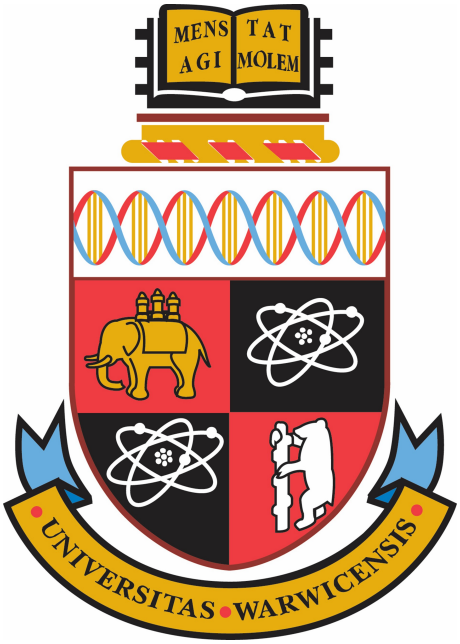
An Assessment of the Potential Behaviours of Self-Driving Cars

CS310 Third Year Final Report



Author
Luke Guppy

Supervisor
Andrew Hague

University of Warwick
Department of Computer Science

2024

Abstract

This project explores an implementation of curriculum learning to train a self-driving car agent within a simulation environment. The aim is to enable the agent to navigate an environment while following traffic rules and avoiding collisions. Through the use of Unity simulation and the ML-Agents package, the project focuses on developing an agent capable of path following, lane centring, responding appropriately to traffic lights, and environment cars.

Throughout the development process, multiple stages of reward systems were devised and iteratively refined to incentivise desired behaviours, ultimately resulting in a successful framework that allows for incremental introduction of the named behaviours. The resulting agent demonstrates a sufficient reward system where the said behaviours are applied to develop a complex agent capable of navigating the environment.

This project contributes to the advancement of autonomous car technology and machine learning by providing insights into the effectiveness of curriculum learning methodologies for training self-driving car agents. The modular nature of the reward system and simulation allows for flexible implementation of behaviours and evaluation of various training strategies. It gives a strong insight as to how a reward system can be developed while giving an understanding of the limitations of the Deep-Reinforcement Leaning system applied.

Keywords: Autonomous cars, Reinforcement learning, Simulation, Curriculum learning, Reward systems

Acknowledgements

I would like to thank my supervisor, Andrew Hague, for his valuable insight and guidance in completing this project, providing support and assistance which was crucial to its success.

Contents

1 Introduction to the Project

1.1 Refined Problem Statement and Addressed Gap

In the world of deep-reinforcement learning (DRL), specifically in the field of autonomous cars, a gap exists in research of how to effectively develop a comprehensive reward system that allows a cumulative approach to implementing driving behaviours by autonomous agents. While existing research has made significant strides in designing reward systems tailored to specific functionalities, there remains a notable gap in the research concerning the creation of a cohesive framework for integrating various behaviours and facilitating agent learning systematically. Although the field of autonomous driving is one of the largest growing industries in the world, "there are a few successful commercial applications, there is very little literature or large-scale public datasets available" [1]. This is an incentive for more research into how DRL can be applied to implement driving behaviours.

This project seeks to address this gap by making use of curriculum learning principles to train autonomous agents within a simulated environment. The refined problem statement centres on the design and implementation of a holistic reward system capable of accumulating a range of driving behaviours, including lane navigation, traffic light interactions, obstacle avoidance, and path following.

Unlike traditional approaches, which focus on isolated features and corresponding reward functions, this project aims to establish a structured methodology for accumulating rewards and incrementally building up the capabilities of autonomous agents. By adopting a curriculum learning framework, the project aims to provide a scaffold for DRL, enabling agents to progressively achieve complex driving behaviours.

Through this approach, the project seeks to lay the foundation for the creation of an adaptable and resilient autonomous agent capable of integrating new behaviours and functionalities as it evolves.

1.2 Motivation

The motivation behind this project is rooted in the potential of self-driving cars, which stand at the forefront of modern innovation in technological advancements. With the promise of enhanced safety, efficiency, and accessibility: autonomous cars represent a shift in the way we perceive and interact with transportation systems [2].

A key driving force behind this project is the progression and advancements in the field of DRL. Building upon the foundations of traditional reinforcement learning, DRL has emerged as a powerful paradigm for training autonomous agents to make complex decisions and navigate complex environments. The integration of deep neural networks with reinforcement learning algorithms has unlocked new possibilities for training agents to exhibit human-like behaviours and adaptability [3].

The field of artificial intelligence (AI) and DRL has provided a unique opportunity to overcome the inherent dangers posed by current road environments, characterised by diverse and unpredictable traffic scenarios, road conditions, and human behaviours. The prevalence of accidents and road-related fatalities underlines the need for an understanding of how to develop robust and reliable autonomous driving systems that can navigate safely and effectively [4]. By leveraging advancements in reinforcement learning and curriculum learning methodologies, this project aims to address these challenges and provide an understanding of how to develop such systems not only in the field of autonomous driving but for any DRL agent.

Curriculum learning offers a promising structure for training autonomous agents by breaking down complex tasks into manageable sub-tasks to accumulate behaviours and adapt to increasingly complex environments [5]. This learning methodology allows for better adaptability and progression where previous learning methods have struggled to implement complex behaviours.

1.3 Objectives

- **Develop a Realistic Simulation Environment:** Create a Unity-based simulation environment that abstracts real-world driving scenarios, focusing on essential features while omitting minor details.
- **Design and Implement Learning Agents:** Develop a reinforcement learning agent, ensuring it can navigate the simulation environment effectively.
- **Conduct In-Depth Research:** Carry out comprehensive research on AI-driven cars, traffic dynamics, and human behaviour to inform the development of the simulation and learning agent.
- **Training and Performance Analysis:** Train the agent and conduct rigorous performance analysis on the relevant features.
- **Varied Situations:** Provide a series of different situations for learning (e.g., traffic lights, cross junctions, T-junctions, pedestrians) to challenge the agent to learn more general and applicable behaviours.
- **Adapt Agent Based On Feedback:** Tweak and adjust the learning agent's rewards depending on its results to better improve its learning capability.

In adapting the core objectives to better suit the specific focus of the project, slight adjustments were made to the key definitions to better align with its goals. While developing a Unity-based simulation environment with a learning agent navigating it, the project's scope shifted towards a more targeted exploration of curriculum learning for self-driving cars. Rather than aiming for an exploration of human-AI interaction, the project concentrates on analysing the implementation of behaviours using curriculum learning techniques.

The slight impact this had on the objectives was a change in the measure of success from being based on safety, speed, and efficiency to a focus on the training quality of the agent. This shift focused the project more on how to effectively implement specific behaviours, as opposed to analysing general driving.

2 Background Research

2.1 Introduction to Self-Driving Cars and Curriculum Learning

2.1.1 Benefits of Autonomous Cars

Safety is a primary motivation for the development of autonomous cars. Research has shown that human error is the leading cause of road accidents, with factors such as distraction, fatigue, and impaired driving contributing to the majority of collisions [6]. Autonomous cars have the potential to reduce accidents by eliminating human error.

Efficiency is another advantage of autonomous cars. By optimising driving patterns, reducing traffic congestion, and minimising unpredictable behaviour, autonomous cars can improve the overall efficiency of road systems while simultaneously reducing emissions through optimised driving.

Another key consideration is accessibility in transportation planning, and autonomous cars have the potential to improve mobility for individuals who are unable to drive due to disabilities, age, or other reasons. The success of autonomous agents, for people with disabilities, has been proven in tests where "computers are in control about 90% of the time, and they've given 5,000 rides since 2022 without any accidents" [7]. By providing reliable and convenient transportation options, autonomous cars can improve transport accessibility, particularly for those who struggle to access public transport [8].

2.1.2 Training Challenges

Training self-driving car agents poses a multitude of challenges. Research shows the complexities involved in accurately perceiving and interpreting the surrounding environment, including detecting and classifying objects, understanding road signs, and predicting the behaviour of other road users [9].

Real-world environments encompass varying weather conditions, which impact the performance of the sensors. For instance, lidar, a popular remote sensing method used in the development of autonomous vehicles [10], struggles with the diffraction of light through rain, fog, and snow, and most testing for autonomous vehicles is currently performed in sunny environments [11].

Additionally, diverse road infrastructure, such as different road markings, signage, and traffic patterns, adds to the complexity of navigation and decision-making for self-driving car agents. Autonomous vehicles require very precise maps of the street and surroundings, which need to be much more detailed than current map services [12]. An alternative is using advanced machine learning techniques to categorise road infrastructure accurately [13], which is currently a more commonly implemented practice.

Furthermore, human behaviour introduces uncertainty into the driving environment, as drivers may exhibit unpredictable actions and behaviours on the road. Another consideration is that "non-human-like" behaviour may be surprising to other road users with potential negative safety consequences [14].

2.1.3 Curriculum Learning

Curriculum learning, proposed by Bengio in 2009 [15], acknowledges that learning complex tasks from scratch can be challenging for machine learning models. By presenting training examples or tasks in a structured curriculum, starting from simpler instances and gradually increasing complexity, curriculum learning provides a scaffolded learning environment that enables an effective accumulation of knowledge and skills.

The rationale behind curriculum learning lies in its ability to learn in a manner that mimics human learning. Humans learn new concepts or skills by initially tackling simpler tasks before progressing to more challenging ones [16]. Similarly, curriculum learning can be used to train agents to achieve complex behaviours by exposing them to a sequence of increasingly challenging tasks. As the agent demonstrates proficiency in simpler tasks, the complexity of the tasks gradually increases, allowing adaptive learning and enabling an agent to tackle more complex scenarios [17] [18].

In the context of this project, the agent is trained to achieve basic manoeuvres such as path following, and gradually progress to handle complex situations such as traffic lights and avoiding collisions with other cars. This approach enables targeted training of specific driving behaviours, learning one behaviour at a time.

A key reason why curriculum learning is highly-regarded in DRL is its capability of learning without getting stuck in local maxima (suboptimal solutions that the learning algorithm may converge to). This is because behaviours are learnt in smaller, easily defined stages rather than a larger, more complex reward system, which can be difficult to navigate and converge on.

2.2 Related Work

In the field of AI driving, researchers have explored various methodologies and strategies to address the challenges of creating complex agents. Studies in the field provide valuable insights into methodologies and reward systems that can be used to effectively implement specific driving behaviours.

2.2.1 Simulation to Real-world

Simulation-to-real (sim2real) training methods represent a key aspect of autonomous driving research. These, involve training theoretical models in virtual simulation environments and transferring the learnt behaviours to physical cars. By bridging the gap between simulated environments and real-world applications, such approaches emphasise the importance of generalisation and safety in autonomous driving systems [19]. This approach is common practice for the training of self-driving cars, allowing for extensive testing and refinement in a controlled virtual environment before deployment in real-world scenarios.

A sim2real study exploring lane-following is that of [20] which introduces a progressively optimised reward function using a form of DRL known as Deep Deterministic Policy Gradient (DDPG). The aim of the reward function was to implement driving within lane boundaries, where the agent was trained in a simulated environment and its learnt behaviour were translated to the real world. This offers a practical example of the potential of DRL algorithms and how virtual training can translate to a real-world application. However, this example lacks the depth of complexity as only one behaviour of lane-following was trained. This project, on the other hand, aims to introduce a larger range of behaviours within the virtual training environment.

2.2.2 Driving in Junctions

Existing research in training specifically junction scenarios, shows potential issues with implementing DRL with self-driving agents. In [4], the implementation of current DRL systems displayed issues with the agent "freezing", and a significantly high collision rate. This study compared 3 common systems for implementing DRL. Most notably, it compared a deep q-learning approach with the Proximal Policy Optimisation (PPO) system used in this project. In this comparison, PPO showed worse results with a 35% collision rate, 7% higher than the proposed model. While this study shows successful results of a DRL implementation, the failure of PPO could be due to a limited implementation, and this project aims to successfully implement a PPO model, testing its effectiveness in a more dynamic situation.

Another key note of this study is that it lacks elements fundamental to driving. Most importantly, it does not address dynamic actions, as its action space is limited solely to acceleration. A diverse action space is crucial for adapting to changing scenarios on the road, rather than being limited to purely speed control. Additionally, the article focuses on standard DRL techniques for implementing just one reward function and does not explore incrementally introducing additional behaviours through curriculum learning.

2.2.3 Obstacle avoidance

Another specific driving task commonly explored is that of obstacle avoidance, as demonstrated in [21]. The study explores the effectiveness of DRL, specifically deep q-network, agents for navigating urban environments while avoiding randomly generated obstacles. The study adopted an action space of three actions: no change in direction, a 10° turn to the left, and a 10° turn to the right. While showcasing

the potential of training self-driving agents for detecting and avoiding obstacles, it is limited by its constrained action space and lack of variety in encountered situations and does not reflect the depth or dynamic nature of real driving or car control.

2.2.4 Reward Functions

One of the most critical areas of DRL research is that of designing effective reward systems, as displayed by [22]. By balancing collision avoidance and maintaining desired speeds, the study explores strategies for navigating roads which lack lane structure. While it is once again a specialised situation, with the agent aiming purely to drive down one stretch of lane free road, this study provides a foundation for how to approach designing a reward system to encourage specific behaviours. Specifically, the study incorporated various terms within the reward function, each aiming to implement a desired behaviour based on distances to relevant vehicles. This inspired the design for an incremental reward function which could be more effectively implemented with curriculum learning.

2.2.5 Summary

Collectively, these studies contribute to the body of knowledge about autonomous driving agents, providing methodologies for developing specific driving systems. However, they generally lack the depth of information about how to develop a more complex system, reflecting the accumulation of multiple behaviours required to achieve a complex agent.

For this reason, this project aims to create a foundation where progressively complex behaviours can be accumulated and trained using PPO to create a driving agent with the potential addition of any number of desired behaviours.

2.3 Extending AI-Driven Car Research

Extending the existing areas of research, this project implements an agent that successfully achieves a complex series of behaviours. The agent learnt to drive in a virtual environment that is designed to be a realistic abstraction of the real-world, where features can be added as the agent progresses in development.

2.3.1 Road Sections with the Highest Collision Rates

The environment has a specific focus on scenarios with the highest real-world collision rates. By focusing on these critical driving situations, this project provides a foundation for how self-driving agents can incrementally learn behaviours to navigate environments that are at the forefront of causing human casualties. Road sections that cause the highest rate of serious collisions are junctions, specifically cross-junctions and T-junctions [23]. Implementing and exploring these specific sections provides a challenging training environment for a self-driving agent to navigate.

2.3.2 Existing Reward Functions for Driving Simulations

The development of reward functions is fundamental for training driving agents. The models explored in Section 2.2 and numerous other studies have explored various reward functions tailored to specific driving scenarios. These functions play a vital role in shaping the behaviour of agents, such as safely following other cars and adhering to traffic rules. It is often difficult to find the right balance to imitate human behaviour, as small adjustments can make the agent too passive or aggressive and, as a result, inefficient or dangerous [24].

A key aim of reward functions is to implement speed and path following behaviours. This is a largely explored area, with many studies specifically focusing on racing lines and optimal driving for speed [25]. Even though in this project the agent is not in a racing scenario, these implementations give good insight as to how to approach corner turning for an autonomous agent and specifically how to perceive a corner using current and future targets to informatively analyse a turn [1].

2.3.3 Sensory Inputs and Visualisation of the Environment

The integration of sensory inputs is imperative for self-driving cars to navigate their environments. Research provides a comprehensive overview of current self-driving systems, emphasising the importance of sensory data processing and environment visualisation. By using sensor technologies, agents can gather real-time information about their surroundings and make informed decisions [10]. Additionally, visualisation methods enhance situational awareness, enabling systems to interpret traffic scenarios and respond appropriately to changes in the environment.

In the context of real-world self-driving cars, the implementation of these observations varies from more expensive lidar, sonar, and radar technology [10], to a simple accumulation of cameras and machine learning such as Tesla cars, the currently most successful autonomous vehicles, "which are not equipped with radar and instead rely on Tesla's advanced suite of cameras and neural net processing" [26]. Ultimately, all techniques result in an accumulation of the same information with varying accuracy and prices.

2.3.4 Inputs and Outputs of Neural Networks for Learning in Automated cars

Neural networks (NNs) serve as fundamental components of learning agents, processing input data and outputting actions. Existing research extensively explores NN architectures tailored to collision avoidance and general driving tasks [27]. These studies explore the role of NNs in processing sensory inputs, such as car trajectories and environmental data, to generate appropriate action outputs.

By understanding common sensory inputs for NNs in autonomous driving (as discussed in Section 2.3.3), the agent can be given a model reflecting information observable in a real-world scenario. In the case of this project, these systems will be simplified by assuming perfect knowledge of the environment, as sensing systems are in themselves a separate branch of machine learning in autonomous driving.

In this project, the sensory inputs vary with each curriculum but include features such as information about the centre of the lane, the edge of the road, the relative velocity of other cars, the agent car itself, and other features to create a holistic understanding of the environment.

