CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Simulating Traffic Flows and Analysing Road Network Design

Investigating the relationship between road network design and traffic congestion

Bachelor's thesis in Computer science and engineering

MARTIN BLOM
FELIX JÖNSSON
HANNES KAULIO
MARCUS SCHAGERBERG
JAKOB WINDT

Simulating Traffic Flows and Analysing Road Network Design

Investigating the relationship between road network design and traffic congestion

MARTIN BLOM

FELIX JÖNSSON

HANNES KAULIO

MARCUS SCHAGERBERG

JAKOB WINDT

UNIVERSITY OF GOTHENBURG

CHALMERS

UNIVERSITY OF TECHNOLOGY

Simulating Traffic Flows
Investigating the relationship between road network design and traffic congestion
MARTIN BLOM FELIX JÖNSSON HANNES KAULIO
MARCUS SCHAGERBERG JAKOB WINDT

Cover: Description of the picture on the cover page (if applicable)

Typeset in LATEX
Gothenburg, Sweden 2023

iv

Simulating Traffic Flows
Investigating the relationship between road network design and traffic congestion
MARTIN BLOM, FELIX JÖNSSON, HANNES KAULIO, MARCUS SCHAGER-BERG, JAKOB WINDT

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This document is *only* a LATEX template. It is not meant to suggest a particular structure. Also, even if this document is written in English, is is not meant to suggest a report language. You can adopt it to your language of choice. In this document, the bibliography is hand made. However, we suggest that you strongly consider using BIBTEX, to further automate the creation of the bibliography.

Sammandrag

Keywords: put, here, keywords, describing, areas, the, work, belongs, to

Acknowledgements

If you want, you can here say thank your supervisor(s), company advisors, or other people that supported you during your project.

Contents

List of Figures

List of Tables

Glossary

Agent: Autonomous entities that inhabits an environment and act based on pre-defined rules.

Agent Based Model (ABM): A computer simulation model in which agents interact with each other and their environment to produce emergent behavior.

Data Structure: A way of organizing and storing data in a computer so that i can be accessed, manipulated, and modified efficiently. Some common examples are arrays, stacks, and linked lists.

Information Visualization: Field that focuses on creating meaningful and easy to to interpret graphical representations of data.

MonoBehaviour: Base class for Unity scripts. Provides access to event functions such as Start(), Update(), and so on.

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.

Scrum: The scrum agile project management framework provides structure and management of work and is popular among software development teams.

Scrum-boards: A bulletin board that keeps track of a backlog, the current sprint, and completed stories.

Story: In the scrum framework, a story is essentially a set of tasks that will result in a new or updated desired functionality/product.

Unity: Cross-platform game engine.

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.

UI: User Interface (UI) is the point between human-computer interactions. It is what is used for user interactions with the program.

UX: User Experience (UX) refers to the overall experience of the actual user of a product. The goal of good UX design is to create intuitive and enjoyable products.

C#: C# is a programming language developed by Microsoft that runs on the platform .NET Framework. C# is pronounced as "C sharp" and belongs to the programming language family of C.

C++: C++ is one of the most popular general purpose programming languages. C++ is pronounced as "C plus plus" and belongs to the programming language family of C, with the major distinction from C that C++ is an object oriented language.

A*: A* is a popular graph traversal and path searching algorithm due to its completeness and optimal efficiency. A* is used to find the shortest possible path from one node to another.

Repository: A repository acts as a container that stores a projects files and their individual revision history.

Wire Frames: Wire Frames depict how the UI layout will appear during different stages of the program.
OSM:

1

Introduction

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 a higher increase of time and fuel wastage. Moreover, delaying transportation of goods can lead to an increase in economical costs, food waste and general inconvenience for the effected 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 go down, 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. But if you look at some of the bigger 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 stood 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 that 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 both in real-time, and as post-simulation data 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 traffic flow, travel time, and 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 highly 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 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 extensible and user-friendly programming interface for Unity's resources to create the foundation for a powerful and extensible 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, ensuring a more robust and accurate

simulation. These events get delegated to so called Steppers, which are a modular component that encapsulates a targeted behavior or action. These are made to be easily added and removed to enable researcher 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, and 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 implementaitons such as Epstein and Axtell's Sugarscape model[13], Reynolds' Boids model[14], and Schelling's segregation model[15]. These examples demonstrate the flexibility and capability of the framework, as well as offer guidance for users looking to develop their own models with 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 different end-users of various occupation connected to traffic planning, and offer these users insight about the efficiency and emission associated with different set up of road networks. 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 limitation 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 Unity3D. 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, artificial intelligence, and more. 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 3D and 2D environments, and it offers native support for building on a wide variety of platforms. Although there is a wealth of underlying theory supporting the engine, arguably the most prominent is 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 concrete 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 up to the event architecture and respond to specific events such as "OnCollisionEnter" 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 (sometimes 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 in 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's define this as $P_0 \rightarrow P_1$. A visual representation can be found in figure 2.1.
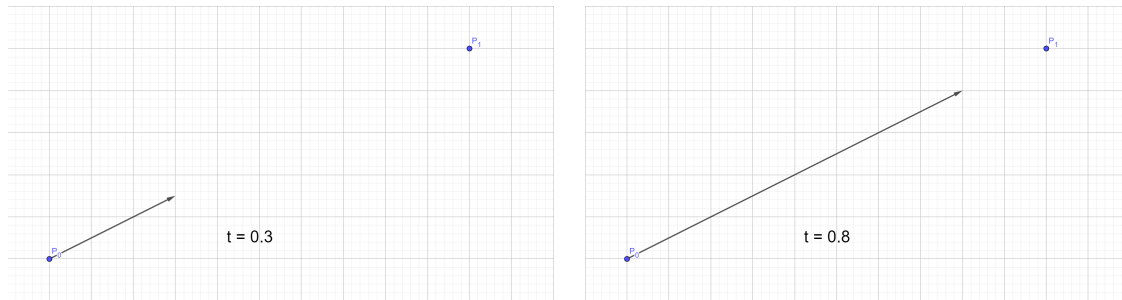
Figure 2.1: A linear interpolation for two values of t

