Computational Modeling and Simulation of Road Traffic Dynamics using Unity Engine

AZOUBI Othmane*, HAJJOUT Safae*, YEFFOU Jaafar*

Supervisor: Pr. MAURER Alexandre

School of Computer Science - College of Computing

University Mohammed VI Polytechnic

College of Computing

University Mohammed VI Polytechnic

Date of submission: January 10, 2024

*: These authors contributed equally in this paper and the project

1. INTRODUCTION

1.1. PROJECT CONTEXT:

As second-year computer science students, our primary objective for this project was to pick a topic that would compel us into exploring new concepts as well as addressing real world problems, all while respecting the given timeline. That is when we started reflecting on the real life challenges our developing country, Morocco, faces. The ever developing framework of our country served as a compelling path. Once we observed the different existing issues, we opted for the topic of traffic management and road handling. Large cities' roads are often susceptible to accidents, traffic jams and tailbacks, especially during rush hours. Albeit it is partly due to driver behavior, an important culprit in these issues is faulty urban road design. Thus, this project combines elements of computer science, problem-solving and mathematics in order to study, case by case, what makes a road more or less prone to such complications. This project's main context as such is finding solutions on how to optimize road traffic.

1.2. WHAT THIS PROJECT AIMS TO ADDRESS:

In order to be efficient with our work, we recognized the necessity of breaking down our main problem into smaller more manageable parts that we resumed into the following questions: *How to limit the number of lanes in a road while still keeping it smooth? How to lay down speed limits and different traffic signs to help make roads more fluid? How to effectively calculate optimal trajectories for cars?*

These inquiries laid the foundation for three principal axes guiding our project's trajectory:

- **In-Depth Infrastructure Studies:**
  Engaging in a comprehensive examination of infrastructure elements (number of lanes, crossroads, roundabouts, traffic signs, etc.) necessary for fostering the fluid

functionality of a road network.

- **Optimal Trajectory Calculations:**
  Methodically calculating optimal car trajectories, a crucial element in enhancing road network fluidity and smoothness.

- **Analytic Road Studies:**
  Conducting analytical road studies to refine our findings based on various metrics and measures, a crucial step that will be elucidated further in the heart of the project.

1.3. METHODOLOGY FOR PROBLEM STUDY

Our approach synthesizes betweem mathematical principles such as graph theory, and mechanical physics for aspects such as speed calculations. Additionally, we made use of the A* pathfinding algorithm, to enhance the efficiency of our simulation. The integration the Unity engine facilitated real-time testing of simulation outcomes based on varying inputs, such as the number of cars and traffic signs. The comprehensive data generated is then systematically gathered through Python for detailed analysis, laying the groundwork for potential further optimization. This iterative cycle encapsulates the essence of our project's methodology:

Road selection $\rightarrow$ *Graph layout* $\rightarrow$ *Traffic setup* $\rightarrow$ *Simulation execution* $\rightarrow$ *Results gathering* $\rightarrow$ *Subsequent modifications*

1.4. CHALLENGES

Amongst the challenges we faced as we worked on this project, was the novelty of the material we were handling. Additionally, our approach as we started was overly ambitious. Our first idea was to establish a custom placer system within a grid and formulate unique calculations for numerous points. We thought this would be the simpler approach according to references we had seen. However, this proved to be a daunting task when scaled up to the design of all road elements, and it did not allow for as much flexibility in road design as our final approach did. Thusly, we ultimately made the decision to abandon this approach in favor of starting anew. This shift proved to be transformative, providing us with valuable insights and facilitating a more refined and effective development process.

2. THE PROJECT

The primary aim of this research is to craft and implement an all-encompassing traffic simulation system using Unity, meticulously addressing the intricacies of road representation, car behavior, and the effective application of traffic rules. This endeavor is driven by the overarching goal of contributing to the advancement of traffic management strategies and elevating the authenticity of simulated urban environments.

2.1. INITIAL STEPS

As previously stated, we had an initial approach in mind, despite it now being deprecated, it is still worth exploring within this paper as to see our thought process and delve deeper into the project's core.

