Rijken Julian

Supervisor: Vanden Abeele Alex
Coach: Geeroms Kasper

Physics-Based Traffic Simulation for Games

Graduation Work 2024-2025

Digital Arts and Entertainment

Howest.be

Contents

Abstract

English

This paper investigates the implementation details for a physics-based traffic simulation to be used in a gaming environment, focusing on agent behavior and traffic rule adherence. Leveraging Unity as a testing platform, the research evaluates the impact of different path-following, inertia management, collision avoidance, and intersection management strategies. This shows how different techniques and variables play a role in the final traffic simulation.

A foundational understanding of vehicle dynamics is developed through the integration of a virtual Proportional-Integral-Derivative (PID) controller for path alignment, speed regulation, and obstacle avoidance. The study introduces innovative methodologies such as lookahead interpolation for smoother path adherence and combined speed limit / stop point methods for managing the agent's inertia.

This study aims to leave the reader with a complete understanding of the workings for a simple, effective physics-based traffic simulation capable of interacting with the player and environment.

Dutch

Deze scriptie onderzoekt de implementatiedetails van een op fysica gebaseerde verkeerssimulatie, ontwikkeld voor gebruik in een gamingomgeving, met een focus op het gedrag van agents en de naleving van verkeersregels. Door Unity als testplatform te gebruiken, wordt de invloed geëvalueerd van verschillende strategieën voor padvolging, snelheidsbeheer, botsingsvermijding en kruispuntbeheer. Het onderzoek belicht hoe diverse technieken en parameters bijdragen aan de algehele prestaties en realiteit van de verkeerssimulatie.

Een fundamenteel begrip van voertuigdynamiek wordt bereikt door de integratie van een virtuele Proportioneel-Integraal-Derivatief (PID)-regelmechanismen voor padvolging, snelheidsregeling en obstakelvermijding. Deze scriptie introduceert innovatieve benaderingen, zoals interpolatie, vooruitkijken voor soepelere padvolging en gecombineerde snelheidslimiet / stopmethoden voor effectief snelheidsbeheer van agents.

Het doel van deze studie is om de lezer een uitgebreid inzicht te bieden in het ontwerp en de functionaliteit van een eenvoudige, maar effectieve op fysica gebaseerde verkeerssimulatie, die dynamisch kan interageren met zowel spelers als de virtuele omgeving.

Keywords

Traffic Simulation, Autonomous Vehicles, Simulation Framework, Multi-Agent Systems, PID Control, Unity, Agent, Physics Simulation

Preface

Ten years ago, I discovered my passion for games and game development, a journey that has profoundly shaped my personal and professional pursuits. Since that time, I have been deeply immersed in both creating and playing games, with a particular fascination for open-world experiences. Among these, the Grand Theft Auto (GTA) series has stood out as a source of inspiration, especially for its sophisticated traffic simulations.

Games like the GTA series are remarkable for their ability to create dynamic, immersive worlds. Their traffic systems, in particular, contribute significantly to the realism and engagement of the player. These systems convincingly simulate the complexities of urban traffic, scaling seamlessly even as players soar above the cityscape in an aircraft. Such ingenuity captured my interest and fueled my curiosity about the underlying techniques that make these simulations possible.

Recently, while revisiting an older entry in the series, GTA IV, I found myself captivated by its traffic mechanics. Each glance at the bustling streets reminded me of the intricate systems working behind the scenes to make the world feel alive. This experience motivated me to embark on a personal project: creating my own version of a GTA-inspired game. The project progressed well, and I successfully developed a drivable car with realistic suspension physics. However, when it came time to implement a traffic system, I encountered a significant challenge. My research revealed a surprising lack of accessible resources on the topic. Most existing solutions were either overly simplistic, inflexible, or prohibitively expensive. This gap in available knowledge and tools further inspired me to develop my own traffic simulation system and document my findings in this paper.

Despite my extensive experience in game development, one area I had always avoided was artificial intelligence (AI). Creating non-player characters (NPCs) and enabling them to make autonomous decisions has always seemed daunting. Writing this paper offered me a unique opportunity to delve into a subject that had long been outside my comfort zone. I chose traffic

simulations as my focus, both to expand my understanding of AI and to address a tangible challenge in game development.

Initially, I envisioned this paper as a broad exploration of how to make in-game cities feel alive. However, after consulting with my supervisor, I realized that such a wide scope would be impractical, as it would encompass topics ranging from human agent behavior to artistic design considerations. Consequently, I narrowed my focus to comparing existing traffic simulation methodologies. Even this refined goal presented challenges, particularly due to the limited access to commercial simulation packages. As a practical developer, I felt compelled to create something of my own alongside the research.

Ultimately, I settled on building my own simulation and concentrating the paper on analyzing established formulas and methods for traffic simulation. To streamline the process, I started with an open-source package that provided a foundational framework, though with limited features. This approach allowed me to focus on extending and improving the system while conducting a comparative study of existing techniques.

This paper represents a deeply personal journey into a subject that has long intrigued me. It is my hope that this work contributes to the body of knowledge on traffic simulations and inspires others to explore this fascinating and publicly under-researched area of game development.

List Of Figures

Introduction

Traffic simulations have been integral to creating engaging and realistic game environments. From city-building games like Cities Skylines to open-world experiences such as the Grand Theft Auto series, these systems not only enhance immersion but also define core gameplay elements. Despite their significance, there is a noticeable lack of accessible resources detailing the methodologies for designing game-specific traffic simulations, particularly with a focus on physics-based interactions.

This paper investigates the implementation of a physics-based traffic simulation tailored for gaming environments. Unlike real-world traffic simulations, which prioritize accuracy for urban planning, traffic simulations in games must strike a balance between realism and computational efficiency, with the primary objective of allowing the player to physically interact with the other vehicles on the road, especially in the case of open-world games. The study aims to address this unique balance by exploring agent behaviors, traffic rule adherence, and the interplay of physics-based dynamics.

Through this work, the paper aims to contribute not only to the theoretical understanding of traffic simulations in gaming but also to the practical development of tools and frameworks that empower game developers to craft dynamic and immersive virtual worlds.

1. Research Question

This paper goes into detail to answer the following question:

“How are physics-based traffic simulations for games affected by agents and traffic rules?”

In order to show the effects of these rules, multiple tests will be conducted using different sets of rules, comparing and distinguishing the results using quantitative methods.

2. Hypothesis

Several hypotheses have been formulated to help answer the research question. This paper will use these to further refute or confirm the following assumptions.

H0 (Null):
“Agent and traffic rules have no visible impact on the simulated traffic”

H1 (Agent Decisions):
“Different agent decisions show a visible impact on the final simulation when it comes to speed and path deviation”

H2 (Collision Avoidance):
“Accurate methods for collision avoidance are vital for creating a physics-based traffic simulation”

H3 (Intersection Flow):
“Different types of intersections have an effect on traffic flow”

Literature Study / Theoretical Framework

1. Introduction to Traffic Simulations for Games

1.1 Historical Context and Importance in Gaming

Traffic simulations have long played a pivotal role in enhancing the realism and interactivity of virtual environments. Early examples can be traced back to city-building games such as SimCity (1989) [1], which introduced players to simplified traffic systems as part of urban planning mechanics. Games these days require increasingly sophisticated simulations tailored to their genre and objectives. For instance, Cities Skylines employs realistic agents with individual routines, making traffic flow and congestion a core challenge and narrative element—essentially the "main character" of the gameplay. In contrast, open-world games like Grand Theft Auto (GTA) treat traffic as a dynamic backdrop, designed to enrich the environment and provide reactive elements for player interaction, while keeping the player themselves as the central focus. This difference in how traffic is approached across genres is an observation I made and appears to be a novel perspective not previously described in existing literature, underscoring the diversity of traffic simulation roles in modern gaming.

1.2 Real-World vs. Game Traffic Simulations

Real-world traffic simulations focus on accuracy and precision, utilizing advanced computational models and data analysis, to optimize urban planning and infrastructure design. For example, tools like SUMO (Simulation of Urban Mobility) provide detailed simulations of traffic flow, road network design, and intersection behavior for real-world applications, enabling city planners to evaluate infrastructure efficiency and address congestion issues [2].

In contrast, game traffic simulations aim to balance realism with performance constraints and gameplay considerations. These simulations often employ simplified algorithms and heuristic approaches to create believable traffic systems that are computationally lightweight enough for real-time rendering. This is further discussed in the development diary #2: Traffic AI by Paradox [3]. Also, *Cities Skylines* uses procedural generation and agent-based modeling to simulate realistic traffic dynamics, making traffic flow a core gameplay challenge [1]. Similarly, games like *Grand Theft Auto V* prioritize immersive,

physics-driven vehicular interactions to enhance the player's experience within a dynamic, open-world environment [4] [5].

The divergence in methodologies reflects the differing objectives of these systems. Real-world models prioritize data-driven accuracy for optimizing urban mobility, while game simulations emphasize visual plausibility and interactivity. Frameworks like Unity's ITS package exemplify this balance, combining physics-based vehicle simulations with adaptive agent behaviors tailored to gaming environments [6].

1.3 Challenges in Current Traffic Simulation Methodologies

The development of effective traffic simulations for games presents several challenges:

Computational Efficiency

Ensuring real-time performance when simulating thousands of agents within a dynamic environment.

Realism vs. Gameplay

Striking a balance between realistic traffic behaviors and maintaining engaging gameplay.

Agent Decision-Making

Designing intelligent, adaptive non-playable characters (NPCs) that respond to a constantly changing environment.

Transparency and Accessibility

The limited availability of open-source frameworks specifically tailored to game development needs. Many existing frameworks are either proprietary, such as *Mobile Traffic System v2.0* by Gley for Unity or *Traffic Control System* by Overtorque Creations for Unreal Engine, or designed for real world use cases while running in game engines. Take for example SUMO (Simulation of Urban MObility) [2], an free and open-source traffic simulation software.

2. Foundational Theories

2.1 Path Finding Algorithms

Path following is crucial in traffic simulations, enabling agents to navigate predefined routes. Core algorithms like A* and Dijkstra's form the basis for pathfinding, while techniques like Bezier curves and spline models ensure smoother trajectories. In advanced simulations, such as in *Cities Skylines II*, pathfinding is not purely proximity-based but instead relies on a cost model that considers multiple factors like time, comfort, and financial costs. Agents dynamically adjust their paths based on real-time conditions, such as traffic jams, accidents, or roadblocks, optimizing routes accordingly [3]. This approach moves beyond basic pathfinding, incorporating road hierarchy and lane changes to improve traffic flow and responsiveness to unforeseen events.

2.2 Agent-Based Modeling and Behavior

Agent-based modeling [7] (ABM) treats each vehicle as an autonomous agent capable of individual decision-making. ABM frameworks simulate interactions among agents, incorporating factors such as speed, acceleration, and proximity to other vehicles. Behavioral models, including rule-based systems and finite-state machines, guide agent actions based on environmental conditions.

2.3 Game Theory in Traffic Simulations

Game theory [8] provides a framework for understanding the strategic interactions between agents, offering valuable insights into cooperation and competition within traffic systems. Concepts such as Nash equilibrium and payoff matrices are employed to model behaviors such as merging, yielding, and conflict resolution at intersections, ultimately contributing to more realistic and coordinated interactions among agents. These approaches are discussed in detail in [9], [10], [6], and [11].

3. Relevant Frameworks and Case Studies

3.1 Open-World Traffic Simulations

Games such as *Grand Theft Auto* (GTA) and Cities Skylines showcase diverse approaches to traffic simulation. While GTA emphasizes immersive, physics-based vehicular dynamics, *Cities Skylines* [12] focuses on systemic traffic flow management and urban planning. The distinction between physics-based simulations, which add realism to vehicular interactions, and more simplified, attached models that reduce computational effort is important here. As discussed in “4.2 Physics-Based and Attached Models” the hybrid approach to traffic simulation in these games blends elements of both methods to achieve a balance between realism and computational efficiency.

3.2 Intersection Design and Traffic Flow Analysis

Intersection behavior is critical in both real-world and game traffic systems. Studies on uncontrolled, stop-controlled, yield-controlled, and signalized intersections provide a basis for modeling decision-making processes and optimizing flow. For instance, Champion [9] discuss coordination mechanisms in traffic simulations applicable to uncontrolled intersections, while Cortés-Berrueco [10] analyze stop-controlled intersections using game theory. Lahdenranta [6] provides insights into behavior-based models for yield-controlled intersections, and Mandiau [11] explore decision-making processes at signalized intersections.

4. Overview of Tools and Techniques

4.1 FOSS Solutions and Proprietary Systems

Free and open-source software (FOSS) [13] offers accessible and customizable tools for traffic simulation. However, these solutions often lack specialized features tailored for gaming applications. For instance, SUMO (Simulation of Urban Mobility) [2] is a prominent FOSS tool used for real-world traffic simulations, but it may not fully meet the specific needs of game developers.

On the other hand, proprietary systems provide advanced capabilities and comprehensive support, albeit at a higher cost. These systems are often more feature-rich and user-friendly, making them attractive to developers who require robust and reliable solutions. Examples include various traffic simulation packages available on the Unity Asset Store and Epic Games' Fab

platform. However, the high cost and licensing restrictions of proprietary systems can limit their adoption, especially among independent developers. Consider, for instance, the Intelligent Traffic System (iTS), which is utilized by Lahdenranta in their study [6].

Epic Games' ZoneGraphs [14] is an example of a proprietary system that offers sophisticated traffic simulation capabilities. ZoneGraphs allows for the creation of detailed and dynamic traffic systems within the Unreal Engine, providing a high level of control and realism. This system is particularly useful for developers looking to implement complex traffic behaviors and interactions in their games.

4.2 Physics-Based and Attached Models

Physics-based simulations significantly enhance realism in physical player interactions, making them particularly suitable for applications such as Grand Theft Auto (GTA) or Euro Truck Simulator, where the player assumes the role of a driver. In contrast, attached models, which keep agents attached to the road network at all times, require less computational effort because they do not simulate the physical interactions of each agent. These agents simply follow the road network in a predefined manner, but this approach lacks the same level of realism in terms of physical interaction, for example, with a player-controlled vehicle, as seen in Cities Skylines [3].

The distinction between physics-based and simplified approaches is not always rigid. Many systems adopt a hybrid approach. For instance, GTA employs physics-based vehicle simulations to enable dynamic interactions, such as collisions involving the player. However, when the player is flying at high altitudes, detailed simulations of individual vehicles are unnecessary, and computational resources are likely optimized by reducing simulation fidelity. Conversely, in *Cities Skylines*, vehicles typically follow abstracted, path-based rules constrained to the road network. In *Cities Skylines II*, vehicles may switch to basic physics simulations under specific conditions, such as during incidents, introducing a layer of emergent behavior, as described in the "Traffic Accidents" section of the Development Diary #2: Traffic AI [12].

Notably, the nuanced interplay between physics-based and simplified simulations has received limited attention in the existing literature. This observation is based on my personal experience, having extensively played the GTA series for over 3,000 hours, coupled with an analysis of how these systems adapt to player proximity and interaction.

4.3 Existing Studies on Traffic Dynamics

The study of traffic dynamics has been a subject of research for several decades, with various approaches to modeling and simulating vehicular movement and interactions. Existing studies highlight the use of multi-agent systems, game theory, and decision-making models to improve traffic simulation and management.

Multi-Agent Systems and Game Theory in Traffic Simulation
One prominent approach to modeling traffic dynamics is through the use of multi-agent systems, as explored by Champion in [9]. Their work investigates coordination mechanisms among agents using game theory to optimize traffic flow and reduce congestion. Similarly, Cortés-Berrueco [10] discuss the application of traffic games to model freeway dynamics, where vehicles act as autonomous agents making strategic decisions to improve overall efficiency.

Decision-Making and Behavioral Models
Decision-making is another critical aspect of traffic simulation. Lahdenranta [6] provides insights into behavior-based models used in intersections, emphasizing the role of artificial intelligence (AI) in urban traffic simulation. Mandiau [11] extend this idea by presenting a behavior-based coordination method based on decision matrices for agents within an urban traffic environment. These studies illustrate the application of AI to facilitate effective traffic management and decision-making among vehicles.

Wikipedia and Media Perspectives
Additionally, platforms like Wikipedia provide general overviews of AI in video games [4] and traffic simulation [5], which serve as accessible resources to understand broader concepts in the field. Meanwhile, popular media such as YouTube videos [15] and [16] offer content that discusses real-world traffic issues, phantom traffic jams, and the functioning of traffic signals, contributing to public awareness and interest in traffic dynamics.

In summary, the existing body of research on traffic dynamics combines theoretical models using multi-agent systems and game theory with practical implementations in gaming and simulation environments. These efforts continue to evolve as they integrate real-time computational solutions and user interactions to better understand and manage urban mobility.

Case Study

1. Introduction

This case study will focus on the development of a physics-based traffic simulation tailored specifically for use in games. Traffic simulations encompass a broad range of concepts, with numerous nuanced decisions shaping their design and implementation. Given that the simulation is created for a gaming environment, Unity has been selected as the testing platform. The primary reason for Unity is the engine's component-based architecture which is preferred to separate the simulation into different parts, over a more inheritance-based system like Unreal Engine. Unity also supports in-engine recording, which is great for experiments and also has by far the largest user base allowing most developers to understand the engine specific concepts explained in this study.

Traffic simulation is a very broad topic. This paper specifically focuses on the agent and traffic rules with the focus on agent path following and intersections. The case study will explore in depth the decisions an agent has to make to stay on a path with the least deviation. Agents must also communicate with each other to avoid collisions, especially when it comes to intersections.

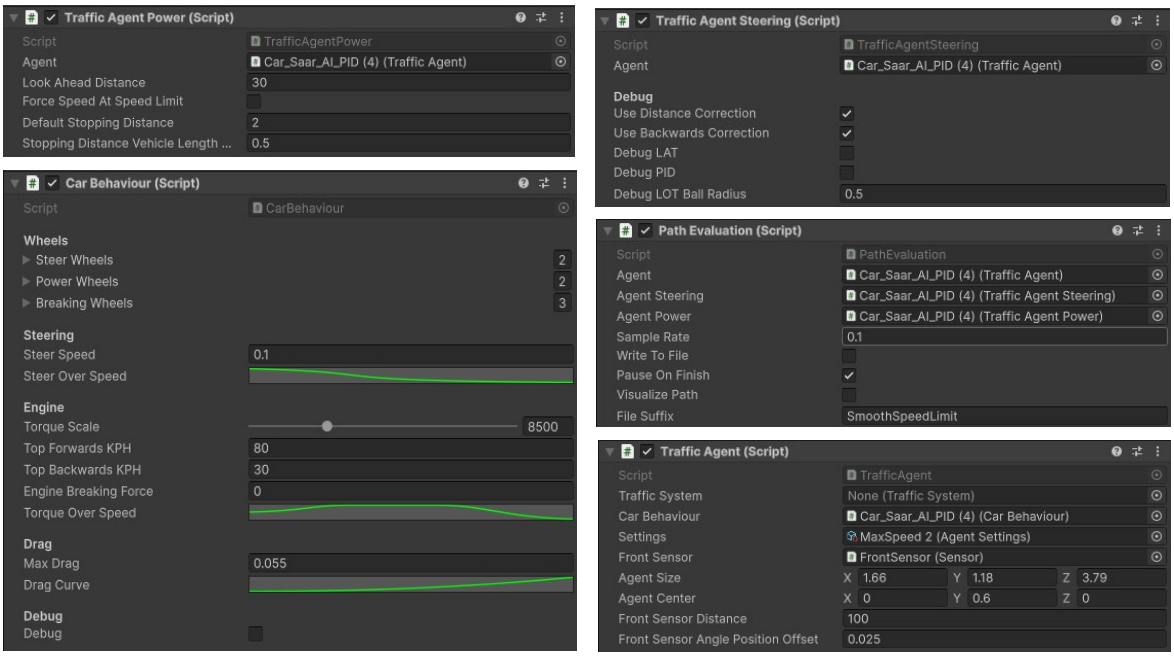


Figure 1. Example of how the traffic system is created by separating it into smaller components.

2 Road Network

In any traffic simulation, the road network is a key element. Roads must be modeled in such a way they can represent various types of infrastructure, such as highways, intersections, and residential streets.

2.1 Existing Systems

Firstly, I would like to highlight a few examples of how other systems define their road network.