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

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

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.
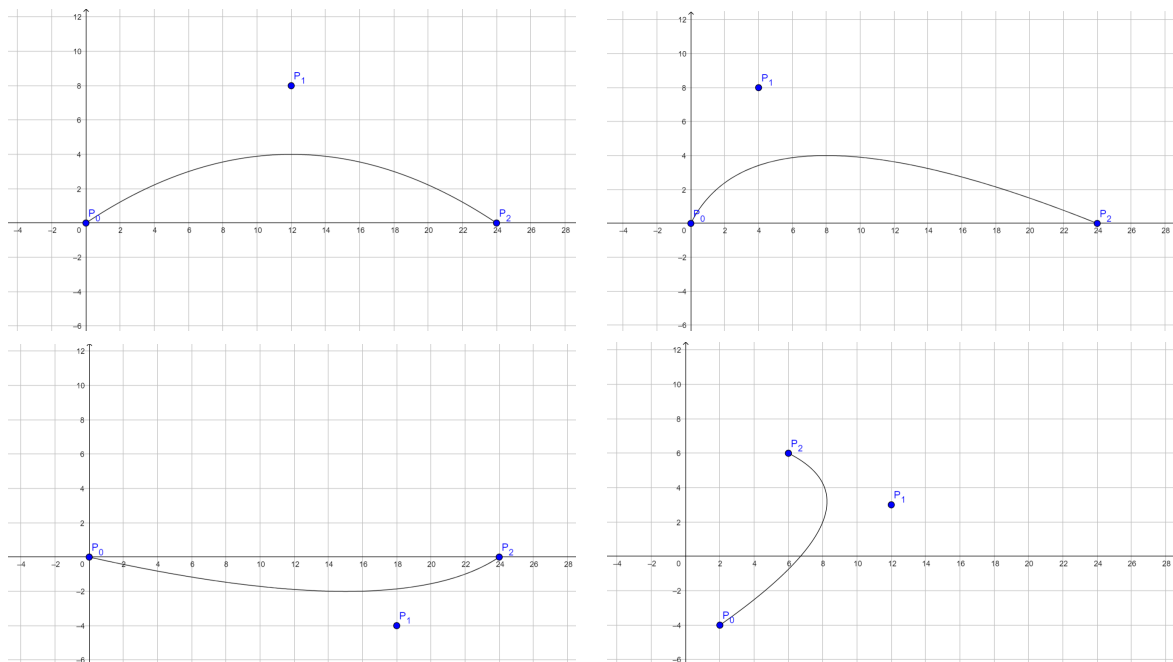
Figure 2.2: Four examples of quadratic Bézier curves

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 interpolations. A cubic Bézier curve has four control points, two of which are endpoints.
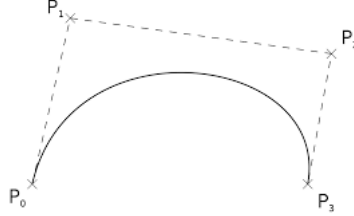
Figure 2.3: Cubic Bézier curve with control points P_0, P_1, P_2 and P_3

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 [28] can be used. 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.

$intersections \leftarrow$ Empty list
$epsilon \leftarrow$ 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**
  $intersections \leftarrow$ Midpoint between A and B
  return
 end if
 $A_1, A_2 \leftarrow SplitWithDeCasteljau(A, 0.5)$
 $B_1, B_2 \leftarrow SplitWithDeCasteljau(B, 0.5)$
 $FindBezierPathIntersections(A_1, B_1)$
 $FindBezierPathIntersections(A_1, B_2)$
 $FindBezierPathIntersections(A_2, B_1)$
 $FindBezierPathIntersections(A_2, B_2)$
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 the segments approaches points 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 other 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[29] as part of a project focused on constructing a mobile robot capable of autonomously devising its actions. 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 (POI). Here, POIs primarily serve as target destinations for our agents and may include parking slots, fuel depots, 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 is 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 using the following steps:

1. Label the start node s as "open" and compute $f(s)$.

2. Choose the open node n with the smallest 'f' value. Break ties randomly, but always prioritize 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. Going back 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 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 is delivered in small complete chunks, this 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, while still achieving 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 relative 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 is 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 occurs 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, across-level 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 its 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.". 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. 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 (Figure 2.4) is a behavioral design pattern that allows you to separate 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 (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.