2.1.1. Road Placement System

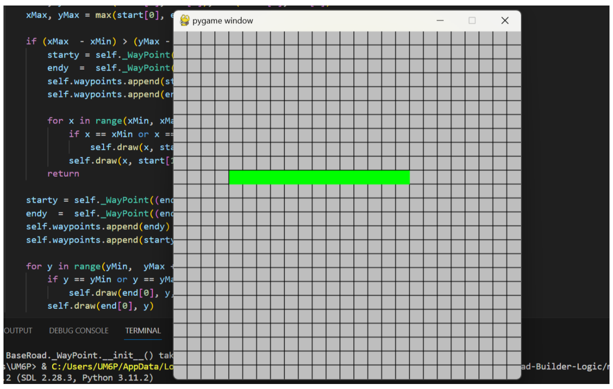
Our initial approach to structuring roads provided valuable insights into the refinement of our final product, albeit differing in two fundamental aspects. First and foremost, we adopted a grid system with a fixed n*n size for road placement, as opposed to a free-form graph. Secondly, the road wasn't designed and entered all at once; rather, it followed a block-by-block approach, reminiscent of the structure seen in some game development processes. This placement system, emulated through the grid structure, was meticulously implemented as follows: The base of this system is the Grid Singleton class, serving as the nucleus of the 'core logic.' This class is intricately structured as a graph, housing waypoints and their corresponding connections. For instance, Waypoint 1 might lead to Waypoint 2, and at an intersection, the final waypoint of one road seamlessly connects to the starting waypoints of other roads. Several critical considerations come into play when working within such a system:

- **Grid Size:** The grid maintains a constant size. The asset designer would then create square sprites aligned with the grid size unit. For instance, roundabouts necessitate a minimum of 4 grids, while simpler roads suffice with 1 grid.

- **Path finding:** The Grid class assumes responsibility for implementing routes and handling intricate behaviors, such as navigating through a roundabout and facilitating lane changes.

- **Element Placement:** The Placer class assumes a crucial role in the system, overseeing the precise placement of road elements on the grid. This includes ensuring proper alignment and establishing connections between roads and intersections.

This approach has its advantages, the user has more control over each node and how the cars will navigate the graph, it is also more intuitive from a user interface point of view as opposed to uploading your own road design for treatment using a third party software. However, this approach had its limits. For one, it was tasking to execute, it took us a considerable time to get the exact logic then implement, using python, a single lane. You may see in the figure below how our first placer prototype operated.



```
xMax, yMax = max(start[0], e

if (xMax  - xMin) > (yMax -
    starty = self._WayPoint(
    endy  =  self._WayPoint(
    self.waypoints.append(st
    self.waypoints.append(en

    for x in range(xMin, xMa
        if x == xMin or x ==
            self.draw(x, sta
        self.draw(x, start[1
    return

starty = self._WayPoint((end
endy  =  self._WayPoint((end
self.waypoints.append(endy)
self.waypoints.append(starty

for y in range(yMin,  yMax -
    if y == yMin or y == yMa
        self.draw(end[0], y,
    self.draw(end[0], y)
```

```
BaseRoad._WayPoint.__init__() tak
s\UM6P> & C:/Users/UM6P/AppData/Lo                 ad-Builder-Logic/r
.2 (SDL 2.28.3, Python 3.11.2)
```

Figure 2.1: Placement from waypoint A (leftmost) to waypoint B (rightmost)

As we starting exploring how roundabouts, crossroads and other such structures would work within this system, we found both design and code limitations. It also did not allow for flexibility in terms of road design, as the user had to work within a fixed size grid and had to already have a model envisioned in mind before starting work. The reason why we discarded this approach, instead going for a more flexible layout in which the user directly inputs an image of a road network to the software, and it gets treated according to added details as a graph structure. The waypoints logic was still preserved. This logic will be further explored in the road representation section.

