Microsoft Engage Mentorship Program 2020

Meeta Malviya
Fathima Zarin Faizal
Team: Batman and Robin
Mentor: Vinay Kotakonda

IIT Bombay

July 2020
URL: https://meeta14.github.io/pf_mars/
GitHub link: https://github.com/Meeta14/pf_mars

Contents

1 Introduction

This project is our unique take on the “Navigate the Mars rover” project. We have implemented a pathfinding visualizer website that finds the shortest path between two points for the Mars Rover to take.

2 Instructions to use

- Select terrain from ‘Set Terrain’ panel and click within the white grid and drag your mouse to draw obstacles/hills/valleys.
- Select number of destinations from the same panel.
- Drag the green node to set the start position.
- Drag the red node to set the end position.
- Choose an algorithm from the right-hand panel.
- Choose weights/ heuristic/ don’t cross corners options from the panel
- Click Start Search in the lower-right corner to start the animation.

3 Graph Traversal Algorithms

We have implemented the following graph traversal algorithms:-

- A* star
- Dijkstra’s
- Best First
- Breadth First

3.1 A* Search

A-star search is an algorithm which finds the most optimal path between two nodes. It uses two properties and tries to minimize sum of those properties. // properties are :- g(n) = the movement cost to move from the starting point to a given square on the grid
h(n) = the estimated movement cost to move from that given square on the grid to the final destination, called heuristic.
$f(n) = g(n) + h(n)$

Types of Heuristic

Different types of heuristics that we have implemented:-
The destination node is refereed as (a, b) where a represents destination node's 'x' coordinate and b represents destination node's 'y' coordinate.
The current node is refereed as (x, y) where a represents current node's 'x' coordinate and b represents current node's 'y' coordinate.

Manhattan

$$h(x, y) = |(x - a)| + |(y - b)|$$

Euclidean

$$h(x, y) = \sqrt{(x - a)^2 + (y - b)^2}$$

Diagonal

$$h(x, y) = Max(|(x - a)|, \ |(y - b)|)$$

Octile

$$h(x, y) = Max(|(x - a)|, \ |(y - b)|) + (\sqrt{2} - 1) \times Min(|(x - a)|, \ |(y - b)|)$$

Pseudo code for A star search:-

```
// A* Search Algorithm
1.  Initialize the open list

2.  Initialize the closed list put the starting node on the open
    list

3.  while the open list is not empty
    a) find the node with the least f on
       the open list, call it "q"

    b) pop q off the open list

    c) generate q's 8 successors and set their
       parents to q

    d) for each successor
         i) if successor is the goal, stop search
           successor.g = q.g + distance between
                               successor and q
           successor.h = distance from goal to
           successor (This can be done using many
           ways, we will discuss three heuristics-
           Manhattan, Diagonal and Euclidean
           Heuristics)

           successor.f = successor.g + successor.h
```

```

        ii) if a node with the same position as
            successor is in the OPEN list which has a
           lower f than successor, skip this successor

        iii) if a node with the same position as
            successor  is in the CLOSED list which has
            a lower f than successor, skip this successor
            otherwise, add  the node to the open list
     end (for loop)

    e) push q on the closed list
    end (while loop)
```

3.2 Conditions for heuristics

- At one extreme, if h(n) is 0, then only g(n) plays a role, and A* turns into Dijkstra's Algorithm, which is guaranteed to find a shortest path.
- If h(n) is always lower than (or equal to) the cost of moving from n to the goal, then A* is guaranteed to find a shortest path. The lower h(n) is, the more node A* expands, making it slower.
- If h(n) is exactly equal to the cost of moving from n to the goal, then A* will only follow the best path and never expand anything else, making it very fast. Although you can't make this happen in all cases, you can make it exact in some special cases. It's nice to know that given perfect information, A* will behave perfectly.
- If h(n) is sometimes greater than the cost of moving from n to the goal, then A* is not guaranteed to find a shortest path, but it can run faster.
- At the other extreme, if h(n) is very high relative to g(n), then only h(n) plays a role, and A* turns into Greedy Best-First-Search.

3.3 Best First Search

This algorithm is functionally same as A-star Search, but this algorithm finds path based on least heuristic only.
Hence, we use A-star algorithm with a slight modification , which is, we change h(n) to the following

$$h(n) = g(n) + weight \times h(n)$$

Here weight is the user-input variable set to 1000 by default. The user can decide how much to uplift heuristic in competition to g(n)

3.4 Dijkstra's Algorithm

This algorithm is also functionally same as A-star Search, but this algorithm finds path based on least g(n) only.
There isn't any heuristic function involved in this search. We set h(n) = 0; henceforth g(n) = f(n).

