2022

Don’t Beep At Me: Using Google Maps APIs to Reduce Driving Anxiety

Daniel Chechelnitsky
Macalester College, chechelnitskd@gmail.com

Macalester College — Saint Paul, MN

Department of Mathematics, Statistics, and Computer Science

Honors Thesis, Spring 2022

Don’t Beep At Me: Using Google Maps APIs to Reduce Driving Anxiety

Completed by:	Daniel Chechelnitsky
Advised by:	Prof. Dr. Susan Fox
Honors Committee Members:	Prof. Dr. David Shuman
	Prof. Dr. Rivi Handler-Spitz

Abstract

Stress while driving is a significant issue that causes automobile incidents. Along with the physical injuries, there is often baggage and trauma associated with these accidents. Wearable health monitoring technology, like Smartwatches, has a real possibility to help people further understand the stress inducing processes of driving. Thus to help with this issue, I propose a Google Maps app extension called: "Don't Beep At Me". This project creates a map that is layered by heart rate instead of speed limit and has great potential to be useful for reducing driving anxiety.

Table of Contents

0 Acknowledgements

I would like to thank my supervisor for this thesis Professor Dr. Susan Fox for extending her careful guidance, expertise, and for motivating me in all stages of this project. I would also like to thank Professor Dr. David Shuman and Professor Dr. Rivi Handler-Spitz for taking the time to be on the committee, participate as readers, and to help polish and discuss this project. Lastly, I would like to thank my COMP 221 Project Partner Thy Nguyen '21 and my COMP 221 Professor Dr. Lauren Milne both for teaching and helping me further understand the structure of path-finding algorithms like A* that are used in the back-end of Google Maps API.

1 Introduction

1.1 Inspiration & Background

For this project, I designed an application to recommend low-stress driving routes. I implemented the front end of the app and created a "proof of concept" implementation for the pathfinding part of the task. Throughout this process I also discovered the existing limitations of the Google Directions API that prevent a complete app from being possible at this time.

I began this project knowing that I wanted to build something meaningful and helpful for many people that also connects with me on a personal level. I also knew that I wanted this to be something that has the potential to be scalable and widely used by many people. After having spent the summer of 2021 taking an online Google Cloud course where I learned about some of the various tools that it provides, I was drawn to see how I could use them to solve a real world problem.

One of the Google Cloud libraries that quickly grabbed my attention were the collection of APIs in the Google Maps Platform. Upon trying out a few of them, I noticed that the API's tools have potential to be used in a variety of ways outside the scope of that even Google uses in their own Google Maps app, such as in the context of mobile health or incorporation with another algorithm. Given my past experience with pathfinding algorithms from my Computer Science courses, I was specifically curious about how the Directions API parameters can be changed for what makes a path efficient or optimal. This can be used to generate paths based on a different criteria than "shortest time".

By plugging in a starting location and destination into the Google Maps app, the user can see what the fastest route is along with 2 to 3 alternate routes that might be similar or slightly less efficient. The user can even give feedback and select a less optimal calculated path that will then override the calculation and become the preferred path. From my own use, I have noticed that sometimes the path that Google Maps outputs can even take into account other obstacles like road construction and extreme weather.

Around this time last summer I also injured my right leg, which meant I was not able to drive for an extended period of time. By the time I had recovered in the early Fall, it had been several months since I had last driven. Upon getting back into the driver's seat, I would have spontaneous waves of panic rush through my entire body. I felt like I was a novice driver all over again, but somehow even worse because now the things stressing me out seemed to occur without pattern or reason. Sometimes these waves cause my hands to shake uncontrollably so much so that I would have to pull over to calm myself down. These negative driving experiences in turn lead me to my main question for this project: can we create a system to find routes from one place to another that minimize stress for the driver? I would soon discover I am far from the only one that has this question.

1.2 Exploring Causes & Solutions for Driving Anxiety

Although I did not feel it was necessary to take a formal survey, many people in my life who drive have also spoken with me on how driving can sometimes be a source of stress. Moreso, existing research done on driving anxiety clearly shows that this is a significant and relevant real world problem that impacts thousands of people's lives daily [1]. Many alternatives to driving exist: biking, moped, walking, but even so there is still a significant percent of the population that uses and needs cars for their day-to-day activities. I'd argue even if this wasn't the case there is still stress that can be associated with road transportation of any form. With this being such a widespread issue, I researched work on driving stress and ways to alleviate it.

In the research I studied, I found that there are multiple stressors of driving that require persistent and prolonged solutions: presence of negative driving-experiences, driving-related phobias, and PTSD related to driving automobiles [2]. Most of these are very generalized problems that require multiple interventions and as such there is no end-all solution to solving driving stress. Many of these stressors require careful and individual treatment such as counseling therapy and/or medical expertise. An app that could produce lower-stress driving routes can't solve driving stress, but it could make life easier for those who experience it.

Still, I was curious to see if there was anything I could do using my computational skills. In my research into driving stress, I came across a study by Elgendi et. al., where they were trying to determine which of the different biosignals: electrocardiogram (ECG), electromyogram, hand galvanic skin resistance, foot galvanic skin resistance, and respiration, was most correlated with driving anxiety [3]. Upon performing a statistical analysis that incorporated both supervised learning and unsupervised learning techniques, the latter being the aggregate of three separate models, the study found that "ECG is the most informative biosignal for detecting driving stress, with an overall accuracy of 75.02%." [3]. Other studies also had also found similar positive correlations with heart rate data [4][5]. This information is helpful because it means we can use heart rate as a quantitative measurement for stress, given its significant correlation.

I know from work on a prior summer research project using the Flutter development framework about how Apple Watches function and how they can be used to monitor and display health statistics, along with their integration with Apple Health and all of its features. These Smartwatches in general, along with Apple/Google Health, have also made heart rate data significantly easier to collect and process. Thus, I further began to put the pieces together of a potential project I could build that could help reduce driving stress.

1.3 Project Outline

It all of a sudden clicked in place for me: I wanted to put all of these elements together into building an application that provides a map and driving directions for the user that incorporates heart rate data to find the least stressful path from point A to B. The heart rate data would be pulled directly from the user's smartphone Apple/Google Health apps, which includes time and location. This project would help me explore how the Google Maps Platform APIs work, learn how they could be applied to be impactful for others, and in the end build a final project that also is significant to others as well as myself. Ultimately, heart rate data would be collected from many users of the app using Google Health to access and store the data.

The app would work similarly to the already existing Google Maps online app, but with the following distinctions and inclusions:

1. Google Maps combines speed limit and distance to produce a measure of how long a given segment of road takes to drive. The speed limit component of this measurement would be replaced with a combination that instead takes into account the aggregate heart rate, in beats per minute.
2. The navigation path drawn would display the most optimal 'least stressful' path between points A and B.
3. The ECG data would be collected from Google/Apple Health data, where each ECG reading also includes a location and time. This data could then be stored in a database; the data in the database could then be used to supply the weights for the pathfinding algorithm.
4. Users can add their heart rate data to the database by clicking a 'upload heart rate' button within the app, where then their local Google/Apple Health will be uploaded to the secure database.

In terms of scalability of the app, I knew an app of this complexity would require more time to fully develop. Therefore, there were some components that I knew I would not be able to implement going into the project because it would not be possible without having many users. These include:

1. Ensuring the aggregate mean calculated for the ECG is accounting for potential outliers and biases while still also showing statistically meaningful data.
2. Assessing the efficiency and performance of the app to observe what it is like running a Google Maps size app when all of the pieces of data are in place.
3. Creating a whole database to store the ECG data while also making it scalable to become quite large, so a Cloud Firestore or similar online database server would be preferred over a local SQL database.

Below in Figure 1, I visually display what components make up the app, and how they will interact and connect with each other in several phases. “Phase 1" is the ECG data collection and processing, "Phase 2" is the generation of the path, "Phase 3" is the presentation of the path, and "Phase 4" is the return and recalculation of information. Notice how the Smartwatch is already connected to Google/Apple health, as being integrated with the Apple/Google Health system is already a component of the device.

