# City simulation using multithreading

December 3, 2019

## 1 Introduction

Concurrent programming allows the creation of dynamic simulations that perform better due to the multiple calculations needed to be performed in real-time. This simulation focuses on the management of resources and how to avoid deadlocks altogether

## 2 Development

### 2.1 Architecture

The code for this project uses the Go programming language due to its features that allow easy management of threads. Altough the program lacks a GUI we improvised by using the terminal as the GUI itself.

### 2.2 Algorithms Implemented

The implementation uses the BFS algorithm to allow each car to find its destination. Every car starting position as well as the destination is randomized.

### 2.3 Resource management

The challenge for this project was how to handle and avoid deadlocks. Suppose that we have an intersection towards which 4 cars are heading over from every path. Without a semaphore, in the literal sense of the word as well as the abstraction used in concurrency, every car would wait for the other ones to proceed to continue their path; however every car would stay in this state forever since all of them are waiting for the street resource to be freed causing a deadlock which will affect also the cars that will try to enter the crossroad.

## 3 Results

The implementation of sempahores into the city simulation allowed us to create a more dynamic and fluid representation on how traffic generates along the city, the code for the project was also optimized using the Lamport's Bakery algorithm so the semaphores would synchronize accordingly to the status of the crossroad.

## 4 References

## References

[1] Lamport Bakery Algorithm https://www.geeksforgeeks.org/bakery-algorithm-in-process-synchronization/

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

## 5 Source Code

The code for this project can be found at https://github.com/DiegoDeDios/ap-labs/tree/master/challenges/city-traffic