CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Development of a Traffic Simulation Tool for Analysing Road Network Performance

An intuitive tool for analysing traffic flows

Bachelor's thesis in Computer science and engineering

MARTIN BLOM
FELIX JÖNSSON
HANNES KAULIO
MARCUS SCHAGERBERG
JAKOB WINDT

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Bachelor's thesis 2023

Development of a Traffic Simulation Tool for Analysing Road Network Performance

An intuitive tool for analysing traffic flows

MARTIN BLOM
FELIX JÖNSSON
HANNES KAULIO
MARCUS SCHAGERBERG
JAKOB WINDT

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Developing a Traffic Simulation Tool
An intuitive tool for analysing traffic flows
MARTIN BLOM FELIX JÖNSSON HANNES KAULIO
MARCUS SCHAGERBERG JAKOB WINDT

Supervisor: Natasha Bianca Mangan, Interaction Design and Software Engineering
Examiner: Jasmina Maric, Interaction Design and Software Engineering

Bachelor's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2023

Abstract

With the rise of large and densely populated cities, increased vehicular traffic is becoming more common and with it comes an expected increase in traffic delays. The aim of this bachelor's thesis is to combat these problems by developing a traffic simulation tool, used to analyse road network performance in the hopes to foresee probable issues and help with finding solutions. The simulation tool was developed within Unity, using the programming language C# and allowed users to input traffic parameters such as traffic volume, vehicle types, and road infrastructure details. One of the main goals was for the tool to be intuitive and easy to operate for users. Therefore, a lot of weight was put into developing a user-friendly presentation of statistics. To help with achieving this, user tests were carried out and any interfaces in need of change were tweaked and refined according to the feedback. Moreover, the tool offered real-time visualisation of the simulation, which were often not found in similar tools. Finally, the results were visualised using graphs and charts to provide a comprehensive analysis of traffic flow, including aspects such as traffic density, average speed, and fuel consumption.

The results from the user tests showed that the tool was intuitive and easy to use, however lacked in feature density and simulation abilities when compared to already existing tools. To compete with these tools, the simulator would need a simplified mode for simulating large road networks.

Sammandrag

Med en ökning av städer med hög befolkningstäthet blir ökad fordonstrafik vanligare, och med det följer förväntade ökningar av transporttider. Syftet med denna kandidatuppsats är att bekämpa dessa problem, genom att utveckla ett trafiksimuleringsverktyg som kan användas för att analysera prestandan i ett vägnät. Detta i förhoppningen om att förutse potentiella problem och bidra med att hitta lösningar. Simuleringsverktyget utvecklades i Unity med programspråket C# och tillät användare att mata in parametrar som trafikvolym, fordonstyper och väginfrastrukturinformation. Ett av huvudmålen var att verktyget skulle vara intuitivt och lätt att använda. Därför lades stor vikt vid att utveckla en användarvänlig presentation av statistiken. För att hjälpa till med detta utfördes användartester och alla gränssnitt som behövde justeras förbättrades enligt feedbacken. Dessutom erbjöd verktyget realtidsvisualisering av simuleringen, vilket ofta saknades i liknande verktyg. Slutligen visualiserades resultaten med grafer och diagram för att ge en omfattande analys av trafikflödet, inklusive aspekter som trafiktäthet, genomsnittlig hastighet och bränsleförbrukning.

Resultaten från användartesterna visade att verktyget var intuitivt och lätt att använda, dock saknades en del funktionalitet i jämförelse med redan tillgänglig programvara. För att konkurrera med dessa verktyg skulle simulatorn behöva en förenklad simulering för att hantera stora vägnätverk.

Keywords: traffic, simulation, flow, emissions, vehicles, driving, transportation planning

Acknowledgements

We would like to express our gratitude towards our supervisor, Natasha Bianca Mangan, who provided great guidance through our development process. With her experience, she helped us with specialised advice in the development software used in the project. In addition, her feedback for this report was invaluable.

Special thanks go to our testers, without whom our tool would not have become as intuitive to use or as accessible for users without extensive experience in transportation planning. With their feedback we were able to improve the user experience.

Martin Blom, Felix Jönsson, Hannes Kaulio, Marcus Schagerberg, Jakob Windt
Gothenburg, May 2023

Contents

List of Figures

List of Tables

Glossary

A*: A popular graph traversal and path finding algorithm due to its completeness and optimal efficiency. A* is used to find the shortest possible path from one node to another.
Agent: Autonomous entities that inhabits an environment and act based on predefined rules.
Agent Based Model (ABM): A computer simulation model in which agents interact with each other and their environment to produce emergent behavior.
C#: C# is a general purpose programming language developed by Microsoft. C# is pronounced "C sharp" and belongs to the programming language family of C.
C++: One of the most popular general purpose programming languages. C++ belongs to the programming language family of C, with the major distinction from C that C++ is an object oriented language.
Data Structure: A way of organizing and storing data in a computer so that it can be accessed, manipulated, and modified efficiently. Some common examples are arrays, stacks, and linked lists.
FPS: Frames Per Second, is a unit used for measuring software performance, usually in games. It is the number of images displayed per second. A higher FPS results in a smoother experience.
Information Visualization: Field that focuses on creating meaningful and easy to interpret graphical representations of data.
MonoBehaviour: Base class for Unity scripts applied to in-game objects. Provides access to event functions and other predefined functionality used by the objects.
OSM: OpenStreetMap is an open source database containing real world map data.
Pooling: A technique used in programming to improve performance by reusing objects instead of creating new ones.
Prefab: A reusable object in Unity that stores a configuration and can be used as a template for creating assets.
Repository: A repository acts as a container that stores a project's files and their individual revision history.
Scrum: The scrum agile project management framework provides structure and instructions for management of work and is popular among software development teams.
Scrum-board: A bulletin board used for tracking tasks and planning development.
Story: In the scrum framework, a story is essentially a set of tasks that will result in a new or updated desired functionality/product.
UI: UI, or User Interface, is the connection between humans and computers. It determines how the user interacts with the program.
Unity: A cross-platform game engine used to develop games.
Unity Asset: A file containing reusable content that can be imported into Unity projects. Can be accessed through Unity's official platform or imported via third-party repositories.

User Testing: A method of testing and evaluating a product by observing and gathering data from real users.
UX: UX, or User Experience, refers to the overall experience for someone using a product. The goal of good UX design is to create intuitive and enjoyable products.
Pull Request: A pull request is a built in request to merge code from one branch into another in Git
API: API, or Application Programming Interface, is a way for different programs to communicate and send data to each other

1

Introduction

This section will present the purpose and goal of the project at large. It also introduces what the project tries to achieve. Finally, it will include some related work and the ethical aspects of this project.

1.1 Background

Traffic congestion is the result of the demand for road and railway travel exceeding the supply. This problem can be seen all around the world[1], and it impacts our quality of life. As vehicular traffic builds up, other vehicles like bikes, cars, buses and trams can move significantly slower resulting in increased delays and fuel wastage. Moreover, delaying transportation of goods can lead to an increase in economical costs, food waste and general inconvenience for the affected parties.

Not only does this affect our society on the larger scale, but also on an individual scale. If the average commuting time does not decrease, the average citizen will spend around a year of their life commuting. According to Trafikanalys' data of traffic habits[2], the average Swedish citizen's daily commuting time during 2019 was just under 1 hour and dropped to around 45 minutes post-COVID. Some amount of commuting time is inevitable in our current society. However, if you consider some of the larger cities in the world such as London, an estimate of 156 hours per person was lost in just traffic delay alone during 2022[1].

Other than leading to loss of time and resources, congestion and traffic in general leads to air pollution which poses health hazards and also lowers the quality of life[3]. A study of air pollution in connection to cars made in the USA during 2022, shows that transportation accounted for 27% of the total greenhouse gas emissions[4]. By reducing the amount of combustion sources that contribute to air pollution we can work on solving multiple problems at once.

In section 1.2 we will present how we aim to create a tool to help with both understanding why, and solving societal, economical and resource problems related to traffic and congestion.

1.2 Purpose

The purpose of the project is to design and construct a 3D traffic simulation tool with high accessibility, that should provide detailed and accessible data. This allows the user to evaluate the performance of different road networks and traffic scenarios, and make informed decisions about urban planning and infrastructure. Data should be presented in real-time through presentation of relevant statistics. By adjusting the parameters of the simulation, the user should be able to witness the effect of their tweaking, and easily see if their changes have a positive or negative impact across relevant environmental dimensions such as congestion level or emissions.

1.3 Related Work

This section will provide a review of various existing traffic simulation tools, their underlying models, and a comparison among them. We also go on to discuss the Agent-Based Modeling Framework for Unity3D (ABMU), and highlight some key features that this framework has implemented to allow for the successful creation of an ABM framework within Unity's environment.

1.3.1 Microscopic Traffic Simulations

There exists a plethora of different available tools for traffic simulation, which are in turn built upon different underlying models. In a paper from 2021[5], the authors classifies current available simulations according to the following four categories, with regards to granularity of each model: Macroscopic, Microscopic, Mesoscopic, and Nanoscopic. These tools allow researchers to answer complex questions and evaluate different scenarios in both real-time observations and through post-simulation data analysis. Agent-based traffic models position themselves within the Microscopic category and allow for a highly realistic representation of traffic flow. In this context, emergent behaviors such as congestion and bottleneck formation can occur due to the natural interplay of the autonomous agents within the simulation. A bottleneck formation refers to a localized area where traffic flow is significantly reduced, often due to factors such as road geometry, traffic incidents, or high demand. These bottlenecks can lead to increased travel times, reduced efficiency, and a greater likelihood of accidents[6].

Simulation of Urban Mobility (SUMO)[7][8] is a popular and freely available microscopic traffic simulation that was initially developed at the German Aerospace Center (DLR). It provides the users with the ability to model a range of transportation agents including cars, buses, bicycles and pedestrians, in both urban and rural environments. The simulation is deterministic by default, but users have the option to introduce stochastic processes in different ways, making it a highly versatile tool for traffic simulation and analysis. The software offers various tools for the creation of networks and allows the user to modify these networks through an editor. This editor can also import and export network data from external sources. In addition to this, SUMO provides the user with features for visualizing the obtained data and

analyzing it through various reports and plots. Users can also customize SUMO to accommodate their specific need through the application programming interface (API) and integrate the simulation with other software. SUMO is also capable of modeling emission based on vehicle type and speed.

Another popular solution for simulating traffic is the commercially available software PTV Vissim. This software is designed by the German-based company PTV group which specializes in mobility and transportation solutions. It is designed to be quick and simple to set up with no scripting required by the user, and comes with a highly customizable editor. The software is part of a larger suite named PTV Traffic Suite, which allows it to exchange data and collaborate across multiple platforms. PTV Vissim offers a similar feature list to SUMO but differs in some important areas. Firstly, they are built upon different car following models. SUMO implements the Krauss model which is based on the idea that drivers adjust their vehicle's speed and headway based on their perceived safety and comfort. Though a relatively easy to understand model, it has the disadvantage of assuming that the drivers only react to the speed and distance of the vehicle in front of them, and excludes a lot of factors such as the traffic signals, shape of the road, and driver psychology. Meanwhile, PTV Vissim implements the Wiedemann model[9] which share a lot of the same model parameters as the previously mentioned Krauss model but differ in their mathematical formulations and the way they calculate the acceleration of a vehicle. The model also introduces additional parameters, for example, a parameter for setting driver aggressiveness which regulates how risk-taking a driver is willing to be, and a parameter to regulate reaction time. Due to the additional parameters introduced here, the Wiedemann model is considered more realistic compared to the Krauss model, but is at the same time deemed to be more complex and requires a significant amount of parameter calibration.

A crucial difference between the two simulation tools is that SUMO natively only supports graphical representation of a traffic environment in low detailed 2D, while PTV Vissim offers a feature rich 3D visualization. The latter provides a range of tools for customizing the 3D visualization, including options for importing third party 3D models, setting and creating custom textures, and defining various customized visual effects. This advanced visualization capability can significantly enhance the user experience and facilitate a more intuitive understanding of the simulated environment, which can prove to be beneficial when used for public presentations and stakeholder engagement.

1.3.2 ABMU

Agent-Based Modelling Framework for Unity3D (ABMU) is an open-source 3D agent-based modeling platform developed with the Unity3D game engine[10]. It was developed as a response to the lack of support for 3D ABMs[11], and offers an extendable and user-friendly programming interface for Unity's resources to create the foundation for a powerful and extendable model[12]. This framework gives researchers and developers the tools to create highly immersive and visually appealing simulations that can provide deeper insights into complex systems and provide more

effective communication of results.

Some key features of ABMU include event scheduling and synchronous updates by a dedicated scheduler class, to delegate events in a manner that is decoupled from Unity's native event execution order. This ensures a more robust and accurate simulation. These events get delegated to so called Steppers, which are modular components that encapsulates a targeted behavior or action. The events are made to be easily added and removed to enable researchers and developers to adjust parts of their simulation without the need the modify the core structure of the model. Furthermore, ABMU introduces wrappers around native Unity methods, enabling them to be used as Steppers within the framework. The benefit of this is allowing for easy extension using existing Unity libraries which can provide complex behaviors such as advanced pathfinding and physics simulation systems.

