NETFLOW

A REALISTIC VIEW OF TRAFFIC DENSITY.

Team Members:
Reid Stagemeyer
Zoe Fu
Dat Nguyen

1. INTRODUCTION

1.1 PURPOSE

This is application will allows the user, Mayor Mann, manage mayor city traffic. It simulates the flow of traffic within a preloaded city map. Relevant goals and desired features of the application are explained below:

- Structure and organize the entire traffic flow for the city.
- Facilitate fast, unbiased, and accurate interactions at each intersection.
- Provide a flexible, automated, and interactive interface between the user and the program by showing live statistics of the traffic flow.

1.2 SCOPE

This project will allow Mayor man to investigate the impact of changing traffic component placement (stop lights, stop signs, etc.) at intersections to determine their optimal combinations/locations.

1.3 CORE OF THE SYSTEM

Application will be able to:

- Allow users to dynamically create the city map or load map from csv files.
- Allow users to adjust the start/stop locations and number of cars.
- Allow user to adjust the location and combinations of traffic components (traffic lights, stop signs).
- Allow users to run multiple traffic simulations.
- Provide a GUI that allows user to drag and drop traffic components.
- Provide a terminal window to show live traffic flow statistics.
- Display a full report of the traffic flow at the completion of a simulation run.

1.4 OBJECTIVES AND SUCCESS CRITERIA

Our application will closely simulate actual city traffic for the Mayor. This will give him the most realistic and accurate look of the traffic flows to find the optimal placement of traffic components. The optimal placement of components is defined as getting the most cars to their destination the fastest. The system should allow multiple simulations to be run with various combinations/placements to be analyzed. A simulation is finished once every car on the map reaches their destination, however the user can terminate the current simulation at any time. Many constraints will be included, first, all moving traffic components (i.e cars) will be moving at the same speed and making consistent turning decisions at each intersection. The simulation will be running in a one-time thread.

1.5 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

OOP: Programming language model organized around objects rather than “actions” and data rather than logics

UML: Diagram based on the UML (Unified Modeling language) with the purpose of visually representing a system along with its main actors, role.

Intersection: An area shared by two or more roads.

Tile: a smallest countable unit on the City Boost simulation map.

GUI: Graphical User Interface.

Time thread: the smallest sequence of programmed instruction that can be managed independently by a scheduler.

Turn: Action that each car agents will take at the intersection

1.6 REFERENCES

https://searchmicroservices.techtarget.com/definition/object-oriented-programming-OOP

https://tallyfy.com/uml-diagram/

2. CURRENT SYSTEM

N/A

3. PROPOSED SYSTEM

3.1 OVERVIEW

The proposed system must allow Mayor Mann to efficiently determine the optimal placement of traffic components (traffic lights, stop signs) throughout the city. This will be done by creating an application that lets Mayor man company run multiple simulations using various locations and combinations of those traffic components. He will also be able to change the number of cars and their start/end points. Various statistics and reports will be output to allow the user to determine what combination is most desirable.

3.2 FUNCTIONAL REQUIREMENTS

F1. Run multiple simulations

F2. Let user change the location of traffic components

F3. Let user change the combination of traffic components

F4. Analyze traffic performance

F5. Allow user to change number of cars

F6. Allow user to change start/end locations for each car

F7. Allow user to change the layout of the map

3.3 NONFUNCTIONAL REQUIREMENTS

N1 Usability:

- Our program will provide statistical data of running time along with customer's traffic combination decisions and speed of cars.
- User can run this program by some desktop applications, like Java applet.
- User will be able to see a 2D digital graph represented car moving.

N2 Reliability:

- The program will be able to recognize that when User enter starting points or destinations out of the map. It will give the warning and stop the program until getting correct input.
- The User will have control that the cars only move along the road and they cannot move out of map.
- The Program will make sure that user will only be able to set the stop sign or traffic light in intersections, and make sure every intersection gets one and only one stop sign or traffic lights. When an intersection doesn't get assigned with any traffic sign, the program will be able to automatically set a traffic light here. If an intersection gets 2 assigned traffic signs, the program will random select one sign here.

