*Florida International University*

*School of Computing and Information Sciences*

Software Engineering Focus

Final Deliverable

Traffic Simulator 1.0

**Team Members:** Max Leone, Humberto Chacon, Salvador Gandara

**Product Owner(s)**: Kianoosh G. Boroojeni

**Mentor(s)**: Mohsen Taheri

**Instructor**: Masoud Sadjadi

The MIT License (MIT)

Copyright (c) *2016 Florida International University*

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

***Abstract***

*Our project was to design a navigation API for driverless cars to help route traffic in the most beneficial way for society.*

*In the near future, we will see autonomous e-vehicles which self-navigate themselves based on the traffic flow of streets. Under these circumstances, an open network platform is how the vehicles should be navigated such that they can avoid/mitigate traffic congestion in densely populated areas. This problem can’t be solved using classic network routing problems as the traffic pattern of vehicles is oblivious (unknown) or highly stochastic. Our product provides a smarter navigation experience with real-time smart grid routing analytics for driverless cars to disperse traffic in the most beneficial way for society.*

**Table of Contents**

**Introduction** …………………………………………………………………………………………………………… 5

**User Stories**

Implemented User Stories ………………………………………………………………………………………………………… 6

Pending User Stories ………………………………………………………………………………………………………… 7

**Project Plan**

Hardware and Software Resources ………………………………………………………………………………………………………… 8

**System Design**

Architectural Patterns ………………………………………………………………………………………………………… 9

System and Subsystem Decomposition ………………………………………………………………………………………………………… 9

Design Patterns ………………………………………………………………………………………………………… 10

**System Validation**

……………………………………………………………………………………………………………10

**Appendix**

Appendix A - UML Diagrams ………………………………………………………………………………………………………… 11

Appendix B - User Interface Design ………………………………………………………………………………………………………… 15

Appendix C - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents ………………………………………………………………………………………………………… 16

# Introduction

This project involved implementing oblivious network routing algorithms, an API, and a simulation. The purpose of the simulation is to demonstrate the performance of various routing algorithms when they are used to route vehicles to their destinations. Performance is in terms of traffic congestion created when many vehicles are routed along the same paths at once. One algorithm may decrease overall traffic congestion where another may try to find the best path for each individual vehicle. We can use these simulations to visualize how algorithms will perform at scale in order to determine what is the most effect for routing real world traffic.

The algorithms implemented are demonstrated, with proofs, in the book “Oblivious Network Routing” by S. S. Iyengar and Kianoosh G. Boroojeni. In this work, the top-down integral routing scheme was implemented using Python for rapid development. As compared with a shortest-path algorithm such as Dijkstra, it can be clearly demonstrated in the simulator that the top-down routing scheme makes more efficient use of the network when routing cars from one point to another.

The API was designed for simple usage by client machines, which can be virtual simulators or any machine requesting a path in the map. With Flask as the web framework, the client uses two endpoints in order to set the graph and then retrieve paths from point A to point B. In addition, the usage of the API allows us to use authentication and authorization methods to determine who is making calls to our system, whether or not they have access in general, and whether or not the particular user has reached a limit on the number of calls that can be made. The API serves as the entry point for users to be able to call the routing algorithms we have implemented and get a path to travel.

## 

## 

# User Stories

## 

## Implemented User Stories

* Simulator
  + Prototype Simulator in Unity3D
  + Research API’s for creating map based web applications
  + Custom graph editor tool for Unity3D
  + Traffic simulation in Unity3D
  + Convert geojson data to an adjacency matrix
  + Traffic simulation in Javascript web app
* Algorithms
  + Integrated NetworkX for classical algorithms
  + Implemented Algorithm 3.1
  + Implemented Algorithm 3.2
  + Implemented Algorithm 3.4
  + Generated in and out nodes for each node in graph
  + Remove cycles from generated integral routing scheme paths
* API (Application Program Interface)
  + Research API Solution for Traffic Simulator Project
  + Create API endpoint that accepts a JSON representation of Graph
  + Update API to support Unity engine and be able to call get\_path for different algorithms
  + Migrate API from Hug to Flask
  + Ensure Flask API works with both Unity simulator and Javascript webapp.
  + Implement Authorization for Flask API
  + Make Dev Endpoints to Simplify Testing