---

[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: For-loop

2.7.1   The Observer Pattern

One of the first design patterns introduced in the aforementioned book by the "Gang of Four", the Observer pattern is a behavioral pattern that addresses several different key problems in object-oriented programming. It is implemented by creating two separate interfaces: the publisher who is responsible for publishing events 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.
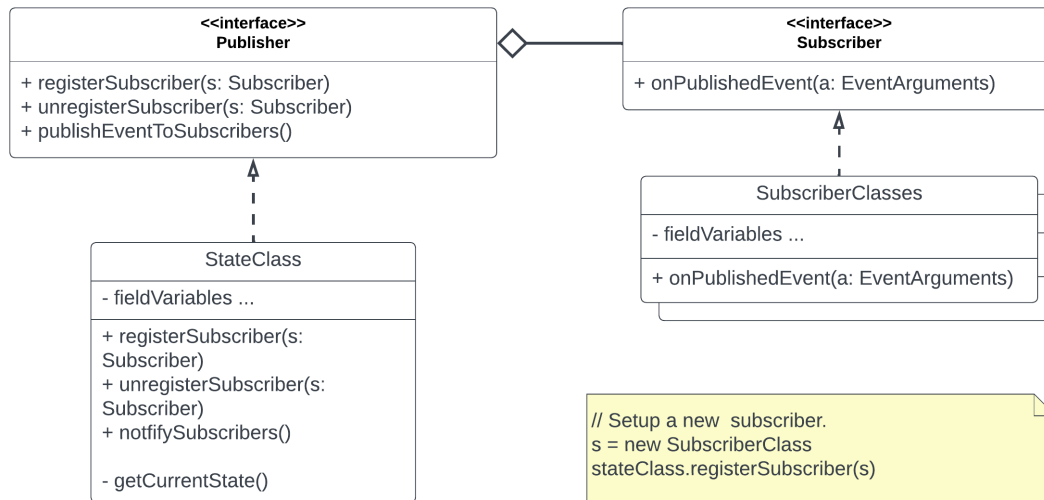
Figure 2.6: UML diagram of generic publisher/subscriber-relation implementation.

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 facilitates 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 2.7. The list that follows highlights some of the most prominent components that are used to promote the usage of this pattern:

- **Delegates (C#)**: Define a type representing a specific user-defined method signature, allowing for a method-based event system[35]. 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[36], 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[37]. 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 run-time.

- **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[38]. 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 that ensures only one instance of a specific class can be created at any given time, providing global access points to that instance. 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 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. 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 told to. This main class can be described as mediator between different states, and offers the developers an 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

---

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

concept of finite state machines (FSM) which are computational model used across a wide spectrum of domains such as control systems and artificial intelligence. A FSM consists of a finite amount of states, the initial state, and the adhering transitions between them. While both FSM and the State pattern deals with managing system behavior through various states and transition, FSM is a more general concept that focuses on the overall structure of a system, while the State pattern is a specific object-oriented concept focusing on adhering to good object-oriented principles.

By seperating the state logic into a seperate interface, 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 realized by concrete states classes that provides 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.
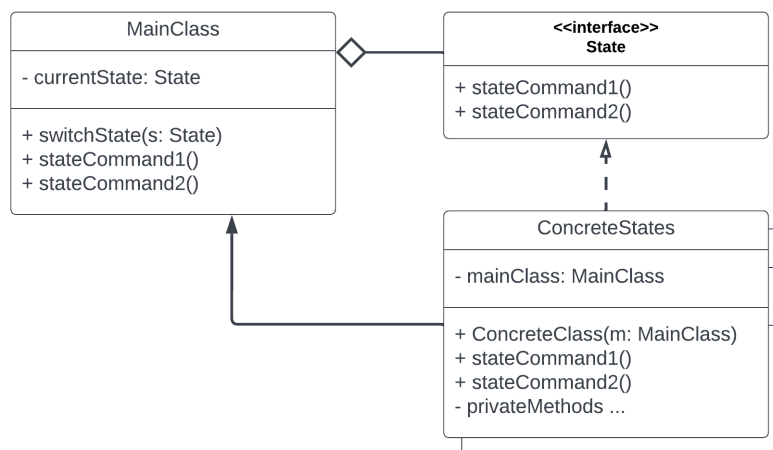
Figure 2.7: UML diagram of generic State pattern implementation.

- https://www.amazon.com/-/dp/0195019199

---

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

- Design Patterns: Elements of Reusable Object-Oriented Software

- https://gameprogrammingpatterns.com/observer.html

2.8   OpenStreetMap

OpenStreetMap, or OSM for short, is a free database which contains geographical data. The database is maintained and updated by volunteers. Volunteers can collect data about geographical areas and add to the database for everyone to use. Data about features such as roads, railroads, buildings, trees, etc and their properties exist.

3

Methods

3.1 Tools

3.1.1 Unity

The traffic simulation tool is built in a well-known game-engine called Unity. 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 some of the team members had previous experience with. Furthermore, C# is a higher level language compared to C++, the main language of Unreal Engine, making it easier for the team members without experience to learn. Because of this, the time it took to begin programming in the early stages of the project was most likely shorter, compared to if Unreal Engine was chosen as the platform.