ABMU not only offers a powerful and extensible platform for creating agent-based models, but also includes a diverse collection of example models to showcase its flexibility and potential applications. These examples includes demonstration of model implementations such as Epstein and Axtell's Sugarscape model[13], Reynolds' Boids model[14], and Schelling's segregation model[15]. The models demonstrate the flexibility and capability of the framework, as well as offer guidance for users looking to develop their own models following best practices.

1.4 Societal and ethical aspects

Ethical aspects can be broken down into two parts: aspects related to the method of the project, and possible consequences for users of the final product and society as a whole. With regards to making sure that our methods adhere to an ethical practice, the main thing to be aware of here lays in data handling during user testing. It will be crucial to ensure that data is both collected and stored in a responsible manner. This will involve structuring clear and properly formatted consent forms[16], anonymizing data, and adhering to relevant data protection regulations such as the General Data Protection Regulation (GDPR)[17].

The ethical aspects of the finished product however, is accompanied by more complex considerations. One of the goals of this project is to create a tool that can be used by end-users of various occupations connected to traffic planning, and offer these users insight about the efficiency and emission associated with different road network configurations. Since these insights might lead to real-life decisions regarding actual infrastructure, careful consideration will have to be taken regarding the design we choose to implement and what sort of consequences these might have in the finished product. To instill model credibility and prevent model realism bias, we will have to communicate any assumptions and limitations of the model in an easy and accessible manner. ABMs are generally considered challenging with regards to validation and traceability[18], and failing to mitigate these might lead to decisions being implemented on obscure premises.

2

Theory

This section lays the theoretical foundation for the essential concepts that underpin the implementation of our agent-based traffic model in Unity. It covers a variety of topics, including specific mathematical functions, code design principles, pathfinding algorithms, and the theoretical aspects of software development project management frameworks. By establishing a solid understanding of these core concepts, readers will be better prepared to grasp the intricacies of the traffic simulation presented later in this thesis.

2.1 Unity

A game engine is a software framework designed to facilitate the creation of video games by providing the most commonly needed functionalities. These include complex tasks such as physics calculations, animation, rendering, and artificial intelligence. The advantage of using a pre-existing game engine is that it allows developers to focus on the unique aspects of their game and accelerates the development pipeline, as they do not have to code these complex systems from scratch.

Unity is a game engine initially released for Mac OS X in 2005 at the Apple Worldwide Developers Conference[19]. The CEO of the company has said that the mission was to democratize game development, making it widely accessible to a broad audience[20]. By 2022, the company had secured a significant market share of 38%, signifying its wide acceptance and popularity within the industry[21].

Unity can be used to produce both 2D and 3D environments, and it offers native support for a wide variety of platforms and operating systems. Although there is a wealth of underlying theory supporting the engine, of which the most prominent is considered to be the Component-Based Object Model. Development in Unity is centered around so-called GameObjects, which serve as the base class for all entities in Unity scenes. These GameObjects can then receive different components, which can take the form of a wide range of things, such as scripts, textures, cameras, and so on. This pattern allows for a flexible and modular development approach, as many GameObjects can reuse the same component while customizing each component parameter to their specific requirements.

Another central underlying theory is that of Event-Driven Programming and the

implementation of a so-called game loop[22]. By using an event-centric communication method for many of its native systems such as input handling and collision detection, components such as scripts can define methods that hook onto the event architecture. These can then respond to specific events such as "OnCollisionEnter", which is called when a collision occurs[23][24]. This Event-Driven approach works in symbiosis with the theory of a game loop. A game loop is a ubiquitous architecture technique used within the game engine sphere. The basic concept is that an update event occurs each frame, also called a tick, where all GameObjects and their components have the opportunity to react and update themselves according to the current state of their environment. Since these updates occur with high frequency, often many times per second, this allows the game to simulate real-time behavior.

2.2 Bézier curves

The project requires the creation of roads, which follow paths. These paths need to be represented somehow. A type of representation is through a mathematical definition that describes the path. This allows it to be graphically drawn by the computer, as well as for calculations to be made in order to find points along the path. A common path representation is as a Bézier curve, which is useful for the project in defining the roads and enabling editing of their paths. To receive a clearer understanding of the math behind Bézier curves it is recommended to use an interactive tool while reading the following section. This gives a visual representation of the mathematics and helps understanding the following chapters. One of these tools is the Bezier Playground[25].

A Bézier curve is a parametric curve between two points, that curves according to a set of intermediate points[26]. The points are called control points, where the first and last point are the endpoints of the curve. A linear Bézier curve only has two points, which means that it is a line between the points. It is defined by the following function:

$$P(t) = P_0 + t(P_1 - P_0), \quad 0 \leq t \leq 1$$

Note that $P_1 - P_0$ is the vector starting in P_0 and ending in P_1. The parameter t is the ratio along the line, with $t = 0$ and $t = 1$ marking the endpoints. This is what is known as linear interpolation in mathematics. A linear Bézier curve is therefore simply a linear interpolation between the points P_0 and P_1. Let us define this as $P_0 \rightarrow P_1$. A visual representation can be found in figure 2.1.

Expanding on this, a quadratic Bézier curve consists of two linear interpolations:

$$A) \quad P_0 \rightarrow P_1$$
$$B) \quad P_1 \rightarrow P_2$$

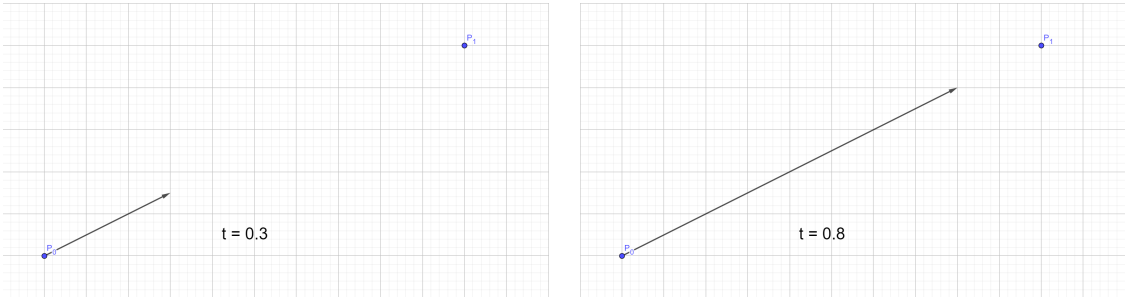

Figure 2.1: A linear interpolation for two values of t

It is then defined as the linear interpolation between these points, i.e $A \to B$. All linear interpolations in this case depend on the same t, which is what creates the curvature of the Bézier curves.

Since a quadratic Bézier curve has three points, it will have two endpoints as well as an additonal control point between them. By moving the control point, the shape of the Bézier curve can be altered. This is presented with the examples in figure 2.2, the first three of which have static endpoints demonstrating how the middle control point can be used to form the curve. The final example eludes to the fact that the control points can be placed anywhere without the requirement of any order.

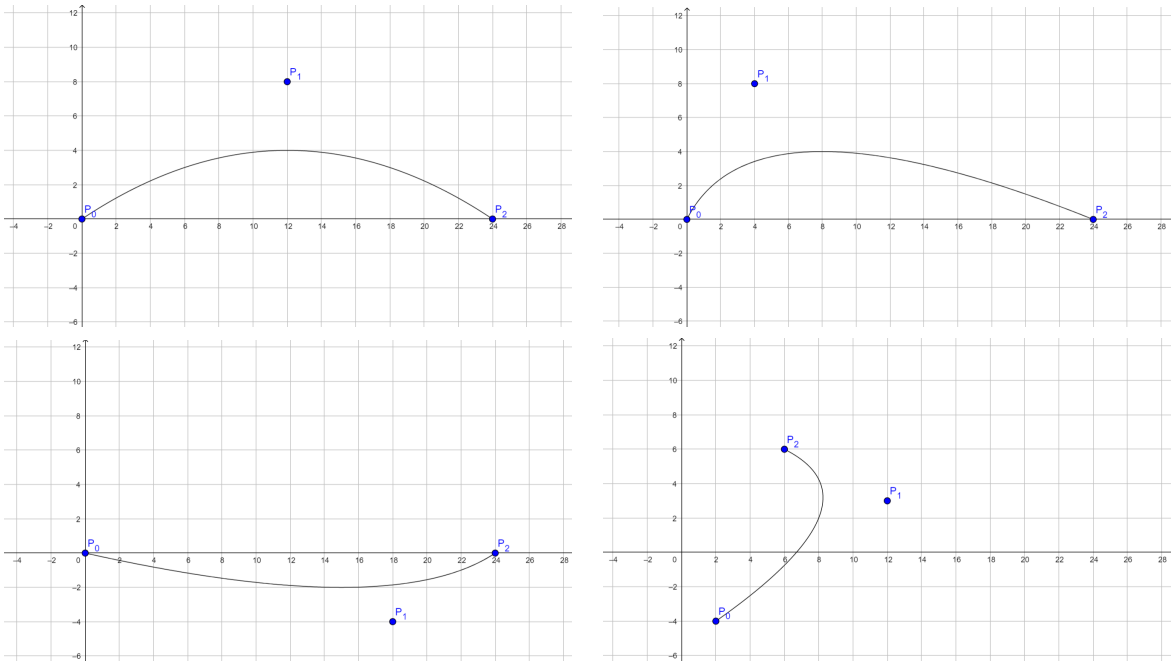

Figure 2.2: Four examples of quadratic Bézier curves

2.2.1 Cubic Bézier Curve

A cubic Bézier curve expands on the quadratic curve in the same fashion as the quadratic expanded on the linear Bézier curve, adding another layer of linear in-

terpolations. A cubic Bézier curve has four control points, two of which are endpoints.

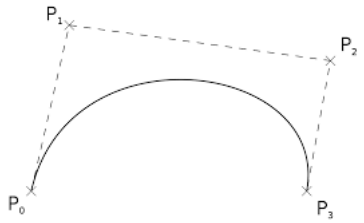

Figure 2.3: Cubic Bézier curve with control points P_0, P_1, P_2 and P_3

The cubic Bézier curve can be defined by the formula[27]:

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3, \quad 0 \leq t \leq 1$$

Properties of the Cubic Bézier curve relevant to this paper are the following:

1. The endpoints P_0 and P_3 lay on the curve
2. The curve is continuous, infinitely differentiable, and the second derivatives are continuous.
3. The tangent line to the curve at the point P_0 is the line P_0P_1. The tangent to the curve at the point P_3 is the line P_2P_3.
4. Both P_1 and P_2 lay on the curve only if the curve is linear.
5. A Bézier curve is contained within the convex hull of the control points. For the application in Unity, this means that a Bézier curve is completely contained within the bounding box created by its control points.

2.2.2 De Casteljau's algorithm

In 1959, the french mathematician Paul de Casteljau constructed an algorithm for dividing a Bézier curve into two. The union of these Bézier segments is equivalent to the original curve. With this algorithm it is possible to split Bézier curves into sections and is a core part of the Bézier Clipping algorithm.

2.2.3 Bézier Clipping

Finding the intersection points between two Bézier paths is not as straight forward as for something like two lines. To solve this, an algorithm called Bézier clipping explained in a publication by Thomas Sederberg can be used[28]. It utilises the convex hull property of Bézier curves - and therefore Bézier paths - as well as de Casteljau's algorithm for splitting curves. An implementation for finding all intersection points between two Bézier paths using Bézier clipping is outlined below.

```
intersections ← Empty list
epsilon ← A value >0 small enough for the desired accuracy
procedure FindBezierPathIntersections(A, B)
    if A.BoundingBox does not intersect B.BoundingBox then
        return
    end if
    if A.BoundingBox.Size + B.BoundingBox.Size < epsilon then
        intersections ← Midpoint between A and B
        return
    end if
    A_1, A_2 ← SplitWithDeCasteljau(A, 0.5)
    B_1, B_2 ← SplitWithDeCasteljau(B, 0.5)
    FindBezierPathIntersections(A_1, B_1)
    FindBezierPathIntersections(A_1, B_2)
    FindBezierPathIntersections(A_2, B_1)
    FindBezierPathIntersections(A_2, B_2)
end procedure
```

Worth noting is the *Midpoint*, which is one of many possible approximations of the intersection point. For a small enough *epsilon*, the approximation used is trivial as a segment approaches a point as *epsilon* approaches 0.

2.2.4 Composite Bézier curve

A composite Bézier curve is a spline made out of Bézier curves. The series of Bézier curves are joined together end to end with the start point of one curve coinciding with the end point of the previous curve. This is used in the project as it allows for chaining of cubic Bézier path segments, creating a spline.

2.3 A* Algorithm

A* is an algorithm widely used for pathfinding and graph traversal. Peter Hart, Nils Nilsson, and Bertram Raphael first presented the algorithm in 1968 as part of a project focused on constructing a mobile robot capable of autonomously devising its actions[29]. It is classified as an informed search algorithm since it greedily explores the pathfinding environment by taking into account both the cost of the path from the starting node to the one that is currently being explored. A heuristic function that estimates the distance between the currently explored node and the goal node is also used[30]. Given a start and end node in a weighted graph, the algorithm will find the shortest path between the nodes. Together, these two form an estimate function of the best path towards the goal. A* is complete under the precondition that the search space is finite, and the branching factor is also finite, which guarantees that if a path exists, it will be found. Furthermore, if some additional conditions are fulfilled with regards to the heuristic function, A* can be guaranteed to return an

optimal path. For this to be the case, the heuristic function needs to be admissible or consistent, since a consistent function is also, by definition, admissible[31].

