EleNa - User Documentation

1. Overview:

EleNa (Elevation based navigation) is a tool perfect to find a route between any two points that suits your elevation preferences. It also allows you to choose the amount of deviation you would like to take to meet your elevation goals.

2. How to Run it?

2.1. Clone the Repository : https://github.com/willjeong204/Project-Elena
2.2. Go to terminal, and navigate to /Project-Elena
2.3. Run: $ ant
2.4. Then Run: $ ant Main

3. Who is it for?

Perfect for Hikers/Bikers or athletes who want to maximize their elevation gain during a workout routine.
Also for those with special mobility needs who wish to choose the path with minimal elevation to reach their destination.

4. Features in EleNA:

4.1. UI (Satellite and Mapr view +elevation click feature):

EleNa comes with everybody's favourite Google maps like looking map with two modes to switch between, Map and Satellite (Top Right Corner) Toggle between these two views to display the map in two different styles.
You can also click anywhere on the map to see the elevation at those points.

4.2. Buttons and what they do:

4.2.1. Starting Point and Ending Point are two text fields that take in your source and destination. Inputs can only be Physical address within the current map scope. A list of some examples have been mentioned later in the document.
4.2.2. Minimum Elevation and Maximum Elevation are two buttons, that are mutually exclusive. Use these buttons to set your elevation preferences.
4.2.3. Clear button clears the Starting point and Ending point fields. It also clears all routes from the map. It resets the elevation preference as well.
4.2.4. The "GO" button: when clicked,starts looking for a route according to your preferences that are set. It stays lit until a route is returned.

4.2.5. Add to Fav and its benefits: Once the route is generated, the “Add to Fav” button is enabled. With this button you can add the route to your favourites to be able to load it faster the next time.

4.2.6. Changing API key: A text box under deviation, that helps you easily swap in a new key if the application runs out of API calls. Directions to generate a new key are described in section [6.1]

4.3. Compare routes:

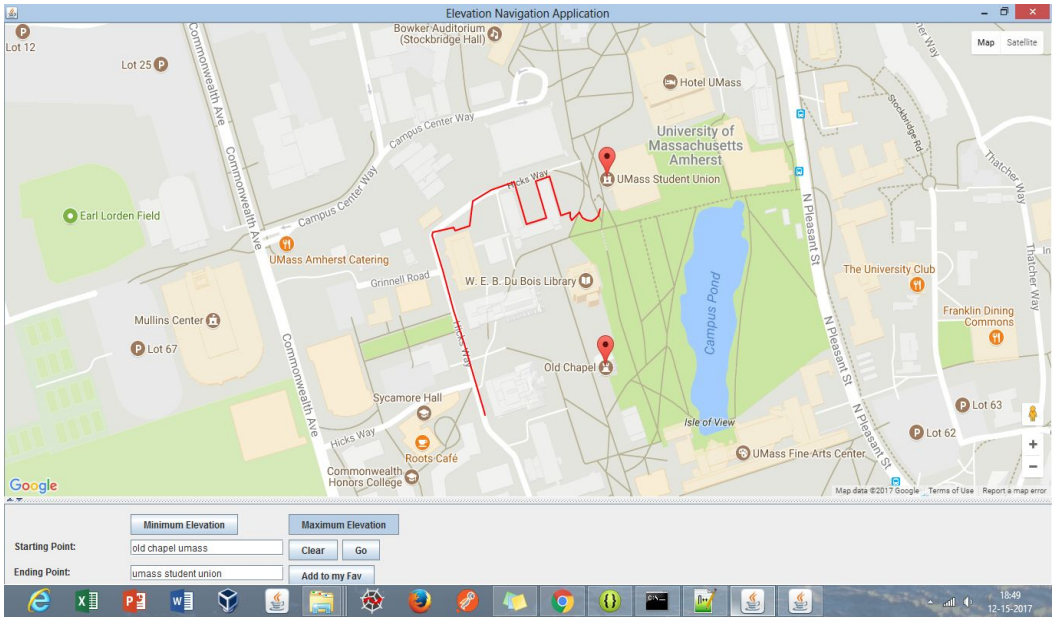
Change preferences and hit go over these choices without clearing the fields and the routes will populate one over the other in different colors for you to eyeball a quick comparison.