2.3.5 Relationships Between AI and Human Behaviours

For an agent, "ideal behaviour" is a relatively abstract and subjective topic and is approached differently with varying implementations: whether it is to follow some set of rules or to imitate human behaviour gathered from a large data set, such as how Tesla processes the data of its users to train autonomous cars in fleet learning. Specifically, Tesla makes use of "48 (neural) networks that take 70,000 GPU hours to train. Together, they output 1,000 distinct tensors (predictions) at each timestep" [28]. The use of 48 individual NNs emphasises the complexity of environment sensing and shows the vast array of systems that make up a real-world self-driving agent.

As mentioned in Section 2.3.3, finding the balance between confidence and hesitancy is a crucial factor for how successful an agent is: a difficult balance which even the most advanced self-driving systems struggle with. Junctions are massively affected by this balance and small adjustments can drastically affect efficiency of the agent [24]. However, this problem is not so significant in an environment controlled by traffic lights which guide the cars and thus reduce the consequence hesitancy. A "smart" traffic light system could replace the need for these decisions while maintaining efficient road systems.

2.3.6 Summary

In summary, this analysis of the field of autonomous driving and DRL implementations gives an overview of the current state of autonomous driving research. By focusing on driving scenarios with high collision rates and refining relevant reward functions, the project provides insights into creating a functional agent. It builds on existing knowledge with a progressive reward system, as other studies often lack complexity or solely focus on a specific feature of driving.

3 Technical Aspects

This section explains the technical features of the project, focusing on the tools and methodologies used in implementing a functioning curriculum DRL framework and the simulation environment.

3.1 Unity

Unity serves as the primary platform for constructing the 3D simulation. Using Unity's robust framework, a realistic environment is created to mimic an abstraction of a real-world driving scenario, including road networks, traffic signals, other cars, targets for path following, raycasts for object detection, and other key components. Unity's extensive library of assets allows for easier tailoring of a training environment and implementation of the agent.

Unity was chosen as the platform for this project due to the vast amount of online resources and support for understanding and using the system. It allowed the implementation of the system without the need to create a physics system or other non-unique features from scratch. Unity also provides the opportunity to use ML-Agents, a package specifically designed for implementing DRL in a Unity environment [29]. For these reasons, it was chosen as the system used to implement the simulation and agent.

Key features are Unity's car controller standard asset, along with a free car model asset and road prefabs to visually represent the driving scenario. The car controller provides a framework for implementing car physics and controls, allowing accurate simulation of a car's behaviour within the environment with realistic braking, accelerating, suspension, and handling.

Specifically, Unity version 2022.3.10 was used during this project. This choice ensured compatibility with the car model, as it the version in which it was designed, and guaranteed a stable and tested version of ML-Agents avoiding any potential bugs with newer versions of Unity.

The road prefabs were used to construct road infrastructure within the environment providing a convenient way to layout roads, traffic signals, and other essential features. By assembling these prefabs, an environment was created resembling an abstraction of a real-world driving environment.

Additional features could be added to the road prefabs to create a more complex environment. These additional features include hit boxes for road detection, targets for path following, and the traffic system, all of which are controlled through the use of C# scripts.

Overall, the use of Unity enables a representation the world with a high degree of physical realism. This provides a solid foundation for training agents to navigate complex driving scenarios in an environment reflective of the real-world such that it could, in the future, be transferred in a sim2real system.

3.2 Understanding Deep-Reinforcement Learning

Deep Reinforcement Learning (DRL) represents an approach to machine learning combining principles from deep neural networks and reinforcement learning [30]. By joining these powerful techniques, DRL enables agents to learn complex tasks directly from sensory inputs without the need for handcrafted features.

3.2.1 Deep Neural Networks

A neural network (NN) is a computational model inspired by the structure and function of the human brain [31], designed to learn from data and perform tasks such as classification and pattern recognition. It consists of interconnected nodes, or neurons, organised into layers. In a feed-forward NN, as used in this project, information flows from the input layer through one or more hidden layers to the output layer. Each neuron receives input signals, processes them using an activation function, and generates an output that is passed to the neurons in the subsequent layer. Through training, a NN adjusts the weights between neurons to improve its ability to make accurate predictions on new, unseen data.

A Deep Neural Network (DNN) is a specific type of NN with a greater depth. A DNN includes at least three hidden layers (not input or output layers) within the NN allowing the ability to find more complex patterns between the input features, resulting in a more advanced and complex system. However, as proven by countless implementations of DNNs, higher complexity results in a reduction in training speed [32].

The DNN serves as the foundation for this implementation of DRL. In the context of DRL, specifically PPO (section 3.3.2), DNNs are typically used to approximate value functions or policy functions, allowing agents to make decisions and take actions based on sensory input while attempting to predict future rewards as to best guide its decision-making.

3.2.2 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning concerned with learning optimal decision-making policies through interaction with an environment. In RL, an agent learns to maximise a cumulative reward by taking actions in an environment and observing the resulting state and rewards. RL algorithms enable agents to learn optimal policies through trial and error, with improvements based on a reward system defined by the developer. This reward system defines how well the agent is performing while the agent improves by maximising the reward accumulated during an episode.

3.2.3 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) combines the power of DNNs with the decision-making capabilities of RL algorithms. In DRL, DNNs can be used to approximate value functions, policy functions, or both. By using deep learning techniques, DRL enables agents to learn from low-level representations of the environment and make informed decisions based on observed states. The integration of deep learning with RL has led to significant advancements in AI across various domains, including robotics [33, 34], gaming [35], and of course self-driving cars.

In the context of self-driving cars, DRL offers a powerful methodology for training an agent to drive in real-world environments autonomously. In this project, DRL is used to train an agent to handle driving scenarios, from basic lane following to complex manoeuvres at traffic light intersections involving other cars. This approach allows for adaptive behaviour, where the integration of DRL represents the real-world capabilities of autonomous cars.

3.3 ML-Agents

ML-Agents serves as a powerful bridge between machine learning algorithms and Unity-based simulations. Developed by Unity itself, this open-source python based package is tailored to integrate deep reinforcement learning into Unity environments [36].

At the core of ML-Agents is a Reinforcement Learning (RL) framework. This framework allows the training of intelligent agents, guided by a self-implemented reward system. In this project, the agent's aim is to accumulate the highest reward it can. However, the success is strongly dictated by the implementation of the reward system which drastically impacts the behaviours it learns.

3.3.1 Neural Network Policies

ML-Agents employs the use of a DNN to model the policies of the agent. The DNN adapts and refines its parameters during training to optimise the decision-making process of the agent. In this project, the DNN represents the policy for the agent to follow desired driving behaviours as the curriculum progresses. This is governed by a series of sensory inputs such as information about the current target, other cars, the lane positioning, and other inputs defined in Section 2.3.3. The policy takes a state representation of its environment and returns a vector of output values which represent the agent's actions.

3.3.2 PPO

In the context of ML-Agents, the machine learning algorithm applied is Proximal Policy Optimisation (PPO), a DRL algorithm. Introduced in 2017 by OpenAI [37], PPO is a reinforcement learning algorithm designed to optimise policy functions in an iterative manner. It strikes a balance between exploration and exploitation, allowing agents to learn optimal policies through trial and error without diverging quickly away from current configurations which could otherwise result in instability.

At the core of PPO lies two fundamental components: the value network and the policy network. The value network is responsible for estimating the expected reward of taking a series of actions in a given state. This estimation helps the agent assess the potential consequences of its actions and inform its decision-making process. On the other hand, the policy network determines a probability distribution over actions given the current state. Based on the future prediction, it adjusts over time to produce optimal actions and accumulate a larger future reward. By learning a policy that maximises expected rewards while ensuring stability, PPO aims to achieve effective reinforcement learning in complex environments.

PPO differs from other DRL systems due to its use of a trust region, which ensures policy updates occur within a constrained area to maintain stability. The trust region keeps the new policy proximal, or close, to the previous one by constraining the magnitude of policy updates during training. This facilitates a gradual learning process and prevents destabilising the policy with drastic changes [38]. Such a careful balance is effective for learning complex driving scenarios without adopting unpredictable behaviours.

PPO is one of the two algorithms implemented by ML-Agents chosen based on research at Cornell University: "Our experiments test PPO on a collection of benchmark tasks, including simulated robotic locomotion and Atari game playing, and we show that PPO outperforms other online policy gradient methods, and overall strikes a favourable balance between sample complexity, simplicity, and wall-time" [38]. This gives a reliable foundation for how PPO is effective in similar complex environments. The alternative algorithm available in ML-Agents is Stochastic Actor-Critic (SAC) which is a different structure of DRL based on encouraging exploration of states. However, this project uses PPO due to numerous projects which reflect a similar complexity and have applied it to create successful agents, and SAC is more computationally intensive.

3.3.3 Hyperparameters

The technical complexities also extend to hyperparameter tuning, where factors such as learning rates, discount factors, and exploration parameters optimise the training process. These can have a large impact on the quality of training and can be responsible for the success or failure of an agent. Details of the specific parameters and implementation will be discussed in Section 5.2.3.1.

3.3.4 Non-Linearity in ML-Agents

Activation functions are essential components within neural networks. These functions introduce non-linearity, which is crucial for capturing complex patterns in agent behaviours. By performing a non-linear function on input signals, activation functions enable the network to find complex patterns between observations to implement advanced behaviours. Without them, the network's capacity would be limited to linear mappings, hindering its ability to find complex relationships.

ML-Agents predominantly employs the Swish activation function for the hidden layers in its DNNs. Swish is an activation function introduced in 2019 by Google Brain [39], that implements non-linearity by smoothly blending the linear and nonlinear activations. Specifically, it is defined as $x * sigmoid(x)$ and avoids previous issues with other activation functions such as the vanishing gradient problem, which is the key issue for functions such as $sigmoid(x)$ [40]. This problem occurs when the gradient tends to 0 with extreme values, causing difficulties in gradient descent for learning and progressing towards the local maximum. It is an issue for many common activation functions.

Swish allows neurons to output the relation with a smooth curve, promoting stronger gradients and faster convergence during training. It was chosen by ML-Agents due to its promising results over other popular activation functions, as detailed by Google Brain [39].

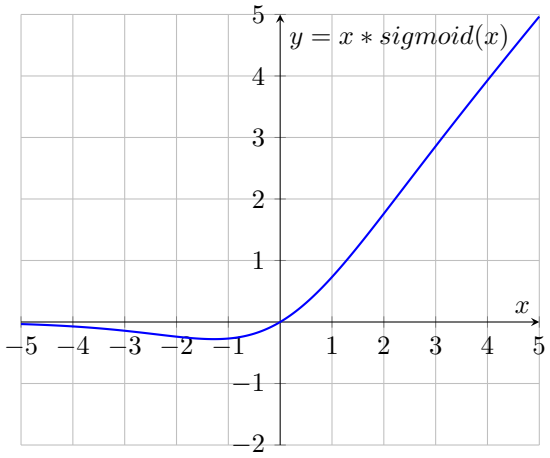


Figure 1: Swish Graph

3.4 TensorBoard

TensorBoard provides a valuable tool for analysing and visualising the results obtained from training ML-Agents models [41]. By providing an interface for graphing, TensorBoard allows an analysis of the training process. Using TensorBoard allows the monitoring of various metrics such as training loss, cumulative rewards, and exploration rates throughout training. These metrics are represented by interactive graphs, allowing for easy comparison and analysis of different reward implementations.

Additionally, TensorBoard offers functionalities for visualising the structure of neural network hyperparameters, specifically how they are adjusted and how capable the model is at predicting future rewards. These insights proved to be valuable by identifying training issues, optimising hyperparameters, and improving performance.

3.5 Git

Git is a distributed version control system widely used in software development [42]. It offers valuable functionalities for monitoring project progress and safeguarding against local issues such as accidental file deletion or corruption. Git's commit-based approach allows effective tracking of changes made to project files. By regularly committing changes with descriptive messages, Git enables an oversight of feature implementations, bug fixes, and project development.

Git makes use of a local repository that feeds into a centralised GitHub repository, providing a backup solution. In the event of local problems such as system failures or accidental file modifications, Git's version control offers a safety net [43]. This proved useful in the project, as it allowed easy reversion to previous commit states, minimising disruptions caused by unforeseen modification issues.

3.6 Community Support and Documentation

ML-Agents and Unity were selected for the project in part due to their strong community support and extensive documentation. ML-Agents benefits from a collaborative environment of researchers and developers, providing resources for reinforcement learning and the implementation of simulations. Similarly, Unity's widespread adoption and comprehensive documentation provide many tutorials and forums, allowing efficient development and the ability to quickly learn how to use the system. Together, a vast range of information and support is available to help overcome unforeseen circumstances and challenges in the process.

4 Project Management

4.1 Methodology

The project adopted an Agile methodology, characterised by weekly 30-minute meetings with the project supervisor, which served to evaluate the success of previous goals and set objectives for the next stage of development. Through Agile principles, the project benefited from a flexible and iterative approach, allowing for quick adaptation to changing requirements and to respond quickly to challenges that disrupted progress. This methodology promotes regular communication and collaboration with the supervisor, allowing for a solid understanding of the project and its goals from both sides.

One of the key advantages of Agile methodology lies in its emphasis on incremental progress and continuous improvement. By breaking down the project into manageable tasks and setting short-term goals, a measured and expected amount of progress was made each week. Additionally, Agile promotes transparency and accountability, as progress is regularly reviewed and discussed during the weekly meetings. This enabled early detection of issues, allowing for timely resolution to reduce the impact of these problems.

Furthermore, Agile methodology encouraged a collaborative work structure where feedback and insights from the supervisor were incorporated into the project, changing its direction and priorities on a week-by-week basis. This iterative feedback enabled the project to evolve dynamically in response to changing objectives and challenges. Overall, Agile methodology provided a structured yet flexible framework, ensuring efficient progress towards project objectives while facilitating continuous learning and improvement.

4.2 Timeline and Milestones

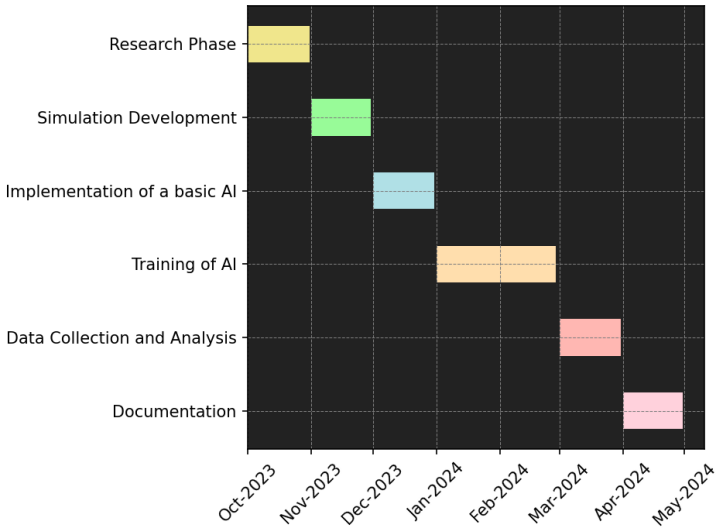


Figure 2: Original Project Timeline

Figure 2 shows the original plan given in the specification. As an agile methodology is applied, a strongly structured plan is not necessary, as it is unlikely to be followed accurately and does not allow for flexibility in the face of variations against the timeline. Therefore, this plan gave a general structure of the key stages of the project and an estimate of the time needed to complete them.

The outlined timeline in figure 2 provided a solid layout for the project, detailing key stages of the project. However, the actual execution encountered some challenges, particularly in the development of the simulation environment, which extended beyond the initially estimated timeframe. This issue is discussed in Section 8.1; however, it is important to note that despite the setback, the overall plan and

timeline proved to be successful for the project. It allowed early detection of the deviation from the projected schedule, enabling proactive adjustments to address the issue.

4.3 Risk Management

Table 1: Project Risks and Mitigation Strategies

Risk	**Chance**	**Impact**	**Problem**	**Mitigation Strategy**
Hardware Failure	Low	High	The computer system used for development is critical. If it experiences hardware failure, there is a risk of project delays or failure.	Regular backups of project files will be maintained on GitHub, and there will be access to alternative computer resources if needed.
Unity	Low	High	Unity may release updates or changes that affect the project's compatibility with existing assets or may affect the licensing availability for the project.	The project will use a stable version of Unity, and any news about software unavailability will be assessed frequently.
Complexity of AI Models	Medium	Medium	DRL may lead to unpredictable behaviours or difficulties in training. Optimising these can be time-intensive and may not result in the expected outcome.	Researching existing strategies for optimising training, such as techniques of approaching reward functions and adjusting hyperparameters carefully.
Technical Challenges	High	Low	Developing an AI driving simulation can be technically challenging, requiring advanced knowledge of Unity, ML-Agents and more general concepts about machine learning.	Initial research into the required technical knowledge to learn how to effectively use the systems.
Integration Challenges	Low	Medium	Integrating ML-Agents into existing systems may present integration challenges or compatibility issues.	Research existing implementations of ML-Agents and how the environment was designed and structured to be compatible.
Resource Limitations	Medium	Medium	Limited computational resources or hardware constraints may hinder the training process, slowing down experimentation and development progress.	Explore options for optimising resource usage and scalability, such as training at reduced speeds or using a no-graphics setting.

