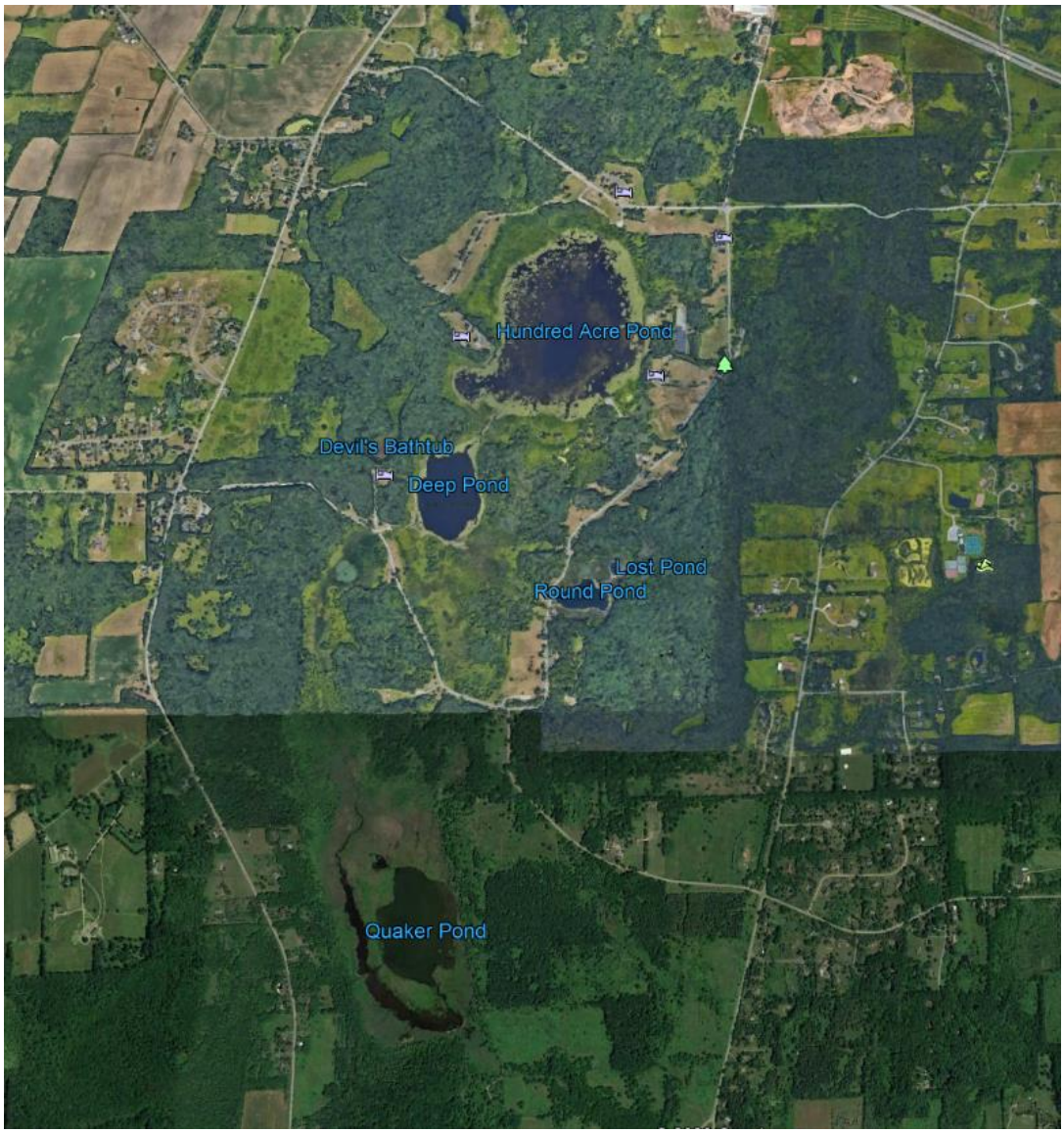
Shortest path on a real-world 3D map

This project finds the shortest path between the start and end points on a 3D terrain considering the changes due to different seasons.

