SFWRENG 2XB3 Final Project

Design Specification

Department of Computing and Software
McMaster University

Group 8
Project Name: NavSafe
Version Number: 1.0

Arkin Modi - 400142497
Benson Hall - 400129627
Joy Xiao - 400125285
Leon So - 400127468
Timothy Choy - 400135272

Last updated: April 12, 2019

1 Revision

1.1 Team Members and Responsibilities

Team Member	Student Number	Responsibilities
Arkin Modi	400142497	Head of Back-End
Benson Hall	400129627	Team Leader, Scrum Leader, Back-End Programmer
Leon So	400127468	Back-End Programmer
Timothy Choy	400135272	Head of Front-End and Programmer
Joy Xiao	400125285	Log Admin and Back-End Programmer

1.2 Revision History

Date	Changes
March 7, 2019	Added skeleton
April 12, 2019	Added content to all sections

1.3 Attestation and Consent

By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through [CS] or [SE]-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.

2 Contribution

Name	Role(s)	Contributions	Comments
Arkin Modi	Head of Back-End	Requirements Specification, Design Specification	
Benson Hall	Team Leader, Scrum Leader, Back-End Programmer	Graph construction and associated "shortest-path" algorithm, project log.	
Leon So	Back-End Programmer	All read modules, Collision ADT, Intersection ADT, Street ADT, both searching algorithm (SearchIntersections.java, SearchCollisions.java), Top-down MergeSort for sorting intersections alphabetically, Top-down MergeSort for sorting collisions by severity codes, fatalities, and injuries.	
Timothy Choy	Head of Front-End and Programmer	Project Log, Everything to do with front end (designing the interface, Google Maps API integration, connecting front end to back end), Decomposition Tree	
Joy Xiao	Log Admin and Back-End Programmer	Project Log, Top-Down MergeSort for sorting the collisions according to their severity, fatalities, number of severe injuries, then number of injuries, created the unit testing for all the read modules, searching modules, and sorting modules.	

3 Executive Summary

The purpose of NavSafe is to provide a path from a start point to an end point in the City of Seattle. This path however will be generated with past collisions data to improve the safety of the path. Traveling in a city can be dangerous, and accidents tend to happen particular stops in the city. Vehicle collisions can lead to injuries or even death. Mistakes by pedestrian or driver can lead to a higher chance of a fatal collision in these intersections. With this trend we can use past data to plot a path avoiding this areas. This is done by assigning a safety weighting to each street and intersection. Using this, we can calculate the shortest safest path.

Contents

4 Design Overview

4.1 Decomposition Explanation

NavSafe is broken down into four major sections, Data Types, Read, Graph and Search/Sort. The Data Types are used to organize the data into a form that is more usable to the developer. This section contains the Collision, Intersections and Street modules.