Table 1 shows the table of risks created in the initial stages of this project. It describes each of the possible risks that could occur during development that pose a threat. It also shows mitigating strategies to best avoid the risks and reduce the chance or scale of their impacts.

Fortunately, only minor issues impacted the project, namely the technical complexity of implementing the simulation and the complexity of the AI model. Ultimately, this resulted in being unable to keep precisely to the estimate timeline in Section 2. The mitigating action of researching existing projects in the field led to the issue not causing an impact on the overall project delivery as it provided a strong foundation of experience in similar situations.

4.4 Legal and Ethical Consideration

Given that this project operates exclusively within a simulated environment, ethical considerations are not a concern. However, from a legal standpoint, attention was given to licensing considerations for the assets, notably the car model and prefab assets. It was ensured that these assets were free to use without any licensing obligations, adhering to any legal requirements.

5 System Implementation

5.1 Simulation Environment

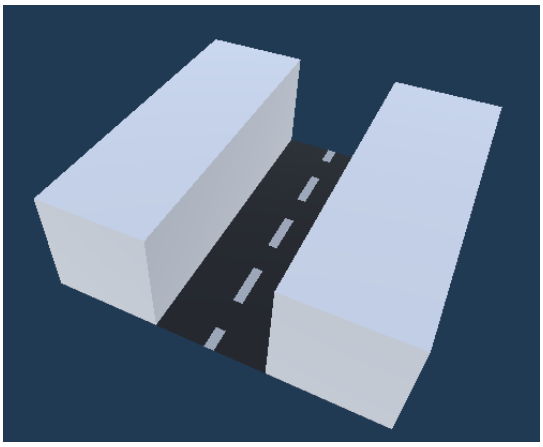
This section explains the implementation of the simulation environment in Unity, which serves as a virtual platform for testing and training the agent. Each of the elements discussed contributes to a complex environment in which the agent learnt to drive. This simulation provided a complex environment, replicating a simplified abstraction of real-world driving.

5.1.1 Roads

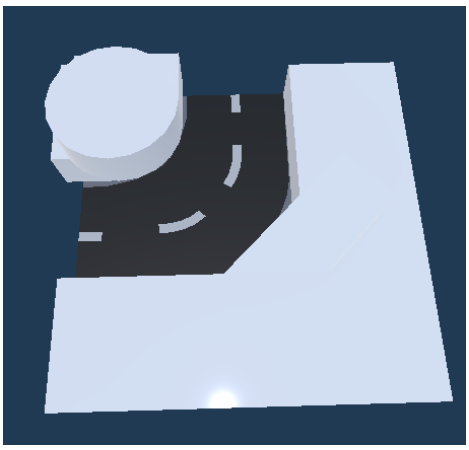
In the simulation, road prefab assets give a visual representation of the road system. However, to accurately map the roads and enable interaction with the agent, Unity's shape primitives were used along with collision detection. This approach allowed the shaping of the geometry of the roads and their interaction with other objects, namely the environment cars and the agent itself.

The road components are categorised into four main types: straight roads, corners, three-way junctions, and four-way junctions: each requiring specific geometry and collision mapping representation. For instance, straight roads involved simple linear shapes marking the road boundaries, while corners required a combination of straight road boundaries and cylinders. Three-way and four-way junctions necessitated a similar design to corners with a combination of straight road boundaries and cylinders.

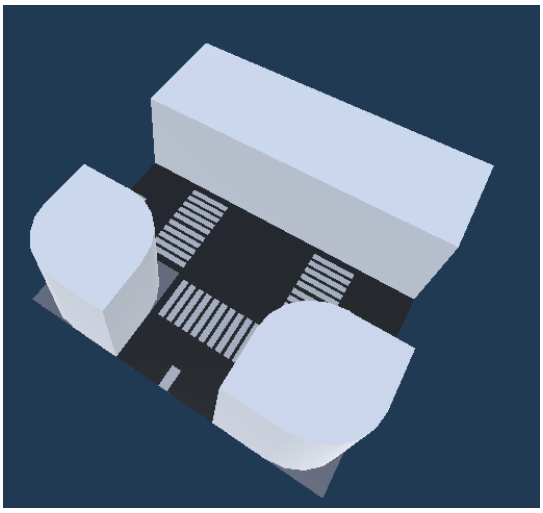
Although in the simulation these geometric mappings are invisible, they are visualised in figure 3 and show how they are constructed to resemble the shape of each road component.



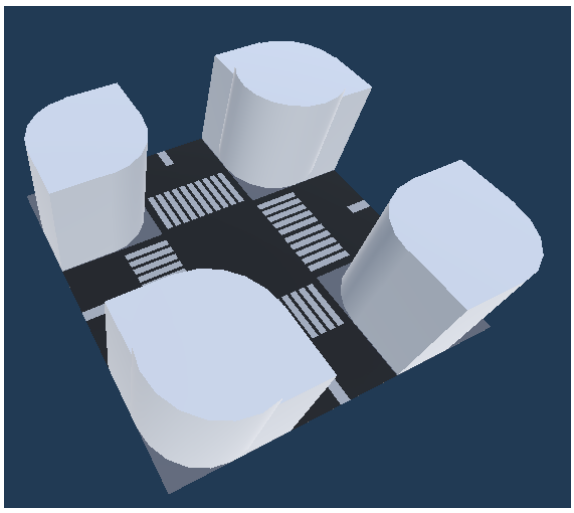
(a) Geometry of the straight road prefab



(b) Geometry of the corner prefab



(c) Geometry of the 3-way intersection prefab



(d) Geometry of the 4-way intersection prefab

Figure 3: Geometry mappings of road prefabs

To enable the agent to perceive and interact with the roads, raycasts and detection triggers were used. Raycasts allowed detection of road boundaries through the use of collisions, allowing the agent to navigate and map the environment. Triggers were used to signal when a car had left the road boundaries, defining when the agent had driven off-road, and to catch mistakes made by the other cars in the environment that cause them to drive out of bounds.

The map of the road system is shown in figure 4. This system is designed to incorporate a variety of challenging aspects, including short segments of road between junctions, a range of corners, and a mix of both 3-way and 4-way junctions. This design provided a sufficient challenge for the agent such that it could learn complex decision-making and implement a range of behaviours required to be able to navigate it successfully. The complexity is further increased by the introduction of the traffic light system and environment cars.

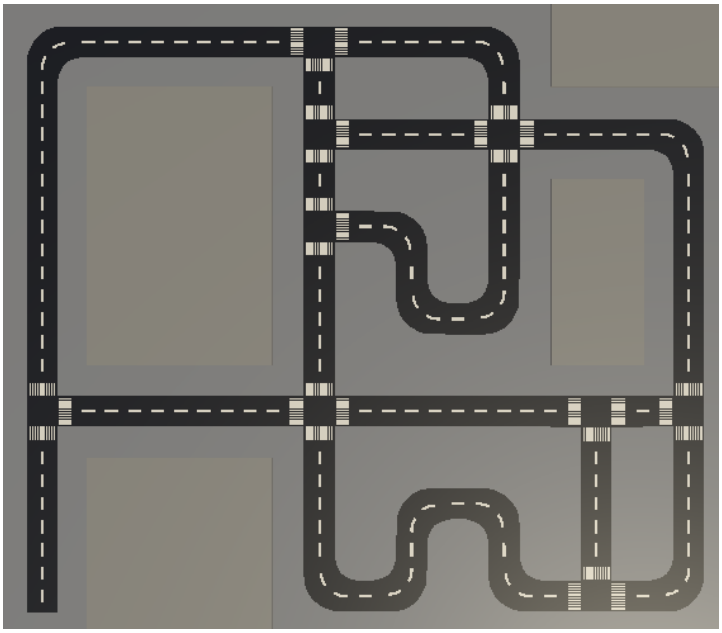


Figure 4: Map of the road system

5.1.2 Pathing

In the simulation, pathing defines the routes that both the environment cars and the agent navigate. The pathing system involved placing targets on the road prefabs to define potential routes. These targets serve as waypoints that guide the cars along designated paths. An example of how the pathing implementation is applied using targets to a 3-way junction is shown in figure 5.

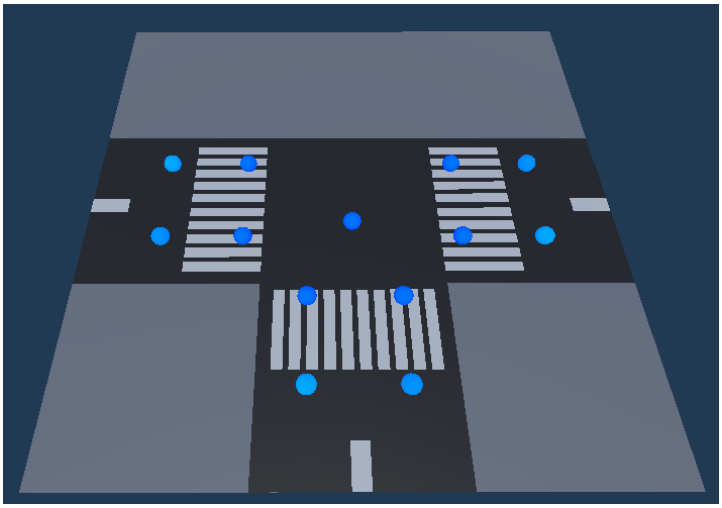


Figure 5: Implementation of pathing in a 3-way junction

Specific routes were defined as an ordered list of targets that the cars cycle through. As the car progresses along its designated route, it moves from one target to the next representing an abstraction of GPS path planning. In the context of the environment cars, eight separate routes were created to cover the whole map and develop a complete road system. These routes are shown in figure 6 for 8 groups of cars following set routes, with the number of cars in each group depending on the size of the route and the expected traffic flow.

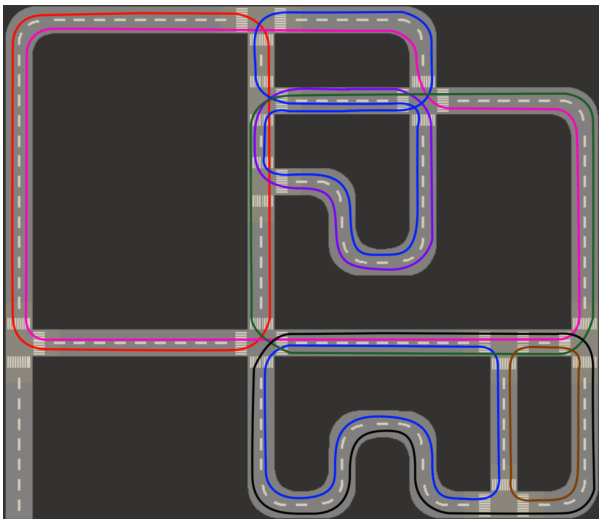


Figure 6: Car routes for the environment cars

Furthermore, for the implementation of pathing functionality, a script for the *Target* class was developed to interact with the targets. This script enables the targets to function as objects within the Unity simulation environment, allowing them to be referenced and manipulated as needed during navigation. Specifically, the script helps govern when a target has been reached, by some distance threshold, triggering the transition to the subsequent target.

5.1.3 The Car Control System

The car control system applies the Unity standard car controller asset to govern the movement of cars. This asset provides a pre-built framework for simulating realistic car dynamics, including features such as acceleration, steering, braking, suspension, traction control, down-force, and gears. This provides a realistic implementation of car control for the cars with inputs of accelerate, brake, and steer as continuous values.

However, an unexpected issue arose during the development process, where cars were unable to accelerate after coming to a complete stop. This issue posed an unexpected challenge to the project, specifically the implementation of the cars. As it is a standard asset developed by Unity themselves, it was assumed that it would function flawlessly, and debugging this issue caused a delay of 1-2 weeks.

Resolving this issue required a complete understanding of the underlying mechanics of the standard car controller script, and through experimentation, adjusting the brake torque settings within the script resolved the problem. This issue was specifically solved by adding an else condition in the handbrake handling:

```
if (handbrake > 0f)
{
    var hbTorque = handbrake * m_MaxHandbrakeTorque;
    m_WheelColliders[0].brakeTorque = hbTorque;
    m_WheelColliders[1].brakeTorque = hbTorque;
    m_WheelColliders[2].brakeTorque = hbTorque;
    m_WheelColliders[3].brakeTorque = hbTorque;
}
else // resolving the issue
{
    m_WheelColliders[0].brakeTorque = 0;
    m_WheelColliders[1].brakeTorque = 0;
    m_WheelColliders[2].brakeTorque = 0;
    m_WheelColliders[3].brakeTorque = 0;
}
```

The addition of the else statement ensures that when the handbrake value is zero, the brake torque applied to all four wheels is set to zero. This is crucial because it prevents the brake torque from persisting when the handbrake is not engaged. Without the else statement, the brake torque remains applied to the wheels even when the handbrake is released. This fix ensures that the wheels are free to rotate and the car can accelerate forward once the handbrake is released.

5.1.4 Environment Cars

The environment cars are the non-agent cars in the simulation environment. They follow a pre-defined path and are controlled by a script that dictates actions given to the car controller. The environment car script, *AutoCarControl*, considers factors such as the angle and distance to the next target, the car's speed, the traffic signal of the next traffic light, the distance and relative velocity of the car in-front, and other features required to drive naturally. The goal was to emulate natural driving behaviour, such that the cars drive realistically within the simulation.

5.1.4.1 Car Functionality

The environment car controller script imitates natural behaviour by dynamically adjusting acceleration, braking, and steering based on factors such as proximity to traffic lights and the relative speed of nearby cars. Through iterative refinement, the behaviour of the environment cars resembles the general behaviours of human drivers. This enhances the realism of the simulation and enables the project to accurately model and analyse relatively realistic traffic interactions.

5.1.4.2 Logical Implementation

A description of the environment car controller script is as follows:

- The script initialises acceleration, brake, and steering values to defaults of 1, 0 and 0 respectively (accelerating forwards).
- Retrieves the current speed of the car from the Unity Standard Asset Car Controller.
- Identifies the next traffic light and path target along the path.
- Calculates the local positions (relative to the car's position) of the targets.
  - The distances and relative angles are calculated from the relative positions and the car's orientation.
- The relevant car in-front is found to calculate the brake distance:
  - Iterates through the over cars in a defined field of view in-front, using the orientations of the subject car and the compared car to define an angle of relative orientation.
  - The angle of relative orientation is compared to a threshold as to only consider cars which are in a similar orientation (on the same side of the road and in its path).
  - From this group, the closest car is used as the "car in-front", otherwise a null value is used if there is no such car.
- Checks criteria to initiate braking at lights:
  - If the distance to the next light is within a certain threshold, and it is red or amber.
  - If the distance is within the threshold, and the light is green, but there is no space at the exit of the junction.
  - If either of the previous conditions are met and the traffic light is closer than the car in-front.
  - If the speed is greater than a threshold when the car is close to a traffic light (to slow down at junctions for turning).
- The braking distance is dynamically calculated based on the current speed of the car and the maximum speed:
  - Brake distance = current speed $\times (1 + \frac{\text{current speed}}{\text{maximum speed}})$
  - This formula was found by trial and error and adjusted until the cars represented a natural human rate of slowing down.
- If the previous criteria are not met, the relevant car in-front (if not null) is used as reference:
  - If the distance to the other car is less than a threshold it fully applies the handbrake to avoid crashing.
  - If the distance to the other car is less than the defined brake distance, and the relative velocity is greater than 0 (i.e., the car is moving faster than the car in-front), a deceleration is applied proportionally to $\frac{\text{distance to the car in-front}}{\text{brake distance}}$.
  - This encourages the car to decelerate more the closer it is to the car in-front within the range defined by the brake distance.
- Calculates the steering proportionally to the angle towards the next target.
- Decelerates if the angle towards the next target and the current speed of the car are above defined thresholds to slow the car down around tight corners.

5.1.5 Traffic Light System

By controlling traffic lights at each junction, this system simulates a traffic system, ensuring car flow and realistic interactions at intersections. Most importantly, it provides another behaviour that the agent needs to learn within the curriculum learning system.

5.1.5.1 Target Types

The same targets used in the path-following behaviour serve as key components in defining the functionality of the traffic light system. These fall under three categories: traffic light targets, finish targets, and normal targets. An example of how these are applied within the context of a 3-way road prefab is shown in figure 7: Dark blue spheres represent normal targets; light blue spheres are finish targets; and red spheres represent a traffic light (with red lights active).

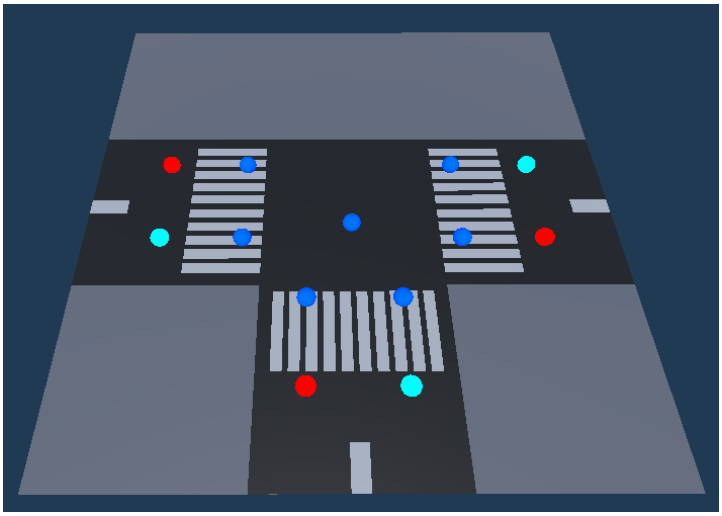


Figure 7: Implementation of different target types in a 3-way junction

Traffic light targets are positioned on the junction prefabs to represent the locations of traffic lights. Each can exhibit three distinct states implemented using boolean values: stop (red), slow (amber), or go (green), corresponding to standard traffic signals. Furthermore, these targets store information regarding the number of active cars within their respective junction and the count of cars waiting at the associated traffic light section.