The map I've chosen is of the Mendon Ponds Park of Rochester, NY.

This is a satellite image from Google Earth outlining the park, it's trees, and the ponds.



As we can see, it has a different elevation. It has deep ponds and small hills.

The elevations dataset is gathered from National Elevation Dataset.

Rochester has four seasons:

Summer, which is when these images were taken.

Fall, the land near any tree is cover with leaves.

Winter can get harsh, and most of the water bodies freeze.

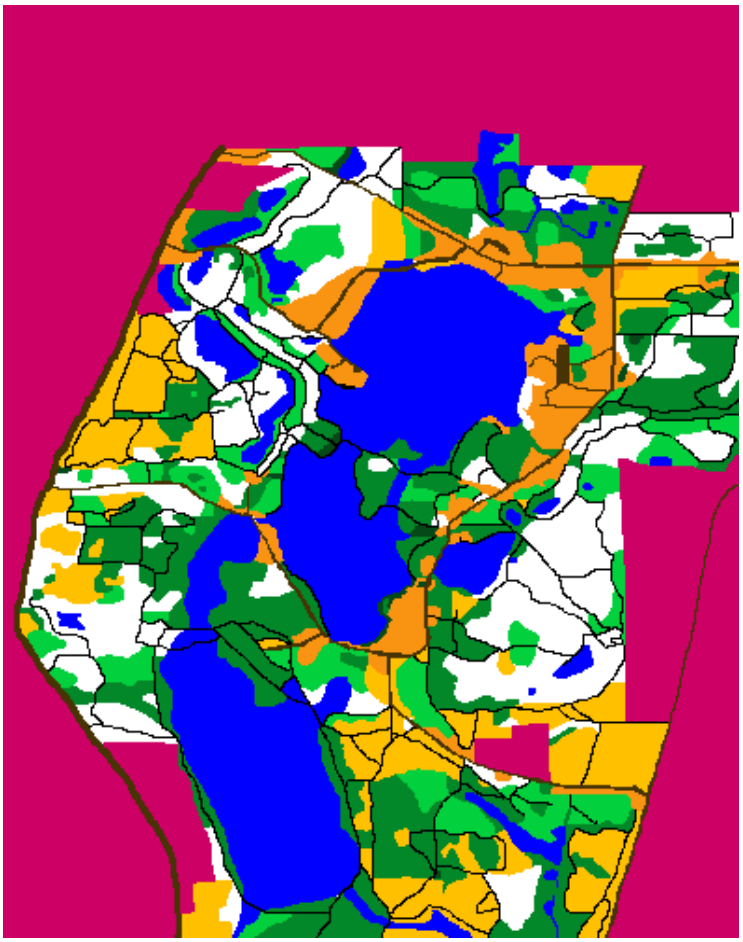
Spring, where the snow begins to melt, and it gets muddy.

Considering these scenarios, the map changes.

The park has various terrains; from open lands, rough meadows, forests, impassable vegetation, lake, paved roads. The movement speed will be different for different terrain. To make things easier, these different terrains are portrayed with colors on a map.

Terrain type	Color on map & movement speed
Open land	6 m/s
Rough meadow	5.5 m/s
Easy movement forest	4.5 m/s
Slow run forest	4 m/s
Walk forest	3 m/s
Impassible vegetation	0.05 m/s
Lake/Swamp/Marsh	1 m/s
Paved road	8 m/s
Footpath	7.5 m/s
Out of bounds	10^{-8} m/s
Ice	0.5 m/s
Mud	0.4 m/s