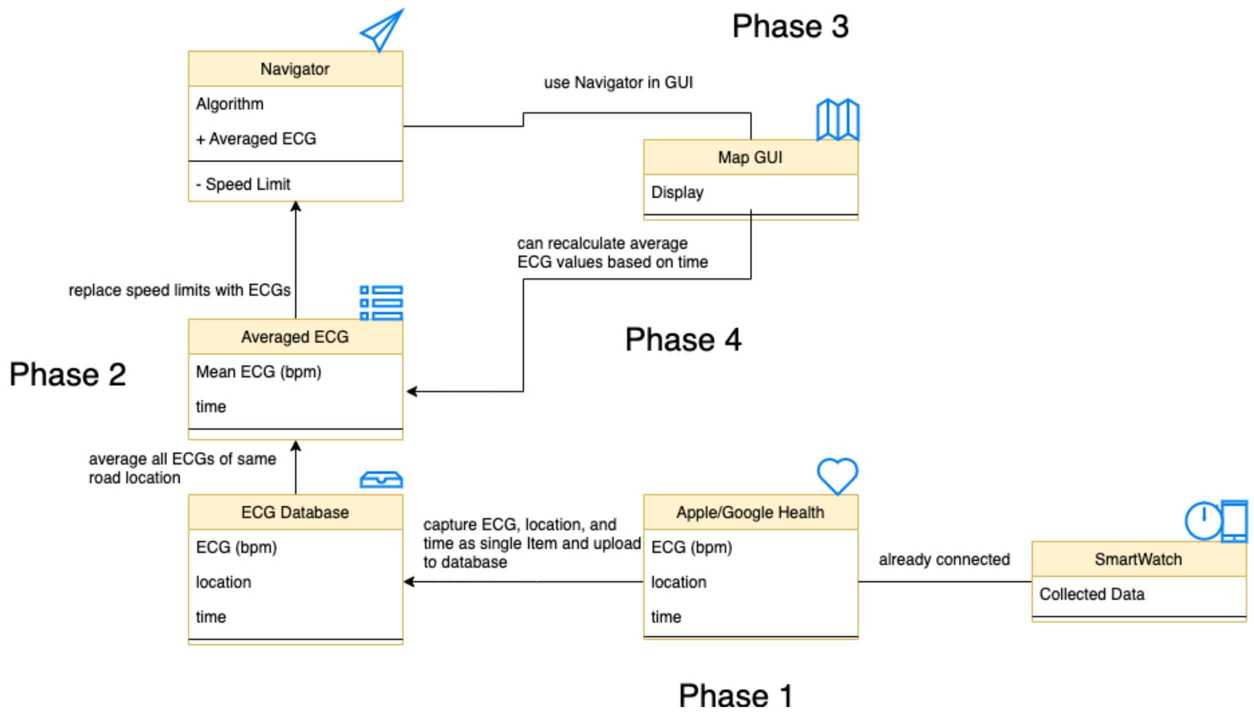


Figure 1: *Framework of Project*

Starting in Phase 1, the data captured from Apple/Google health will be sent to a database. This will contain three parameters: ECG value, location coordinates, and time/date. Each location will then be ‘snapped’ to the nearest road segment. The ECGs of each of these road segments will then be averaged, but this average could be different depending on which time parameters are selected. Say, for example, someone wants to average only the times of day of rush hour: between 5 and 7 pm. Thus, the averaged ECG would be different for this segment of road than if it is averaged in total, or during different times.

Then for Phase 2, these average ECG values of the road segments are then swapped in for the speed limit data within the navigator, and thus the algorithm computes a different result since it is now using the ECG data as weights instead of the speed limits. Following to Phase 3, this navigator is then implemented in the map user interface, where the map can be interacted with by the user of the app. When the user plugs in two points and then finds the path from one to the other, this navigator will then calculate and return the optimal path based on the ECG data, also known as the 'least stressful' path.

For Phase 4, the user can then optionally recalculate the averages when filtered for date and/or time, following back along the arrow back to update the navigator again for Phase 2 and 3.

2 Related Work

2.1 Overview

Before I dove into building my project, I was curious to see how/if others have tackled situations similar to this to know two things: 1. See if my project is feasible and 2. Be prepared for potential hurdles I could potentially run into along the way using the Google Cloud Platform. I will review three of these projects: Transit Accessibility with Direction API, GIS with Maps API, and an implementation of the Sort Filter Skyline with Maps API. All of these, while different in functionality, are all utilizing Google Maps APIs as tools and are also themselves solving their respective problems. After this, I will then summarize my general takeaways of where the field of applied Google Maps Platform implementation currently is.

2.2 Transit Accessibility with Direction API

In their research, Haitao et al. explored two different methods of measuring access to public transit transit in major cities [6]. These methods were: 1. the Google Direction API calculator for finding paths to nearby transit entrances and 2. the existing local data analysis from prior collected data. Haitao et al. compared these methods in a case study using the Beijing city map. Here, they used the Google Direction API to calculate the walking distances to transit points given the URL strings containing the coordinates of the transit origin and destination, filter for travel type of travel (driving, biking, etc.), and API keys. The API keys are authorization codes that give the user access to the Google Direction API. These results were then mapped to a grid of the Beijing city blocks and run through the grid validator function.The study in the end found that the Google Direction API was in fact accurate generally in predicting areas that had more access points to public transit.

For the context of my problem, it was interesting for me to see how the Direction API could be applied to various means of transportation and not just cars. Upon looking further, I even discovered that the API accounts for different units of measurement: kilometers and miles, as well as an integrating feature with the system clock to calculate an estimated time of arrival. Along with these specificities for my project, I was also interested to see the discussion section where the Google Cloud Platform advantages and faults were brought up for solving the accessibility problem.

Overall, it seems that Haitao et. al. were satisfied with the many simplifications of the Google Direction API: the whole geographic calculation process being duplicatable, no data collection, fewer computing resources, and easily interpretable results [6]. It seems that for the context of the research problem, which was to determine how well the application of the Direction API was in determining accessibility, the Google Cloud Platform was a convenient and effective solution.

Still, there were a few setbacks. The API is based on sampling data so the more generalized and larger the gridded cell, the more likely there are to be errors [6]. However, adding too many cells would then increase the amount of computing needed to be done, and would make the algorithm more time consuming. Finding the balance will be different on a case by case basis, so a large-scale implementation of this project would need to be done carefully. Moreso, the socio-economic characteristics that can also impact walking to transit access points were outside of the scope of this project and by not addressing them this project also has limitations.

Lastly, I also thought it was fascinating to see how a real world problem, another very important one: access to public transportation, could have a solution using Google Maps APIs. While there is not a perfect solution, it could still be fully functional and scalable. This gave me more confidence that the problem I was trying to solve with driving anxiety could also similarly have a solution with Google Maps APIs, specifically the Direction API as well.

2.3 Interactive Clinic GIS (Geographic Information System) with Maps API

This article discusses the work done by Rahmi et al. in which they build a mobile app to help facilitate and accommodate doctor-patient interaction [7]. The app consists of a search finder where the patient can find nearby available clinics, see them on a map, and get directions to a selected clinic. The app also has features for the user to pick out a doctor from a list of doctors at the clinic, message them directly, as well as store and send health data.

What I thought was interesting about the article was that it went in depth to give context of patient-doctor interactions, the health field, and how important connectedness is to further move the field further . Similar to the study of Haitao et al., this complex real world problem could have a potential solution using technology [6]. From this, I better understood what and why each component of the app was there, and I understood the scope of the project in the larger context of mobile health development.

And while the app had other features for storing and transferring data using both a local and online database, the component that was of most interest to me was the find location feature which was created using the Google Cloud Platform's Maps API. The app uses both the Map API to display the map and clinic icons, as well as the Directions API to calculate the fastest path to the selected icon [7]. While simple, it was interesting to see that the Google Cloud Platform APIs are usable in a mobile development environment. Mobile health is still a rapidly evolving industry, and given that it has been a few years since this project was completed it is possible that even more can be done.

2.4 Implementation of Sort Filter Skyline Algorithm with Maps API

Lastly, I observed a study by Annisa that similarly to my project was looking at specific ways to expand on existing Google Maps APIs [8]. The main question this project tackled was to see the number of surrounding available facilities/work spaces for a user inputted location. Currently, the Google Maps app does not have a location selection feature based on the number of facilities nearby [8]. To solve this problem Annisa proposes that they can find a way to implement the Sort Filter Skyline (SFS) algorithm, a popular query method used in location selection, using the existing Google Maps APIs.

This approach involved two Google Maps' APIs: 'Find Place', specifically using the Place APIs, and 'Nearby Search'. These APIs use a location as input and return one or more place name values of nearby locations respectively. Along with this, a python implementation of SFS was run and the output, the number of attributes around the location, was stored in a data table. A front-end user interface was then also built so the user of the app can input a location and get in return the number of available spaces nearby.

This study demonstrates that it is possible to incorporate both Google Maps existing APIs into other functions: in this case the SFS algorithm. For sample locations, Annisa used locations around IPB University in Indonesia. The results of the study show that the algorithm, Google Maps APIs, and user interface all do in fact run. The only factor hindering performance was the size of the data. Generally, the more data there is to process, the longer the SPS algorithm will take to run and thus slow down the application.

I thought this study in particular was closest to what I was trying to do: create a user interface as well as incorporate several components of the Google Maps APIs into the backend of my app. Thus, I thought it was interesting to see how not only was it possible to do this quite easily, but also how the other non-Google component, the SFS algorithm, was relatively easily added on after the two other APIs were run. Still, I was beginning to get concerned about if there were ways to actually alter the ways an existing Google Maps API functions.

2.5 Takeaways

Even though my project is different from the three I read above, I learned a lot from reading about the various applications of Google Maps APIs. Each study I read helped give me more clarity in 1) The scope of my project and 2) The problems I would likely need to work around to complete the project. I will discuss each of these below.