5. **Limitations with the Application:** As of now, the application is limited to a section of Umass. This is mainly because of the number of free API calls we get for Google distance Matrix API. If we choose the pay, we can scale this application to larger locations. We can also try caching the distance matrix to save the number of API calls and this would be one of the first things to try in the future.

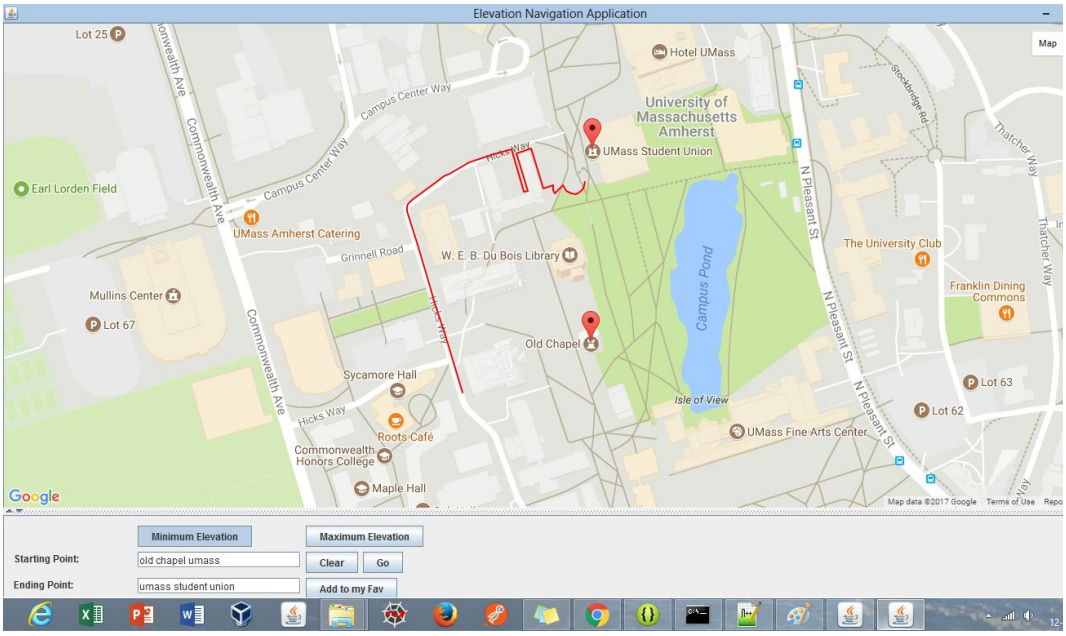
6. Some Examples

One among the many examples we tested is by giving starting point as old chapel umass and end point as umass student union. We tested it for the following features:

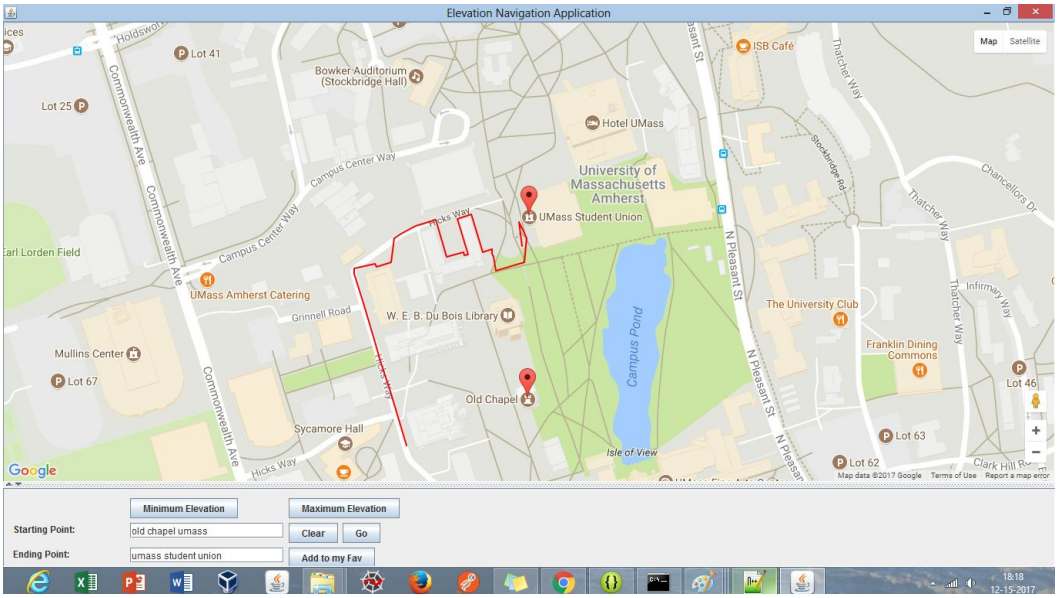
1) **Maximum Elevation:**



2) **Minimum Elevation**:



3) **Shortest Path**



The map under consideration is restricted to a section in Umass. Our application can be tested for the following places:

1) Old chapel, umass
2) Sycamore hall, umass
3) Umass student union
4) Umass Amherst Catering
5) Umass outing club
6) Hotel umass
7) Du Bois library
8) Facilities planning, umass
9) Roots cafe, umass
10) South college, Amherst
11) Machmer hall
12) Fine arts center
13) Thompson hall
14) Umass Amherst career services
15) Maple hall
16) Oak hall
17) Elm hall
18) Birch hall
19) College of humanities & fine arts
20) Umass conference services

7. Known issues and Bug fixes:

7.1. API key issue:

You will need several API keys to run this application, most of which will not run out, so the ones we have loaded in should be just fine. However, the Google Distance Matrix API has a very small limit to the number of elements it can return and might run out.
In such a scenario, please generate a new key from https://developers.google.com/maps/documentation/distance-matrix/get-api-key. And plug it into the text field that says API Key.

7.2. Entering an invalid location:

An invalid input or an input out of scope from our map will cause the location text to be cleared. Please re-enter correct places in this case. Also note, our application only takes in physical addresses for input and not (lat,lng) coordinates. This could be a value addition in future work.

7.3. Approximation issue:

Since we are overlapping two different APIs to get the best of both worlds, (Google for its View and interface and OSM for underlying routing), we have some approximation issues. There might be a slight noticeable drift from the start and end point Markers and the begin and end of the route. The same with the route following the paths that are displayed in google. Not every path in google translates to OSM and vice versa. This would be one major point for any future work to concentrate upon.

8. Some details about the routing Algorithm:

For routing, we have chosen A* with a modification to its score/cost function. This choice is due to its performance and Accuracy. Since we are using google API to get the distances between two points to populate a distance matrix, we are limited on the number of API calls and requests. We found that A* outperforms Dijkstra by a huge margin in the number of nodes it expands and hence makes fewer API calls

A* is an informed search algorithm, or a best-first search, meaning that it solves problems by searching among all possible paths to the solution (goal) for the one that incurs the smallest cost (least distance travelled, shortest time, etc.), and among these paths it first considers the ones that appear to lead most quickly to the solution. It is formulated in terms of weighted graphs: starting from a specific node of a graph, it constructs a tree of paths starting from that node, expanding paths one step at a time, until one of its paths ends at the predetermined goal node.

The original cost function to find the shortest route is given by :
$f(n) = g(n) + h(n)$
$g(n)$ - distance from the source to the current node ‘n’
$h(n)$ - distance from the source to the current node ‘n’ to destination node

Now this function has to be modified in to the following two rules :
In case of Minimizing the elevation:
$f(n) = d * (g(n) + h(n)) + e(n)$

In case of Maximizing the elevation:
$f(n) = d* (g(n) + h(n))) + e'(n)$

Where,
g(n) - distance from the source to the current node ‘n’
h(n) - distance from the source to the current node ‘n’ to destination node
d - is the deviation factor. Ex: if the deviation allowed is 10% from the shortest route, d will be set to 0.9.
e(n) - elevation at the point ‘n’
e’(n) - compliment of the elevation at point ‘n’, i.e, e’(n) = MAX_ELEVATION(in the area) - e(n)

The Algorithm makes its choices based on the above equations at every point. This enables us to get a maximum or minimum elevation gain routes.