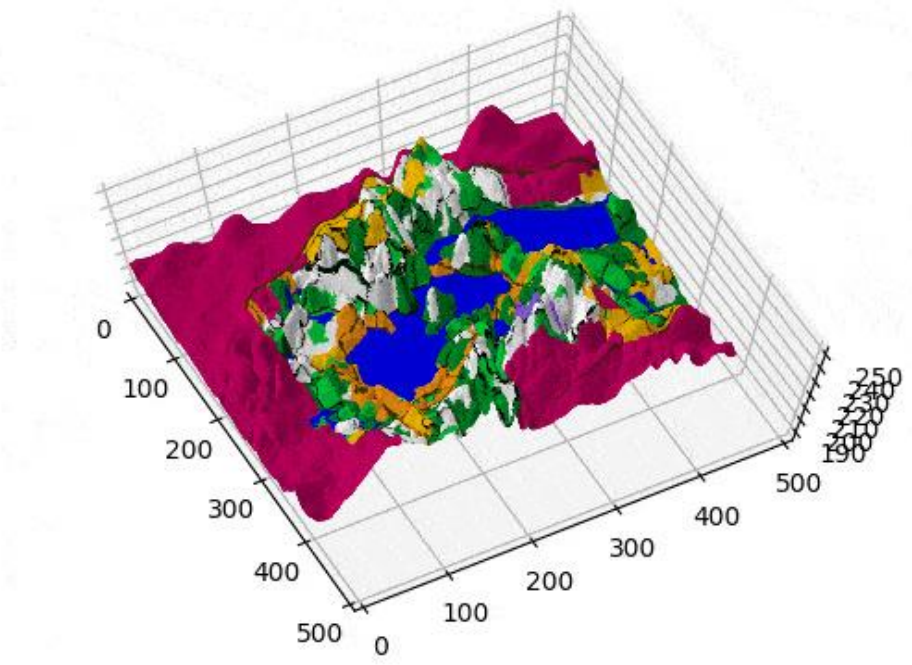
Summer (Base map)



This 395x500 map will be used to consider the various terrains. The text file mpp.txt is used for the elevation.

The interpixel distance to real world distance is 10.29 m in X-axis & 7.55 m in Y-axis.

Considering the colored map and elevations from the text file, this map is generated (3d_image.gif)



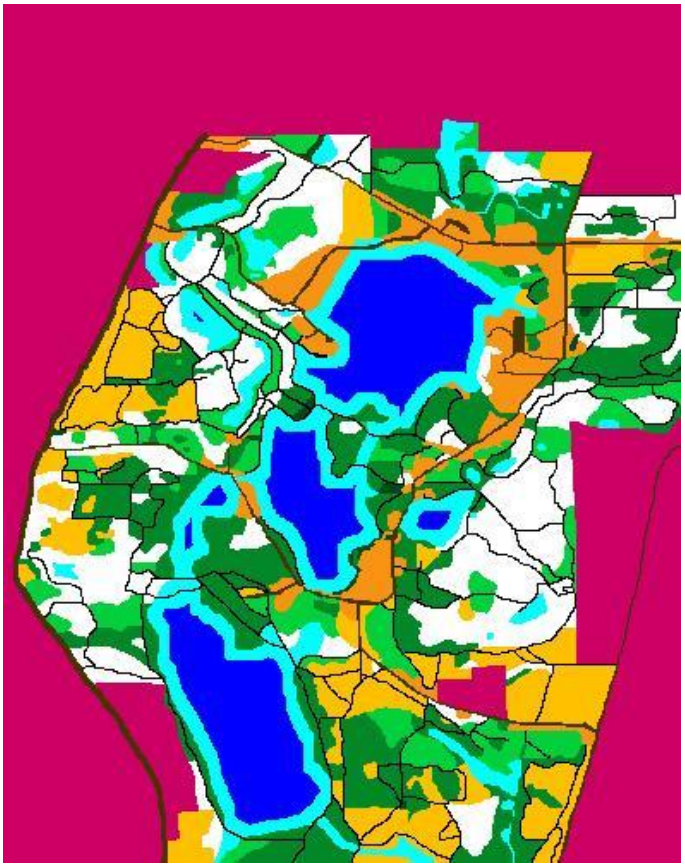
The code takes input as

$python3 path_find.py terrain.png elevations.txt check_points.txt <season_name> output_image_file_name