The main takeaways from all of these projects were that they performed case studies: Haitao et al. used the Beijing city grid, or locations around IPB University in Indonesia in Annisa's study [6][8]. Thus, it made sense to me to either pick sample locations that I am familiar with, like the Merriam Park area near Mac-Groveland, in my study for using the Google Maps APIs, or if in the case they do not work to construct my own map of Macalester.

Similarly, the scalability of the existing work was not really addressed since they only built demos. The Transit Measurement and SFS studies both addressed the runtime slowing down as a potential problem, but neither proposed ways on how to fix this issue [6][8]. This is a potential problem in terms of app functionality on a mass-scale, the scale I envision for this project. More work would need to be done to figure out how to store, aggregate, and use the heart rate data effectively.

Another issue I also figured would be a problem is that it appears directly altering Google Maps APIs might not be easily done. All three studies above, while incorporating the Google Maps APIs, are not actually using them outside of their set function [6][7][8]. They are essentially being used as either add ons or location matchers. Even though I wanted to modify the weights used by the Direction API, I began to realize that might not be possible with the Google Maps APIs.

To accommodate all this, I went into the project with 2 plans. The first was to see if it was even possible to build my initial vision using only the existing Google Maps APIs: see which API specifically is responsible for the speed limit data storage and also show how the paths differ depending on whether the map was trying to find the fastest or least stressful route. I also had a back-up plan that if for some reason I am unable to alter the speed limit, then I would simply build my own map and navigator from scratch and then show how they vary when using the ECG data or the speed limit for the navigator function.

3 Approach

3.1 Purpose of App

Ultimately, my goal with this project was to propose and solve a real world problem using the existing data and technologies that Google Maps APIs and Google Health provide. The app helps users find the least stressful path from their current location or a starting point to a destination. At the core, the functionality from the user's perspective is:

1. A map that is easily navigable with either mouse-clicks and drags. Icons for zooming in and zooming out that are clickable, and icons on the map that are clickable.

2. Two address selection text boxes, the first being the current location and/or starting point, and the second being the destination.
3. A navigator that calculates the ‘least stressful’ path between the two points.
4. A ‘Sync Google Health’ button that when clicked, will upload the user’s ECG data from their Google Health account to the app.

From the backend, the following will have to be implemented in order to connect both the map display, navigator, and Apple/Google Health data components:

1. Access an accurate map and display it.
2. Replace the existing speed limit data with the ECG data.
   i. Create and store ECG data in a database, where it is also averaged based on road segments.
   ii. Find a way to access speed limit data in the current Directions API.
   iii. Substitute in ECG data using the ‘snap to road’ function.
3. Use the navigation path algorithm to compute the optimal, "least stressful" path between points A and B, and display this path on the map.

While this app as designed could be scaled up to include more features, like being able to filter the average heart rate for specific times of day or potentially being able to show multiple options for least stressful paths, the initial project vision was to simply help the user be able to find the least stressful path. These, as well as other potential features, are outside the scope of this project, which is simply to see if building such an app is even possible.

3.2 Overview

The following sections cover what I was able to successfully implement using Google Maps APIs as well as other Google libraries and products like Google Health and Flutter. From the user perspective outlined in section 3.1, I was able to implement all aspects, except the functionality of the "Sync Google Health" button, as this was related to the back-end issues I had run into when attempting to build a database in SQL and store the data. As for the rest backend, each of the three components mentioned in 3.1 were only partially worked on and had unique problems:

1. With Google Maps I can display the map and produce routes, but I can't modify the optimization parameter. However, I did end up finding an alternative way to use the parameter I wanted to construct a map representation and load it and use it to produce routes. As of writing I still can't display the routes nicely, but if given more time that can change.
2. For the ECG data: I collected ECG data, made a demo, and am currently storing it in a text file for now. However, I would like to create a database to allow for scaling up.
3. For the navigation algorithm: the Google Maps API ultimately wouldn't work. I did end up finding a way around this, and implemented an alternative.

In general for my app development process, my goal was to have it be easily recognizable and accessible to users who are already familiar with Google Maps. I also wanted my app to function exactly the same way that Google Maps does, as the existing pathfinding algorithm it uses has been proven to be efficient in its current form. This would also give me, as the developer, more experience with using Google Maps APIs and help teach me how to use them in action.

The end result was intended to be a clone of the Google Maps app that functions the same way, with the same back-end direction calculating algorithm. This demonstrates that the APIs work in the way they are intended to, and can even be hooked up onto a website's simple HTML back-end with the use of API keys and calls. This also gave me the opportunity to work on the layout of the app, and what it would look like to the user.

3.3 UI Design

Similar to the functionality, for the front end design of the project my goal was to keep it as similar to a Google Maps style interface as possible, since that has proven to be an easy to navigate interface for millions of people already. Thus, I decided to only have one page that displays both the map as well as the user input features. Figure 2 below shows the layout and look of the web app I implemented. While this functions, it cannot work with the ECG data needed for this project. There is also more room on the right hand side to potentially add more features in future advancements of the project.

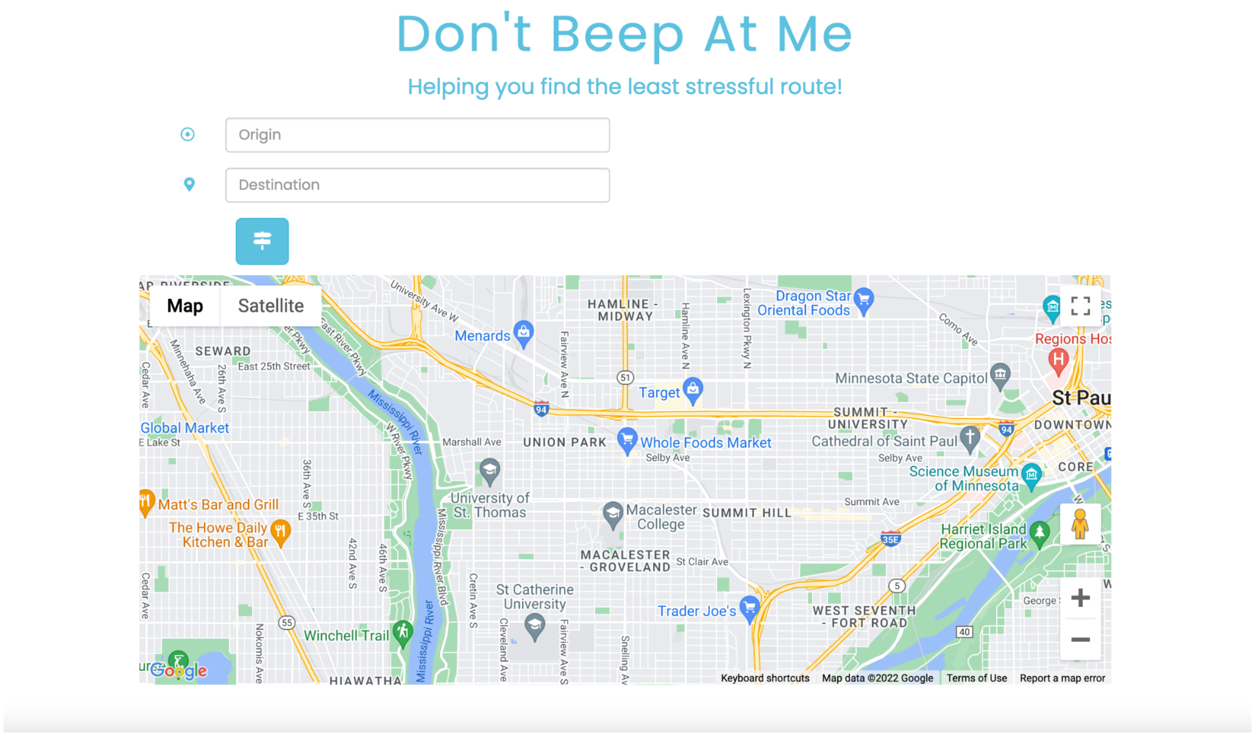
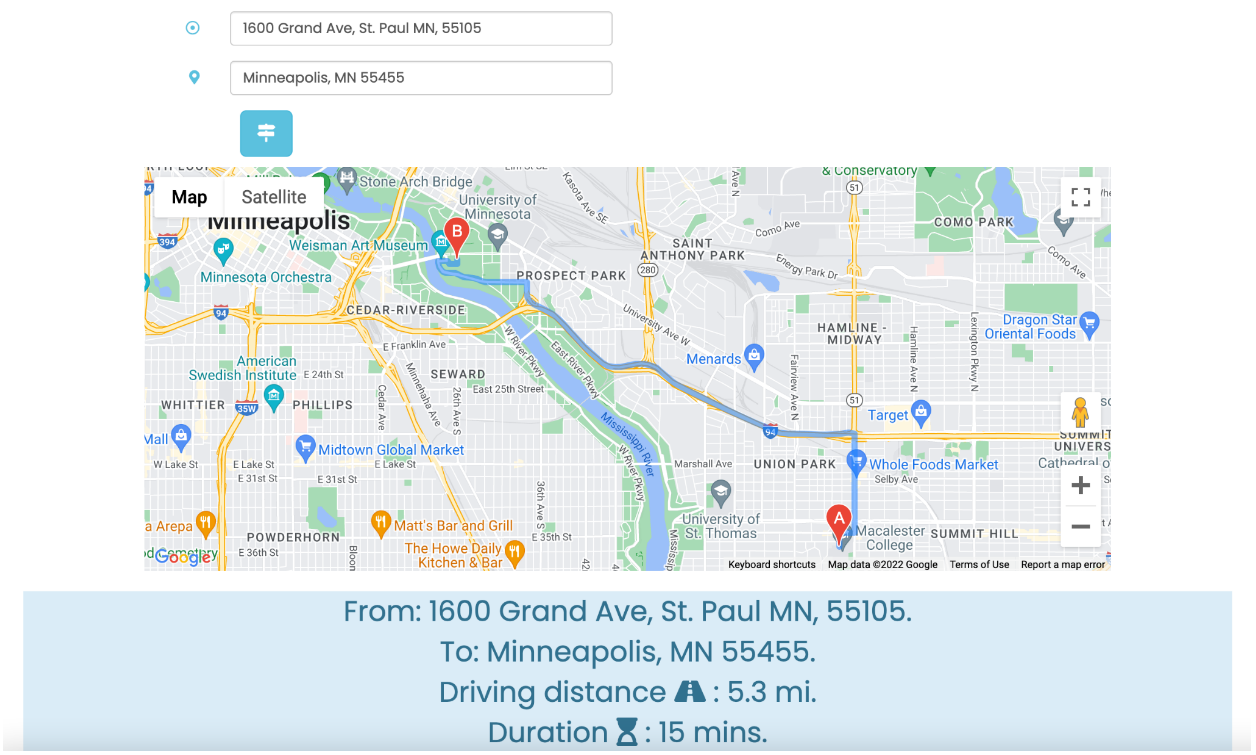


