# NavSafe
## A Safer Way To Get Around

Arkin Modi
Benson Hall
Joy Xiao
Leon So
Timothy Choy

Department of Software and Computing, McMaster University

April 4, 2019

# NavSafe Objective

NavSafe seeks to meet the following objective:

- ▶ Determine the safest route for a person to travel on based on data of collisions in the Seattle area.

# Scope

Features required to accomplish this project:

- ▶ Searching algorithm
- ▶ Custom edge-weighted graph (bidirectional)

    Vertices = Intersection

    Edges = Path between pair of intersections

    Weight of an edge/vertex based on cumulative severity indices

- ▶ Shortest path algorithm (i.e., Dijkstra) to find safest route

# Motivation

- Vehicle collisions
    - Potential risk of **injury** or **death**
- Many high-risk intersections with flawed designs
    - Factor out of the traveler's control
    - Mistakes by pedestrian or driver has a higher chance of being fatal in these intersections
- Areas where flawed design could occur:
    - Road width
    - Speed limit
    - Markings and signs
    - Intersection infrastructure such as dividers and shoulders

# Dataset(s) Used

- ▶ "Collisions" from Seattle GIS Open Data
  - ▶ Number of Collisions
  - ▶ Weather, road, and daylight conditions
  - ▶ Type of collision (pedestrian/vehicle)
  - ▶ Collision details (left/right turn, etc.)
  - ▶ Severity of collision
- ▶ Intersections dataset from the City of Seattle's data site
- ▶ Streets dataset from the City of Seattle's data site

# Requirements Specification

- ▶ Functional Requirements
  - ▶ Read Module
  - ▶ Collision ADT Module
  - ▶ Intersection ADT Module
  - ▶ Graphing Module
  - ▶ Searching Module

- ▶ Non-Functional Requirements
  - ▶ Reliability
  - ▶ Accuracy of Results
  - ▶ Performance
  - ▶ Human-computer Interface Issues
  - ▶ Constraints

# Design Specification

- ▶ Read Module: Read CSV files and create ADTs to be used
- ▶ Graphing Module - Stage 1: Create custom edge-weighted graph
- ▶ Search Module: Search for collisions that occurred on an edge and factor them into the weighing of the edge
- ▶ Graphing Module - Stage 2: Using edge-weighted graph, find shortest path from starting intersection to destination intersection.

# Verification and Validation

Quality Control Procedures

- ▶ Unit Testing
  - ▶ Verify that individual units of code work as intended
- ▶ System Testing
  - ▶ Verify the program and specifications are aligned

# Screenshot of implementation