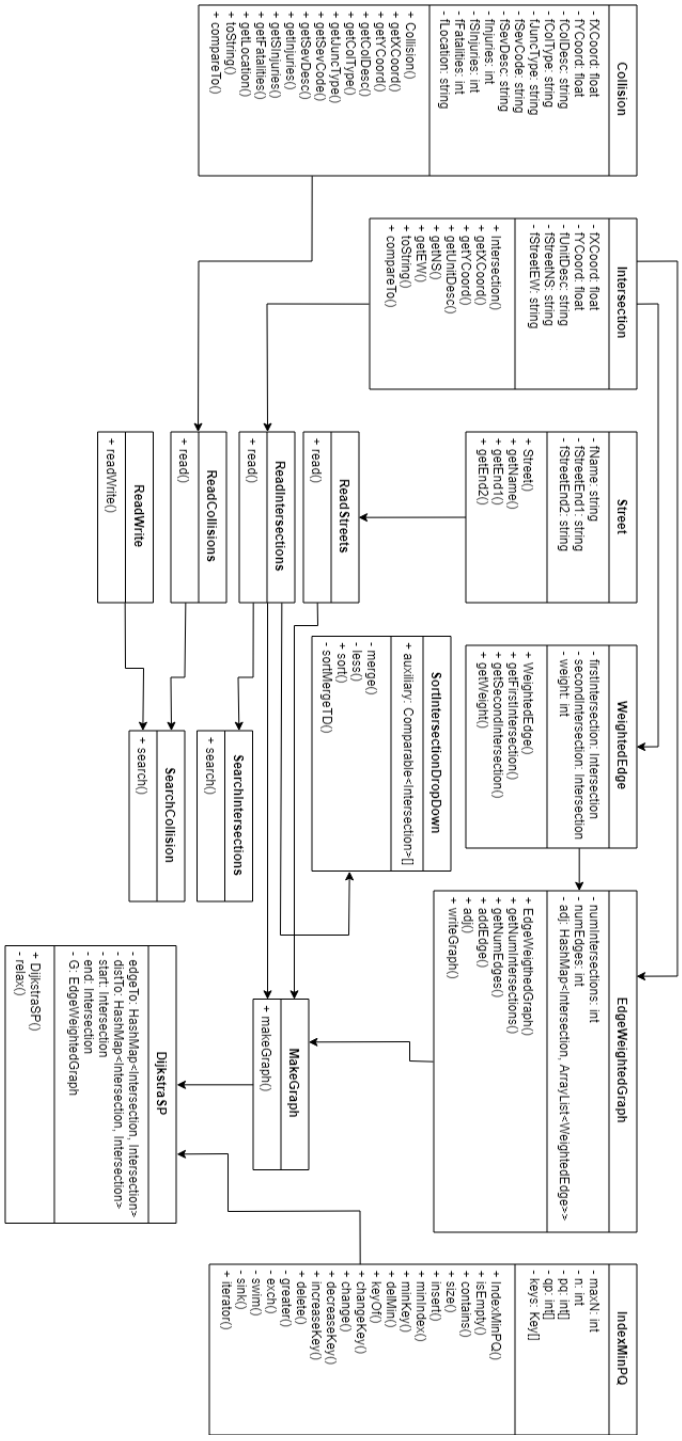
The Read section is broken down into four modules. The ReadWrite module is more of a pre-processor with removes usable data in our data sets. The other three modules, ReadCollisions, ReadIntersections and ReadStreets are used to read and parse the data from the data sets.

The Graph section is broken down into five modules, EdgeWeightedGraph, IndexMinPQ, MakeGraph, WeightedEdge and DijkstraSP. These are used to find the shortest safest path from point A to point B.

The Search/Sort section is broken down into three modules, SearchCollision, SearchIntersections and SortIntersectionDropDown. These are primarly used for their own independent features in the front-end.

The reasoning to why the program was decomposed into the four major sections is emphasis the idea of separation of concerns. We also wanted to promote high cohesion and low coupling. With each module is closely related to the other modules in its respective section, we can achieve high cohesion. Also, by not having the modules be dependent on many other modules, which achieves low coupling. These are the reasons to why NavSafe was decomposed the was it was.

4.2 UML Class Diagram



5 Module Interface Specification

5.1 ReadWrite Module

Module

ReadWrite

Uses

None

Trace Back to Requirements

This is a part of the Read Module. This helps with Accuracy of Results by removing inaccurate data.

Syntax

Exported Access Programs

Routine Name	In	Out	Exceptions
readWrite			

Semantics

Environment Variables

collisionData: File containing the data set of collisions

State Variables

None

Assumptions

The collisionData file will not change format.

Access Routine Semantics

readWrite():

- transition: All the rows missing any data are removed from collisionData

5.2 Read Intersections Module

Module

ReadIntersections

Uses

Intersection

Trace Back to Requirements

This is a part of the Read Module. This helps with Accuracy of Results by taking only the data we need.