Figure 2: *Front-end design of app*

Of the Google Maps APIs, this specific project above uses the Map API to display the already hard coded map seen above, along with the tabs that are included. The rest of the text and buttons outside of the map square are part of the UI interface coded by myself. The two input text boxes take the address value of a starting and destination point, and the blue crossroad button calls on the function to navigate and find an optimal path between the two points.

When the button is pressed, the map image updates to what can be seen below in Figure 3, demonstrating how the map display changes once the user has plugged in a start and end address and pressed the blue button:


1600 Grand Ave, St. Paul MN, 55105

Minneapolis, MN 55455

From: 1600 Grand Ave, St. Paul MN, 55105.
To: Minneapolis, MN 55455.
Driving distance : 5.3 mi.
Duration : 15 mins.

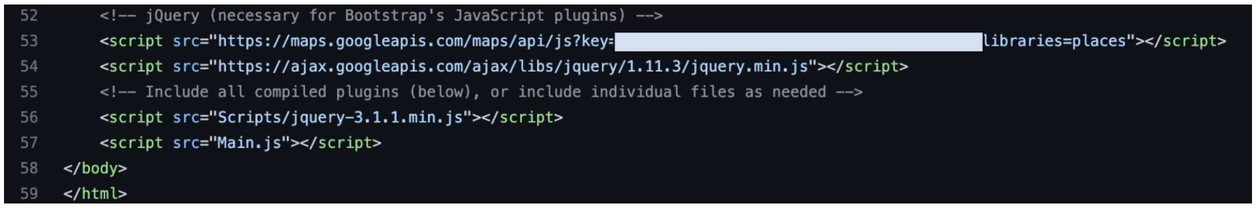
Figure 3: Result screen for fastest route between points A & B

This blue path that appears on the map display is present once the Directions API returns the path, and accounts for every street intersection along the way from points A to B. Similarly, the total distance of this path is returned, along with an estimated time.

This whole process should be quite familiar to those who have used Google Maps, since the start and end nodes appear on the map as icons A and B, and the blue path that connects them just like it would appear on a Google Maps app. The user can keep reentering locations and the map will keep updating to find the new optimal path. Furthermore, if the user enters the same two points again it is not guaranteed that the same exact path will be outputted again, as the Directions API takes into account many features to calculate the optimal path.

3.4 Back-End of Map-Display

The code behind this app contains the API keys for both the Maps API and the Directions API, as seen below in Figure 4. The API keys serve as an access token to call on an existing library/query, not accessible by the coder, that has the code to build/draw/construct whatever the user may need [9]. Since this is set up through a Google Cloud account, my API keys are unique and specific to my app and will grant access to both the Google Maps Map API and the Directions API. Using these API keys, I am then able to use the Google Maps APIs libraries of map images, algorithms, and more.

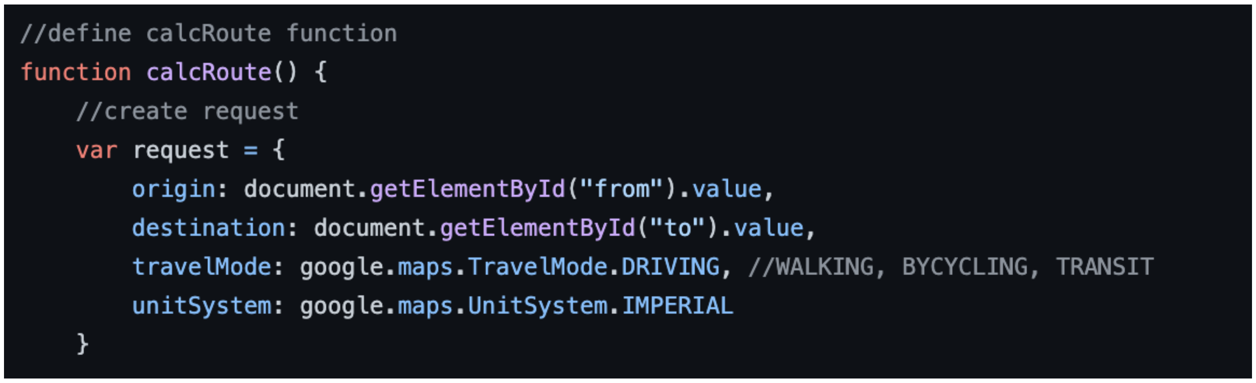


```
    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script src="https://maps.googleapis.com/maps/api/js?key=[redacted]libraries=places"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
    <!-- Include all compiled plugins (below), or include individual files as needed -->
    <script src="Scripts/jquery-3.1.1.min.js"></script>
    <script src="Main.js"></script>
</body>
</html>
```

Figure 4: *First 2 lines show the code for the API keys, with actual key value blurred out for security reasons*

Even though these keys are only two lines, these are essential to have to use and deploy the API and their contents within our app. Without them, the app would not work as the code would not have access and be calling on the code that builds the map in the Maps API, nor find the path with the Directions API.

To see an example of one of these calls, observe the code segment to calculate the route in Figure 5. Notice how here it is creating a request to calculate the route and simply plugging in our desired values.



```
//define calcRoute function
function calcRoute() {
    //create request
    var request = {
        origin: document.getElementById("from").value,
        destination: document.getElementById("to").value,
        travelMode: google.maps.TravelMode.DRIVING, //WALKING, BYCYCLING, TRANSIT
        unitSystem: google.maps.UnitSystem.IMPERIAL
    }
```

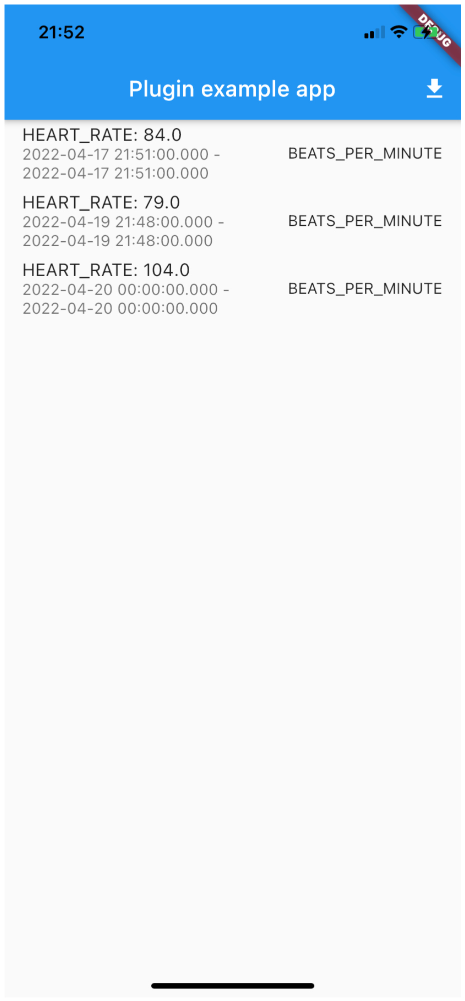
Figure 5: *The Google Maps API method to calculate a route*

However the limitations of the Directions API can be seen. Here, the user can only customize the type of travel mode and unit system. While this gives the user some freedom it is also incredibly limiting, especially for a project like mine that requires the extraction and insertion of specific values. The API keys simply act as a means for a user to authenticate access to the Google Maps Platform [9]. Thus, the actual code to build and run the navigation algorithm is not accessible and as such, it is impossible to directly alter the code.