Unreal Engine Zone Graphs [14] [17]

Different simulations have various implementations. For instance, Unreal Engine makes use of Zone Graphs [14] to create the City Sample Project for Unreal Engine [18]

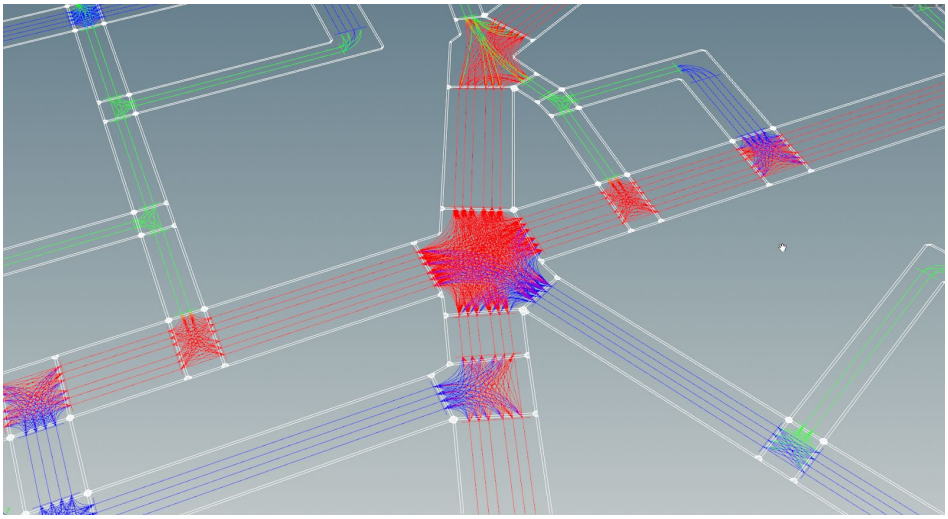


Figure 2: A top-down preview of Unreal Engine Zone Graphs with colors indicating road size.

iTS

Karl Lahdenranta describes an intelligent traffic system (iTS) [19]. The system stores lane and connector data in the so called "iTS Main Manager" script, which handles information such as speed limits, lane density, width, and vehicle restrictions. Lanes are represented as lists of points and are designed. Connectors help link lanes and enable overtaking.

SUMO (Simulation of Urban Mobility) [2]

Road network information is stored in an XML-based file format called a network file. This file typically has a .net.xml extension and represents all relevant aspects of the road network, including roads, intersections, lanes, traffic signals, and connections between these elements.

The network itself sill comes down to a list of nodes with a lot of specific information, more can be found about SUMO's road network system on their website "SUMO docs – PlainXML" [20].

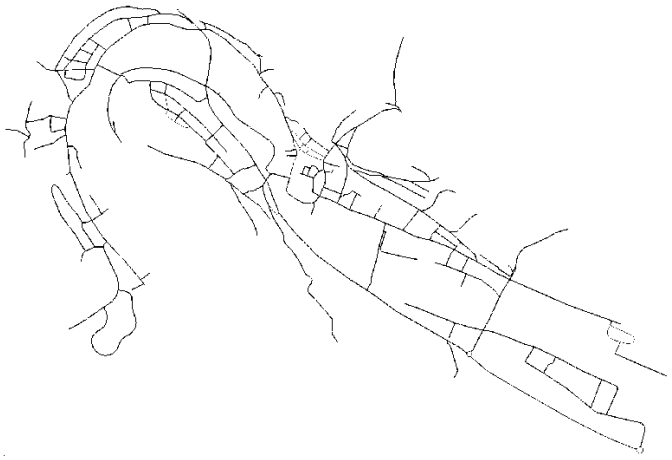


Figure 3. net file opened in sumo-gui. It shows the map of the German city Eichstätt. Sources from SUMO documentation [21].

Clara [22]

Clara, another opensource solution grounded in Unreal Engine, which is used for (e.g. learning driving policies, training perception algorithms, etc.). The road network in Clara is defined using ASAM OpenDRIVE® [23]. Like SUMO this uses the xml format to store road information. The system is once again devised of nodes, links and roads.

2.2 Proposed Methodology

The systems discussed above demonstrate various approaches to defining road networks in traffic simulations. When we inspect them, they all come down to a network of nodes that are connected in various ways. They are surrounded and grouped by connections and roads. To conduct the tests in this paper, a simplified method is employed. The primary objective is to examine the differences between agent behaviors and adherence to traffic rules. To achieve this, the road network is constructed using a straightforward structure based on segments and nodes.

Components of the Simplified Network:

1. **Nodes:**
   Nodes represent the fundamental points in the network. Each agent navigates from one node to the next, forming the basis for path-following behavior. These nodes act as waypoints that guide vehicle movement.

2. **Segments:**
   Segments are defined as a series of connected nodes. Each segment corresponds to a distinct road or pathway. This segmentation makes it easier to differentiate between various road types and facilitates the modular design of the network.

3. **Connections:**
   Connections define the relationship and direction between segments. They allow segments to join, split, or transition seamlessly, providing the flexibility needed to model intersections, merges, and other complex road structures.

By using this simplified method, the paper emphasizes the core mechanics of traffic simulation while providing a customizable framework for testing agent behaviors. However, the manual nature of this approach highlights a trade-off between simplicity and efficiency when compared to automated systems like Zone Graphs.

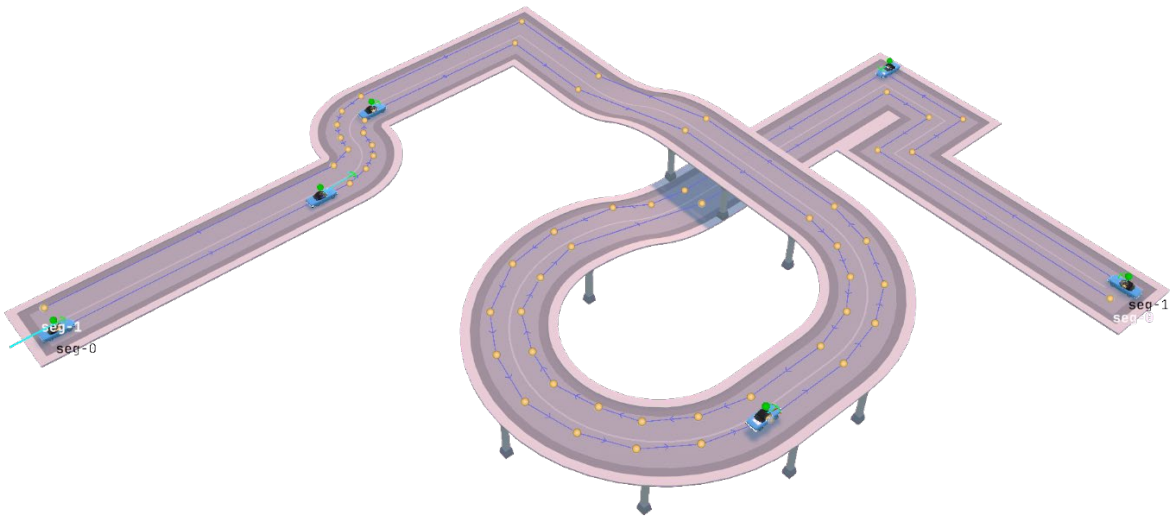


Figure 4: An example of the nodes laid out over simple road tile models by Kenney [24]. These two segments, going in opposite directions, are used as a test ground for the path following part of this case study.

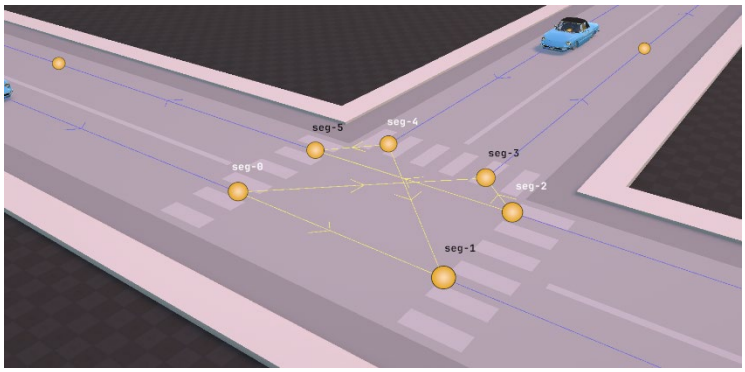


Figure 5. Illustration of the connectivity between different segments creating an intersection. The endpoint of each segment is denoted by white text, while the starting point is indicated by black text. Segments are connected by a yellow line.

As a quality-of-life feature, the road network in Unity that was developed includes a functionality to restructure the network with the click of a button. This automatically updates all waypoint connections and adjusts their names, accordingly, significantly reducing manual labor. Additionally, I incorporated a feature that allows all waypoints to be snapped to the floor with a slight offset. This is a crucial element, as vehicles must be positioned on the path. Since the path can also change in height, it is important that the heights of both the vehicle’s body and the waypoints are aligned.

While Unity offers splines through a package, I have chosen not to use this feature in this instance, as my focus is on the agents rather than the road network itself. However, the spline package would be ideal for future work, particularly in creating a smoother and more refined road network.

This approach prioritizes flexibility and customization but comes at the cost of increased manual effort compared to more automated systems, such as Unreal Engine's Zone Graphs. By using this method, we ensure a controlled environment that facilitates a deeper exploration of agent dynamics and rule compliance. The system is not stored in a specific file format and a possibility for future work would be saving the road network in a specific file format

3 Physics-Based Testing

In Section “4.2 Physics-Based and Attached Models” I discussed the distinctions between physics-based and attached modes, emphasizing that the choice between them is not always clear-cut. While both models have their respective advantages, this paper focuses on agent and traffic decision-making and does not delve deeply into the impact of using one model over the other. In general, many rules can be implemented through different methods, yielding similar results.

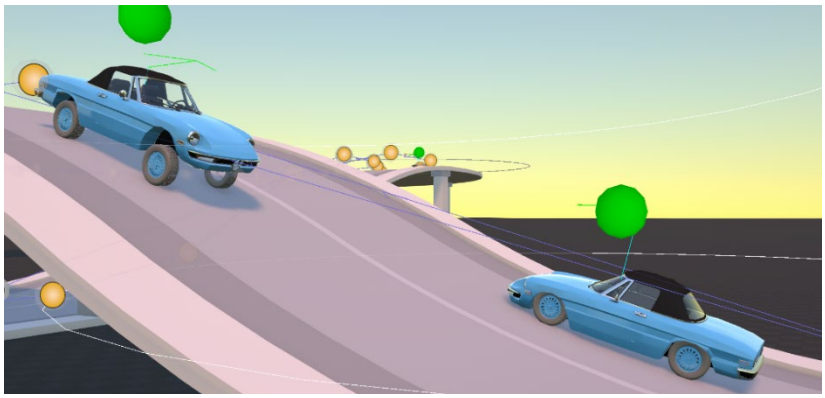


Figure 6. Example of an agent driving up and down a slope, demonstrating the use of a physics-based vehicle model and the suspension system in action.

For the purposes of this research, and in alignment with my interest in open-world games as discussed in the Preface chapter, will adopt a physics-based system. This entails that the vehicle is fully simulated at all times, which allows for more realistic interactions between the player-controlled vehicle and AI vehicles within the road network. Rather than putting the vehicle on tracks, as shown in “Development Diary #2: Traffic AI” [3], we instead interface with our physics based agent via the following input parameters:

- **SteerWheelInput**: Ranges from (-1.0, 1.0), controlling the steering of the vehicle. *(Acts as the steering wheel)*
- **ThrottleInput**: Ranges from (0.0, 1.0), regulating the vehicle's acceleration. *(Acts as the throttle paddle)*
- **BrakeInput**: Ranges from (0.0, 1.0), determining the vehicle's deceleration. *(Acts as the brake paddle)*

These parameters are managed by a car controller class, which oversees the vehicle’s inputs. The AI, which operates separately from the controller, adjusts these input values within predefined limits, ensuring that the code remains modular and adaptable. This separation of concerns facilitates the development of a more generic system that can easily be expanded or modified.

4. Agent Steering

In the context of traffic simulation for games, path following is a foundational concept. Whether for a simple car navigating a road, or complex interactions between thousands of vehicles, every traffic simulation must address how vehicles follow designated paths. The challenge lies not only in the pathfinding process itself but also in the real-time decision-making that agents must make based on traffic conditions, obstacles, and environmental factors.

4.1 Path Alignment Using PID

As discussed in 2.2, “Physics-Based testing” we will make use of a physics based system which, unlike in an attached system, does not feature perfect alignment as we do not simply move the agent along a spline [25]. This creates flexibility in vehicle movement and requires the system to replicate agent steering behavior as seen in the "Development Diary #2: Traffic AI," [3].

Our simulated vehicle has many different parameters acting upon it. For instance, the steering is not instant as the steering wheel takes time to rotate toward the desired rotation, together with additional factors like the car sliding or going off course because of other physical objects. This makes our situation quite difficult, and we therefore require a solution to keep the car on track from any given position.

The challenge we face when using a physics-based agent is keeping the agent as close to the path as possible just by changing the inputs of the vehicle. Based on secondary research, there is little to be found about this topic when looking for solutions created for games. However, this concept is well researched outside of traffic simulations based on autonomous vehicles.

This brings us to **PID-Control** [26]. The Proportional-Integral-Derivative (PID) controller is a widely used feedback control system that aims to continuously adjust the inputs of a system in order to reach and maintain a desired target. In the context of a traffic simulation, specifically for path following, PID-control can be used to keep an agent (vehicle) on the path despite the complexities of real-world physics, such as road curvature, obstacles, and the natural delay in vehicle response.

When applied to a physics-based vehicle model in a traffic simulation, PID-Control works as follows:

- **Error**: The perpendicular distance the agent is from the path.
- **Accumulated Error:** The accumulated error over time used for I.

- **P - Proportional**: Acts as a **spring**, pulling the car back to the path.
- **I - Integral**: Acts as a **memory**, reducing the steady state error.
- **D - Derivative**: Acts as a **damper**, reducing the oscillation of the car.

The following results will highlight how the Proportional and Derivative part of the formula affects the trajectory of the agent.

P (Proportional) & D (Derivative) - Control Results

using 10 m/s max speed, with a non-continuous two-part road

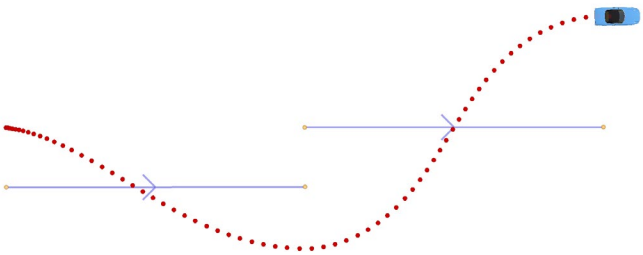


Figure 7. Only **P** with **low** proportional gain.
P = **0.2**
D = 0.0
I = 0.0

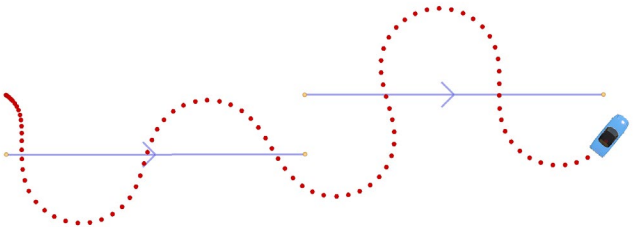


Figure 8. Only **P** with **medium** proportional gain
P = **0.7**
D = 0.0
I = 0.0

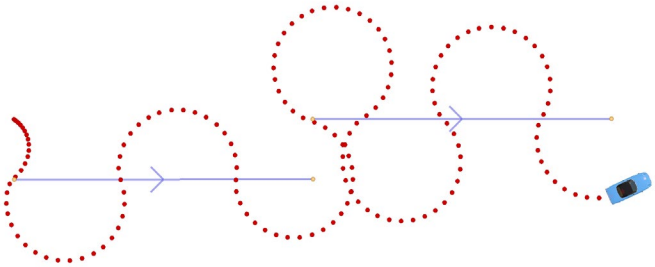


Figure 9. Only **P** with **high** proportional gain
P = **2.0**
D = 0.0
I = 0.0

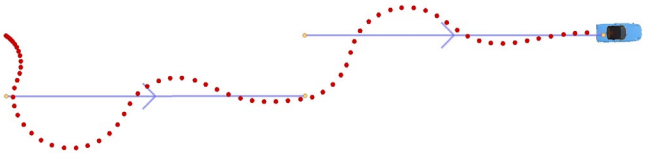


Figure 10. **P** and **D** with **low** derivative gain.
P = 0.2
D = **0.05**
I = 0.0

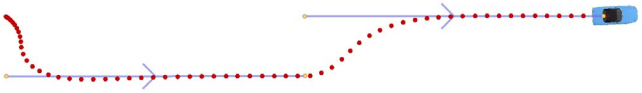


Figure 11. **P** and **D** with **ideal** derivative gain
P = 0.7
D = **0.3**
I = 0.0

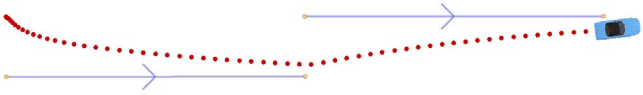


Figure 12. **P** and **D** with **high** derivative gain
P = 2.0
D = **2.0**
I = 0.0

These tests demonstrate that finding the appropriate PID values is crucial for minimizing an agent's deviation while driving. Each of these tests adjusts the steering value, as described in section 2.2 “Physics-Based Testing." When we modify the proportional (P), integral (I), or derivative (D) gain, we essentially multiply the steering value by the corresponding gain and use it as the vehicle's steering input. It is important to note that the error value is calculated based on the perpendicular distance to the path, meaning these values are effective only in a world with realistic units. Similarly, the steering value is adjusted between the range of [-1,1], and the AI controller does not require any information about the vehicle’s steering angle. However, such information could be important when dealing with different types of vehicles. For future changes to the system, PID should ideally be calculated from the steering angle and not the steering alpha [-1,1].

Now that we have a clear understanding of how to steer the car, we are left to consider the role of the **I (Integral)** component in the equation. In brief, this component addresses past accumulated errors to eliminate steady-state offsets. If the agent is influenced by external factors, such as being pushed by another vehicle or having a broken wheel that causes it to tilt left, the Integral component accumulates the error over time. This enables the agent to steer back toward the correct path.

In order to showcase the effect of the accumulated error the back wheel has been broken of this poor little agent. It can now serve as the perfect example for our testing.

Figure 13. Example of an agent with a broken back wheel forcing the agent to deviate from the path.

I (Integral) - Control Results

using 10m/s max speed & broken wheels

Figure 14. Showing **I** with **no** integral gain.
P = 0.7
D = 0.3
I = 0.0

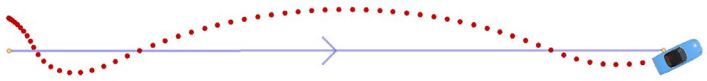


Figure 15. Showing **I** with **high** integral gain.
P = 0.7
D = 0.3
I = 2.0

Figure 16. Showing **I** with **ideal** integral gain.
P = 0.7
D = 0.3
I = 2.0

The vehicle simultaneously exhibits advanced, physics-based behavior with distinct characteristics, such as a steering curve that dynamically adjusts with speed and a gradual response when accelerating or decelerating. These behaviors, while sophisticated, can introduce challenges in maintaining stability, which the integral component of our PID formula effectively addresses by compensating for accumulated errors over time.

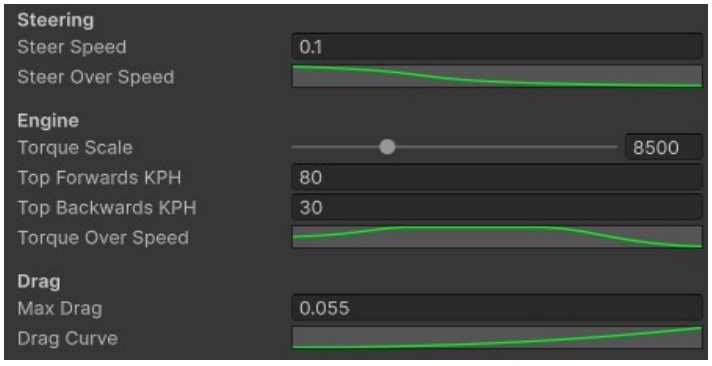


Figure 17. Screenshot from the physics agent behavior component. Example of how the agent has advanced parameters that might make it behave in a difficult to predict fashion.

4.2 Path Alignment Correction

A significant limitation of the system is that the agent's steering decisions are based solely on which side of the path it is positioned, without considering its current orientation. As a result, the agent often takes an unnecessarily long route to re-align with the path.