Finish points mark the exit points of junctions and ensure that the environment cars do not enter congested junctions if there is insufficient space at the exit. The active count of each finish target determines whether a car should proceed through the junction or wait, preventing gridlock scenarios and improving traffic efficiency. The active count is defined as the sum of cars currently active in the proceeding stretch of road.

5.1.5.2 Logical Implementation

The implementation of the traffic light system is controlled through a *JunctionController* script associated with each junction in the simulation environment. This script is responsible for adjusting the status of traffic lights. By modifying the boolean variables corresponding to red, amber, and green states for each traffic light target, the scripts control the sequencing of traffic signal phases. This flexible implementation allows for the creation of customised signal patterns and enables the simulation of a various traffic scenarios, such as adaptive signal timing based on traffic conditions.

Each junction also has individually specified active times for each light. This is based on the size of the road segment leading up to the light and the expected traffic flow; lights that are responsible for a larger amount of congestion are granted a longer active time. These active times are used in the timed system, which follows the set structure:

- The active counts for the traffic lights are summed to give a cumulative active count of cars in the junction.
- Each time the traffic light changes, a timer begins, and the light turns green.

- Once the timer reaches the threshold defined by the active time for the current light, the light turns amber.
- After a further threshold of 1.5 seconds, the light turns red.
- To avoid collisions of environment cars, the light waits until the cumulative active count is 0 to move to the next traffic light.

The need for incorporating waiting, and active times arose from a critical gridlock scenario, as illustrated in figure 8. This situation occurred when the cars reached a standstill, unable to move forward. At the junction entrances, cars had to wait for a clear junction before proceeding, which led to a gridlock, with both cars stuck until the highlighted cars cleared the intersection. However, the movement of these highlighted cars depended on the waiting cars, causing the cyclical issue.

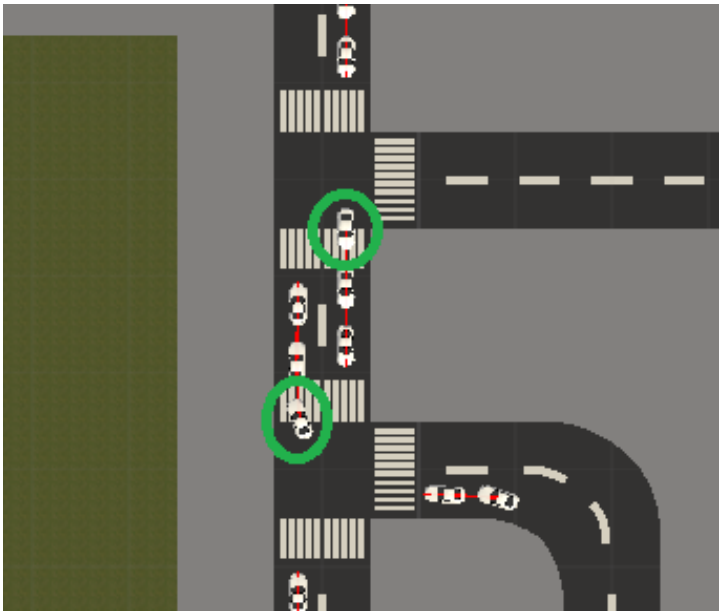


Figure 8: Gridlock issue

Introducing waiting and active counts fixed this issue, as it resulted in a system where cars can only enter a junction if there is sufficient space for them at the exit. This is defined by a variable stating how many cars can be on the stretch of road following a finish target. This is compared to the active count of the finish target to give a result about whether there is sufficient space for an additional car. However, this oversight caused a delay in the progress of the project against the timeline shown in figure 2.

5.1.5.3 Car Interaction

The interaction between cars and traffic lights and finish targets is necessary for their functionality. Each car stores the next traffic light target and the previous finish target as fields in its behaviour script. Upon reaching a traffic light target, the waiting count of the target is decremented, and its active count is incremented, indicating its presence as active in the junction. Conversely, when reaching a finish target, the traffic light active count is decremented and the finish point active count is incremented, signifying its transition from the junction to the next segment of the road.

To accommodate for car spawns and resets within each stage, the targets are dynamically updated. When a car is spawned or reset, its associated traffic light target and finish target are adjusted accordingly based on its current position. This ensures that the cars interact with the traffic lights and finish points appropriately.

5.1.5.4 Smart Traffic

This system also allows the capability of developing "smart" traffic lights. A basic implementation of this could be done by changing the lights earlier if no cars are waiting at the active traffic light. Upon testing, it proved to reduce waiting times for cars at the other traffic lights in the junction. However, this is not essential to the agent and makes little to no impact on its perception of the environment, so it will be left as future work.

5.2 The Agent

As discussed in Section 3.3, the agent employs Proximal Policy Optimisation to train its decision-making policy. Specifically, the reinforcement learning component of the agent is implemented using the *Agent* class provided by the ML-Agents package. The agent interacts with the environment by observing the current state, taking actions, and receiving rewards. Through this interaction, the policy gradually improves by maximising cumulative reward over time.

5.2.1 Controls and Action Space

The agent makes use of the same standard asset car controller script and model used for the environment cars (with the difference of a green appearance). This allows a realistic driving simulation where the car has realistic features previously described. This car controller script defines the action space for the agent and, thus, the output of the deep neural network. These actions are values for accelerating, braking, and steering represented in table 2.

Action	**Domain**
Accelerate	$[-1, 1]$
Brake	$[0, 1]$
Steer	$[-1, 1]$

Table 2: Actions and domains of the agent

5.2.2 ML-Agents Interaction

The *Agent* class serves as the base class for defining DRL agents within the Unity environment. It provides a set of methods and properties that allow interaction with the simulation environment and learning processes of the PPO system.

The structure of the *Agent* class includes key methods such as *Initialize()*, *OnEpisodeBegin()*, *CollectObservations()* and *OnActionReceived()*:

Initialize() is called when the agent is initialised, allowing for any necessary setup or initialisation tasks to be performed. In the case of the agent in this project, this simply attaches the car controller script, so it is able to perform its actions.

OnEpisodeBegin() is called at the beginning of each episode, providing an opportunity to reset the environment and prepare for a new episode. Episodes are a defined period of time between the car starting from a defined position and reaching some terminal condition. In this environment, the terminal conditions are defined by the following events:

- Colliding with another car.
- Driving out of the road boundaries.
- Reaching a time-out threshold since reaching the last target (to avoid issues caused by the car remaining stationary forever).

The *CollectObservations()* method is where the agent collects observations from the environment, which are fed as inputs into the deep neural network. These observations include relevant information which define the state for the agent to perform an informed action. Throughout the project, the number

of observations grew as new behaviours were added and more state information was required. The observations and their associated curriculums are shown in table 3.

Observation	**Curriculum**
Ray cast sensors	1
Current target information	1
Forward velocity	1
Next target information	1
Centre information	1 & 2
Next traffic light information	1, 2 & 3
Next traffic light state	1, 2 & 3
Car in-front information	1, 2, 3 & 4
Car in-front relative velocity	1, 2, 3 & 4

Table 3: Observations and Associated Curriculums

OnActionReceived() is called when an action is received from the decision-making process. This method is responsible for interpreting the received action and applying it to the agent's car controller. It also acts as an update of the agent's condition, such as checks about collision status and reaching targets, updating information required for reward calculations. Once updated, the reward is calculated, and decisions are made related to restarting the current episode.

5.2.2.1 Object Information

Table 3 references "information" about key objects in the environment. In this context, "information" is defined by 2 key components: the angle relative to the agent's forward facing vector and the distance from the agent. Although the simulation itself is 3-dimensional, the position of these objects can be abstracted along a 2-dimensional plane, as the elevation of the objects is not relevant. This allows for a reduced number of observations required to represent the environment and thus improves the quality of training.

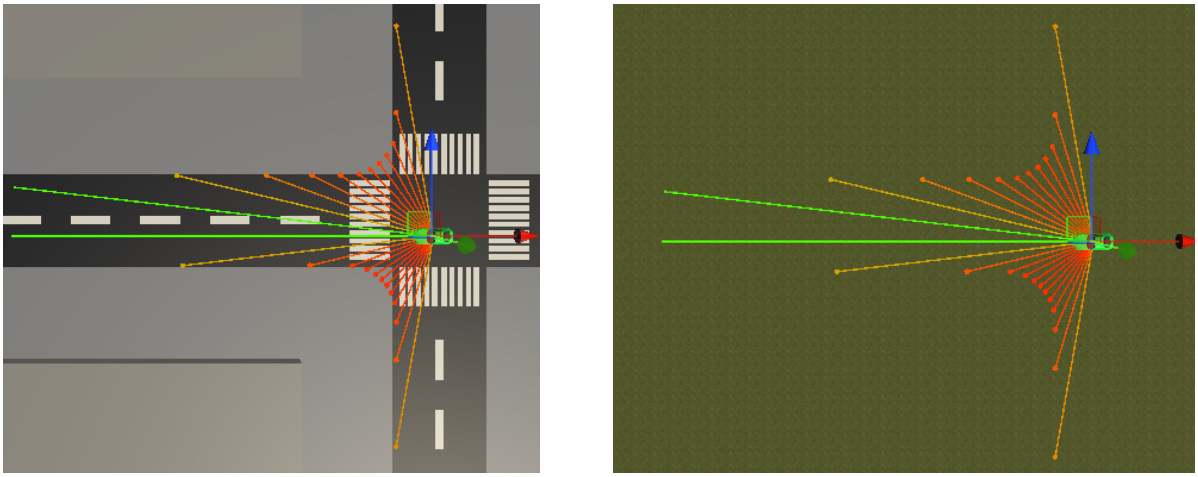
5.2.2.2 Ray Cast Sensors

The ML-Agents *3D Ray Perception Sensor* is used to detect road boundaries, imitating real-world lidar detection [10], which is applied to detect obstacles in the real-world. The road boundaries themselves would be implemented through the use of camera machine learning systems, however, the use of ray cast collisions gives a comparable abstraction.

Specifically, the ray sensor used by the agent is configured with:

- 12 rays projected in each direction (left and right) offers a fine balance between sufficient rays to give a precise representation of the environment while not being so many as to overcomplicate the model.
- The maximum ray degrees is set to 80° from the forward vector, creating a field of view of 160°.
- A ray length of 45m provides adequate reach for detecting distant objects.
- The sensor's focus on road boundaries is implemented by a ray layer mask, this means the ray cast will only detect road boundaries and not other objects.
- Three stacked ray cast results provide a sense of short-term temporal information as the agent is given access to the current ray cast information and that of the 2 previous states. This gives an improved perception of the environment and how it changes as the car moves.
- A vertical start and end offset of 0.5m avoids potential collision issues with the floor and represents the ray casts being situated 0.5m above the floor.

The individual sensors are normalised according to the ray length of 45m. This means that a collision at 0m it set to a value of 0 and a collision at the maximum range of 45m (or no collision) gives a value of 1. This gives a representation of the environment, which can be seen in figure 9.



(a) Agent's view in context of the environment

(b) Agent's view from its perspective

Figure 9: 3D ray perception sensor implementation

This ray perception sensor is key for the agent to be able to interpret its surroundings, specifically to be able to stay within the road boundaries. However, it comes at the cost of being the most complex feature of those inputted into the neural network, as 12 ray casts on each side stacked 3 times gives a total of 72 inputs.

5.2.3 Deep Neural Network Architecture

As described in 3.2, a Deep Neural Network has at least three hidden layers, i.e. layers that are not the inputs or outputs. These can be configured to represent various depths of complexity within a neural network and can be adjusted in terms of the number of neurons per layer and the total number of hidden layers. Figure 10 shows a simple example of a neural network with a 3x4 configuration, i.e. the neural network has 3 hidden layers with 4 neurons per layer.

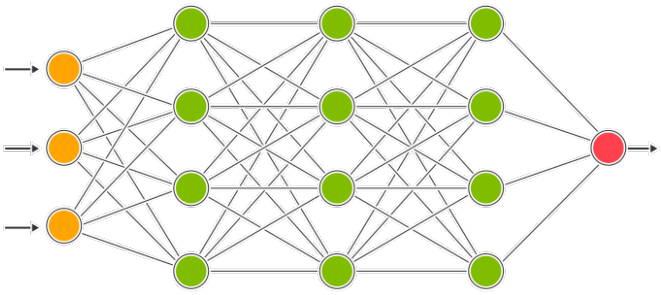


Figure 10: An example neural network
(source: www.spotfire.com/glossary/what-is-a-neural-network)

Each neuron in the hidden layers takes a weighted sum of the input values from the previous layer and performs the swish activation function (discussed in Section 3.3.4). The edge weights are adjusted through gradient descent to improve the policy of the agent, along with bias inputs for each neuron, which are additional parameters that translate the activation function and improve learning capability.

While other systems allow a different number of neurons for each hidden layer, ML-Agents is limited to the same number of neurons for all hidden layers. During the project, the number of hidden layers

and neurons was adjusted in response to training quality, but ultimately, a configuration of 4x256 was used, meaning the network was configured with 4 hidden layers each with 256 neurons.

5.2.3.1 Configuration and hyperparameters

The configuration of the neural network in ML-Agents is fundamental for training an effective agent. In ML-Agents, this configuration is defined in a YAML file, which specifies various aspects of the neural network architecture, such as the number of layers and other hyperparameters. Throughout the project, these configurations were continuously refined and adjusted based on the training progress.

Tweaking these hyperparameters and neural network configurations was a complex but important part of the project that had a significant impact on the quality of the agent. Initially, commonly used values were used as suggested in the ML-Agents training documentation [44].

Based on observations from training, adjustments were made to individual hyperparameters or configurations to assess their impact on training quality and resulting behaviour. This incremental approach allowed for a better understanding of how each parameter influenced the learning process. Feedback from training, such as convergence speed, stability, and final performance, guided further adjustments to the DRL model. This refinement continued until satisfactory results were achieved, balancing training efficiency and desired behavioural outcomes with the final parameters given in table 4.

Parameter	Value	Description
Hyperparameters		
batch_size	250	Number of experiences in each iteration of gradient descent
buffer_size	5000	The max size of the experience buffer
learning_rate	0.0003	The rate at which the model learns from the data
beta	0.003	Strength of the entropy regularisation, “more random”
epsilon	0.25	Influences how rapidly the policy can evolve during training
lambd	0.925	How much the agent focusses on close rewards vs. future ones
num_epoch	5	Passes through experience buffer during gradient descent
learning_rate_schedule	linear	Linear decay of the learning rate
epsilon_schedule	linear	Linear decay of the epsilon value
network_settings		
normalize	false	Whether to normalise inputs
hidden_units	256	Number of neurons in the hidden layers
num_layers	4	Number of layers in the neural network
vis_encode_type	simple	Type of encoding used for visual inputs
reward_signals		
extrinsic gamma	0.99	Discount factor for future rewards coming from the environment
extrinsic strength	1.0	Strength of the extrinsic reward signal
curiosity strength	0.1	Strength of the curiosity reward signal
curiosity gamma	0.99	Discount factor for future rewards coming from curiosity
keep_checkpoints	20	The maximum number of model checkpoints to keep
max_steps	4500000	The maximum number of steps for training
time_horizon	500	Steps to collect before adding the experience to the buffer
threaded	true	Whether to use threading for parallel processing

Table 4: Behaviour Configuration for Self-Driving Agent

In table 4: The hyperparameters section specifies the settings that control the training process; the network settings section defines parameters related to the neural network architecture, such as the number of hidden neurons and layers; and the reward signals section configures the rewards used during training, including parameters such as gamma (the discount factor for future rewards) and strength (the importance of future rewards).

In the context of DRL, a step refers to a single interaction between an agent and its environment, consisting of one observation, action, reward cycle. An experience represents a defined number of steps

taken by the agent, which is implemented according to a timeframe relevant to the situation.

The time horizon (length of an experience as a number of steps), was greatly influential on the quality of the training as it defined how long of a period of time the agent should learn from: too long, and the errors become insignificant and the reward unpredictable; too short, and the agent is not given enough context for decision-making. The time horizon, which was ultimately set to 500 steps, as seen in table 4, gives sufficient information about the result of a series of actions without trying to predict rewards too far into the future. 500 steps roughly equates to a few seconds, which is similar to the length of predictable time in real-world driving.

Other key features that are configured by these parameters include the experience buffer and gradient descent:

- The experience buffer is a data structure used to store a collection of recent experiences, which the agent can sample during the training process to learn from past interactions.
- Gradient descent is an optimisation algorithm used to adjust the weights of the neural network [45] in order to minimise the difference between predicted and actual values and thus learn the best actions to optimise its rewards. This is the process that encapsulates learning within the system and allows adjustment of the weights and biases.

