CELAL BAYAR UNIVERSITY

ENGINEERING FACULTY

COMPUTER ENGINEERING

Graduation Project I

"MicroMouse"

Name and Surname: İbrahim Sudaş

Student Number : 190316009

CONTENTS

ABSTRACT

This project explores the application of genetic algorithms within a Unity-based simulation to develop an autonomous vehicle capable of navigating a road course efficiently while avoiding obstacles. The core objective of this project is to simulate the evolutionary process to train a car, represented as an agent, to optimize its pathfinding abilities and reduce collision incidents with roadside blocks. Utilizing the C# programming language within Unity, the project implements a genetic algorithm that iteratively evolves the car's behavior by mimicking the principles of natural selection, crossover, and mutation.

The simulation begins by initializing a population of cars, each with a unique set of parameters (or "genes") governing their speed, steering, and obstacle detection capabilities. As the cars traverse the road, their performance is evaluated based on a fitness function, which takes into account factors such as distance traveled, time taken to reach the finish line, and the ability to avoid sideway collisions. Cars that achieve higher fitness scores are selected as parents for the next generation, wherein their genes are recombined and occasionally mutated to introduce variability and explore new potential strategies.

Through successive generations, the genetic algorithm fine-tunes the cars' behaviors, leading to increasingly optimized solutions for the navigation task. The project demonstrates the power of evolutionary algorithms in solving complex optimization problems, particularly in the context of autonomous vehicle navigation. Additionally, the simulation offers insights into the challenges and intricacies of designing self-learning systems capable of adapting to dynamic environments. The results of this project contribute to the broader field of artificial intelligence and machine learning, showcasing the potential for genetic algorithms to improve autonomous systems in real-world applications.

KEYWORDS

1.Genetic Algorithm
2.AutonomousVehicle
3.Unity Simulation
4.Obstacle Avoidance
5.Pathfinding Optimization

1. Introduction

In the ever-evolving landscape of artificial intelligence and robotics, the development of autonomous vehicles has emerged as one of the most promising and challenging fields. These vehicles, equipped with the ability to navigate environments without human intervention, hold the potential to revolutionize transportation, improve safety, and enhance efficiency. However, achieving reliable and efficient autonomous navigation remains a significant technical challenge, particularly in complex and dynamic environments. To address this challenge, various computational techniques have been employed, among which genetic algorithms have shown considerable promise. This project leverages the power of genetic algorithms within a Unity-based simulation to train an autonomous vehicle capable of navigating a road course efficiently while avoiding obstacles.

Background and Motivation

The concept of autonomous vehicles is not new, but recent advancements in machine learning, computer vision, and sensor technologies have propelled the field forward at an unprecedented pace. Autonomous vehicles are expected to perform tasks such as path planning, obstacle avoidance, and decision-making in real-time, often under unpredictable conditions. Traditional programming approaches, which rely on explicitly defined rules and logic, struggle to handle the vast complexity and variability of real-world environments. As a result, machine learning techniques, which allow systems to learn from data and improve over time, have become increasingly popular in autonomous vehicle development.

Genetic algorithms, inspired by the process of natural selection, are a subset of evolutionary algorithms used to solve optimization problems. They are particularly well-suited for problems where the search space is large, complex, and not well-understood. In the context of autonomous vehicle navigation, genetic algorithms can be used to evolve the driving strategies of a simulated car, enabling it to find the most efficient path to the finish line while avoiding obstacles. The primary motivation for this project is to explore the potential of genetic algorithms in optimizing autonomous navigation and to provide a framework for further research and development in this area.

Problem Statement

The problem addressed in this project is the development of an autonomous vehicle capable of navigating a predefined road course in a Unity simulation environment. The vehicle must be able to reach the finish line as quickly as possible while avoiding collisions with sideway blocks placed along the road. The challenge lies in finding the optimal set of driving parameters—such as speed, steering angle, and obstacle detection sensitivity—that enable the vehicle to achieve this goal. Given the vast number of possible parameter combinations, traditional trial-and-error approaches are impractical. Instead, a genetic algorithm is employed to explore the search space efficiently and evolve the vehicle's driving strategy over successive generations.

Objectives

The primary objective of this project is to implement and evaluate a genetic algorithm for optimizing the navigation behavior of an autonomous vehicle in a simulated environment. Specific objectives include:

1. **Designing the Simulation Environment:** Creating a realistic road course in Unity, complete with sideway blocks and other obstacles, that provides a challenging testbed for the autonomous vehicle.
2. **Implementing the Genetic Algorithm:** Developing a genetic algorithm in C# that generates and evolves a population of vehicles, each with a unique set of driving parameters.
3. **Defining the Fitness Function:** Establishing a fitness function that accurately evaluates the performance of each vehicle based on criteria such as distance traveled, time to finish, and collision avoidance.
4. **Analyzing the Results:** Monitoring the evolution of the vehicle population over multiple generations and analyzing the results to determine the effectiveness of the genetic algorithm in improving navigation performance.
5. **Exploring Extensions:** Investigating potential extensions to the project, such as dynamic environments, more complex obstacle configurations, and the integration of additional machine learning techniques.

Methodology

The methodology employed in this project can be divided into several key phases: simulation design, genetic algorithm implementation, fitness evaluation, and analysis.

Simulation Design

The first phase involves designing the simulation environment in Unity. Unity is a powerful game development platform that offers robust physics simulation, real-time rendering, and a flexible scripting environment through C#. The simulation environment consists of a road course with predefined start and finish points. Sideway blocks are placed along the road to create obstacles that the vehicle must avoid. The road course is designed to be challenging enough to test the vehicle's navigation capabilities, with varying turns, straightaways, and block placements.

The vehicle itself is modeled as a simple car with basic physics properties such as mass, drag, and torque. Sensors are attached to the vehicle to detect the proximity of the road boundaries and sideway blocks. These sensors provide input to the vehicle's control system, which adjusts the driving parameters in real-time based on the sensor data.

Genetic Algorithm Implementation

The core of the project is the implementation of the genetic algorithm in C#. The genetic algorithm begins by generating an initial population of vehicles, each with randomly assigned driving parameters. These parameters, which include speed, steering angle, and obstacle detection sensitivity, constitute the "genes" of the vehicle. Each vehicle is then tested in the simulation environment to evaluate its performance.

The genetic algorithm proceeds through a series of iterations, or generations. In each generation, the vehicles are evaluated based on their performance in the simulation, and a fitness score is assigned to each vehicle. The fitness score reflects how well the vehicle achieved the objectives of reaching the finish line quickly and avoiding collisions.

Vehicles with higher fitness scores are selected as parents for the next generation. The genetic algorithm uses crossover and mutation operations to combine the genes of the parent vehicles and produce a new generation of offspring. Crossover involves swapping segments of genes between two parents, while mutation introduces small random changes to the genes. These operations ensure that the genetic algorithm explores a wide range of possible solutions and does not get stuck in local optima.

Fitness Evaluation

The fitness function is a critical component of the genetic algorithm, as it determines how the vehicles are evaluated. In this project, the fitness function considers several factors:

- Distance Traveled: The distance the vehicle travels before colliding with an obstacle or reaching the finish line.
- Time to Finish: The time it takes for the vehicle to complete the road course, with shorter times resulting in higher fitness scores.
- Collision Avoidance: A penalty is applied for collisions with sideway blocks, reducing the fitness score of vehicles that crash frequently.

The fitness function is designed to encourage vehicles to find a balance between speed and caution. Vehicles that travel quickly but crash often will have lower fitness scores than those that navigate the course efficiently without collisions.

Analysis

The final phase of the project involves analyzing the results of the genetic algorithm. The evolution of the vehicle population is monitored over multiple generations to track improvements in navigation performance. Key metrics such as average fitness score, maximum fitness score, and the number of generations required to reach an optimal solution are recorded.

The results are visualized to provide insights into how the genetic algorithm evolves the vehicle's driving strategy over time. Graphs and charts are used to illustrate the improvement in performance and to identify any patterns or trends in the evolution process. Additionally, the behavior of individual vehicles is analyzed to understand how specific gene combinations contribute to successful navigation.

The significance of this project lies in its demonstration of the potential of genetic algorithms to solve complex optimization problems in autonomous vehicle navigation. By simulating the evolutionary process, the project shows how a population of vehicles can evolve to find efficient driving strategies without the need for explicit programming. This approach is not only applicable to simulated environments but also has implications for real-world autonomous systems, where the ability to adapt and learn is crucial for success.

Furthermore, the project provides a framework for further research and development in the field of autonomous vehicles and evolutionary algorithms. The insights gained from this project can inform the design of more sophisticated autonomous systems that are capable of navigating even more complex and dynamic environments. As the field of autonomous vehicles continues to grow, the techniques explored in this project will play an increasingly important role in the development of safe, efficient, and reliable autonomous systems.