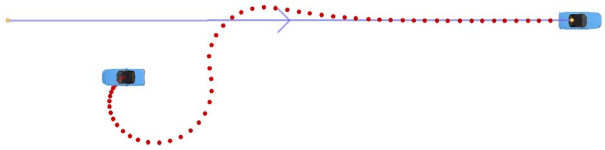


Figure 18. **Showing Issue.** Illustrating how the agent follows a longer path due to its inability to account for its own orientation.

Another notable issue arises when the agent strays significantly from the designated path. The instability occurs due to the behavior of the PID-Control system. Typically, a PID controller operates on a direct error value, which drives the system toward the target or causes overshooting depending on the control strength. However, in this scenario, the agent's continuous steering actions cause the error value to increase even as the PID controller applies corrective inputs. This dynamic leads to further instability instead of reducing the error. This is also briefly touched upon in a video by AerospaceControlsLab at 1:29, but with no further solution. [27]

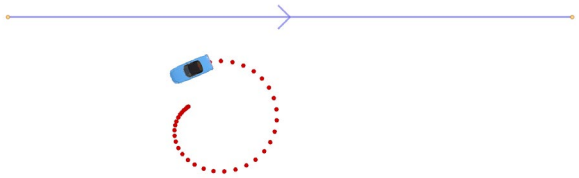


Figure 19. **Showing Issue.** Shows how the agent becomes unstable when it deviates too far from the path.

This issue is likely because in most cases, just a PID controller is good enough. However, the simulation developed for this paper aims to have the agent recover itself from any given position and therefor introduces the following error corrections:

- **Backwards Correction**. The correction applied to the agent that overrides the PID controller when the agent's orientation is > 90° in either direction from the segment direction.

- **Instability Correction**. The correction that is applied when the agent is more than a set distance away from the path. Instead of PID the system simply steers the agent in the direction of the sample position.

Backwards Correction Results

using 10m/s max speed - 5.0 Direction Error Gain

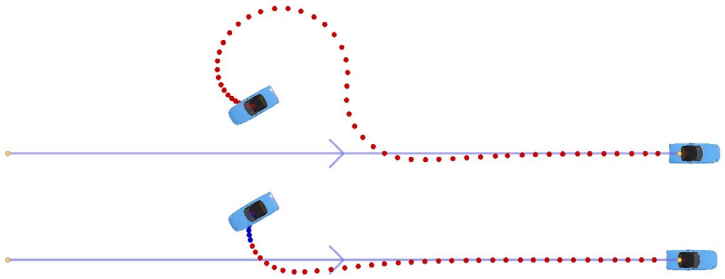


Figure 20. **Showing Without & With Backwards Correction.** Shows how the agent switches between Backwards Correction and PID.

The backward correction operates by calculating the angle between the path direction and the agent's forward direction. It is important to note that this method does not use PID control, as the steering angle is not a continuous error value like the agent's distance from the path. Consequently, only the proportional (P) component of the equation is applied, scaling the angle by dividing it by 360 and multiplying by a directional error gain.

At first glance, this may appear to be a simpler solution than PID. However, it merely adjusts the vehicle's direction. When following a path, both the direction and position of the vehicle must be updated simultaneously to ensure accurate path adherence.

Instability Correction Results

using 10m/s max speed - 5.0 Direction Error Gain

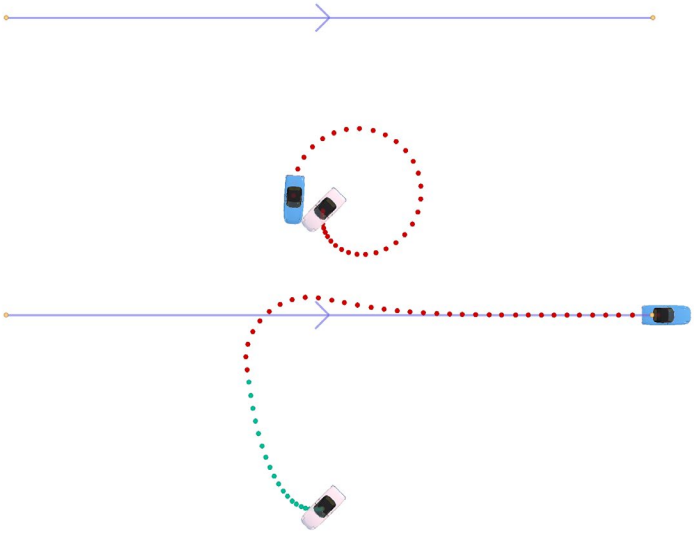


Figure 21. **Showing Without Instability Correction.** Shows how the agent becomes unstable when it deviates too far from the path. Similar to fig 17.

Figure 22. **Showing With Instability Correction.** Shows how the agent starts outside the IC distance of **5 meters**. And switches to PID (Shown in red).

The results of the instability control demonstrate how the agent can correct itself and realign with the path before becoming unstable. The following formula is employed to steer toward a direction, where d serves as the error direction gain.

$$s = \frac{\arctan 2\left(v_{1x}v_{2y} - v_{1y}v_{2x}, v_{1x}v_{2x} + v_{1y}v_{2y}\right)}{\pi} \cdot d$$

Figure 23. Formula used to calculate the steering input.

The same formula is utilized for both **instability** and **backward** correction. Notably, this approach is applied exclusively in 2D, as steering the agent in the vertical direction is not feasible. In the implementation, the process is further simplified by using Unity's `Vector2.SignedAngle()` method and dividing the result by `360.0f`. This method does not rely on `arctan2` and is used purely as a matter of preference.

```
private float GetSteerInputToDirection(Vector3 direction) {
   Vector2 directionToPath = new Vector2(direction.x, direction.z).normalized;
   Vector2 agentDirection = new Vector2(_agent.Forward.x,
_agent.Forward.z).normalized;
   float angleError = Vector2.SignedAngle(directionToPath, agentDirection) / 360.0f;
   return angleError * _settings.DirectionError_Gain;
}
```

Figure 24. Formula used of directional steering implemented in C# using Unity

```
var sample = _segment.GetSampleFromPosition(_agent.Position);
float error = sample.SignedDistanceFromPath;

float p = _settings.Proportional_Gain * error;

float errorRate = Vector3.Dot(_agent.Velocity, sample.DirectionRight);
float d = _settings.Derivative_Gain * errorRate;

_accumulatedError += error * Time.deltaTime;
_accumulatedError = Mathf.Clamp(_accumulatedError, -_settings.Integral_Limit,
_settings.Integral_Limit);
float i = _settings.Integral_Gain * _accumulatedError;

_agent.SteerWheelInput = p + i + d;
```

Figure 25. Snippet of PID implementation for agent in C# using Unity
A formula for PID can be found in the appendix.

This concludes the basic steering aspect of this case study. In summary, PID-Control is highly effective for guiding dynamic variables toward a target, akin to steering a car along a desired path. However, certain modifications are necessary to enable the agent to recover even when it deviates significantly from the path. To address this, I introduced instability and backward correction using a simple formula that relies solely on the proportional component and steers based on direction rather than strictly following a path. This combination results in a system capable of robust path-following.

That said, this does not encompass advanced concepts such as corner cutting or lookahead strategies for minimizing path deviation, which remain areas for further exploration.

4.3 Path Deviation Optimization and Smoothing

With the agent now capable of following a path and aligning itself to designated points along it, the next objective is to evaluate its performance in navigating more complex paths. The primary aim is to minimize the agent's deviation from the path, defined as the perpendicular distance between the agent and the path at any given point.

In scenarios involving sharp corners, the agent consistently overshoots these turns due to its inability to anticipate upcoming changes in the path's trajectory. This limitation stems from the fact that the agent's current steering calculations only consider its immediate position relative to the path, without accounting for the curvature or direction of the path ahead.

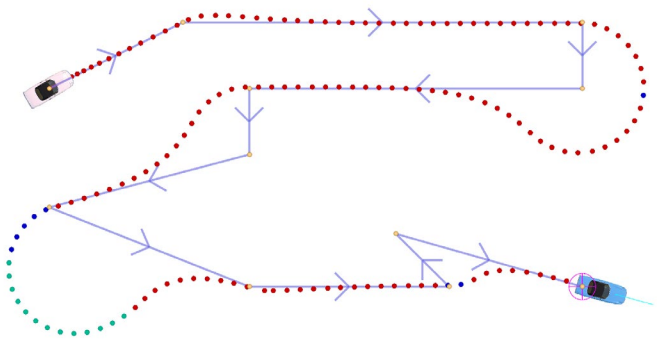


Figure 26.
Visualization of agent following the path and **overshooting the corners.** Notably the agent skips the last corner as he simply overshoots on to another part of the path.

One solution to this issue is the use of a technique called lookahead. Lookahead takes many forms and similar to PID has many different use cases. When we talk about lookahead we practically talk about calculating the future value based on its change over time.

$$L = v \cdot \Delta t$$

Figure 27. Formula for calculating lookahead

Lookahead is used in many other scenarios, for instance moving a camera to follow a player character. Many side scrollers incorporate this concept to allow the player to look a bit ahead of where the character is moving. In our case we use it to sample a point further along our path. Our agent has access to the current distance along the path, so in order to get the position along the path in the future we simply calculate the lookahead value based on the vehicle's forward speed multiplied by the lookahead time.

```
float LookaheadDistance = _agent.CarBehaviour.ForwardSpeed *
_agent.Settings.LookaheadTime;
_futureSample =
   _agent.CurrentSegment.SampleFromDistance(_agent.CurrentSample.DistanceAlongSegment
+ LookaheadDistance);
```

Figure 28. Snippet of LAT implementation for agent in C# using Unity

Now that we have a future sample, we can steer the agent based on this future position rather than the current one. However, this does not fully resolve our smoothing challenges. While the agent begins aligning with the path ahead of time, it produces the unintended effect of deviating from the path before reaching the corner.

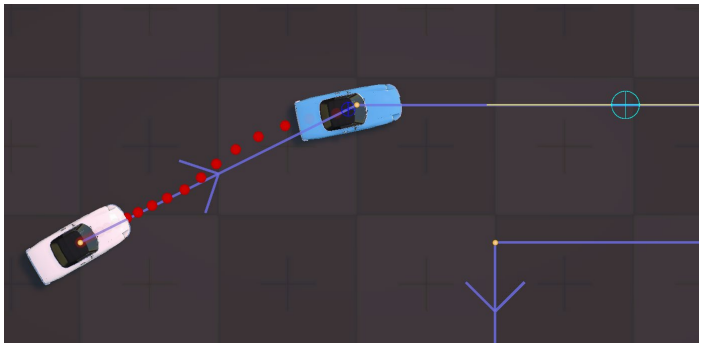


Figure 29. Unwanted deviation

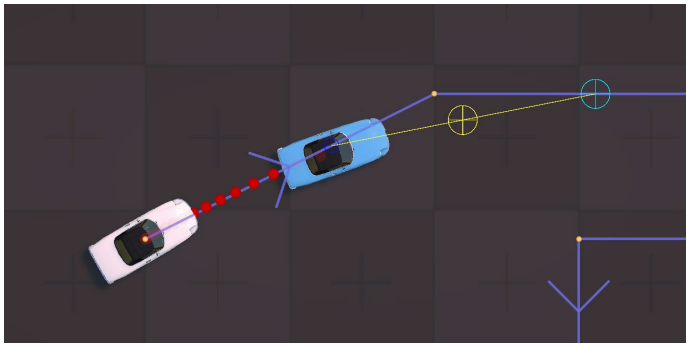


Figure 30. Interpolating solution

4.4 Proposed Methodology for Path Smoothing

Current to Future Sample interpolation.
In order to properly smooth out the motion with maximum control we will take a page out of splines.

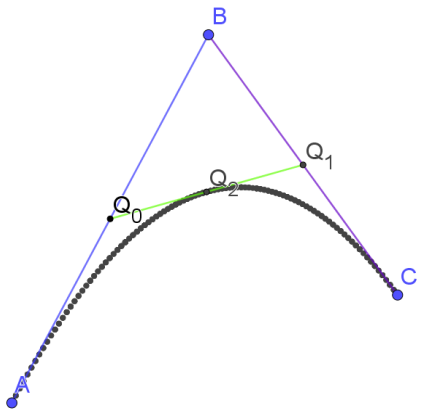


Figure 31.
Showing **quadratic** Bezier curve where t = 0.5, visualized in GeoGebra.

A quadratic Bezier curve is a parametric curve defined by three control points: **A**, **B**, and **C**. The curve is generated through a process of linear interpolation applied recursively between these points, parameterized by a value **t**, where **t** ∈ [0,1]. The interpolation process ensures smooth transitions along the curve as **t** progresses.

The way we smooth our path is very similar to the way quadratic Bezier curves work with one important distinction:

Unlike traditional Bezier curves, we do not require interpolation between the control points (A, B, and C). Instead, we define **Q0** as the current path sample of the agent and **Q1** as the future sample selected based on the lookahead time. We then introduce another sample, which we refer to as the interpolated sample. This sample represents the direction from **Q0** to **Q1**, with its position always centered between **Q0** and **Q1**. This effectively creates a continuous curve over the path, allowing the agent to smoothly navigate corners.

This approach shares similarities with splines, and I highly recommend watching "The Continuity of Splines" by F. Holmér [25], which inspired much of the methodology of this smoothing technique, for the interpolating between points similar to how Bezier curves work and are explained in the video.

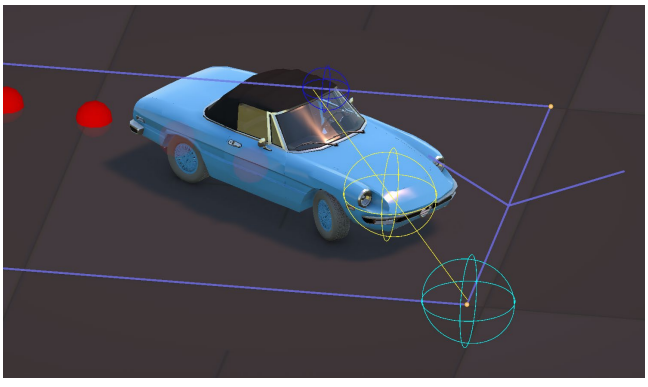


Figure 32.
Visualization of the agent, with the current sample represented by the small dark blue ball, the yellow ball indicating the interpolated sample, and the cyan ball representing the future sample. The agent is shown cutting the corner as it follows the interpolated sample.

Deviation Optimization Interpolation Results

deviation measured as perpendicular distance to path over distance along the path. using 10m/s max speed

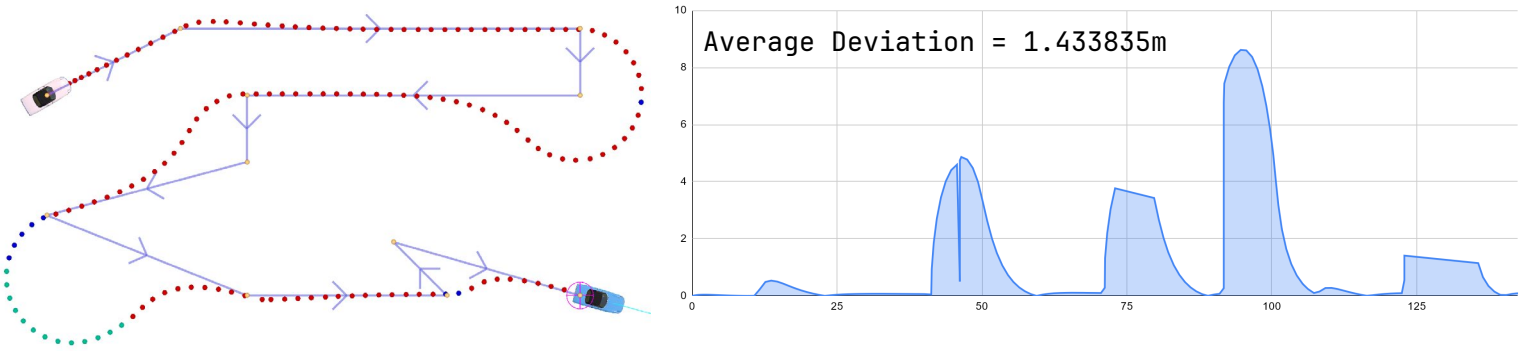


Figure 33. **Showing 0.0 LAT** with very direct connection to the path but very abrupt deviations.

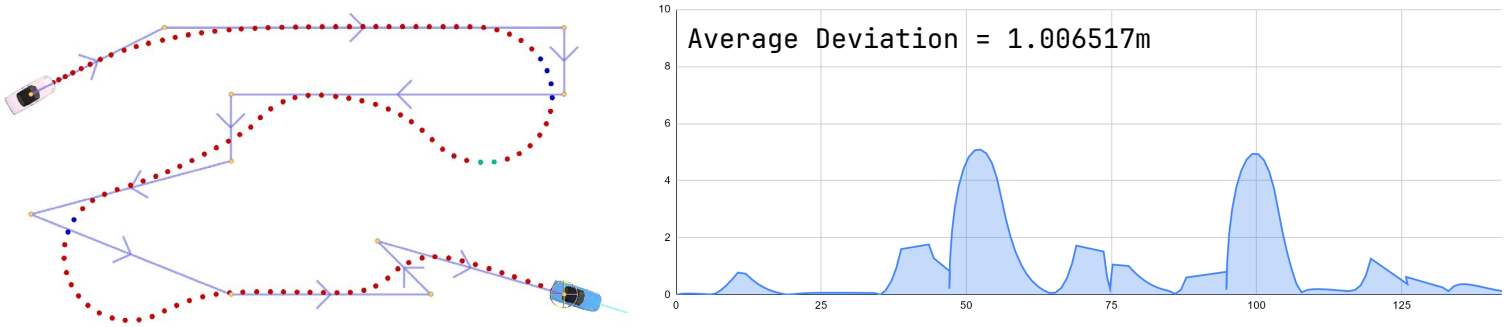


Figure 34. **Showing 0.75 LAT** allows for some smoothing between the corners. More deviation is created but with an overall smoother result.

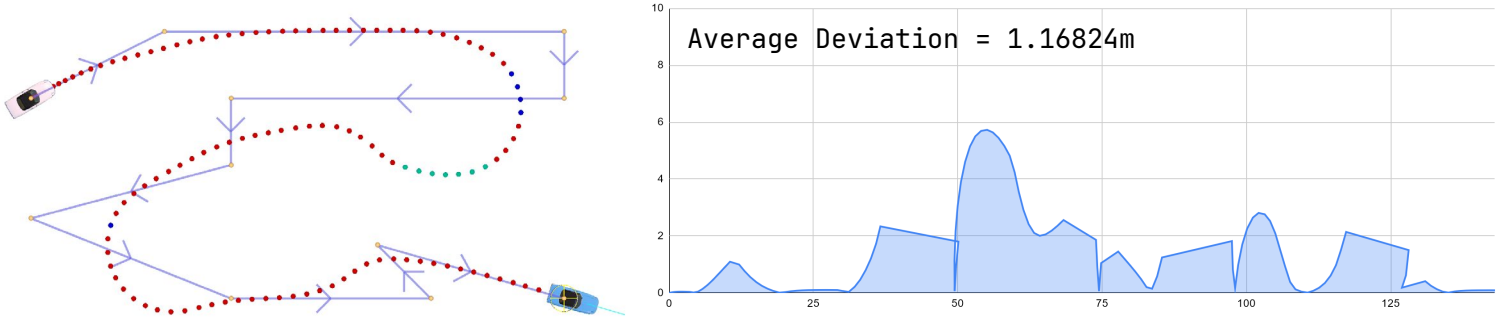


Figure 35. **Showing 1.25 LAT** creating a very smooth path but. At the cost of accuracy in return creating even more deviation.

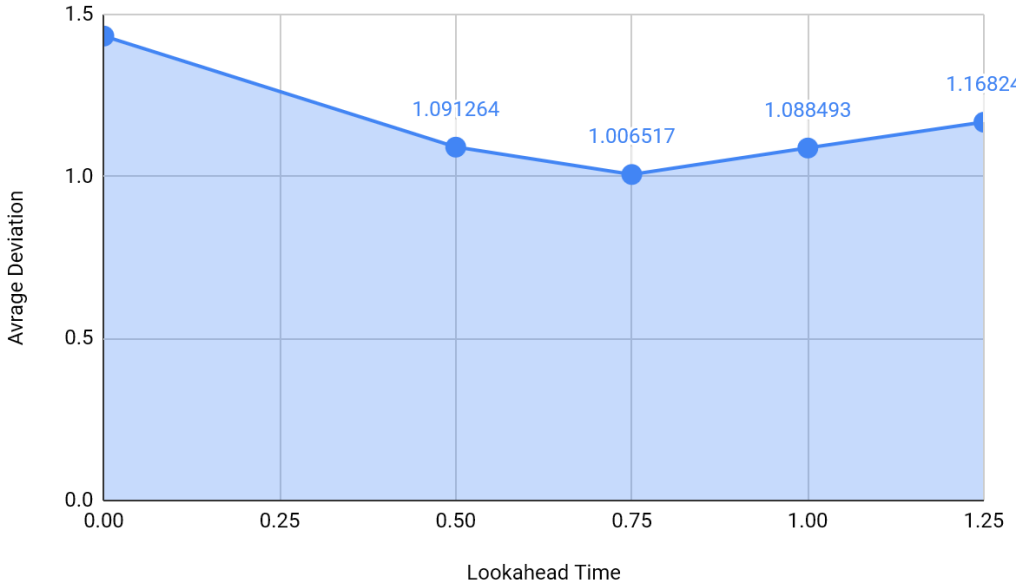


Figure 36. Visualization of **Average Deviation** over **Lookahead Time** showing how balancing the 2 values is important for minimal deviation.