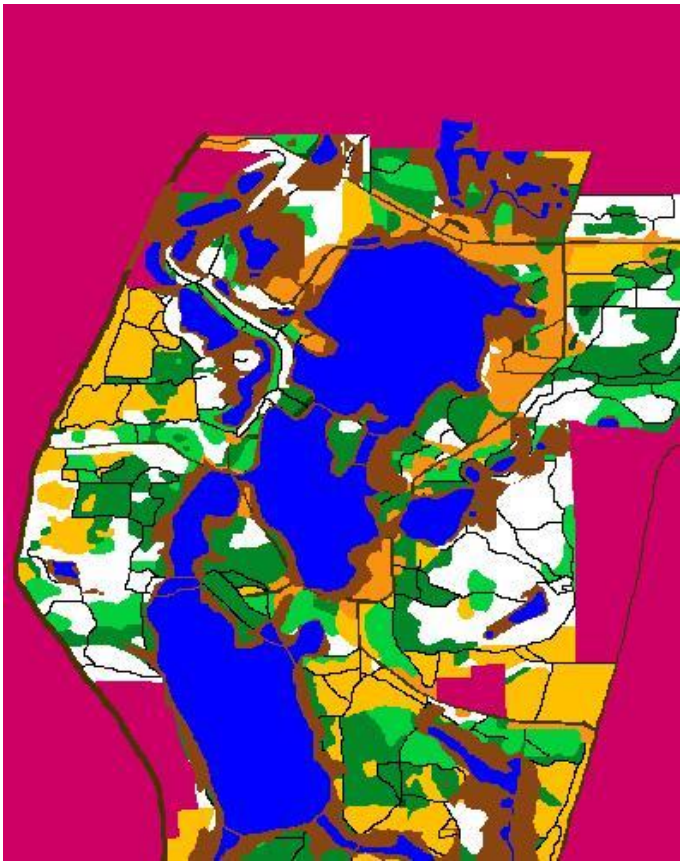
- **terrain.png** is the color-coded image for representing different vegetations.
- **elevations.txt** contains the elevation of each point corresponding to the pixel in the map
- **check_points.txt** contains the start point, all/no check points, end point.
- **<season_name>** can either be Winter, summer, fall, or spring.
- **output_image_file_name** is the name of the output image to be generated.

The seasons change the map in the following way:

In winter, a 7-pixel wide ice layer is formed on the water where the water meets land.

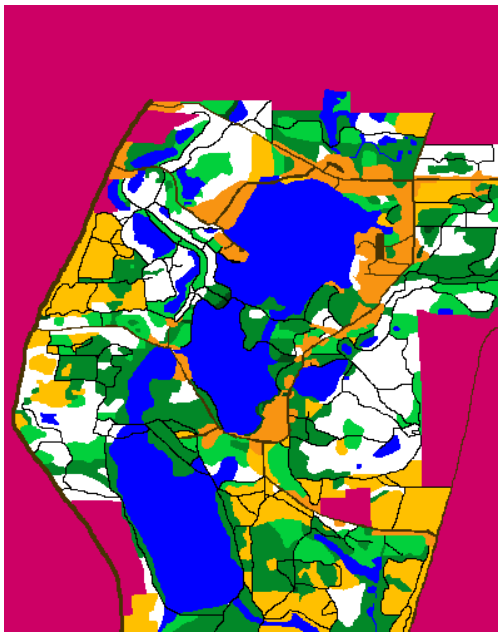


In Spring, all points within a 15-pixel radius of a waterbody and with less than 1-meter elevation from the neighboring waterbody is covered in mud due to all the ice/snow melting from the winter season.