3.5 Integration with other components

Another aspect of the project was to collect ECG data from Google Health, along with its geo-location and time/date. Using code from a prior project I had worked on, I was able to successfully find a way to pull heart rate data from Apple Health along with the date and time of its recording.

I built a simple app using Google's Flutter framework that extracts various health data from the Google Health app. Specifically, I ran the code on my iPhone XR to make sure that it can read in heart rate data from the Apple Health app, and as seen below it takes the heart rate data, date, and time as attributes for a reading in Figure 6.



HEART_RATE: 84.0
2022-04-17 21:51:00.000 -
2022-04-17 21:51:00.000
BEATS_PER_MINUTE

HEART_RATE: 79.0
2022-04-19 21:48:00.000 -
2022-04-19 21:48:00.000
BEATS_PER_MINUTE

HEART_RATE: 104.0
2022-04-20 00:00:00.000 -
2022-04-20 00:00:00.000
BEATS_PER_MINUTE

Figure 6: *Sample flutter app that extracts heart rate data from Apple Health, along with date and time*

This app could read the data and store and display it in the mobile development framework. In the future, this data would be uploaded to a database, and I could use the Google Location API to connect the heart rate's location to the nearest road segment.

3.6 The A* algorithm as part of the Directions API

One specific component of the backend, the Directions API, actually implements a pathfinding algorithm to find the most optimal path between points A and B. Even though this code is not directly accessible, much speculation has shown that the

Directions API appears to use a heavily modified version of the A* algorithm, which in turn is a modified version of Dijkstra's algorithm [10]. While we can't do anything directly with this information within the Directions API, a fundamental understanding of how the path traversing functions is important for understanding what exactly within the algorithm we want to change, and what other parts could potentially be affected.

3.6.0 Connected Graph

To think about it more abstractly, we can look at our map as a connected weighted graph that consists of vertices, intersections of roads, and edges, the road segments themselves. Figure 7 below shows a simple example, with the m values being the number of edges and n values amount of nodes.

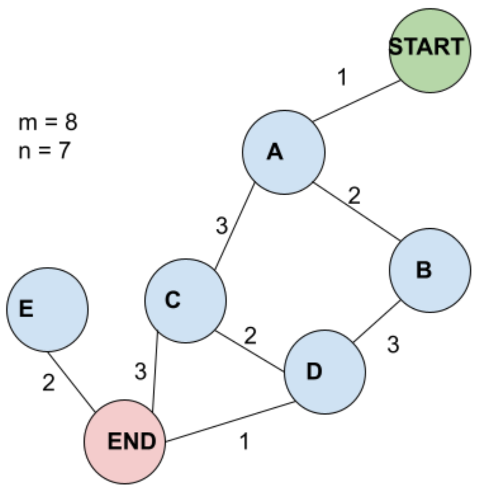


Figure 7: Connected weighted graph

While pathfinding can appear complicated, it is essentially a weighted graph search/traversal problem: finding the lowest-cost path from the start to the end. The algorithms will run on a graph with thousands of edges and nodes in the same way they do on a graph of this size, ultimately the size does not matter. As already stated, the

Google Maps Directions API pathfinding algorithm is a heavily modified version of the A* algorithm, which in turn is an adjusted Dijkstra's algorithm: an algorithm that uses weighted values for finding the most optimal path between A and B [10]. By assigning these weights to the edges and using various nodes as start and end points, I can use these algorithms to traverse a graph and find the optimal path between the start and end point.

3.6.1 Dijkstra's Algorithm & General Pseudocode

At the base level, Dijkstra's Algorithm is a greedy pathfinding algorithm that uses edge weights to calculate a path cost [10]. At each intersection, a path cost is computed and the lowest cost path is then selected. It finds the optimal path from the start to every other vertex, and as such will be guaranteed to find the final optimal path.

The general pseudocode for Dijkstra's algorithm is given below:

Dijkstra's Algorithm Pseudocode

```
• Place the start node on the Open list. This list keeps track of possible nodes to explore.
• While Open is not empty:
    – Pop the node off Open that has the lowest weight score 'w' and put it in the Closed list of explored nodes.
    – For each of its reachable neighbors: compute the weight score 'w' and
        * If the node is not in Open, make the current node its parent, record its scores, and put it in Open.
```

* If it is in Open, check to see if this path to it is better by comparing its current 'w' score and the computed 'w' score. If it is, change its parent to be the current node and update its score.

While this algorithm can function on any scale, the ultimate problem with Dijkstra's is that it will not stop once it finds the destination and will continue looking until it finds every possible set of paths between all the points [10]. Also since this is a greedy algorithm the weights, while helpful, can also potentially lead the algorithm down paths that won't necessarily be optimal in the long run. These are just some of the reasons why Dijkstra's algorithm can be computationally exhaustive and slow, especially if the graph has thousands of nodes and edges. Thus, some adjustments can be made to make the algorithm more efficient.

3.6.2 The A* Algorithm

This leads us to the A* algorithm. The reason the A* algorithm is one of the most popular choices for pathfinding in computer science due to its efficiency, optimality, and completeness [11]. It is also fairly flexible and can be used in a wide range of contexts. A* is an informed version of Dijkstra's as it knows in advance the target destination, and also knows to stop looking for more paths once the target destination is found. It also factors in this knowledge with a heuristic function, the approximate cost from the current node to the goal. Each node x is assigned an f score, where $f(x) = g(x) + h(x)$, with $g(x)$ being the distance from the current node to the starting node, and $h(x)$ the heuristic distance from the current node to the destination. A* only expands on a node if its $f(x)$

score is the current smallest, therefore reducing the amount of nodes that are considered in regular Dijkstra’s.

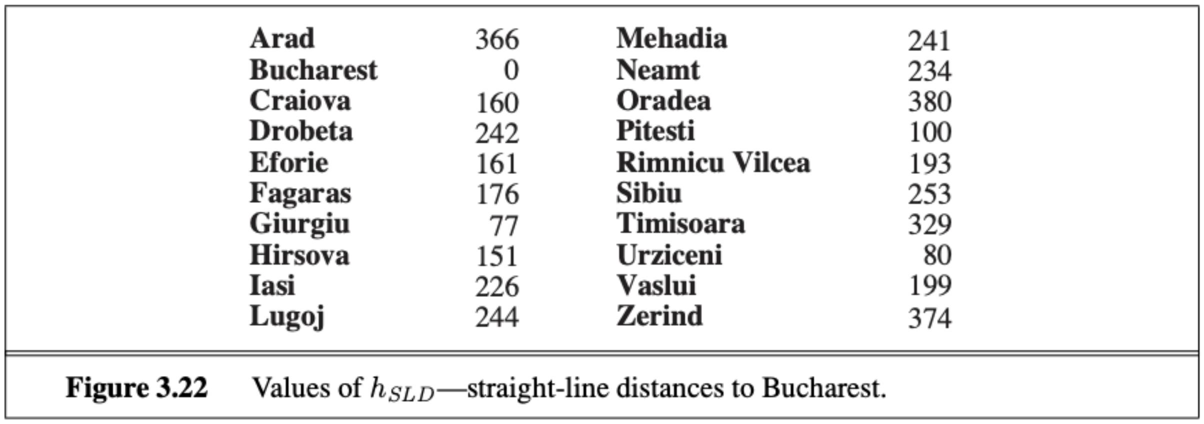
3.6.3 A* Pseudo-Pseudocode

The most basic implementation of A* given the start and end nodes is as follows:

A* Algorithm Pseudocode

```
• Place the start node on the Open list. This list keeps track of possible nodes to explore.
• While Open is not empty:
    – Pop the node off Open that has the lowest f score and put it in the Closed list of explored nodes.
    – If the node is the destination, terminate the program and return the path.
    – For each of its reachable neighbors: compute its f, g, and h based on the current node.
        * If the node is not in Open, make the current node its parent, record its scores, and put it in Open.
        * If it is in Open, check to see if this path to it is better by comparing its current f score and the computed f score. If it is, change its parent to be the current node and update its scores.
```

An example of this can be seen in Figures 8 & 9 below with Romanian cities mapped to a connected graph [12]. The cities are nodes and the distances between the cities are edges, with the edges having weighted values. Ultimately, this implementation of the algorithm still functions by using the node objects as weights.



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.

