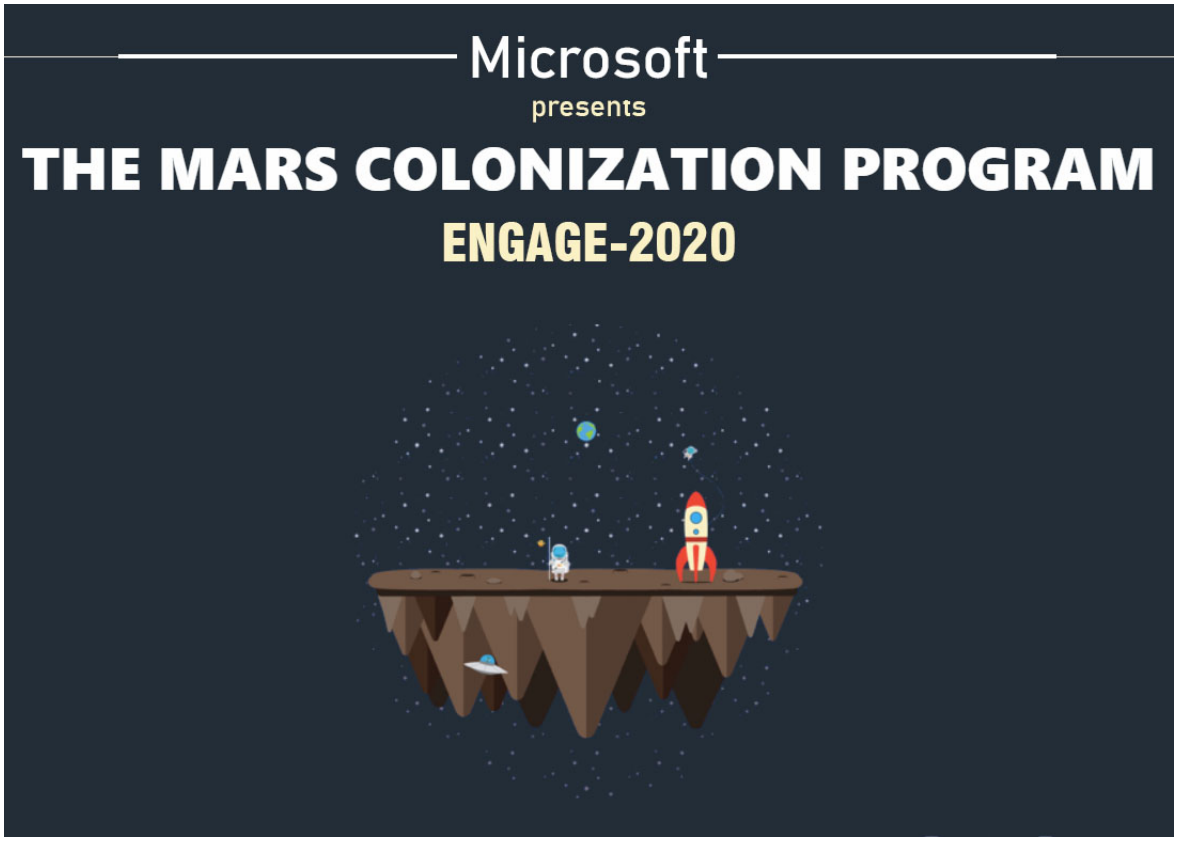
MicroMARS 42

Microsoft Engage-2020

The MARS Colonisation Program

Avyakta Wrat
Anjali Yadav
Nayan Barhate
Mantri Krishna Sri Ipsit

July 24, 2020



Contents

1 ACKNOWLEDGEMENT

We would like to thank the entire team of **Microsoft** to introduce programs like '**Engage**' and organize initiatives like the '**MARS Colonisation Programme**' under them. We thank them for floating projects which help young minds like us to learn new things, hone our skills, and contribute to some of the bigger goals. We express our gratitude to all the people associated with this program, especially the speakers who took out their valuable time to make this an insightful learning experience.

We extend special thanks to our mentor **Mayank Rathore** who took out time from his schedule to help and guide us through this project, held regular meetings to track the progress, as well as to help with all the doubts we had and made efforts to bond with us in the process.

2 MEET OUR TEAM

MicroMARS 42

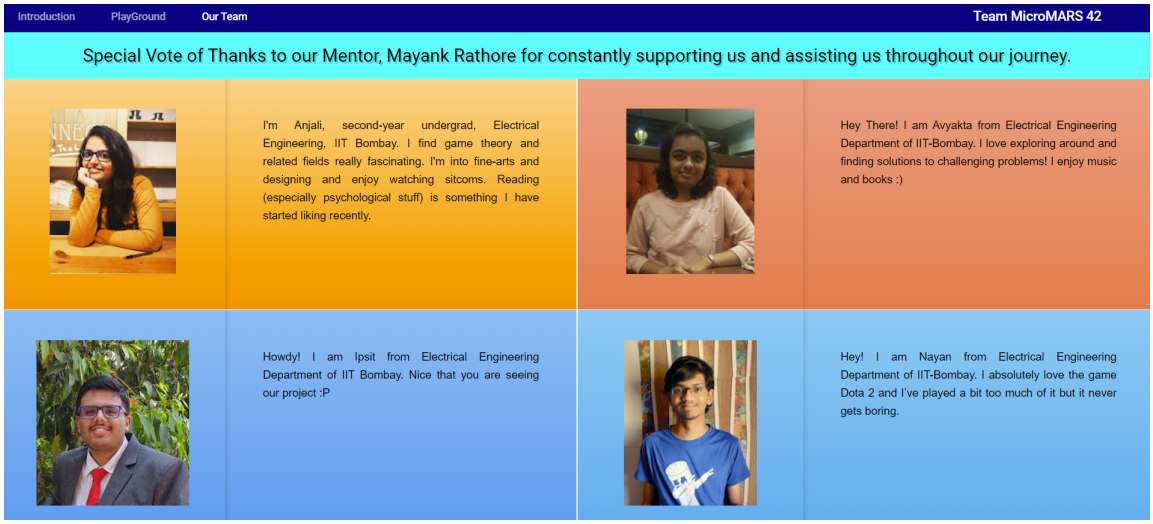


Figure 1: Meet Our Team

Resonating with the mission to have colonization on mars, we named our team **MicroMARS**, as we consider our efforts a small but significant contribution in getting a step ahead in the **MARS** mission of **Microsoft**. The number **42** is, in The Hitchhiker's Guide to the Galaxy by Douglas Adams, the "Answer to the Ultimate Question of Life, the Universe, and Everything", calculated by an enormous supercomputer named Deep. As we set out our curious minds to discover the wonders of mars planet and explore the possibility of life on it, we encounter many questions in our way and try and make an effort to unfold the solutions to some of them through our project.
We as a team strive to get selected as a part of the crew that is chosen to establish a permanent human settlement on Mars. We intend to contribute to the mission

by applying the concepts and skills we possess and the ones we learned during this program, in building a browser-based web-application for Project 1 i.e. **Navigate The Mars Rover**.