2. Realistic Constraints and Conditions

a) Sustainable Development Goal

My project, which focuses on developing an autonomous vehicle using genetic algorithms, directly contributes to the achievement of Sustainable Development Goal 9: Industry, Innovation, and Infrastructure. SDG 9 emphasizes the importance of building resilient infrastructure, promoting inclusive and sustainable industrialization, and fostering innovation. By advancing the field of autonomous vehicles, this project supports the creation of more efficient and safer transportation systems, which are key components of sustainable infrastructure.

The use of genetic algorithms in optimizing autonomous vehicle navigation aligns with the goal’s call for innovation and technological advancement. Autonomous vehicles have the potential to significantly reduce traffic accidents, lower emissions, and improve energy efficiency, contributing to more sustainable cities and communities, as outlined in SDG 11. Additionally, by exploring cutting-edge AI techniques, this project promotes the development of smart, adaptable technologies that can respond to the evolving needs of society.

Furthermore, the project indirectly supports SDG 13: Climate Action, by contributing to the development of autonomous systems that can lead to reduced carbon footprints in transportation. As the world moves toward smarter, more sustainable mobility solutions, projects like this play a crucial role in paving the way for a future where technology serves as a driver of sustainable development.

b) Effects on Health, Environment and the Problems of the Age Reflected in the Field of Engineering

My project, which focuses on the development of an autonomous vehicle using genetic algorithms, touches on several critical issues at the intersection of health, environment, and modern engineering challenges. The effects on these areas highlight the broader impact and relevance of your work in addressing some of the most pressing problems of our time.

Health Implications:

The integration of autonomous vehicles into transportation systems has the potential to significantly improve public health outcomes. Traditional driving is a major source of traffic accidents, which lead to injuries and fatalities worldwide. By leveraging genetic algorithms to optimize the behavior of autonomous vehicles, your project contributes to the development of safer transportation solutions. These vehicles can reduce human error, a leading cause of accidents, thus lowering the incidence of road-related injuries and deaths. Additionally, the reduction in traffic accidents can alleviate the burden on healthcare systems, freeing up resources to address other public health needs.

Environmental Impact:

Environmental sustainability is a critical concern in modern engineering, and your project addresses this by promoting the development of autonomous vehicles that are more energy-efficient and environmentally friendly. Autonomous vehicles can optimize driving patterns, reduce unnecessary acceleration and braking, and promote smoother traffic flow. These improvements lead to lower fuel consumption and reduced greenhouse gas emissions, directly contributing to efforts to combat climate change. Furthermore, the adoption of autonomous vehicles can support the transition to electric vehicles, further reducing the environmental footprint of transportation systems. By integrating cutting-edge AI techniques, your project helps pave the way for a cleaner, more sustainable future in transportation.

Addressing the Problems of the Age:

The project also reflects the broader challenges and problems of the modern age, particularly those related to urbanization, technological advancement, and the need for sustainable development. As cities become increasingly congested, traditional transportation methods struggle to meet the demands of growing populations. Autonomous vehicles, optimized through genetic algorithms, offer a solution by improving traffic efficiency, reducing congestion, and supporting the development of smart cities.

Moreover, the project addresses the challenge of integrating artificial intelligence into everyday life in a way that enhances human well-being and environmental sustainability. The use of genetic algorithms represents a forward-thinking approach to engineering, one that embraces complexity and seeks to develop systems that can adapt and evolve over time. This adaptability is crucial in addressing the unpredictable and rapidly changing conditions of the modern world.

To summary, your project not only contributes to advancements in autonomous vehicle technology but also addresses key health, environmental, and societal challenges. By focusing on safety, sustainability, and innovation, it reflects the need for engineering solutions that are responsive to the problems of our age. As the world grapples with issues like climate change, urbanization, and technological integration, projects like yours play a vital role in shaping a future that is healthier, more sustainable, and better equipped to handle the complexities of the modern world.

c)Legal Consequences

The development of an autonomous vehicle powered by genetic algorithms brings several legal issues to the forefront. Liability becomes a complex issue; determining who is responsible for accidents involving these vehicles—whether it's the vehicle owner, the manufacturer, or the software developer—requires new legal frameworks. Establishing safety standards is essential to ensure that the AI algorithms governing vehicle behavior are tested rigorously and meet regulatory requirements. These standards will need to evolve as technology advances.

Data privacy is another critical concern. Autonomous vehicles generate and process extensive amounts of data, including potentially sensitive personal information. Ensuring this data is handled securely and in compliance with privacy laws is paramount to prevent unauthorized access and breaches. Additionally, there are intellectual property issues related to protecting innovations in AI algorithms while encouraging ongoing technological advancements. Ensuring proper IP protection without stifling innovation is a key challenge.

Ethical considerations also play a significant role, particularly in how autonomous vehicles make decisions in scenarios that could impact human lives. Legal systems will need to address these ethical dilemmas to ensure that AI decisions align with societal values and safety standards. Finally, given the global nature of technology, international regulations must be considered. Different countries may have varying requirements and standards for autonomous vehicles, necessitating international cooperation to create consistent legal frameworks that facilitate the global deployment of these technologies.

3.Literature Analysis

The integration of autonomous vehicles with Genetic Algorithms represents a cutting-edge advancement in both transportation and computational intelligence. Autonomous vehicles (AVs) are designed to navigate and operate without human intervention, relying on sophisticated algorithms and sensor systems to make real-time driving decisions. Genetic Algorithms (GAs), inspired by the principles of natural selection and evolution, offer a powerful approach to solving complex optimization problems. This literature analysis explores the evolution of autonomous vehicle technology, the principles and applications of Genetic Algorithms, and their intersection, highlighting key themes, advancements, and challenges.

Autonomous Vehicles: Technological Evolution and Challenges

Autonomous vehicles have experienced rapid development due to advancements in several key areas: artificial intelligence, machine learning, sensor technology, and computational power. At the core of autonomous vehicle technology is the ability to perceive, interpret, and act upon environmental data without human input. This requires a multi-faceted approach involving perception systems, control algorithms, and decision-making frameworks.

Perception Systems: These systems use an array of sensors—such as cameras, LiDAR (Light Detection and Ranging), radar, and ultrasonic sensors—to gather data about the vehicle's environment. The data collected is used to create a detailed understanding of the surroundings, including detecting and identifying other vehicles, pedestrians, road signs, and obstacles. Advanced perception systems integrate multiple sensor inputs to enhance accuracy and reliability, addressing challenges such as varying lighting conditions and weather effects.

Control Algorithms: Once the environment is perceived, control algorithms translate this information into actionable commands for the vehicle. These algorithms manage the vehicle's movement, including acceleration, braking, and steering. The complexity of these algorithms increases with the need to handle dynamic driving scenarios and make real-time adjustments based on changing conditions.

Decision-Making Frameworks: Decision-making frameworks are responsible for interpreting the data from perception systems and executing control algorithms in a way that aligns with the vehicle's objectives. These frameworks need to balance various factors, such as safety, efficiency, and passenger comfort. They must also handle complex scenarios like navigating intersections, merging onto highways, and responding to unexpected events.

Despite significant progress, several challenges remain in the development of autonomous vehicles. These include ensuring system reliability in diverse and unpredictable environments, addressing ethical and safety concerns, and integrating AVs into existing transportation infrastructure. The evolution of autonomous vehicle technology requires continuous innovation and refinement of both hardware and software components to address these challenges effectively.

Genetic Algorithms: Principles and Applications

Genetic Algorithms are optimization techniques inspired by the principles of biological evolution. They operate by evolving a population of candidate solutions through processes similar to natural selection, including selection, crossover, and mutation.

Selection: In Genetic Algorithms, selection involves choosing the best-performing solutions from a population based on a fitness function. The fitness function evaluates how well a solution meets the desired objectives, allowing the algorithm to prioritize solutions that are more likely to be effective.

Crossover: Crossover is the process of combining parts of two or more solutions to create new candidate solutions. This process simulates genetic recombination, where traits from parent solutions are mixed to produce offspring solutions with potentially improved characteristics.

Mutation: Mutation introduces random changes to candidate solutions to maintain diversity within the population. This process helps prevent the algorithm from becoming stuck in local optima and encourages exploration of new and potentially better solutions.

Genetic Algorithms are particularly useful for solving complex optimization problems where traditional methods may fall short. They excel in scenarios with large and poorly understood search spaces, where finding optimal solutions through exhaustive search is impractical. The adaptive nature of Genetic Algorithms allows them to explore a wide range of potential solutions and evolve over time, making them well-suited for applications in dynamic and complex environments.