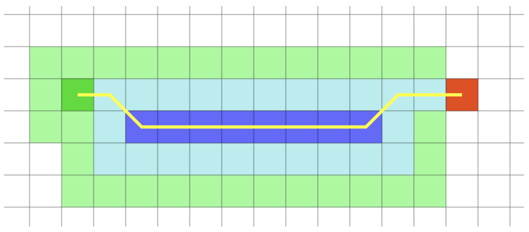
3.5 Breadth-First Search

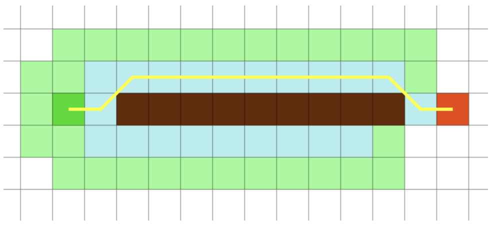
BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.
Pseudo code for A star search:-

```
BFS (G, s)                   //Where G is the graph and s is the source node
      let Q be queue.
      Q.enqueue( s ) //Inserting s in queue until all its neighbour vertices are marked.

      mark s as visited.
      while ( Q is not empty)
           //Removing that vertex from queue,whose neighbour will be visited now
           v  =  Q.dequeue( )

          //processing all the neighbours of v
          for all neighbours w of v in Graph G
               if w is not visited
                        Q.enqueue( w )             //Stores w in Q to further visit its neighbour
                        mark w as visited.
```

4 Terrain

The Martian surface is not smooth and includes different kinds geographies. The tallest mountain in the Solar System(Maunea Kea) is on Mars. Hence we felt the need to incorporate geological differences which will definitely play a huge role in finding shortest path. We have included three kinds of terrain: hill, flat land and valley. Hill is the hardest terrain to pass through, flat land second highest and valley is the easiest terrain. We implemented this by putting weights for the edges between nodes. We have allotted 15 units for hill, 5 units for flat land and 1 unit for valley. If two adjacent nodes were of the same type of terrain, the above mentioned weights are used for the edge in between. If the two adjacent nodes were of different kinds of terrain, the absolute difference of the two weights is used.
Among the algorithms that we have implemented, A* star search and Dijkstra's work for weighted graphs and hence takes into account the effect of terrain, while Breadth First Search and Best First Search do not work for terrain.

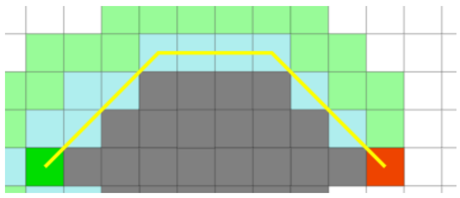

(a) Blue colour denotes valley

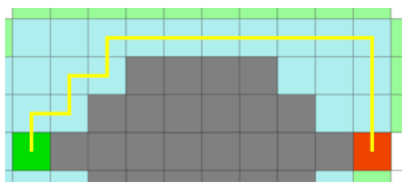

(b) Brown colour denotes hill

5 Other Options

5.1 Diagonal Neighbours

In the grid, a node (denoted visually by a square) is connected to eight neighbours (each direction).
Inside Grid.js there is a getNeighbour function that uses a hyper parameter which is set true if we want to include all 8 neighbours and false if we only want to include 4 neighbours namely north, east, west, south.

