Welcome to my Pathfinder. This document provides an overview of the app that I have built.

WHAT I’M SOLVING

The basic problem:   
Consider a grid with a start point and an end point, both of which can be moved around. Additionally, obstacles can be created on the grid as well. The task is to come up with an algorithm/ code that will find the shortest path from the start to the end around these obstacles.

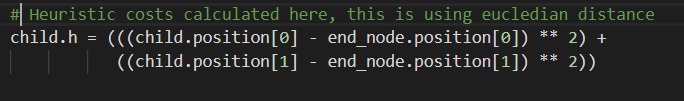
The algorithm:   
We will be implementing the A\* algorithm which is a popular heuristic based pathfinding algorithm. It involves the calculation of three parameters for every node in the grid:

* g denotes the *exact* distance from the start node to the node in consideration,
* h denotes the *approximate* distance from the node in consideration to the end node – it is calculated using heuristics,
* f is the sum total of these two parameters.

Procedure:

1. We will first declare a class Node to create an object for every node with parameters parent, g, h, f, current position and cost (assumed to be 1 for now).
2. We define a path function to return the path from start node to end node from the chain of nodes.
3. We define a search function as follows:
   1. Initialize all variables.
   2. Add the start node to a “yet to visit” list. Define a stop condition with max iterations/ end node.
   3. Repeat till stop criteria:
      1. Look for the lowest f cost square on the “yet to visit list.” This square becomes the current square.
      2. Check if the current square is the same as target square (then we have found the path)
      3. Use the current square and check four squares adjacent to this current square to update the children node. If it is not movable or if it is on the “visited list,” ignore it. Otherwise, create the new node with the parent as the current node and update the position of the node.
      4. Based on various conditions the child with least f is picked as the next node.
      5. Finally the last node visited is passed the path function, which chains it back to its parent and so on until the start node to create the path.

Heuristics:   
Here we have used the Euclidean heuristic to calculate the value of h as follows:

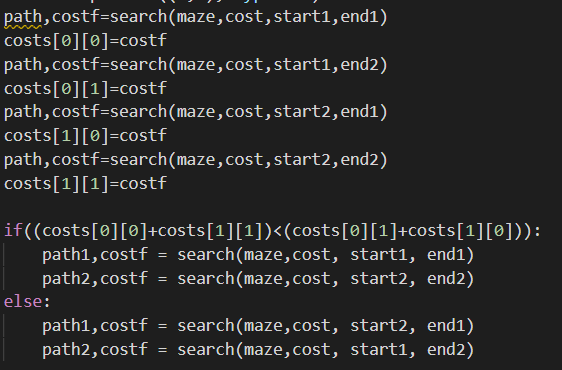


Other possible heuristics include Manhattan and Octal distances.

Here I have used a python backend with flask and implemented object oriented programming.

POSSIBLE FEATURES AND WHAT I HAVE IMPLEMENTED

* Including other algorithms like Dijkstra.
* Including an option to select cost.
* Including more nodes in between the start and end to create the likes of a travelling salesman problem.
* **The problem I implemented:** A specific problem inspired by the robot task allotment problem. Let there be multiple start and end nodes (equal in number in this case). The problem is to find the most optimized combination of start and end nodes – essentially the allotment of robots to destinations with the least cost. Here I have included two pairs of start-end nodes. Since it is a simple web application  
  used the distances directly to create the “cost function” which is essentially the sum of distances for each start-end combination. This cost function is minimized to find the best combination:



However, this is not optimized for a large number of nodes. In a case where the number of nodes is large, a possible solution would be to use a multi-layer perceptron or a Hopfield Neural network to optimize the function. Another possibility is to use metaheuristic algorithms such as the Genetic Algorithm to find the best combination.

**Other features:** You can add obstacles to the grid through multiple ways:

* + Toggle   -   You can toggle obstacles on or off
  + Obstacle options   -   You can select between three different obstacle densities
  + Draw obstacles   -   You can click or click and drag to generate obstacles with the pointer

THE BACKEND

For the backend Python was used as the scripting language, using Flask as the framework. In order to make communication possible between the frontend (JavaScript) and backend a Flask REST API was created using Flask-Restful. Data is sent from the frontend to the backend using POST requests via CORS.