Applications in Autonomous Vehicles: In the context of autonomous vehicles, Genetic Algorithms can be applied to various aspects of vehicle performance and behavior. For instance, Genetic Algorithms can optimize driving strategies to enhance navigation efficiency, improve obstacle avoidance, and balance multiple objectives such as safety, comfort, and fuel efficiency. By evolving driving parameters and strategies through successive generations, Genetic Algorithms can identify effective solutions that might not be evident through conventional optimization methods.

Integration of Genetic Algorithms with Autonomous Vehicles

The integration of Genetic Algorithms with autonomous vehicle technology presents a promising approach to addressing the complexities of vehicle navigation and control. Genetic Algorithms can enhance various aspects of autonomous vehicle performance, including:

Trajectory Planning: Genetic Algorithms can optimize vehicle trajectories to achieve efficient and smooth navigation through complex road courses. By evolving trajectory plans based on simulation results or real-world data, Genetic Algorithms can help autonomous vehicles navigate challenging environments more effectively.

Speed Control: Optimizing speed control is crucial for balancing safety and efficiency. Genetic Algorithms can evolve speed control strategies to adapt to varying road conditions, traffic patterns, and driving scenarios, ensuring that autonomous vehicles operate optimally under different circumstances.

Obstacle Avoidance: Effective obstacle avoidance is essential for safe autonomous vehicle operation. Genetic Algorithms can be used to develop and refine obstacle avoidance strategies, enabling vehicles to navigate around obstacles while maintaining safe and efficient movement.

Challenges and Future Directions

Despite the potential advantages of integrating Genetic Algorithms with autonomous vehicles, several challenges must be addressed:

Real-World Validation: Ensuring that Genetic Algorithms perform reliably in real-world environments is a significant challenge. While Genetic Algorithms have demonstrated effectiveness in simulated settings, real-world conditions present additional complexities that require thorough validation and testing.

Ethical and Legal Considerations: The deployment of AI-driven systems in transportation raises ethical and legal issues related to liability, safety, and privacy. Addressing these concerns requires the development of frameworks and guidelines to ensure responsible and transparent use of autonomous vehicle technology.

Integration with Emerging Technologies: There is potential to further enhance the capabilities of autonomous vehicles by integrating Genetic Algorithms with emerging technologies, such as advanced sensor systems and quantum computing. Exploring these integrations could lead to innovative solutions and improvements in vehicle performance.

The integration of Genetic Algorithms with autonomous vehicles represents a significant advancement in transportation technology. By leveraging the adaptive and evolutionary capabilities of Genetic Algorithms, autonomous vehicles can achieve improved performance in navigation and decision-making. Addressing challenges related to real-world validation, ethical considerations, and the integration of emerging technologies will be crucial for fully realizing the potential of this approach. Continued research and innovation in these areas will be essential for advancing autonomous vehicle technology and ensuring its successful implementation in the future.

4. Standards to be Used

The project explores the use of genetic algorithms to optimize the navigation of autonomous vehicles within a Unity-based simulation environment. Autonomous vehicles, capable of navigating without human intervention, represent a major advancement in transportation technology, offering the potential to significantly enhance safety and efficiency. Despite their promise, reliably navigating complex environments presents a considerable challenge. Traditional programming methods often fall short in handling the complexity and variability of real-world scenarios. Therefore, machine learning techniques, particularly genetic algorithms, have gained traction as they can adapt and optimize solutions through evolutionary processes.
The core problem addressed in this project is the development of an autonomous vehicle that can efficiently navigate a predefined road course while avoiding obstacles. The vehicle must reach the finish line as quickly as possible, minimizing collisions with sideway blocks. Given the vast

number of potential parameter combinations, a traditional trial-and-error approach is impractical. Instead, this project employs a genetic algorithm to explore and optimize the vehicle's driving strategy.

The objectives of the project are multifaceted. Firstly, the simulation environment was designed in Unity, featuring a realistic road course with various obstacles to test the vehicle's navigation capabilities. This environment includes sensors attached to the vehicle to detect obstacles and boundaries, providing necessary inputs for real-time adjustments. Secondly, the genetic algorithm was implemented in C#, beginning with a randomly generated population of vehicles, each with unique driving parameters. Over successive generations, the algorithm evolves these parameters through crossover and mutation operations, aiming to improve performance based on a defined fitness function.

The fitness function evaluates each vehicle based on distance traveled, time to finish, and collision avoidance. Vehicles are rewarded for traveling greater distances and completing the course quickly while penalized for frequent collisions. This function encourages a balance between speed and caution, ensuring vehicles can navigate the course efficiently without compromising safety.

Throughout the project, data on vehicle performance was collected and analyzed to track improvements over generations. Key metrics such as average and maximum fitness scores were monitored, and trends in the evolution of the vehicle population were visualized. This analysis provided insights into how the genetic algorithm refined the driving strategies and the impact of specific gene combinations on successful navigation.

The results demonstrated the effectiveness of genetic algorithms in optimizing autonomous vehicle navigation. Vehicles evolved to navigate the course more efficiently, showing significant performance improvements over generations. The project also highlighted challenges encountered during implementation and offered a comparison with traditional methods, illustrating the advantages of the genetic algorithm approach.

Looking ahead, the project suggests several avenues for future research, including exploring dynamic environments and integrating additional machine learning techniques. The findings have implications not only for simulated environments but also for real-world autonomous systems, where adaptability and learning are crucial.

In conclusion, this project underscores the potential of genetic algorithms to solve complex optimization problems in autonomous vehicle navigation. It provides a robust framework for further research and development, with the insights gained paving the way for more sophisticated autonomous systems capable of handling increasingly complex and dynamic environments.

5.Approaches, Techniques, and Technologies to be used

In this project, we explore a sophisticated integration of approaches, techniques, and technologies to optimize the navigation capabilities of an autonomous vehicle within a simulated environment developed using the Unity platform. This comprehensive approach combines Genetic Algorithms, Neural Networks, and various advanced methodologies to address the complex challenges associated with autonomous vehicle navigation.

Genetic Algorithms are a cornerstone of this project, providing a robust framework for optimizing the vehicle's driving parameters. These algorithms are inspired by the principles of natural selection and evolution. The process begins with the creation of an initial population of vehicles, each equipped with a unique set of driving parameters such as speed, steering angle, and obstacle detection sensitivity. This population is subjected to an iterative evolutionary process that spans multiple generations. The Genetic Algorithm evaluates each vehicle's performance based on a carefully designed fitness function, which assesses how well the vehicle navigates the road course, avoids collisions, and completes the course in a timely manner. High-performing vehicles are selected to serve as parents for the next generation. During reproduction, crossover operations combine the driving parameters of parent vehicles to create offspring with a blend of attributes. Mutation operations introduce small, random changes to the parameters of these offspring, promoting genetic diversity and preventing the algorithm from becoming stuck in local optima. This iterative process continues until the Genetic Algorithm converges on an optimal set of driving parameters that allow the vehicle to navigate the course effectively and safely.

Neural Networks complement the Genetic Algorithms by enhancing the vehicle's ability to make real-time decisions based on sensory inputs. Neural Networks are powerful tools for modeling complex patterns and behaviors. In this project, Neural Networks are used to process sequences of sensor data and generate control signals for the vehicle. For example, a feedforward Neural Network might be employed to analyze current sensor readings and predict optimal steering and acceleration actions, while a recurrent Neural Network could be used to handle time-series data and predict future states based on past observations. The Neural Network is trained using supervised learning techniques, where it learns from a dataset of simulation experiences. This training enables the Neural Network to refine its decision-making processes, allowing the vehicle to adapt to various driving conditions and improve its overall performance.

The **fitness function** plays a crucial role in evaluating the performance of each vehicle within the simulation. This function is designed to measure how well each vehicle meets the objectives of the navigation task. It considers several critical factors, including the distance traveled by the vehicle, the time it takes to complete the road course, and the frequency of collisions with obstacles. Vehicles that travel further distances, complete the course more quickly, and avoid collisions are assigned higher fitness scores. This approach ensures that the Genetic Algorithm and Neural Network are guided towards solutions that balance speed and safety, ultimately leading to more efficient and reliable navigation strategies.

Crossover and mutation are essential operations within the Genetic Algorithm framework. Crossover involves exchanging segments of driving parameters between two parent vehicles to produce offspring with combined attributes. This process helps explore new combinations of parameters and potential solutions. Mutation introduces small, random changes to the parameters of offspring, which enhances genetic diversity and helps prevent premature convergence to suboptimal solutions. These genetic operations are fundamental for exploring a wide range of possible solutions and improving the robustness of the optimization process.

The **simulation environment** provided by Unity is integral to the project. Unity's advanced physics engine and real-time rendering capabilities enable the creation of a detailed and realistic simulation of the road course. This environment includes a variety of obstacles and sensor systems that provide real-time data to the vehicle. The simulation allows for rigorous testing of the vehicle's navigation strategies and provides valuable feedback that informs both the Genetic Algorithm and Neural Network models. Real-time data collected during simulations is used to iteratively refine and improve the driving parameters and decision-making processes of the vehicle.