N3 Performance:

- Program must be able to work for multiple cars moving together.
- When the map changes, the program will also be able to work by implementing the new roads.

N4 Supportability:

- The program should be able to run on any platform supporting Java.
- Project will be able to work in some web application with a 2D visualization.

N5 Implementation:

- System will implement US traffic law. For example, at an alway stop sign, car agent will stop and follow the first come first serve service.

3.4 SYSTEM MODELS

USE CASE MODEL

1. Users load CSV file indicate # of cars, position of cars. Loads combinations of traffic lights and stop signs (optional: system set combinations randomly)

2. Start the simulation, users decide to press stop and change traffic combination. Restart at the end of simulation
3. Track progress and show statistic of traffic.
4. System shows final report of the traffic flow
5. Restart simulation

Use case above is described in figure 3.4.1

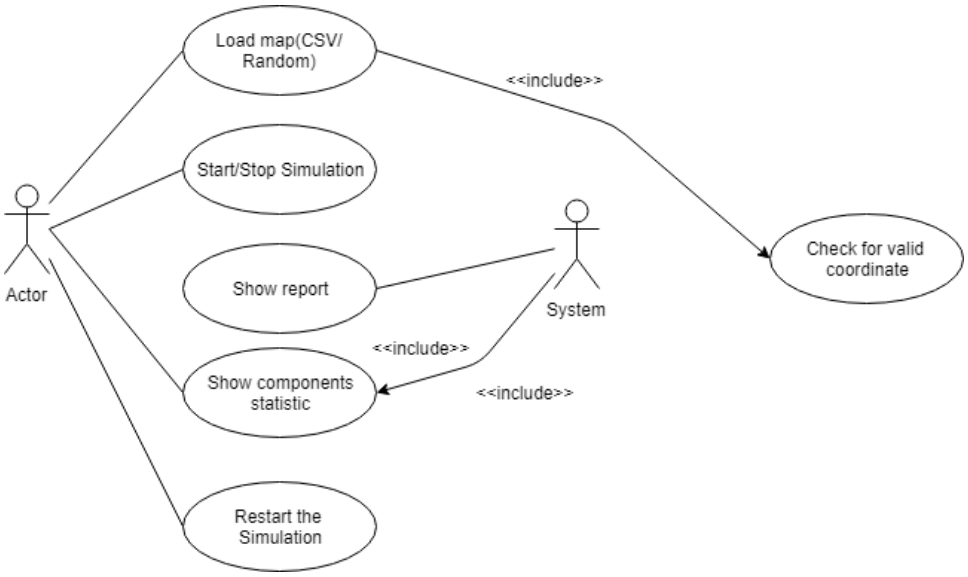


Figure 3.4. 1

STRUCTURAL MODEL

Class Diagram:

Abstract class Console will be the interface to run user command, and it implements CLI (Command Line Interface) and GUI (Graphic Line Interface) classes.

Car class will have basic information of every car component.

One Map will have multiple Tiles, which is an abstract class. A tile could be either ground, road, or intersections (traffic light or stop sign).

Interface report will show the statistic report of each car running time and final statistic report based on different combinations of traffic components. The Display class will show live map and traffic object on map.

Simulator will be our main class. It will be the event controller to deal with the change to every cars under different conditions.

Car, Simulator, Console, Display and Report will be associated with Map. Map has a public attribute to get carList, and it will be shared to every other associated classes.