Syntax

Exported Access Programs

Routine Name	In	Out	Exceptions
read		seq of Intersections	

Semantics

Environment Variables

intersectionData: File containing the data set of intersections

State Variables

None

Assumptions

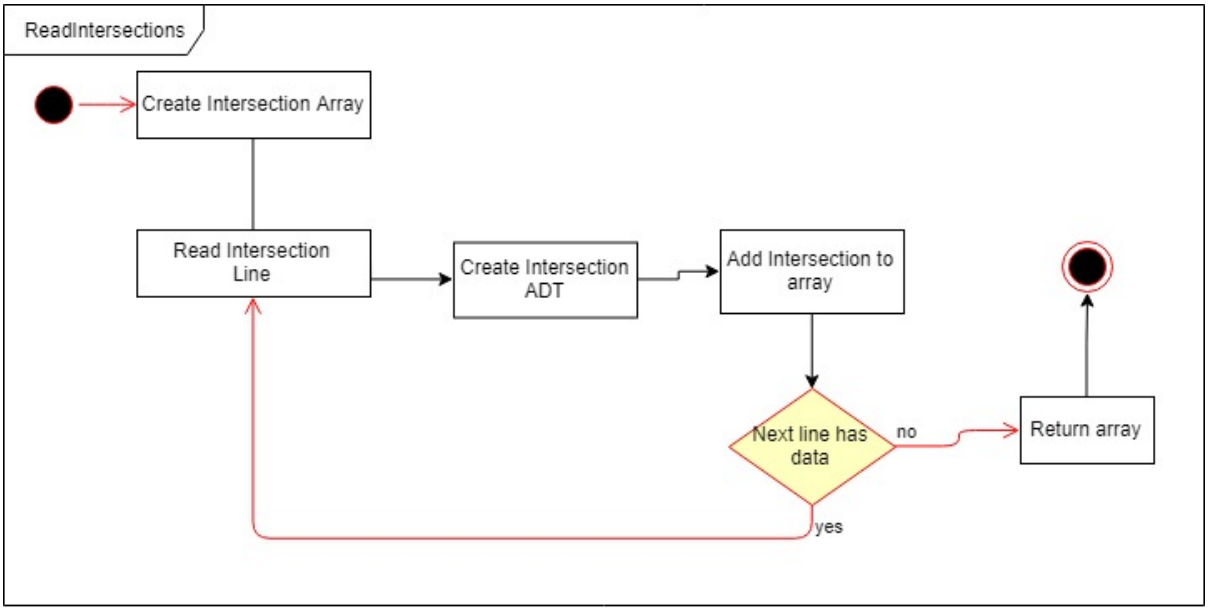
The intersectionData file will not change format.

Access Routine Semantics

read():

- output: A sequence of intersections where each intersection is parsed from each line in the IntersectionData file

UML State Machine



5.3 Read Streets Module

Module

ReadStreets

Uses

Streets

Trace Back to Requirements

This is a part of the Read Module. This helps with Accuracy of Results by taking only the data we need.

Syntax

Exported Access Programs

Routine Name	In	Out	Exceptions
read		seq of Streets	

Semantics

Environment Variables

streetData: File containing the data set of streets

State Variables

None

Assumptions

The streetData file will not change format.

Access Routine Semantics

read():

- output: A sequence of streets where each street is parsed from each line in the streetData file

5.4 Read Collisions Module

Module

Read

Uses

Collision

Trace Back to Requirements

This is a part of the Read Module. This helps with Accuracy of Results by taking only the data we need.

Syntax

Exported Access Programs

Routine Name	In	Out	Exceptions
read		seq of Collisions	

Semantics

Environment Variables

collisionData: File containing the data set of collisions

State Variables

None

Assumptions

The collisionData file will not change format.

Access Routine Semantics

read():

- output: A sequence of collisions where each collision is parsed from each line in the collisionData file

5.5 Intersection Module

Module

Intersection

Uses

None

Trace Back to Requirements

This helps with Reliability as it makes it easier for the developer to get the required data. This will also improve Maintability.

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine Name	In	Out	Exceptions
new Intersection	seq of string	Intersection	
getXCoord		float	
getYCoord		float	
getUnitDesc		string	
getNS		string	
getEW		string	
toString		string	
compareTo	Intersection	int	

Semantics

State Variables

fXCoord: X-coordinate of intersection location
fYCoord: Y-coordinate of intersection location
fUnitDesc: Intersection description
fStreetNS: Street in north-south direction
fStreetEW: Street in east-west direction

Access Routine Semantics

Intersection(s):

- transition: fXCoord, fYCoord, fUnitDesc, fStreetNS, fStreetEW := float(s[0]), float(s[1]), s[2], s[3], s[4]

getXCoord():

- output: out := fXCoord

getYCoord():

- output: out := fYCoord

getUnitDesc():

- output: out := fUnitDesc

getNS():

- output: out := fStreetNS

getEW():

- output: out := fStreetEW

toString():

- output: out := "StreetNS: " + fStreetNS +" StreetEW: " + fStreetEW

compareTo(s):

- output: out := compares intersection to s based on Unit Desc's letters (alphabetical order) and return -1 is less, 1 if greater, or 0 if equal

5.6 Collision Module

Module

Collision

Uses

None

Trace Back to Requirements

This helps with Reliability as it makes it easier for the developer to get the required data. This will also improve Maintability.