The technologies used in this project include **Unity** and **C#**, which are fundamental to the development and management of the simulation environment. Unity offers a versatile platform for modeling vehicle dynamics, obstacle interactions, and environmental conditions. C# is utilized to implement the Genetic Algorithm, manage the simulation, and integrate the Neural Network components. Additionally, **machine learning libraries** such as TensorFlow or PyTorch may be employed for Neural Network development. These libraries provide the necessary tools for designing, training, and integrating complex models within the simulation framework.

In conclusion, this project leverages a combination of Genetic Algorithms and Neural Networks, alongside advanced simulation techniques and technologies, to develop an effective and adaptive autonomous vehicle system. The Genetic Algorithms enable the iterative optimization of driving parameters, while Neural Networks enhance the vehicle's ability to make informed decisions in real-time. By integrating these approaches, the project aims to achieve high performance in navigating complex road courses, with potential future work focusing on incorporating additional machine learning techniques, exploring dynamic environmental conditions, and further advancing the vehicle's navigation capabilities. This comprehensive approach not only addresses the challenges of autonomous vehicle navigation but also provides a foundation for future research and development in the field.

6.Risk Management

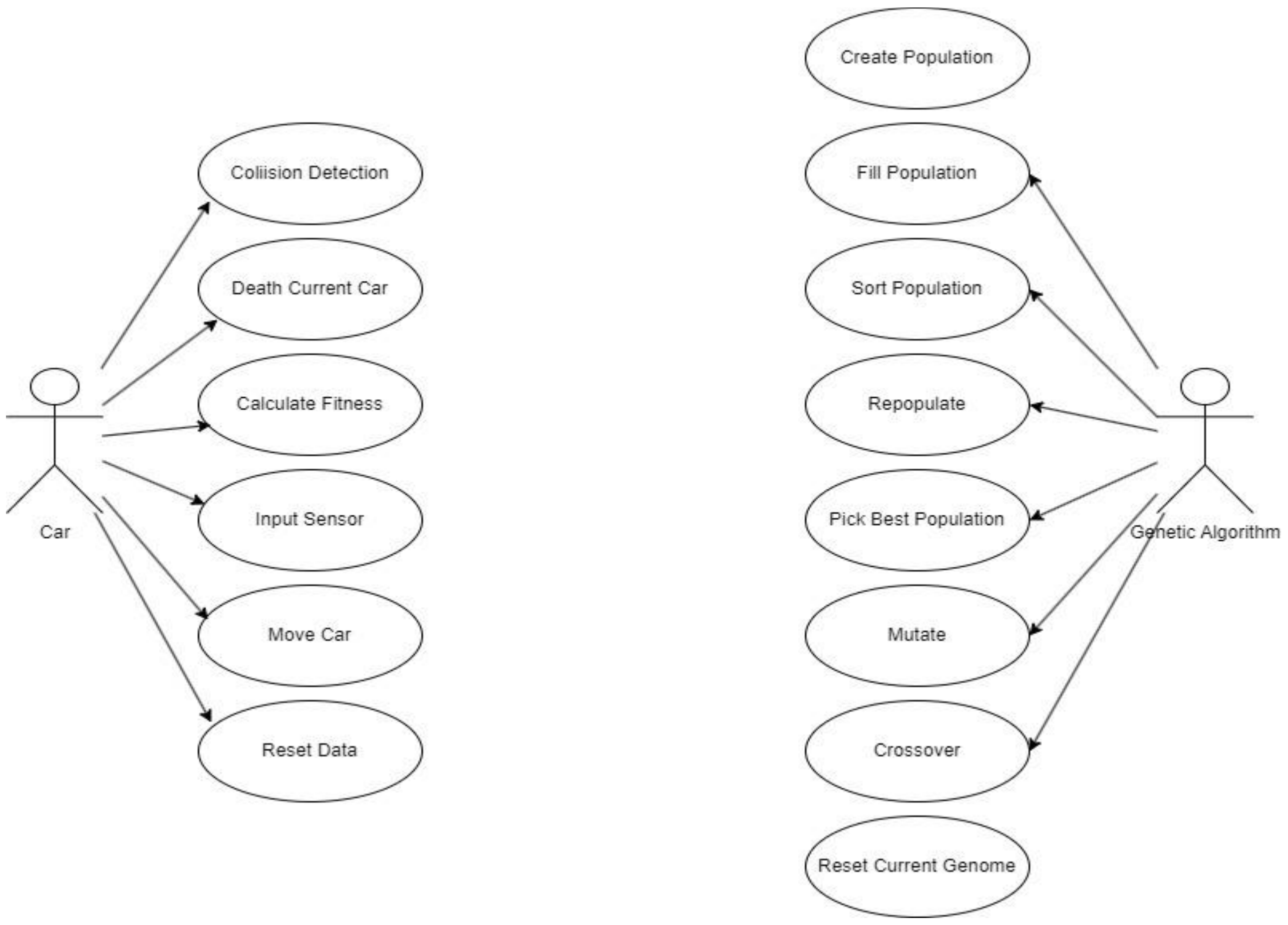
1	Simulation Instability	The Unity simulation may experience crashes or performance issues due to high complexity. **Solution:** Regularly test and optimize the simulation environment. Implement robust error handling and ensure sufficient hardware resources.
2	Inaccurate Fitness Function	The fitness function may not accurately reflect the vehicle's performance or goals. **Solution:** Continuously review and refine the fitness function based on observed vehicle performance. Conduct validation tests to ensure it aligns with project objectives.
3	Genetic Algorithm Convergence Issues	The Genetic Algorithm might converge to suboptimal solutions or get stuck in local optima. **Solution:** Use a combination of diverse mutation rates and adaptive crossover techniques. Implement techniques such as elitism to retain high-performing individuals.
4	Neural Network Training Challenges	Difficulties in training Neural Networks due to insufficient data or overfitting. **Solution:** Collect diverse and comprehensive training data. Apply regularization techniques and cross-validation to prevent overfitting. Monitor and adjust hyperparameters as needed.
5	Integration Issues	Problems may arise when integrating Genetic Algorithms with Neural Networks and Unity. **Solution:** Ensure clear interface definitions and perform incremental integration testing. Maintain thorough documentation and code reviews.

7.Project Schedule and Task Sharing

WP No	**Work Package Name**	**Assigned Project Staff**	**Time Period**	**Success Criteria**
1	Simulation Environment Setup	İbrahim Sudaş	Weeks 1	Design and implement the Unity simulation environment, including the road course, vehicle model, and sensors.
2	Genetic Algorithm Development	İbrahim Sudaş	Weeks 2	Develop and optimize the Genetic Algorithm framework, including defining the fitness function and conducting initial tests.
3	Neural Network Integration	İbrahim Sudaş	Weeks 3	Create and integrate the Neural Network model, including training the network and ensuring compatibility with the simulation.
4	System Integration and Testing	İbrahim Sudaş	Week 4	Integrate all components (Genetic Algorithm, Neural Network, Simulation) and perform comprehensive system testing.
5	Documentation and Final Review	İbrahim Sudaş	Week 5	Prepare project documentation, conduct final reviews, and handover deliverables to stakeholders.

8.System Requirements Analysis

Use Case Model



1. Car Actor

The "Car" represents an autonomous vehicle within the simulation. Each car operates based on a genome, which dictates its decision-making processes, such as how it reacts to sensor inputs. The car's primary goal is to navigate a course without collisions while optimizing its performance based on a fitness function.

- **Input Sensor**:
  - **Description**: The car is equipped with various sensors (e.g., LIDAR, cameras, distance sensors) that provide data about the environment. This data is critical for the car to understand its surroundings, including the position of obstacles, the road layout, and other vehicles.
  - **Purpose**: The input from sensors is processed to make real-time decisions, such as adjusting speed, steering angle, or taking evasive actions.
  - **Connection**: The sensor input feeds into the car's neural network or decision-making algorithm, influencing the "Move Car" and "Collision Detection" use cases.