2.1.2. Gipps Model

In terms of car behavior inner workings explorations. The Gipps Model was compelling as it had several similarities with what we desired and seemed integral to our pursuit of a realistic and dynamic traffic simulator. First and foremost, the Gipps Model is renowned for its capacity to realistically replicate individual vehicle behavior, offering a nuanced portrayal of how drivers adjust their speed and spacing within a traffic stream. This level of behavioral accuracy was deemed crucial for our simulation, aligning with our project's overarching goal of creating a lifelike representation of traffic scenarios. Additionally, the Gipps Model's versatility across various applications, including traffic flow simulation, transportation planning, safety analysis, and autonomous vehicle development, made it a promising candidate for our project. The prospect of employing a model with broad applicability added an extra layer of feasibility and adaptability to our simulation efforts. Its working is as follows: it is a model that has two main sub components which consist of acceleration and deceleration sub-models[1]. These sub-models are described by the following equations:

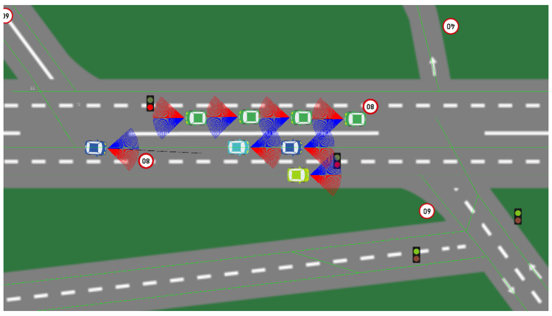
- **Acceleration Equation:**
  $v_n^{acc}(t+\tau) = v_n(t) + 2.5 a_n \tau \left(1 - \frac{v_n(t)}{v_n^d}\right) \sqrt{0.025 + \frac{v_n(t)}{v_n^d}}$
- **Deceleration Equation:**
  $v_n^{dec}(t+\tau) = -\tau d_n + \sqrt{\tau^2 d_n^2 + d_n \left\{2\left[x_{n-1}(t) - x_n(t) - S_{n-1}\right] - \tau v_n(t) + \frac{v_{n-1}(t)^2}{d_{n-1}}\right\}}$

According to these two equations, the speed and position of a car may be meticulously calculated at any given moment in time with high accuracy levels thanks to these two formulas.

- **Car Speed:** $v_n(t+\tau) = \min\{v_n^{acc}(t+\tau), v_n^{dec}(t+\tau)\}$
- **Car Position:** $x_n(t+\tau) = x_n(t) + v_n(t+\tau)\tau$

With:

- τ: Reaction time.
- $v_n(t)$ and $v_{n-1}(t)$: Speeds of vehicles n (follower) and $n-1$ (leader) at time t.
- d_n^v and a_n: Follower's desired speed and maximum acceleration.


(a) Small safety distance (car's raycast) due to cars stopping at red light

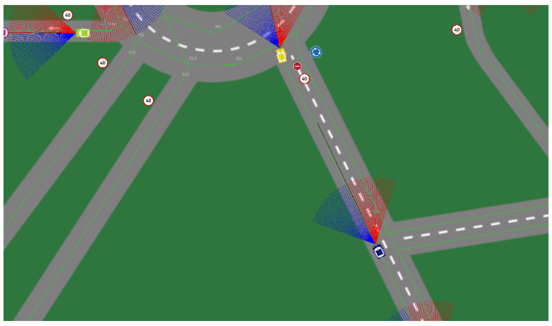

(b) Expanded safety distance due to car's speed (60)

Figure 2.2: Safety Distance Function

- d_n and d_{n-1}: Most severe braking the follower wishes to undertake and the leader's estimated most severe braking capability ($d_n > 0$ and $d_{n-1} > 0$).
- $x_{n-1}(t)$ and $x_n(t)$: Leader's and follower's longitudinal positions at time t.
- S_{n-1}: Leader's effective length, combining the leader's real length L_{n-1} with the follower's desired inter-vehicle spacing at stop s_{n-1} (between front and rear bumpers).

If vehicle n has a large headway, the minimum speed is determined by the acceleration equation, allowing the vehicle to accelerate freely towards the desired speed. In other cases, minimum speed is determined by the deceleration equation, allowing the follower to come to a stop using its maximum desired deceleration d_n, without encroaching on the safety distance. Assumptions and Derivations: *- Assumes the leader brakes according to d_{n-1}.*

- Follower cannot commence braking until a reaction time τ has elapsed.

- Implicitly allows for a safety margin (θ) in the reaction time.

- Gipps assumed θ to be $\tau/2$, hence it does not appear explicitly in the equations (Ciuffo et al., 2012).