The results demonstrate a clear improvement in path deviation. It is important to highlight, however, that path deviation graphs are not the sole criterion of interest. Our objective is not merely to minimize deviation (although this serves as a useful indicator), but rather to ensure that the vehicle exhibits realistic driving behaviors. This is why smoothing plays a crucial role, as most drivers tend to make minor adjustments when possible.

There is also an argument to be made for eliminating smoothing altogether, on the grounds that the path itself should be smoothed rather than the agent. Nonetheless, I conclude that a combination of both approaches is more ideal and flexible, as it enables the agent to exhibit more realistic behavior within a less precisely defined road network.

This concludes the discussion on path deviation optimization with constant speed. The subsequent chapter will focus on speed management, which also plays a significant role in maintaining the agent's trajectory. It is also crucial that the agent is capable of steering optimally even without constantly reducing speed. While additional optimizations are possible, such as allowing the agent to deviate from the path and counter-steer to improve overall performance, or optimizing the driving line to enhance speed, these improvements are outside the scope of this paper and will be addressed as part of future work. At this stage, we have developed an agent that is suitable for use in a traffic simulation for a game.

5. Agent Inertia Management

Now that our agent can steer itself, it's time to make it brake so that it does not cause an incident. In the previous tests, the agent accelerated with the maximum value and simply stopped accelerating once it reached the target speed of 10 m/s. To make the agent suitable for a traffic simulation, it requires the minimum functionality of stopping when reaching the end of the road and matching the speed limit of the road (or slightly less when it comes to my grandma). Based on my other research, I discovered that the PID controller I have used for steering the agent toward the centre of the road is highly effective at correctly adjusting a changing value to align with a changing target. When it comes to the agent's speed there are two primary factors; the **speed limit** and the **stopping point**, which is used for keeping distance from other agents and intersections. An agent requires to drive the speed limit, but this does not require a direct responds. We therefore use a proportional gain to reach the target and smooth out the value by moving the input over time. The other part is making sure when given a position along the path, the agent is capable of exactly stopping on the path point without overshooting. This second part is ideal for PID and is very similar to the steering.

Generic PID controller

To keep code quality up we define a generic PID controller to be used by both the steering and the brake point power input.

```
public PIDResult Evaluate(float error, float errorRate, float deltaTime) {
    float p = Settings.ProportionalGain * error;
    float d = Settings.DerivativeGain * errorRate;
    _accumulatedError += error * Settings.IntegralGain * deltaTime;
    _accumulatedError = Mathf.Clamp(_accumulatedError, -Settings.Integral_Limit,
Settings.Integral_Limit);
    float i = _accumulatedError;

    return new PIDResult {
        Proportional = p,
        Integral = i,
        Derivative = d,
        Total = p + i + d
    };
}
```

Figure 37. Code snippet showing how the PID controller is made generic using an evaluate function within the PIDController class.

Another advantage of this is that we can store the settings of the PID in a struct, we then make this struct [Serializable] so that we can use it in our agent settings scriptable object, allowing us to create files for the settings and have saved configurations.

```
[Serializable]
public struct PIDSettings
{
    [Tooltip("Proportional acts as a spring, pulling the car back to the path")]
    public float ProportionalGain;

    [Tooltip("Integral acts as a memory, reducing the steady state error")]
    public float IntegralGain;

    [Tooltip("Derivative acts as a damper, reducing the oscillation of the car")]
    public float DerivativeGain;

    [Tooltip("Integral saturation limit, to prevent windup")]
    public float Integral_Limit;
}
```

Figure 38. Code snippet showing a [Serializable] struct for the PID settings

The model employs a realistic framework that simulates the operation of both the gas and brake pedals, as discussed in Section "3 Physics-Based Testing" Each pedal's input ranges from [0, 1], reflecting proportional values. To streamline the output for the PID controller, these two inputs are combined into a single value ranging from [-1, 1]. In this range, the negative half represents the brake pedal, while the positive half corresponds to the gas pedal. This configuration allows the PID controller to adjust the combined value based on its calculations while preserving the realistic behavior of the vehicle together with the code composition. Drivers in real life often just let go of the gas paddle to let the car come to a stop and because our model is physical and PID just cares about the error change, this phenomenon is replicated similar to real life. There is also an important distinction to be made between the two values. As the gas paddle adds a force to the vehicle based on the torque curve while the brake paddle makes changes to the grip of the vehicles wheels. This makes it so that the vehicle has different speeds for slowing down and speeding up.

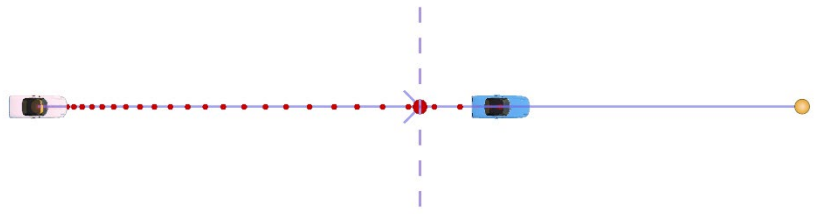


Figure 39. Showing the physical braking test with full throttle and full braking.

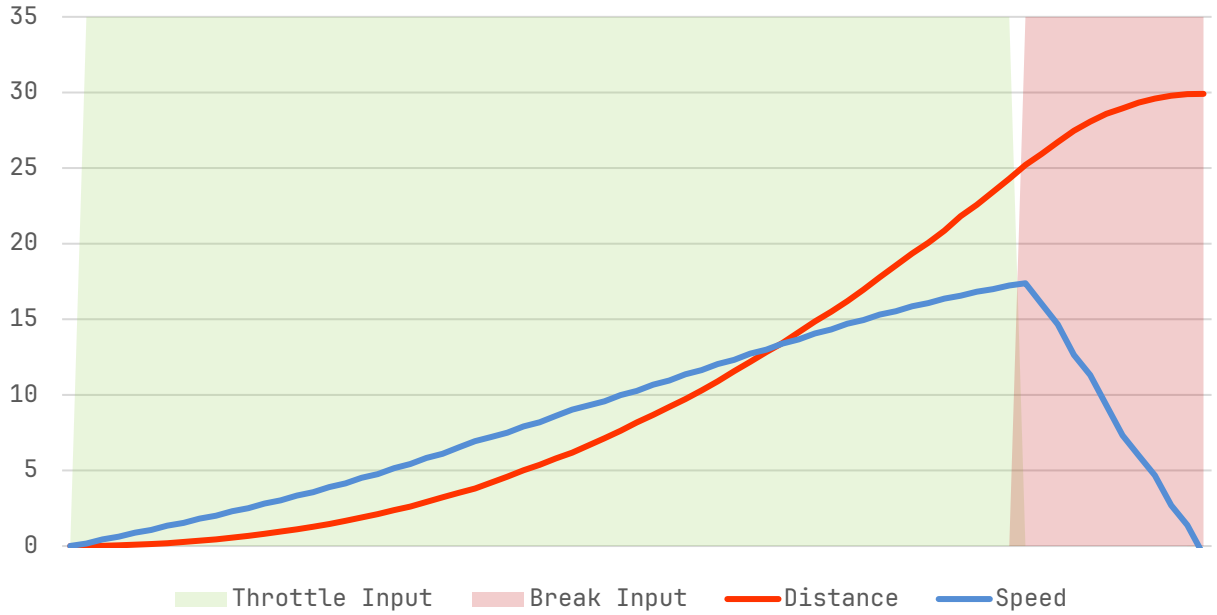


Figure 40. An example of a vehicle's inertia shown using Distance & Speed together with Throttle & Brake. using full brake and throttle based on braking line without PID control.

As shown in these graphs the agent seems to have a smooth ride based on the distance but when taking a look at the speed it is clear that the passengers experience a very rough ride. This is a result of full throttle and full braking.

Figure 41. Example of agent full braking making the car tilt forward due to inertia.

5.1 Speed limit

Roads are governed by speed limits, which agents in traffic simulations must follow intelligently. In real-world scenarios, drivers transitioning between speed zones typically do not abruptly reduce their speed by slamming on the brakes. Instead, they decelerate gradually until they comply with the new speed limit. The objective in the simulation is to replicate this behavior by ensuring that the agent adheres to speed limits in a smooth and realistic manner. The proposed method involves utilizing a Proportional Controller. This approach calculates the proportional value as the difference between the target speed and the current speed along the sampled direction. By multiplying this value by a gain factor, the controller ensures that the target speed is achieved. This method is simple and does not require a derivative component, as the speed in this context is not a moving value.

$$P = K_p \cdot \left(v_{target} - v_{current}\right)$$

Figure 42. Formula for calculating speed limit proportional input

The code snippet below shows the final calculation for the speed limit. More about the smoothing part is discussed below.

```
private void UpdateSpeedLimitInput()
{
    // Calculate the desire input to reach the target speedLimit
    float speedAlongDirection = Vector3.Dot(_agent.CarBehaviour.Velocity,
_agent.CurrentSample.DirectionForward);
    float gain = speedAlongDirection > _agent.CurrentSample.SpeedLimit ?
_agent.Settings.SpeedLimitBrakeProportionalGain :
_agent.Settings.SpeedLimitThrottleProportionalGain;
    float targetSpeedLimitInput = (_agent.CurrentSample.SpeedLimit -
speedAlongDirection) * gain;
    _speedLimitInput = Mathf.MoveTowards(_speedLimitInput, targetSpeedLimitInput,
_agent.Settings.SpeedLimitInputMaxCangeRate * Time.deltaTime);
}
```

Figure 43. Code snippet showing the final calculation for speed limit.

The results below show how the agent fairs with different gain values. It is important to note that it’s not important for the agent to immediately reach the max speed. It is much more important that the agent does so in a smooth matter.

Speed Limit Proportional Gain Test Results

Throttle & Brake Inputs over time [0,1]. Speed m/s over time [0,25].
Same gain used for both throttle and brake input.

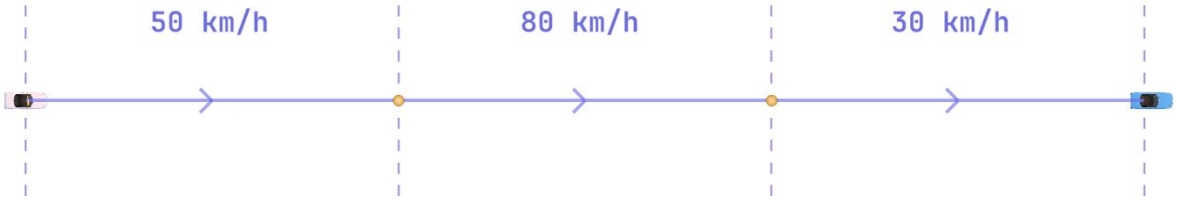


Figure 44. Showing the physical speed limit test used for the graphs below.

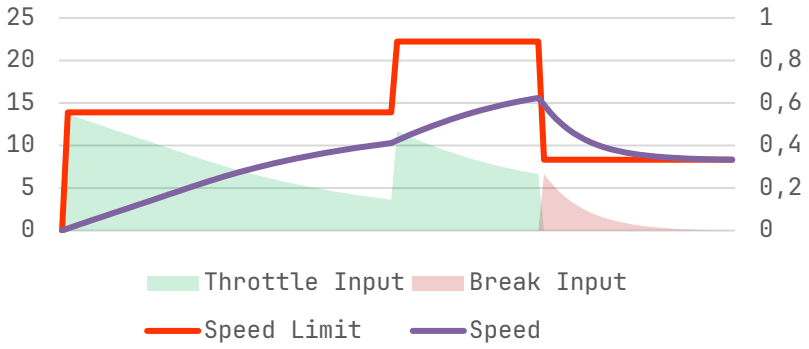


Figure 45. **Speed limit** test with gain set to: **G = 0.04**

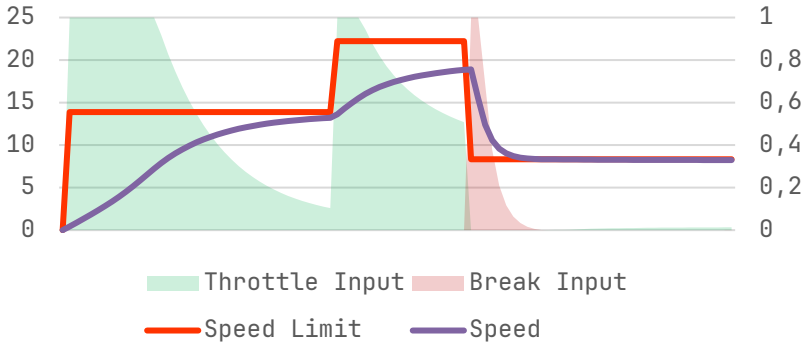


Figure 46. **Speed limit** test with gain set to: **G = 0.15**

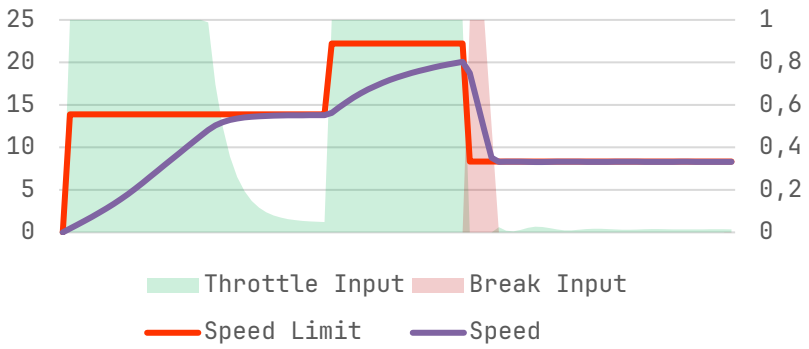


Figure 47. Speed limit test with gain set to: **G = 0.5**

These results show that we can make our agent follow a target speed by using a proportional value. This creates a smooth out braking and throttle response so that the agent slowly decreases input when getting closer to the target speed. There is still quite a shock when the speed is far away from the target speed. We do want our agent to give a proportional input, but we still want to smooth out the sudden changes in input. A proposed method is to similarly to "4.3 Path Deviation Optimization and Smoothing" is to smooth out the final input. Ideally the smoothing created when nearing the target speed is preserved. We can apply the smoothing by moving the resulting input value and allowing the value to only move with a set maximum delta. This allows for a slow gradual change in input to be preserved but limits the large abrupt changes. For this we use a Unity function called `Mathf.MoveTowards()`, which is implemented with the following formula:

$$f(p, z, \delta) = \begin{cases} z & \text{if } |z - p| < \delta \\ p + \text{sign}(z - p) \cdot \delta & \text{otherwise} \end{cases}$$

Figure 48. Example of move towards formula used for limiting the change in input for the speed limit calculation

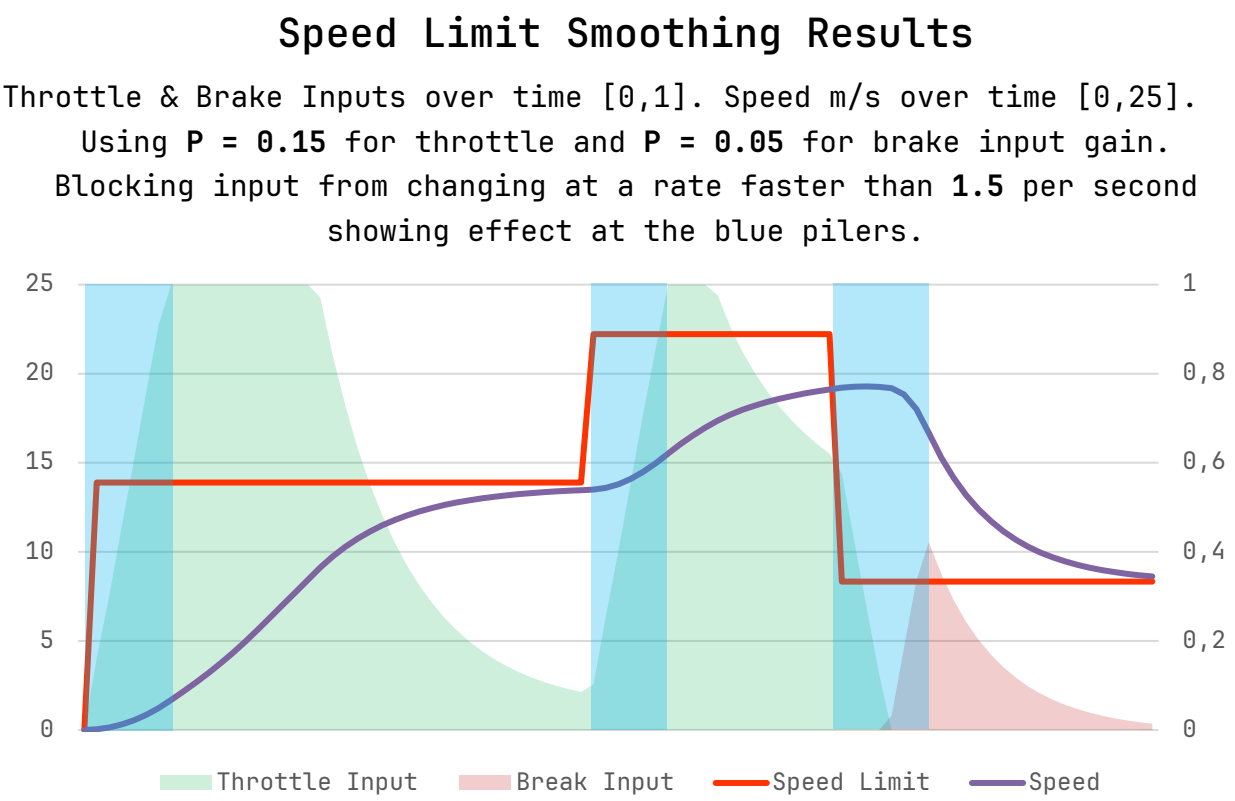


Figure 49. Graph showing the speed limit smoothing results.

5.2 Stop Point

The stop point is simply defined as a position. The agent will regulate the brake and throttle controls accordingly so that the vehicle comes to a stop precisely at the stop point. This method is very similar to the steering from “4.1 Path Alignment Using PID” as we are again moving the vehicle to a point, only this time in the straight direction instead of sideways and using the throttle and brake input instead of the steering input. Therefor this can be accomplished using a PID-controller. In this scenario we don’t ever want the agent to overshoot like he might when it comes to steering. This is why it’s important to correctly tune the PID-controller. The following test results will show how different P&D values compare to creating a smooth stopping experience.

Stop Point Results

Throttle & Brake Inputs over time [0,1]. Speed m/s over time [0,25]

Figure 50. Showing a top-down view of the test scene for the **stop point test**.