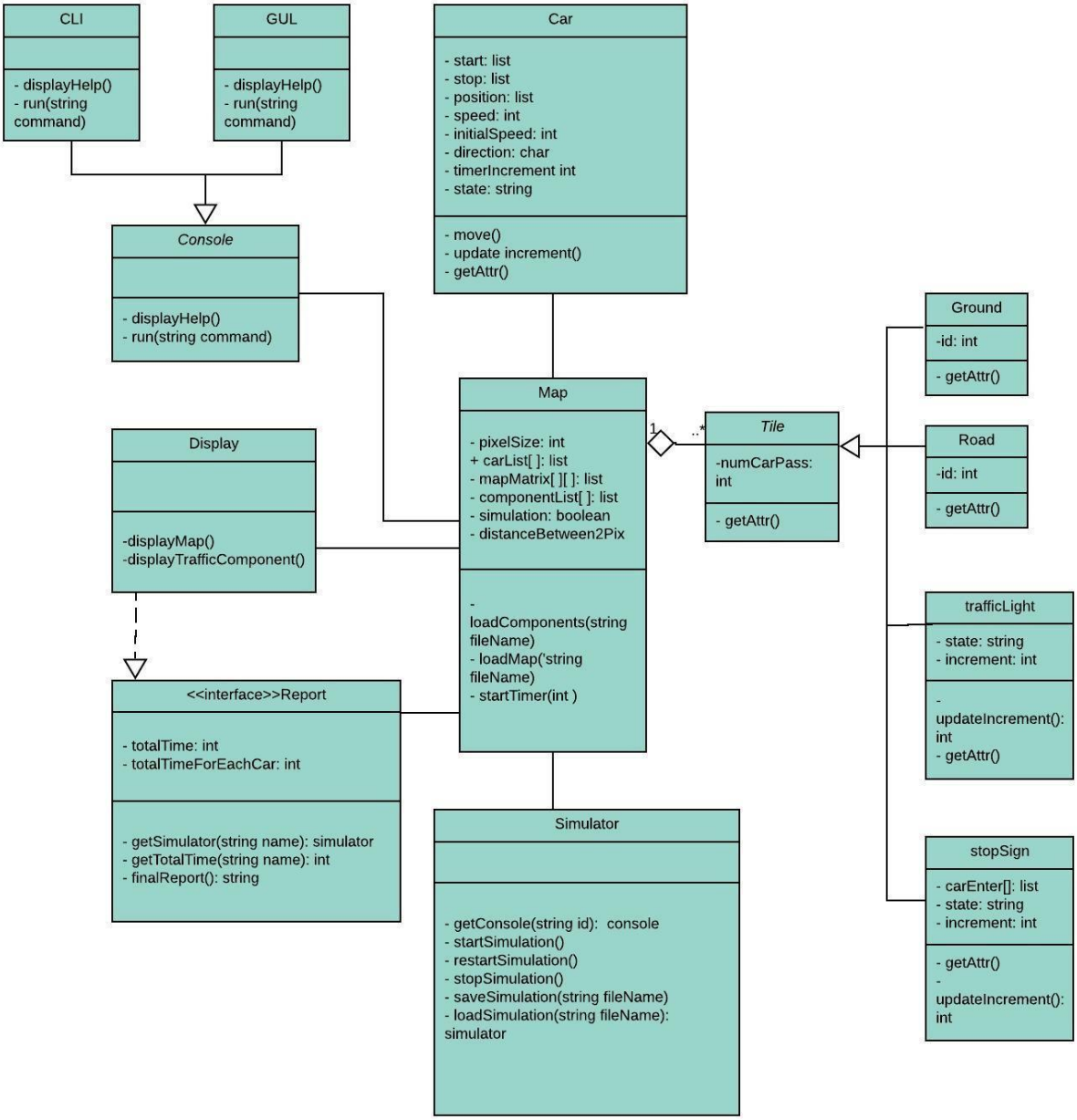

CLI
- displayHelp()
- run(string command)
GUL
- displayHelp()
- run(string command)
Car
- start: list
- stop: list
- position: list
- speed: int
- initialSpeed: int
- direction: char
- timerIncrement int
- state: string
- move()
- update increment()
- getAttr()
Console
- displayHelp()
- run(string command)
Ground
-id: int
- getAttr()
Map
- pixelSize: int
+ carList[ ]: list
- mapMatrix[ ][ ]: list
- componentList[ ]: list
- simulation: boolean
- distanceBetween2Pix
- loadComponents(string fileName)
- loadMap('string fileName)
- startTimer(int)
1
..*
Tile
-numCarPass: int
- getAttr()
Road
-id: int
- getAttr()
Display
-displayMap()
-displayTrafficComponent()
trafficLight
- state: string
- increment: int
- updateIncrement(): int
- getAttr()
<<interface>>Report
- totalTime: int
- totalTimeForEachCar: int
- getSimulator(string name): simulator
- getTotalTime(string name): int
- finalReport(): string
Simulator
- getConsole(string id): console
- startSimulation()
- restartSimulation()
- stopSimulation()