In the project, we aim to implement the A* algorithm to find the shortest path on a graph consisting of nodes representing intersections, road ends, and points of interest (POIs). Here, POIs primarily serve as target destinations for our agents and may include parking lots, fuel stations and other relevant locations. By implementing the A* algorithm, we can develop dynamic heuristic functions that adapt to real-time events such as traffic accidents or road closures. This adaptability will lead to a more responsive system, ultimately improving the overall performance of the traffic simulation.

In order to implement the A* algorithm, an open and closed set of nodes are utilized, as well as a few essential variables and functions. These are the key elements used in the algorithm:

- Start node s: The initial position from which the search begins.
- Current node n: The node being evaluated during the search process.
- Target set T: Contains one or more goal nodes that the algorithm is trying to reach.
- Total estimated cost function $f(n)$: The sum of the cost from the start node to node n (denoted by $g(n)$) and the heuristic estimate of the cost from the node n to the target node (denoted by $h(n)$).

With these definitions in place, the A* algorithm can be described with the following procedure:

1. Label the start node s as "open" and compute $f(s)$.
2. Choose the open node n with the smallest $f(s)$ value. In the case of a tie, the node is chosen randomly, but always prioritizes nodes in the target set T.
3. If n is in T, label n as "closed" and conclude the algorithm.
4. Otherwise, mark n as closed and generate all adjacent nodes by exploring the neighboring nodes that can be reached from n in the graph. Compute f for each adjacent node of n and label each adjacent node not already marked closed as "open". If a closed node n_i is an adjacent node of n and its current $f(n_i)$ is smaller than its previous f value when it was marked closed, relabel it as "open". Return to Step 2.

2.4 Procedural mesh generation

All physical objects in Unity have an associated mesh, i.e. their surfaces. A cube for example can be thought of as having a mesh consisting of 6 different surfaces. In computer graphics, a triangle mesh is a type of mesh where the surfaces are created

through a set of points, called vertices. These vertices are then joined together by a set of triangles. Returning to the cube example, a cube in its simplest form would have 12 triangles and 8 vertices. The eight vertices are at the corners of the cube. Each face of the cube has the shape of a square, which can be created with two triangles, hence double the amount of triangles as square faces.

Procedural mesh generation is the act of generating these meshes through a script. This is done mainly for the roads, where the mesh has to be rebuilt whenever the road is changed. The road meshes are generated along the Bézier curves that define them, creating the vertices from points offset along the normal of the curve.

2.5 Scrum and Agile Software Development

Agile Software Development is a software development framework which emphasizes vertical development cycles, where software should be delivered frequently in atomic slices to enable quick feedback and high flexibility with regards to how the product develops. When developing complex products, and especially when the development team has not worked on anything similar to the current developed product, implementing an agile framework can be particularly important. Since features are delivered in small complete chunks, it minimizes the investment risk compared to a more horizontal feature development.

2.6 ABM

Agent-Based Modeling (ABM) is a computational modeling approach that facilitates the analysis and simulation of complex systems by depicting a system's individual elements (agents) and their interactions[32]. This method enables researchers to investigate how the combined behavior of a system emerges from the attributes and actions of its individual components. In contrast to conventional models, which typically depend on mathematical tractability and differential equations for portraying behavior from a macroscopic viewpoint, ABMs face fewer restrictions and can encompass more aspects of real-world systems[33]. As a result, these models can simulate intricate scenarios without relying on equally complex mathematics. It achieves this with satisfactory, and sometimes, even more precise outcomes compared to models that overlook the individual behaviors ABMs are capable of representing. It should be noted, though, that ABMs can also integrate more sophisticated mathematics and techniques, like neural networks or advanced learning approaches, to more accurately depict the complexities and dynamics of individual agents within the system.

2.6.1 Key features

ABMs consist of individual agents that interact with each other and their environment. Agents can represent various entities such as organisms, humans, businesses, and so on. These agents are characterized by their uniqueness, local interactions,

and autonomy. They can have different attributes such as size, location, and resource reserves, and they interact with their neighbors in a specific "space", such as a geographic area or a network[32]. The mentioned space is typically relatively small in the scope of the total simulation space. Agents act independently and pursue their own objectives, adapting their behavior according to their current state, the state of other agents, and their environment.

2.6.2 Emergence and across-level modeling

ABMs are particularly useful for studying emergent system behaviors that arise from the interactions and responses of individual components to each other and their environment. This allows researchers to explore how a system's dynamics are linked to the characteristics and behaviors of its individual components. Due to this, ABMs are considered across-level models because they focus on the interactions between the system level and the individual agent level[32]. In these across-level models, the agents' behaviors and decision-making processes are modeled explicitly by the researchers, while the emergent properties of the system as a whole stem from these micro-level interactions that occur at run-time.

Across-level models allow for a more nuanced understanding of complex systems, as they enable researchers to bridge the gap between micro-level interactions and macro-level outcomes. By capturing the heterogeneity of agents and their responses to their environment, across-level models can shed light on the mechanisms that drive system-level behavior, facilitating the identification of key feedback loops and dependencies within the system.

Additionally, the models enable researchers to investigate the impact of various factors at both the individual and system levels, such as how changes in individual behaviors or environmental conditions may affect the overall system dynamics. This approach allows for a more thorough exploration of the robustness and adaptability of the system, providing valuable insights for policy development and system management.

2.6.3 Advantages and applications

ABMs can address complex, multilevel problems that are too difficult to tackle with traditional models. Predator-prey dynamics serve as a classic example of a system traditionally modeled using differential equations and advanced calculus. However, these systems can also benefit from being modeled by an ABM[34]. By employing ABMs to study predator-prey interactions, researchers can gain deeper insights into the adaptive behaviors and decision-making processes of individual organisms. ABMs allow for the representation of heterogeneous agents and the examination of emergent properties arising from their interactions, which can be particularly valuable in understanding the complexities of real-world systems. Applying ABMs can bridge the gap between theoretical and empirical research, highlighting gaps in our knowledge of individual behaviors, and contribute to refining existing theories.

Although the method appears straightforward to apply, researchers argue that this can create a false impression that the underlying concepts are just as simple to grasp. While ABM might seem technically uncomplicated, it possesses considerable conceptual depth, which frequently results in incorrect utilization.

2.7 Design Patterns

In software engineering, design patterns are common solutions for recurring problems encountered when building complex software. A design pattern can be described as a tried and tested blueprint based on well-known object-oriented principles, such as the SOLID[1] principles that can be applied in many different contexts to solve various problems.

Christopher Alexander initially introduced the idea of patterns in his book, "A Pattern Language: Towns, Buildings, Construction"[35]. In this work, he presented a "vocabulary" for designing urban landscapes. The building blocks of this vocabulary consist of patterns that address various aspects of urban design, such as the height of windows, the number of floors in a structure, the size of green spaces within a community, and other similar elements.

This concept was later adapted by the "Gang of Four" (Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides) and translated to the domain of software engineering in their seminal book "Design Patterns: Elements of Reusable Object-Oriented Software," which was published in 1994[36]. This book offers a catalog of 23 reusable design patterns for object-oriented programming that are based on industry experience and observations from the authors.

Today, many design patterns have been integrated into programming languages themselves and are therefore taken for granted by users. For example, the Visitor pattern used in Figure 2.4, is a behavioral design pattern that allows for separation of the algorithm from the object structure it is supposed to operate on. One concrete realization of the Visitor pattern's integration into a modern programming framework can be found in the ubiquitous for-each loop exemplified in Figure 2.5. The for-each loop allows for iteration over a collection of elements without the need for an explicit counter index, effectively separating the algorithm responsible for the iteration from the underlying data structure.

2.7.1 The Observer Pattern

The Observer pattern is one of the first design patterns introduced in the aforementioned book by the "Gang of Four". It is a behavioral pattern that addresses several different key problems in object-oriented programming[37]. It is implemented by creating two separate interfaces: the publisher who is responsible for publishing events

[1]SOLID is an acronym that represents five important design principles for object-oriented programming. These are: Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, and Dependency Inversion Principle. The purpose of these principles is to improve maintainability, readability, and extensibility.

```
static void DoubleAndLog(int number)
{
    Console.WriteLine(number * 2);
}

List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };
numbers.ForEach(DoubleAndLog);
```

Figure 2.4: For-each loop implementing Visitor pattern

```
List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };

for (int i = 0; i < numbers.Count; i++)
{
    Console.WriteLine(numbers[i] * 2);
}
```

Figure 2.5: A generic for-loop

of interest for the rest of the system, and subscribers who are interested in knowing when the publisher has published such an event. Implementing these interfaces allows the system to achieve loose coupling, as the publisher and subscribers can evolve independently, promoting a maintainable and adaptable system. Developers can easily introduce new subscribers with minimal modifications. Furthermore, the pattern facilitates dynamic relationships between scripts that can change at runtime by having the subscribers unsubscribe from the publisher. This, accompanied by the publisher's event broadcasting ability, ensures that the entire system remains in a consistent and traceable state.

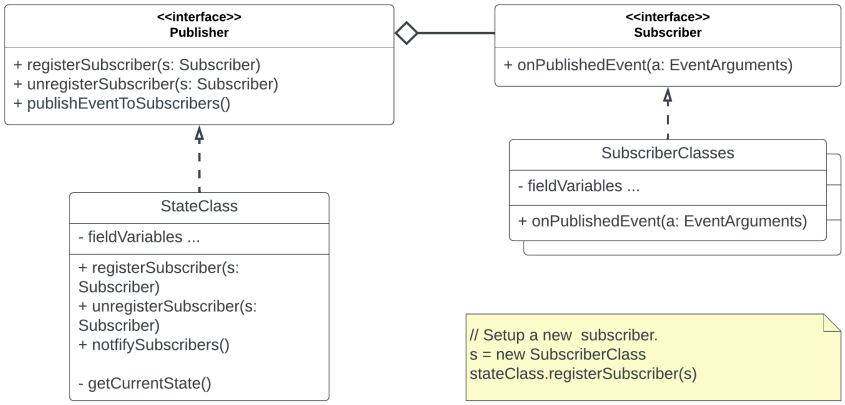

Figure 2.6: UML diagram of generic publisher/subscriber-relation implementation.

2.7.2 Overview of the Observer Pattern in Unity and C#

Both the C# programming language and the Unity API includes several features that facilitate the use of the Observer pattern. Most modern languages opt for a method-based implementation of the Observer pattern, as compared to the class-based implementation as seen in Figure 2.6. The following list highlights some of the most prominent components that are used to promote the usage of this pattern:

- **Delegate (C#)**: Defines a type representing a specific user-defined method signature, allowing for a method-based event system[38]. This custom type can be realized with any method sharing the same signature as the delegate, enabling a type-safe way of passing methods as arguments. They resemble C++ function pointers but provide object-oriented capabilities by encompassing both a function and its associated object instance.

- **Event Handlers (C#)**: Methods defined in the subscriber class that conform to a specific delegate signature[39], typically a delegate with two parameters: one object representing the publisher and one event data object containing event-specific information. These event handlers are responsible for processing incoming events and performing any necessary actions when the triggering event is published.

- **Events (C#)**: A convenient feature in C# built upon the foundation of delegates[40]. They provide an easy way to define, subscribe, and publish events in C#. Publishers define the event, while subscribers can subscribe or unsubscribe using the event handler. Events enforce encapsulation by allowing only the class that owns them to publish them while still enabling other classes to subscribe or unsubscribe at runtime.

- **UnityEvents (Unity)**: A built-in event class offering a flexible and powerful way to facilitate event-driven systems that can be configured in a user-friendly way through the Unity editor[41]. Being serializable, they can easily be set up and managed through the editor using a drag-and-drop approach within the editor.

2.7.3 Singleton

The Singleton pattern is a creational pattern, ensuring that only one instance of a specific class can be created at any given time, providing global access points to that instance[42]. The pattern has garnered criticism for violating core object-oriented principles, such as The Single Responsibility Principle[2], promoting tight coupling, and making testing more difficult due to challenges in isolating tests, replacing instances with mocks, and managing shared global state. However, some argue for its responsible usage, applied only to classes that genuinely require a single instance and where global access is necessary. It is crucial to manage dependencies and shared states carefully to minimize the risk of creating hard-to-maintain, tightly-coupled

[2]The 'S' in SOLID. States that a class should only have one responsibility, promoting good separation of concern and modularity.

code. Common use cases for the Singleton pattern in game development include managing access to different manager classes, such as managers for input, audio, or pooling.

The pattern is implemented by having a private static field in the singleton class for storing the instance of the class. This instance is instantiated through a public static creation method, which uses "lazy initialization" to create a new singleton object instance through a private constructor if it is the first time the instance is being called, or returns the pre-existing instance otherwise. To ensure thread-safety in multi-threaded applications, a locking mechanism can be implemented to prevent multiple threads from creating separate instances simultaneously. This can be achieved using the "double-checked locking" pattern, where the lock is only acquired if the instance is null, reducing the performance overhead of locking in cases when the instance is already created.