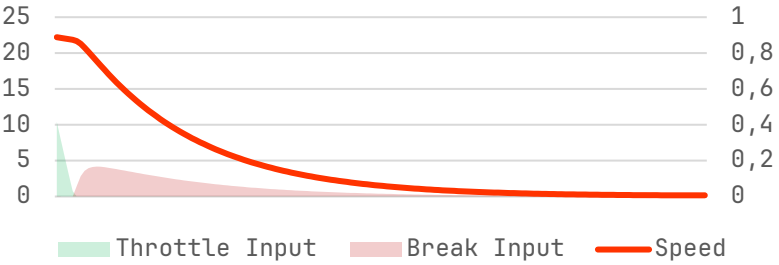


Figure 51. Showing braking test with gain set to:
P = 0.05
D = 0.15

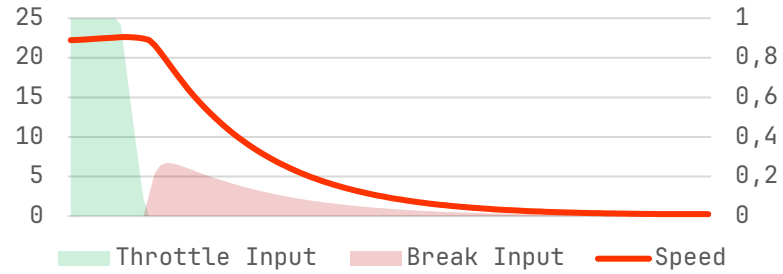


Figure 52. Showing braking test with gain set to:
P = 0.1
D = 0.2

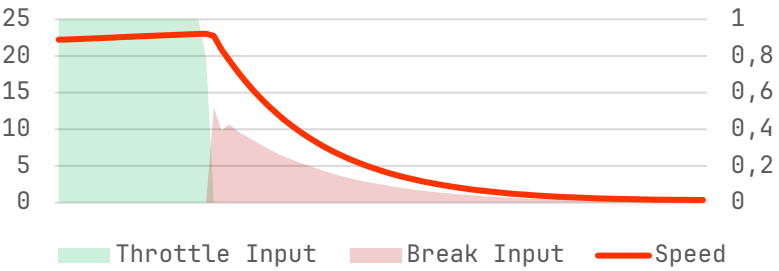


Figure 53. Showing braking test with gain set to:
P = 0.6
D = 0.8

The stop point PID controller uses the signed distance based on the forward direction vector of the agent and the direction towards the stop point. The error rate is defined as the velocity of the agent projected in the direction towards the stop point.

```
// Calculate the input to reach the stop point
Vector3 fromPosition = new Vector3(_agent.CarBehaviour.Position.x, 0.0f,
_agent.CarBehaviour.Position.z);
Vector3 toPosition = new Vector3(_stopPoint.x, 0.0f, _stopPoint.z);
Vector3 directionToStopPoint = (toPosition - fromPosition).normalized;

// Calculate the error
float directionRelativeToAgentForward =
Vector3.Dot(_agent.CarBehaviour.ForwardPlanner, directionToStopPoint);
float distanceToStopPoint = Vector3.Distance(_agent.CarBehaviour.Position,
_stopPoint);
float signedDistanceToStopPoint = distanceToStopPoint *
Mathf.Sign(directionRelativeToAgentForward);
float error = signedDistanceToStopPoint;

// Calculate the error rate
float velocityAlongDirectionToStopPoint = Vector3.Dot(_agent.CarBehaviour.Velocity,
directionToStopPoint);
float errorRate = -velocityAlongDirectionToStopPoint;

// Evaluate the PID controller
_stopPointInput = _stopPointPIDController.Evaluate(error, errorRate,
Time.deltaTime).Total;
```

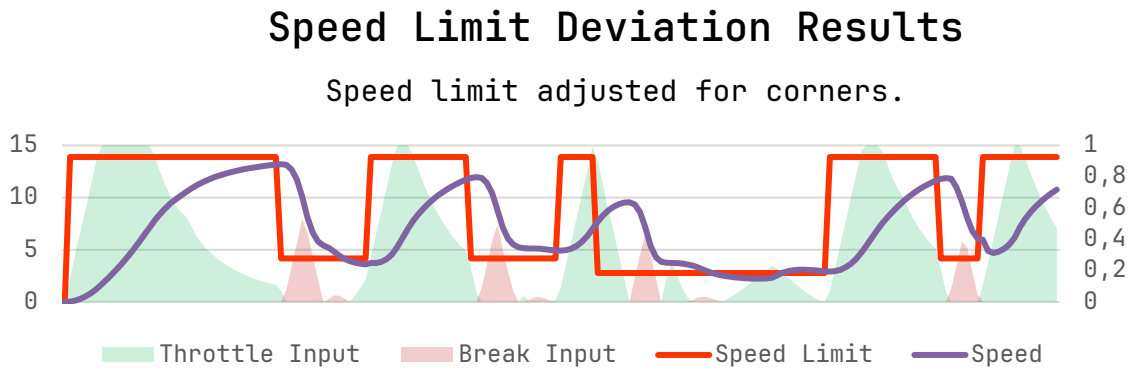
Figure 54. Code snippet for calculating the stop point input value

These PID values are currently manually tested and compared but there are many algorithms that can help you tune these values to be exactly correct in moving your value towards the target.

5.3 Sharp corners

A road network should ideally consist of roads that are governed by speed limits that comply with the sharpness of the corners to avoid agents from overshooting. However, not all road systems are perfectly set up and even in real life there is often a suggested speed sign for a corner as the speed limit does not always represent how fast you should take a corner.

When the deviation tests were performed in “4.3 Path Deviation Optimization and Smoothing” the agent drove at a constant speed of 10 m/s. When that same scene is adjusted to include speed limits the result is majorly improved, especially in the first sharp corner, at the cost of manual work for placing the speed limits.



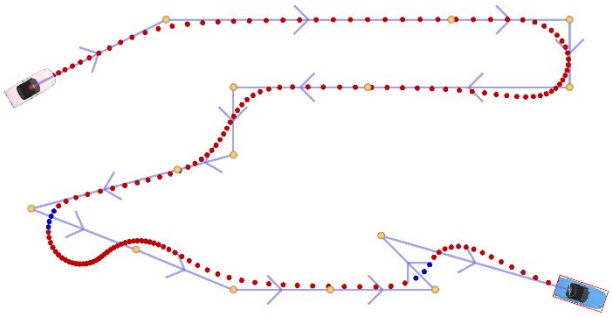


Figure 55. Top-down view of deviation results with speed limits.

Average Deviation = **0.505**

Manually adjusting the corners is not ideal and another method is calculating the danger level for the upcoming corner by sampling the angle of the corner ahead in the path. This paper, however, does not further explore these additions as they are not mandatory. Hopefully this can be addressed as part of future work.

Figure 56. Example of the types of danger values would be included if such a system would be implemented

5.4 Combined Input

The agent can adjust its speed to adhere to the speed limit while simultaneously driving towards a designated stop point. In a traffic simulation, however, it is essential for the agent to perform both tasks concurrently. Therefore, the results of these two inputs are combined.

The process of combining these inputs was developed through straightforward reasoning. Both the speed limit and the stop point serve to restrict the agent's acceleration. The throttle input is determined by selecting the minimum value between the two inputs, as the speed limit should establish the maximum permissible throttle. Conversely, the brake input is determined by selecting the maximum value, ensuring that the vehicle consistently decelerates to a complete stop, whether prompted by the speed limit or the stop point.

$$I = clamp01\big(\min\big(I_l, I_p\big)\big) - clamp01\big(\max\big(-I_l, -I_p\big)\big)$$

6 Obstacle Avoidance

Having developed the necessary components to enable our agent to regulate inputs for speed control, we can now focus on the next step: collision avoidance. The agent must be capable of responding to potential obstacles of various types, such as other agents or debris on the road. Currently, the agent does not account for obstacles in its path, resulting in collisions.

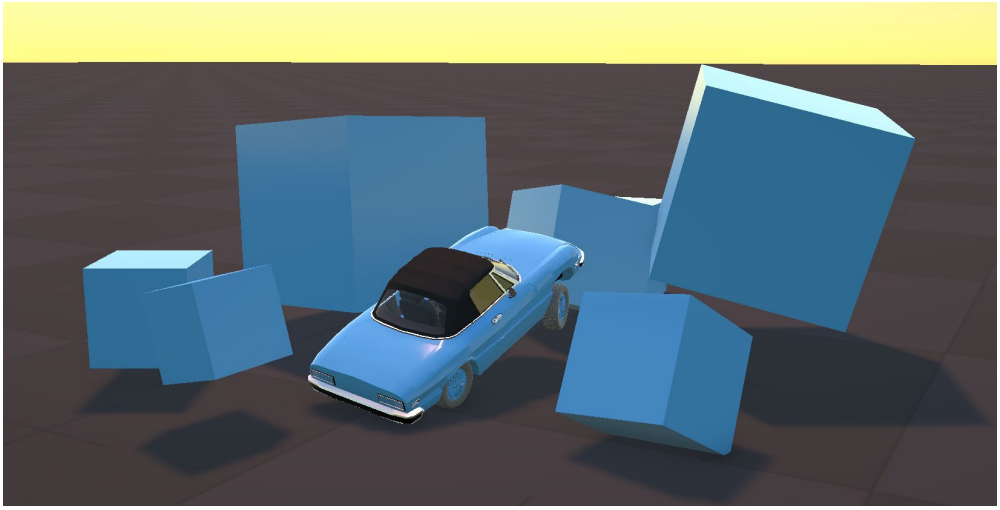


Figure 57. Agent crashing into a pile of boxes.

6.1 Directional Front Sensor

To begin with, it is necessary for the agents to detect any obstacles in their path. Several methods can be employed to achieve this. One potential approach involves sampling the path and identifying other agents present on the same road. However, this method does not account for dynamic obstacles beyond the agents themselves. An alternative solution is to perform a box cast from the agent's bumper, extending in the direction of the steering angle for a specified distance. This can be accomplished using Unity's physics system.

Once again to maintain code quality, the sensor is split into a different component. The biggest reason this is more flexible is that we can attach the sensor to the front of the vehicle as a different game object, making use of the transform.

```
public class Sensor : MonoBehaviour {
    [SerializeField] private Vector2 _size = new Vector2(1.0f, 1.0f);
    [SerializeField] private LayerMask _layerMask = 0;

    public bool Sense(float distance, out RaycastHit hit) {
        return Physics.BoxCast(transform.position, _size, transform.forward, out hit,
transform.rotation, distance, _layerMask);
    }
}
```

Figure 58. Code snippet of sensor component.

Due to the separation of the sensor, it is easy to use the transform component to rotate the sensor direction in the direction of the vehicle steering angle, and to move it slightly to avoid it colliding with the vehicle body.

```
float steeringAngle = _carBehaviour.SteeringAngleDegrees;
_frontSensor.transform.localRotation = Quaternion.Euler(0.0f, steeringAngle, 0.0f);

Vector3 target = _frontSensor.transform.localPosition;
target.x = steeringAngle * _frontSensorAnglePositionOffset;
_frontSensor.transform.localPosition = target;

_frontSensor.Sense(_frontSensorDistance, out _frontSensorHit);
```

Figure 59. Code snippet of using sensor component while rotating it towards the steering angle.

Directional Front Sensor Corner Navigation Results

One limitation of the current system is its performance in navigating corners. The code snippet shown in "Figure 57" provides a reasonable solution by rotating the cast in the steering direction; however, the steering direction is only the start of the curve. This also does not account for the future position of the agent as it might change direction. “Figure 60” shows how the directional front sensor fails to collide with the other agents on the path.

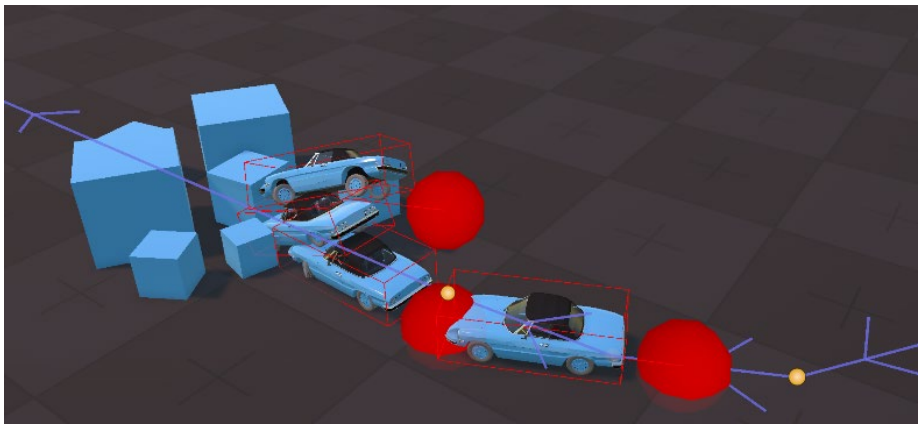


Figure 60. Top-down view of agent driving along a curved path with sensor aimed in the steering direction. With the red ball representing the stop point. There is an error shown as the front sensor does not see the box in the center of the path.

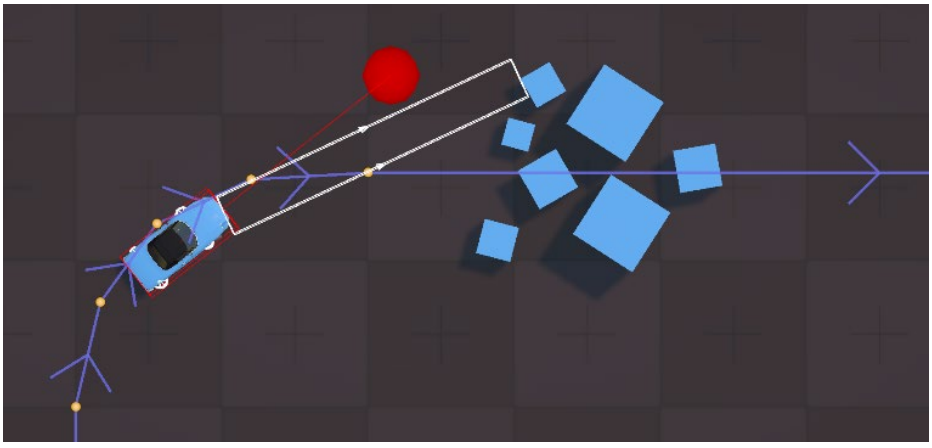


Figure 61. Example of agents driving along a curved path. They are not correctly capable of stopping for each other when there is a sharp corner.

6.2 Rotational Front Sensor

Another approach would be to check the steering direction of the agent and predict its future position based on the bicycle kinematic model. One such method is described in “The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?” [28] with a detailed approach in “Kinematic Bicycle Model” by Mario Theers and Mankaran Singh [29].

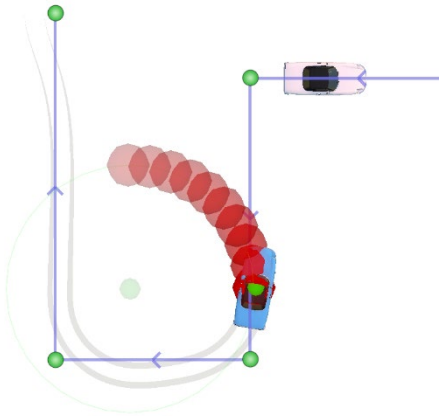


Figure 62. Showing sensor prediction using the kinematic bicycle method failing to detect agent ahead of the path.

```
_turningRadius = wheelbase / Mathf.Tan(_steeringAngle);
_turningCenter = backWheelCenter - _turningRadius *
backWheelSideways;
```

Figure 63. Code snippet of calculating the turning center.

However, it is made clear from “Figure 62” that this simple implementation of the approach lacks the knowledge of the path which the agent will follow.

6.3 Path Sampling Front Sensor

The agent has access to the path it will follow. This information can be used to sample points along the path from the car position to be used for collision avoidance. This method samples points and then performs a box cast between each of these points.

When the agent has deviated off the path, this sensor might not detect the trajectory which the agent will follow to get back to the path. The position of the front sensor will be used as the first sampled point. This allows the agent to still detect obstacles between it and the path.

The agent has included additional code that can sample points from its current segment up until the end of the next segment. The agent always has a target next segment but does not know further what path to take as it does not do any path finding, only following.

The number of samples influences the accuracy of this sensor. However, sampling the path and especially casting between the two samples comes at a steep performance cost. The front sensor has a setting defining the samples per meter.

Path Sample Number Results

how sample count affects the accuracy of the path sensor.

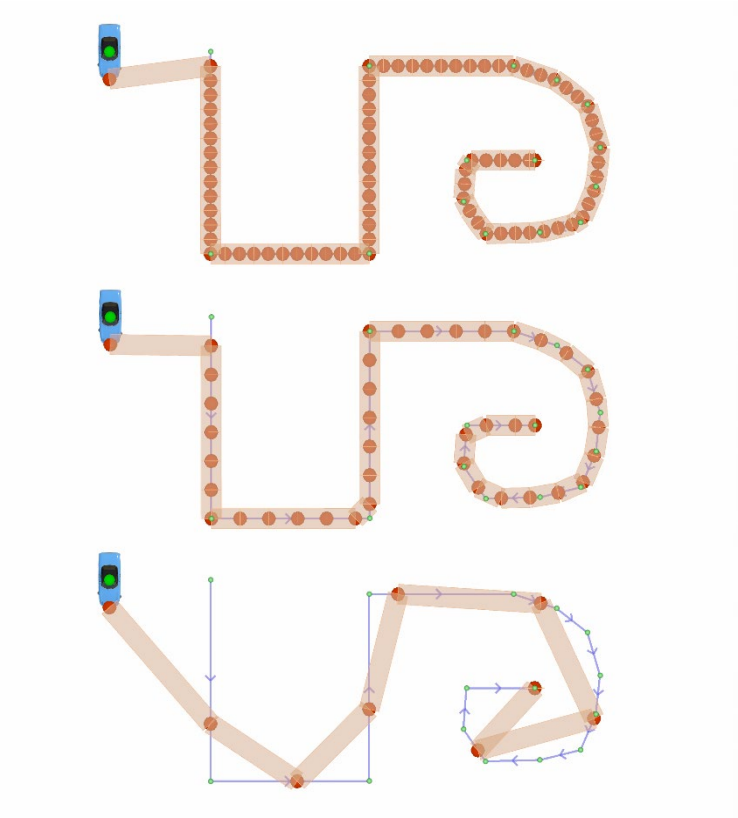


Figure 64.
Showing front sensor sampling path points with samples per meter set to: **1.0**

Figure 65.
Showing front sensor sampling path points with samples per meter set to: **0.5**

Figure 66.
Showing front sensor sampling path points with samples per meter set to: **0.1**

It is important to note here that, ideally the number of samples is as low as possible while still upholding a decent resolution for detecting obstacles along the path.

One more optimization technique that can be used is removing points based on how steep the angle is. In “Figure 65” it is made clear that the number of samples is correct when sampling in the spiral, but when taking a closer look to the straight sections, it shows that there are an unnecessary number of samples.

For this optimization I came up with the following algorithm: It iteratively removes points, until there are none left to be optimized based on the minimum angle. This algorithm borrows inspiration from the “Ramer-Douglas-Peucker algorithm” [30]. Part of future work could be going further into this optimization.

```
while (true)
{
    int removedSamples = 0;
    for (int i = 1; i < _samples.Count - 1; i++)
    {
        Vector3 directionToCurrentSample = _samples[i] - _samples[i - 1];
        Vector3 directionToNextSample = _samples[i + 1] - _samples[i];
        float angle = Vector3.Angle(directionToCurrentSample, directionToNextSample);

        if (angle < _minAngle)
        {
            _samples.RemoveAt(i);
            removedSamples++;
        }
    }

    if (removedSamples == 0)
        break;
}
```

Figure 67. Code snippet of line smoothing algorithm used for the optimization of the front sensor.

Path Sample Angle Optimization Results

how removing points based on the angle affects the number of samples.
Samples per meter set to 0.5

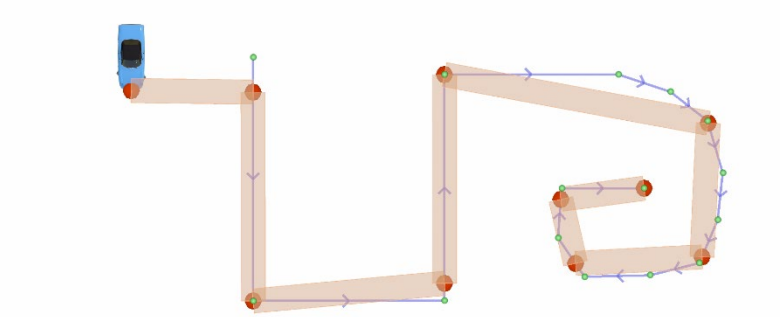


Figure 68.
Path Sampling Optimization using minimum angle of:
60 degrees.

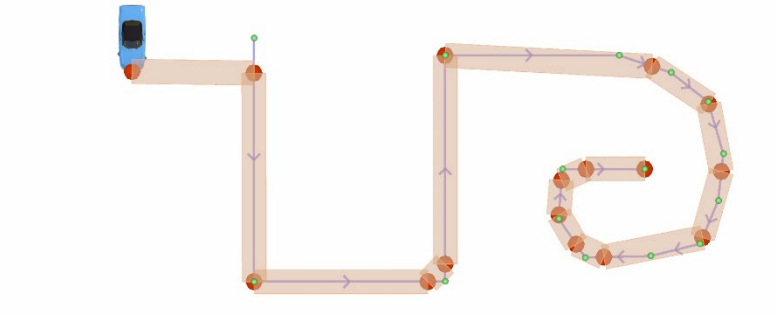


Figure 69.
Path Sampling Optimization using minimum angle of:
20 degrees.