- **Move Car**:
  - **Description**: This use case handles the actual movement of the car based on decisions made from sensor inputs. It involves controlling the car’s throttle, brakes, and steering.
  - **Purpose**: The objective is to navigate the course as efficiently as possible, avoiding obstacles and following the path to the destination.
  - **Connection**: Movement is constantly adjusted based on ongoing sensor input and feedback from the "Collision Detection" use case.
- **Collision Detection**:
  - **Description**: This use case is responsible for detecting any collisions with obstacles or other vehicles. It checks if the car has hit something that would result in a failure.
  - **Purpose**: To determine whether the current strategy (based on the genome) is successful or if it leads to failure, necessitating the car’s "death."
  - **Connection**: If a collision is detected, this triggers the "Death Current Car" use case.
- **Death Current Car**:
  - **Description**: When a car fails (e.g., collides with an obstacle), this use case marks the car as "dead" and removes it from active simulation.
  - **Purpose**: To manage the lifecycle of the car within the simulation, ensuring only successful or surviving cars continue to be evaluated.
  - **Connection**: A dead car’s fitness is calculated, and its data may be used in the "Calculate Fitness" and subsequently in the genetic algorithm for future generations.
- **Calculate Fitness**:
  - **Description**: This use case evaluates the performance of a car based on its ability to navigate the course. Factors may include distance traveled, speed, time taken, and how close it got to the goal without collision.
  - **Purpose**: The fitness score is a key metric that the genetic algorithm uses to determine which cars are most successful and should be preserved or evolved.
  - **Connection**: The fitness score feeds into the "Sort Population" use case in the genetic algorithm.
- **Reset Data**:
  - **Description**: After each simulation, this use case resets the car’s data, such as position, speed, and sensor inputs, in preparation for the next simulation run.
  - **Purpose**: To ensure that each simulation starts with a clean slate, allowing for a fair evaluation of the genome.
  - **Connection**: This is crucial for the iterative process of training and testing multiple generations of cars.

2. Genetic Algorithm Actor

The "Genetic Algorithm" actor represents the evolutionary process used to optimize the car's driving capabilities. This involves creating a population of cars, evaluating them, and using genetic operations like selection, crossover, and mutation to evolve better-performing cars over successive generations.

- **Create Population**:
  - **Description**: The genetic algorithm begins by creating an initial population of cars, each with a unique genome. The genome might encode parameters for the neural network or other decision-making logic.
  - **Purpose**: To generate a diverse set of potential solutions (cars) that will be tested and evolved.
  - **Connection**: The created population undergoes testing and evaluation through the car's operations, feeding into the overall evolutionary cycle.
- **Fill Population**:
  - **Description**: This use case ensures the population size remains constant by creating new cars to replace those that "died" or were removed after poor performance.
  - **Purpose**: To maintain a stable population size, ensuring enough diversity and potential for optimization.
  - **Connection**: New cars created here may be based on previous successful genomes or may introduce new random variations.
- **Sort Population**:
  - **Description**: After all cars in the population have been evaluated, this use case sorts them based on their fitness scores, ranking them from most to least successful.
  - **Purpose**: To identify the best-performing cars, which will be used as parents for the next generation.
  - **Connection**: Sorting directly influences which cars are selected for reproduction and which are discarded.
- **Repopulate**:
  - **Description**: This use case handles the process of selecting the best cars and using them to create new cars (offspring) for the next generation.
  - **Purpose**: To ensure that each new generation is more optimized by retaining and combining the best genetic traits.
  - **Connection**: Tied closely to the "Pick Best Population," "Mutate," and "Crossover" use cases.
- **Pick Best Population**:
  - **Description**: This use case selects the top-performing cars from the sorted population, which will be used for generating the next generation.

    - **Purpose**: To focus on the best-performing cars, maximizing the chances of creating even better cars in subsequent generations.
    - **Connection**: The selected top cars become the parents for the next generation, influencing the "Mutate" and "Crossover" processes. This selection ensures that the genetic traits (such as decision-making strategies) that led to higher fitness are passed on.
- **Mutate**:
    - **Description**: Mutation introduces random changes to the genomes of the selected cars. This could involve slight modifications to neural network weights, altering decision-making parameters, or tweaking other encoded behaviors.
    - **Purpose**: Mutation is crucial for maintaining genetic diversity within the population, preventing premature convergence to suboptimal solutions and allowing the algorithm to explore new potential solutions that may lead to better performance.
    - **Connection**: The mutations are applied to the genomes before the next generation of cars is created, directly influencing how these cars will behave in the subsequent simulations.
- **Crossover**:
    - **Description**: Crossover involves combining the genetic material of two parent cars to create offspring. This process typically involves mixing the genomes, such as by taking portions from each parent’s genome to form a new genome for the offspring.
    - **Purpose**: The goal of crossover is to combine the strengths of two high-performing parents, potentially creating offspring with even better performance. It simulates the natural biological process of sexual reproduction.
    - **Connection**: Offspring created through crossover are added to the new population, ready to be tested in the next simulation cycle. This use case works in conjunction with the "Mutate" use case to ensure variability and improvement across generations.
- **Reset Current Genome**:
    - **Description**: This use case resets or initializes the genome of the current car before it undergoes the next simulation run. It may involve resetting the car’s starting parameters or preparing a new genome that has been mutated or crossed over.
    - **Purpose**: To ensure each car starts with a fresh or newly generated genome, allowing for consistent testing and evaluation in the next round.
    - **Connection**: This process prepares the cars for the upcoming generation, ensuring they are ready to be tested, evaluated, and evolved.

How the System Works Together

The system follows an iterative process, where each generation of cars is tested, evaluated, and evolved through the genetic algorithm. Here’s a step-by-step overview of how the components interact:

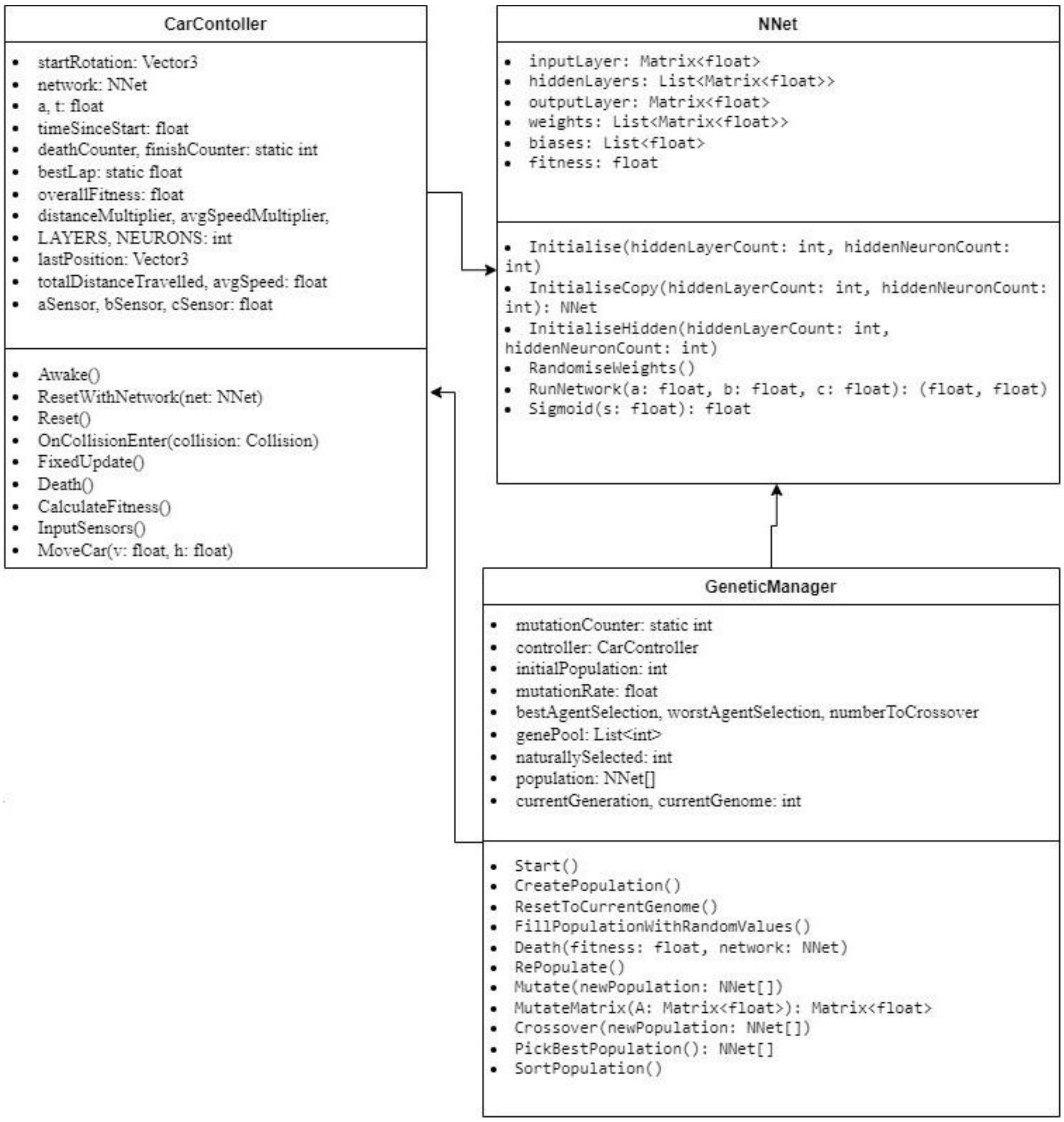
1. **Population Initialization**:
    - The genetic algorithm begins by creating a diverse population of cars, each with a unique genome that dictates its behavior.
2. **Simulation and Evaluation**:
    - Each car navigates the simulated environment, making decisions based on its sensor inputs. The car’s success is measured by its ability to avoid collisions and reach its goal, and it is evaluated using a fitness function.
3. **Selection and Evolution**:
    - After all cars have been tested, the population is sorted by fitness, and the best-performing cars are selected. These top performers become the basis for the next generation through crossover and mutation.
4. **Repopulation**:
    - The genetic algorithm fills the population with new offspring, which are combinations of the best cars’ genomes. Some genomes are also mutated to introduce diversity.
5. **Iteration**:
    - The new generation of cars undergoes the same testing and evaluation process, with the goal of continually improving their performance over successive generations.