5.2.3.2 Final Neural Network Structure

Section 5.2.2 gives an overview of the observations fed into the neural network. Although each curriculum involved the introduction of more observations, the final configuration had 14 values for the non-ray cast inputs given in table 3 and 72 input values for the ray casts. This gives an overall structure of 86 input neurons, 4 hidden layers each with 256 neurons, and the output layer consisting of 3 values to represent the action space.

5.2.3.3 Excluded features

Initially, the project explored the implementation of Generative Adversarial Imitation Learning (GAIL) and behavioural cloning to incorporate imitation learning techniques into the self-driving agent. Imitation learning, particularly in the context of self-driving, aims to teach an agent by mimicking a demonstration of a human driving the same route as the agent. Such approaches were considered promising due to their potential to apply human expertise to improve the speed of training. This represents an implementation of the real-world learning of autonomous cars, as many implementations, notably Tesla's fleet driving system [46], rely on human data to improve the quality of learning.

However, after testing and experimentation, it was found that these techniques were counterproductive and slowed the agent's learning rate, as later discussed in Section 7.1.1. Therefore, the decision was made to discontinue the use of GAIL and behavioural cloning in the project.

5.2.4 Training Path

In designing the path for the agent, consideration was given to creating a sufficiently complex route that encompasses a range of challenges. The chosen path, represented in figure 11, incorporates a variety of elements essential for the agent's learning, including both left and right turns, 3-way and 4-way junctions, and sections featuring quick successions of corners. By following this path, the agent was given a set of scenarios representative of the environment. The limitation of not allowing complete exploration of the map allows for testing using a different route to assess the quality of the agent.

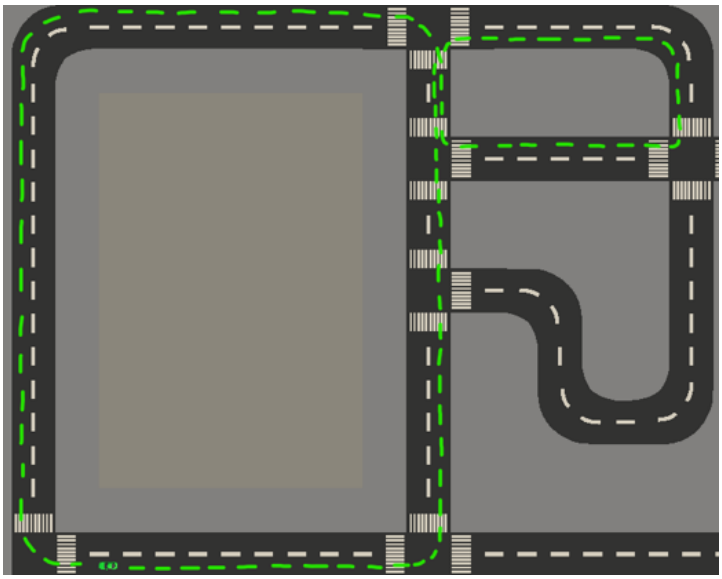


Figure 11: Training path for the agent

5.3 Reward Systems

The reward systems define the behaviour of the agent. In this project, the primary aim is to develop an agent that follows a designated path, slows down at corners and junctions, stays within the lane, stops at traffic lights, and avoids collisions with other cars. To achieve these goals, various reward system approaches were developed, and behaviours were accumulated through curriculum learning.

The implementations of the most significant reward systems are given in this section, and their results discussed in Section 6.

5.3.1 Reward Types

The reward system encompasses several types of rewards for different aspects of the agent's behaviour and interactions with the environment.

Target-reaching Rewards: One aspect of the reward system is focused on providing rewards for successfully reaching path targets. These rewards serve as positive reinforcement for the agent with the main goal of following the designated path, encouraged by receiving a large reward at each target. By associating the series of actions leading up to the reward, the agent learns to prioritise actions that lead to reaching the targets and view them as the primary focus of its behaviour. This type of reward is specifically a sparse, extrinsic reward, as it is a reward that is obtained infrequently and defines the primary objectives of the agent.

Negative Rewards for Boundary Violations: In addition to rewards for positive actions, the reward system also includes penalties or negative rewards for undesirable behaviours. In this case, the key negative rewards are given when the agent crashes into another car or leaves the road boundaries. These negative rewards in this project often also define the end of an episode, restarting the agent to re-attempt driving the training path. This is a design choice to avoid slow learning, as allowing the agent to continue in these situations is undesirable: it should not learn behaviours of driving off-road, nor should it attempt to continue after a collision.

Action-specific Rewards: Furthermore, the reward system incorporates action-specific rewards to provide feedback on the quality of specific actions taken by the agent. Actions that contribute positively to the agent's progress, such as making a larger amount of progress towards the next target or slowing down for a red light, are rewarded with positive feedback. Conversely, actions that impede progress receive lower or negative rewards, encouraging the agent to make better decisions over time.

These action-specific rewards are a form of dense shaping reward. A dense reward differs from a sparse reward as the reward is given frequently, in this case every step. Shaping rewards help to shape the agent's behaviour without defining the large-scale primary goals.

The values of these key rewards varied slightly throughout the project, but this was negligible, and the key final values are shown in table 5.

Event	**Reward**
Reaching a target	25
Action reward per step	Defined by the current reward system
Leaving the road boundaries	-50 & episode reset
Collision with another car	-50 & episode reset
A defined time-out period after reaching the previous target	-50 & episode reset

Table 5: Implemented rewards

A key observation about table 5 is that the same negative reward is applied to the three undesired events. This is to avoid any issues of favouring certain negative behaviours, an example being that the agent may "fear" driving off-road and view timing-out as more desirable, causing it to remain stationary until the timeout. This could occur, as it may be more probable that the agent accumulates a larger reward by timing out without further progress than attempting to make progress with a chance of collision.

By combining these different types of rewards, the reward learning system shapes the agent's behaviour and decision-making processes, guiding it towards behaviours that lead to successful task completion while discouraging actions that may lead to undesirable outcomes.

5.3.2 Weighted Average

The initial approach involved a simple weighted average of different factors to encourage desired behaviours. However, this approach proved to be somewhat naïve, as it was blindly encouraging each behaviour without considering how the agent would apply it in context. The reward was based on a weighted sum of the factors given by the formula in figure 12.

$$\frac{0.05}{a+b+c} \times (a \times \frac{2.5-\text{angle}}{180}) + b \times (-\frac{\text{distance}}{100}) + c \times (\frac{\text{speed}}{\text{max speed}} - 1))$$

Figure 12: Weighted average reward function

Values a, b and c represent scalar values that were adjusted slightly through the training in response to activities being neglected or favoured too strongly. The attempted variations of values used are not worth noting, as this reward system did not yield a useful system of training. However, it should be noted that the reward is based on a value of 0.05 so that the shaping rewards do not overpower the extrinsic rewards and they can maintain their significance.

In the 'a' term, it can be seen that the angle was subtracted from a value of 2.5 in an attempt to make the term positive when the angle is within a 2.5° threshold of the target angle, although with further research, this turned out to have no mathematical consequence, only translating the reward, and did not improve learning capability. Another constant shown in the 'b' term is an arbitrary scaling value of 100 used to scale the distance so that the reward obtained is not too large.

The key factor to note about this function is that each term is usually negative. The implementation of negative step rewards is often used in DRL to encourage an agent towards an extrinsic goal in order to accumulate some positive reward. This aims to avoid idling and accumulating a positive reward without progressing, ultimately encouraging active progress.

5.3.3 Positive Weighted Average

Building upon the previous approach, discussions during supervisor meetings formed a new implementation of a reward system that made use of a weighted average of positive values. This modification aimed to provide more intuitive feedback to the agent by rewarding desired behaviours and allowing a better shaping of them. This aimed to allow the agent to accumulate a positive reward without reaching a target and thus learn the correct behaviours quicker and more effectively. The same factors were encouraged but adjusted to be positive, as given in the formula in figure 13.

$$\frac{0.05}{a+b+c} \times (a \times (1 - \frac{\text{angle}}{180}) + b \times (1 - \frac{\text{distance}}{100}) + c \times \frac{\text{speed}}{\text{max speed}})$$

Figure 13: Positively average reward function

As in Section 5.3.2, the coefficients a, b, and c represent the weights of each behaviour desired. These were slightly adjusted through the training process, only slightly deviating from the final values of 5, 3 and 2 respectively. This shows a prioritisation of angle behaviour, then of decreasing the distance, and finally of increasing the speed of the agent.

5.3.4 Scaled Reward System

In refining the previous attempts, the idea of normalised scaling factors was introduced for desired behaviours as opposed to a weighted average. This provided the foundation for the final reward system and an effective method of reward implementation. In this reward function, the features of maximising speed and rewarding closer distances to the target were replaced by a change in normalised distance to the next target since the previous action. This represents a better measure of progress as the sole aim is to decrease the distance along the path, combining both features of driving towards the target and encouraging a higher speed.

The change in normalised distance provided a foundation for the reward system as it quantitatively defines the progress made towards the target, which can be scaled according to how well it matches a desired behaviour — in this case, a scalar of how close the angle of direction is to the target. This reward function is represented in figure 14 where a scaling factor, θ, for the desired angle behaviour is applied.

$$reward = \begin{cases} -1 \times |\Delta d_n \times \theta| & \text{if } \Delta d_n < 0 \text{ or } \theta < 0 \\ \Delta d_n \times \theta & \text{otherwise} \end{cases}$$

- d_n = the normalised distance to the next target at the given step
- $\theta = \frac{\text{angle to target}}{90}$ (angle scaling factor)

Figure 14: Scaled reward function

A key note about the reward function in figure 14 is the domain of θ: The angle of 90˚ is used as a reference, as opposed to 180˚, as it covers a domain of [-1,1] allowing both negative and positive rewards. If the agent has an angle to the next target greater than 90˚, it implies it is driving in the opposite direction and should therefore receive a negative reward. Increasing the domain also causes deviations from the desired angle to have a more significant impact on the reward, helping to more strongly encourage optimal behaviour and punish suboptimal behaviour.

Figure 14 also includes conditioning where the result is negative if either of Δd_n or θ are negative. This is to avoid behaviour where both instances are negative, resulting in a positive reward. This would be the case if the agent were to drive in the opposite direction.

5.3.4.1 Normalised Distance

As mentioned previously, the distance is normalised, aiming to guarantee a normalised reward. The distance to the next target is scaled relative to the previous target, as represented in figure 15.

Figure 15: Normalisation of the distance between targets

5.3.5 Curriculum Learning

Once a foundation for the reward system had been established, the curriculum learning system could be implemented. In this approach, the difficulty of the tasks was gradually increased. This was achieved by incrementally introducing new behaviours, each shaped by the addition of a new scalar, such as θ in Section 5.3.4. Each of these additional scalar values are designed aiming to cover the domain of [-1,1] or [0,1] as to normalise the reward and scale it according to the quality of the behaviour.

The main stages of the curriculum were to:

- Follow the path (as designed in Section 5.3.4)
- Slow down around corners and through junctions
- Stay in the centre of the lane
- Stop at traffic lights
- Avoid collisions with other cars

5.3.5.1 Implementation of a Curriculum

Within the context of the *Agent* class, a reward at a specific stage in the curriculum is defined by its own method. During each stage, the relevant method is applied, and the agent is trained to reach the desired behaviour — decided by visual analysis or a reward threshold. Then, the next behaviour is implemented, and the shaping reward changed to the next advancement (another method in the class).

5.3.6 Corner-Slowing

Slowing down appropriately when approaching corners and junctions was the first stage where curriculum learning was introduced. This was achieved by adjusting the target reward to penalise excessive speed at corners or junctions. By incentivising cautious driving, the aim was to improve the agent's ability to navigate complex road layouts without being too strongly influenced by its reward for driving quicker.

The implementation of this was defined by a threshold speed, which was set to a value of 15mph, when reaching a target. This behaviour is implemented when reaching a target, as these are only situated in corners and junctions and not straight road sections. Above this threshold, the target reward is scaled down linearly until it reaches the max speed, where it receives a reward of 0. Adding this to the successful reward system, which taught the agent to follow the path, does not influence the step reward function but only scales the reward for reaching a target, as shown in figure 16.

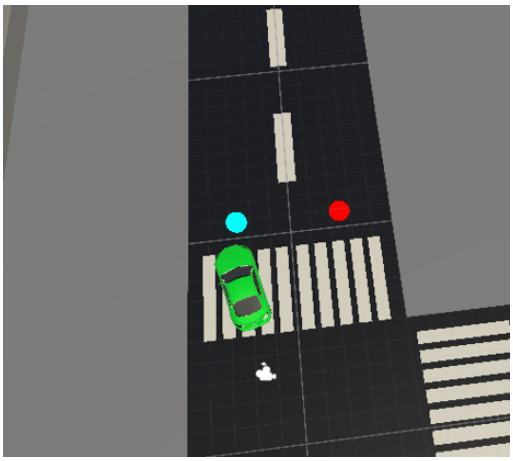
$$\text{Target reward} = \begin{cases} 25 & \text{if } speed \leq 15mph \\ 25 \times \frac{\text{max speed} - \text{speed}}{\text{max speed} - 15} & \text{otherwise} \end{cases}$$

Figure 16: Corner/junction slowing reward

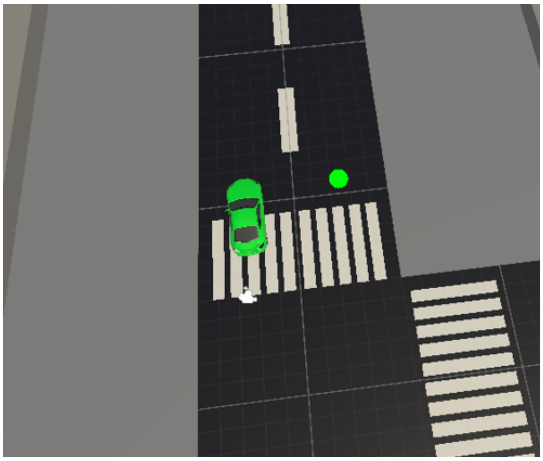
5.3.6.1 Denormalising Rewards

An additional feature at this stage of the project was denormalising the change in distance. This is because it biassed close-together targets and massively reduced the reward obtained by targets further apart. Removing the normalisation removes this issue and allows all actions to be equally scaled.

This change, however, revealed an issue with the reward system by using the change in distance to the next target. As seen in figure 17, at the point of reaching a target, the distance to the next target massively increases, resulting in a large negative change in distance and thus a negative reward.



(a) Car before reaching the target (distance ≈ 0)



(b) Car after reaching the target (distance $\gg 0$)

Figure 17: Change in distance issue

This resulted in a behaviour of the car stopping before reaching a target, completely halting the training progress and posing a massive challenge for the project. After extensive debugging, the issue was resolved by defining the reward as 0 if the agent reached a target in the previous step. This reward of 0 is inconsequential in the scale of such frequent rewards and is not strictly necessary as, during the same step, the agent receives a large extrinsic reward for reaching the target.

5.3.7 Lane-Centering

Once the agent learnt to slow for corners and junctions, reaching a similar reward achieved in Section 5.3.4, the lane centring behaviour could be implemented. This was implemented by a scalar of how far the agent is from the centre of the road. Due to the targets being centralised in the road, the centre is defined by the line connecting the previous and next targets, using vector maths to find the distance to the nearest point on this line. In terms of neural network inputs, this was given as a positive or negative value to indicate direction to the road centre. Whereas, for the shaping reward, the absolute value is used because direction is irrelevant.

This scalar, λ, is defined relative to the width of the lane, more specifically: half the width minus the car's width, equating to 1.6m. This definition allows the implementation of a linear scalar where being in the centre results in a value of 1 and 0 when the edge of the car is touching the lane border. Beyond this, the linear scalar continues into negative values to discourage driving on the wrong side of the road.

Applying this to the shaping reward in Section 5.3.4 gives the function represented in figure 18.

$$reward = \begin{cases} -1 \times |\Delta d \times \theta \times \lambda| & \text{if } \Delta d < 0 \text{ or } \theta < 0 \text{ or } \lambda < 0 \\ \Delta d \times \theta \times \lambda & \text{otherwise} \end{cases}$$

- $\lambda = \frac{1.6 - \text{distance to lane centre}}{1.6}$

Figure 18: Lane centring reward function

5.3.8 Traffic Lights

At this stage in the curriculum, the agent has successfully implemented behaviours to: follow the defined path; slow down for corners and junctions; and stay close to the centre of the lane. This provides a foundation for basic driving in a simple environment to which more complex dynamic behaviours can be implemented. The first stage of this is to introduce the traffic light system to the environment.