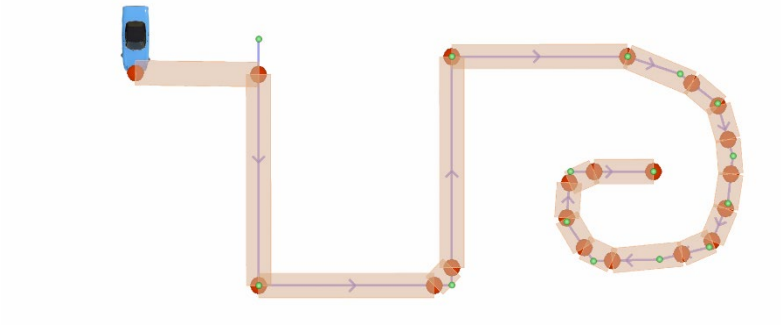


Figure 70.
Path Sampling Optimization using minimum angle of:
10 degrees.

A small addition made to the agent is the introduction of reaction speed. Each agent is assigned a minimum and maximum reaction speed, from which it selects a random value. The agent senses at an interval determined by this reaction speed. This enhancement improves optimization by reducing the number of queries each frame, and enables the agent to replicate the reaction speed of a real driver, adding to the realism.

6.4 Stop Point Positioning Using Front Sensor

Using the sensor data we can move the stop point discussed in “5.2 Stop Point” to slow down the agent and prevent it from crashing into the boxes. We can apply the following formula where; P represents the current position of the agent, from the center. F represents the forward vector of the agent. d_{front} represents the front sensor hit distance. d_{offset} represents the stopping distance offset ($d_{offset} = agentLength \cdot VLM$).

$$stopPoint = \boldsymbol{P} + \boldsymbol{F}(d_{\text{front}} - d_{\text{offset}})$$

Figure 71. Formula for calculating the stop point.

Stop Point Positioning Results

Paused when the first agent in line reaches the stop point
With VLM defined as the Vehicle Length Multiplier

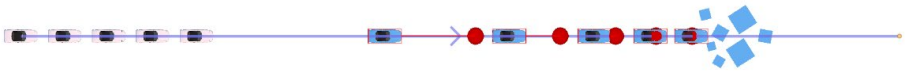


Figure 72. Stop point positioning. Using: **VLM = 0.0**

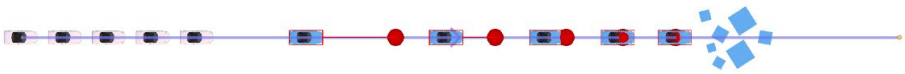


Figure 73. Stop point positioning. Using: **VLM = 0.5**

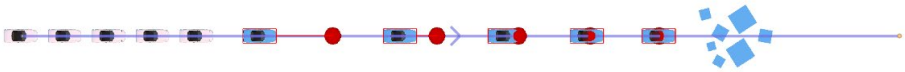


Figure 74. Stop point positioning. Using: **VLM = 1.0**

For the VLM to be calculated we require the agent size. This size is defined in the agent component and is show in “Figure 64” below:

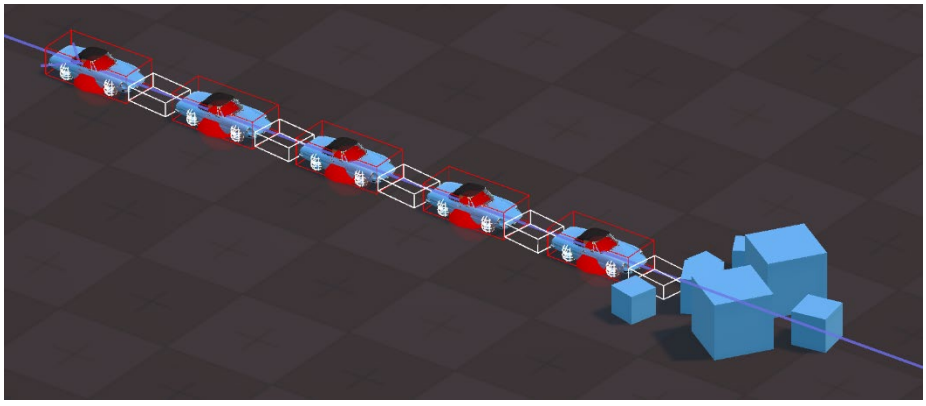


Figure 75. Showing agents lined up behind each other as they brake for the stopping point. Red boxes represent the **agent size**, and the white boxes represent the **physics box cast**.

7. Agent Driving Practical Results

The agent is capable of steering and managing its inertia to reach the speed limit while avoiding collisions. To evaluate its performance effectively, it is appropriate to test the agent on a road alongside other agents.

The results demonstrate that the agents perform effectively, with all of them successfully reaching the end of the path without any collisions.

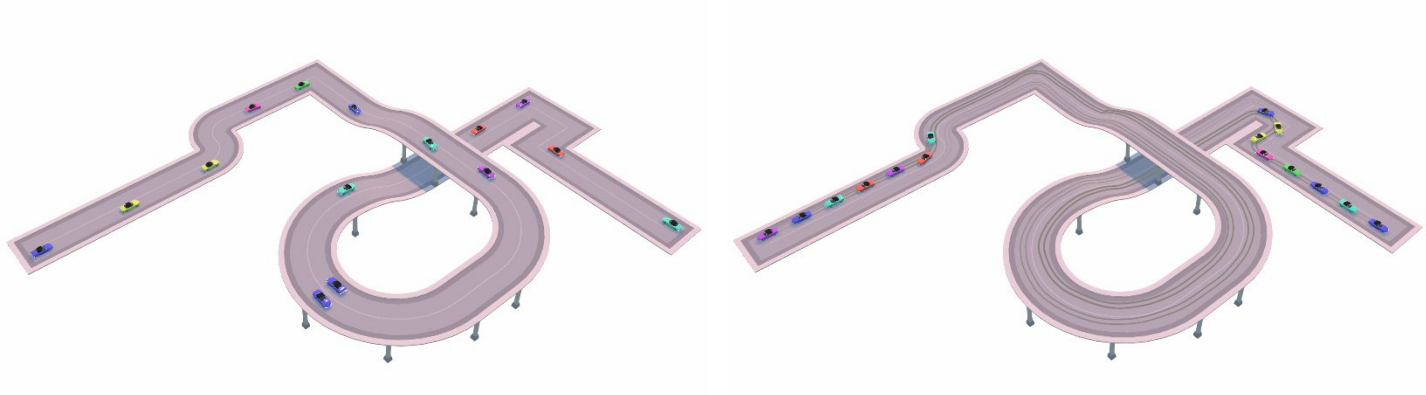


Figure 76. Showcase of agents driving on a two directional single lane road. Going from start to end visualized by the tire marks.

In order to test collision avoidance, physics cubes have been placed on the track. This demonstrates the ability of the agents to avoid collisions even when an obstacle is not defined as an agent on the path.

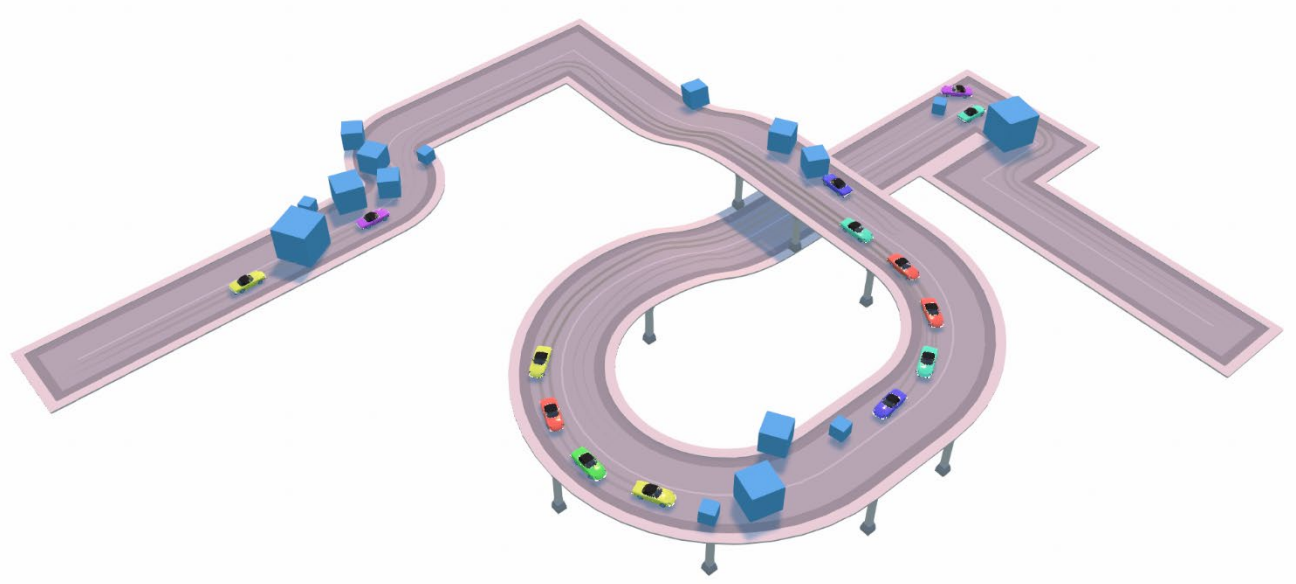


Figure 77. Example of road blocked by obstacles.

7.1 Phantom Traffic Jams

A phantom traffic jam is a phenomenon often experienced by drivers. This is a traffic jam that seemingly appears out of thin air, and is an effect of drivers having to suddenly brake as they don't keep enough distance from other drivers. This phenomenon is amazingly described by Benjamin Seibold in their Ted-Ed [31], CGP Gray with "The Simple Solution to Traffic" [15], and "Phantom Traffic Jams" from the BBC One Show, Series 3 [32].

This phenomenon can easily be visualized when putting agents on a circular road. The agents get a sphere surrounding them that visualizes their speed to easily identify the stopped agents. Below is a top-down perspective of the road with "Figure 78" showing all cars with no disturbance and "Figure 79" showing simulation after introducing disturbance.

This phantom traffic jam proceeds to infinitely travel in the opposite direction of the driving agents, eating their speed up like a snake.

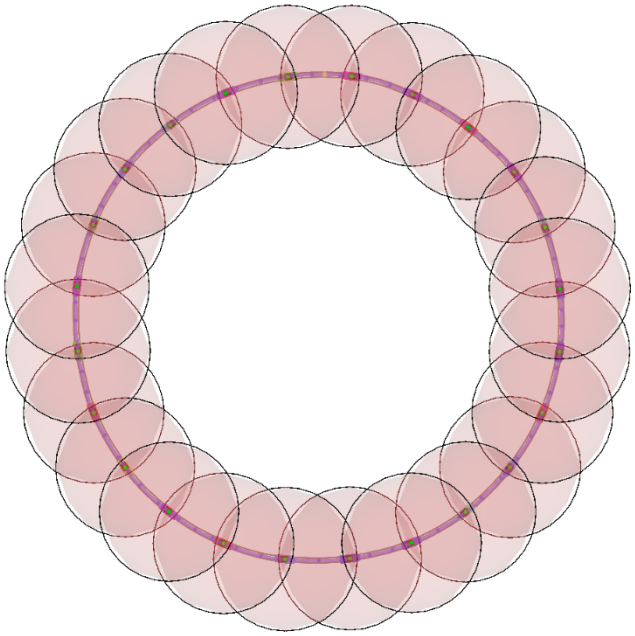


Figure 78. Example with the lack of a phantom traffic jam due to no disturbance.

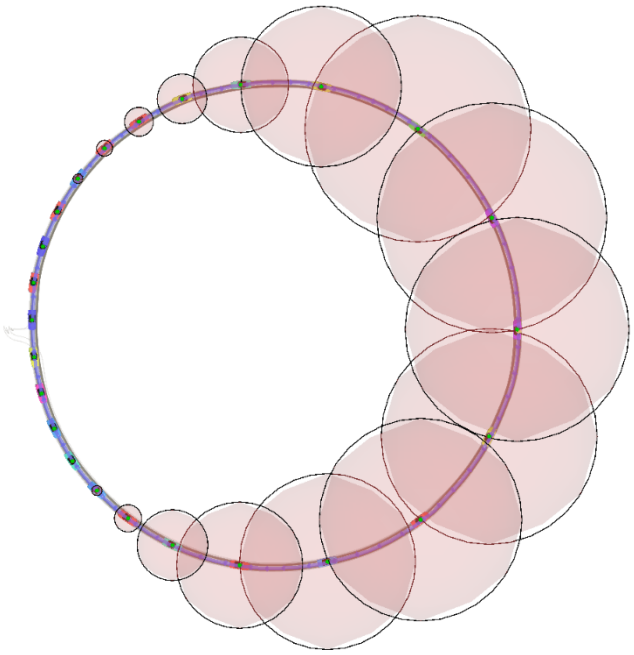


Figure 79. Example of phantom traffic jam with disturbance.

8. Intersections

The agents are controlled by AI (Artificial Intelligence), but they are not really all that intelligent. Even when we predict the path, the agents still can’t predict the paths of other agents, causing them to collide. This is why intersections are needed. The goal of an intersection is to coordinate the agents driving through a crossing, avoiding collisions.

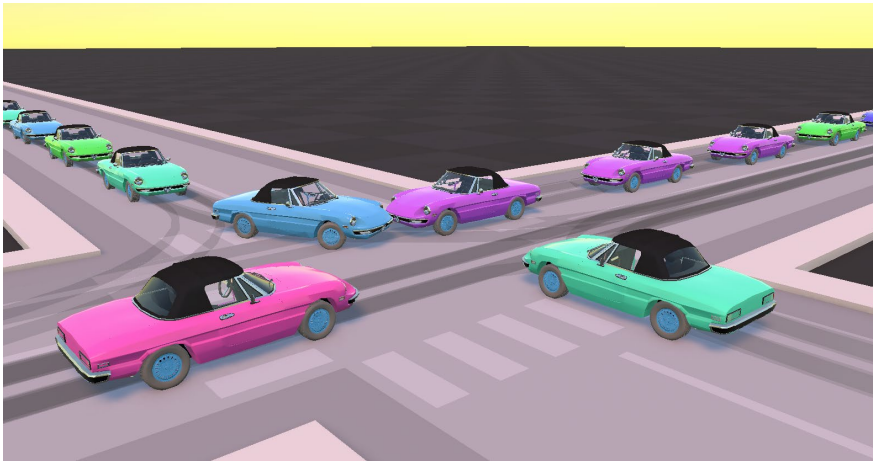


Figure 80. Agent crashing due to the lack of intersection guidance.

Intersections, in short, are simply when a segment from the road network crosses with a different segment. In this simulation, intersections are set up by connecting multiple segments together. The intersection which regulates the agents is defined as a game object containing an intersection component and box collider component set to be used as a trigger.

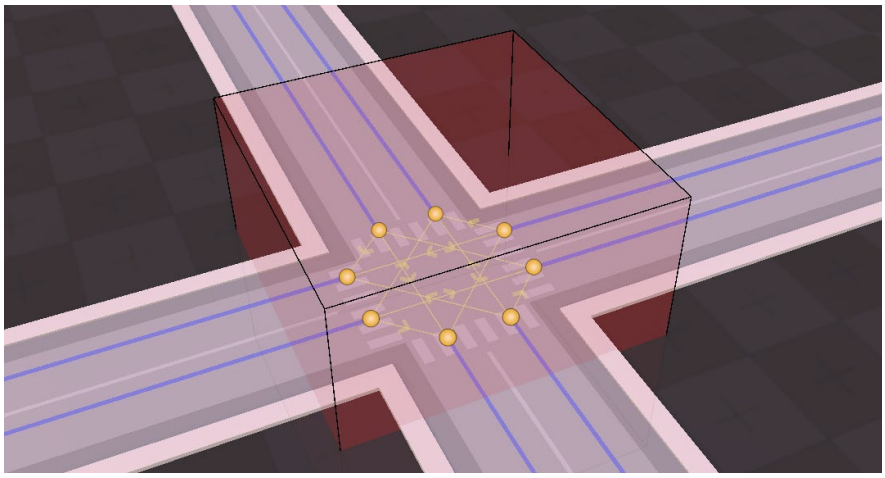


Figure 81. Example of intersection defined by a red box surrounding the waypoints making up the intersection.

The intersection generates a list of possible turns and crossings based on 2D line intersection. It also stores information about the angle of the turn, the distance, and direction. Each turn also defines a state describing whether the turn is clear, occupied, or blocked. This case study will not dive further into the implementation specific details, these details can be found in the appendices.

Due to the intersection having information about each turn and crossing, it can smartly open and close turns based on its occupation. This works similar to how rail signals work. The intersection updates in-range agents' intersection state which consists of none, waiting, and moving. The agents can directly respond to this and drive as if there is no intersection at all. One future improvement could be to incorporate driver reaction speed into the simulation as agents now directly get information about the intersection. The intersection currently updates these states when an agent enters the intersection or when an agent leaves the intersection.

One small addition made is that when an agent spends too much time in an intersection it will simply ignore the intersection state and force its way tough using the regular collision avoidance. This is similar to how games like GTA work, you can see this in action when blocking a highway, cars will slowly start to ram each other when stuck for an extended duration [33].

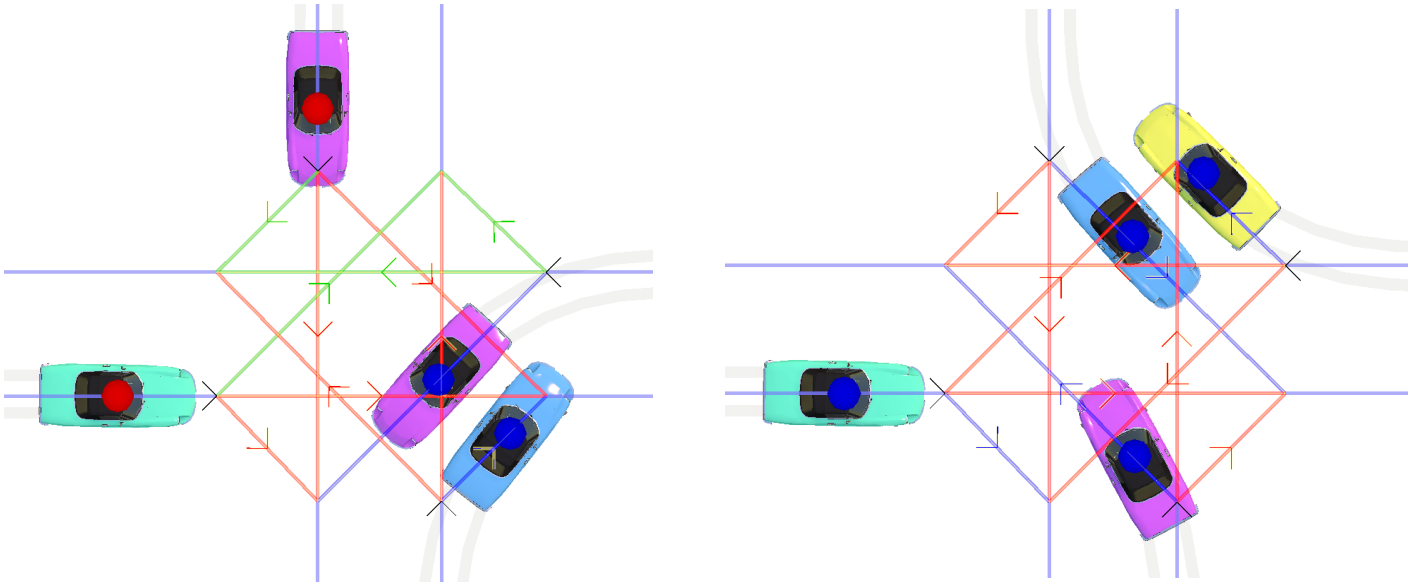


Figure 82. Example of intersection. Green lines representing clear turns, red as blocked turns and blue for occupied turns. Agents with a blue ball are set to moving, agents with a red ball are set to waiting.

8.1 Intersection Priority Rules

Intersections with no priority rules only really work in a simulation, as real drivers require time to figure out what other drivers will do. However, agents in this simulation directly respond to the intersection clearing up. Even with the agents capable of instantly communicating, it would be ideal for the simulation to incorporate priority rules, as this would add a layer of realism.