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine Name	In	Out	Exceptions
new Collision	seq of string	Collision	
getXCoord		float	
getYCoord		float	
getColDesc		string	
getColType		string	
getJuncType		string	
getSevCode		string	
getSevDesc		string	
getInjuries		int	
getSInjuries		int	
getFatalities		int	
getLocation		string	
toString		string	
compareTo	Collision	int	

Semantics

State Variables

fXCoord: X-coordinate of collision location
fYCoord: Y-coordinate of a collision location
fColDesc: Collision description
fColType: Collision type
fJuncType: Junction type
fSevCode: Severity code
fSevDesc: Severity description
fInjuries: number of injuries
fSInjuries: number of serious injuries
fFatalities: number of fatalities
fLocation: Location of collision

Access Routine Semantics

Collision(s):

- transition: fXCoord, fYCoord, fColDesc, fColType, fJuncType, fSevCode, fSevDesc, fInjuries, fSInjuries, fFatalities, fLocation := float(s[0]), float(s[1]), s[2], s[3], s[4], s[5], s[6], int(s[7]), int(s[8]), int(s[9]), s[10]

getXCoord():

- output: out := fXCoord

getYCoord():

- output: out := fYCoord

getColDesc():

- output: out := fColDesc

getColType():

- output: out := fColType

getJuncType():

- output: out := fJuncType

getSevCode():

- output: out := fSevCode

getSevDesc():

- output: out := fSevDesc

getInjuries():

- output: out := fInjuries

getSInjuries():

- output: out := fSInjuries

getFatalities():

- output: out := fFatalities

getLocation():

- output: out := fLocation

toString():

- output: out := "X-coord: " + fXCoord + " Y-coord: " + fYCoord + " Collision Description: " + fColDesc + " Collision Type: " + fColType + " Junction Type: " + fJuncType + " Light Condition: " + " Severe Injuries: " + fSInjuries + " Severity Code: " + fSevCode + " Fatalities: " + fFatalities + " Injuries: " + fInjuries

compareTo(s):

- output: out := Compares collision to s based on number of severity code, then fatalities, then serious injuries, then number of injuries and return -1 is less, 1 if greater, or 0 if equal

5.7 Street Module

Module

Street

Uses

None

Trace Back to Requirements

This helps with Reliability as it makes it easier for the developer to get the required data. This will also improve Maintability.

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine Name	In	Out	Exceptions
new Street	seq of string	Street	
getName		string	
getEnd1		string	
getEnd2		string	

Semantics

State Variables

fName: Street Name
fStreetEnd1: One end of the street
fStreetEnd2: One end of the street

Access Routine Semantics

Street(s):

- transition: fName, fStreetEnd1, fStreetEnd2 := s[0], s[1], s[2]

getName():

- output: out := fName

getEnd1():

- output: out := fStreetEnd1

getEnd2():

- output: out := fStreetEnd2

5.8 Search Intersection Module

Module

SearchIntersections

Uses

Intersection
ReadIntersections

Trace Back to Requirements

This is part of the Search Module. This helps with Accuracy of Results as this module will guranntee the result is accurate.

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine Name	In	Out	Exceptions
search	string, string, string	seq of Integer	

Semantics

State Variables

None

Access Routine Semantics

search(x, y, z):

- output: returns the index of two Intersections where the first index contains x and y, and the second index contains x and z

5.9 Search Collision Module

Module

SearchCollision

Uses

ReadWrite
Collision
Read

Trace Back to Requirements

This is part of the Search Module. This helps with Accuracy of Results as this module will guarantee the result is accurate.