3 PROJECT CHOSEN

We as a team mutually agreed to work on project 1 out of the two projects floated on the website. The project aims to help the **Mars Curiosity Rover** find the shortest path between two points while avoiding obstacles on the way. And we set out to provide a unique shade to this solution by putting in efforts to make it more practical and evolved, by deploying some of the impressive ideas and better approaches we came up with, in our very own personal team meets.

3.1 Project Template Already Available

The surface of the mars has been visualized as a grid that fits the user's browser window and enables them to build their own custom paths including craters and hills to resemble the uneven surface of mars. The project template that was already available on the Microsoft website lets you choose the starting and destination point as per your preference in the grid setup available to you. You can toggle between the various algorithms, already employed, to choose the one you want to use. The algorithms also come up with heuristic settings and options like Allow diagonals, Bidirectional, Don't cross corners, etc. You have the freedom to create barrier walls as you click and drag through the grid. You have the option of clearing walls if you want to reset your grid. And, then finally, when u go for the 'Start Search' button, it shows you the shortest path between the two points you located, using the algorithm you chose, avoiding the barrier walls you added, adhering to the heuristics and options you checked on.

4 OUR PROJECT

Recommended: *You view this project on a laptop/tablet/computer to get the best results.*

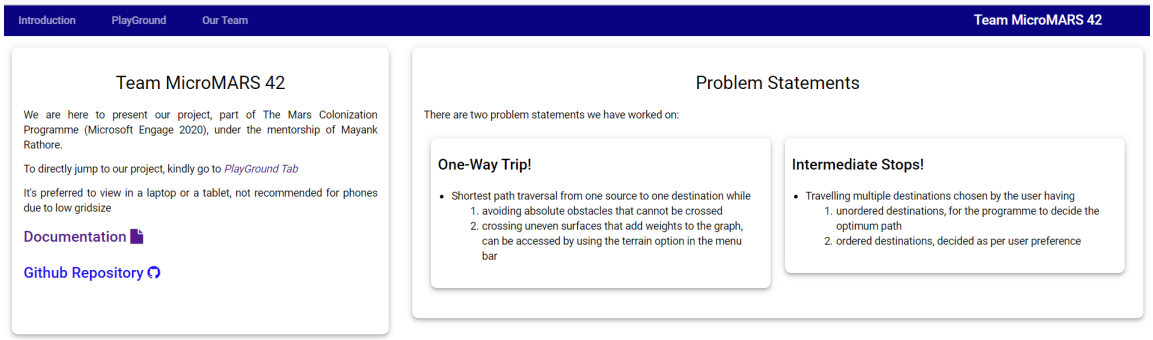


Figure 2: Our Project

4.1 Getting Introduced with the Playground

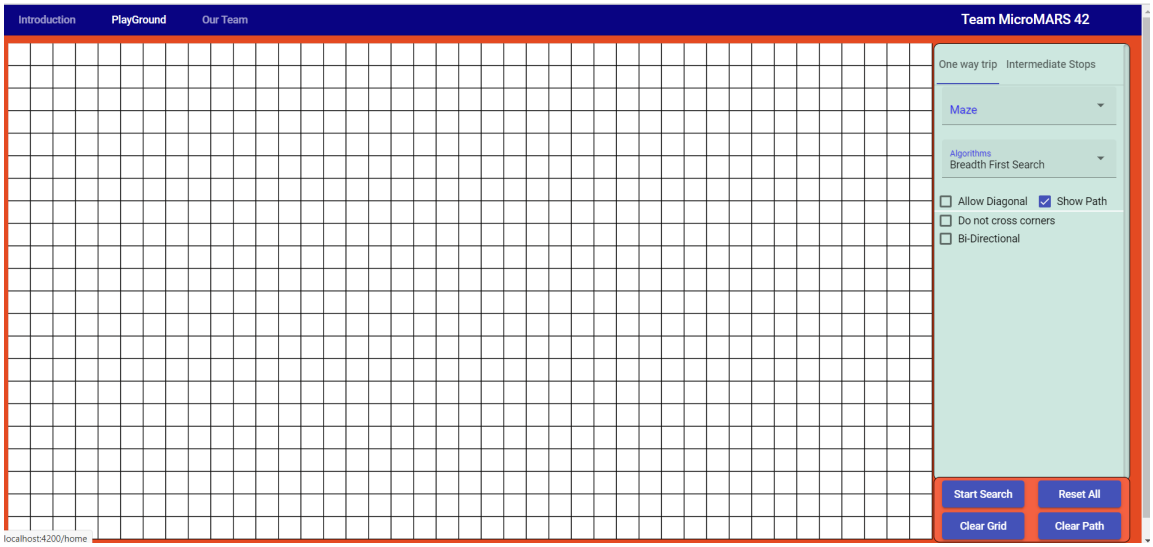


Figure 3: Our Playground

As you land on the Playground section of our web app, you see the usual grid template and an options area on your right. We solved the problem by assuming the grid as a graph and explored its connectivity.
The options area provides you with two main menus to toggle between, named **One-way trip** and **Intermediate stops**. The One-way trip menu provides you with the feature to choose a starting point and only a single destination point. On the other hand, inspired by the traditional '**Travelling Salesman Problem**', Intermediate stops option gives you the liberty to choose a starting point and vary the number of destination points, by using the sliding tool to get up to five destination points.

Both of the options carry the feature named '**Maze**' which makes the effort of creating barrier walls easier for you by providing you with options of some default barrier wall structures that you can easily create just by selecting those options under this feature.

