Tecnológico de Monterrey

City traffic

Francisco Márquez Bocardo - A01227697

Máximo Bautista Aguirre - A01228858

Jorge Andrés Pietra Santa Ochoa - A00226522

Ángel Cedeño Castro - A00344367

09/06/2020

1. Introduction

a. Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system. This document is meant to be used as a quick overview of the project by the professor of the class, also as a starting point for developers interested in collaborating.

b. Scope

This Software Architecture Document provides an architectural overview of the City traffic simulator. This document covers the information of the algorithms applied in the development of the City traffic simulator, as well as the technologies involved and the results obtained.

2. Architectural Representation

This document presents the architecture of the project; class diagram, flow diagram and use cases. These are diagrams on an underlying Unified Modeling Language (UML).

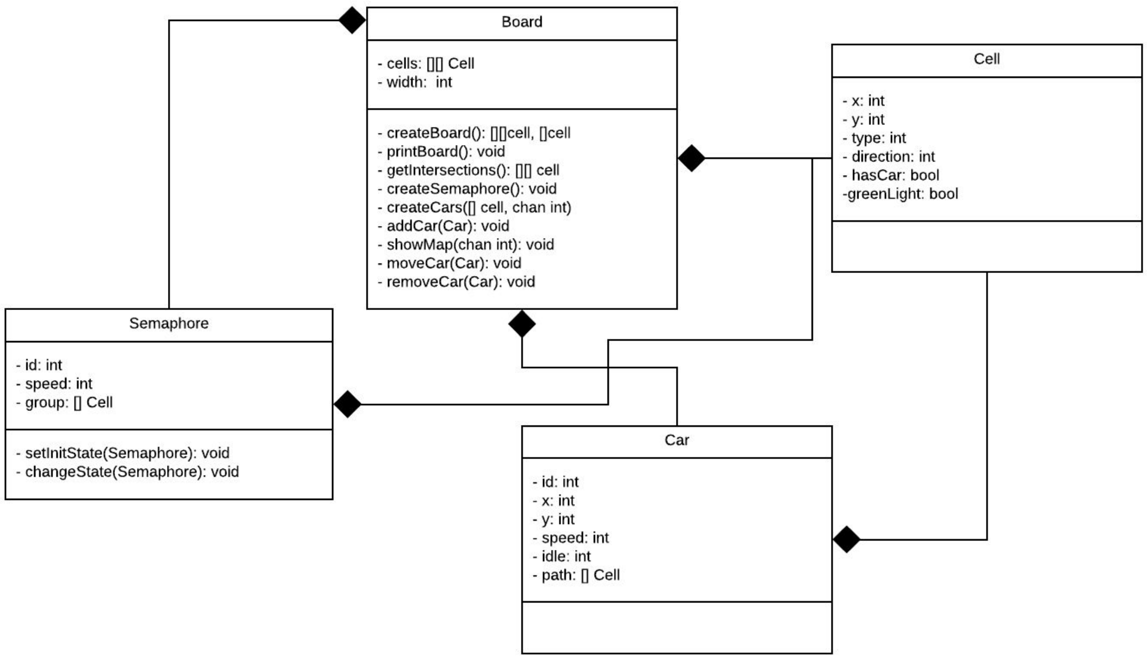
3. Architectural Goals and Constraints

There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

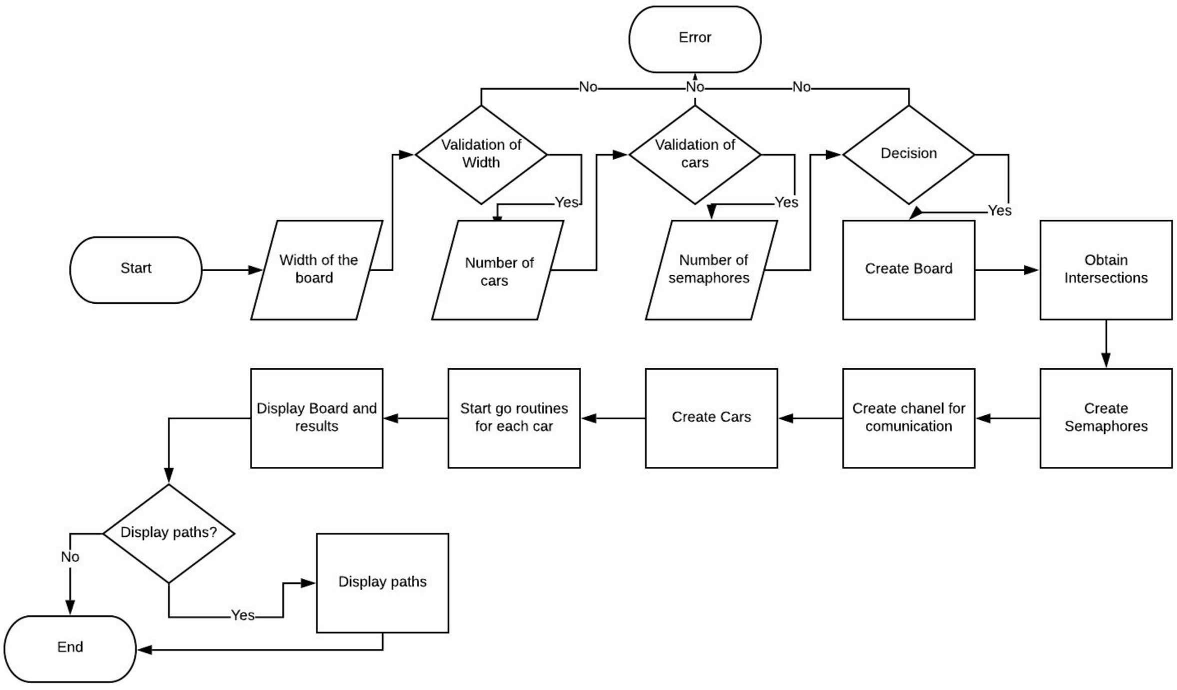
- The city's map can be static or automatically generated.
- Cars and semaphore numbers can be configured on game's start.
- For every car, define a random start and destination point.
- Define a random speed for each car.
- If a car detects another car on his route and it's slower, it must slow down its speed.
- Each car and semaphore behaviour will be implemented as a separate thread.
- Cars and Semaphores threads must use the same map or city layout data structure resource.
- Display finished cars' routes.
- Display each car's speed.
- Multithreaded core backend
- Program must be written in C or Go.

4. Design

a. Class Diagram



b. Flow Diagram



5. Implementation

The implementation of the software is done in Golang. The simulation is performed in a board, the board has a square form and there are different types of cells, there are streetcells where the cars can move, there are building cells that cannot be utilized by cars. Most of the cells can only be traveled in one way, except for the ones that are in an intersection that can be traveled in two different directions. The board is created using the same pattern every time, but the width and number of semaphores are provided by the user.