Learning this behaviour and defining its reward function is not as trivial as the previous behaviours, as it involves a complex array of behaviours taken for granted in humans. This involves when to consider slowing down for a light, how far away to start braking, and how much to brake to avoid harsh braking. This behaviour depends on the relationship between the distance to the red or amber light and the current speed of the car, which should roughly be equal to encourage natural braking.

$$reward = \begin{cases} -1 \times |\Delta d \times \theta \times \lambda \times \tau| & \text{if } \Delta d < 0 \text{ or } \theta < 0 \text{ or } \lambda < 0 \text{ or } \tau < 0 \\ \Delta d \times \theta \times \lambda \times \tau & \text{otherwise} \end{cases}$$

$$\tau = \begin{cases} 1 & \text{if the light is green} \\ 1 - 0.1 \times \max(0, v) & \text{otherwise if } 0 \leq d \leq 5 \\ \frac{d-5}{\max(d-5, v)} & \text{otherwise if } v > d - 5 \text{ and } d > 0 \\ \frac{d}{5} & \text{otherwise if dotProduct} < 0 \\ 1 & \text{otherwise} \end{cases}$$

$$\text{dotProduct} = \begin{cases} 1 & \text{if agent forward vector} \cdot \text{vector to light} > 0 \\ -1 & \text{otherwise} \end{cases}$$

- v = forward velocity
- d = distance to light $\times$ dotProduct (to indicate forward or backward direction)

Figure 18: Traffic light reward function

The reward function in figure 18 shows an increase in complexity from the previously defined scalar values. A key note is the constant of 5m being applied throughout, which defines a buffer zone where the car aims to stop, such that inside the zone the agent should be stationary. Therefore, braking is relative to the point 5m before the light, such that it stops and is stationary within this zone.

The traffic light scalar, τ, covers all cases in which the agent may encounter, which are, respectively:

- The light being green, where the result is trivially 1

- The agent being within the 5m buffer zone where the result decreases proportionally to the forward velocity, where a 0.1 multiplier is used to exaggerate any movement as opposed to $\frac{1}{\text{max speed}}$
- The agent being outside the buffer zone and approaching the light, where the result decreases proportionally to the amount its forward velocity exceeds the distance
- The agent has driven beyond the red light and is punished proportionally to its distance away
- Otherwise, an assumed value of 1 for when the agent should not consider braking.

5.3.8.1 Change In Distance Consideration

Until this stage of the curriculum, the sole aim of the agent has been to navigate the path where covering the largest distance towards the next target is optimal. In a situation where it needs to slow down relative to lights and even come to a stop, a large change in distance should not be encouraged, as it is optimal to decrease with braking or be 0 when stationary. To overcome this issue, Δd is redefined when the light is red or amber. This is given by the following function, which is applied from this stage of the curriculum:

$$\Delta d = \begin{cases} 0.035 & \text{if the light is red or amber} \\ \text{previously defined } \Delta d & \text{otherwise} \end{cases}$$

The value of 0.035 was chosen specifically to be less than the estimated average value of this change in distance between steps. This was a design choice such that the agent would not be confused and gain a larger reward by remaining stationary and waiting for the next light, i.e., favouring traffic light situations too strongly. However, it is sufficiently large enough to give the agent context in the quality of its actions and ensure it is able to learn the desired behaviour.

5.3.9 Environment Cars

Finally, once the agent can successfully navigate the environment with traffic lights, the reward curriculum addresses the challenge of avoiding collisions with the environment cars. The key difference between this reward scalar and the traffic light reward scalar is the velocity of the car. For the traffic lights, the stopping point remains fixed, so only the velocity of the agent is relevant. In the context of following another car, the velocity of the other car is considered: if the car in-front is slower, braking may need to be applied; if the car is faster, it does not need to consider braking. Therefore, instead of using the agent's velocity, the *relative* velocity to the car in-front is used.

The relevant car is chosen by a searching algorithm that linearly searches through the other cars, checking if their orientation relative to the agent is within a desired threshold (85˚) and if the car is within a defined viewing angle (implemented as 75˚ in each direction). The closest car that meets this criterion is chosen; otherwise null if none do.

The final stage of the curriculum implements the shape reward function defined in figure 19 using a 7.5m stop distance similar to the buffer zone described in Section 5.3.8.

$$reward = \begin{cases} -1 \times |\Delta d \times \theta \times \lambda \times \tau \times \gamma| & \text{if } \Delta d < 0 \text{ or } \theta < 0 \text{ or } \lambda < 0 \text{ or } \tau < 0 \text{ or } \gamma < 0 \\ \Delta d \times \theta \times \lambda \times \tau \times \gamma & \text{otherwise} \end{cases}$$

$$\gamma = \begin{cases} 1 - max(0, v_r) & \text{if } 0 \leq d \leq 7.5 \\ \frac{d-7.5}{max(d-7.5, v_r)} & \text{otherwise} \end{cases}$$

- v_r = velocity relative to the car in-front
- d = distance to the car in-front

Figure 19: Environment cars reward function

6 Results

6.1 Evaluation Metrics

Various metrics can be used to evaluate the performance and learning progress of the agent. These metrics provide insights into different aspects of the agent's effectiveness in completing its current objective. Tensorboard (section 3.4) provides a range of graphs from the training process, where the most important indicator is the cumulative reward achieved by the agent throughout training.

The cumulative reward serves as a measure of the agent's success against the defined reward system, representing the quality of the agent for each episode. Monitoring the trend of cumulative reward over training episodes allows assessment of the agent's learning process, the success of each reward system, and the hyperparameters used.

This cumulative reward is represented on the y-axis, and the x-axis represents the number of steps or actions taken by the agent. The x-axis, therefore, can be viewed as a representation of training time as actions are taken at regular intervals.

In addition to cumulative reward, other metrics such as the extrinsic value estimate and curiosity forward loss also show different aspects of the agent's performance and learning dynamics. However, due to the better representation of cumulative reward and its direct correlation with the agent's overall success, it remains the primary focus for evaluating the project's outcomes and assessing the agent's learning quality.

6.2 Reward System Analysis

6.2.1 Weighted Average

The weighted average reward function (defined in Section 5.3.2) encountered challenges during the training process. One of the primary reasons for the limitation was the blind implementation and lack of focus by providing the agent with a generalised set of encouraged behaviours as opposed to a quantitative analysis of its performance. The cumulative reward through the training process is represented in figure 20.

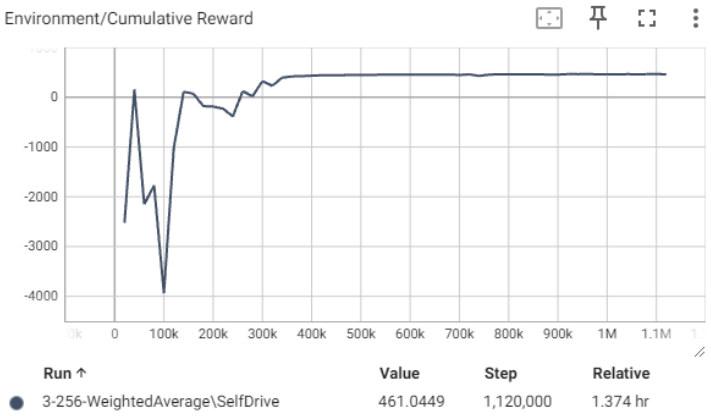


Figure 20: Cumulative reward graph for the weighted average reward function

The graph in figure 20 demonstrates large instability in the initial stages of training, illustrated by the large fluctuations in cumulative reward, followed by a plateau at around 500 cumulative reward. This appears to be an arbitrary value, however, it roughly equates to the agent driving half-way around the circuit before crashing as it was unable to learn to navigate the section with shorter road segments.

The main reason for failure is the scope of the reward function, as it generally encourages behaviours without conceptualising their context in terms of the agent's goals. Another primary cause was the initial implementation of the hyperparameters; some generic values were chosen within ranges commonly implemented in PPO systems, which became problematic for learning quality. Suboptimal values for parameters such as learning rate, batch size, and buffer size hindered the agent's ability to achieve a desirable performance.

6.2.2 Positive Weighted Average

The positive weighted average reward function, defined in Section 5.3.3, was an improvement on the weighted average reward and focused on giving a positive reward such that the agent could gauge its progress rather than crashing quickly so as to accumulate less of a negative reward. The cumulative reward for the training with this reward function is shown in figure 20.

This reward function showed an improvement as the agent was able to make it further around the path. However, it was limited by the agent's decision to remain stationary at a certain point. This could be due to the agent still obtaining an increasing cumulative reward and thus having the illusion of making progress. Specifically, a positive reward could still be accumulated based on the distance and angle to the next target, even without a reward for speed. This caused the agent to time-out after remaining stationary for a prolonged period of time and not making any further progress.

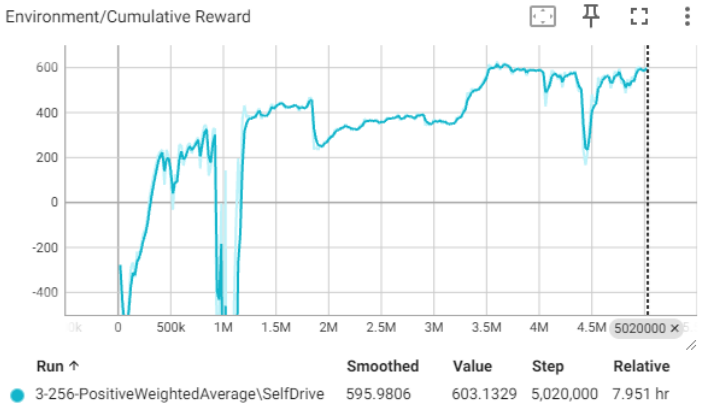


Figure 21: Cumulative reward graph for
the positive weighted average reward function

A key feature to notice is the huge deviation (reaching -4000 reward) around the 1M step mark. This emphasised the instability of the training configuration, likely due to the configuration of the hyperparameters. Aside from this deviation, this reward function marked an improvement where the agent was to navigate a greater portion of the training path before the plateau where it could not learn to make further progress. It also showed a faster learning rate as the shaping reward provided a clearer direction for the agent to learn.

6.2.3 Scaled Reward Function

The scaled reward function, defined in Section 5.3.4, provided a foundation for the remainder of the project. The change in distance towards the next target provided a representation of progress towards the next target. If the car was slower or stationary, it would not achieve any reward, thus overcoming the problem with the weighted average system.

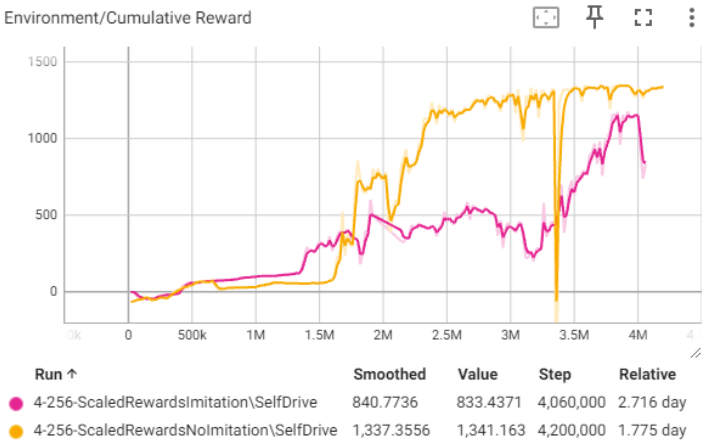


Figure 22: Cumulative reward graph for the scaled reward function

6.2.3.1 Imitation Analysis

Figure 22, shows two plots representing the system using imitation learning, in pink, and not using imitation learning, in yellow. Until this point, the reward systems used imitation learning based on a demonstration of a human driving around the training path, to which it could compare its accuracy against the "correct" behaviour (discussed in Section 5.2.3.3). This result shows greater success when not using imitation learning, and thus it was not used for the remainder of the project.

6.2.3.2 Analysis of the Function

This analysis will discuss the result that did not use imitation learning (in yellow), which resulted in an agent which successfully navigated the complete training lap. This resulted in an accumulated reward of approximately 1300, which was on par with the reward given by a human driving the same route. It can be observed that once learning the key concepts to navigate the first turn (up to 1.5M steps), it quickly learnt to navigate the entire circuit as represented by the relatively sharp increase in cumulative reward from 1.5M to 2.5M steps. This result proved the success of the system for learning which could be applied for the curriculum.

6.2.4 Lane-Centering Curriculum

This stage of the curriculum involved introducing the agent to both slowing at corners/junctions and staying in the centre of the lane.

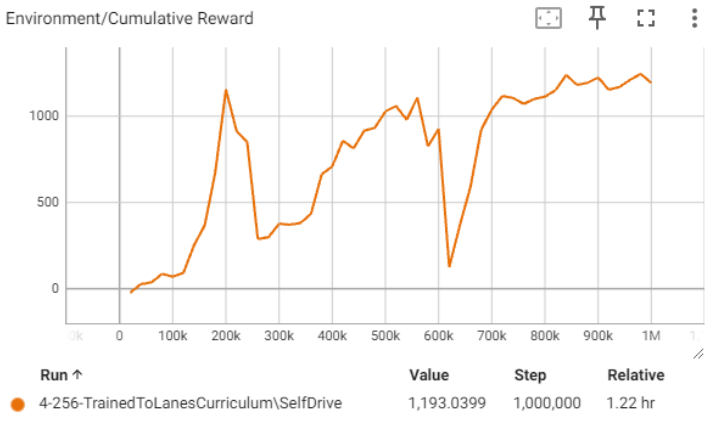


Figure 23: Cumulative reward graph for the lane centring curriculum

Figure 23 shows 3 key stages of the training process:

- Steps 0-200k: The agent learnt to follow the path as defined by the previous scaled reward function.
- Steps 200k-600k: The agent was introduced to slowing at corners and junctions as described in Section 5.3.6.
- Steps 600k-1M: The agent was introduced to the behaviour of lane centring as described in Section 5.3.7.

A key note is the change in scale of the x-axis. Previous training attempts were slow and unstable due to suboptimal hyperparameters and required as many as 5M steps to learn the same behaviours, which were now learnt in as few as 200k steps. This was the result of research into the PPO system before attempting the implementation of this curriculum. The final hyperparameters used are those specified in table 4, which massively improved the training quality. This resulted in no massive deviations, consistent progress, and training speeds up to 20 times faster.

The most notable feature of this graph is the quick rises in cumulative reward after the implementation of each curriculum stage. This represents the desired behaviour of making small adjustments to its already learnt behaviours, resulting in rapid progress. The result of the lane training curriculum was an agent which consistently completed the training circuit, staying close to the centre of the lines as desired.

At the end of the training period, the agent consistently achieved a cumulative reward in the range of 1200 to 1250. This result is slightly lower than the previous results in Section 6.2.3, due to the introduction of an additional scaling factor. With this new scaling factor, the agent is required to maintain near-perfect behaviour consistently to achieve comparable outcomes. The implementation of this scaling system enables effective comparisons across results where, despite the minor decrease in cumulative reward, the agent's performance remains close to optimal, indicating its effectiveness in the newly learnt lane and slowing behaviours.

6.2.5 Traffic Lights Curriculum

This stage of the curriculum posed as a challenge for the agent due to its dynamic nature. The PPO system implements a value network, as described in Section 3.3.2, which tries to predict future rewards. As the agent has no concept of temporal information, it is impossible to predict the seemingly random changes in traffic light signals making it challenging to predict future rewards.

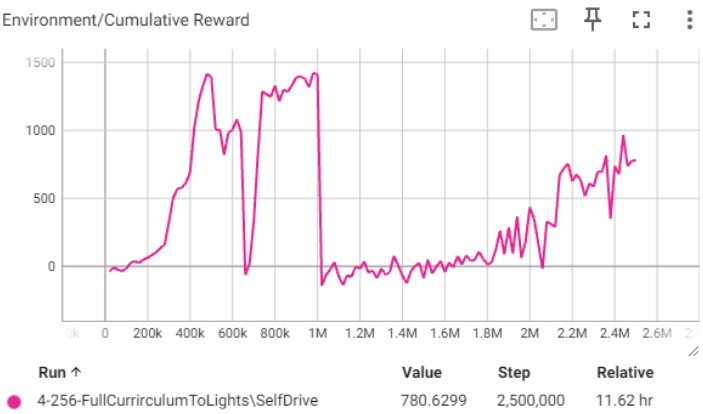


Figure 24: Cumulative reward graph for the weighted average reward function

Figure 24 shows the introduction of the traffic light system at 1M steps, where the agent was already able to follow the path, slow at corners and junctions, and stay close to the centre of the lane. Unlike the previous stages of the curriculum, the agent learnt the new behaviour much more slowly and with a higher level of instability. This is due to the randomness described previously and the punishing nature

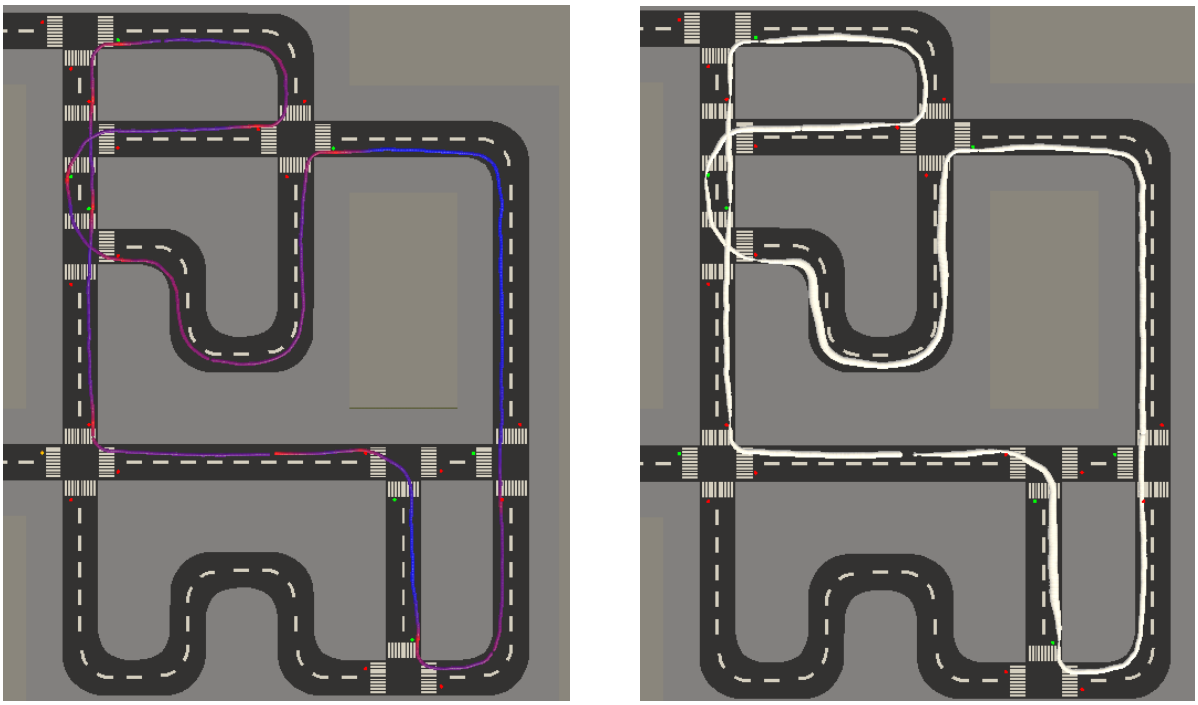
of the reward system, where driving past a red light would result in the accumulation of a significant negative reward.