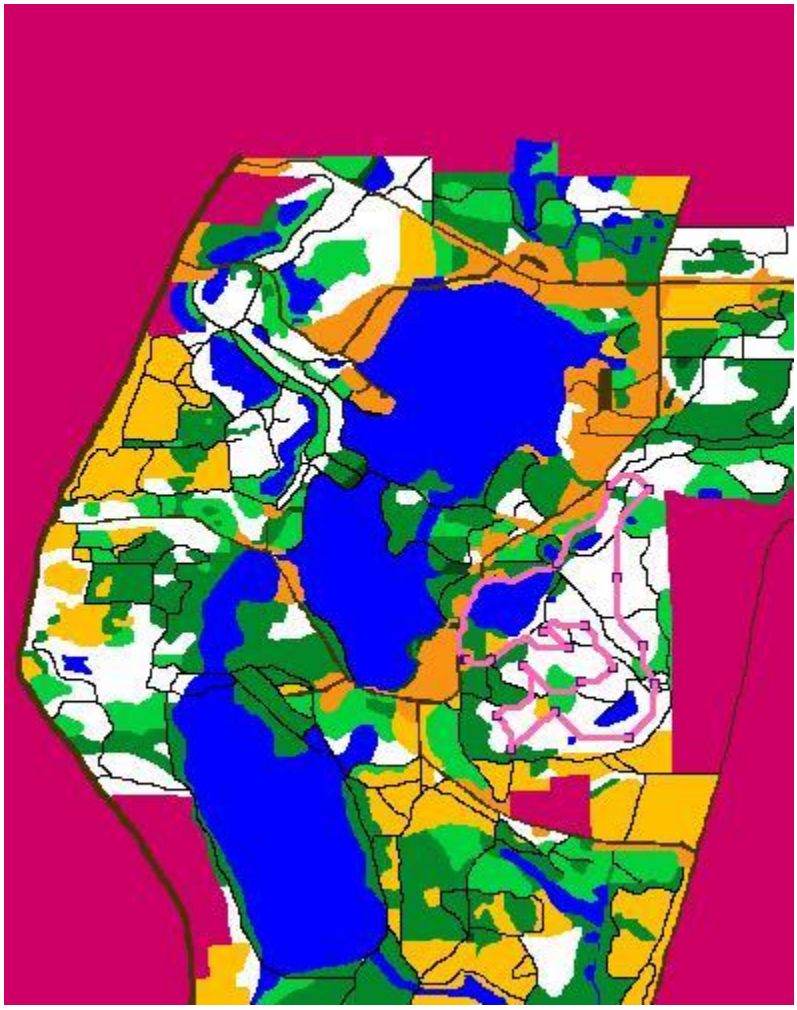
In Fall, the movement speed along the paths near the forested areas is decreased as the leaves have covered all the visible paths.

The base map is the summer season.



When using multiple checkpoints between start and end, the total distance covered, and path is displayed. The coordinates from the text-file (check_points.txt) are:

x	y
230	327
276	279
303	240
322	242
306	286
319	320
325	339
312	366
275	353
253	372
246	355
259	330
288	338
304	331
290	310
269	313
282	321
243	327
230	327



Here, the path between the individual check points is highlighted in light-pink and points are black dots.

To see how the algorithm works, let's assume only a start and end point coordinate.

Our goal is to reach the end point in least amount of time.

