return grid

    #for development purposes
    def printGrid(self):
        width = self.width
        height = self.height

        print(" __"*width)

        for i in range(height):
            for j in range(width):
                cell = ""
                if self.grid[i][j].wallLeft:
                    cell += "|"
                else:
                    cell += " "
                cell += "  "
                print(cell, end="")
            print("|")

            for j in range(width):
                cell = ""
                if self.grid[i][j].wallLeft:
                    cell += "|"
                else:
                    cell += " "

                if self.grid[i][j].wallBottom:
                    cell += "__"
                else:
                    cell += "  "
                print(cell, end="")
            print("|")
```

```
import './App.css';
import { Component } from 'react';
import DisplayGrid from './components/DisplayGrid';
import Menu from './components/Menu';
import MenuKey from './components/MenuKey';

class App extends Component {
  constructor(props){
    super(props);
    this.state = {
      grid: null,
      algorithm: null
    }
    this.fetchGrid = this.fetchGrid.bind(this);
    this.setAlgorithm = this.setAlgorithm.bind(this);
    this.clearGrid = this.clearGrid.bind(this);

    this.clearGrid();

  }

  async fetchGrid(){
```

```
    let grid = await fetch("https://jkrlv64tsl.execute-api.eu-west-2.amazonaws.com/default/NEA?width=15&height=15&algorithm="+this.state.algorithm)
    grid = await grid.json();
    console.log(grid)
    this.setState({
      grid:grid
    })
  }

  setAlgorithm(algorithm){
    this.setState({
      algorithm: algorithm
    })
  }

  async clearGrid(){
    let grid = await fetch("https://jkrlv64tsl.execute-api.eu-west-2.amazonaws.com/default/NEA?width=15&height=15&algorithm=null")
    grid = await grid.json();
    console.log(grid)
    this.setState({
      grid:grid
    })
  }

  render(){
    return (
      <div className="App">
        <Menu setAlgorithm={this.setAlgorithm} generate={this.fetchGrid} clearGrid={this.clearGrid}/>
        <MenuKey />
        <DisplayGrid grid={this.state.grid}/>
      </div>
    );
  }
}

export default App;
```

```
import React, { Component } from 'react';
export default class Menu extends Component {
    constructor(props) {
        super(props);
    }


    render() {
        return(
            <div className="menu">
                <button className="button settings">Settings</button>
                <select className="algorithms" name="algorithms" id="algorithms" onChange={(e) => {this.props.setAlgorithm(e.target.value)}}>
                    <option value="select">Select Maze Algorithm</option>
```

```
                    <option value="prims">Prims</option>
                    <option value="recursive_backtracking">
    recursive backtracking</option>
                </select>
                <button className="button" onClick={this.props.
    generate}>Run</button>
                <select className="algorithms" name="algorithms" id
    ="algorithms" onChange={(e) => {return}}>
                    <option value="select">Select Solving Algorithm
    </option>
                    <option value="prims">Prims</option>
                    <option value="recursive_backtracking">
    recursive backtracking</option>
                </select>
                <button className="button clear" onClick={this.
    props.clearGrid}>Clear</button>
            </div>
        )
    }
}
```

```
import React, { Component } from "react";

export default class MenuKey extends Component {
    render(){
        return(
            <div className="key">
                <div className="key_item">
                    <div className="key_node_start"></div>
                    <p>Start Node</p>
                </div>
                <div className="key_item">
                    <div className="key_node_end"></div>
                    <p>Finnish Node</p>
                </div>
                <div className="key_item">
                    <div className="key_node_via-point"></div>
                    <p>Via Point</p>
                </div>
            </div>
        )
    }
}
```

```
import React, { Component } from "react";
import DisplayNode from "./DisplayNode";
export default class DisplayGrid extends Component{
    constructor(props){
        super(props);
        this.state = {
            start: [0,0],
            end: [14,14],
            dragObject: ""
        }
        this.renderTable = this.renderTable.bind(this);
        this.handelDrop = this.handelDrop.bind(this);
        this.setDragObject = this.setDragObject.bind(this);
```

```
14      }
15
16      handelDrop(pos){
17          switch (this.state.dragObject){
18              case "start":
19                  this.setState({start: pos});
20                  break;
21              case "end":
22                  this.setState({end: pos});
23                  break;
24              default:
25                  break;
26          }
27      }
28
29      setDragObject(type){
30          this.setState({
31              dragObject:type
32          })
33      }
34
35      renderTable(){
36          return(
37              <table>
38                  <tbody className="column">
39                      {Array.from(Array(this.props.grid.height).keys
    ()).map((_, i) => {
40                          return(
41                                  <tr className={`row wall_right ${i
    === 0 ? "wall_top" : ""}`}>
42
43                                      {Array.from(Array(this.props.
    grid.width).keys()).map((_, j) => {
44                                          return(
45                                              <DisplayNode key={i+j}
    wallLeft={this.props.grid.grid[i][j].wallLeft} wallBottom={this
    .props.grid.grid[i][j].wallBottom} pos={[i, j]} start={this.
    state.start} handelDrop = {this.handelDrop} end={this.state.end
    } setDragObject={this.setDragObject}/>
46                                          )
47                                      })}
48                                  </tr>
49
50                          )
51                      })}
52                  </tbody>
53              </table>
54          )
55
56      }
57
58      render(){
59          if (this.props.grid){
60              return(
61                  <div className="grid" style={{padding:10}}>
62                      <this.renderTable />
63                  </div>
```

```
            )
        }else{
            return(
                <div className="grid"></div>
            )
        }

    }
}
```

```
import React, { Component } from "react"
export default class DisplayNode extends React.Component{
    constructor(props){
        super(props)
        this.state = {
            style:{}
        }

        this.handelDragStart = this.handelDragStart.bind(this);
        this.handelDragEnd = this.handelDragEnd.bind(this);
        this.handelDragLeave = this.handelDragLeave.bind(this);
        this.handelDragOver = this.handelDragOver.bind(this);
        this.handelDrop = this.handelDrop.bind(this);

    }

    handelDragStart(){
        this.props.setDragObject(this.start ? "start" : this.end ?
    "end" : "")
    }

    handelDrop(){
        this.setState({style:{}});
        this.props.handelDrop(this.props.pos)
    }

    handelDragOver(e){
        e.preventDefault();
        this.setState({
            style:{
                backgroundColor:"pink"
            }
        })
    }

    handelDragLeave(){
        this.setState({
            style:{}
        })
    }

    handelDragEnd(){
        this.setState({
            style:{}
        })
    }
    render(){
        this.classList = ["node"];
```

```
48          this.draggable = false;
49          this.start = false;
50          this.end = false;
51
52          if (this.props.wallLeft){
53              this.classList.push("wall_left")
54          }
55          if (this.props.wallBottom){
56              this.classList.push("wall_bottom")
57          }
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          }
63
64          if (this.props.pos[0] === this.props.end[0] && this.props.
    pos[1] === this.props.end[1]){
65              this.classList.push("node_end")
66              this.draggable = true
67              this.end=true
68          }
69
70          return(
71              <div style={this.state.style} className={this.classList
    .join(" ")} draggable={this.draggable} onDragStart={this.
    handelDragStart} onDrop={this.handelDrop} onDragOver={this.
    handelDragOver} onDragLeave={this.handelDragLeave} onDragEnd={
    this.handelDragEnd}>
72              </div>
73          )
74      }
75
76  }
```

5 Testing

6 Evaluation