The training result shows fluctuations as, depending on the episode, small inaccuracies could result in a large deviation in the reward, bringing down the average over the 20,000 step period (each point on the graph). As an additional note, the longer episodes accounted for a larger portion of each 20,000 step period, so slight deviations are more impactful on the average, causing larger fluctuations in the graph.

By 2.5M steps, the agent reached an acceptable level of behaviour consistently completing the circuit, where accumulating a reward of close to 1000 marks a successful completion. This is a lower cumulative reward than the results from the lane centring portion of the curriculum as each time the light was red or amber, a change in distance of 0.035 was used. As explained in Section 5.3.8.1, this value is significantly lower than the average change in distance when driving. This reduced the cumulative reward and thus translated the threshold of suitable behaviour. It is also worth noting that this value of a "successful" cumulative reward depends on the specific episode where it may range from a set of only green lights to waiting for the full time-cycle at each junction, where it could potentially accumulate a larger reward.

6.2.5.1 Testing the Policy

Once the training period ended, the model would choose the most successful policy from the previous 10 policies as its final result. This resulted in the selection of a successful policy that exhibited the correct behaviours. In machine learning, the final result should be tested in an environment that is not the same as the one in which it was trained. For this reason, a different, more complex route was chosen, as seen in figure 25, with more junction paths and turning sections.



(a) Colour representation of the traffic light route

(b) Size representation of the traffic light route

Figure 25: The testing path for the agent with traffic lights

The path of the agent is shown in figure 25, where the colour represents the agent's speed in the first subfigure (figure 25a), with blue indicating maximum speed and red indicating 0 speed. Whereas, in the second subfigure (figure 25b), the width of the path represents the speed.

Figure 25 shows the success of the agent as it achieves all the desired behaviours: reducing its speed around corners and through junctions, stopping at traffic lights when red or amber, and staying within the lane boundaries.

6.2.6 The Final Curriculum

The introduction of the environment cars marked the final stage of the training curriculum and was similarly challenging for the agent. This stage presented a different form of unpredictability in the sense, the cars do not exhibit as random behaviours as the traffic light systems. However, the "relevant" car in-front may change unpredictably as another car takes its place. Also, the traffic light system is predictable in the sense that it is stationary and has limited behaviours, whereas the environment cars have their own velocity and changing distance, which is a challenge to predict for the PPO value network.

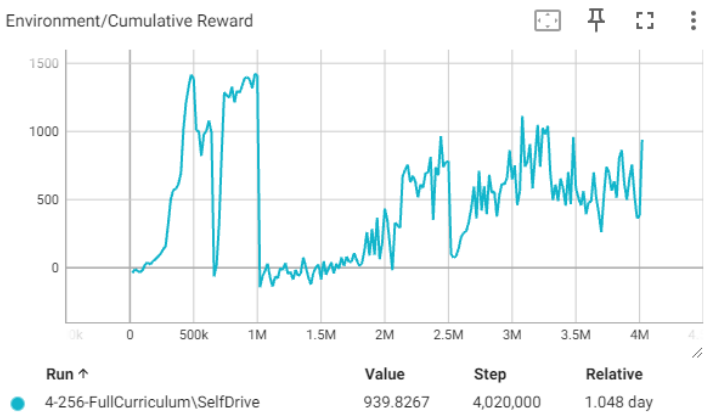


Figure 26: Cumulative reward graph for the weighted average reward function

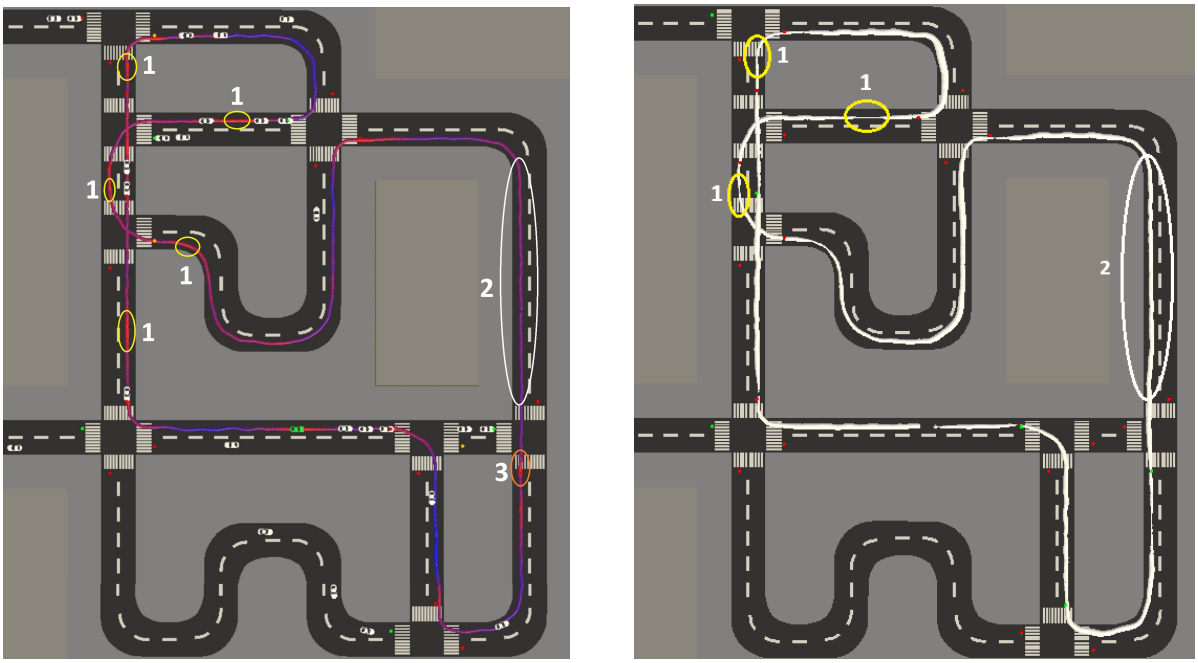
Figure 26 shows the implementation of the environment cars stage of the curriculum at 2.5M steps. Unlike the introduction of traffic lights at 1M steps, it shows a relatively quick increase in cumulative reward reaching the 1000 cumulative reward threshold at 3.1M steps. This can be explained by the new desired behaviour not being so different from the already learnt behaviour from the traffic light stage and due to a less punishing change in the reward system.

However, once reaching the threshold of 1000 cumulative reward, the result shows fluctuations in cumulative reward. This is again a result of the averages being affected more by deviations in reward and due to bugs in the traffic light system where the environment cars would sometimes not trigger the finishing targets. This resulted in the agent being in the queue of cars waiting at the light and timing-out before reaching the junction. The issue could be resolved manually by adjusting the count of active cars at the problematic finish point in the Unity editor, but it still had a significant impact on the training.

Despite this issue, the majority of the training was in a functioning environment where it was able to successfully learn the behaviours of naturally following cars in-front. By the end, the agent was consistently completing the training route, maintaining a safe distance, and driving in a relatively natural manner to accumulate a reward of around 1000, equating to a successful completion of the training route.

6.2.6.1 Testing the Final Policy

In the same way as section 6.2.5.1 was tested, the resulting model at the final stage of the curriculum was tested along the test route.



(a) Colour representation of the final route

(b) Size representation of the final route

Figure 27: The testing path for the final agent

Figure 27 shows the path of the agent with the same colour and size representations as in figure 25. The following major observations can be noted:

- The yellow sections (marked as 1) show key points where the agent successfully waits in queues behind cars at junctions.
- The white section (marked as 2) shows the speed during a straight where it followed another car, this can be compared to figure 25, where in an environment with no other cars, it was not limited by the car in-front's speed.
- The orange section (marked as 3) shows a slight issue with this policy, where occasionally the agent would drive past a red light and correct itself by reversing.

These key observations show the successful behaviour of the final policy, with a slight issue of the agent occasionally driving beyond the traffic lights; however, it knows to correct this error and quickly reverses to position itself correctly behind the light. This is a feature of the specific policy chosen, which would be ironed out with further training and experience.

7 Conclusion

7.1 Main Conclusions

As discussed at the start of this report, the main aim of the project was to successfully implement a complex self-driving agent using the PPO DRL system capable of incrementally introducing behaviours through curriculum learning. This goal was successfully achieved and bridges the gap and limitations of current DRL agents discussed in Section 2.

7.1.1 Imitation Learning

As mentioned in Section 6.2.3.1, the initial reward systems were implemented using imitation learning. This approach involves training an agent by mimicking the actions of a human demonstration, referred to as behavioural cloning. During training, PPO optimises the policy network based on a combination of these imitation and reinforcement learning objectives. The aim is to use the expertise of a human demonstrator to accelerate learning and guide it towards desired behaviour.

However, while imitation learning can be beneficial for guiding the agent towards desired behaviours, it often introduces biases or limitations. This was the case as shown in figure 22, where, when applying imitation learning, the agent was "confused" by the demonstration and learnt at a slower pace, reaching a lower maximum cumulative reward. This could be due to many factors, ranging from the inability to explore a wider range of states to the demonstration being suboptimal, causing the agent to conform to undesired behaviours. In the case of the agent in this project, the latter is more probable as it is difficult to perform a demonstration that aligns optimally with the defined reward system.

For these reasons, imitation was not applied to the final curriculum, which proved successful and shows that it is often the case that imitation learning is not applicable, particularly in complex environments where human behaviour is often suboptimal. However, it could potentially be useful after further experimenting with the strength of the bias towards the demonstration, such that the agent is not biassed so strongly.

7.1.2 Curriculum Learning

Curriculum learning has proven to be a successful methodology in the project, showcasing a structure that allows iterative implementation of new behaviours. Through the curriculum, the agent was able to learn new behaviours, from basic path following to navigating a more complex environment with traffic lights and other cars. This structured approach ensured that the agent attained proficiency in earlier behaviours before confronting more advanced ones through visual analysis or reward thresholds. The success is best represented by the final stage of the curriculum in Section 6.2.6, where the agent successfully encapsulated all the desired behaviours from each stage of the curriculum.

7.1.3 Final Reward System

The final reward system encompasses the implementation of curriculum learning and advancements of rewards at each stage. Specifically, this describes the system of introducing an addition scaling factor for new behaviours, as shown in Section 5.3. This implementation proved to be successful as it provided a solid foundation for any desired behaviour, where the only necessity is defining a normalised scaling factor to represent it. This provides a structure that is easy to extend and allows for relatively easy comparisons between training policies.

7.2 Limitations

Several limitations are presented for the project, warranting consideration for future improvements. Firstly, as detailed in Section 6.2.6, a small bug emerged during the final stage of the curriculum learning process. This setback hindered the agent's ability to thoroughly learn within the training environment and consequentially, the capabilities of the agent.

The nature of PPO, with its reliance on predicting future rewards, poses a significant challenge in an unpredictable and somewhat random environment. Despite its effectiveness in certain contexts, the agent's capacity to accurately predict future rewards may be limited. One potential mitigation strategy could involve reducing the length of the training experiences, meaning the agent is only required to predict the rewards in a shorter, more predictable time period. However, this comes at the cost of the agent's perception of the future, where experiences may not provide sufficient information to learn effectively.

Furthermore, while the training environment aims to simulate real-world driving scenarios, it inevitably operates under large assumptions that do not align completely with reality. For instance, the agent assumes perfect knowledge of the environment, including exact knowledge of road boundary mappings and information about other cars, whereas in real-world applications, such comprehensive knowledge may be subject to inaccuracy. For this reason, the system provides an insight into training methodologies as opposed to a fully realistic environment and an autonomous driving agent.

A key issue with the structure of the neural network applied in this system is the limitation in changes to the network structure. To implement a new behaviour, if new sensor inputs are required, the structure of the neural model needs to be changed. This structural change is not possible for an already trained agent, and the agent needs to be retrained with the new input layer structure. This means that the model must be designed relative to the final stage of the curriculum, which could reduce training efficiency and limit the effectiveness of adding new behaviours.

Finally, the time-intensive nature of training posed a relatively large limitation. Training the agent and addressing performance issues demands considerable time and computational resources. Within the time frame of this project, a successful agent was implemented and trained, but with additional time and more advanced computer systems, a higher level of behaviour could be achieved, and any undesired behaviours ironed out.

7.3 Achievement Discussion

The project has achieved a successful implementation of a curriculum learning reward system, which has enabled the agent to learn a range of complex behaviours essential for this basic abstraction of realistic driving. The developed curriculum effectively guides the agent through progressively challenging tasks: path following, slowing at junctions and corners, lane centring, traffic light adherence, and collision avoidance.

This curriculum system provides the agent with a foundation to learn further behaviours using the easily extendable design of the rewards. The project has demonstrated the feasibility of training a self-driving agent capable of exhibiting behaviours reminiscent of human driving using the PPO DRL model. This achievement lays a solid foundation for future reward systems in autonomous cars and other machine learning environments, and represents the potential of curriculum learning as a powerful tool for training complex AI systems.

8 Evaluation

8.1 Project Management

The project management approach adopted the principles of the agile methodology, as detailed in Section 4. By following a structured framework of weekly sprints, clear goals were set during supervisor meetings for the subsequent week, and the results of the previous sprint were discussed.

This strategy proved effective, allowing for flexibility in the face of challenges and evolving requirements. The iterative nature of the sprints enabled continuous refinement of the project's aims, ensuring alignment with the objectives. Moreover, the regular supervisor meetings provided useful opportunities for feedback, guidance, and improvements as needed. Overall, the agile project management methodology contributed significantly to the project's success by allowing a good balance of planning and flexibility.

8.1.1 Objectives fulfilment

- **Develop A Realistic Simulation Environment:** The project successfully developed a realistic simulation environment by incorporating diverse road layouts and dynamic elements such as traffic lights and other cars. This environment captures some features of real-world driving, providing a complex training environment for the agent.
- **Design and Implement Learning Agents:** The project designed and implemented an agent using the ML-Agents framework, specifically employing PPO. This agent demonstrated the ability to learn and adapt to achieve desired behaviours.
- **Conduct In-Depth Research:** Thorough research into autonomous driving by studying existing literature, real-world self-driving car applications, and exploring related reinforcement learning implementations. Insights from academic papers and open-source projects provided valuable knowledge to help overcome challenges and helped implement a relatively realistic environment.
- **Training and Performance Analysis:** The agent was trained extensively, and Tensorboard was used to visualise the metrics of the agent at each stage. This process involved monitoring training progress, analysing the success and limitations of each iteration, and fine-tuning hyperparameters and the reward system to optimise agent performance.
- **Varied Situations:** The project provided a wide range of driving scenarios, with the road system incorporating a challenging series of junctions, traffic lights, and turns. By using this system, the agent was able to learn complex behaviours and perform effectively in an unknown testing environment.
- **Adapt Agent Based On Feedback:** As discussed, the reward system and hyperparameters were adapted based on feedback received during training and performance evaluation. This iterative feedback allowed for continuous refinement of the agent's behaviour, addressing challenges and improving performance over time.

8.1.2 Progress Evaluation

The original project timeline defined in Section 4 was relatively flexible and did not define the exact implementation of each specific feature, rather giving a general overview of the key stages of the project. Figure 28, however, shows the actual timeline of the implementation of the project.