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine Name	In	Out	Exceptions
search	Intersection, Intersection	Integer	

Semantics

State Variables

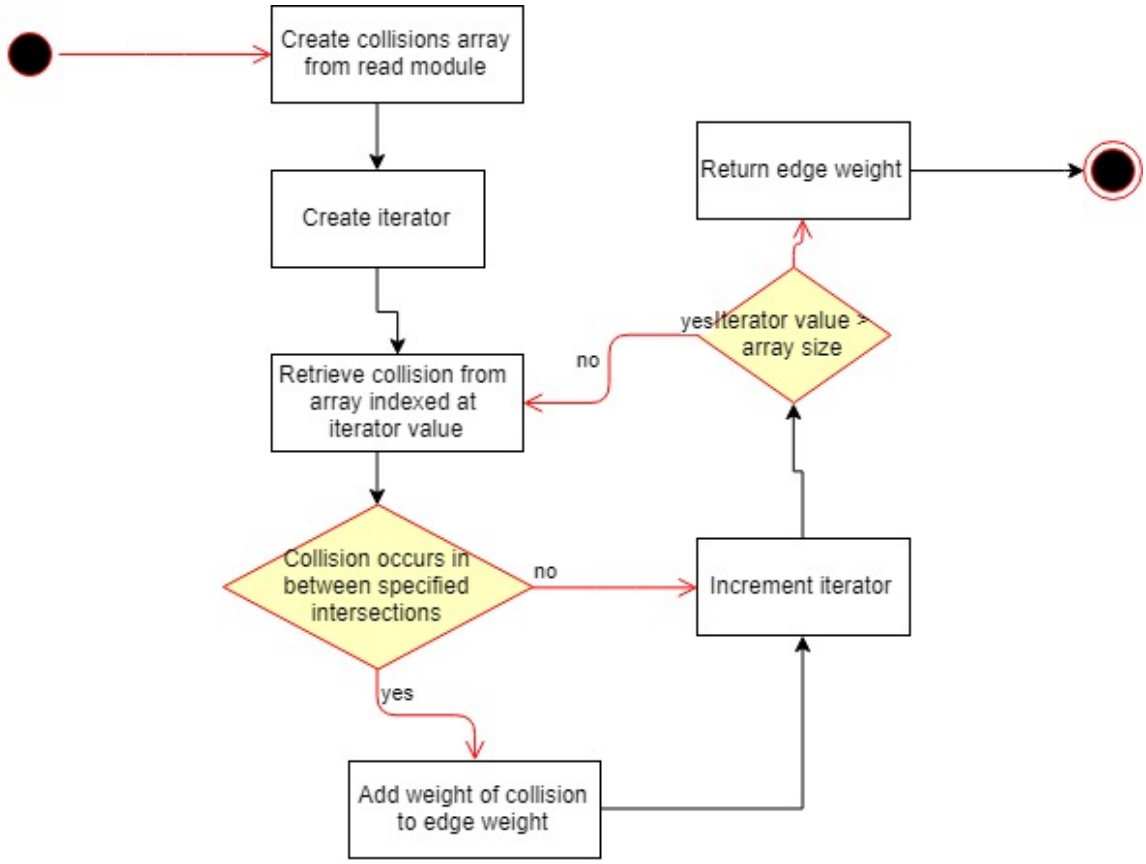
None

Access Routine Semantics

search(x, y):

- output: searches for all collisions contained in street/edge between 2 intersections, x and y, and returns summed weight of all collisions calculated based on severity code

UML State Machine



5.10 Sort Intersection Module

Module

SortIntersectionDropDown

Uses

ReadInsertions

Trace Back to Requirements

This is part of the Sort Module. This helps with Performance as this module utilizes an efficient algorithm. The faster run time also helps with Human-computer Interface Issues.

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine Name	In	Out	Exceptions
sort	Comparable, Integer		

Semantics

State Variables

None

Access Routine Semantics

sort(x, y):

- transition: sorts the comparable in order of the predefined compareTo() method

5.11 Weighted Edge Module

Module

WeightedEdge

Uses

Insertion

Trace Back to Requirements

This is part of the Graph Module. This helps with Performance by creating a data structure that will reduce the search time for calculating the shortest path.