The project's approach underwent a significant revision, marked by a conscious decision to simplify the Gipps model. This simplified approach considers the following key parameters: the car's desired speed, prevailing speed, braking force, and safety distance. Motivated by time constraints and the potential overcomplication of the project, this adjustment allowed us to maintain timelines and focus on critical aspects such as data analysis, traffic props, and path-finding algorithms. While imperfect, this renewed approach portrays closely how a car operates, as illustrated in the figure below.

2.2. WORKING WITH UNITY

Having outlined the considerations that preceded the development of our initial prototype, it is pertinent now to provide an inner insight into the operational framework of our project. To begin, we extensively utilized the Unity engine's tools[2], complemented by Object-Oriented Programming principles to develop our project.

- **Waypoint Class:**
  The Waypoint class is dedicated to handling road representation within our simulation. It plays a crucial role in defining the spatial layout and navigation routes.

- **Vehicle and VehicleSpawner Classes:**
  The Vehicle class, along with its counterpart, the VehicleSpawner class, collectively governs the dynamics of the car entities within our simulation. These classes manage the instantiation, behavior, and movement patterns of cars.

- **TrafficProps Class:**
  The TrafficProps class assumes responsibility for the integration and control of traffic signs within our simulation.

This strategic division into specialized classes enhances the modularity and maintainability of our codebase. It enables a systematic and organized approach to the implementation of distinct functionalities, promoting clarity and ease of development. For more details on code implementation, check the appendice 1 for the classes structure, and the link to the github repository for a complete overview.

2.2.1. Road representation

The Waypoint class forms the backbone of our simulation, playing a pivotal role in constructing the main graph. Its design simplicity is notable, with each node representing a waypoint possessing a dynamic list attribute storing connections.

The Vehicle class incorporates a comprehensive set of features, including acceleration, deceleration, movement control, and obstacle detection, providing a sophisticated emulation of real-world driving. Inspired by Gipps' model, the class integrates classical mechanical physics formulas with considerations for applied traffic laws, ensuring safety distances based on neighboring vehicles within the designated lane. The steering

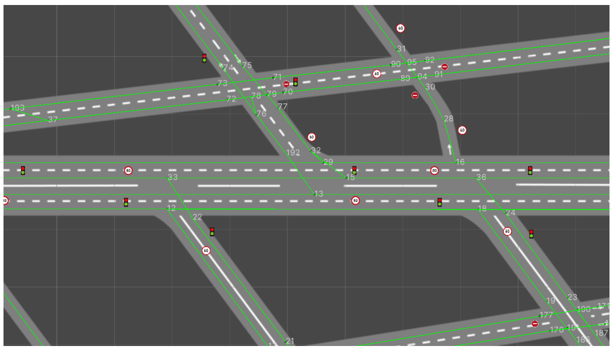


Figure 2.3: Waypoints Graph

mechanism, based on interpolation and dynamic recalibration, guides the vehicle's path through interconnected waypoints. The `FindNextTarget()` method employs the A-star algorithm[3] for intelligent waypoint navigation, ensuring a smooth run through the simulated road network. The `CheckForObstacles()` method utilizes advanced raycasting and trigonometric formulas for efficient obstacle detection, enhancing the vehicle's navigation in the virtual traffic environment and contributing to overall code realism.

The VehicleSpawner class has a straightforward role: placing car spawners at specific locations with initially configured parameters (entry point, exit point, max speed, etc.), one of the most important points is that each car is assigned a color based on its exit point, for facilitated analysis.

2.2.2. Enforcement of Traffic Rules

Orchestrated through the TrafficProps class, traffic rules enforcement involves rule-specific implementations, including StopSign, SpeedSign, TrafficLight, and Restriction. Each rule class adheres to the ITrafficRule interface, defining the ApplyRule method. This systematic design ensures a versatile and adaptable approach for seamlessly integrating various traffic rules within the Unity engine.