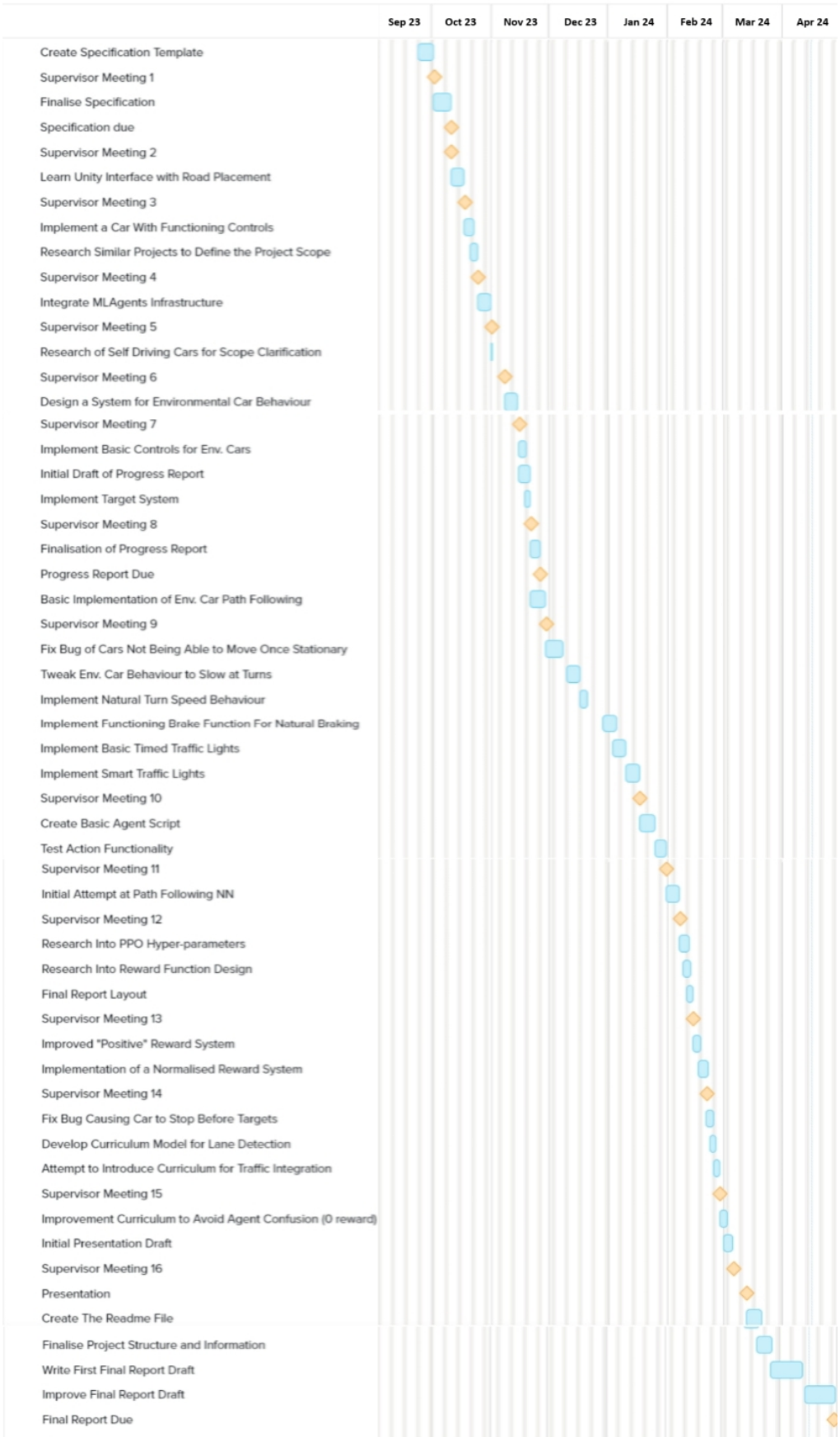


Figure 28: Timeline of Actual Implementation

The yellow diamond markers define key checkpoints along the way, namely the report deadlines and each weekly supervisor meeting. The blue markers detail the specific tasks for implementing the system. It can be seen how strongly these coincide with supervisor meetings, which mark the start or end of most sets of tasks. This emphasises the success of the methodology applied and how weekly goals allowed for effective adaptation by using short-scale goals and analysis.

However, as discussed in 7.2, the simulation environment took longer than expected to implement, requiring 2 months of time (from mid-November to mid-January), as opposed to the original plan in figure 2, which designated 1 month of time for this. This is due to a few unforeseen challenges, and it being during the Christmas period when less work could be achieved.

The delay in the simulation slightly delayed the start of implementing the agent, which began in mid-January. However, this time difference was noted, and a series of smaller goals were created each week to advance in more measurable and predictable steps towards the final goal. Ultimately, this resulted in a more compact work load for the DRL stage of the project, but the issue was foreseen early and thus successfully overcome to finish the project and provide a successful and functioning agent as planned.

8.2 Oversights and Issues

The project encountered a few challenges and oversights during its execution, highlighting areas where improvements could be made in future projects.

8.2.1 Implementation of the Simulation

As discussed in Section 4.3, one significant challenge came from the implementation of the simulation, which took longer than initially anticipated. This delay was primarily due to unexpected issues with the car controller, which required extensive debugging. Furthermore, defining the behaviour of the environment cars revealed numerous edge cases that needed to be accounted for to imitate relatively natural behaviour, further prolonging the implementation phase.

8.2.2 Complexity of Hyperparameter Configuration

Another challenge came from the complexity of configuring the hyperparameters for the PPO system. Due to a lack of experience in machine learning and neural networks, difficulties were encountered in achieving optimal settings, leading to an extensive amount of research using the ML-Agents documentation [47] and iterative adjustments to determine the most effective configuration.

8.2.3 Issues in the Reward Functions

Another challenge came from issues in the reward functions. A main example of this is that described in Section 5.3.6.1, where the denormalising of the change in distance revealed a major flaw in the system. When observing training, it is difficult to find the cause of issues in a reward system. It often takes a considerable amount of time to trial new changes and find the cause of the issue. This limited the rate of progress, as slight issues during the training stage required extensive debugging.

8.2.4 Time Limitations in Training

Moreover, time limitations emerged as a significant constraint, as discussed, a large amount of time was required to train the agent for each small change in implementation. The extensive training periods constrained the project's agility and responsiveness to iterative improvements. This could have been reduced by more experience with AI training and by the use of more powerful computational resources. However, this was not a consequential issue, as it was assumed from the beginning of the project that training would take a relatively long period of time and was therefore designated a large portion of time available for the project.

8.3 Future Work

Looking ahead, there are several directions for further development to create a more advanced system.

8.3.1 Bug Fixing and Optimisation

First and foremost, addressing any existing bugs and optimising the implementation remains a top priority. This includes resolving the bug encountered during the final stage of the curriculum learning process and improving the efficiency of the simulation for more efficient training speeds.

8.3.2 Exploration of Smart Traffic Systems

Exploring the integration of smart traffic light systems represents a promising direction for improving real-world traffic efficiency, which is currently limited by the inefficient and simplistic system currently used. By using machine learning with camera systems, dynamic traffic management systems could lead to more efficient traffic flow, and improve time efficiency for all drivers. This was partially explored in this project, as mentioned in Section 5.1.5.4, but the current implementation is merely a result of avoiding grid lock issues caused by the lack of human adaptiveness in the environment cars.

8.3.3 Integration of New Behaviours

Expanding the agent to include desired behaviours, such as introducing pedestrians, would improve realism within the simulation. Integrating new behaviours can give a better and more complex system, testing the capabilities of the curriculum system designed in this project.

8.3.4 Avoiding Assumptions of Perfect Knowledge

Improving the agent's perception to more realistically represent a real-world scenario would involve a more accurate implementation of environment sensing, replacing the assumptions about perfect knowledge in this project. In a more realistic implementation, decision-making frameworks that account for uncertainty can provide a viable implementation for a valid Sim2Real driving agent.

8.3.5 Exploration of Different Neural Network Architectures

Finally, exploring alternative neural network architectures, such as Long Short-Term Memory (LSTM) networks, offers potential benefits in capturing temporal information and improving the agent's ability to learn complex sequential patterns. Investigating the suitability of different models and architectures could lead to improvements in the agent's learning efficiency and performance, avoiding the limitations posed by PPO.

References

[1] K. Muhammad, A. Ullah, J. Lloret, J. D. Ser, and V. H. C. de Albuquerque, "Deep learning for safe autonomous driving: Current challenges and future directions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4316–4336, 2021.

[2] M. C. for Future Mobility, "The future of autonomous vehicles (av)," *McKinsey*, 2023. [Online]. Available: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenient-and-connected

[3] L. M. Rigoli, G. Patil, H. F. Stening, R. W. Kallen, and M. J. Richardson, "Navigational behavior of humans and deep reinforcement learning agents," *Frontiers in Psychology*, vol. 12, 2021. [Online]. Available: https://doi.org/10.3389/fpsyg.2021.725932

[4] Z. Cao and J. Yun, "Self-awareness safety of deep reinforcement learning in road traffic junction driving," *arXiv preprint arXiv:2201.08116*, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2201.08116

[5] C. Romac, R. Portelas, K. Hofmann, and P.-Y. Oudeyer, "Teachmyagent: a benchmark for automatic curriculum learning in deep rl," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 9052–9063. [Online]. Available: http://proceedings.mlr.press/v139/romac21a.html

[6] D. Shinar, "Crash causes, countermeasures, and safety policy implications," *Accident Analysis and Prevention*, vol. 125, pp. 224–231, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0001457518312028

[7] T. Leys and K. H. News, "How autonomous vehicles could help people with disabilities," *Scientific American*, 2023. [Online]. Available: https://www.scientificamerican.com/article/autonomous-vehicles-give-people-with-disabilities-hope-for-independence/

[8] R. K. Rohini Vijaygopal and R. Bennett, "Driverless cars could be a revolution for people with disabilities – but they also have good reason to be worried," *The Conversation*, 2023. [Online]. Available: https://theconversation.com/driverless-cars-could-be-a-revolution-for-people-with-disabilities-but-they-also-have-good-reason-to-be-worried-213314

[9] S. Chen, X. Hu, J. Zhao, R. Wang, and M. Qiao, "A review of decision-making and planning for autonomous vehicles in intersection environments," *World Electric Vehicle Journal*, vol. 15, no. 3, 2024. [Online]. Available: https://www.mdpi.com/2032-6653/15/3/99

[10] S. A. Bagloee, M. Tavana, M. Asadi, and T. Oliver, "Autonomous vehicles: challenges, opportunities, and future implications for transportation policies," *Journal of Modern Transportation*, vol. 24, 2016. [Online]. Available: https://doi.org/10.1007/s40534-016-0117-3

[11] Y. Almalioglu, M. Turan, N. Trigoni, and A. Markham, "Deep learning-based robust positioning for all-weather autonomous driving," *Nature Machine Intelligence*, vol. 4, no. 9, pp. 749–760, 2022. [Online]. Available: https://doi.org/10.1038/s42256-022-00520-5

[12] L. Kent, "Autonomous cars can only understand the real world through a map," *HERE*, 2023. [Online]. Available: https://www.here.com/learn/blog/autonomous-cars-can-understand-real-world-map

[13] F. Sana, N. L. Azad, and K. Raahemifar, "Autonomous vehicle decision-making and control in complex and unconventional scenarios—a review," *Machines*, 2023. [Online]. Available: https://doi.org/10.3390/machines11070676

[14] J. Engström, R. Wei, A. D. McDonald, A. Garcia, M. O'Kelly, and L. Johnson, "Resolving uncertainty on the fly: modeling adaptive driving behavior as active inference," *Frontiers in Neurorobotics*, vol. 18, 2024. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnbot.2024.1341750

[15] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 41–48. [Online]. Available: https://doi.org/10.1145/1553374.1553380

[16] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu, "Automated curriculum learning for neural networks," 2017.

[17] X. Wang, Y. Chen, and W. Zhu, "A survey on curriculum learning," *arXiv preprint arXiv:2010.13166*, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2010.13166

[18] P. Soviany, R. T. Ionescu, P. Rota, and N. Sebe, "Curriculum learning: A survey," *International Journal of Computer Vision*, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2101.10382
[19] X. Hu, S. Li, T. Huang, B. Tang, R. Huai, and L. Chen, "How simulation helps autonomous driving: A survey of sim2real, digital twins, and parallel intelligence," *CoRR*, 2023. [Online]. Available: http://arxiv.org/abs/2305.01263
[20] J. Chen, T. Wu, M. Shi, and W. Jiang, "Learning personalized autonomous driving behavior with progressively optimized reward function," *Sensors*, vol. 20, no. 19, p. 5626, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/19/5626
[21] A. Names, "Autonomous driving system based on deep q learning," *Journal of Intelligent Transportation Systems*, 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8494053
[22] A. Karalakou, D. Troullinos, G. Chalkiadakis, and M. Papageorgiou, "Deep reinforcement learning reward function design for autonomous driving in lane-free traffic," *Systems*, vol. 11, no. 3, p. 134, 2023. [Online]. Available: https://www.mdpi.com/2079-8954/11/3/134
[23] L. Vogel and C. J. Bester, "A relationship between accident types and causes," in *Proceedings of the 24th Southern African Transport Conference (SATC 2005)*, SATC. Pretoria, South Africa: University of Stellenbosch, 2005, pp. 1–6. [Online]. Available: https://repository.up.ac.za/bitstream/handle/2263/6327/025.pdf
[24] D. Hayashi, Y. Xu, T. Bando, and K. Takeda, "A predictive reward function for human-like driving based on a transition model of surrounding environment," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 7618–7624.
[25] T. Kohsuke, S. Yuta, O. Yuichi, and S. Taro, "Design of reward functions for rl-based high-speed autonomous driving," in *2022 IEEE 15th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, 2022, pp. 32–37.
[26] Tesla. (2024) Autopilot and full self-driving capability — tesla support united kingdom. [Online]. Available: https://www.tesla.com/en_gb/support/autopilot
[27] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2022.
[28] Tesla, "Ai and robotics," 2019. [Online]. Available: https://www.tesla.com/AI
[29] A. Nandy and M. Biswas, *Unity ML-Agents*. Berkeley, CA: Apress, 2018, pp. 27–67. [Online]. Available: https://doi.org/10.1007/978-1-4842-3673-4_2
[30] F. Agostinelli, G. Hocquet, S. Singh, and P. Baldi, *From Reinforcement Learning to Deep Reinforcement Learning: An Overview*. Cham: Springer International Publishing, 2018, pp. 298–328. [Online]. Available: https://doi.org/10.1007/978-3-319-99492-5_13
[31] Y. Tian, M. Shu, and Q. Jia, "Artificial neural network," *SpringerLink*, 2022. [Online]. Available: https://link.springer.com/referenceworkentry/10.1007/978-3-030-26050-7_44-1
[32] D. Urda, F. J. Veredas, J. González-Enrique, J. J. Ruiz-Aguilar, J. M. Jerez, and I. J. Turias, "Deep neural networks architecture driven by problem-specific information," *Neural Computing and Applications*, vol. 33, no. 15, pp. 9403–9423, 2021. [Online]. Available: https://doi.org/10.1007/s00521-021-05702-7
[33] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016. [Online]. Available: https://arxiv.org/abs/1504.00702
[34] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.
[35] M. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, pp. 253–279, 2013.
[36] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2020. [Online]. Available: https://arxiv.org/pdf/1809.02627.pdf
[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[38] Y. Wang, H. He, X. Tan, and Y. Gan, "Trust region-guided proximal policy optimization," *arXiv preprint arXiv:1901.10314*, 2019. [Online]. Available: https://arxiv.org/abs/1901.10314

[39] P. Ramachandran, B. Zoph, and Q. V. Le, "Swish: a self-gated activation function," *arXiv preprint arXiv:1710.05941*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1710.05941

[40] M. Roodschild, J. G. Sardiñas, and A. Will, "A new approach for the vanishing gradient problem on sigmoid activation," *Progress in Artificial Intelligence*, vol. 9, pp. 351–360, 2020. [Online]. Available: https://link.springer.com/article/10.1007/s13748-020-00218-y

[41] U. Technologies, "Using tensorboard to observe training - unity ml-agents toolkit," 2023. [Online]. Available: https://unity-technologies.github.io/ml-agents/Using-Tensorboard/

[42] "What is git? - azure devops — microsoft learn," 2022. [Online]. Available: https://learn.microsoft.com/en-us/devops/develop/git/what-is-git

[43] "What is a distributed version control system? — gitlab," 2023. [Online]. Available: https://about.gitlab.com/topics/version-control/benefits-distributed-version-control-system/

[44] U. Technologies, "Training configuration file," 2024. [Online]. Available: https://github.com/Unity-Technologies/ml-agents/blob/develop/docs

[45] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016. [Online]. Available: https://arxiv.org/pdf/1609.04747.pdf

[46] S. Pollo, "Machine learning: The engine inside tesla's automated driving technology," *Harvard RC TOM*, 2018. [Online]. Available: https://d3.harvard.edu/platform-rctom/submission/machine-learning-the-engine-inside-teslas-automated-driving-technology/

[47] U. Technologies, "Unity ml-agents github," 2024. [Online]. Available: https://github.com/Unity-Technologies/ml-agents