2.7.4 State

The State pattern is a behavioral pattern that creates a modular and extensible system architecture for managing transition between different object states[43]. The pattern does so by decoupling the logic for each possible state into a separate interface. A main class then manages these states by offering methods for interacting with different state objects and delegating any necessary command to the current state when instructed. This main class can be described as a mediator between different states, and offers the developers a user-friendly way of managing state actions and transitions.

The pattern was first introduced by the aforementioned book "Design Patterns: Elements of Reusable Object-Oriented Software", and draws its inspiration from the concept of finite state machines, FSMs, which are computational models used across a wide spectrum of domains such as control systems and artificial intelligence. An FSM consists of a finite amount of states, the initial state, and the adhering transitions between them. While both the FSM and State pattern deal with managing system behavior through various states and transitions, FSMs are a more general concept that focuses on the overall structure of a system. Compared to the State pattern which is a specific object-oriented concept focusing on adhering to good object-oriented principles.

By separating the state logic into different interfaces, this makes it easy to accommodate for change and extensibility when modifying or adding a new state. This adheres to the Open/Closed principle[3] as it allows developers to introduce new states without altering the existing state classes or the main class responsible for managing state transitions.

The pattern is implemented by defining the common State interface that should be able to handle any state specific requests and transitions. This interface is then

[3]The 'O' in SOLID. Says that a class should be easy to extend without needing to modify any existing code.

realized by concrete state classes that provide their own unique logic and behaviour. To mediate between these states, you then implement a main class, sometimes called the Context class, that holds a reference to the current state and delegates function calls to it. This main class is also responsible for changing the current state based on transition logic defined in the concrete State classes. A UML diagram of a State pattern is shown in figure 2.7.

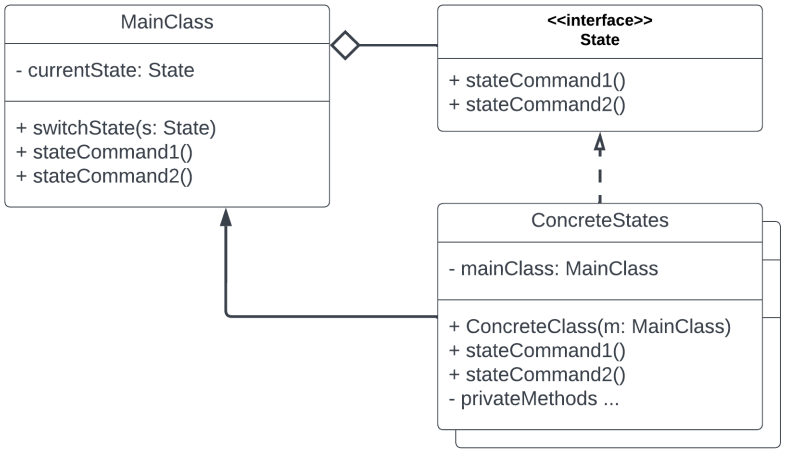

Figure 2.7: UML diagram of a generic State pattern implementation.

2.8 OpenStreetMap

OpenStreetMap, or OSM for short, is a free and open-source database which contains geographical data [44]. The database is maintained and updated by volunteers. Volunteers can collect data about geographical areas and add to the database for everyone to use. The data includes features such as roads, railroads, buildings and trees as well as their properties. OSM can be used to generate environments based on existing locations.

3

Methods

This section will present the tools and methods used throughout the project, and how certain aspects were implemented. Firstly we explain our thought processes on choosing any existing tools, and then present any tools that were created in order to aid in the development. Lastly we present the workflow that was used, from the planning to the organization and testing methodology.

3.1 Tools

In the following section we will review the software tools that significantly influenced our development process, and the motivation behind choosing these tools. Central to our discussion is the choice of Unity as our primary game engine, over other established alternatives such as Unreal Engine. We will delve into the key factors that influenced this decision, encompassing the team's familiarity with Unity's primary programming language, C#, the advantageous size of the Unity Asset Store, and the engine's flexibility. Finally, we will provide a brief overview of how we utilized GitHub for efficient version control.

3.1.1 Unity

The traffic simulation tool is built in the Unity game engine. There are a few reasons why it was chosen as the development platform for the project, instead of a similar game engine like Unreal Engine. To begin with, C# is the main programming language supported by Unity, which most of the team members had previous experience with. Furthermore, C# is an inherently simpler language compared to C++, the main language of Unreal Engine, making it easier for the team members without experience to learn. Therefore, the development of the tool was started in a shorter time frame than what would have been expected had Unreal Engine been chosen as the platform.

Another reason is Unity's larger user base, resulting in more resources such as tutorials and useful development tools. They also have a more comprehensive asset store compared to Unreal Engine, making finding models and other helpful assets likelier and easier. One of the most notable assets used is Edy's Vehicle Physics that allows for realistic vehicle physics[45]. This circumvents the need to develop a

custom vehicle physics engine, instead enabling the team to use the asset to quickly configure the vehicles.

The final reason why Unity was chosen, was due to its flexible developing structure and higher level of freedom when it comes to implementing solutions. Unity revolves around developing with scripts and conventional object oriented programming. In comparison, Unreal Engine has more advanced tools that might yield better results, although end up taking longer to learn. The level of customization available inside the engine is greater in comparison to Unreal Engine. However, because of this, Unity ends up being more unstable whereas Unreal is far more stable and robust.

3.1.2 Third-Party Assets

Built into Unity is their asset store. Instead of creating everything from scratch, the team opted to utilise some assets. An asset can be anything from a 3D model to animations and scripts. Multiple assets were taken advantage of throughout the development period of the project. These assets include impactful ones such as Edy's Vehicle Physics and the Bézier Path Creator[46]. Edy's Vehicle Physics is a package that includes a tool that allows its user to easily implement realistic vehicle physics into 3D models, while the Bézier Path Creator is a light-weight editor for creating Bézier paths in the editor. This saves a substantial amount of time since they eliminate the need to design custom tools for these tasks.

3.1.3 GitHub

A commonly used tool when developing software in larger groups is Git[47]. Git is a free and open-source version control system that allows its users to collaborate in an easy and efficient way.

GitHub[48] is an online software development platform that utilizes Git to store and track software projects. It allows for users to work in their own separate branches, and later merge those into the main repository. This feature was used in the project. Prior to merging new code to the main repository, the code was required to be reviewed by at least one other team member to ensure that the code was well commented, functional, and that it followed the C# coding standard.

3.2 Simulation Design and Implementation

This section provides an overview of the development and functionality of the various software components we developed to create a traffic simulation. We delve into the use of composite Bézier curves for realistic road generation, the implementation of points of interest, and how our agents interact with these. In addition, we explain how we employed nodes for vehicles' navigation. Furthermore, we outline how we integrated the previously mentioned A* algorithm for optimized pathfinding, and how we utilized OpenStreetMap data to generate authentic cityscapes for the simulation

environment.

3.2.1 Road Generation

Composite Bézier curves were used in order to create realistic roads with smooth curves. This can be seen in figure 3.1. A number of parameters can be changed in the Bézier path to modify the appearance. The Bézier control points will shape the road and its characteristics. The position and sharpness of a turn can be modified by changing the location of the control points.

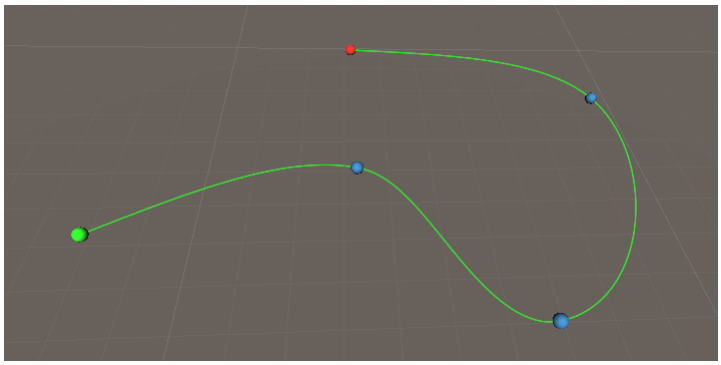

Figure 3.1: Composite Bézier path

While the Bézier curves provides a good base for the road implementation, it is hard to build and implement logic for it since it is a continuous path. The Bézier logic and its control points were abstracted away with a node implementation in addition to the Bézier path. A number of nodes called RoadNodes are placed along the Bézier path at a rate dependent on the curvature of the road. The nodes are all connected to their previous and next nodes along the path. The goal of these nodes is to carry enough information to procedurally build the road mesh, and provide the logic needed for the agents to navigate the road network.

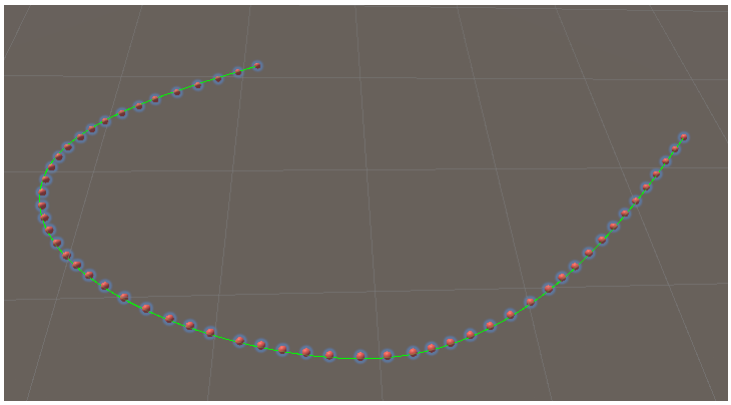

Figure 3.2: Visual representation of RoadNodes places along a Composite Bézier path

By using the RoadNodes, visualised in figure 3.2, generating the road mesh is

possible. The mesh is procedurally generated by placing vertices offset from the RoadNode by the width of a lane. If multiple lanes should be generated, additional vertices are added in each normal direction. Triangles are then drawn between the vertices to create the mesh. To add the road material, the lanes are split into sub meshes, making it easy to generate the road lines. The power of procedural generation is the ability to customize the different road parameters. These include the number of lanes, width of each lane as well as the thickness of the road.

While the RoadNodes carry critical logic, it lacks logic for driving along the lanes. This logic is added through a different node type, called LaneNodes, placed along the lanes. LaneNodes are positioned at both sides of each RoadNode, in the center of each lane. These nodes are used by the vehicles as steering targets, as well as to notify other vehicles if they are currently occupied.

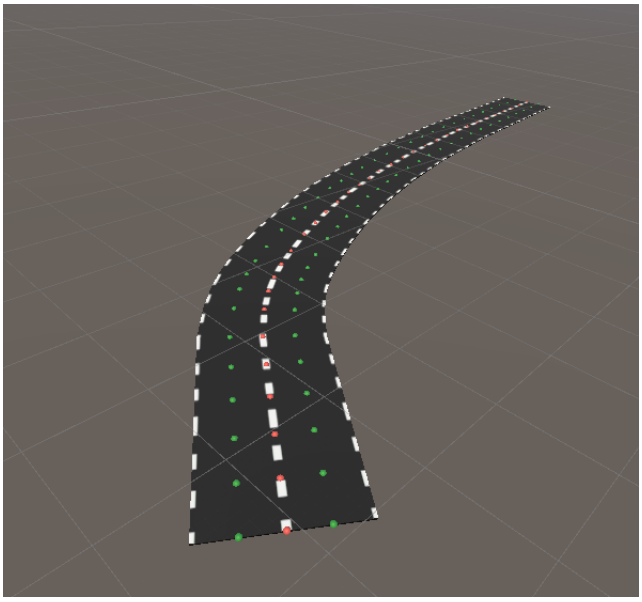

Figure 3.3: Visual representation of RoadNodes (Red) and LaneNodes (Green)

When any of the control points of a road are moved, the new Beziér path is scanned for intersections with any other road path, using the Bézier clipping algorithm described in 2.2.3. At each intersecting point, a junction is then generated. In figure 3.4, a generated intersection is presented. The two Bézier curves of the roads can be seen crossing each other in the center of the intersection.

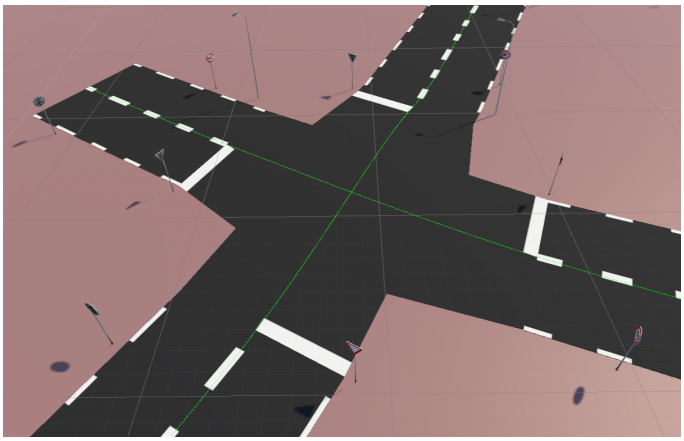

Figure 3.4: An intersection generated at the intersecting point between two roads. Their Bézier paths are visible in green

3.2.2 Points of Interest

Points of Interest, or POIs for short, define locations of importance. In the simulation, POIs add places that the vehicles can navigate to and interact with. The POIs supported by the tool are parkings, bus stops, fuel stations and houses. These add another layer to the simulation. Bus stops can be placed around the road system, and later added to bus routes for buses to follow, stopping at every bus stop along the way. A bus following its route and stopping at a bus stop can be seen in figure 3.5. This is something to consider when designing road networks, as well as choosing bus routes in the real world, since they have to follow the same routes which makes buses vulnerable to congestions. To mitigate this, bus lanes, or even dynamic bus lanes, can be added to prioritise public transport [49]. By simulating buses using the bus stop POIs, it is possible to identify how well the public transport performs given the circumstances in the simulated road network. It is also possible to compare how changes to the road network affects the public transportation system.