Another reason would be that Unity comes with the Unity Asset Store, a marketplace for acquiring creator made assets. This feature is important because, for example, instead of having to create custom models for the vehicles, they could instead be purchased using the given budget. This saves a lot of time, that could be better spent on other parts of the project. One of the more notable purchased assets is Edy's Vehicle Physics[39] that are used to rig vehicle models with realistic physics. Instead of having to develop custom vehicle physics for each model, the team could instead use the asset to quickly configure a model with physics.

The final reason why Unity was chosen, was due to its flexible developing structure. The level of customization available inside the engine is a lot 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 GitHub

A commonly used tool when developing software in larger groups is Git[40]. Git is a free and open-source version control system that allows its users to collaborate in an efficient and easy way.

GitHub[41] 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. Before a team member can merge their new code to the main repository, the code would have to be reviewed by at least one other team member to ensure that the code was well commented, functional, and that it follows the C# coding standard.

3.1.3   Balsamiq Wireframes

During the first stage of creating a UI, it is important to start with a simple mock-up design. This is what the tool, Balsamiq Wireframes[42], is used for. The user can quickly design wire frames depicting how the UI will appear during different times in the program. This includes everything from buttons to pop-up menu's that might appear in the simulation tool.

3.1.4   Third-Party Assets

Built into Unity is their asset store. Instead of creating everything from scratch, the team opted to purchase some assets. An asset can be anything from a 3D model to animation and scripts. The two main assets purchased for the project are Edy's Vehicle Physics and European Road Signs. Edy's Vehicle Physics is a package that includes a tool that allows its user to easily implement realistic vehicle physics into 3d models. This saves a substantial amount of time because there would be no need to create custom physics attribute for each vehicle model.

As the name states, the European Road Signs assets include a plethora of street signs, as well as an editor to customize them. Without this asset, there would have been a need to create custom 3D models and texture, which no team member had previous experience with.

3.2   Simulation Design and Implementation

3.2.1   ABM

3.2.2   Road Generation

In order to achieve realistic roads with adequate curves, composite Bézier curves were used. The Bézier control points will shape the road and its characteristics. A number of parameter can be changed in the Bézier path to change the appearance. The position and sharpness of the turn can be modified by changing were the control points are placed in relation to each other.
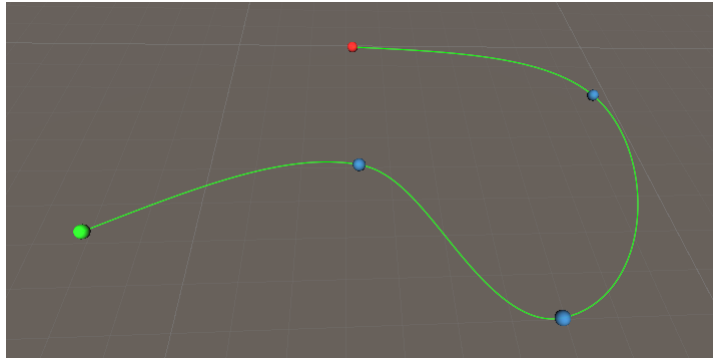
Figure 3.1: Composite Bézier path

While the Bézier curves give a good ground level for the road implementation, it is hard to implement and build logic based on it since it is a continuous path. The Bézier logic and its control points was abstracted away with a node implementation placed on top of the Bézier path. A number of nodes called RoadNodes is placed along the Bézier path at a rate dependent on curve of the road. The nodes are all connected the its previous and its next node along the path.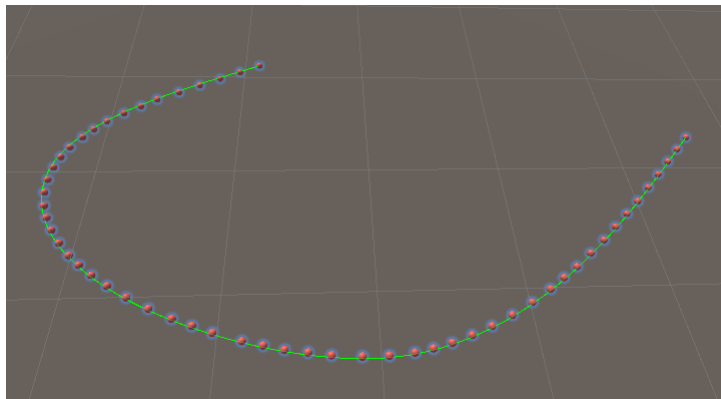 The goal of these nodes is to carry enough information to procedurally build the road mesh as well as carry some logic needed for agents to navigate the environment.