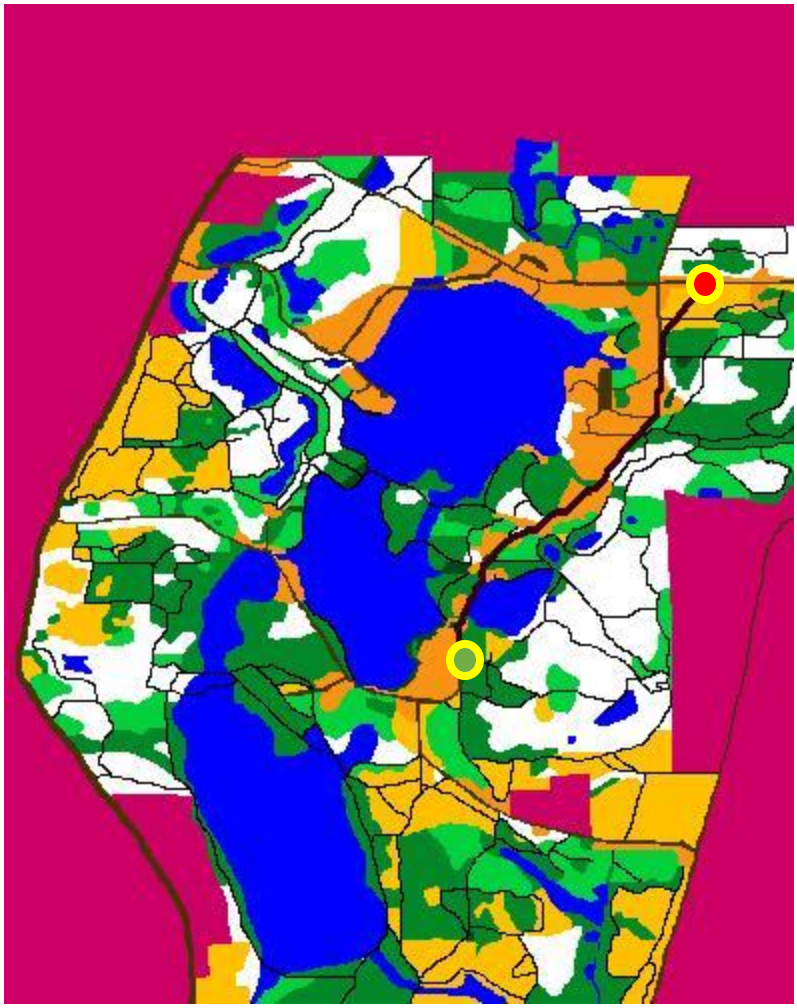
In this case, [230, 327] is the start point (big green point) & [350, 139] is the end point (big red point).

To see how the algorithm chooses this path:

1) All the neighboring points to the start point are considered.

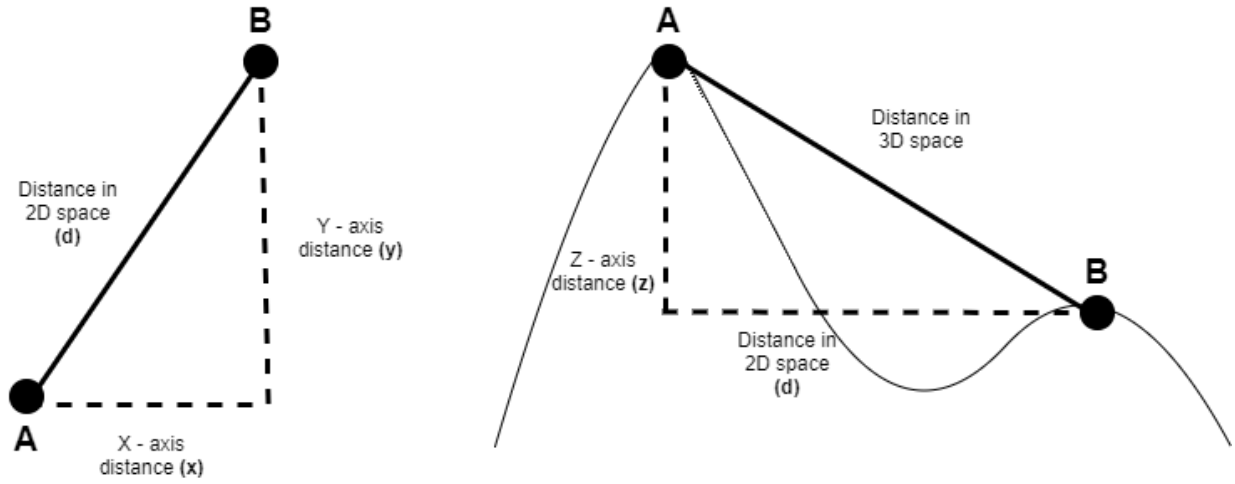
2) Now, we have 8 coordinates with their respective elevations.

3) As we already know the location of the end point, we can calculate a heuristic cost based on that.



Heuristic cost (h): The direct distance between the selected point and the destination point assuming a level and direct paved path between them.

We need this cost to basically guide the algorithm in the right direction.