The options that we have for you right now are:
- Sidewinder Algorithm
- Depth First Search
- Prim's Algortihm
- Binary Tree
- Random Maze
- Staircase
- Mountain on end
- Mountain on start
- Mountain between start and end

4.1.1 ONE WAY TRIP MENU

Now, the One Way Trip menu gives you the options to choose between algorithms **A*** and **Dijkstra**. You have the following **Heuristic** options for A* and Best-First Algorithms:

- Manhattan
- Euclidean
- Octile
- Chebyshev

You also get the option to choose for features like '**Allow Diagonal**' and '**Show Path**'.

Figure 4: Terrain and Gaussian-distributed Hills

You also encounter some new and interesting features like '**Terrain Distribution**' as well as '**Gaussian Distribution**' which you may choose to add to your interface.

The feature of terrain distribution lets you create barrier walls with different rigidness. It allows you to choose a value with the sliding option. This value helps you set how rigid the barrier is and how difficult it is to cross it, on a scale of **0 to 100**. The rigidity is mapped in proportion to the opacity values of the barrier walls shown on the interface, where value 0 reflects plain ground, no rigidity, easy to cross, and value 100 reflects a strong barrier wall, very high rigidity, cannot be crossed.

To further enhance a users' experience, the feature of adding terrain obstacles with variable rigidity is integrated with the option of Gaussian Distribution, where you can set up the x-axis and y-axis co-variances, as per your wish and can create a mountain obstacle modeled in the Gaussian distribution adhering to the coordinate values you chose. It makes the entire process of adding realistic barriers to your grid all the more efficient and gets you towards the goal of reflecting realistic terrain topology of mars onto your grid. We have chosen the bi-variate Gaussian distribution by taking inspiration from the **empirical rule** of statistics, which states that the probability of finding a normally distributed random variable in a width of thrice the standard deviation around mean is 99.7%. By using this the user can create hills that will match his/her imagination to the maximum possible extent.

4.1.2 INTERMEDIATE STOPS MENU

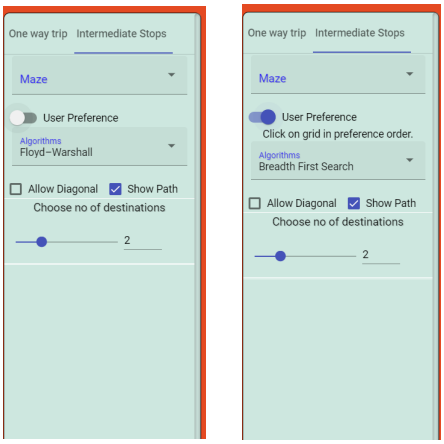


Figure 5: Intermediate Stops Menu

While choosing multiple destinations under the purview of this option, you get the independence of either setting your preference order of destinations or just leave it to the interface to prioritize them with shortest path optimization.

The '**User Preference**' toggle button provides you with this liberty. If you check it on, the preference order will be set in the order you select your destination points. Else, it will just use the shortest path optimization to select the preference order among the destination points you chose. While doing so, it will **display the preference order** on the path shown, once it's done finding the shortest route for our rover.

The non user preference part has been deployed using the '**Floyd-Warshall**' algorithm, again with options like '**Allow Diagonal**' and '**Show Path**' whereas the user preference is done by finding shortest path between every pair of start and end point using the previous One Way Trip Algorithms.

4.1.3 KEY BUTTONS SECTION

Figure 6: Key Buttons Section

Just below the options section, sits the Key Buttons Section.

- **Start Search:** You push this button to start with the shortest-path algorithm which calculates the same as per the problem statement you customized, and shows you the path as well as display the results in the grid when done.
- **Clear Grid:** Pushing this button clears the grid displaying the results.
- **Clear Path:** This button allows you to clear the path, while still preserving the customized construct you created.
- **Reset All:** Push this button to undo all your customization and construct and get a fresh interface to start all new.

5 UNIQUENESS OF SOLUTION

We as a team tried to come up with impressive ideas and better approaches for evolving the project thus taking us ahead in our bigger dream of human colonization on Mars, and made efforts to implement them in our interface.

While trying to achieve all this, we always focused on two key points. Firstly, any additions that we make should take us closer to the goal of establishing a permanent human settlement on Mars. It should increase the extent of practicality and realism adhering to the problem statement prototype, in our interface. Secondly, the interaction of the end-user with our own interface should be very smooth and comfortable. He/She should be very comfortable in using our interface to create the construct as per his own problems for which he seeks answers to, modifications and alterations in the construct at any point of time should be very feasible and drawing out relevant information from the results we provide with the help of our interface should come very easy to him/her.

5.1 Maze Options

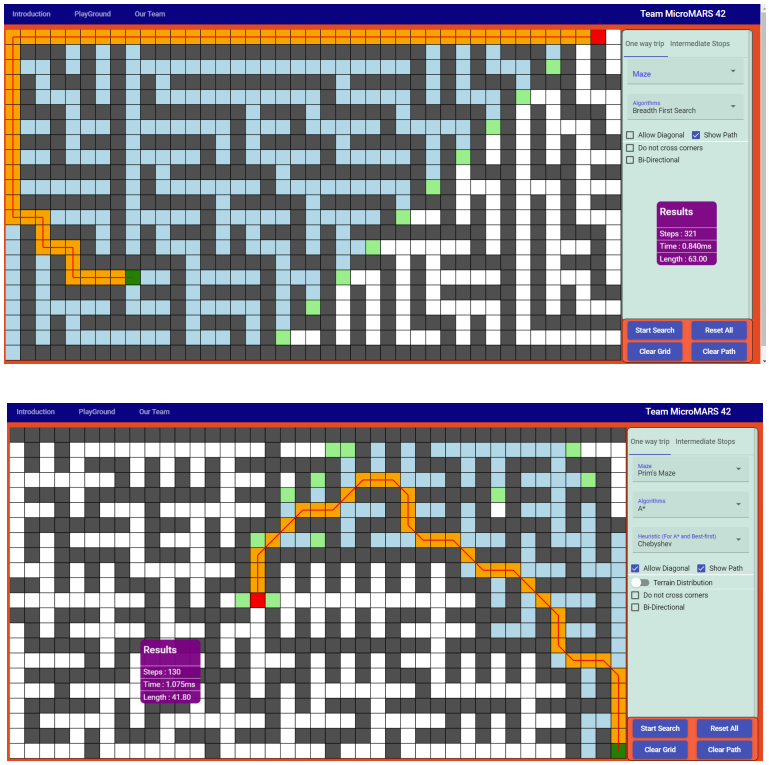


Figure 7: Maze Working

In order to make it very easy for the user to construct some of the basic barrier topology on the interface, we introduced the section named '**Maze**' which contained a list of various options on different structures of mazes he/she can choose to imbibe. He/She can then also make changes to these default mazes by building up some more barriers to get the whole structure close to what he/she wishes to achieve.