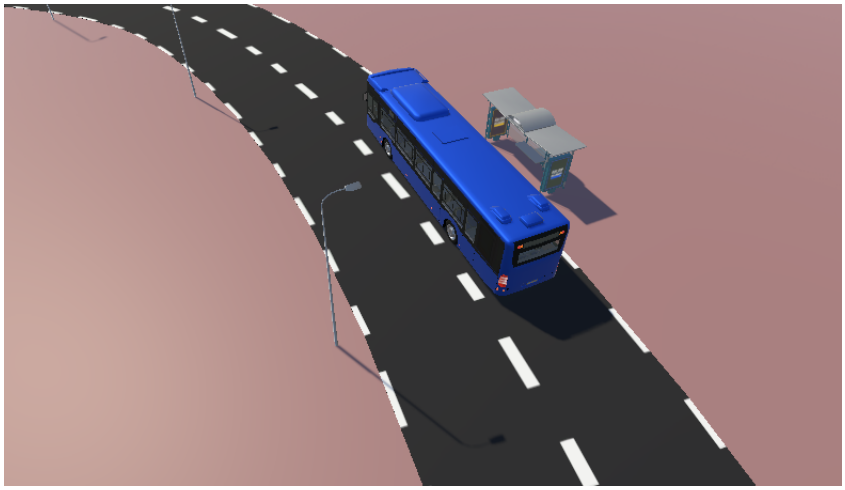

Figure 3.5: Bus stopping at a bus stop

When the vehicles are almost out of petrol, they will navigate to a fuel station. With limited access to fuel stations, queues might arise when multiple vehicles need to refuel at the same time. Houses are targets for the vehicles to simulate driving to or from a home, meaning vehicles might gravitate towards residential areas during certain times of the day which could also cause congestion depending on how well the road system can handle it.

3.2.3 ABM

The different vehicles are simulated as independent agents. These agents perform calculations based on their environment and choose actions based on them. The environment is the road network with the roads, traffic lights and intersections. Agents will check for traffic rules, other agents, traffic lights and intersections and decide were to steer and whether to accelerate or brake, this interaction can be seen in figure 3.6.

Figure 3.6: Agents interacting with the environment

3.2.4 Vehicle Driving Implementation

The vehicles need to be able to navigate the road systems, and so a script had to be written to control them and allow them to follow roads. The vehicles rely on the previously mentioned generated LaneNodes to follow the roads as they curve. The LaneNodes contain information to allow for passive vehicle communication. This is to improve performance, as vehicles otherwise continuously would have to look for other vehicles nearby, searching in a 3D space each frame which is costly. Instead, the LaneNodes allow the vehicles to store some information that can then be read by the other vehicles. Each frame, every vehicle assigns itself to all the LaneNodes it is currently sitting above, thereby letting other vehicles know which LaneNodes are occupied. This is then utilised for making the vehicles brake for occupied nodes, thereby preventing vehicles from crashing into each other. As a consequence, this

also implements behaviours where vehicles queue up behind each other once the vehicle in front has stopped.

With a few simple additions to the logic, such as the LaneNodes also containing information about any traffic lights or traffic signs placed at that location. This allows the vehicles to stop for red lights or stop signs. The fairly simple logical implementation allows for non-colliding traffic in isolated roads, however, collisions can still occur in intersections. This is a complex problem with several solutions.

For this implementation, two new concepts have to be introduced. Both are related to yielding. The first is yielding for blocking nodes, and the second is yielding for crossing nodes. Yielding for blocking nodes means that the vehicles will yield, i.e. stop and wait, for any vehicles currently on any nodes in the way of the vehicle's path. This means the vehicles will avoid any traffic inside the intersections. However, as vehicles are travelling towards each other in the intersection, this is not quick enough as they will notice each others' occupation too late. This leads us into the second concept, which is yielding for crossing nodes. Upon entering an intersection, each vehicle will also check the nodes of any crossing paths, looking as far back as required to ensure that no other vehicles will arrive at the intersecting point before the vehicle itself. This means that vehicles will stop for other vehicles heading into the intersection, on any crossing paths that are close enough to interfere.

Both yielding for blocking as well as yielding for crossing nodes is dependent on the path the vehicle is trying to take, and therefore have to be calculated with the context in mind. For example, if the vehicle is travelling straight across an intersection with traffic lights it does not need to yield for any crossing paths, as all crossing paths will be those required to yield. The same goes for the blocking nodes, different paths are in the way and can block depending on the path the vehicle is heading for. To improve performance, all possible paths in the intersections are precalculated. During that time, all blocking and crossing paths are also precalculated for each path. This means that once the vehicle tries to follow its path in the intersection, it will already have the information related to which nodes it needs to check in order to navigate the intersection safely.

As the vehicles are individual agents and programmed through an object oriented approach, implementing these rules for every vehicle means that traffic flows will arise and a complex behaviour can be simulated through these individual rules. A popular example of this is the flocking behaviour of birds which can be simulated through only three simple rules; separation, alignment and cohesion, creating a complex behaviour[50].

3.2.5 Navigation

In addition to being able to follow the roads and handle intersections, the vehicles need to be able to navigate the road system. The road system can be thought of as a graph, with the roads as edges and each intersection and POI as a node, as seen in figure 3.7. After generating this graph from the road system, it is then possible to find the shortest paths between two points in the road system. In this project an

implementation of the A* algorithm is used for this purpose.

Figure 3.7: Visual representation of a navigation graph

The graph representation of a road system is a weighted directed graph. The edges, which correspond to the roads, are weighted with a cost that is calculated as the distance multiplied by the speed limit. This is an estimate of the time it would take to travel that edge, which lets the algorithm find the fastest path. The graph is directed since one way roads are supported. This limits the possible paths the agents can take. The agents navigate to a given end node by receiving a path of edges from the A* algorithm. When the agent reaches its target, it will receive a new path and subsequently navigate to it.

Multi-target navigation is also possible. By receiving a number of destinations, the agent will travel to them in the given order. This is achieved by using the A* algorithm to find the shortest path to the first target, then it is repeatedly calculating the shortest path from the last target to the next. This is for example used by the buses to map out their entire bus route.

3.2.6 OpenStreetMap integration

To aid in simulating cities and larger areas, existing real life locations are generated from OSM data. An OSM file for a specified area can be downloaded and then used as an input for the simulation. The data in the file specifies the latitude and longitude of every road, and its characteristics such as the path, speed limit, road name and the amount of lanes. Building and nature data is also available. The simulation tool will then parse the file and generate a to-scale replica of the road network in the given location with its characteristics. To add to the level of

realism, the buildings and nature are also generated. This feature allows testing of existing locations without having to build a replica by hand. A generated map of Masthugget, a district of Gothenburg, can be seen in Figure 3.8.

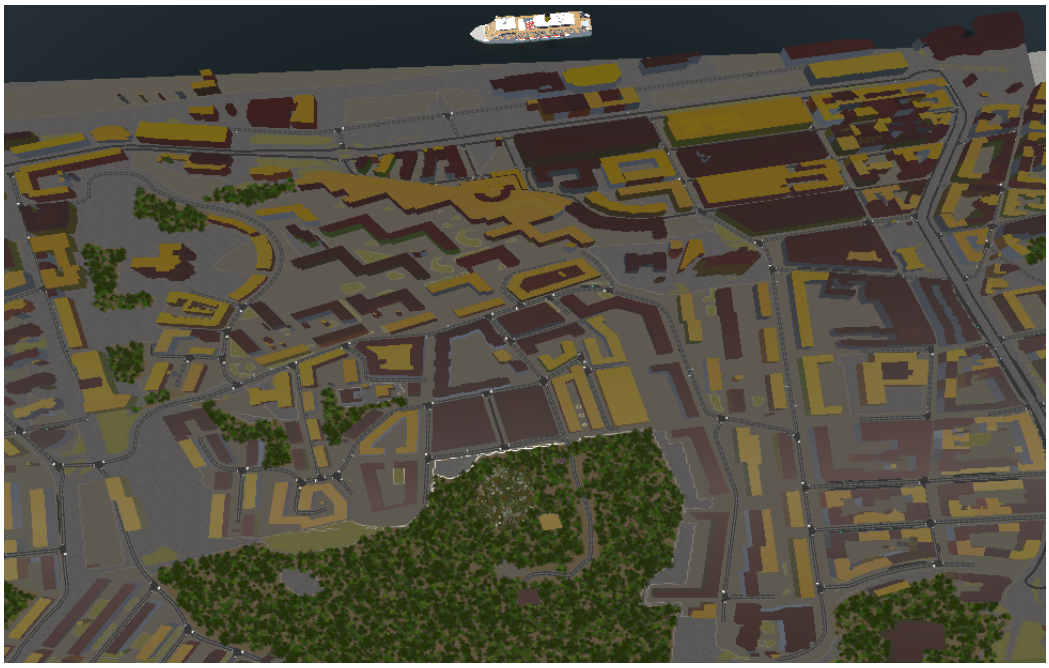

Figure 3.8: Masthugget, Gothenburg, generated from an OSM file

Furthermore, the POI system is integrated with the OSM tool. The parking lots, bus stops, fuel stations and houses that exist in the given file are simulated as POIs. This allows the agents to interact with their environment, as outlined in section 3.2.2.

3.3 Performance

This section addresses design choices made to optimize the performance of the simulation. Given the goal for the simulation to host a large number of simultaneous agents, this was a critical consideration. We highlight optimization strategies employed to handle performance intensive aspects of the simulation, such as concurrent active vehicles and the roads themselves. Performance benchmarks are then presented to give the reader a comparison between different settings.

3.3.1 Quality vs Performance

An important aspect of software is how well it runs. Therefore, it was decided early on that it should be possible to change the simulation quality.

While testing the simulation during development, the most noticeable performance cost were the vehicles. This is because of Edy's Vehicle Physics, an asset that simulates real-world physics for each vehicle in the network. To circumvent this

issue, a vehicle performance mode was implemented. This performance mode would disable the EVP asset, and instead move the cars by offsetting their individual object transform. As a result, the performance cost of the vehicles would decrease, and allow for more traffic in the road network.

3.3.2 Optimization

When creating software of any kind, it is important to make sure it is able to run smoothly. This was achieved through the use of optimizations. There are two areas in the simulation that are costly performance-wise: the vehicles on the roads and the roads themselves. To optimise the vehicles, as mentioned earlier, a performance mode was implemented. This allowed the simulation to halt calculating the physics for each vehicle.

Furthermore, since the simulation is in 3D, the details of the vehicle models had to be accounted for. A 3D model is created with vertices and triangles as described in section 2.4. The amount of triangles in a model determines its resolution. To improve the performance of the simulation, the models that were chosen had a low triangle count.

3.3.3 Performance Benchmarks

A series of performance benchmarks were completed to assess the performance of the simulation when different amount of agents are navigating the road system. A script was written to sample the average FPS of the simulation. The script benchmarks the simulation by moving a camera to a number of different positions, calculating the current FPS at each one. The resulting FPS' are then averaged to determine the overall performance of the simulation. This is repeated for different agent counts and quality modes.

3.4 User Interface

This section will delve into how we implemented our user interface, which serves as the primary point of interaction between the user and the simulation. We focus on its two main elements: the start menu and the runtime overlay. Furthermore, we provide an overview of our approach to presenting statistics gathered from the simulation during runtime.

3.4.1 Design

To allow the user to interact with the simulation, a user interface was made. The user interface, or UI, consists of two main parts: the start menu and the runtime overlay.

The start menu contains three buttons, used to start the simulation, enter the settings menu and exit the program. The start menu is shown in figure 3.9. The

settings menu allows the user to change the volume and simulation quality as well as enable an FPS counter or enter full screen. The settings menu is shown in figure 3.10. In addition, the start menu also contains a field for setting the number of simulated vehicles.

Figure 3.9: Start menu presented upon launch of software

Figure 3.10: Settings menu

The overlay that is visible while the simulation is running allows the user to interact with the simulation itself, see figure 3.11. There are two main types of buttons: the camera buttons and the UI menu buttons. As the name suggests, the camera buttons are used to control the user's point of view. The default point of view in the simulation is isometric. An isometric point of view is an angled top-down view that is commonly used in video games to produce a 3D-like effect. The other two

buttons allow the user to enter first person view in the driver's seat of a vehicle, and a camera that follows the selected vehicle. The menu buttons are used to access the main menu or display simulation statistics.

Figure 3.11: Overlay UI

3.4.2 Statistics

The statistics panel displays simulation statistics for an individual vehicle if one is currently selected. If no vehicle is selected, it displays the aggregated statistics for the entire simulation. The panel is a movable pop-up window that can be displayed or hidden using the statistics menu button.