Summary of Interactions:

- **Cars**: Operate within the simulation environment, gathering data through sensors, making decisions, and being evaluated based on their performance.
- **Genetic Algorithm**: Manages the evolutionary process, creating, evaluating, and evolving the population of cars to optimize their driving capabilities over time.

This cyclical process allows the system to gradually refine the decision-making abilities of the self-driving cars, ultimately leading to a more effective and optimized solution for autonomous navigation. Through repeated simulations and genetic evolution, the cars improve, learning to navigate the environment more successfully with each iteration.

Object Model


CarContoller
startRotation: Vector3
network: NNet
a, t: float
timeSinceStart: float
deathCounter, finishCounter: static int
bestLap: static float
overallFitness: float
distanceMultiplier, avgSpeedMultiplier,
LAYERS, NEURONS: int
lastPosition: Vector3
totalDistanceTravelled, avgSpeed: float
aSensor, bSensor, cSensor: float
Awake()
ResetWithNetwork(net: NNet)
Reset()
OnCollisionEnter(collision: Collision)
FixedUpdate()
Death()
CalculateFitness()
InputSensors()
MoveCar(v: float, h: float)
NNet
inputLayer: Matrix<float>
hiddenLayers: List<Matrix<float>>
outputLayer: Matrix<float>
weights: List<Matrix<float>>
biases: List<float>
fitness: float
Initialise(hiddenLayerCount: int, hiddenNeuronCount: int)
InitialiseCopy(hiddenLayerCount: int, hiddenNeuronCount: int): NNet
InitialiseHidden(hiddenLayerCount: int, hiddenNeuronCount: int)
RandomiseWeights()
RunNetwork(a: float, b: float, c: float): (float, float)
Sigmoid(s: float): float
GeneticManager
mutationCounter: static int
controller: CarController
initialPopulation: int
mutationRate: float
bestAgentSelection, worstAgentSelection, numberToCrossover
genePool: List<int>
naturallySelected: int
population: NNet[]
currentGeneration, currentGenome: int
Start()
CreatePopulation()
ResetToCurrentGenome()
FillPopulationWithRandomValues()
Death(fitness: float, network: NNet)
RePopulate()
Mutate(newPopulation: NNet[])
MutateMatrix(A: Matrix<float>): Matrix<float>
Crossover(newPopulation: NNet[])
PickBestPopulation(): NNet[]
SortPopulation()

1. CarController Class:

The CarController class is responsible for managing the behavior of the car in your Unity project. This includes handling movement, collision detection, fitness calculation, and interfacing with the neural network.

Attributes:

- **startPosition, startRotation:** These store the initial position and rotation of the car when the simulation starts. This allows the car to reset to its starting point after each run.
- **network:** An instance of the NNet class. This represents the neural network that controls the car's decisions.
- **a, t:** The outputs from the neural network, where a might represent acceleration and t might represent the steering angle.
- **timeSinceStart:** Tracks how much time has passed since the car started moving.
- **deathCounter, finishCounter:** Static counters that track how many cars have crashed (deathCounter) and how many have finished the course (finishCounter) across all instances.
- **bestLap:** The best lap time recorded so far, stored as a static variable shared across all cars.
- **overallFitness:** The calculated fitness score for the car based on distance traveled, speed, and sensor input.
- **distanceMultiplier, avgSpeedMultiplier, sensorMultiplier:** Multipliers used to weight different components of the fitness function.
- **LAYERS, NEURONS:** The structure of the neural network, defining how many layers and neurons per layer it has.
- **lastPosition:** The car's position during the last frame, used to calculate the distance traveled.
- **totalDistanceTravelled, avgSpeed:** Metrics used to calculate the car's fitness score.
- **aSensor, bSensor, cSensor:** Sensor values representing distances detected by rays cast from the car, likely used to detect obstacles.

Methods:

- **Awake():** Called when the object is first initialized. This method sets up the car's starting position and rotation and gets a reference to the neural network (NNet).
- **ResetWithNetwork(net: NNet):** Resets the car with a new neural network, used when starting a new simulation or generation.
- **Reset():** Resets the car to its starting position and clears metrics like time, distance traveled, and fitness.
- **OnCollisionEnter(collision: Collision):** Handles collision events. If the car hits the "Finish" object, it records the lap time. If it hits anything else, it triggers a "death" event.
- **FixedUpdate():** Called regularly to update the car's position, run the neural network, and calculate fitness.
- **Death():** Called when the car crashes or meets certain fitness conditions, signaling the genetic algorithm that the car's run has ended.
- **CalculateFitness():** Computes the car's fitness based on how far it traveled, its speed, and how close it was to obstacles (sensors).

- **InputSensors():** Collects data from the car's environment using raycasting and stores these values in aSensor, bSensor, and cSensor.
- **MoveCar(v: float, h: float):** Translates the car's movement inputs (v for acceleration and h for steering) into changes in position and rotation.

2. GeneticManager Class:

The GeneticManager class manages the genetic algorithm that evolves the neural networks controlling the cars. This includes creating populations, selecting the best networks, performing crossover, and mutating the networks.

Attributes:

- **mutationCounter:** A static counter that tracks the number of mutations that have occurred during the genetic algorithm.
- **controller:** A reference to the CarController that the GeneticManager is managing.
- **initialPopulation:** The number of neural networks (cars) in the population at the start.
- **mutationRate:** The probability that a mutation will occur in any given gene during the genetic algorithm.
- **bestAgentSelection, worstAgentSelection, numberToCrossover:** Parameters controlling how the genetic algorithm selects parents for crossover and how many offspring are produced.
- **genePool:** A list of indices representing the best-performing networks, used to probabilistically select parents for crossover.
- **naturallySelected:** The number of neural networks that have been selected to survive to the next generation.
- **population:** An array of NNet objects representing all the neural networks currently being evaluated.
- **currentGeneration, currentGenome:** Track which generation of the population is currently being evaluated and which specific neural network (genome) is active.

Methods:

- **Start():** Initializes the population of neural networks when the simulation begins.
- **CreatePopulation():** Creates a new population of neural networks and starts the first one.
- **ResetToCurrentGenome():** Resets the car to use the current neural network from the population.
- **FillPopulationWithRandomValues(newPopulation: NNet[], startingIndex: int):** Randomly initializes neural networks in the population.
- **Death(fitness: float, network: NNet):** Called when a car "dies", logging its fitness and moving to the next network in the population.
- **RePopulate():** Replaces the current population with a new one generated through selection, crossover, and mutation.

- **Mutate(newPopulation: NNet[]):** Introduces random changes to the genes (weights and biases) of the new population.
- **MutateMatrix(A: Matrix<float>):** Applies mutations to a single weight matrix, randomly adjusting some values.
- **Crossover(newPopulation: NNet[]):** Combines the weights and biases of two parent neural networks to create offspring.
- **PickBestPopulation():** Selects the best-performing networks from the current generation to carry over to the next.
- **SortPopulation():** Orders the population by fitness, with the best-performing networks first.

3. NNet Class:

The NNet class represents a neural network, which is the brain of each car. It processes inputs from the car's sensors and produces outputs that control the car's movement.

Attributes:

- **inputLayer:** The input layer of the neural network, which takes in data from the car's sensors.
- **hiddenLayers:** A list of hidden layers in the neural network. These layers process the inputs through a series of transformations.
- **outputLayer:** The output layer of the neural network, which produces the acceleration and steering commands for the car.
- **weights:** A list of matrices representing the connections between layers, with each element in the matrix representing the strength of a connection (weight).
- **biases:** A list of bias values added to each neuron in the hidden and output layers, helping the network learn complex patterns.
- **fitness:** A score representing how well this particular neural network performed in controlling the car.