We need the distance in 2D space and the maximum speed that is achievable on this map, hence we use those parameters to calculate the time required to go from **A** to **B**.

Distance (d) = $\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$

Speed = 8 m/s

Time taken = d/8 s

Path cost (g): We use the 3D distance, this time we need the actual time required to go from **A** to **B** considering the difference in elevations and terrains, the final time taken changes drastically.

Distance (D) = $\sqrt{(z_a - z_b)^2 + (d)^2}$

Speed = $[(\text{speed})_A + (\text{speed})_B]/2$

Hence, we get the path cost (g) and heuristic cost (h).

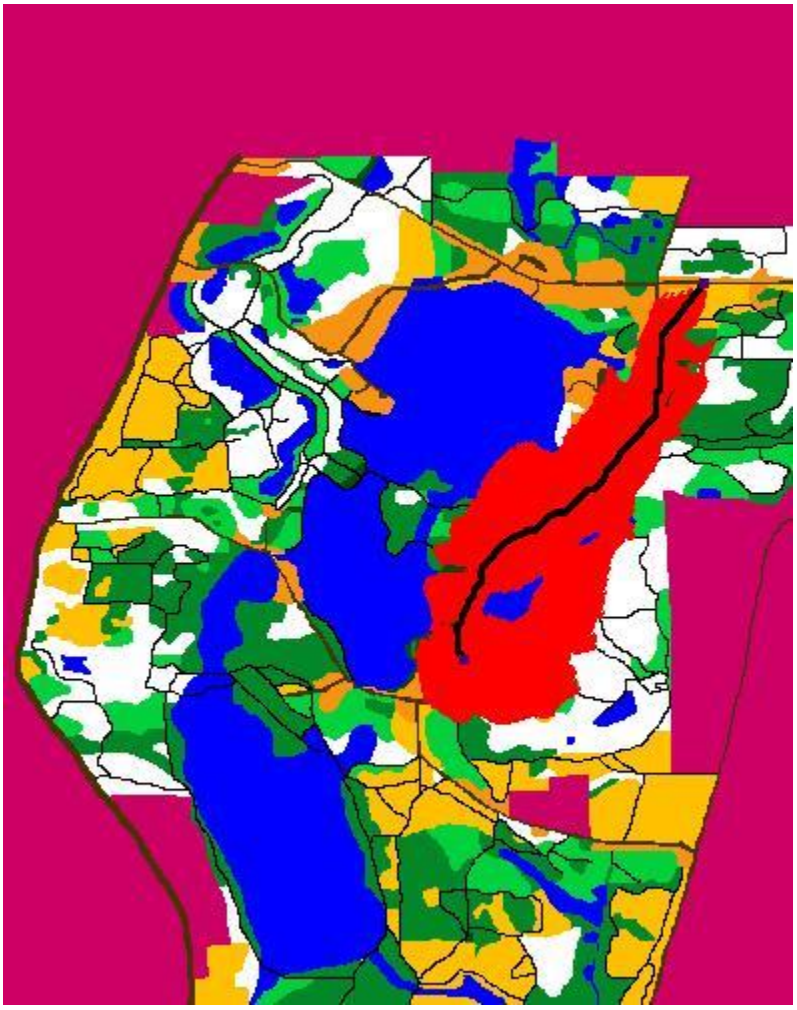
5) Now, we have a bunch of points in a queue with their respective function costs (f).

6) The point at the top of the queue will the point with the lowest function cost. We select that point and then considers its neighbors. This way we always consider the points with the best chance of being considered in the best path first.

7) This procedure is continued until we've reached the final point.

The algorithm does a basic best-first search; always considers the coordinates with the lowest cost (f = g + h). By doing an informed heuristic search like A* the answer might not be the best possible result, but it provides a good-enough result in the shortest time possible by traversing the least possible nodes/coordinates.

The red area indicates all the coordinates searched for going from point A to B.



As we can see, the heuristic function chosen does it's job by guiding the algorithm in the correct direction.

The distance travelled was 2200 m. It's the best path considering the distance travelled and the time taken to reach there.