The individual vehicle statistics are used to give the user an easy overview of how a specific car is performing in traffic, see figure 3.12. It shows statistics such as fuel consumption, average speed and distance traveled. This can be useful to follow specific vehicles and investigate their behaviour in the road network. This is important to take into consideration because changes in the road system can lead to certain vehicles being negatively impacted. This allows the user to find these vehicles and investigate the issues.

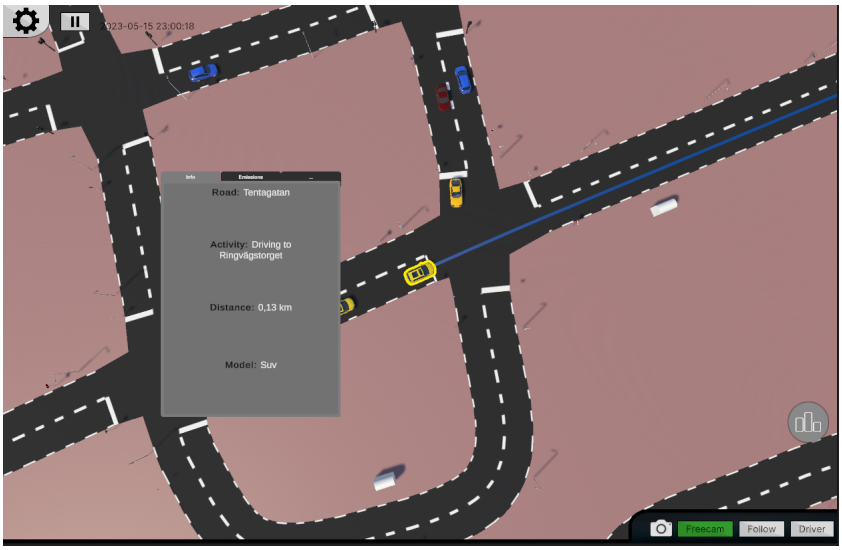

Figure 3.12: The statistics window on the info tab

The world simulation statistics window gives the user an overview of the road networks overall efficiency, see figure. It indicates which areas and roads that are the busiest and therefore moves at the slowest pace. This is useful when locating the causes of traffic jams, and might give insight into possible solutions. Moreover the window is also useful to see how traffic would differ when changing the amount of public transport. Not only will this quickly show the differences in traffic flow but also the CO2 emission amount and fuel consumption over time, see 3.13.

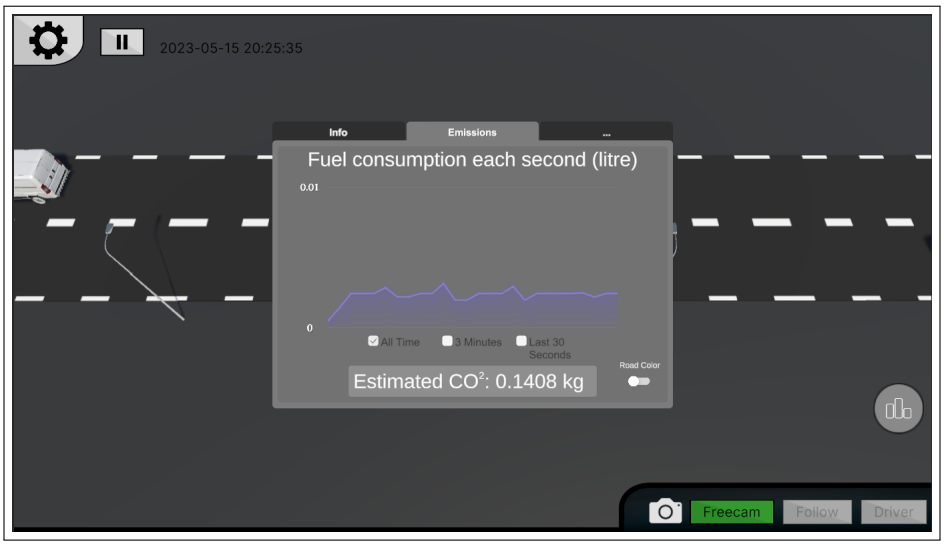

Figure 3.13: Graph of fuel consumption

3.5 Workflow

When developing larger softwares, the amount of work and information can quickly grow beyond the level of ones own comprehension. Therefore, these kinds of projects

require rigorous planning and strategizing for them to function smoothly.

To achieve this, a strict workflow framework was developed, where the first step was to analyze the work load and disposable time. This included drafting a time plan for the entire scope of the project, see figure 3.14.

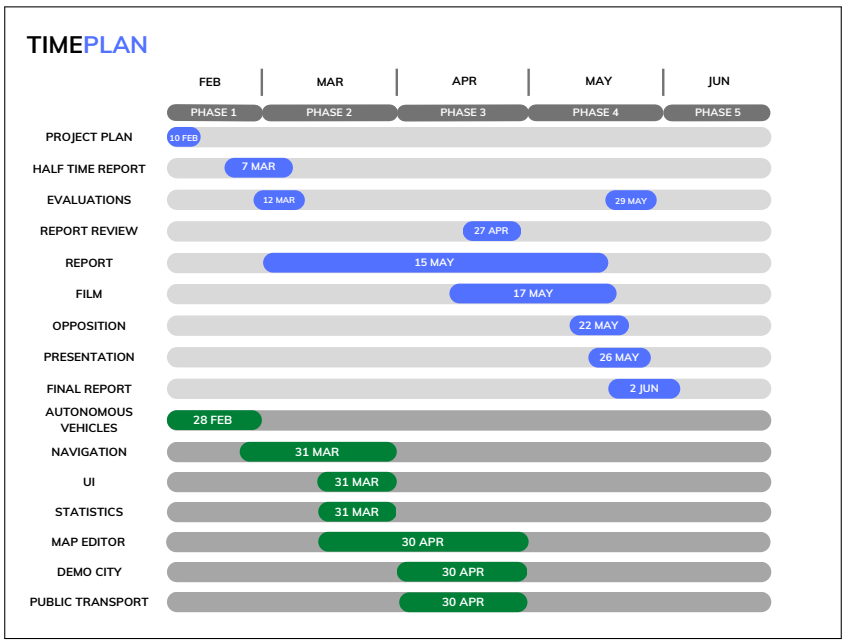

Figure 3.14: Project Time Plan

With this system in place, it is easier to keep track of the general progress of the project and plan short term goals. Coincidentally, this is the next step in the workflow model. The short term goals where planned using a scrum framework with weekly sprints, explained later in 3.5.1. These sprints were upheld for the duration of the project to keep a steady flow of progress. Together with the time plan, they create a clear way of visualizing the current state of the project.

The third aspect of the workflow is the approval of progress. As mentioned earlier, a large project requires a substantial amount of planning. The approval of progress can be just as important as the planning and execution itself. Without a proper method of approving new advancements or functionality, the project can quickly falter. If progress never goes through the process of approval, it can lead to problems later on. Evidently, inflexible code can cause issues that are easily preventable through quick inspections. Code can even be considered good, but with no input from the rest of the team, visions of how higher order elements will be implemented can differ. This can implicitly create more complex problems further on, which can be difficult and time consuming to resolve. To solve this, code reviews were part of the workflow, as explained in section 3.5.3.

3.5.1 Weekly Sprints

The weekly sprint model stems from the scrum framework, which is a framework for developing and sustaining complex products. The sprint model follows 4 repeating stages of development: Planning, Implementation, Review and Retrospect. Each sprint starts out in the planning stage, where a meeting is held to determine the sprint goals. This includes moving or creating stories for the backlog, and the current sprint. The stories are mainly chosen by the project manager, then developed in unison with the scrum master, with continuous input from the rest of the team. The next stage of the sprint is the implementation itself. This is the time where the team focuses solely on delivering completing the tasks for the sprint, and if time allows, work on the backlog.

Next up is the review stage, not to be confused with code reviewing. In this stage another meeting is held, called a "Demo meeting", where all members get to do a small demonstration of their results from the previous sprint. This is an important step to make sure all members understand the new functionality. When a story is regarded as fully complete, it is archived.

Lastly, during the retrospect stage, the team reflect on the previous sprint and offer any improvements for the next iteration. After the final stage, a new sprint is commenced.

3.5.2 Trello

Early on in the project, it was decided that the workflow should follow scrum and agile software development practices. When doing so, having a scrum-board is essential for implementing the methodologies that accompany these practices. A scrum-board is a visual tool that helps the team keep track of tasks that need to be worked on during the weekly sprints. Each task on the board, which is called a "story", is placed in a column representing the different stages of development depending on where the progress of the task is currently at. Trello is a website that can host scrum-boards in a user-friendly way, which the development team made use of to set up a custom scrum-board template according to their needs[51].

3.5.3 Code Reviewing

An important part of any larger code base that is being developed and maintained by several developers, is reviewing the code. This serves several purposes, one of which is making sure any new code follows the existing coding standard. However, one of the most useful purposes is that other developers who review the code, could identify potential issues or bugs. This also allows for feedback or solutions to be provided.

The repository is set up so that the main branch is protected, meaning no new code can be written within the main branch. Instead all code has to be written within separate branches, which are then merged to the main branch. Before any code is merged, it has to be reviewed and approved. This makes sure all code has been

viewed by at least two pairs of eyes, increasing the chances of spotting bugs or badly implemented code.

3.6 Testing

In order to impartially evaluate the tool and determine if it achieves the purpose, several user testing sessions were held. In these sessions, the users were given a brief explanation of the tool, and then asked to perform a task without any guidance. By analysing the user while trying to perform the tasks, it is possible to follow their intuitions to validate whether the interface is easy to understand and intuitive.

By asking questions and discussing with the testers during the sessions, we are also able to understand what the users like and dislike, as well as what improvements could be made. By integrating the testing into the development process, we were able to improve the tool and iterate the design before it was finished, thus creating a better result. This created a feedback loop, where we could gather information about what we needed to work on, and improve it according to the feedback before the next testing session. During the next session, we were able to validate whether the changes improved the experience or not. In addition to design related feedback to make the tool intuitive, we also had testing sessions with users experienced with existing transportation planning software in order to assess the features of the tool. This gave insight into what needed to be added, and what could be omitted, as well as what the advantages and disadvantages the tool had compared to existing software.

4

Results

This section will give an account for different results gathered from both our experience with the product, performance benchmarks, and feedback that were collected during user testing.

4.1 Final product

The end product is a simulation tool that is capable of simulating vehicles driving in a road network. The road network can either be generated through an imported OSM file, or custom built with the use of the built in Unity Editor. The road network has support for both three and four way intersections. These intersections can either be controlled by traffic lights or stop signs depending on the user's needs. The vehicles themselves are all capable of individually navigating a correctly built road network. The vehicles interacts with their surroundings using the RoadNodes and LaneNodes built into the roads. From these nodes, the vehicle can collect data about whether a node ahead of them is occupied as well as the current speed limit of the road, and if the vehicle needs to brake.

To use the simulation tool, the users can interact with the simulation through the provided UI. It consists of a main menu, with an included settings page. In addition, there is an overlay menu, that is visible while the simulation is running. With this overlay menu the user can access statistics about an individual vehicle or the entire road network. The statistics are represented through a series of data points and graphs to easily display different statistical aspects of the simulated road network. The user can also change the number of simulated vehicles.

By using the Unity Editor, the user can as mentioned customize the simulation world. The user can also create roads, add intersections, and add POIs. Furthermore, it is possible to add buses that travel along the bus stops. Parking lots can also be added. These parking lots can either be attached to a road as a street side parking, or as a separate parking lot.

4.2 Performance

To test the performance of the tool, an automated testing utility described in section 3.3.3 was used. The tests were run on a simulation of the road system for Masthugget, a district of Gothenburg. The tests were run both in the quality mode as well as the performance mode. The test results are compiled in Table 4.1.

Table 4.1: Performance test results

Configuration	Result	
Number of vehicles	Quality FPS	Performance FPS
5	192	205
50	191	203
100	189	203
250	71	102
500	30	48
1000	10	25

4.3 User tests

User testing in software development is an important aspect of a successful end product. It gives the developers insights into how their software is actually perceived in a deployed environment, and how their intended audience interacts with it. This section presents an account of our performed user testing and the feedback received. The testing was conducted with two distinct groups: "experienced testers", and "generic testers".

4.3.1 Experienced Tester

The first test subject had prior experience in a transportation planning software called PTV Visum, which is the world's most popular software of its kind when it comes to aiding strategic and operative decisions[52]. Compared to Visum which is set in 2D, the user felt that our tool provided more context, and that it was easier to understand the road network. The test subject pointed out that it was harder to see the road connections and intersections in PTV Visum. However, Visum offered a better view for larger networks. The user appreciated the simulation of individual vehicles compared to Visum, which displays traffic as a value of the number of vehicles per road, and felt that it improved the intuition for smaller networks. Being able to see statistics related to individual vehicles was also pointed out as useful. The test subject felt our tool lacked some customisability that Visum offers, where you can change parameters such as road capacity that affect the simulation. The user mentioned that Visum had a learning curve to understand the buttons and features, which was easier to do in our tool although it does not offer as granular control over the traffic as Visum does. The test subject was not able to use the public transportation feature as it was not done at the time of testing, but expressed interest in it and thought that it was a great idea that Visum lacked. Colour coding the

roads based on the congestion level was being implemented at the time of testing, and was something the subject mentioned would also be helpful.