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

By using the RoadNodes, generating the road mesh is possible. The mesh is procedurally generated by placing vertices at the RoadNode and along its normal line in both direction at a length equal to the width of a lane. If multiple lanes is wanted, vertices can be continually added in each normal direction for the lane amount. In addition to these, vertices are also placed below them to add thickness to the road. Triangles are then drawn between these vertices to create the mesh. To add the road material, sub meshes is created for each lane. The power of procedural generation is the ability to customize the roads different parameters. The width of the lanes, width of the lines, thickness of the road and number of lanes can all be changed for each road.
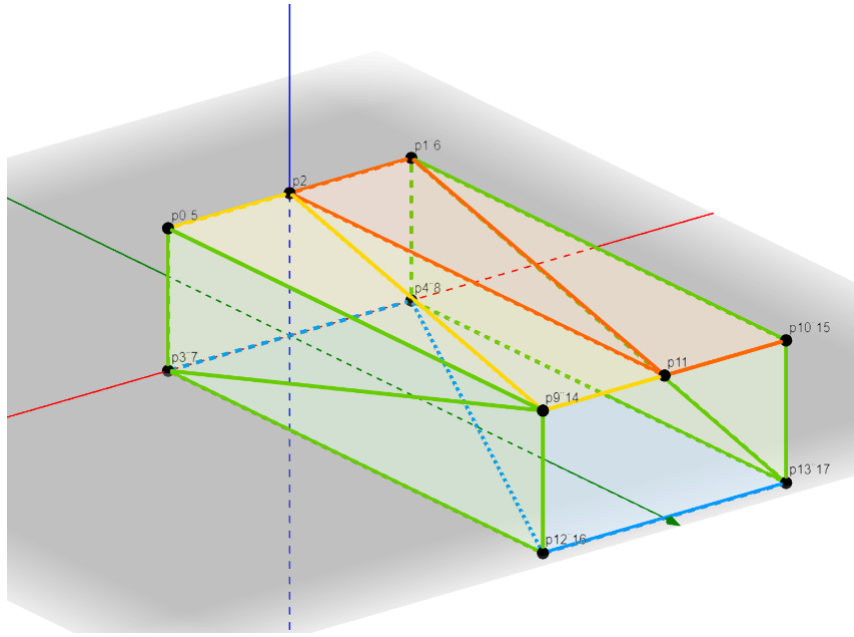
Figure 3.3: The vertices and triangles used to generate the one lane road mesh
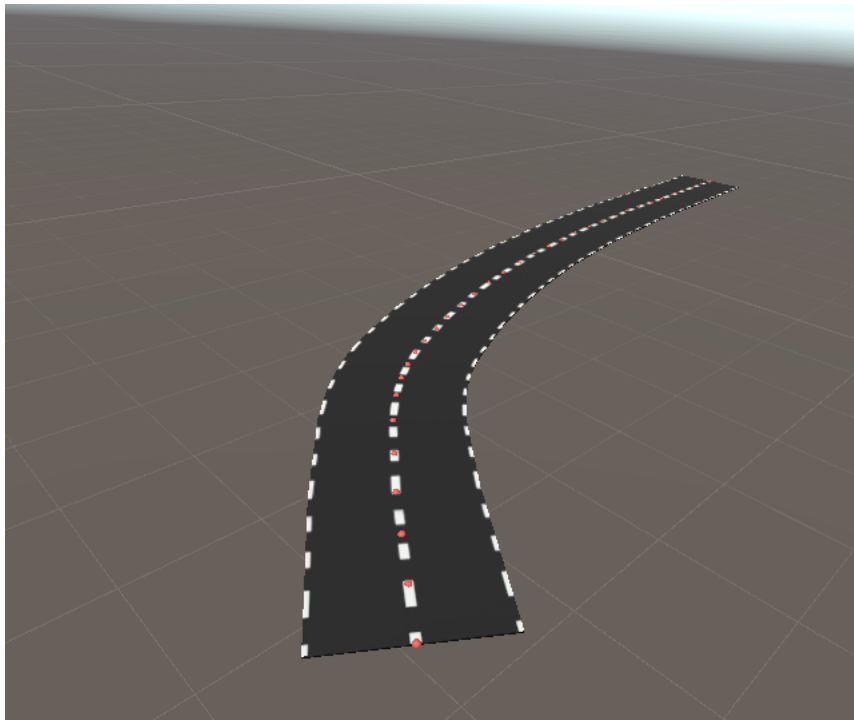
Figure 3.4: Road generated from its RoadNodes

While the RoadNodes carry a lot of critical logic, it is lacking logic for driving along the different road lanes. This logic is added with another node that is placed along the road, the LaneNode. LaneNodes are placed at both sides of the normal line of each RoadNode at the middle of the lanes. These nodes are responsible for the

road steering as well as notifying other agents if they are currently occupied by any agents.
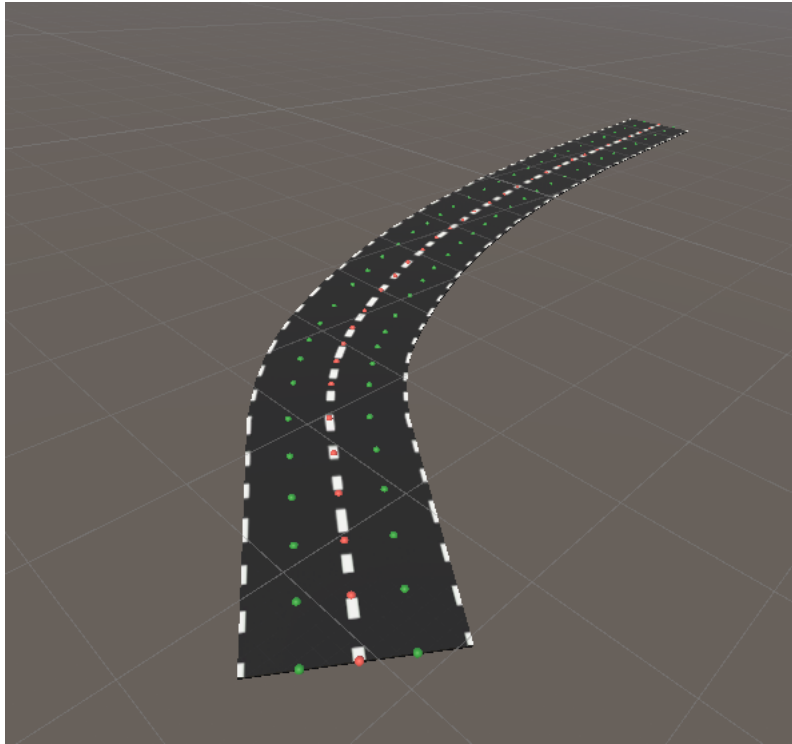
Figure 3.5: Visual representation of RoadNodes (Red) and LaneNodes (Green)