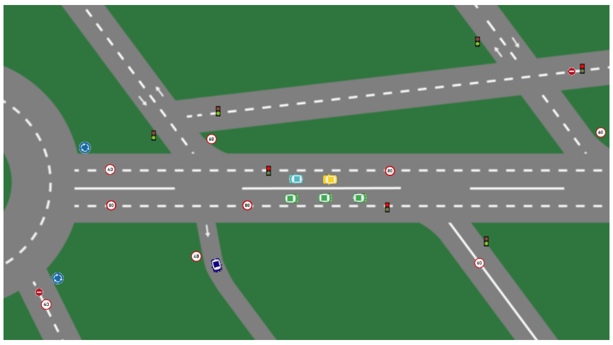


Figure 2.4: Cars stopping at red light

2.3. DATA ANALYSIS

Finally, after road, car and traffic sign implementation and placement. The only step left is to test the road the user has inputted. This process is accomplished in two steps: data collection using simple C# scripts that gather all of the necessary and wanted information, and then treatment via Python. For more details on implementation, see the github repository.

2.3.1. Analysis and Review

In our data analysis process, we utilized Python along with the ’pandas’ and ’matplotlib’ libraries for efficient analysis and visualization of our simulation data. Our primary graphs for the road study include:

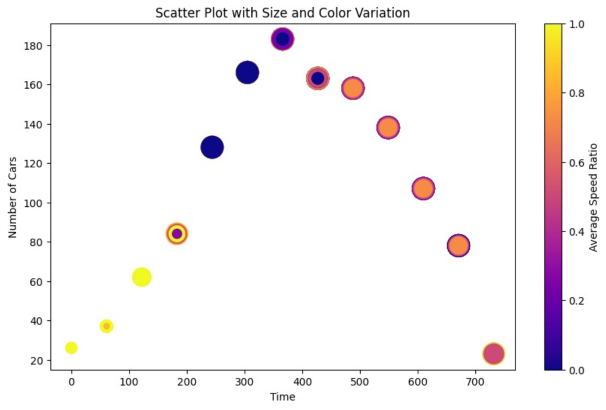
1. **Number of Cars vs. Average Speed:**
   A scatter plot graph to illustrate the correlation between the number of cars and their average speed over time, offering insights into traffic dynamics.

2. **Car Density by Exit Point:**
   A histogram depicting a car’s level of density by color, as in how many cars surround it, aiding in identifying congestion patterns near exit points.

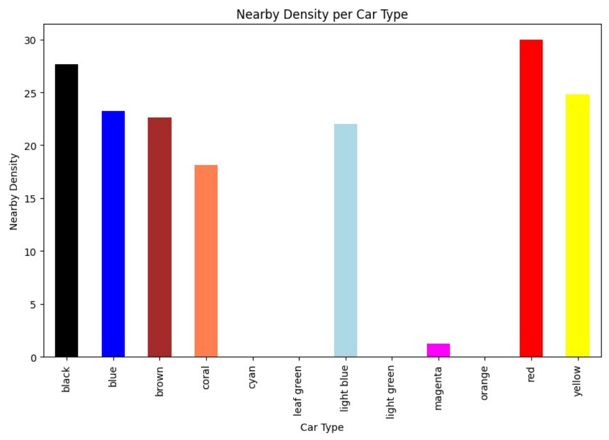
3. **Different Car Speeds by Color:**
   A histogram depicting the distribution of car speeds by color, providing insights into the range of speed values within the simulation according to each exit.

4. **Average Time Taken for a Car to Exit:**
   A histogram illustrating the the calculated average time taken for each car color to reach its exit point.

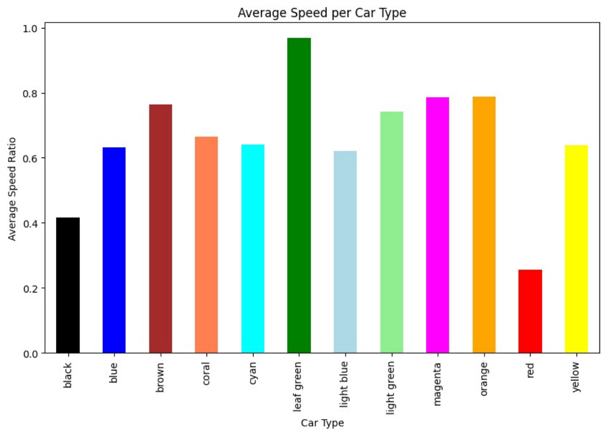
Our approach enhances the interpretability of the simulation data, allowing for efficient visualization and comprehensive analysis of various aspects, such as traffic flow, car density, and exit times.