In order to test the effects priority-rules have on the simulation, the following test scene will be used. This scene includes an intersection with two single-lane roads crossing, both with two turns. The simplicity of this intersection will be useful in visualizing the difference, as with multiple lanes the data will be too cluttered. The test scene also has, on both sides, a set of ten cars. These cars will attempt to drive to the end of the road and will randomly pick one of the two turns presented to them.

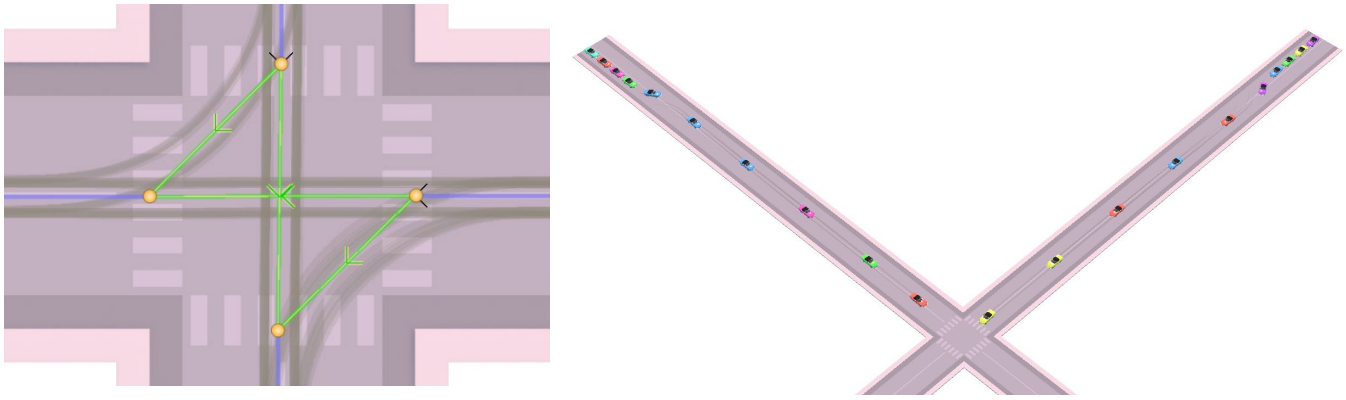


Figure 83. Top-down view of two single lane roads coming together for an intersection. Both roads have two turns. Each side has ten cars totaling twenty cars for this intersection test.

The results below show how both lane occupation (no priority), arrival priority, and directional priority intersection result in similar throughput. This of course is in an ideal simulated environment and this case study does not go further into detail about the reaction time of the agents. The focus here lies on the order in which agents go through the intersection.

Intersection Priority Rules Results

Based on **20 agents** driving on **2 roads** with **4 turns**
as shown in “Figure 83”. Each color represents a lane of agents.

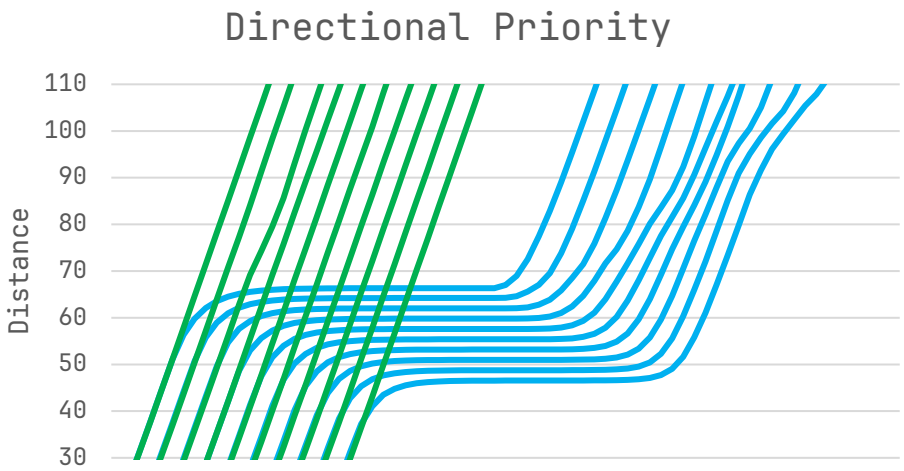


Figure 84. **Directional Priority** showing how agents drive in clusters.

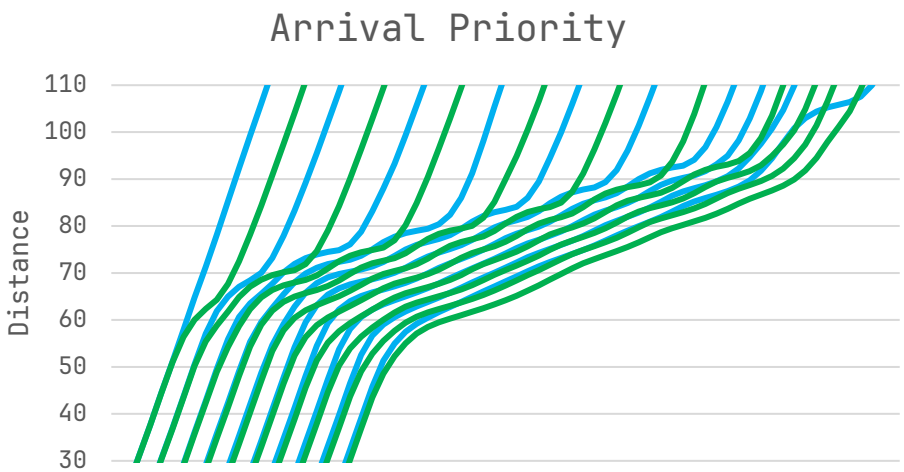


Figure 85. **Arrival Priority** showing consistent one by one driving behavior.

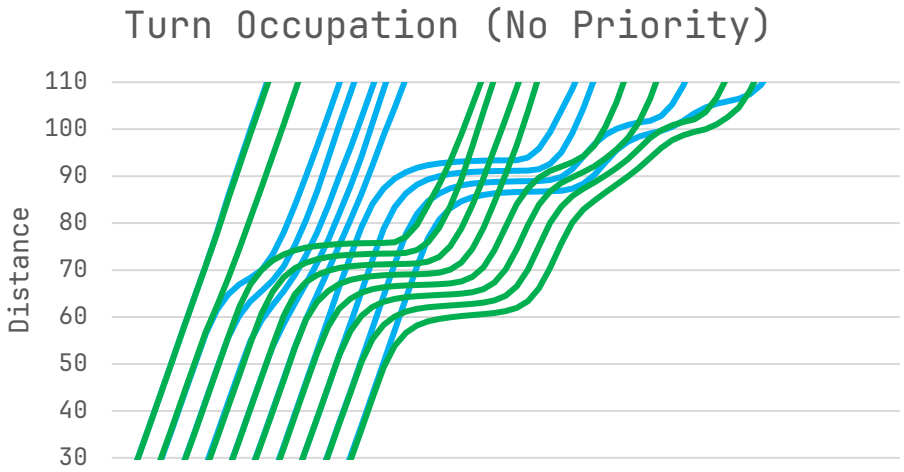


Figure 86. **Lane Occupation** showing how agents flow through the intersection as soon as a turn is available.

8.2 Roundabouts

A simple intersection as seen in “8.1 Intersection Priority Rules” works but is quite slow at handling large amounts of traffic. The building of road infrastructure and optimizing road networks to accomplish optimal flow is an entire field on its own. With the goal of this paper in mind, it would be a great test to see how agents perform when driving on different types of intersections.

To test different types of intersections, the intersection component should be configurable to different priority rules. For instance, to configure a roundabout, the agents driving on it have the right of way. In this scenario, the intersection should be configured with the left-hand rule, where agents coming from the left direction (on the roundabout) have priority over those coming from the right (on the main road). A roundabout could be simplified to four different smaller intersections. As shown below, the roundabout is configured using different intersections all set with directional priority on the left-hand side.

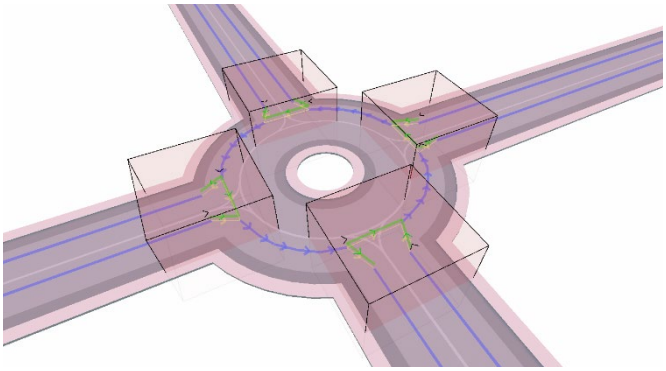


Figure 87.
Example of roundabout setup using four different small intersections set to use left direction priority.

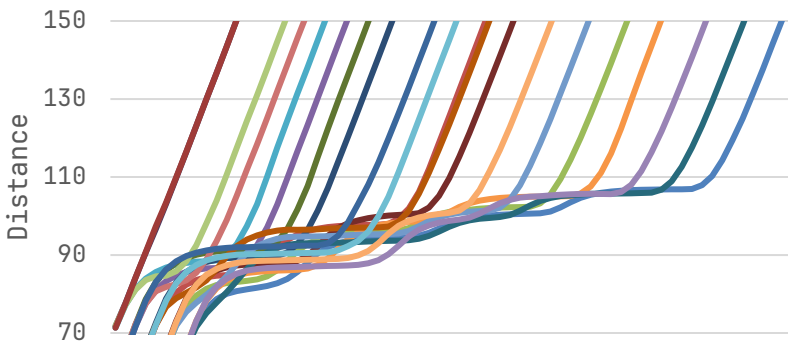
8.3 Intersection Types

To round up this case study, a final experiment is created were the agent with all the previously discussed parameters, including steering, throttle, brake, and collision avoidance will drive on various road networks testing the flow of different intersection types. With the current state of the agents, realistic traffic flow is expected from this experiment.

Twenty agents are placed on a 4-way intersection. Each agent randomly decides their next turns. This might make some agents drive multiple circles on the roundabout. However, even with this issue, the flow still increases with the quality of the infrastructure. This final experiment is highly inspired by Euverus [16], who did this experiment using Cities Skylines.

Intersection Type Results

Based on **20 agents** driving in **4 directions**. Speed limit set to **50 km/h**
Measured between **7 to 40 seconds**.



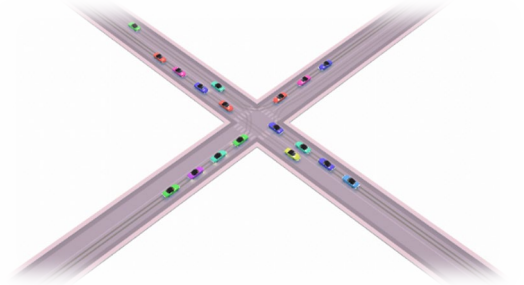
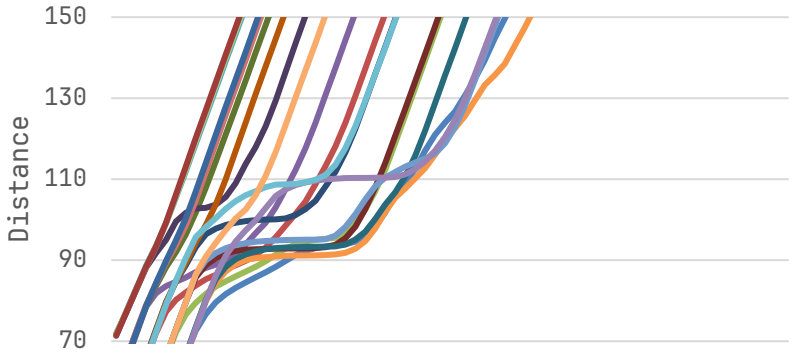


Figure 91. Simple Intersection: Agents take a long time to clear.



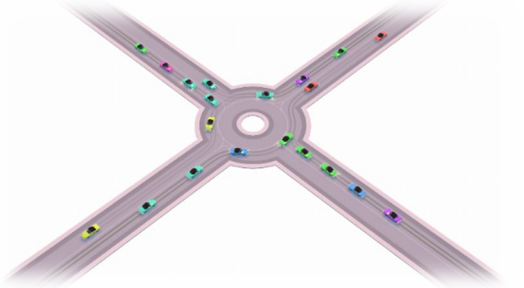
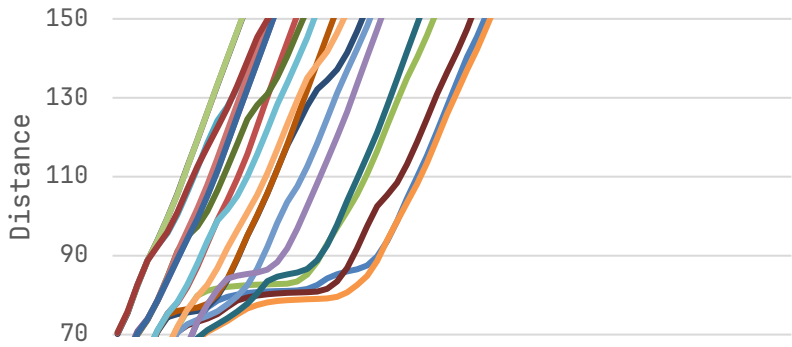


Figure 90. Small Roundabout: Quick driving with minor stopping.



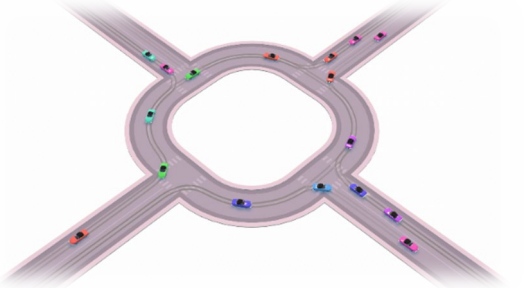
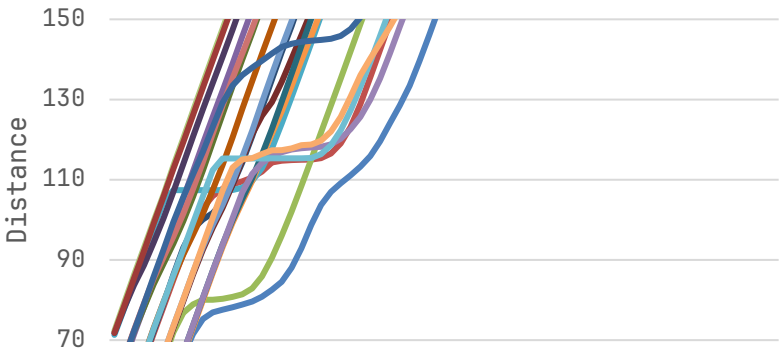


Figure 88. Large Roundabout: Quick driving with no stopping.



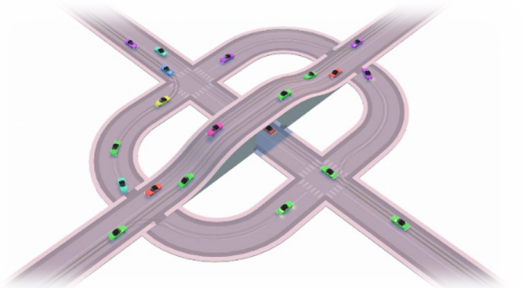


Figure 89. Diamon Interchange: Very quick driving.

Discussion

The results of this research highlight both the strengths and limitations of the implemented traffic simulation framework in relation to agent behaviors and traffic rules. The first key observation is that physics-based models compared to attached models created many challenges such as instability and excessive path deviations, as this model requires techniques used in real world self-driving vehicles. However, the physics-based model also provided a compelling foundation for realistic agent interactions, especially interactions with the surrounding world. An ideal solution would be to use a physics-based model when viewed from up close and an attached simplified model when far away as seen in the GTA series.

The PID controller emerged as a robust solution for controlling agent steering and speed. The results showed that tuning the proportional, integral, and derivative gains significantly impacted performance, allowing agents to adhere closely to paths even in less ideal conditions. Yet, these controllers struggled in scenarios requiring a smooth change in value on the far end as seen in “5.1 Speed limit” where, similar to the steering correction they required extra code to keep the agent smooth and stable.

Path smoothing techniques, such as lookahead interpolation, highlight the importance of predictive modelling. These approaches minimized abrupt directional changes and enhanced movement realism, particularly for curved road networks made of straight-line segments. Similarly, managing agent inertia with blended methods ensured smooth speed transitions while maintaining precise stopping at defined points.

The framework demonstrated its versatility, especially in intersection management. Adjusting stop-point distances allowed agents to approach and stop at intersections smoothly, avoiding abrupt halts seen in simpler implementations, like the framework used as a base for this project. This flexibility enhanced traffic flow and collision avoidance, as agents could begin braking earlier, and more effectively respond to intersection changes. Larger intersection triggers now allowed agents to decide to brake or drive earlier, giving them time to come to a smooth stop right at the intersection, instead of abruptly braking, and requiring a precisely placed trigger to define the stop position.

The stop-point mechanism was also instrumental in collision avoidance. By dynamically adjusting the stop point based on sensor feedback, agents could smoothly decelerate to prevent collisions without abrupt actions. Due to collision avoidance also using the same stop point mechanism the implementation was straightforward.

The final intersection test validated the framework's cohesiveness. Agents demonstrated realistic behavior, navigating intersections, roundabouts, and replicating phantom traffic jams. These results showcase a robust, physics-based traffic simulation capable of replicating real-world effects while remaining applicable to a gaming environment.

Conclusion

This study set out to answer the research question: "How are physics-based traffic simulations for games affected by agents and traffic rules?" Through rigorous testing and analysis, several key findings emerged that validate the formulated hypotheses.

Hypothesis 0 (Null) which posited that agents and traffic rules have no visible impact on the simulated traffic, was conclusively refuted. The results demonstrated that agent behaviors and traffic rules significantly shape the visible and measured aspects of a physics-based traffic simulation.

Hypothesis 1 (Agent Decisions) was supported by the observation that different agent decisions visibly influenced both speed and path deviation. The use of physics-based models combined with PID controllers allowed agents to follow paths with high accuracy. However, deviations in complex scenarios highlighted the importance of supplementary corrections, such as backward and instability controls, for maintaining stability and alignment.

Hypothesis 2 (Collision Avoidance) was validated, showing that effective collision avoidance is crucial to prevent agents from crashing into each other or physical obstacles. Without it, agents would drive unchecked, leading to collisions. Techniques like lookahead interpolation and adaptive stop points ensured agents could navigate smoothly, allowing for realistic phenomena like phantom traffic jams while avoiding crashes.

Hypothesis 3 (Intersection Flow) was also validated through experiments with different intersection types. The results illustrated how intersection rules, ranging from lane occupation to arrival priority, directly affected traffic flow efficiency and agent behavior. Each method had distinct strengths and weaknesses, reinforcing the importance of selecting appropriate intersection rules based on the desired simulation goals.

In conclusion, this study establishes that agents and traffic rules play pivotal roles in shaping physics-based traffic simulations for games. The insights gained not only provide practical methodologies for game developers but also contribute to the broader academic discourse on multi-agent systems.

Future work

The potential for further developing traffic simulation systems is vast, and several key areas can be explored in future work to enhance realism, optimize performance, and broaden the scope of traffic simulations. Below are a few important directions that could be pursued.

1. Special Scenarios

An important area for future traffic simulation research is the integration of special scenarios that can significantly impact traffic flow and agent behavior.

Police chases, for example, introduce high-speed pursuits that create dynamic traffic conditions. **Emergency vehicles** are another key scenario, where simulation of their interactions with other vehicles, as well as their impact on traffic, would make the simulation more realistic. Even **drunk drivers** would be a fun addition especially in a gaming environment.

Additionally, the inclusion of **vehicle collisions** would add complexity by introducing road blockages and delays, further challenging the traffic system.

The inclusion of **large vehicles** like trucks and buses in the simulation would present unique challenges. These vehicles often have different maneuverability, speed, and road usage compared to passenger cars, thus requiring tailored agent behaviors and pathfinding algorithms as some roads might not support those vehicles.

2. Car Lifetime Management

Another significant area for future development is the management of **car lifetimes** within the simulation. This includes the spawning and de-spawning of vehicles, which could be managed more efficiently with techniques such as occlusion culling. By only spawning agents in regions that are visible to the player, it would be possible to significantly reduce the computational load. This topic is broad and complex enough to warrant its own dedicated study and investigating how to handle vehicle lifetimes in the simulation would have a meaningful impact on performance and allow for more dense simulations.