3.2.3   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 for each other 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 currently 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 between roads. 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 path the vehicle is trying to take. This means the vehicles will avoid any traffic inside the intersections, however, as vehicles are travelling towards each other in an intersection this is not quick enough as they will notice each others' occupation too late. This leads into the second concept, yielding for crossing nodes. Upon entering an intersection, each vehicle will also check all nodes of any crossing paths, looking as far back as required to ensure that no other vehicles will arrive at the intersecting points 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 thought of as 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 [43].

3.2.4 City Generation

To aid in simulating cities and larger areas, existing real life locations are generated from OSM data. The OSM data specify the latitude and longitude of every road and its path. Real life building data is also used to generate a representation of the buildings. The OSM file is parsed and the roads are generated with its specified characteristics. The speed limit, road type and if the road is lit is all considered when generating the road.
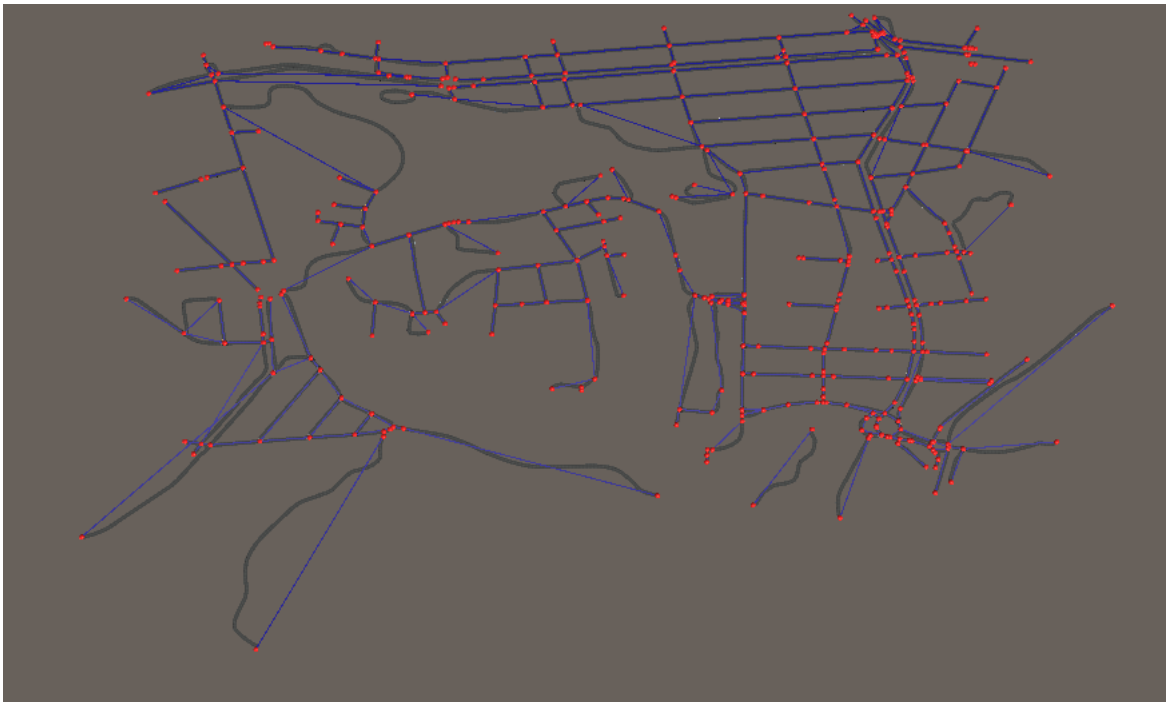
3.2.5 Navigation

The basic navigational responsibility of each agent is the ability to follow the road lanes, avoid colliding into other agents, follow the traffic rules and being able to navigate to a given position.

In order to follow the lanes, each agent follow the LaneNodes on the road. The LaneNodes store all the information needed to navigate the road. The position of the node, the agent that is currently on the node and special traffic rules the vehicles need to follow are stored on the node. Traffic signs such as stop sign are represented as a node and the traffic logic can be accessed by the agent when they encounter the node.

The agents steer towards a node that is a certain distance in front of the car. This distance is influenced by the current speed. By steering towards nodes that are in front of the agent, a smooth and reliable steering is achieved. Similarly to steering, braking is accomplished by looking at nodes at a certain distance ahead. When a node with stop logic is found, the agent will break and stop before that node. This is done by looking for stop nodes at a distance ahead equal to the brake distance of the agent. The agents also claim each node they are over so other agents can stop when the node at the break distance is claimed.