Every car has its own properties, such as position and speed. The movements of the cars are defined via a BFS that finds the path to follow, the starting and ending point of the car are randomly defined. The cars can’t move freely through the streets cells as there may be other cars on their way or semaphores in red. The cars are moved using multithreading, each own is simulated in a different goroutine.

6. Use Cases

Use Case 1	Simulate Traffic
Actor	User of the simulator
Basic Flow	The user inputs the width, cars and number of semaphores. The board is generated while the user sees a loading screen in the console. The simulation starts, and the cars move. The console is updating as the simulation advances. The simulation ends when all the cars reach its destination.
Alternate Flow 1	The user inputs a non valid width, the console throws an error to the user, specifying that the width must be a multiple of 7 with a final addition of 2. The program is terminated.
Alternate Flow 2	The user inputs a non valid number of cars, the console throws an error to the user, specifying that the number of cars must be greater than 0 but lower than the width. The

	program is terminated.
Alternate Flow 3	The user inputs a non valid number of semaphores, the console throws an error to the user, specifying that the number of semaphores must be greater than 0 but lower than the width. The program is terminated.

Use Case 2	Print Paths
Actor	User of the simulator
Basic Flow	The user inputs a “y” in the console. The path of every car is printed in the console.
Alternate Flow 1	The user inputs a “n” in the console. The program is terminated.

7. References

Bakery Algorithm
http://www2.cs.uidaho.edu/~krings/CS240/Notes.F13/240-13-13.pdf

Concurrent Programming
https://www.toptal.com/software/introduction-to-concurrent-programming

Stack
https://godoc.org/github.com/golang-collections/collections/stack

8. Source Code