- saveSimulation(string fileName)
- loadSimulation(string fileName): simulator
stopSign
- carEnter[]: list
- state: string
- increment: int
- getAttr()
- updateIncrement(): int

Figure 3.4.2

BEHAVIORAL MODEL

Each use case is associated with individual sequence diagram as follow.

1. Load traffic components (Figure 3.4.2)

   To load the map along with its traffic components, the User Mayor Mann first makes a call to the function, 'loadMap()'. Next, he loads the cars and traffic components via their '.csv' files. These calls return the car and component objects back to the Map object, which then can display the loaded map to the user.

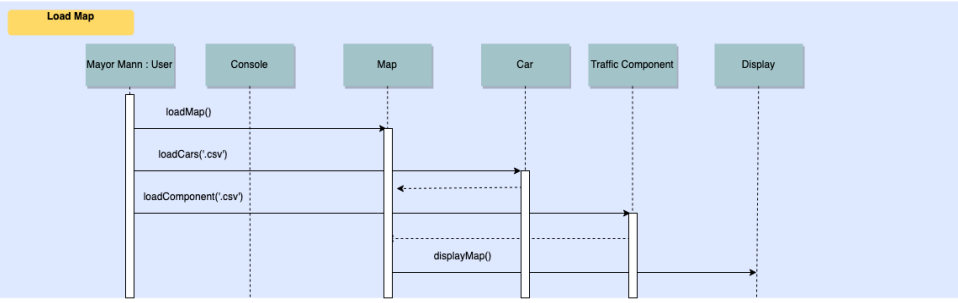


Figure 3.4.2

2. Start and stop simulation (Figure 3.4.3)

   The user chooses to start the simulation by either pressing a button or entering a command via the console which enters the 'running' loop and calls 'startSimulation()'. This starts incrementing the timer/counter which will be used to determine the position of all cars on the map and the timing for the traffic components.