To enable the ability to navigate the roads, a weighted directed graph is created from the roads. The graph nodes are all the road endpoints and intersections and POIs. The edges between the nodes are weighted with a cost that is calculated as the distance * the speed limit. The agents navigate to a given end node by receiving a path of edges from the A* algorithm. When an agent doesn't have an active navigation path, it will be assigned path that will lead it to a given destination. The agent maps out the path and saves instructions for were to turn in each intersection that is passed when navigating to the destination.

Figure 3.6: Visual representation of a navigation graph

3.3   Performance

3.3.1   Quality vs Performance

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

While testing the simulation during development, the most noticeable performance cost were the vehicles on the roads. This is because of the Edy's Vehicle Physics asset that simulates real-world physics to 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 cars in the road network.

3.3.2   Performance Benchmarks

3.3.3   Optimization

When creating software of any kind, it is always important to make sure it is able to run smoothly. To allow the program to run well, optimizations had to be made. 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 skip 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, which are points in a 3d dimensional space. Three of these points are used to construct a triangle, and the triangles in turn build the model. The amount of triangles in a model determines the overall detail of said model. To improve the performance of the simulation, the models that were chosen most contain a low count of triangle, usually less than 20,000.

3.4   User Interface

3.4.1   Design

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

The start menu contains three buttons, one to start the simulation, one to enter the settings menu, and one to exit the program. The settings menu allows the user to change the volume, quality mode, enable a fps counter, and enter full screen.

The overlay that is visible while the simulation runs is what allows the user to

interact with the simulation itself. There are two main types of buttons: the camera buttons and the menu buttons. As the name suggests, the camera buttons are used to control the users 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 allows the user to enter first and third person view for the selected vehicle on the road. The three menu buttons are used to open up their respective menu's. These menu's include the individual vehicle statistics, simulation statistics and the world settings.

3.4.2   Statistics

As mentioned previously, there are two separate UI menus: the individual vehicle statistics and the simulation statistics/world settings. These statistics are presented on movable pop-up windows that can be toggled on or off for convenience.

The individual vehicle statistics are used to give the user an easy overview of how a singled out car is performing in traffic. It shows statistics such as fuel consumption, average speed, distance traveled and time stuck in traffic. This can be useful to see how changes that generally improve traffic flow could still make the traffic worse for a small group of cars. This is important to take into consideration because it can lead the negatively affected cars to take an unexpected route to try and solve their issues, and at the same time negatively alter the traffic flow for the other cars on the road.

The world settings and simulation statistics window gives the user a good overview of the overall efficiency of traffic flow, with detailed information regarding which groups of vehicles account for the most traffic and CO_2 emissions. It also indicates which areas and roads that receive the most traffic and therefore moves at the slowest pace. This is very useful when locating what causes traffic jams and might give insight into possible solutions. Moreover the window is also useful to see how traffic would differ when a certain amount of people choose to take public transport, such as trams or busses instead of driving their own car. Not only will this quickly show the differences of traffic flow but also the improved or worsened CO_2 emissions.

3.5   Workflow

When developing any software larger than just a single use script, the amount of work and information can quickly grow beyond the level of ones own simultaneous comprehension. Therefore, these kinds of projects require rigorous planning and strategizing to work more smoothly through all the different tasks and in a reasonable order, that allows for parallel continuous progress.

To achieve this, a strict work flow 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 3.7.
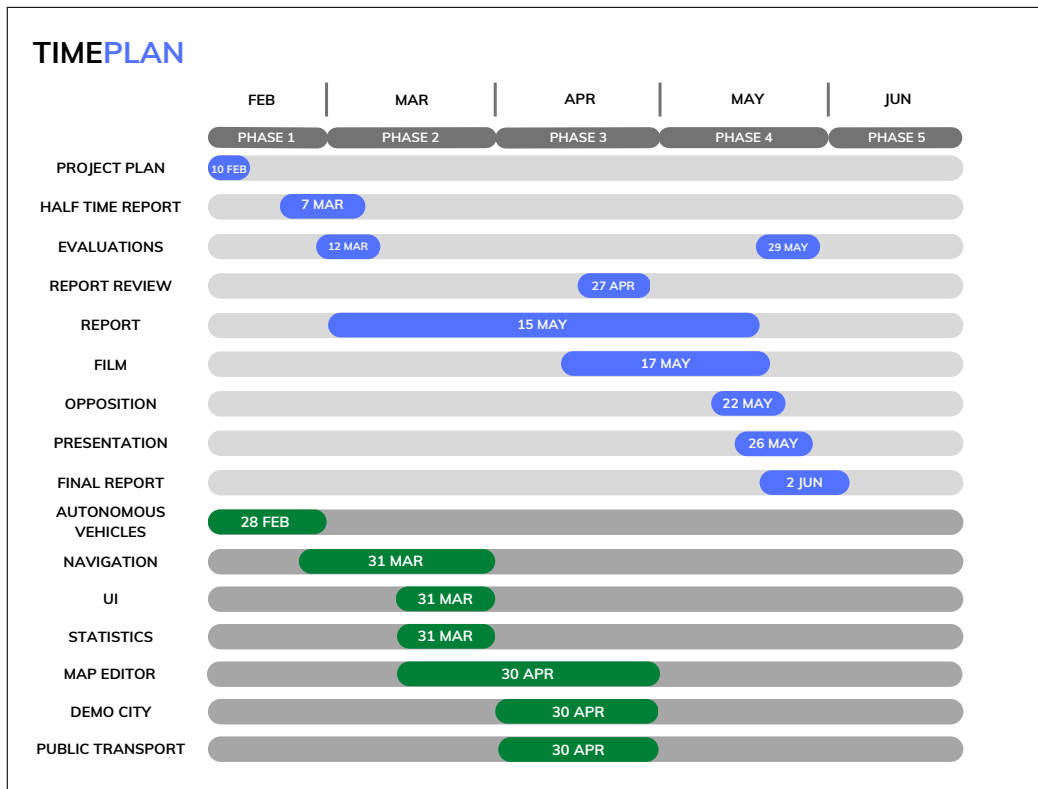
Figure 3.7: Project Time Plan