5.2 Intermediate Stops

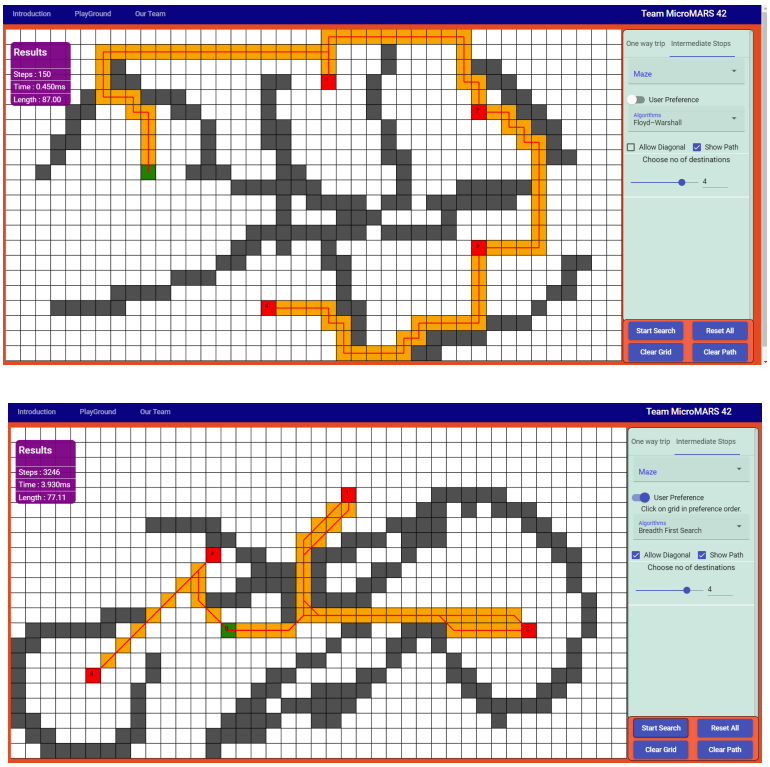


Figure 8: Multiple Destinations Working

When we did some fundamental research on the rovers that have successfully landed on Mars, we got to know about a lot of interesting details. The rovers don't travel long distances, the average distance traveled by a rover ranges between 40-50 km. Thus, it becomes very important that we optimize our interface to achieve maximum results while covering fewer distances. If a rover has several tasks to finish which require it to travel to multiple destinations, it's more efficient to provide the shortest path that spans all the desired locations and optimize the preference order followed for them, while also adhering to user preferences, if pushed any.

Thus, we introduced the feature of Intermediate stops in our interface, where you get to input **multiple destinations**, push in your **preference order** if you wish to do, and retrieve the desired results, thus reducing down the cost and time wasted in spanning each of the destinations separately.

5.3 Variations in the Obstacles Added (Non-weighted, Weighted, Hills)

As we ventured into finding more about Mars as a planet, we made some key observations regarding the topological aspects of this red planet. The northern lowlands comprise about one-third of the surface of Mars and are relatively flat, with occasional impact craters. The other two-thirds of the Martian surface are the southern highlands. The difference in elevation between the hemispheres is dramatic.

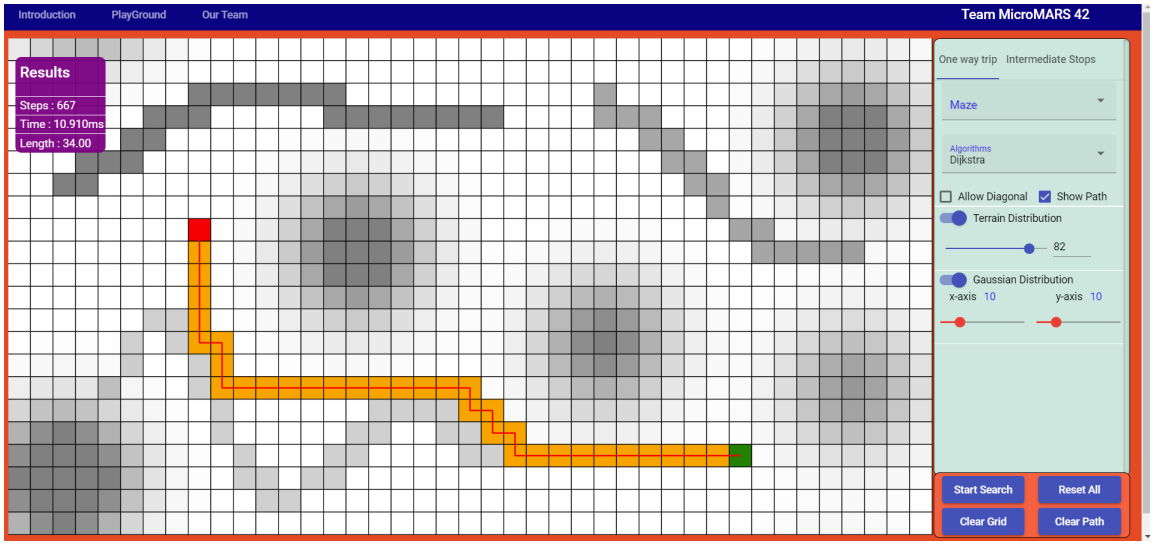


Figure 9: Terrain Working

In order to cater to this varied topology, we decided to modify the feature of adding obstacles thus making it really easy and efficient for the user to replicate the actual terrain on our interface. The obstacle system of the interface available on the website worked in a binary fashion, either it's a plane land that you can pass easily, or it's a rigid barrier that your rover is unable to cross. The obstacles were **non-weighted**. We preserved this idea in our own interface as well, but we gave due consideration to the thought of introducing weighted-obstacles as well. Reality is far away from this binary setup, we have some barriers which the rover has the capacity to cross but it will cost it a bit more when compared against the plain ground, and this cost will vary with every individual obstacle.