3. Agent Specific Behavior

For the next iteration of the simulation, there is significant potential for improving agent-specific behaviors. Pathfinding algorithms can be introduced by providing **specific goals or jobs** for the agents. This would make the simulation more aligned with city-building games, where agents have a purpose beyond just traversing from one point to another. In contrast to open-world simulations, a more structured approach where agents have complex motivations could lead to a more dynamic and realistic traffic experience.

Additionally, introducing **lane-switching and merging behaviors** could allow agents to use multi-lane roads. This would allow the simulation to not be restricted to just single-lane roads, as opposed to the current setup. Additionally, it would require a more advanced road network.

There is also an opportunity to improve the driving strategies by implementing **counter-steering** to optimize vehicle trajectory. For example, agents could take wider corners similar to how race drivers adjust their steering, optimizing for speed and efficiency rather than simply reacting when at the end of a turn.

Splines can offer **smooth road networks;** this paper has deliberately excluded their use, as the focus here is on agent behavior rather than the road network itself. However, incorporating splines in future work would be beneficial, particularly in enhancing the road network and making it more refined. This work could go hand in hand with a more advanced road network.

Furthermore, agents could be programmed to **maneuver around obstacles** instead of simply stopping when encountering a blockage. Agents could track how long they have spent stuck and might attempt more aggressive approaches to getting unstuck, such as slamming into other drivers. This of course would not be ideal outside of the virtual world.

The steering input is based on a -1 to 1 value which, for the agent used for testing, maps to -50 to 50 degrees. This means **the values for the PID controller are actually based on a 50-degree angle.** Future work should use the angle of the agent directly for the error calculation instead of the steering input.

The performance **optimization** of all methods employed in this simulation has not been addressed at all and would be beneficial for a larger simulation.

Critical Reflection

This paper was quite the journey. I have never really written anything before, let alone a paper. This made the writing quite difficult. When I started writing this paper, I did not even know the meaning of a citation or a bibliography or any of the relevant terms for that matter.

I had difficulty starting and it felt like, every time I came up with a research question, after a few weeks of reflection, it again seemed impossible. This process kept repeating itself. The way I battled this difficulty is the way I have always done it: Just start making something. I knew I wanted to do something with traffic simulation for the reasons you will find in the preface. I have difficulty reading so I just started making a traffic simulation without much research beforehand. This was difficult, but I somehow managed to make a working but not so good traffic simulation within a few days. This alone allowed me to come up with so many possibilities when it came to testing and topics to write about. In the end, I ended up remaking it like described in the case study, but this shows that sometimes just doing and failing is much more effective than never starting at all.

Managing scope proved to be incredibly difficult, especially as someone who loves to just mess around with code and create things without much consideration. For instance, I added blinkers to the agents, and started constantly testing and playing around with the simulation, such as seeing if agents could also jump and still follow the path (they can).

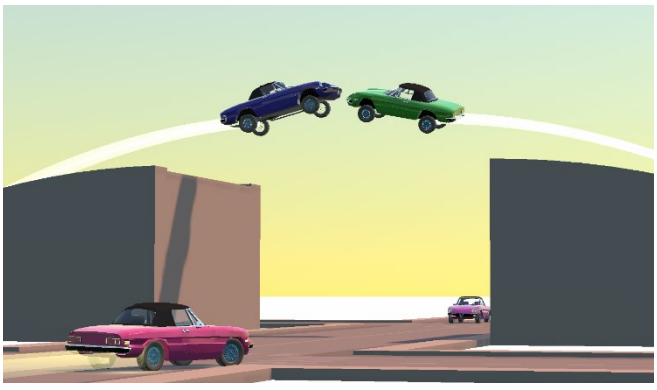


Figure 93. Agents jumping, because Julian apparently does not know how to focus

Figure 92. Agents using their blinkers because Julian once again does not know how to focus

In the end, I somehow managed to write all of it within the deadline (I say 2 days before the deadline), but I do firmly believe that over-scoping is a much better quality than under-scoping, as it has always left me with work I can be proud of. Writing the case study ended up being really fun and I noticed that it ended up almost like a blog post, which is not quite the goal, but it did allow me to stay motivated and I feel like it created a very fun blend between learning, explaining, and talking about test results. Writing in a scientific, formal way was quite the challenge and I hope to still improve on this.

I really enjoyed data collection for this project. I myself often make graphs plotting the electricity usage in my dorm. This paper really forced me to apply this to my creative process. It took quite some time to get the graphs into the paper, I actually initially started making them in Google Sheets and then later moved to Microsoft Word. This is why there is a sudden change in graph styles. The takeaway here is that I actually learned a lot about the performance of the simulation thanks to the graphs. It was quite difficult to spot the differences made by the varying types of intersections, or what the effects were of changing the PID controller values, and those graphs really helped with visualizing the change.

Much of my work consists of just making a game and presenting gameplay, and I have come to realize that presenting the creation of the work is also really important. I have always loved helping others learn and I am super happy to be able to have written a paper that hopefully can contribute to the greater game development industry, helping others get a better understanding. I might even attempt to write a blog post about the paper or even try making a YouTube series as this is always something I wanted to try.

Saying that writing a paper was difficult for me would be an understatement. As a bit of context, when I was just eight years old, I received a certificate confirming I have dyslexia. This document directly described that I have trouble keeping up with academic qualifications due to my predisposition and that my talent would be frustrated due to a failing lack of literacy. This, however, did not ever stop me from achieving my goals and therefore makes writing this paper quite the emotional and poetic end of my academic journey (for now).

References

The references have been documented in accordance with the IEEE citation style.

[1] *How SimCity Works - And Why Cities: Skylines Works Better*. Accessed: Dec. 31, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=AUFFrDkRw-w

[2] “Eclipse SUMO - Simulation of Urban MObility,” Eclipse SUMO - Simulation of Urban MObility. Accessed: Dec. 31, 2024. [Online]. Available: https://www.eclipse.dev/sumo/

[3] “Development Diary #2: Traffic AI,” Paradox Interactive Forums. Accessed: Dec. 31, 2024. [Online]. Available: https://forum.paradoxplaza.com/forum/threads/development-diary-2-traffic-ai.1591141/

[4] “Artificial intelligence in video games,” *Wikipedia*. Dec. 20, 2024. Accessed: Dec. 31, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Artificial_intelligence_in_video_games&oldid=1264116049

[5] “Traffic simulation,” *Wikipedia*. Nov. 20, 2024. Accessed: Dec. 31, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Traffic_simulation&oldid=1258492485

[6] K. Lahdenranta, “AI behavior in equal intersections : modification of Simulandia traffic AI.” Accessed: Dec. 31, 2024. [Online]. Available: http://www.theseus.fi/handle/10024/503767

[7] “Agent-based model,” *Wikipedia*. Nov. 25, 2024. Accessed: Dec. 31, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Agent-based_model&oldid=1259486239

[8] “Game theory,” *Wikipedia*. Dec. 27, 2024. Accessed: Dec. 31, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Game_theory&oldid=1265477029

[9] A. Champion, R. Mandiau, S. Espié, and C. Kolski, “Multi-Agent Road Traffic Simulation: Towards Coordination by Game Theory Based Mechanism,” Oct. 2001.

[10] L. Cortés-Berrueco, C. Gershenson, and C. Stephens, “Traffic Games: Modeling Freeway Traffic with Game Theory,” *PLOS ONE*, vol. 11, p. e0165381, Nov. 2016, doi: 10.1371/journal.pone.0165381.

[11] R. Mandiau, A. Champion, J.-M. Auberlet, S. Espié, and C. Kolski, "Behaviour based on decision matrices for a coordination between agents in a urban traffic simulation," *Appl. Intell.*, vol. 28, pp. 121–138, 2008.

[12] Cities: Skylines, *Traffic AI I Feature Highlights Ep 2 I Cities: Skylines II*, (Jun. 26, 2023). Accessed: Dec. 31, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=wgjpo2lKt7I

[13] "Free and open-source software," *Wikipedia*. Dec. 30, 2024. Accessed: Dec. 31, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Free_and_open-source_software&oldid=1266108823

[14] "ZoneGraph Quick Start Guide | Tutorial," Epic Games Developer. Accessed: Dec. 31, 2024. [Online]. Available: https://dev.epicgames.com/community/learning/tutorials/qz6r/unreal-engine-zonegraph-quick-start-guide

[15] CGP Grey, *The Simple Solution to Traffic*, (Aug. 31, 2016). Accessed: Dec. 31, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=iHzzSao6ypE

[16] euverus, *Traffic flow measured on 30 different 4-way junctions*, (Dec. 05, 2017). Accessed: Dec. 31, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=yITr127KZtQ

[17] J. Samyn, "Traffic Simulation Model integrated into Unreal Engine," Bachelor proef, Howest University of Applied Sciences, Digital Arts and Entertainment (DAE), Kortrijk, Belgium, 2024.

[18] "City Sample Project Unreal Engine Demonstration | Unreal Engine 5.5 Documentation | Epic Developer Community," Epic Games Developer. Accessed: Dec. 31, 2024. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/city-sample-project-unreal-engine-demonstration

[19] K. Lahdenranta, "AI Behavior in Equal Intersections," *Theseus*, 2021.

[20] "PlainXML - SUMO Documentation." Accessed: Jan. 04, 2025. [Online]. Available: https://sumo.dlr.de/docs/Networks/PlainXML.html

[21] "SUMO Road Networks - SUMO Documentation." Accessed: Jan. 04, 2025. [Online]. Available: https://sumo.dlr.de/docs/Networks/SUMO_Road_Networks.html

[22] "CARLA Simulator." Accessed: Jan. 04, 2025. [Online]. Available: https://carla.readthedocs.io/en/latest/

[23] “ASAM OpenDRIVE®.” Accessed: Jan. 04, 2025. [Online]. Available: https://www.asam.net/standards/detail/opendrive/

[24] Kenney, “City Kit (Roads) · Kenney.” Accessed: Dec. 31, 2024. [Online]. Available: https://kenney.nl/assets/city-kit-roads

[25] Freya Holmér, *The Continuity of Splines*, (Dec. 07, 2022). Accessed: Dec. 31, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=jvPPXbo87ds

[26] “Proportional–integral–derivative controller,” *Wikipedia*. Dec. 25, 2024. Accessed: Jan. 04, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Proportional%E2%80%93integral%E2%80%93derivative_controller&oldid=1265194949

[27] *Controlling Self Driving Cars*. Accessed: Dec. 31, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=4Y7zG48uHRo

[28] P. Polack, F. Altché, B. d’Andréa-Novel, and A. de La Fortelle, “The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?,” in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 812–818. doi: 10.1109/IVS.2017.7995816.

[29] “Kinematic Bicycle Model — Algorithms for Automated Driving.” Accessed: Jan. 08, 2025. [Online]. Available: https://thomasfermi.github.io/Algorithms-for-Automated-Driving/Control/BicycleModel.html

[30] “Ramer–Douglas–Peucker algorithm,” *Wikipedia*. Nov. 29, 2024. Accessed: Jan. 08, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm&oldid=1260206065

[31] TED-Ed, *Why the @#$% is there so much traffic? - Benjamin Seibold*, (May 28, 2020). Accessed: Dec. 31, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=TNokBgtSUvQ

[32] SotonTRG, *Phantom Traffic Jams*, (Jun. 30, 2011). Accessed: Dec. 31, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=Rryu85BtALM

[33] settimi_, *GTA V - NPCs drive off overpass and cause never-ending chain reaction explosions*, (Nov. 23, 2021). Accessed: Jan. 08, 2025. [Online Video]. Available: https://www.youtube.com/watch?v=lXAkCYsFfoo

Acknowledgements

I would like to express my deepest gratitude to several individuals and resources who contributed significantly to the completion of this work.

Firstly, I wish to extend my heartfelt thanks to my supervisor, **Vanden Abeele Alex**, whose insightful feedback and guidance was invaluable throughout the research process. His expertise and dedication played a crucial role in shaping this paper.

I am also grateful to my coach, **Geeroms Kasper**, for his support and advice on the reflection report. His assistance helped me better understand and articulate my experiences during this project.

Special thanks to the creators of the "unity-traffic-simulation" package (available at https://github.com/mchrbn/unity-traffic-simulation). This resource was instrumental in providing a solid foundation for developing my own traffic simulation.

The principles of PID-Control have been thoroughly researched, and I would like to express my gratitude **AerospaceControlsLab** [27] for their valuable insights into its implementation-specific detail.

Additionally, I would like to acknowledge the support and feedback provided by my peers **Huens Nigel**, **Weel Bouke**, **Horrie Finian**, **Heyde Robbe**, **Ceulemans Yarno,** and **Devred Matías**. Their collaboration and insights were immensely helpful during the development of my first paper. I would also like to express my deepest gratitude to **Vandamme Saar** for providing the car model seen in almost every figure. Their contribution significantly enhanced the realism and quality of the simulations. And of course, **my mom** for helping me get started on the writing. I would also like to acknowledge the **Coffee.**

To all those who contributed directly or indirectly to this work, I extend my sincere thanks. Your help and encouragement are greatly appreciated.

Appendices

1. Online Repository

The traffic simulation software developed for use in testing was created using **Unity** and **Git**. The repository is available at:
https://github.com/JulianRijken/TrafficSimulation

2. Disambiguation

AI “Artificial intelligence”

In the broadest sense, is intelligence exhibited by machines, particularly computer systems. For our purpose we will not be talking about LLM's but about more simple AI regularly uses in games for NPC

NPC "non-playable character"

Refers to characters or in a game that cannot be played meaning that they are controlled by the computer. The term is generally used to define objects in a game that are controlled by the game.

LLM "large language model"

Is a type of artificial intelligence algorithm that uses deep learning techniques and massively large data sets to understand, summarize, generate and predict new content. The term generative AI also is closely connected with LLMs, which are, in fact, a type of generative AI that has been specifically architected to help generate text-based content.

ITS "Intelligent Traffic System"

A Unity package that uses true physics driven cars to simulate traffic around the player which gives a realistic feel of traffic simulation, also pedestrians can be simulated on the sidewalks and several other kind of vehicles.

Traffic simulation

Or the simulation of transportation systems is the mathematical modeling of transportation systems (e.g., freeway junctions, arterial routes, roundabouts, downtown grid systems, etc.) through the application of computer software to better help plan, design, and operate transportation systems.

Game theory [8]
is the study of mathematical models of strategic interactions. It has applications in many fields of social science, and is used extensively in economics, logic, systems science and computer science.

ABM “Agent-based modeling” [7]
Agent-based modeling is a computational approach used to simulate the interactions of autonomous agents in order to assess their effects on the system as a whole. Each agent in an ABM operates based on predefined rules and can interact with other agents and the environment, leading to emergent behaviors. ABM is widely used in various fields, including traffic simulation, economics, social sciences, and ecology. In the context of traffic simulations, ABM can model individual vehicles or pedestrians as agents that follow specific behaviors, such as adjusting speed based on traffic conditions, making decisions at intersections, or interacting with other agents. This approach helps to better understand complex systems and predict outcomes in dynamic environments.

GTA “Grand Theft Auto”
A popular open-world action-adventure video game series developed by Rockstar Games. The series is known for its immersive urban environments, dynamic gameplay, and the integration of missions, vehicles, and NPCs. Traffic and pedestrian simulations play a crucial role in enhancing the realism of its game worlds.

Cities Skylines
A city-building simulation game developed by Colossal Order and published by Paradox Interactive. It allows players to design, build, and manage cities with detailed traffic systems, zoning, public services, and urban planning tools. Cities Skylines is widely regarded as a benchmark for realistic traffic simulation in video games.

SimCity
A pioneering city-building simulation game where players design, build, and manage a city. The game includes systems for managing urban infrastructure, including transportation and traffic networks, and has influenced many subsequent traffic simulation games.

Open-world games
Games with large, freely explorable environments where players can interact with the world and its systems at their own pace, without fixed objectives.

These games often feature dynamic traffic, AI-driven NPCs, and complex, evolving systems that enhance realism.

FOSS “Free and Open Source Software” [13]

Refers to software that is both free to use and modify, with its source code openly available to the public. This allows users to study, improve, and distribute the software. FOSS promotes transparency, collaboration, and innovation, and it is commonly used in various applications, including operating systems, game development, and productivity tools. Examples include Linux, Blender, and Godot Engine.

SUMO “Simulation of Urban MObility” [2]

Is an open-source traffic simulation software that models traffic patterns, pedestrian movements, and other elements of urban transport systems. It allows for detailed traffic analysis, including vehicle behavior, signal timings, and congestion modeling. **Sumonity** refers to a Unity integration of SUMO, enabling the simulation of realistic traffic and transportation systems within Unity, making it easier to implement traffic simulations in game environments or research applications. It provides a framework for combining traffic modeling with game engines for enhanced realism in simulations.
https://eclipse.dev/sumo/

Bezier Curves / Splines

Are mathematical curves used in graphics, game development, and pathfinding algorithms. Bezier curves are defined by control points and provide smooth, continuous curves. Splines, which include Bezier curves, are used to represent paths, roads, or routes, and are commonly used to model the movement of vehicles or NPCs in simulation games to ensure fluid motion.

A* “A-Star”

Is a widely used pathfinding algorithm that helps find the shortest path between two points in a grid or graph. It combines aspects of Dijkstra's Algorithm and a heuristic approach, making it efficient for real-time applications like games, where it can dynamically calculate optimal paths for NPCs or vehicles in response to changing environments.
Dijkstra
Is an algorithm for finding the shortest paths between nodes in a graph, particularly useful in situations where all edges have the same weight or cost. It’s widely used in navigation systems, map routing, and simulations for determining the quickest route between points, though it is less efficient

than A* for games with more complex environments due to its exhaustive nature.

Game Theory
A field of study concerned with mathematical models of strategic interaction between rational decision-makers. It is applied across various disciplines, such as economics, political science, and computer science, to understand and predict outcomes of competitive and cooperative behaviors.

Nash Equilibrium
A concept in game theory referring to a situation in which no player can improve their outcome by unilaterally changing their strategy, assuming the strategies of the other players remain unchanged. This equilibrium is a fundamental idea in strategic decision-making.

Payoff Matrices
A tool used in game theory to represent the outcomes of different strategies in a game. The matrix provides a structured way to analyze the payoffs or rewards each player receives based on the combination of strategies chosen by all participants in the game.

Proportional-Integral-Derivative Control
PID-Control is a feedback mechanism that adjusts system inputs to minimize error by combining three components:

- **Proportional** (P): Reacts to the current error to provide immediate correction.
- **Integral** (I): Addresses past accumulated errors to eliminate steady-state offsets.
- **Derivative** (D): Anticipates future errors by responding to the rate of error change, helping stabilize the system.

In Steering Cars PID-Control is used to adjust the steering angle for following a desired path or trajectory, such as a lane or racing line.

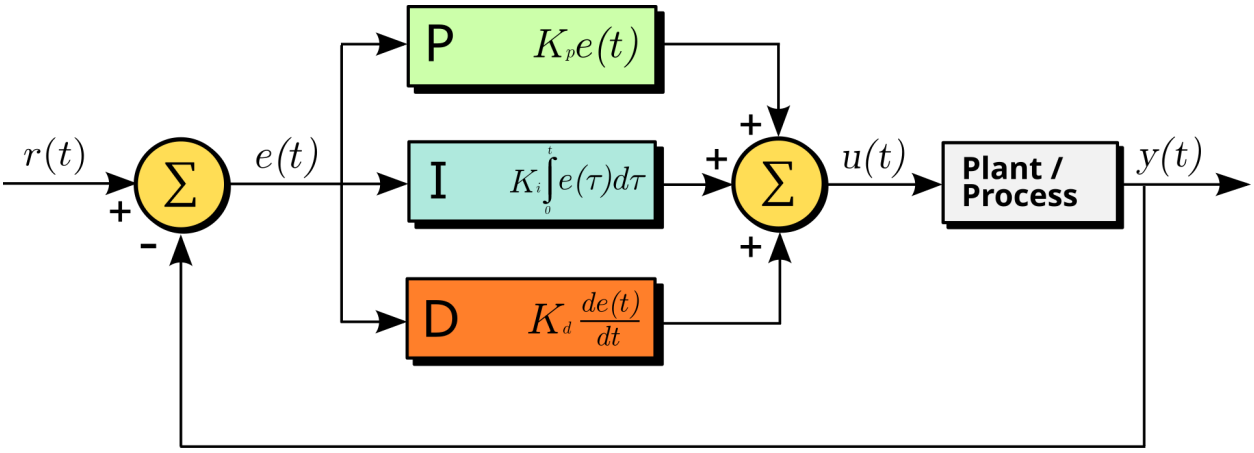


Figure 94. PID controller overview. By Arturo Urquizo - File:PID.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=17633925