THE FRONTEND

For the frontend JSX was used as the scripting language, with React as the framework. Fetch was used for the post requests to call the API.

WHAT I HAVEN’T SOLVED

There are numerous possibilities that can be implemented with this pathfinder, some of which I mentioned above. However, the main problem faced with all of these possibilities is the time taken to fetch data from the API to the frontend.

Due to the CORS preflight requests, it takes extra time for data to actually be fetched from the API. Since this activity is asynchronous, it is important to make sure that the data is obtained *before* the rest of the code is implemented. To solve the problem to some extent async – await could be used, along with setting timeouts. This is the reason that it takes an extra second for the path to be displayed.

Perhaps one problem than should be solved eventually, is the speed of path finding.

Another problem with the implementation of the Robot problem is that it doesn’t check for overlap in the paths; this is an issue both practically and with respect to the overall frontend display (it looks messy). Checking for this overlap would make it more of an AI agent.

HIGH LEVEL DIAGRAM

POST REQUEST: { startpoints, endpoints, grid, cost }

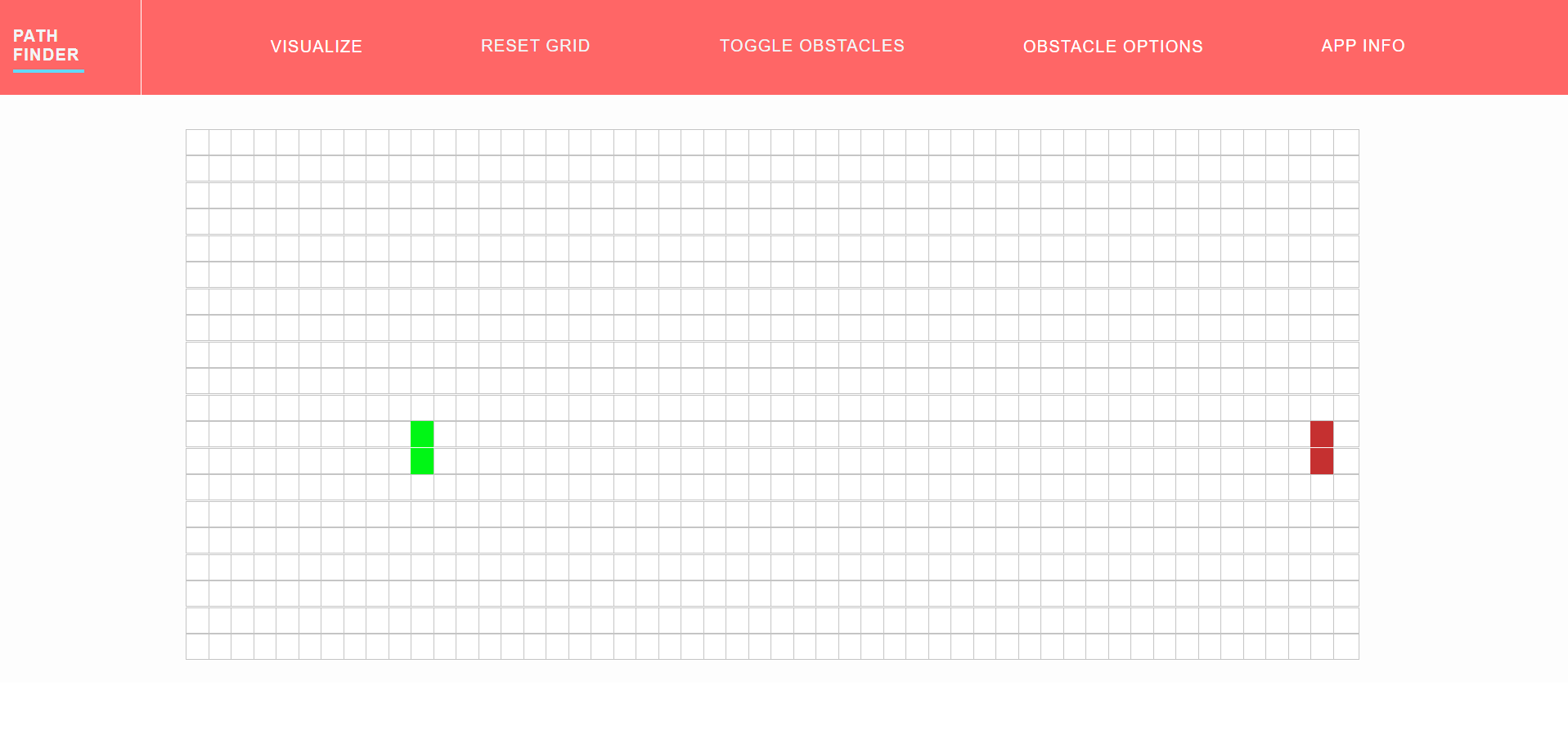
FLASK (BACKEND) SERVER

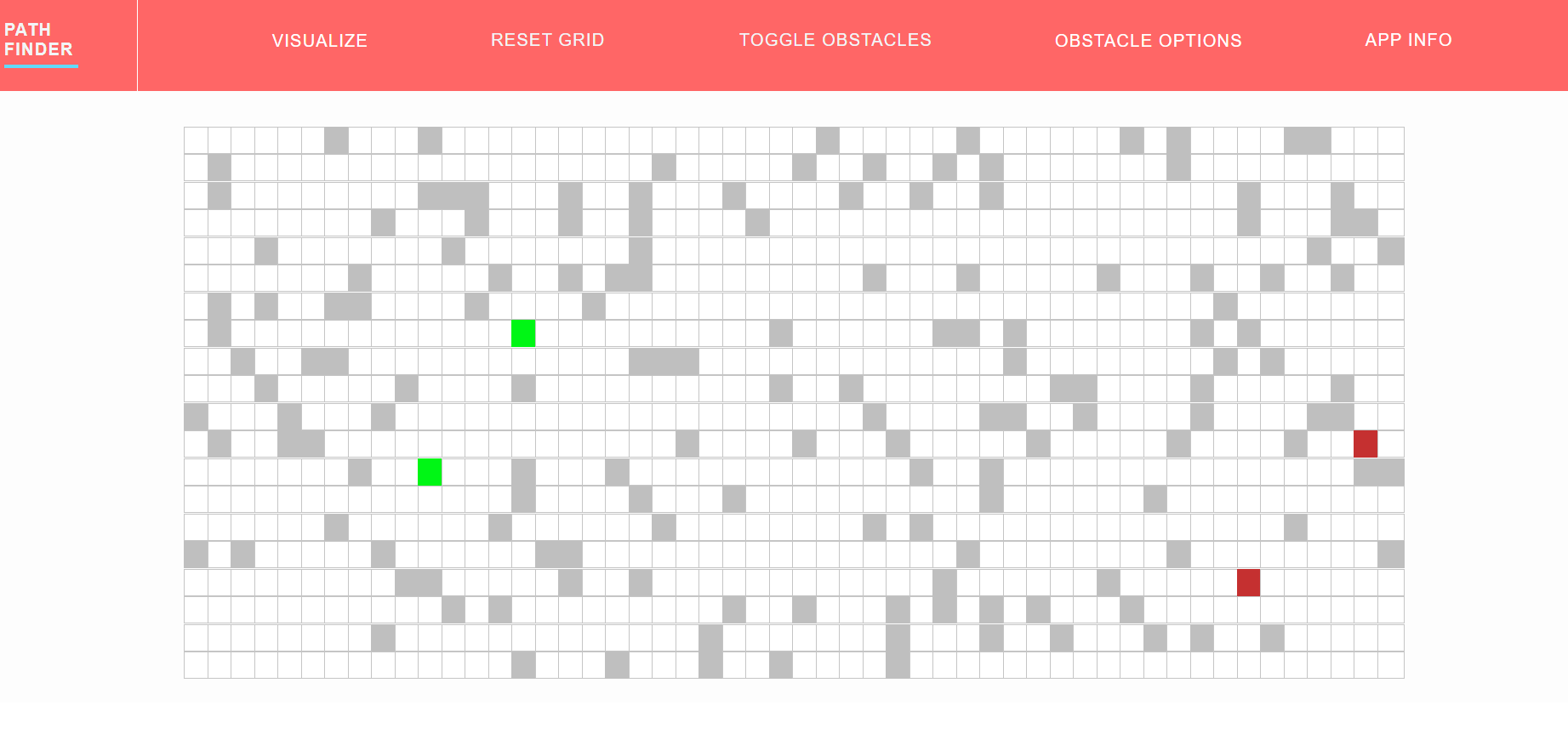
AI AGENT

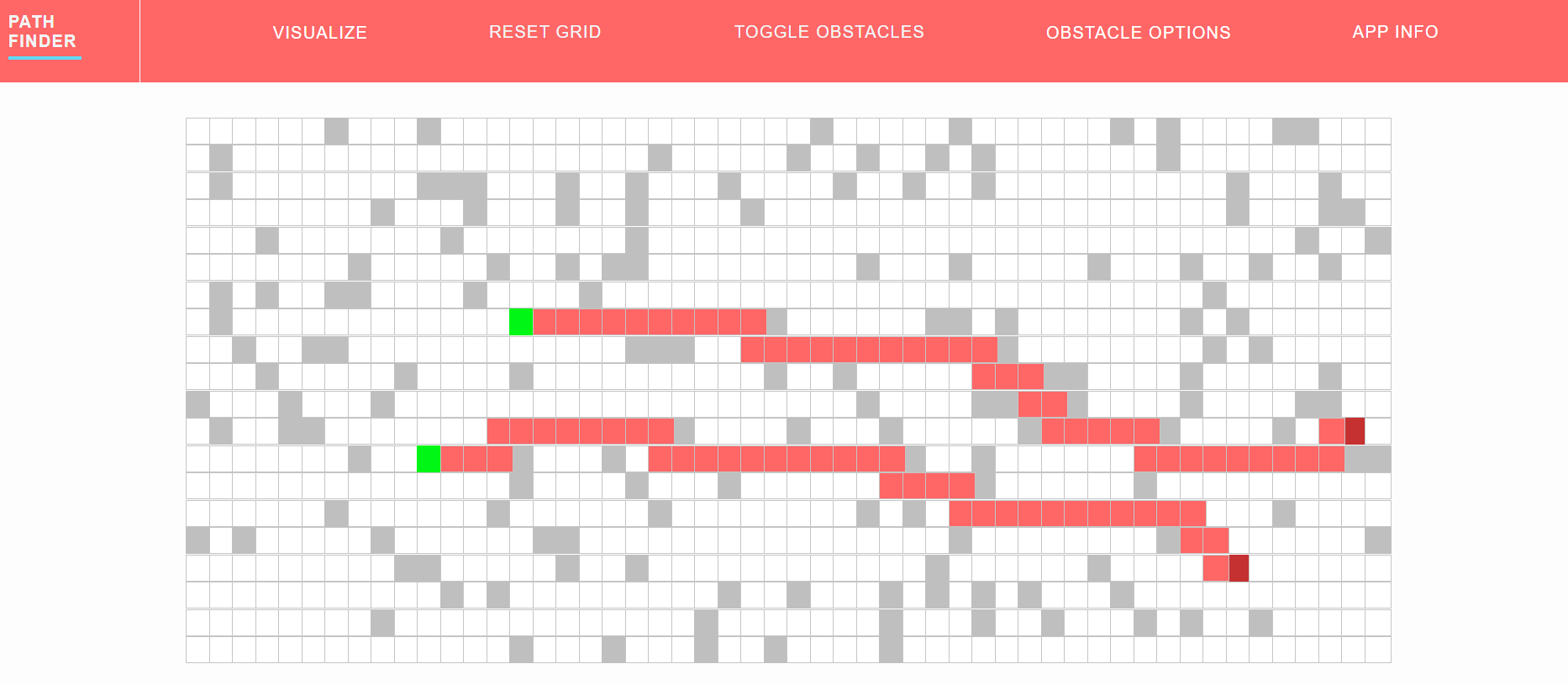
return: {paths}

REACT (FRONTEND) SERVER

SCREENSHOTS







Useful sources:

<https://github.com/qiao/PathFinding.js>

<https://github.com/DMGithinji/PathFinder-React>

<https://towardsdatascience.com/a-star-a-search-algorithm-eb495fb156bb>

<https://www.redblobgames.com/pathfinding/a-star/implementation.html>

<https://www.geeksforgeeks.org/a-search-algorithm/>