Our interface allows you to add **weighted obstacles** and regulate the weight you want to associate with them, by choosing a value between 0 to 100 where you measure the rigidity and the extra cost it will cost to cross the barrier.

But, we had another issue in hand, the dramatic elevation changes in the terrain and how to reflect them. We imagined these dramatic elevations as **hills** and introduced another form of obstacles built using Gaussian distribution principles. You get the option to change the x-axis and y-axis co-variances to feed the span of the hill, and the interface automatically generates a hill-like obstacle as per your preferences, with the peak at the grid point you click on.

5.4 Suggestion Pop-ups

A user new to the interface may face a brief span of time in getting adjusted to the interface and its dynamic nature. Also, while you have immersed yourself creating the perfect construct for fetching out results to your own problem statements, you might lose out on some trivial details or considerations. Our interface helps you to be carefree about this, and not waste your time wondering where have you gone wrong. It pops a suggestion box, whenever you fall out on such details, advising you the necessary course of action which will eventually help you rectify your mistakes, thus making the whole experience extensively smooth for you.

6 COMPONENTS

6.1 How Graphs Helped us

A graph is a very efficient and useful data structure when it comes to solving the problem of the shortest path. All types of movement on a grid can be modeled by providing suitable edge weights to the graph. Thanks to the different types of traversals, and different type of representations of graphs, we were easily able to model the grid cells as nodes, and obstacles as edge weights. We used the **adjacency matrix** and adjacency list representations as per the requirement.

6.2 Meet our Algorithms

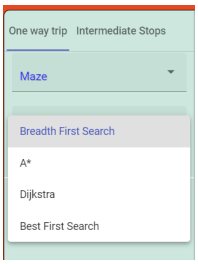


Figure 10: Different Algorithms

6.2.1 Breadth-First Search:

Non-weighted obstacles
A great algorithm; guarantees the shortest path.

6.2.2 A*:

Variable obstacles
Arguably the best path-finding algorithm; uses heuristics to guarantee the shortest path much faster than Dijkstra's Algorithm.

6.2.3 Dijkstra:

Variable obstacles
The father of path-finding algorithms; guarantees the shortest path.

6.2.4 Best First Search:

Non-weighted obstacles
A faster, more heuristic-heavy version of A*; does **not** guarantee the shortest path.

6.2.5 Floyd-Warshall:

Non-weighted obstacles, multiple destinations
Finds all pairs of shortest path problems from a given weighted graph, generates a matrix representing the minimum distance from any node to all other nodes in the graph; guarantees the shortest path.

6.3 Help from heuristics

6.3.1 Manhattan:

It is computed by calculating the total number of squares moved horizontally and vertically to reach the target square from the current square.

6.3.2 Euclidean:

It is the straight-line distance, can never over estimate the distance, but might underestimate it if there is an obstacle to avoid on the line between the current node and the destination.

6.3.3 Octile:

If the x-distance and y-distance to the destination are x and y, the octile heuristic is max(x, y) + (sqrt(2)-1)*min(x, y), diagonal costs are also commonly used for efficiency reasons.

6.3.4 Chebyshev:

It examines the absolute magnitude of the differences between the coordinates of a pair of points.

6.4 Options for Maze

- **Sidewinder Algorithm:** Very similar to the Binary Tree algorithm, and only slightly more complicated, only needs to consider the current row and therefore can be used to generate infinitely large mazes and have just one long passage.
- **Depth-First Search:** Mazes generated have a low branching factor and contain many long corridors because the algorithm explores as far as possible along each branch before backtracking.
- **Prim's maze:** This minimal spanning tree algorithm rather than working edgewise across the entire graph, starts at one point and grows outward from that point to generate mazes.
- **Binary Tree:** One of the very rare algorithms with the ability to generate a perfect maze without keeping any state at all. It is an exact memory-less algorithm with no limit to the size of Maze you can create and can build the entire maze by looking at each cell independently.
- **Random:** Works on the simple algorithm to generate mazes; find available directions, pick one at random, if no available direction backtrack to the last position.
- **Staircase:** Generates a maze similar to the structure of a staircase.
- **Mountain on end:** Creates a terrain with a mountain on your destination point.
- **Mountain on start:** Creates a terrain with a mountain on your starting point.

- **Mountain between start and end:** Creates a terrain with mountains placed exactly at the center of start and destination point.

6.5 Other Freedom of Choices

Just below the options section, sits the Key Buttons Section, with some key buttons on it.

- **Allow Diagonals:** Allows the algorithm to move in 8-directions rather than just 4-directions, by allowing it to also cross over the diagonals, along with edges.
- **Do not cross corners:** This option decides whether to find a path that passes through a corner made by obstacles or take a round turn.
- **Bidirectional:** This option makes the algorithm to start searching from both origin and destination.

6.6 MANIFESTATION OF UNIQUE SOLUTIONS

6.6.1 Maze Generating Algorithms:

To enable different maze options, we made a class implementing different maze generating algorithms in different methods. We then bound an object of this class to the Angular component associated with our playground and based on the option chosen by the user, we changed the grid accordingly. All the code for maze can be found in the "***./src/app/playground/include/maze.ts***" file.