## Pending User Stories

* Simulator
  + Dynamically retrieve geojson data to feed into API (service)
  + Refactor javascript simulator
  + Add a stats panel to javascript simulator
* Algorithms
  + Implement Algorithms using bottom-up data structures (starting in section 3.2)
  + Improve algorithm speeds
* API
  + Create endpoint to process raw geoJSON similar to Javascript webapp
  + Implement encryption of keystore
  + Implement more secure authorization than http headers
  + Create standalone server for production deployment

# Project Plan

## Hardware and Software Resources

The following is a list of all hardware and software resources that were used in this project:

**HARDWARE:**

* Windows, MacOS, or Linux based computer
* Sufficient Processing Power to run Unity engine:
  + Recommended: Dual-core, 1.5 GHz or better processor

**SOFTWARE:**

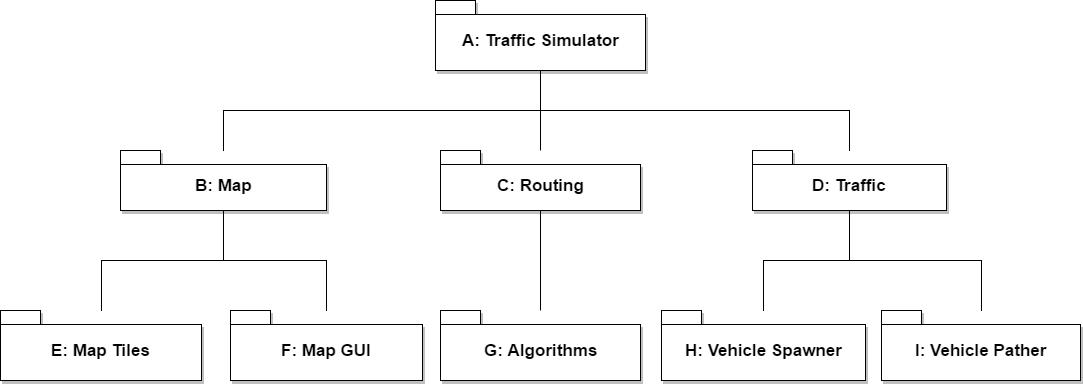
* Python 3.5 (Flask API does not yet support Python 3.6)
* Pip installer for Python
* Python Libraries
  + Numpy
  + NetworkX
  + Cython
  + Flask
  + Flask-Limiter
  + Flask-CORS
* Unity3D 5.5.0 (or greater)
* Javascript
* Javascript Libraries
  + Leaflet 1.0.3
  + Mapquest Leaflet Plugin 2.2
  + JQuery 3.1.1

# System Design

## Architectural Patterns

The system uses the client/server approach. This architectural decision allows the server/algorithms code to live in an optimized computing environment, as well as allow the use of a SaaS model from a business perspective. This greatly facilitates the implementation of the actual clients, which can range from virtual simulators to physical autonomous vehicles, by requiring at most some form of internet connection with an HTTP/S client implementation with the additional possibility of authentication.

## System and Subsystem Decomposition



## Design Patterns

* Simulations
  + Singletons
  + OOP
  + Object Pooling
  + Observers
  + Events
  + Asynchronous Scheduling
* Algorithms
  + OOP
  + Pure/Impure Functional
* API
  + Messaging design pattern (MDP)
  + Client-Server Architecture

# System Validation

Whitebox

Since the extracted GeoJSON street data is not always perfect some test functions were used to ensure the generated adjacency matrix is correct:

* TestForEmptyRows() - ensures there are no unconnected rows in the matrix
* TestForDeadEndRows() - ensures every connected row has a connection out
* TestForSinglyConnectedRows() - ensures every row is connected to at least two other rows (eliminates subgraphs)

For the algorithms, the generated Hierarchical Decomposition Sequence (HDS) is the most important part when generating the top-down integral routing scheme.

* verify\_valid\_hds(G, hds) - ensures that the generated HDS is valid in graph G.