The tester felt that our tool was easy to understand, and significantly simpler to use for presentation purposes than Visum, which was said to be more technical and harder to understand at first glance. The user had used Visum to demonstrate their solution, and said that it was difficult to find a good way of visualising and presenting it.

Apart from the functionality compared to Visum, the test subject also had feedback regarding the UI. The user thought that it was confusing that there were separate buttons for the statistics, depending on if it was showing individual statistics for a vehicle or aggregated statistics for the entire road network. It was also expressed that it would be easier to navigate using the mouse for adjusting the camera rotation while keeping the keyboard for moving the camera around, and that it felt slow to move around a larger network.

4.3.2 Generic Tester

The project also conducted user testing on a group we have classified as "generic testers". These individuals possessed no prior experience with any form of simulation software, and the primary objective during these tests was to discern potential deficiencies in the UX design. Some feedback that was acquired included the lack of alternative control schemes. Every tester tried to move around the simulation using the arrow keys, though the tool's movement keys are WASD. Once identified, the camera movement felt good, but they would have liked a clearer communication of the control scheme somehow. This confusion regarding the control scheme also extended to how the user moved between cameras, and some users felt that it was hard to tell the difference between the follow camera and default camera depending on your initial zoom when transitioning to the default camera.

Furthermore, all generic testers reported a preference for more prominent signaling of the various UI functionalities. This could potentially be achieved through the introduction of informative tooltips, the strategic use of intuitive visual icons, and the implementation of an initial user onboarding walkthrough, that succinctly explains each feature. For example, one tester reported that they felt unsure if they had actually managed to spawn any vehicles when they had pressed the "spawn vehicle"-button, and would have liked some feedback on their input or an explanation of how the spawning functionality actually worked.

5

Discussion

This section will discuss the results of the project. This encompasses examining any unreached goals and presenting the performance evaluation of the tool. Later on the user testing and it's impact will be discussed, as well as the development process it self and any possible future improvements.

5.1 Performance Evaluation

In section 4.2, we displayed a table showcasing some performance statistics of the simulation software. The benchmark summarizes the current expected performance of the simulation. The test was completed using the script, mentioned in section 3.3.3, to determine the FPS with a varying amount of vehicles that are simulated in the Masthugget build. Additionally, the same test was performed in both quality mode, meaning realistic physics modeling for each vehicle, and performance mode, with a much simpler physics engine. The results showcase the overall benefit of running the simulation in performance mode, but also how there is a need to further improve the performance to simulate more expansive road systems.

5.2 Unreached Goals

Some features took more resources and longer time to implement than initially estimated, which limited development in other areas. The most significant example of this is the road creation system. When defining the goals and planning the project, an asset called RoadArchitect had been found that supported this[53]. The asset was well-known and had an extensive documentation. However, when starting the development using this asset, several severe bugs were identified. We think the issue is that it was developed for an earlier release of Unity, and was not compatible with our version even though there was no documentation of this. RoadArchitect supported creation of roads and intersections with many features, such as traffic lights, road markings, railings and bridges. This was a major setback as we had to develop our own road creation tool. Adding all the needed features to the road creation tool was an extensive process that would have been avoided if we could use RoadArchitect as it had nearly all the features needed for this project.

An area which was less prioritised due to the time constraints was the support for

public transport, which was only added with basic support. The goal was to have a working public transportation system, with the ability to change a parameter in the simulation for how much of the traffic is handled through it. This would allow users to find the optimal balance between public transport and cars.

A map editor was supposed to be developed to allow users to create their own maps and road systems to analyse. The time and change in priorities did not allow for the time to add this. This is definitely something that would need to be added in the future if the tool would be released. As of now, it is only possible to edit the maps in the built-in Unity Editor, which is not available when generating an executable version of the tool.

5.3 User Testing Feedback

Receiving feedback on the project was very valuable. However, please note that our user testing pool was limited to only three testers in total. We were also limited to only one iteration of user testing due to time restraints and project scope, which limited our ability to refine and adapt the simulation based on the feedback received. Having additional rounds of user testing, ideally with the same users, would have been highly beneficial. This would allow us to observe how users adapt to the software over time and evaluate any implemented changes made in response to their feedback.

5.3.1 Evaluation of the software

According to the user feedback, we found our tool to be intuitive and easy to use, regardless of the users' prior background. The control scheme, although not clearly communicated in every different state of the simulation, was deemed intuitive and straightforward. Every tester made requests for additional features, but felt that the ones currently implemented were easy to grasp and could be utilized easily by the tester through the minimalistic UI. The generic testers requested more immersive enhancing details, such as rear-view mirrors when sitting inside the car, have the steering-wheel turn in synchronicity with the movement of the vehicle, and add further details to the environment. This feedback would be valuable for future development, but was not prioritized in the scope of this project due to time constraints. The overall impression was that the simulation, as it stands now, has a lot of good features but would need to focus on extending these while maintaining the user-friendly approach to be able to compete with other simulators.

5.3.2 Changes made based on the user testing feedback

In order to eliminate the ambiguity between the two statistics related buttons, it was decided to remove one of them and instead dynamically change the content based on the context. It was changed so that if the statistics panel is open while a vehicle is selected, it displays the individual statistics related to that vehicle. Otherwise, it

displays the overall statistics for the entire road network. This solution was proposed to the tester and the response was that it would be suitable solution.

As some users felt that it was a bit cumbersome to navigate around, the camera controls were changed so that the keyboard is used for movement around the plane, and the mouse to rotate the camera. This change made the tool in line with most games, as well as other software, such as CAD programs. This improves the accessibility and intuitiveness as users can recognise the controls from previous experiences. The movement speed was also changed to depend on the zoom level, so that the camera moves slower when zoomed in on vehicles or intersections, while being faster when positioned at a higher altitude. This allows the user to quickly move between areas of the network, while still being able to have fine control over the movement when zoomed in.

A button was also added to toggle the colour coding feature that can colour the roads based on current statistics, such as congestion levels or emissions. As illustrated in 5.1, this feature is useful for visualising the current performance of the road network and swiftly identifying problematic areas when observing the system from a macro perspective. In response to feedback from the generic testers, we also added an additional control scheme that allows camera movement using the arrow keys.

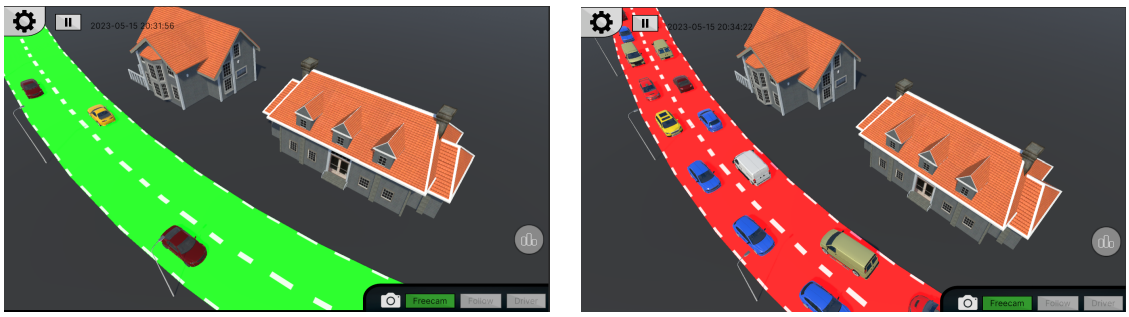

Figure 5.1: Colored road based on current fuel consumption

5.4 Development Process

As previously mentioned in section 3.5.1, the decision was made to follow the scrum framework. This allowed the group to set clear goals for each week, which simplified the process of dividing work. For the most part, this worked as well as expected. However, something that was noticed during the project was that the backlog was slowly increasing with tickets. Each week, a small number of tickets were delayed for different reasons, which was the cause of the problem.

During the development of the simulation, code reviewing was an important factor in making sure the code was correct, commented, and well written. This worked well for the majority of the time, but minor problems arose for larger pull requests. The reason behind this is due to the problem that when someone tries to merge a extensive amount of code at the same time, the time and effort required by the code

reviewer increases. This is because the reviewer first has to try and understand the logic and functionality of the code itself. Therefore, if a code reviewer is tasked with reviewing a large pull request, the reviewer might not have the same attention to detail compared to when reviewing less code, resulting in a worse review.

5.5 Future Improvements

This section will present any future improvements or changes that we would like to implement. We will partly provide motivations as to why these improvements were not made, and if unclear, why we deem them important. Improvements without any explanation as to why they are missing can be presumed to be a result of lack of time.

5.5.1 Road and Intersection Types

When designing large road networks, especially outside cities, highways are critical. Highways are currently not supported in the simulation because of a few different reasons. To begin, there is functionality for multiple lanes when creating roads within the built-in Unity road generator. However, the vehicles are not able to switch lanes in the simulation, making multiple lanes unusable. Since lane switching isn't supported, vehicles do not have the ability to overtake each other. There are also no highway entrances or exits in the simulation, which would need to be implemented for highways to function correctly.

For intersections, the simulation only supports three and four way intersections. These intersections can be created with either one-way or regular two lane roads, or a mix of both. In reality, there are other intersection types such as the roundabout that there currently is no support for. For this to be implemented in the future, the yielding of the cars has to improve from their current state.

Expanding the support for different types of road signs and traffic rules would also be an improvement over other tools that in most cases does not support this. An example of this is how our tool supports the priority to the right rule, which could mean that in certain situations an intersection could be easily congested due to vehicles on one of the roads always having to yield to those on the other. Additional signs such as those defining rules for parkings could also affect the simulation, making certain parking spaces restricted for example during certain times of day or for non-residents. Due to the nature of the tool being low level with simulation of individual vehicles, there is room for improvement with features like these which would separate the tool even more on the market, creating a unique feature set.

5.5.2 OSM

In the current state of the simulation, OSM imports are supported but flexible enough to be fully usable. Currently, OSM datasets can be imported in the pre-build of the simulation. This means that to replicate a real-life location in the simulation,

the user would have to manually import the OSM file into the Unity project, and then change the path to the file within the code. In the future, there should be functionality to either import OSM data directly from within the simulation, or a way to import OSM files without having to manually edit code.

Additionally, OSM imports are not perfect in its current stage. Because of the lack of support for intersection types, the road network can end up generating in an unfinished state. In the same way, there have been other issues, for example, when two intersections are located next to each other. One of them usually fails to generate due to the overlap between the intersections. Finally, not all data from the OSM file is used when generating the world. In addition to the current POIs outlined in 3.2.2, one data point that should be implemented in the future, are charging stations for electric vehicles.

5.5.3 Vehicle Types

With the previously mentioned charging and fuel stations, different vehicle types representing electric and gas driven cars could be included. This would improve the statistical accuracy of the simulation, allowing the user to modify the differential between the two types. The reason this should be implemented is because of the increase in electric vehicles registered in the world, and their effect on emissions[54].

Equally important, other forms of transport should be included as well. These would include trains, trams, and taxis which are commonly found within, or nearby cities. With the inclusion of these vehicles, the amount of cars in a road network should decrease due to the use of public transport. However, this depends on the efficiency, availability, and cost of said public transport. With the reduced amount of cars in the road network, both efficiency will increase, and total emissions should decrease.

5.5.4 Performance Optimisation

One of the major limitations of the tool is the performance. Simulating individual agents is a performance intensive task. The physics engine used by the vehicles to provide realistic handling uses a lot of resources, especially when simulating many vehicles. The performance mode helps with this, but would need to be developed further to allow larger networks to be simulated with reasonable performance. The mathematical calculations it currently uses could be optimized or perhaps even simplified while still fulfilling the requirements.

A great way of achieving a significant increase in performance is by multithreading heavy workloads. Nowadays almost all computer processors have multiple threads allowing them to perform tasks in parallell. Some workloads are more difficult to parallellize, as multiple cores simultaneously reading and writing from the same memory locations can interfere with each other. A heavy task that could easily benefit from multithreading is the OSM import, where there is a clear separation

between several import stages. This would allow the maps to be generated quicker, especially on less powerful hardware. It is also possible to multithread the vehicle physics calculations, however this is much harder as they interact with each other. Therefore, all vehicles need to be updated before the program can move on, which will limit the benefit of this optimisation.

5.5.5 Simulation Improvements

One of the areas where there is a lot of potential for improvement is related to the simulation itself. The simulation could be made more realistic by improving the target assignment algorithm, which determines where the vehicles should navigate to. The algorithm can be expanded to include parameters such as the time of day, to determine what the vehicles should do. This would allow the implementations of rules causing many vehicles to simultaneously drive from houses to companies and industrial areas in the morning, and then returning in the evening. This would further stress the road system and is something to take into consideration.

In addition to the buses following routes and stopping at the bus stops, their behaviour could be extended to follow schedules with a limited pool of available buses. This would limit the amount of people that could use the public transport system, causing more vehicles to appear in the network. Expanding on the public transport integration is something that could make the tool even more useful and is an innovative area, as noted during the user tests where it was mentioned that other tools lack this feature. This could be used for finding the optimal balance between public transport and personal vehicles, and aid in transportation planning.