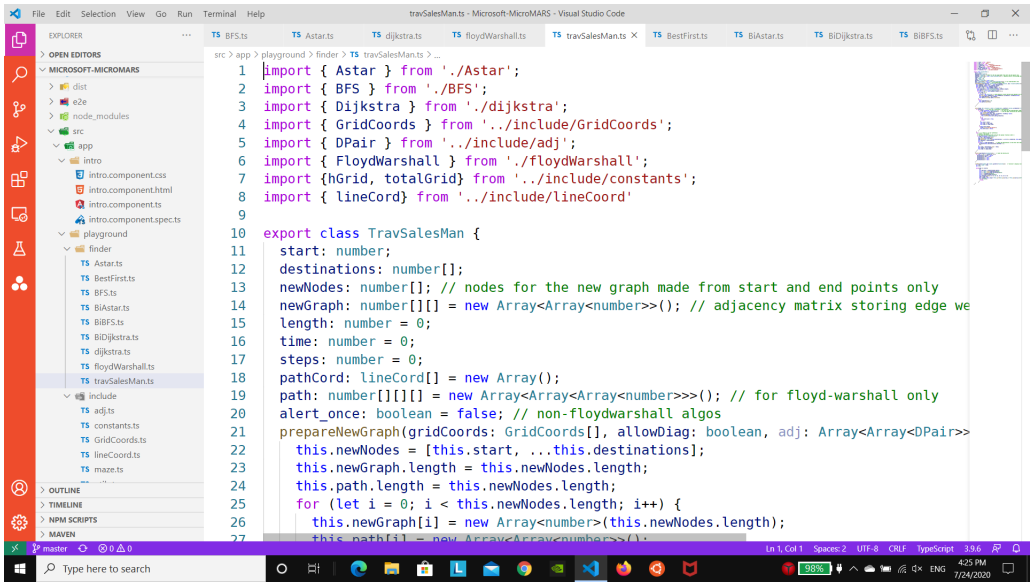
6.6.2 Multiple Destinations/Intermediate Stops:

We implemented this idea by making a class that has two methods, i.e *search* and *prepareNewGraph*. The former is a method that takes a standard shortest-path algorithm along with the user preference as input. When the user has some preference on the destinations, then we use the algorithm chosen by the user to iteratively find the shortest path starting from the source and ending in the last destination.

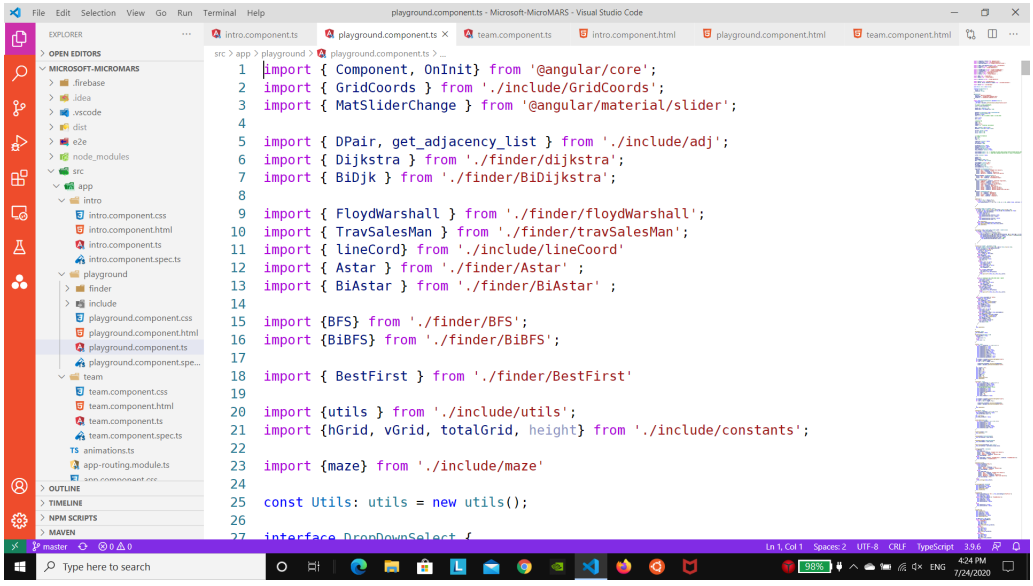
When there is no user preference, we make a new graph fully-connected graph with nodes as the source and destination points and the edges being the shortest paths found between each pair of them using Dijkstra's algorithm. We made this abstraction as the traveling salesman problem is NP-hard and the Floyd-Warshall algorithm which finds all-pairs shortest paths has an $O(n^3)$ run-time complexity. Then we use the Floyd Warshall algorithm on this new graph, find all-pairs shortest paths. After this, we then start from the source, find the nearest destination, shift the source to this destination, and then repeat this process until all the destinations have been covered. The details can be found in "***src/app/playground/finder/travSalesMan.ts***" and "***./src/app/playground/finder/floydWarshall.ts***"

6.6.3 Weighted Obstacles Gaussian-distributed Hills:

We implemented this idea by making a new TypeScript Interface data type called "**GridCoords**" which has some useful properties to handle the grid being displayed



to the client. We used this interface in the Angular component of our playground as an attribute to bing it to the HTML All the interactions of the client/user with the web app fill manifest as changes in the properties of this interface. We then used this interface as a primary input to all our algorithms. The code can be found in "***./src/app/playground/include/GridCoords.ts***"



6.6.4 Suggestion Pop-ups:

We implemented this by putting JavaScript alerts and key points of our algorithms. Whenever all the necessary settings are not provided by the user, the window throws an alert with a relevant message. In this way, we make sure that the algorithm works as the user expects, and for the user to get a crystal clear idea on using our web application.

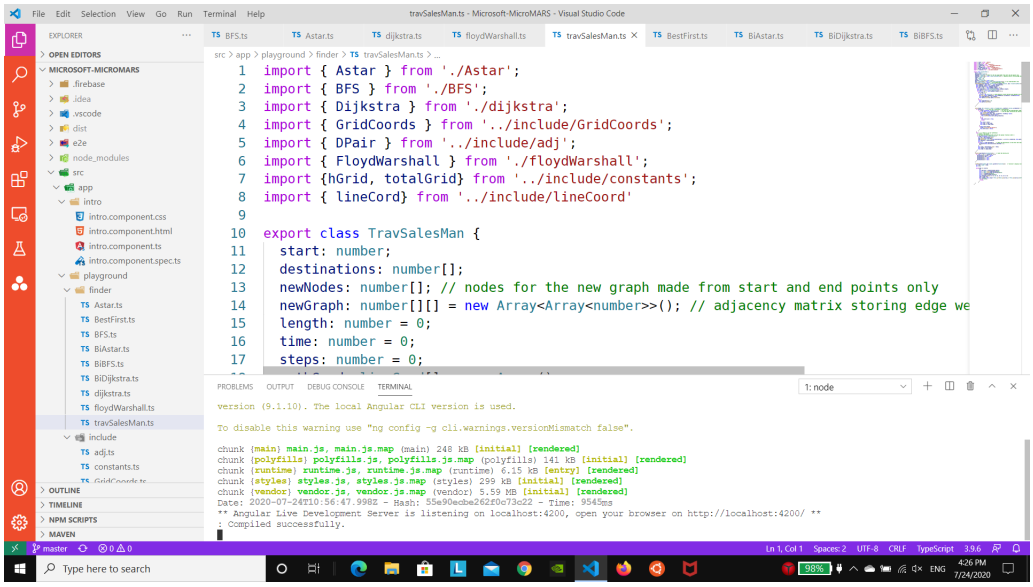


Figure 11: Code Snippets

6.7 Results we Achieved

- **Steps:** This denotes the total number of iterations the algorithm took.
- **Time:** This denotes the total run-time of the algorithm.
- **Length:** This denotes the length of the shortest path which is rendered on the grid.