Figure 8: *Table of Romanian city weights [12]*

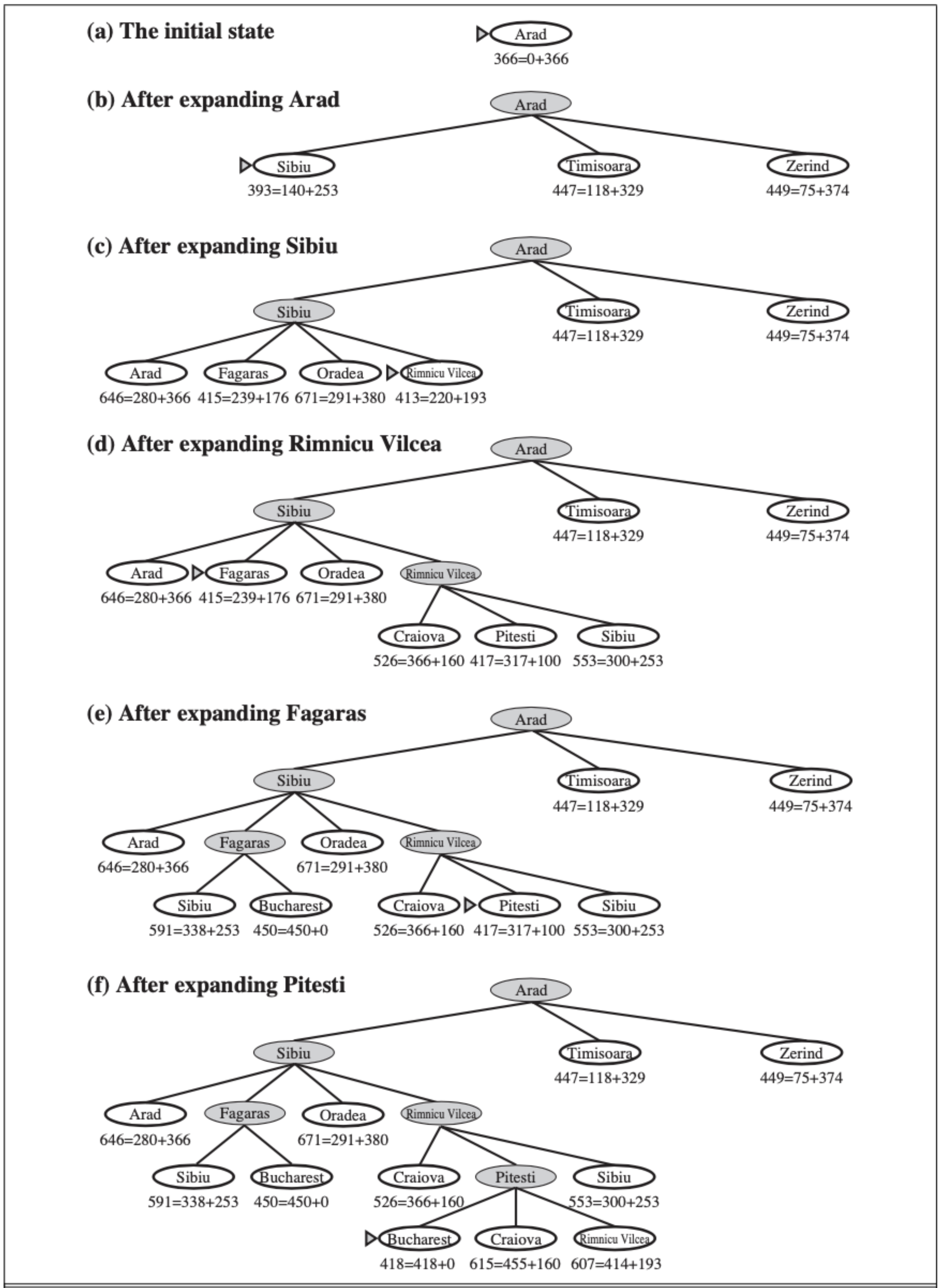


Figure 3.24 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure 3.22.

Figure 9: *Implementation of A* on Romanian cities [12], referencing weights from Figure 8*

Figure 9 shows how the A* optimizes by choosing a path and sticking to it, a more depth-search approach since it also takes into account the final distance as the heuristic function as part of its weight function. The heuristic is the straight line distance between two cities using coordinates, ignoring the landscape or roads. Therefore it will underestimate the actual distance.

3.6.4 Application to Project

In the context of my problem: the edges represent the roads and the vertices represent a street/path intersection. By converting a map of mine to a connected graph, I can use various path finding algorithms to find optimal paths between vertices A and B. This way, the map also accommodates for the selection of the calculated weighted values by using the speed limits.

I also can apply our weights, which in this case is the f(x) heuristic function, at the N number nodes to a maximum number of N-1 edges connecting the nodes, where each edge's new value is the average of the two nodes it connects. More specifically, the speed limit values map to the g(x) component $f(x) = g(x) + h(x)$ where the g(x) component consists of distance (in miles) /max speed limit (in miles per hour). The h(x) is the total calculated distance from the current node to the destination node, and thus these two values added together work together to be the f(x) weight value.

Ultimately, my goal was to see if there is a way to adjust the algorithm's g(x) value to run between the two points using g(x) = distance (in miles) * heart rate (beats per minute) instead of the prior g(x) = distance (in miles) /max speed limit (in miles per hour). However, given the current resources within the Directions API this task was not possible as of the time of writing this thesis. This leads into the next discussion where I also discuss other limitations of Google Maps APIs in general as well as how I implemented the A* pathfinding algorithm separately from the Directions API.

.

3.7 Implementation & Problems

This section describes the parts of the app that didn't work and/or elements of the app that I was not able to complete at the time of writing, like the SQL database. A lot of this was simply due to a lack of resources on utilizing Google Maps APIs in various contexts, as well as Google not having made the source code public. However, other factors were due to lack of knowledge in using other programming languages or frameworks.

The project outline was to build a MVP (minimum viable product) of a potential Google Maps Extension Feature that uses Apple/Google Health ECG data of both the user's heart rate as well as date, time, and location to help "snap" the heart rate to the nearest road using the Roads API "snap to road" function. The front-end of this app was to be presented to the user in a similar way that the Google Maps app is: asking the user to input a start and end address, a find route button, along with a 'Sync to Google Health' button. Upon the press of the find route, the least stressful path would then be calculated and presented to the user. This project would then also be scalable, as with more heart rate data inputted by users the more accurate the map would become to the population.

While the almost finished MVP I currently may seem concise, I still managed to incorporate all the elements I imagine the final extension to have to the core. The only thing that would be missing is the user entered data, and as that is perhaps the most important component of my app. Nevertheless, the app in its current state is still also designed to be accommodating of further extensions: whether it be additional tools, libraries, or even other Google apps like Google Health in order to keep with the mission to make the Don't Beep At Me extension more engaging and accessible. Other scalable parts of development, like calculating aggregate mean, would only be possible to calculate if I have a significant number of users for the app.

Among the obstacles faced, I recalled that my vision for the MVP simply asks the user to enter heart rate, time, date, as well as the nearest point onto a map. It ultimately did not matter what size the map was, as long as there were multiple alternative routes. If we simply build our own map instead of using the Maps API to create the map interface, I also wouldn't have the hurdle of filling in the heuristic values with either speed limits and/or heart rates.

Even though these tasks don't sound inherently complex, replacing the speed limit data with heart rate ended up being both monetarily non feasible to extract all the speed limits. It also was simply impossible to plug the heart rate data into the Directions API without knowledge of what the hardcode behind it looks like. In the end, I had to scale down to a smaller scale project where I could manually input the data. With this, I was able to demonstrate how I pictured the larger scale project. I decided this was a choice I would have to make either way eventually, since this MVP would end up having to be scalable in the long run.

Lastly, I was not able to find a way to use this on the Google Maps clone I had created, where I can assign the speed limit values as weights. I can then replace and calculate the new A* implementation for the heart rates which I can then use to find the least stressful path. I was still curious to see if the speed limit parameter was able to be altered. Although it was costly, there is a way to extract speed limit data from the Direction API [13]. However, there is no functional way as of now to replace it with another value, especially on the scale required for this project. Thus, I had to explore other alternatives to bringing my MVP to fruition.

4 Rework with Smaller Scale Minimum Viable Product

4.1 Replacing Maps API and Directions API

I began looking into ways that I could still deliver a MVP that encapsulates my initial visual while also being functional. I therefore decided to hard code a map of a few neighborhood blocks around the Macalester-Groveland in St. Paul, MN, U.S. Using this smaller model, I was able to replace speed limit data with heart rate data easily, and also calculate the paths for the heart rate.

To make the map, I used a text file to layout each street intersection and road in the style of a connected graph. While there is no front end display of this, the back-end code for the map is created and ultimately will still function the same way. Figure 10 below visually demonstrates how this connected graph looks like on top of the map, with the red boxes being nodes and the white paths between them the edges. Note, the edges are multidirectional. This means that if there are one way streets, they are mapped here as a multidirectional path. This is purely for simplicity purposes of the demo, but ideally in actuality the algorithm will also account for one- directional paths.