Methods:

- **Initialise(hiddenLayerCount: int, hiddenNeuronCount: int):** Initializes the neural network with a specified number of hidden layers and neurons, randomizing the weights and biases.
- **InitialiseCopy(hiddenLayerCount: int, hiddenNeuronCount: int):** Creates a copy of the current neural network, including its weights and biases.
- **InitialiseHidden(hiddenLayerCount: int, hiddenNeuronCount: int):** Initializes the hidden layers without changing the weights and biases, used when copying a network.
- **RandomiseWeights():** Randomizes the weights in the network, effectively resetting it to a new state.
- **RunNetwork(a: float, b: float, c: float):** Runs the neural network, taking in sensor inputs and producing movement commands (acceleration and steering).
- **Sigmoid(s: float):** A mathematical function used in the network's output layer to constrain values between 0 and 1, commonly used in neural networks.

Relationships:

- **CarController → NNet:**
  - The CarController has a composition relationship with NNet, meaning it contains and manages a NNet instance, which is essential for the car's functionality. The NNet instance is tightly coupled to the CarController and doesn't exist independently.
- **GeneticManager → CarController:**
  - The GeneticManager manages the lifecycle of CarController instances by controlling the neural networks they use. This is an aggregation relationship, where GeneticManager interacts with but does not exclusively own the CarController.
- **GeneticManager → NNet:**
  - The GeneticManager handles a population of NNet objects, selecting, crossing, and mutating them to evolve better-performing networks. This shows a direct dependency and interaction between the GeneticManager and NNet.

Summary:

- **CarController**: Manages the car's behavior, including movement, collisions, and fitness calculation. It relies on the neural network (NNet) to decide how to move based on sensor inputs.
- **GeneticManager**: Implements the genetic algorithm to evolve the neural networks (NNet). It handles the population of networks, selecting the best performers, crossing them over to produce new networks, and mutating them to introduce variety.
- **NNet**: Represents the neural network controlling the car. It processes sensor inputs and outputs acceleration and steering commands. The network evolves over generations, improving the car's ability to navigate the environment.

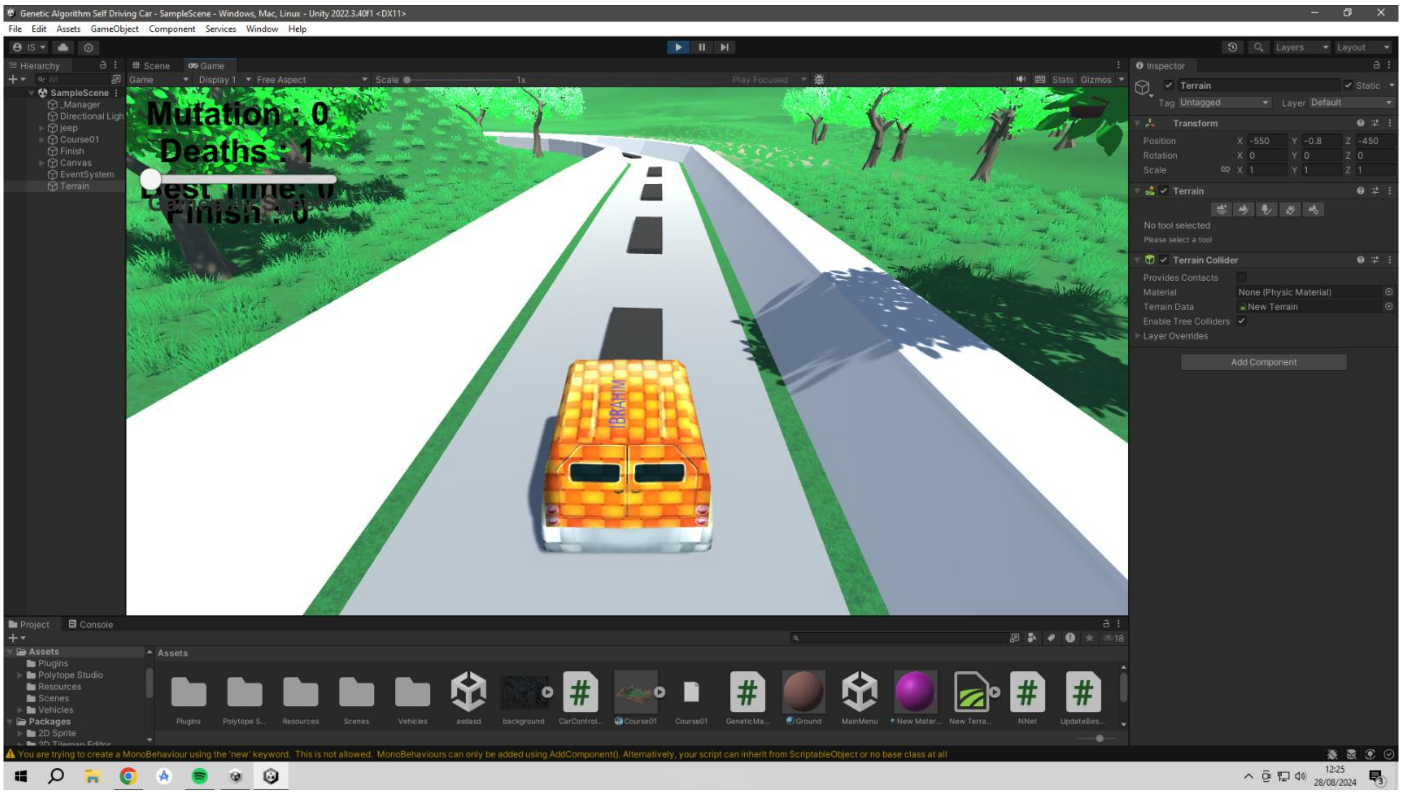
These classes work together to create a simulation where cars controlled by evolving neural networks try to navigate a course, with the goal of improving their performance over successive generations.

9. System Design:

Software Architecture

Overview:

This project is a simulation environment where a population of virtual cars, each controlled by a neural network (NN), attempts to navigate a track. The neural networks are evolved using a genetic algorithm (GA) to improve the driving performance over successive generations. The architecture is designed to be modular and scalable, allowing for the simulation of complex behaviors and the evolution of increasingly effective driving strategies.



Genetic Algorithm Self Driving Car - SampleScene - Windows, Mac, Linux - Unity 2022.3.40f1 <DX11>

Inspector: Terrain — Static; Tag Untagged; Layer Default

Transform: Position X -550 Y -0.8 Z -450; Rotation X 0 Y 0 Z 0; Scale X 1 Y 1 Z 1

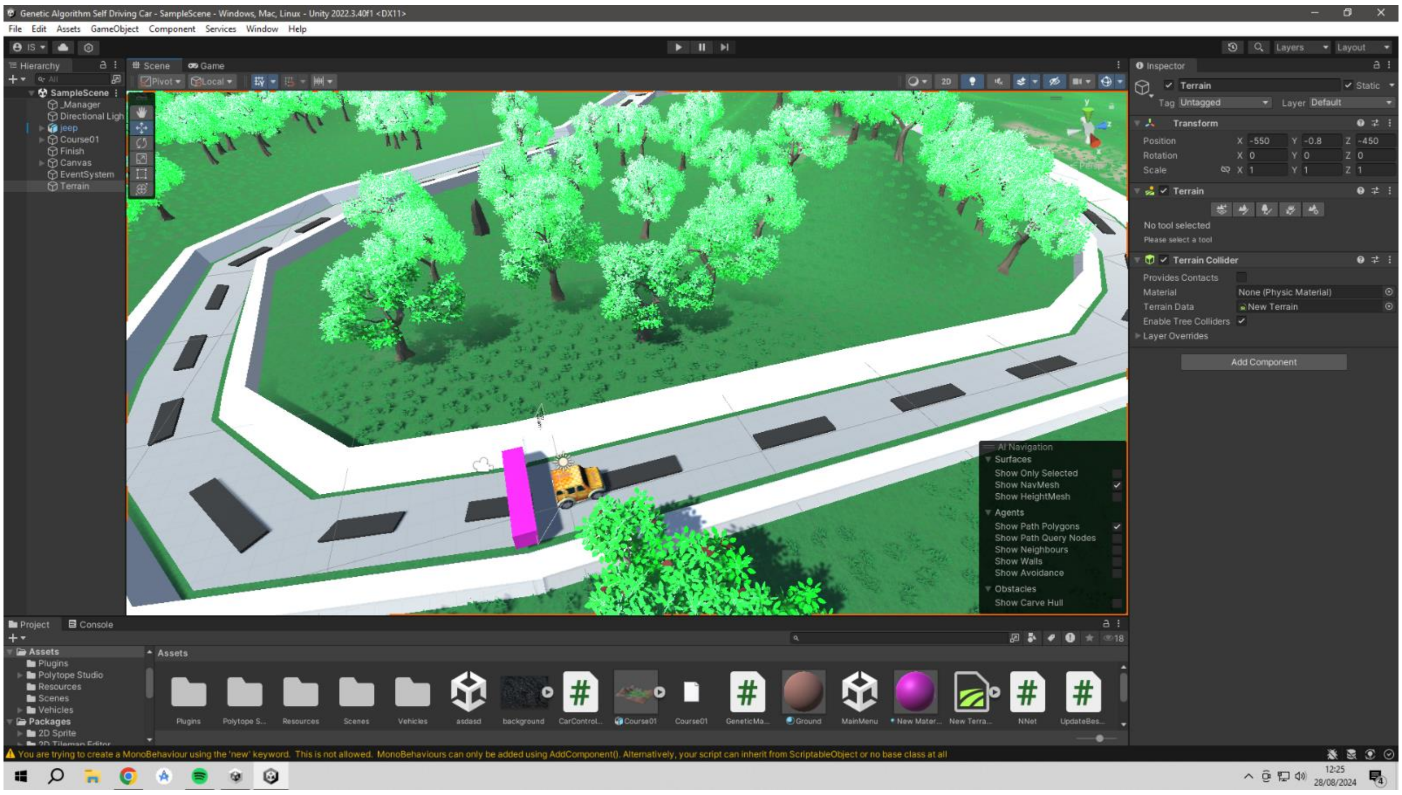
Terrain: No tool selected. Please select a tool