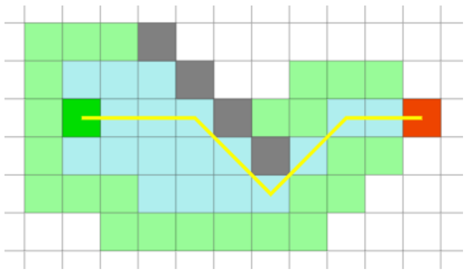

(a) Allow diagonal

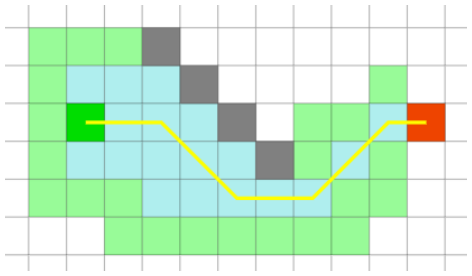

(b) Don't allow diagonal

5.2 Don't Cross Corners

For this we need to maintain and check one more condition before appending into neighbours list.
s0, s1, s2, s3 represents east, west, south, north direction respectively are initially set to false and is set to true if the neighbour is included in the neighbours list.
d0, d1, d2, d3 represent north east, south east, south west and north west respectively// code included to set this option in get neighbours:-

```
if(diagonal==true){
if(dont_cross_corners){
  d0 = s3 && s0;
  d1 = s0 && s1;
  d2 = s1 && s2;
  d3 = s2 && s3;}
else{
  d0 = s3 || s0;
  d1 = s0 || s1;
  d2 = s1 || s2;
  d3 = s2 || s3;}
```


(a) Cross corners


(b) Cross corners

6 Bidirectional

The algorithm's bidirectional search is functionally same as it's normal implementation just that it maintains two open list and searches for it's respective end node and ends search if the current node in same for both the open list.

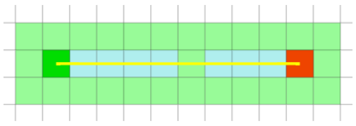


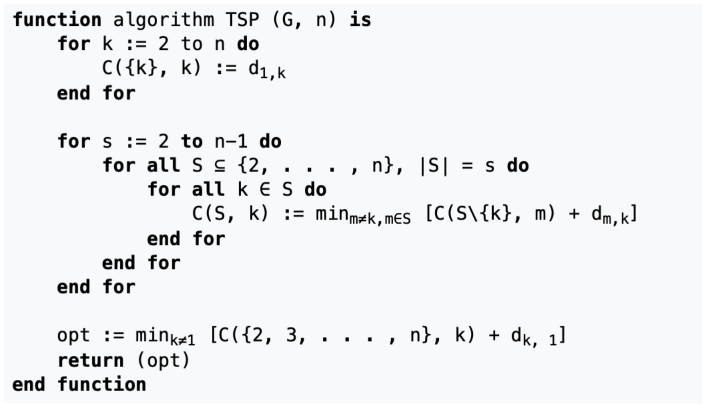
Figure 4: Bidirectional

7 Number Of Destinations

We have incorporated multiple destinations in two ways:

- User gets to order destination nodes in the order they want the optimized path to visit them
- Algorithm finds shortest path from source node that passes through each end node exactly once

The first case is fairly simple, only involves finding paths using above mentioned graph traversal algorithms in the correct order.
The second case is basically the travelling salesman problem. All paths between the source node and each end node is found using the selected algorithm. Then a dynamic programming algorithm called Bellman-Held-Kerp algorithm is applied. The travelling salesman problem is an NP hard problem. The brute force method of generating all permutations of nodes and finding shortest path would have complexity $O(n!)$ where n is the number of end nodes. The dynamic programming approach that we have followed is slightly faster with complexity $O(2^n\sqrt{n})$. The algorithm involves finding subsets of all end nodes which has been achieved using bit masking.



```
function algorithm TSP (G, n) is
    for k := 2 to n do
        C({k}, k) := $d_{1,k}$
    end for

    for s := 2 to n-1 do
        for all S ⊆ {2, . . . , n}, |S| = s do
            for all k ∈ S do
                C(S, k) := $\min_{m \neq k, m \in S}$ [C(S\{k}, m) + $d_{m,k}$]
            end for
        end for
    end for

    opt := $\min_{k \neq 1}$ [C({2, 3, . . . , n}, k) + $d_{k,1}$]
    return (opt)
end function
```

Figure 5: Bellman-Herd-Karp algorithm
Source: Wikipedia

8 Mazes

We have incorporated mazes in our implementation. Mazes make it easier to set pre-defined blocked nodes for the Martian terrain.
We have implemented two maze algorithms:-

- Depth First Search
- Prim's Algorithm

9 Scope for further modifications

- **No custom UI**, since we are a two member team, we focused more on adding new features rather than making the UI look better
- **The terrain options are not customizable**, the user cannot set his own weights for the terrain
- **Terrain options not realistic enough**, same weight is given to ascending a hill and descending a hill, gradients cannot be set for the terrain to show gradual inclines or declines, Martian surface includes craters which have not been included

10 References

- Breadth first search: https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/
- Wikipedia
- www.geeksforgeeks.org
- Mazes:https://hurna.io/academy/algorithms/maze_generator/index.html

- Heuristics: http://theory.stanford.edu/∼amitp/GameProgramming/Heuristics.html