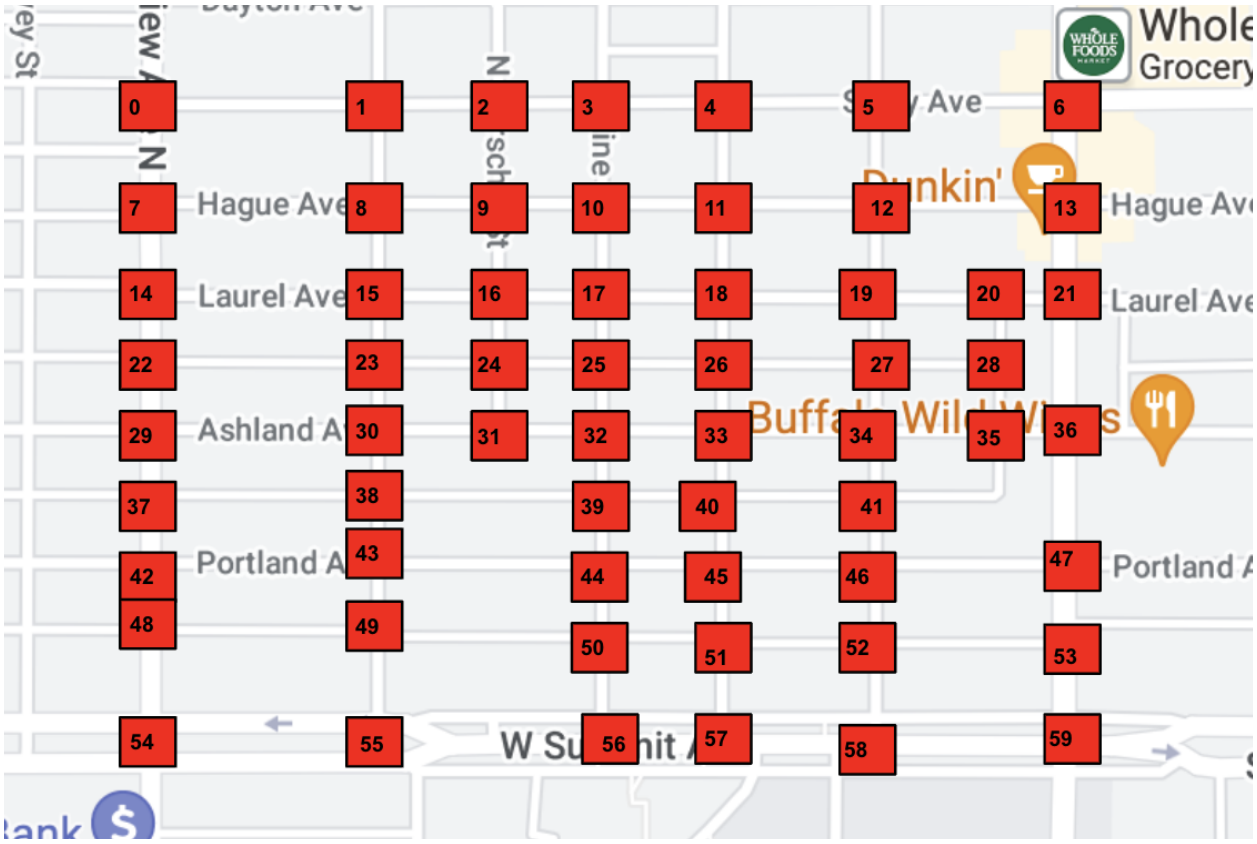


Figure 10: *Map of Macalester-Groveland with Nodes*

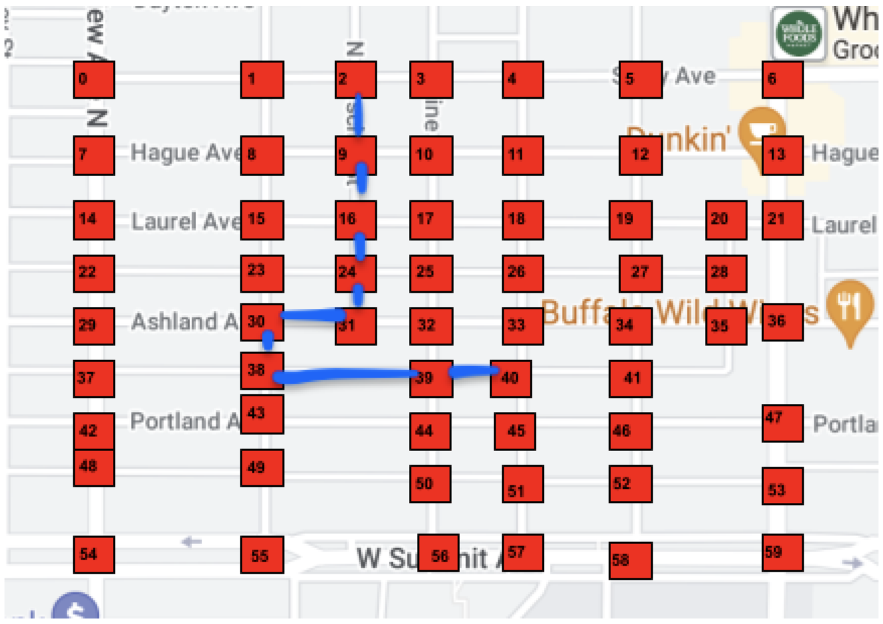
Collecting the speed limit and heart rate data for this project was a lot less tedious since it was on a much smaller scale. I was able to gather my own data rather straightforwardly. The Saint Paul Government Public Works Department data has speed limit data which I used to get the speed limits of the streets [14]. I then drove down every street in the map above and calculated my heart rates. If a street segment was missing a heart rate, I filled it in with 75bpm as that seemed like a good number for the average resting heart rate. While I recognize my heart rate is by no means an accurate representation of the aggregate heart rate of thousands of drivers, for the purposes of this demo and showing that the app works, this data was sufficient.

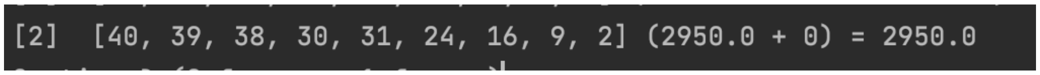
As for coding the algorithm, I used an already existing implemented A* algorithm from Macalester's COMP 484: Intro to AI course. I altered the g(x) from simply being distance for it to instead account for distance and the respective speed or heart rate weight values:

- g(x) = distance (in miles) → g(x) = distance (in miles) / speed limit (miles per hour)
- g(x) = distance (in miles) → g(x) = distance (in miles) * heart rate (beats per minute)

4.2 Comparing Fastest Path to Least Stressful Path

Finally, I then ran the A* on both versions of the map. Almost immediately, I was able to run across an example, specifically involving a narrow one-way street that was quite stressful to drive on with ice. Figure 11 below shows the fastest path between two points below.



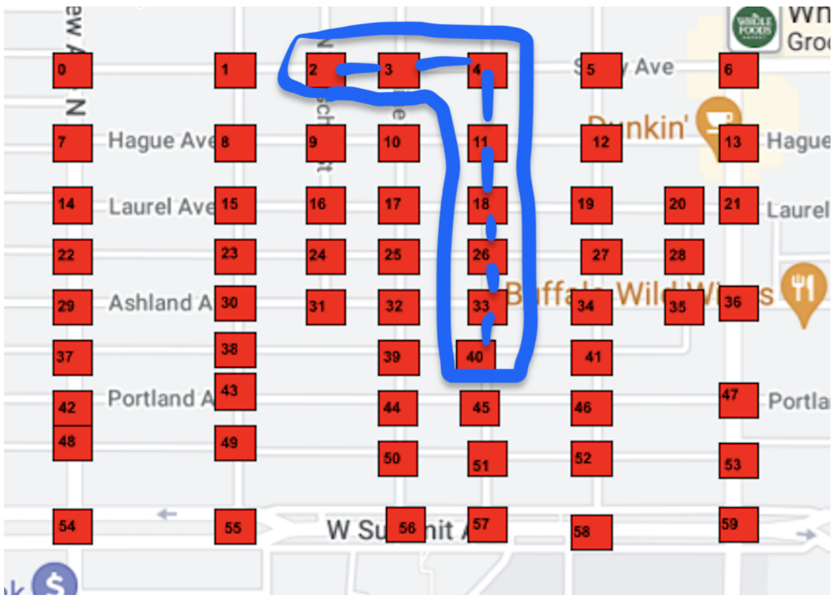


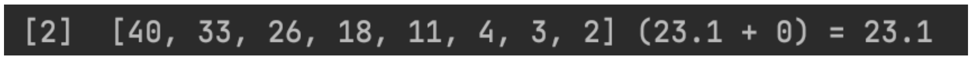
```
[2]  [40, 39, 38, 30, 31, 24, 16, 9, 2] (2950.0 + 0) = 2950.0
```

Figure 11: *Map of Macalester-Groveland for fastest path between nodes 40 and 2.*

Since the fastest path uses the speed limit as the weight, it will follow a path with the fastest route if the driver is traveling near the speed limit. However, as seen in Figure 11, this path involves going along the narrow icy one way street from nodes 31 to 2.