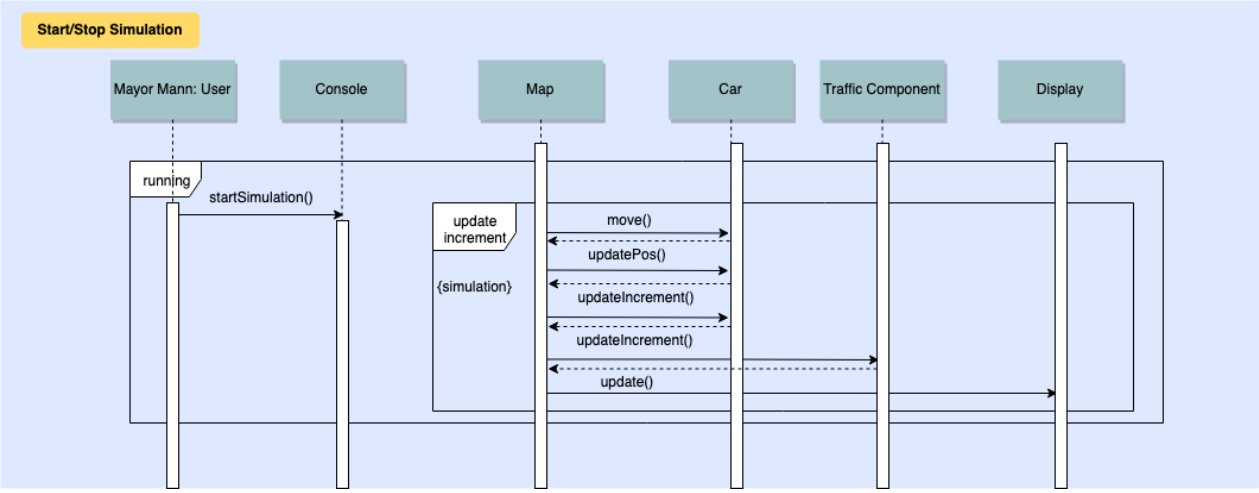


Figure 3.4.3

3. Track live report of each traffic component (Figure 3.4.4)

   To show live statistics, the Display class will make calls to the 'getAttribute()' function of both Car and Intersection (Traffic Component) respectively and display the results to the User.

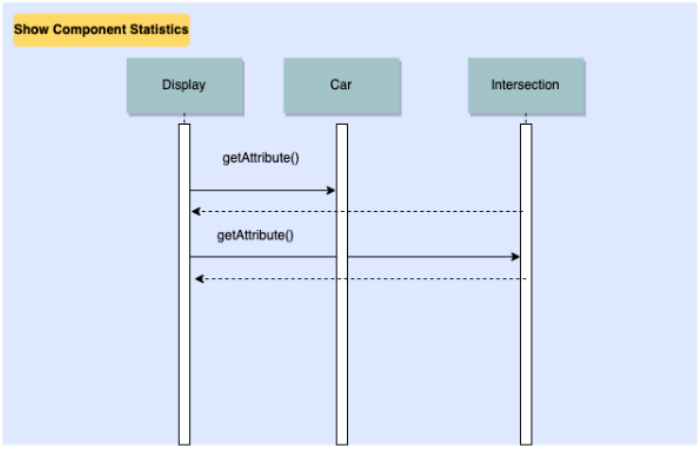


Figure 3.4.4

4. System show final report (as shown in figure 3.4.5)

   To display the final report, the display calls finalReport from the Report Interface which retrieves the car and traffic component statistics via 'getCarStat()' and 'getComponentStat()' functions respectively. The returned attributes to the Report Interface is coalesced and returned as a string to the Display.

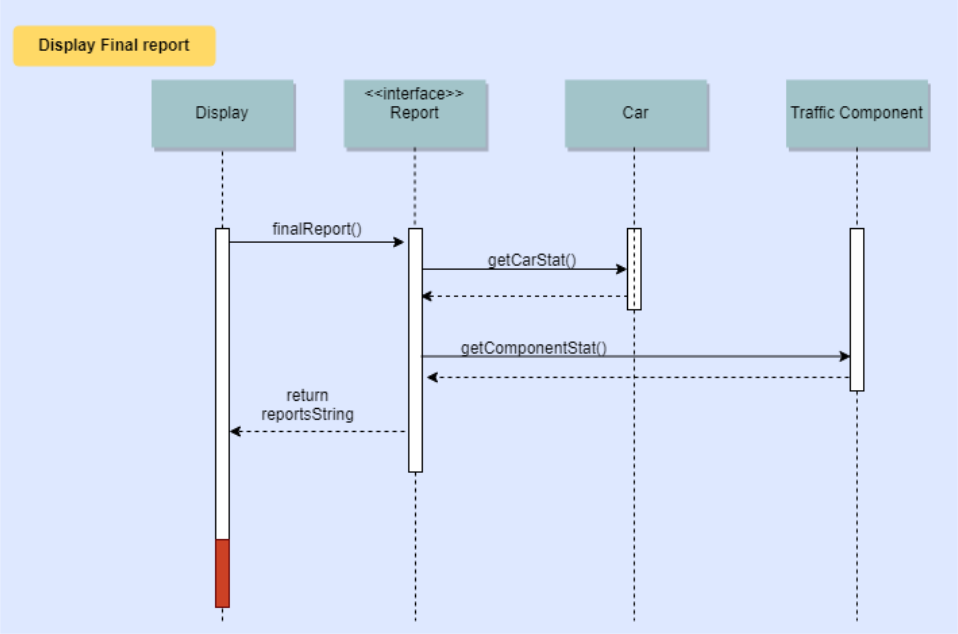


Figure 3.4.5

5. Restart the simulation(figure 3.4.6)

   Once all cars reach their destination it is considered over. The user can also choose to end the simulation early. To restart a simulation, the User either presses a restart button or enters a restart command via the console. This sets the simulation condition to true which reenters the simulation running loop.