Blackbox

Standard blackbox quality assurance techniques were used on the simulators to ensure they function according to the specifications.

# Appendix

## Appendix A - UML Diagrams

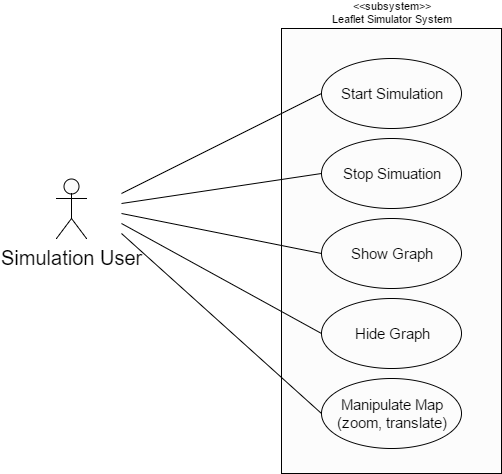
## TrafficSimClass.png

**Figure A.1 - Class Design of Simulator**

System Design.png

**Figure A.2 - General Architecture of whole system**

## UseCaseEntireProject.png



## Figure A.3 - Use Cases for Modules of the System

class_diagram.png

**Figure A.4 - Class Diagram for Algorithms Module**

## 

## Dev Traffic Sim Sequence Diagram.png

**Figure A.5 - Sequence Diagram for API Dev endpoints.**

## 

## 

## Appendix B - User Interface Design

UIDiagram.png

**Figure B.1 - User Interface for Web Based Simulator**

## 

## Appendix C - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

**Leaflet Simulator Install Instructions**  
  
Download the files  
  
- either download the zip or fork the repo  
  
 GOTO: (<https://github.com/FIU-SCIS-Senior-Projects/Traffic-Simulator-1.0>)  
  
- run your local server on your desired port  
  
 Ex. Type "python -m SimpleHTTPServer" in your cmd line (if osx or linux)  
  
- navigate to [http://127.0.0.1:port/](about:blank) in your browser  
  
 Ex.<http://127.0.0.1:8000/>

**Unity Simulator Install Instructions**

Download and install Unity3D v 5.5.0 or greater

Open Unity and create a new project.

Copy and paste the contents of the UnitySim folder into your project’s assets folder

**Algorithms Install Instructions**

1. Install Python 3.5/3.6
2. > pip3 install networkx
3. > pip3 install numpy
4. > pip3 install cython
5. Change directory into algos folder
6. Run > python3 setup.py build\_ext --inplace

**API Install Instructions**

1. Clone or download as zip the code base from the github repo as indicated above.
2. Python 3.5 is required to use flask.
3. The following packages are required to be installed:
   1. Flask
   2. Numpy
   3. Networkx
   4. Flask-limiter
   5. Flask\_cors
   6. Cython
4. Navigate to the \api directory
5. Execute the command:  
   python trafficSimFlask.py
6. The traffic sim server will be running on localhost:5000

**Headers Required for non-dev endpoints**

1. Api\_id: unique found in user\_data.json
2. Api\_key: unique UUID mapped to user in user\_data.json

These headers are required to access the non-dev endpoints, an example of a pair of values that are in the user\_data.json file:

1. api\_id : testuser1
2. api\_key : a798e3d9-3222-4ce6-908f-a08102ece1a3

**Currently Implemented Endpoints:**

1. /initialize\_graph, POST only
   1. Limited to: 20 per minute
   2. Content-Type: application/json
   3. Body:
      1. { “map”: [ [ < float> ] ],  
          “Algos”: [ <int> ] }
2. /initialize\_graph\_dev, POST only
   1. Same as above, but no limit
3. /init\_graph\_unity, POST only
   1. 1000 per second limit
   2. Body:
      1. { “map”: [ “row”: [ <int> ] ], “algos”: [ <int> ] }
4. /get\_path, POST only
   1. Currently no limit. 100 per second commented out.
   2. Body:
      1. { “algorithm” : <int>,  
         “Source” : <int>,  
         “Target” : <int> }
5. /get\_path\_dev, POST only
   1. No Limit
   2. Same body as get\_path