In contrast, Figure 12 below shows the ‘least stressful path’, using my own heart rate values as weights.





```
[2]  [40, 33, 26, 18, 11, 4, 3, 2] (23.1 + 0) = 23.1
```

Figure 12: *Map of Macalester-Groveland for least stressful path between nodes 40 and 2.*

Here it can be observed how the path avoids both of the narrow one-way streets 2-31 and 3-56 and takes the path that may not necessarily be the fastest, but is the least stressful as it only takes wider two way roads. It was also interesting to note how the path ‘least stressful path’ has less turns in general in comparison to the fastest path. This could also be because stressful intersections can affect the heart rate of drivers on street segments adjacent to those intersections, and thus are avoided as well.

4.3 Scalability & Limitations

As discussed previously, the Macalester Map gives front-end and back-end insight into how the Don't Beep At Me extension would function on a larger scale in the Google Maps clone, given it is applied to Google Maps. In order to keep the scalability, I would need a way to connect to Google Maps API so I can extract the speed limit data, even if this in itself is a costly and grandiose task.

Another scalability feature would be for us to account for running the Location API every time to match heart rate with a location. We saw in related studies how running the API over and over can cause runtime issues and potentially also overload the phone memory/battery if the API is running for long periods of time. Thus going forward, we would need to find a way to maximize efficiency in running the Location API repeatedly. Some possible solutions are that could set up a client-server interaction system between the device and a computer, or alternatively we could estimate the location of heart rate by taking the location at specific intervals and the approximate mapping the heart rates in between to an approximate location (this could be done if someone is on a long stretch of road or highway, for example).

It would also be necessary to insert the heart rate data from the Google Health data using other means outside a local SQL database or a temporary local server to instead use something like an online database, Cloud Firebase for example. This would make it so the heart rate data doesn't take up all the user storage in the user's device and instead contains the data in an online server. However, this would require the user's device to have internet access at the time of data collection, which in itself could make the app less accessible and essentially defeat the purpose.

Lastly, the aggregate mean calculation of the heart rate can become not an accurate metric of measurement if it becomes too generalized. Even if filtering out outliers, the average from thousands of people can minimize the magnitude of how stressful roads actually are. A mean based off of a smaller more biased sample will be read the same way as a mean that is more accurate of the population. Similarly, the A* algorithm can also become over generalized if run across a larger graph. The algorithm would need to also be able to filter out specific data points so it knows to avoid a small segment of road that is significantly stressful over a less stressful larger segment.

5 Discussion & Conclusion

5.1 Future Work

Outside of the implementations with scalability, some direct things I ideally want to integrate into the Don't Beep At Me app would be the Google/Apple Health component so the user, instead of manually inputting all the heart rate data, can simply link their heart rate to the map. Ideally in a Google Maps API setting we would do this using the Roads API 'snap to road' function, but unfortunately in the context of this problem I ended up not doing this with the APIs, so another method would need to be implemented to do this.

I already have a way to extract the heart rate data along with the date and time of the reading. However, the reading does not come with a location and as such I would probably still need to use the Google Maps API for ‘Find Location’ and call on it everytime I take a heart rate reading so I can also get the coordinates of the heart rate reading.

To continue with the heart rate thread, a database of some kind would be a useful addition to store all the heart rate data, dates, and locations so that the map grid can then be updated more effectively. However, as I myself have limited SQL coding experience at the time of writing this, I would still need to learn more about database structure and function before I could comfortably move forward with this task.

Lastly, I also had planned to create a graphic user interface for the Don’t Beep At Me program along with a help window that would have described the project’s greater functionality. This would look similar to what the first partial MVP project I had in section 3.1. However, given the time constraints and existing technical challenges of the project, this initiative proved infeasible at the time.

5.2 Reflection

In its current form, the project has a functioning backend to calculate two paths using the A* algorithm: one that is the “fastest” using the speed limit parameter mimicking the existing Google Maps API, which also has a GUI that displays the paths in full, as well as output.

and one that is the most optimal when accounting for driving stress that demonstrates how Don’t Beep At Me would function on a smaller scale. Some of the biggest challenges arose when I was trying to get the speed limit parameter from the Google Maps API. Despite the increased price to access the speed limit data, there is no way to actually replace the speed limit values on a mass scale as this project would require. The only options were ultimately to build a Google Maps clone using the existing APIs and not include the feature I was most interested in incorporating: heart rate, or alternatively to build my own map and pathfinding algorithm from the ground up and incorporate the heart rate data into that.

Ultimately, I hope the work I have done so far with Don't Beep At Me demonstrates how implementing it in Google Maps would be a useful, engaging and accessible tool for users to engage with familiar technology in a new and effective way, while also helping people with driving stress. Simultaneously, I also hope to show the limitations and potential further expansion that Google Maps can incorporate into the future so that programmers like me can use it for more applications.

With the help of my advisors and committee members, I am very proud of the work I have accomplished to reach our goals and produce a tangible and fully-functional final product. I hope that the lessons I have learned, both with exploring the Google Maps APIs and exploring its various applications will inform my future work, in this subfield and in the grander paradigm of AI, algorithms, and computer science.

6 References

[1] B. -G. Lee and W. -Y. Chung, "Wearable Glove-Type Driver Stress Detection Using a Motion Sensor," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 7, pp. 1835-1844, July 2017, doi: 10.1109/TITS.2016.2617881.

[2] Aaron S. Baker, Scott D. Litwack, Joshua D. Clapp, J. Gayle Beck, Denise M. Sloan, The Driving Behavior Survey as a Measure of Behavioral Stress Responses to MVA-Related PTSD, Behavior Therapy, Volume 45, Issue 3, 2014, Pages 444-453, ISSN 0005-7894, doi: 10.1016/j.beth.2014.02.005.

[3] M. Elgendi and C. Menon, "Machine Learning Ranks ECG as an Optimal Wearable Biosignal for Assessing Driving Stress," in *IEEE Access*, vol. 8, pp. 34362-34374, 2020, doi: 10.1109/ACCESS.2020.2974933.

[4] J. E. Taylor, “The extent and characteristics of driving anxiety,” in *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 58, pp. 70-79, 2018, ISSN 1369-8478, doi: 10.1016/j.trf.2018.05.031.

[5] *66% of Americans experience driving anxiety + 8 tips to ...* The Zebra. (2021, October 15). Retrieved April 2022, from https://www.thezebra.com/resources/driving/driving-anxiety/.

[6] J. Haitao, J. Fengjun, H. Qing, Z. He and Y. Xue, "Measuring Public Transit Accessibility Based On Google Direction API," in *The Open Transportation Journal,* vol. 13, pp. 93-108, July 2019, doi: 10.2174/1874447801913010093.

[7] A. Rahmi, I. N. Piarsa and P. W. Buana, "FinDoctor-Interactive Android Clinic Geographical Information System Using Firebase and Google Maps API," in *International Journal of New Technology and Research (IJNTR),* vol. 3, no. 7, pp. 08-12, July 2017.

[8] S. K. Annisa, "Location Selection Based on Surrounding Facilities in Google Maps using Sort Filter Skyline Algorithm," in *Jurnal Ilmu Komputer dan Informatika*, vol. 7, no. 2, October 2021, doi: 10.23917/khif.v7i2.12939.

[9] M. Pegg, (2019, August 22). *How to secure API keys for google maps platform | google cloud blog*. Google. Retrieved April 2022, from https://cloud.google.com/blog/products/maps-platform/google-maps-platform-best-practices-restricting-api-keys.

[10] R. Mitchell, (2021, October 21). *The algorithms behind the working of google maps*. CodeChef. Retrieved April 2022, from https://blog.codechef.com/2021/08/30/the-algorithms-behind-the-working-of-google-maps-dijkstras-and-a-star-algorithm/.

[11] N. Swift, (2020, May 29). *Easy A* (star) pathfinding*. Medium. Retrieved April 2022, from https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2/.

[12] S. J. Russell and P. Norvig. “Chapter 3. Solving Problems by Searching.” Artificial Intelligence: A Modern Approach, Pearson, Harlow, 2009.

[13] Google. (2022). *Speed limits*. Google. Retrieved April 2022, from https://developers.google.com/maps/documentation/roads/speed-limits#parameter_usage.

[14] *Speed limits*. Saint Paul Minnesota. (2021, November). Retrieved April 2022, from https://www.stpaul.gov/departments/public-works/traffic-lighting/speed-limits.