With this system in place it is easier to keep track of the general progress of the project, as well as helps with planning short term goals. Coincidentally, this is the next step of the work flow model. The short term goals where planned using a scrum framework with weekly sprints, explained in 3.5.1. These sprints were upheld for the duration of the project to keep a stead flow of progress, together with the time plan they create a very clear way of seeing the current state of completeness.

The third aspect of the work flow is the approval of progress. As mentioned earlier, a large project requires a substantial amount of planning. The approval of progress can be seen as just as important as the planning and execution itself. Without a popper method of approving new advancements/functionality, the project can quickly falter. If progress never goes through the process of approval many things can go wrong. Evidently, badly written code can cause issues that are easily preventable with a quick inspection. 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 much further on, which can be difficult and time consuming to resolve. To solve this, code reviews found later is section 3.5.3 for every change made, are part of the workflow.

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 set up this weeks sprint goals. This includes moving and 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 good quality solutions to complete all of the current sprints stories, and eventually work on the backlog as time is presented.

Next up is the review stage, not to be confused with code reviewing 3.5.3, in the next section. In this stage another meeting is held called a "Demo meeting", where all members get to do a small demonstration of all their progress during the sprint. This is an important step to onboard all members on new functionality and make sure that desired behaviour is achieved. When a story is regarded as fully complete it's archived to make room for new ones.

Lastly the retrospect stage, which is usually carried out following the review stage. In the retrospect stage, the current sprints efficiency and quality is discussed. The plans to increase these and the overall effectiveness are also considered. When all is done the cycle begins anew until the project has been completed.

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 where the progress of the task is currently at. Trello[44] 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 our needs.

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 before it is allowed to be merged. This makes sure all code has had at least two pairs of eyes to look at it, increasing the chances of spotting any 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 is 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 has compared to existing solutions.

4

Results

4.1 Final product

4.2 Performance

4.3 User tests

4.3.1 Test subject 1

The first test subject had prior experience in a transportation planning software called PTV Visum, which is the worlds most popular software of its kind when it comes to aiding strategic and operative decisions [45]. 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 easier to see the road connections and intersections in our tool. 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 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 be helpful.

The test subject 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 than our tool. 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. It was also thought that our tool would be useful for education purposes, and that it would fit well for teaching different concepts

in transportation planning as it is intuitive and easy to understand what is going on.

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 and pitch while keeping the keyboard for moving the camera around, and that it felt slow to move around a larger network.

5

Discussion

5.1 Unreached Goals

5.2 Future Improvements

5.2.1 OSM

5.2.2 Support Road and Intersection Types

For intersections, the simulation only has support for 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. To implement this, the vehicles needs to be able to complete the correct type of yield that exists in intersections.

Furthermore, when designing large road networks, especially outside cities, highways are critical. Highways are currently not supported in the simulation because of a few different reason. To begin, there is functionality for multiple lanes when creating roads with the road generator in Unity. However, the vehicles are not able to switch lanes in the simulation, making multiple lanes unusable. Since lane switching isn't supported, vehicles don't have the ability to overtake each other which usually happens on highway. There are also no highway entrances and exits in the simulation, which needs to be implemented for highway roads to function correctly.

5.2.3   Performance Optimization

5.2.4   Simulation Improvements

5.2.5   Statistics

5.3   Additional Knowledge

5.4   User Testing Feedback

5.4.1   Evaluation of the software

According to the user feedback, we found our tool to be intuitive and easy to use.

5.4.2   Changes made based on the 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 a good solution.

As some users felt that it was a bit cumbersome to navigate around, the camera controls was changed so that the keyboard is used for movement around the plane, and the mouse to rotate and pitch the camera. This change made the tool in line with most games as well as other software such as CAD programs, improving 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 higher above the ground. 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. This is helpful for visualising the current performance of the road network and to quickly identify problematic areas when viewing the system from a macro perspective.

5.5   Development Process

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

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

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

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

[39] "Edy's vehicle physics," https://assetstore.unity.com/packages/tools/physics/edy-s-vehicle-physics-403.

[40] "Git," https://git-scm.com/.

[41] "Github," https://github.com/.

[42] "Balsamiq wireframes," https://balsamiq.com/.

[43] 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

[44] "Trello," https://trello.com/.

[45] "Ptv visum," https://www.ptvgroup.com/se/loesningar/produkter/visum/.

A
Appendix 1

This is where we will place appendix 1

II

B
Appendix 2

This is where we will place appendix 2