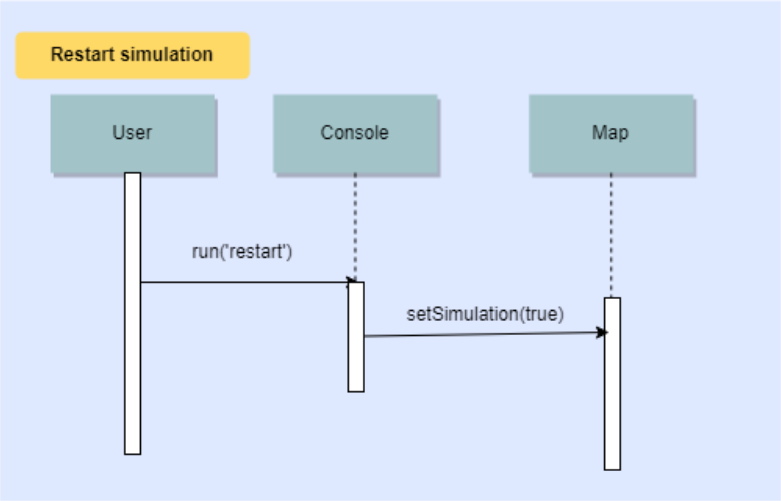
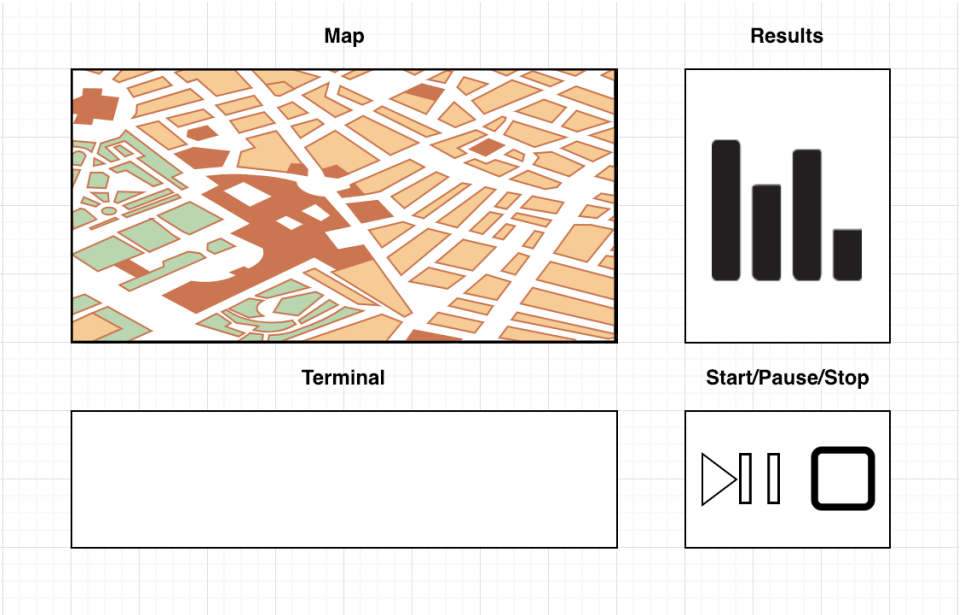


Figure 3.4.5: Sequence diagram for restart simulation use case

USER INTERFACE: NAVIGATIONAL PATHS AND SCREEN MOCKUPS



GLOSSARY

TBD