7 WORKFLOW

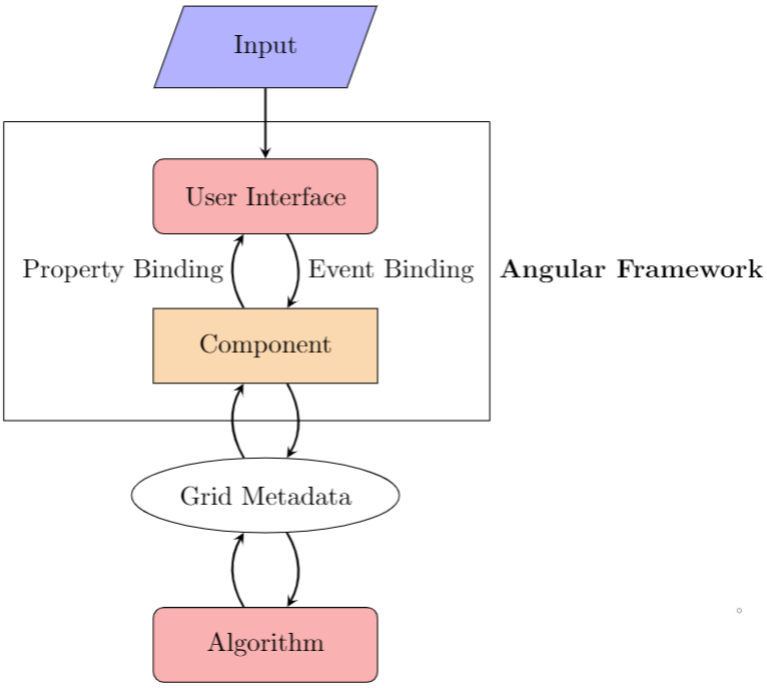


Figure 12: Low-level Workflow

- We developed the entire web application in the **Angular** framework. It is based on **TypeScript**, which enables us to use object-oriented programming.
- In Angular, a web app is made up of components called "**NgModules**", which facilitate the event binding from client-side (HTML) to server-side (Typescript) and property binding from server-side (TypeScript) to client-side (HTML) using the concept of template syntax similar to Jinja.
- All our algorithms are coded in TypeScript, and we made a component to handle the grid that is displayed to the client. An Angular component is nothing but a TypeScript class with some attributes and methods. We used the attributes of this class as inputs to the algorithms that we coded.
- Each algorithm has been coded separately as a TypeScript class in separate files. We also made a class for different maze generation algorithms.