5.5.6 Statistics

Statistics is one of the areas where improvements can almost always be made when analysing data. Therefore, many new ways of visualizing the same data can put fourth to widen the target audience. Moreover, with the extensive amounts of data that can be produced, many different comparisons can be made. The currently implemented statistics and graphs are related to important environmental factors like fuel-consumption and emissions, as well as network performance related statistics such as congestion over time. With the multitude of different techniques of representing data, including new parameters, the possibility of adding more statistics is almost endless.

Statistics could also be improved visually, by adding filters or features that highlight or better indicate the data that the user is searching for. An example would be adding more visual highlighters - in the same fashion that the the roads are highlighted in red when congested - for other variables such as road surface wear.

5.5.7 Map Editor

An area where the tool is lacking, is the creation of the network. As it stands, the only way to manipulate the environment, POIs, roads and other aspects of the

infrastructure is through the built-in Unity editor. This creates a barrier for users without any knowledge of Unity, and might discourage them from further usage of the tool. Even with the understanding of how to do this, it is a tedious and time consuming process. Having to use a separate software for some tasks also not intuitive for the user. To solve this, a built-in map editor would be developed, allowing the user to move, manipulate, create and delete any of the aforementioned aspects inside the tool. This would dramatically improve both the efficiency and the simplicity of the process, further boosting the tools user-friendliness.

6

Conclusion

This section will summarize the project. It goes into detail on how future problem presentations might arise, and how the tool might be configured to be used in other fields. Lastly, it will quickly summarize some of the skills that were learned during the project.

6.1 Additional Knowledge

There are two main areas where additional knowledge would have had the most impact on the final product. The first area revolves around programming, and the game engine that was used. To start off, a large portion of the group members had never worked on a project of this size, nor in a team this big. With a lack of experience, we quickly found it hard to keep progress organized which resulted in the repository becoming cluttered. With more experience and prior knowledge in working with larger software projects, the workflow would have been smoother, and mistakes could have been avoided.

The other area where more knowledge would have been beneficial, is surrounding infrastructure engineering, traffic flow modeling and ABM systems. With some background in these subjects, it would allow for increased accuracy and quicker development of the vehicle simulation system. Moreover, it would have given us a better understanding on which statistics are most important, and the best way to present them to the user.

6.2 Future Problem Presentations

Given the insights gained throughout the development process, another area that might benefit from a tool like this would be airports. They, like the road networks, have traffic they need to handle, and require planning to construct airports with large capacities. They have taxiways where the airplanes move between the gates and the landing strip. These relocating planes interfere with other planes trying to take off or land, which means that a good traffic flow is a requirement for a well-functioning large scale airport. A simulation tool like this one could be tailored towards airports, investigating the different synergies that affect the performance of an airport. The tool should also allow for different airport configurations to be simulated in order

to evaluate them. Airport traffic brings with it a different set of challenges and relationships. One example is that all airport traffic is scheduled, which means that well planned arrival and departure times can improve the performance, increase the capacity and the overall throughput of an airport.

As the future looks to bring more and more autonomous vehicles, transportation planning tools could play an even more important role. If all vehicles are autonomous, the traffic could be centrally controlled to avoid congestions, and balance the load in the road system to optimise its performance. A tool like this could be extended or repurposed in order to analyse these future traffic flows. New possibilities also emerge, since solutions could be based around the fact that the traffic is autonomous, and can therefore easily be redirected to better utilise the available road network. Perhaps it would be beneficial to redirect some vehicles on slight detours, making the travel time slightly longer. This could ease the traffic in some areas, and better distribute the traffic, to overall decrease the travel times in the road system as a whole.

6.3 Future Usage of the Simulation

In the future, many new use cases can be developed for the current simulation. An example for this could be to convert the simulation to a video game, allowing users to select a real-life location, and then drive around in that location with realistic traffic. Other than the previously mentioned improvements, the only feature that would have to be implemented is a way for a user to drive a vehicle on the road. This could quickly be implemented due to Unity's vast support for user inputs and camera control.

Similarly, the simulation could be converted to a tool that would help people learn how to drive. In the same way, first person controls and cameras could be added, which could allow the user to use driving simulator hardware such as a steering wheel, pedals, and a gear shifter. This could be beneficial to driving schools since it could assist the students during the early stages of learning how to drive a car. The student would learn the basics in the simulator, without the risk or cost that comes with driving a real car in traffic with vehicles.

6.4 Learning Outcome

If we were to redo the project from scratch, we would have changed a few aspects of our approach. To begin, we would have lowered the scope of the project, so that we instead could focus on improving the quality. This would result in a simulation with fewer use cases, but with better core functionality. Therefore allowing the project to grow exponentially with more features in the long term, due to the added stability. Another aspect that would change, is that our custom road generator tool would be prioritized to be implemented. With access to this asset from the beginning, we could instead spend our time focusing on adding new features, and improving the simulation at its core.

In the end, this project resulted in us learning numerous new skills which would further our expertise as we continue to grow and learn. In the beginning of the project, only a few of us had used Unity before, and only to a limited state. Now, we are all knowledgeable within the game development platform. In a similar way, this is reflected in many parts of the project, including but not limited to; Git, the scrum framework, C#, road network design, and more importantly, working efficiently as a software development team.

Bibliography

[1] Inrix, "Inrix 2022 global traffic scorecard," https://inrix.com/scorecard/, accessed: 2023-01-30.

[2] "Resvanor," https://www.trafa.se/kommunikationsvanor/RVU-Sverige/, May 2022, accessed: 2023-01-30.

[3] A. J. Cohen, H. R. Anderson, B. Ostro, K. D. Pandey, M. Krzyzanowski, N. Künzli, K. Gutschmidt, C. A. Pope III, I. Romieu, J. M. Samet *et al.*, "Urban air pollution," *Comparative quantification of health risks: global and regional burden of disease attributable to selected major risk factors*, vol. 2, pp. 1353–1433, 2004.

[4] K. Gallagher, "How much air pollution comes from cars?" https://www.treehugger.com/cars-are-causing-air-pollution-we-breathe-new-study-finds-4856825, Aug 2022, accessed: 2023-01-30.

[5] J. Nguyen, S. T. Powers, N. Urquhart, T. Farrenkopf, and M. Guckert, "An overview of agent-based traffic simulators," *Transportation Research Interdisciplinary Perspectives*, vol. 12, p. 100486, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590198221001913

[6] J. Zeng, Y. Qian, Z. Lv, F. Yin, L. Zhu, Y. Zhang, and D. Xu, "Expressway traffic flow under the combined bottleneck of accident and on-ramp in framework of kerner's three-phase traffic theory," *Physica A: Statistical Mechanics and its Applications*, vol. 574, p. 125918, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378437121001904

[7] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker-Walz, "Recent development and applications of sumo - simulation of urban mobility," *International Journal On Advances in Systems and Measurements*, vol. 3&4, 12 2012.

[8] E. Foundation, "Eclipse sumo: Simulation of urban mobility," 2021, accessed: 2023-04-28. [Online]. Available: https://www.eclipse.org/sumo/

[9] H. U. Ahmed, Y. Huang, and P. Lu, "A review of car-following models and modeling tools for human and autonomous-ready driving behaviors in micro-simulation," *Smart Cities*, vol. 4, no. 1, p. 314–335, Mar 2021. [Online]. Available: http://dx.doi.org/10.3390/smartcities4010019

[10] U. Technologies, "Unity user manual (2021.1)," https://docs.unity3d.com/Manual/index.html, 2021.

[11] K. Cheliotis, "Abmu: An agent-based modelling framework for unity3d," *SoftwareX*, vol. 15, p. 100771, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711021000881

[12] A. Developers, "ABMU: Agent-based modelling framework for Unity3D," https://github.com/ABMU/ABMU, 2021.

[13] J. M. Epstein and R. Axtell, *Growing Artificial Societies: Social Science from the Bottom Up.* Brookings Institution Press, 1996.

[14] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 25–34, 1987.

[15] T. C. Schelling, "Dynamic models of segregation," *Journal of Mathematical Sociology*, vol. 1, no. 2, pp. 143–186, 1971.

[16] I. C. Office, "Consent," https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/consent/, n.d.

[17] E. Parliament and Council, "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)," pp. 1–88, 2016.

[18] M. Mora-Cantallops, S. Sánchez-Alonso, E. García-Barriocanal, and M.-A. Sicilia, "Traceability for trustworthy ai: A review of models and tools," *Big Data and Cognitive Computing*, vol. 5, no. 2, 2021. [Online]. Available: https://www.mdpi.com/2504-2289/5/2/20

[19] M. Staff, "Over the edge releases unity 1.0 game engine," https://www.macworld.com/article/176746/unity-3.html, Jun 2005, accessed: 2023-05-09.

[20] A. Webster, "Former ea ceo john riccitiello appointed ceo of unity," https://www.polygon.com/2014/10/22/7039683/electronic-arts-john-riccitiello-unity-ceo, 2014, accessed: 2023-05-09.

[21] S. Christoforou, "Did you know that 60% of game developers use game engines?" https://www.slashdata.co/blog/did-you-know-that-60-of-game-developers-use-game-engines, 2022, accessed: 2023-05-09.

[22] R. Nystrom, "Game loop," https://gameprogrammingpatterns.com/game-loop.html, accessed: 2023-05-09.

[23] U. Technologies, "Order of execution for event functions," https://docs.unity3d.

com/Manual/ExecutionOrder.html, 2023, accessed: 2023-05-09.

[24] ——, "Event functions," https://docs.unity3d.com/Manual/EventFunctions.html, 2023, accessed: 2023-05-09.

[25] S. Gentle, "Bezier playground," 2018. [Online]. Available: https://samgentle.com/playgrounds/bezier

[26] G. Farin, *Curves and Surfaces for Computer-Aided Geometric Design*. Academic Press, 1993.

[27] A. A. Shavez Kaleem, "Cubic bézier curves," 2000. [Online]. Available: https://mse.redwoods.edu/darnold/math45/laproj/Fall2000/AlShav/bezier-dave.pdf

[28] T. Sederberg, "Computer aided geometric design," 2012. [Online]. Available: https://scholarsarchive.byu.edu/cgi/viewcontent.cgi?article=1000&context=facpub

[29] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[30] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson, 2016.

[31] R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of A*," *Journal of the ACM*, vol. 32, no. 3, pp. 505–536, 1985.

[32] S. F. Railsback and V. Grimm, *Agent-based and individual-based modeling: A practical introduction*, 2nd ed. Princeton University Press, 2019.

[33] E. Bonabeau, "Agent-based modeling: Methods and techniques for simulating human systems," *Proceedings of the National Academy of Sciences*, vol. 99, no. Suppl 3, pp. 7280–7287, 2002.

[34] S. F. Railsback, U. Berger, J. Giske, G. I. Hagstrom, B. C. Harvey, C. Semeniuk, and V. Grimm, "Bridging levels from individuals to communities and ecosystems: Including adaptive behavior and feedbacks in ecological theory and models," *Bulletin of the Ecological Society of America*, vol. 101, no. 1, pp. 1–10, 2020. [Online]. Available: https://www.jstor.org/stable/26853212

[35] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[36] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[37] B. Nystrom, "Observer," https://gameprogrammingpatterns.com/observer.html, 2014, accessed: 2023-05-09.

[38] Microsoft, “Delegates (c# programming guide),” https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/, 2021, accessed: 2023-05-09.

[39] ——, “Eventhandler delegate,” https://docs.microsoft.com/en-us/dotnet/api/system.eventhandler?view=net-7.0, 2021, accessed: 2023-05-09.

[40] ——, “Events (c# programming guide),” https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/, 2021, accessed: 2023-05-09.

[41] U. Technologies, “Unityevents,” https://docs.unity3d.com/Manual/UnityEvents.html, 2021, accessed: 2023-05-09.

[42] R. Guru, “Singleton,” https://refactoring.guru/design-patterns/singleton, 2021, accessed: 2023-05-09.

[43] B. Nystrom, “State,” https://gameprogrammingpatterns.com/state.html, 2014, accessed: 2023-05-09.

[44] OpenStreetMap, “Osm,” https://www.openstreetmap.org/about, accessed: 2023-05-09.

[45] “Edy’s vehicle physics,” https://assetstore.unity.com/packages/tools/physics/edy-s-vehicle-physics-403.

[46] “Bézier vehicle creator,” https://assetstore.unity.com/packages/tools/utilities/b-zier-path-creator-136082.

[47] “Git,” https://git-scm.com/.

[48] “Github,” https://github.com/.

[49] J. Olstam, C.-H. Häll, G. Smith, and A. Habibovic, “Dynamic bus lanes in sweden - a pre-study,” 2015.

[50] J. Goerz, “Arts 102 aesthetics of the algorithmic image,” 2005. [Online]. Available: https://www.mat.ucsb.edu/~g.legrady/academic/courses/05f102/jg_flocking.html

[51] “Trello,” https://trello.com/.

[52] “Ptv visum,” https://www.ptvgroup.com/se/loesningar/produkter/visum/.

[53] MicroGSD, “Roadarchitect,” https://github.com/MicroGSD/RoadArchitect, accessed: 2023-05-10.

[54] IEA, “Electric vehicles,” https://www.iea.org/reports/electric-vehicles#, 2022, accessed: 2023-05-10.