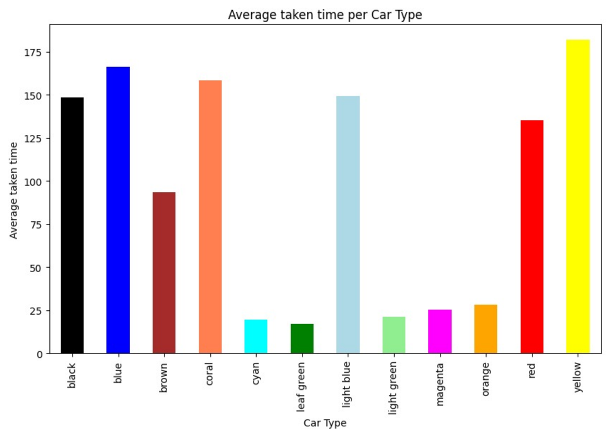
(a) Number of Cars by Average Speed



(b) Car Density by Exit Point



(c) Average Speed per Car Color (Exit)



(d) Average Time taken for a Car to Exit per Color

Figure 2.5: Example Graphs for the Barcelona road study

BIBLIOGRAPHY

[1] L. Vasconcelos, L. Neto, S. Santos, A. B. Silva, Seco, Calibration of the gipps car-following model using trajectory data, Transportation Research Procedia 3 (2014) 952–961. doi:10.1016/j.trpro.2014.10.075.

[2] Unity documentation (2024).
URL https://docs.unity.com/

[3] P. D. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions on Systems Science and Cybernetics 4 (2) (1968) 100–107. doi:10.1109/tssc.1968.300136.

APPENDIX

For more information on our code, please check the following github repository:

Traffic Simulator

APPENDIX 1: CLASSES

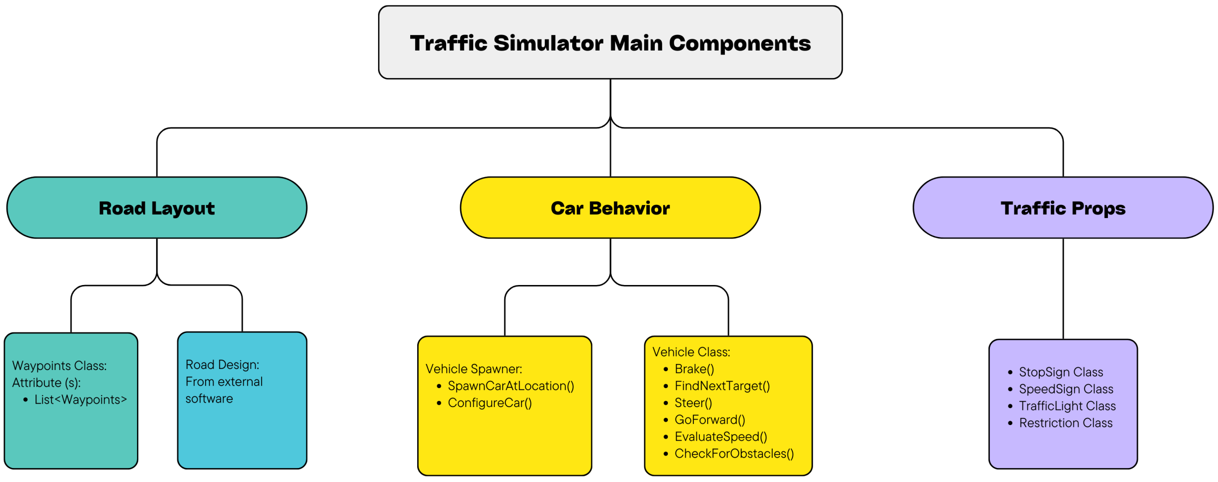

Traffic Simulator Main Components
Road Layout
Car Behavior
Traffic Props
Waypoints Class: Attribute (s): • List<Waypoints>
Road Design: From external software
Vehicle Spawner: • SpawnCarAtLocation() • ConfigureCar()
Vehicle Class: • Brake() • FindNextTarget() • Steer() • GoForward() • EvaluateSpeed() • CheckForObstacles()
• StopSign Class • SpeedSign Class • TrafficLight Class • Restriction Class

Figure 2.6: Classes structure