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine Name	In	Out	Exceptions
new WeightedEdge	Intersection, Intersection, Integer	WeightedEdge	
getFirstIntersection		Intersection	
getSecondIntersection		Intersection	
getWeight		Integer	

Semantics

State Variables

firstIntersection: The first intersection
secondIntersection: The second intersection
weight: The weight of the edge

Access Routine Semantics

WeightedEdge(x, y, z):

- transition: firstIntersection, secondIntersection, weight := x. y. z

getFirstIntersection():

- output: out := firstIntersection

getSecondIntersection():

- output: out := secondIntersection

getWeight():

- output: out := weight

5.12 Edge Weighted Graph Module

Module

EdgeWeightedGraph

Uses

Insertion
WeightedEdge

Trace Back to Requirements

This is part of the Graph Module. This helps with Performance by creating a data structure that will reduce the search time for calculating the shortest path.

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine Name	In	Out	Exceptions
new EdgeWeightedGraph	Integer	EdgeWeightedGraph	IllegalArgumentException
getNumIntersections		Integer	
getNumEdges		Integer	
addEdge			
adj	Intersection	seq of WeightedEdge	
writeGraph	string		

Semantics

Environment Variables

output: A text file

State Variables

numIntersections: Number of intersections
numEdges: Number of edges
adj: Hash Map of Intersection and seq of WeightedEdge

Access Routine Semantics

EdgeWeightedGraph(n):

- transition: numIntersections, numEdges := n, 0
- expection: n ¡ 0 ⇒ IllegalArgumentException

getNumIntersections():

- output: out := numIntersections

getNumEdges():

- output: out := numEdges

addEdge(x, y, z):

- output: out := craete and add the weigthed edge between x and y to adj

adj(x):

- output: out := adj[x]

writeGraph():

- transistion: write the graph to output in a csv format

5.13 Make Graph Module

Module

MakeGraph

Uses

EdgeWeightedGraph
ReadIntersections
ReadStreets

Trace Back to Requirements

This is part of the Graph Module. This helps with Performance by creating a graph and saving it so that the other algorithms do not need to spend time building the graph.

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine Name	In	Out	Exceptions
makeGraph			

Semantics

Environment Variables

output: A text file

State Variables

None

Access Routine Semantics

makeGraph():

- transition: create a EdgeWeightedGraph from ReadIntersections and ReadStreets and write the graph to output

5.14 Dijkstra Shortest Path Module

Module

DijkstraSP

Uses

MakeGraph
IndexMinPQ

Trace Back to Requirements

This is part of the Graph Module. This helps with Performance by utilizing an efficient algorithm. This reduces run time and improves Human-computer Interface Issues as well.

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine Name	In	Out	Exceptions
new DijkstraSP	string, Intersection, Intersection	DijkstraSP	

Semantics

State Variables

edgeTo: Hash Map of edges
distTo: Hash Map of distances with the key being an Intersection and the value being the total weight of the path
G: EdgeWeightedGraph
start: start Intersection
end: end Intersection

Access Routine Semantics

DijkstraSP(s, x, y):

- transition: create the EdgeWeightedGraph G from the file s and relax(G, start)

Local Functions

relax(G, start):

- transition: if new shortest path found then update the adj for G and relax(G, node connected to start)

5.15 Minimum-Oriented Indexed Priority Queue Module

Refer to: `https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/IndexMinPQ.java.html`

Trace Back to Requirements

This is part of the Graph Module. This helps with Performance by utilizing an efficient algorithm. This reduces run time and improves Human-computer Interface Issues as well.

6 Internal Design Review

NavSafe was made with the principles learned in SFWRENG 2AA4 and the algorithm design ideas from SFWRENG 2C03. The decomposition of this program was done with the software engineering principles in mind. Having high cohesion and low coupling is good in terms of the design. By programming in a object-oriented style enabled us to utilize the separation of concerns principle. When it comes to the algorithm choices, we used the best algorithm available. The sorting is done with merge sort. The shortest path is done with Dijkstra's algorithm. When it came to search, because of our specific implementation, linear search was the only option available. This introduced a big increase to the running time of the program. The searching aspect to NavSafe is where we could improve the most. Unfortunately, due to the construction of back-end modules being developed in a non-Android Studio environment, the integration between front- and back-end development became impossible to complete.