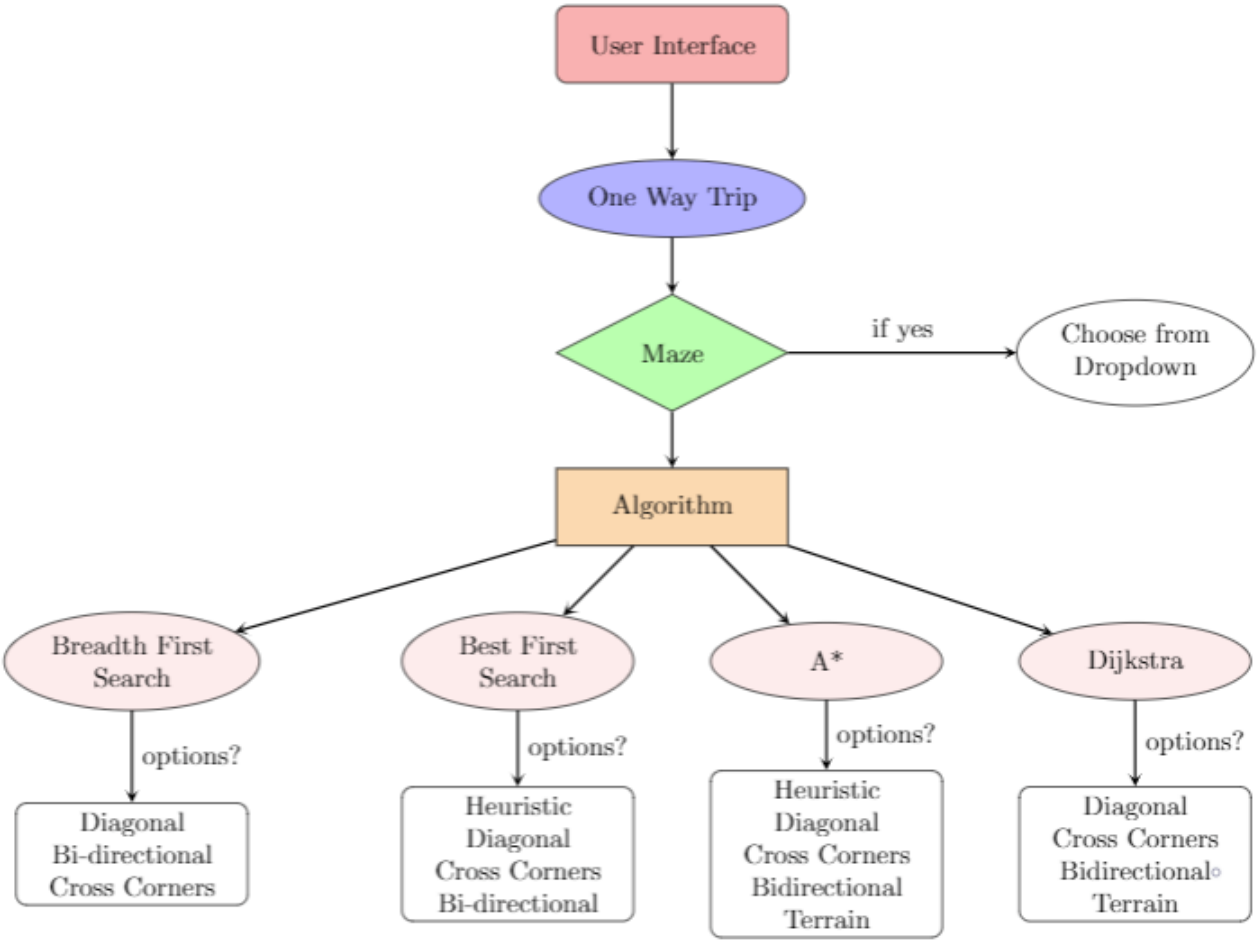


Figure 13: Problem Statement 1: One Way Workflow

Figure 14: Problem Statement 2: Intermediate Stops Workflow

8 TIMELINE WE FOLLOWED

- **June 17:** 'Introduction to the program' Webinar
- **June 18-19:** Team formation and choosing the project among the two options available
- **June 20-21:** Decided team name and completed the Team registration procedure on the website
- **June 22-23:** Decided on which all tools/software to use for the execution of the project and started knowing about them more
- **June 23:** Team meeting to brainstorm on how to approach the problem statement and what unique solutions we can bring to our own interface
- **June 24-28:** Worked on the team page and the Intro tab for our web app. Explored the different path-finding algorithms and heuristics and how we can

imbibe them in our interface

- **June 27:** Updated the GitHub URL for the team project on the website
- **June 28:** Meet with the mentor to talk about the further course of action and deliver the updates to him
- **June 28-29:** Read about graphs and adjacency matrices and how we can use them in our project and implemented the same
- **June 30-July 9:** Implemented the basic prototype of the interface adding the fundamental algorithms and customization
- **July 9:** Team meeting to go through the progress and decide about the further course of action, and also discuss some of the challenges in hand
- **July 10-17:** Started implementing the unique features while continuously optimizing the interface so that the former fit in it nicely
- **July 16, 18:** Meet with the mentor to discuss some of the challenges in hand and give a small demo of what all we achieved in the interface
- **July 17-20:** Fixed the minor bugs, worked on the suggestions by the mentor and added the finishing touches to the interface
- **July 19:** Team meeting to plan for what all was left, and for the final demo with the mentor
- **July 20-24:** Worked on the technical documentation, workflow charts and creating a tutorial video for our project
- **July 22:** Final meet with the mentor for a complete presentation of the interface, code and the technical document
- **July 22-24:** Finished some final touches and walked through the entire project for a final check. Deployed the final project and submitted the web-app URL on the website

9 CHALLENGES WE FACED

9.1 Responsiveness of Webpage:

We had some big and many small struggles in making our interface more presentable and responsive. We always thrived it to be as much user-friendly as it can be, and in the process while fixing one thing we ended up ruining the other. All of this required a lot of patience and determination, but in the end, we can proudly claim to achieve an interface more responsive and comfortable than what we aimed for.

9.2 Testing and Fixing Bugs:

Testing the algorithms we implemented, identifying the errors, finding the bugs in the code and the whole process of rectifying them all was a tough task altogether. But, this was the point when the bonding we shared as a team came into the picture and helped us achieve solutions for all the problems we had in hand, sailing through the arduous times, altogether, like a great team.

9.3 Implementing Maze Algorithms:

The idea of easing out the effort on the user's side by providing him/her with the option of choosing some already-in-place maze algorithms was really intriguing, but, in the pursuit of making this happen, we loaded ourselves with the not so easy extra effort that goes into implementing these maze algorithms and ensuring that they work well with our path-finding measures and are also easily customizable.

9.4 The Dilemmas of Obstacles:

The addition of weighted obstacles to our interface made it all the more relevant to the theme we were targeting, but all of this came with its own dilemmas. One of which was to decide whether we want to keep it a relative or an absolute calculation while moving from one grid point to another. The idea of keeping the calculation sounded more reasonable but resulted in counter-intuitive results, as employing the path-finding algorithms on this calculation rendered the rover to stay on the high obstacle itself to avoid the relative change. On the other hand, absolute calculations biased the rover to stay on the plane ground and not expense on the obstacles even if it can result in a more cost-effective path eventually.

Also, another challenge that we faced was that we had no particular standard or reference point for deciding on the difficulty or rigidity factor that we would eventually associate with the varied obstacles and how all of it will manifest in the already existing framework of the interface.

10 SCOPE OF IMPROVEMENT

10.1 Algorithm Visualizer:

Currently, there is no proper way to visualize how our algorithm is working in the framework you design and input as a problem statement. We can employ certain UI components that will interpret commands into visualizations and will subsequently help us report to the user as to how our algorithms are working in deducing the results they deliver.

10.2 Improvements in Multiple Destinations Problem:

10.2.1 Bi-directional Search:

Currently, there is no option of implementing bi-directional search in the multiple destination problem for the case where the user has no preference among the intermediate stops.

10.2.2 Terrain Options:

The topology accommodating feature of terrain and Gaussian distributed hills are not implemented for the multiple destination problem as of now and can be con-

sidered to allow the user to exercise more freedom in developing his own problem structure.

10.3 Improvement in Gaussian-distributed hills:

As a user, you exercise full liberty in adding and deleting the obstacles as and when needed, even as individual units. But this extended freedom is not entirely available in the case of Gaussian-distributed hills. To delete a particular hill, you need to use the option of 'Clear Grid'. However, introducing the feature of removing the hills individually as and when needed will help the user rectify his mistakes fast and will allow more efficient customization.

11 CONCLUDING STATEMENT

In the end, I would like to say it has been a challenging yet a great experience for all of us. We had our little share of struggles, but all of the learning that we gained in the process was totally worth it. We had a great time learning more about web-development, picking up graph theory from scratch, exploring more about Artificial Intelligence and its multifarious applications. We, once again thank our mentor who has been really enthusiastic and supportive of all that we tried and wished to explore. We thank **Microsoft** for this opportunity which got us busy in this never-ending quarantine and even close among our friends, making it the best quarantine memory so far.

We, **MicroMARS 42**, hope that you have an amazing time playing in our playground arena and that the web-app proves to be very beneficial to those using it. We hope our interface is able to provide solutions to some of the problems that people with a goal to establish life on the red planet may face and help them move a step further in their goal.