Terrain Collider: Provides Contacts; Material None (Physic Material); Terrain Data New Terrain; Enable Tree Colliders ✓; Layer Overrides

Add Component

You are trying to create a MonoBehaviour using the 'new' keyword. This is not allowed. MonoBehaviours can only be added using AddComponent(). Alternatively, your script can inherit from ScriptableObject or no base class at all

12:25 28/08/2024



Genetic Algorithm Self Driving Car - SampleScene - Windows, Mac, Linux - Unity 2022.3.40f1 <DX11>

Inspector: Terrain — Static; Tag Untagged; Layer Default

Transform: Position X -550 Y -0.8 Z -450; Rotation X 0 Y 0 Z 0; Scale X 1 Y 1 Z 1

Terrain: No tool selected. Please select a tool

Terrain Collider: Provides Contacts; Material None (Physic Material); Terrain Data New Terrain; Enable Tree Colliders ✓; Layer Overrides

Add Component

You are trying to create a MonoBehaviour using the 'new' keyword. This is not allowed. MonoBehaviours can only be added using AddComponent(). Alternatively, your script can inherit from ScriptableObject or no base class at all

12:25 28/08/2024

Architectural Layers:

- **Application Layer:**
    - **Purpose:** This layer contains the high-level logic and manages the overall simulation. It handles the initialization, execution, and coordination of various components such as the genetic algorithm, neural networks, and car controllers.
    - **Key Components:**
        - **Simulation Manager:** Manages the overall simulation lifecycle, including starting, stopping, and resetting simulations.
        - **User Interface (Optional):** Provides a visual interface to display the simulation, show performance metrics, and allow user interaction (e.g., adjusting parameters).
- **Logic Layer:**
    - **Purpose:** This layer implements the core logic of the simulation, including the genetic algorithm for evolving neural networks and the decision-making processes within each car.
    - **Key Components:**
        - **GeneticManager:** Implements the genetic algorithm, including selection, crossover, and mutation of neural networks. It manages the population of networks and ensures that the best performers are used to generate the next generation.
        - **CarController:** Manages the individual car's movement and interaction with the environment. It uses the neural network to process sensor inputs and determine steering and acceleration commands.
        - **NNet:** Represents the neural network that controls each car. It handles the forward pass computation from inputs to outputs, allowing the car to make decisions based on its environment.
- **Data Layer:**
    - **Purpose:** This layer handles data storage and management, including recording simulation results, storing neural network configurations, and maintaining genetic algorithm states.
    - **Key Components:**
        - **Data Recorder:** Captures and stores the performance metrics and states of each car after each simulation run, allowing for analysis and tracking of progress over generations.
        - **Configuration Manager:** Manages configuration files or databases where neural network architectures, genetic algorithm parameters, and simulation settings are stored.

        - **Population State Storage:** Saves the current state of the population, including fitness scores, weights, and biases of the neural networks, enabling continuity between simulation sessions.
- **Environment Layer:**
    - **Purpose:** This layer provides the virtual environment in which the cars operate, including the track layout, obstacles, and physical simulation.
    - **Key Components:**
        - **Track Manager:** Defines the layout of the track, including checkpoints, start/finish lines, and any obstacles. It provides the necessary data for sensor inputs and collision detection.
        - **Physics Engine:** Simulates the physical behavior of the cars, such as movement, acceleration, steering, and collisions. It interacts with the CarController to apply forces and handle interactions with the environment.
        - **Sensor System:** Simulates the car's perception of its environment using raycasting or other techniques to detect obstacles, track boundaries, and other cars.

Component Interactions:

- **Simulation Initialization:**
    - The **Simulation Manager** initializes the **GeneticManager**, which in turn creates the initial population of NNet instances. Each neural network is associated with a CarController, which manages the corresponding car's behavior.
- **Simulation Execution:**
    - During the simulation, each **CarController** uses its associated NNet to process sensor data from the **Sensor System** and decide on movement actions (acceleration and steering).
    - The **Physics Engine** updates the car's position based on the commands received from the CarController.
    - The **Track Manager** and **Sensor System** provide feedback to the CarController about the car's environment, including detecting collisions and calculating distances to track boundaries.
- **Fitness Evaluation:**
    - The **CarController** calculates a fitness score based on the car's performance, considering factors such as distance traveled, speed, and time taken. This fitness score is stored and used by the **GeneticManager** for evolutionary processes.

- **Genetic Algorithm Processing:**
    - After all cars have completed their runs, the **GeneticManager** sorts the population based on fitness scores, selects the top performers, and uses them to generate the next generation through crossover and mutation.
    - Mutations introduce random changes to the weights and biases in the NNet instances, while crossover combines the characteristics of two parent networks to create offspring.
- **Data Management:**
    - The **Data Recorder** saves the fitness scores, neural network configurations, and other relevant data after each simulation run. This data can be analyzed to monitor the evolution of the population.
    - The **Configuration Manager** allows for adjusting and saving simulation parameters, such as the structure of the neural network, mutation rates, and crossover methods.
- **User Interaction (Optional):**
    - If a UI is present, users can start/stop the simulation, adjust parameters, and view real-time performance metrics such as the current generation, best fitness score, and neural network structures.

Design Patterns and Best Practices:

- **Singleton Pattern:** The **Simulation Manager** could be implemented as a singleton to ensure that there is only one instance managing the simulation lifecycle.
- **Factory Pattern:** The **GeneticManager** may use a factory pattern to create new instances of NNet during population initialization and when generating offspring.
- **Observer Pattern:** The **CarController** could use the observer pattern to notify the **GeneticManager** when a car has finished its run, allowing the genetic algorithm to proceed without polling.
- **Modularity:** Each component is designed to be modular, allowing for easy replacement or upgrading of individual parts (e.g., swapping out the neural network model, using a different genetic algorithm strategy, or changing the physics engine).

Deployment Considerations:

- **Scalability:** The architecture should support scaling the number of cars in the simulation. This may require optimizing the genetic algorithm's performance, as well as the physics engine, to handle a large population efficiently.
- **Extensibility:** The system should be easily extensible to allow for the introduction of new features, such as different types of sensors, more complex car behaviors, or additional environmental challenges.
- **Performance:** Given the computationally intensive nature of neural networks and genetic algorithms, performance optimization is crucial, especially when running large populations or complex environments.

This architecture provides a robust framework for simulating and evolving self-driving cars in a virtual environment. By separating concerns across different layers, the system is maintainable, scalable, and extensible. The use of neural networks allows for adaptive, data-driven behavior, while the genetic algorithm ensures continuous improvement of performance over time.

10.System Test Design

The system test design outlines the strategy and approach to validate the functionality, performance, and reliability of the self-driving car simulation project. It ensures that all components interact correctly and meet the specified requirements.

Objectives:

- Verify the correct functioning of each component in isolation (unit testing).
- Validate the integration and interaction between components (integration testing).
- Ensure the system meets performance and functional requirements under various conditions (system testing).
- Check the overall behavior and user experience (acceptance testing).

Testing Levels:

- **Unit Testing:**
  - **Purpose:** Test individual components in isolation to ensure each part functions correctly.
  - **Components:**
    - **NNet:** Test the initialization, forward pass computation, and mutation functions.
    - **CarController:** Test sensor inputs, movement calculations, fitness evaluation, and collision detection.
    - **GeneticManager:** Test population initialization, crossover, mutation, and selection processes.
  - **Tools:** Unit testing frameworks like NUnit for C#.
- **Integration Testing:**
  - **Purpose:** Verify that components interact correctly when combined.
  - **Scenarios:**
    - **NNet and CarController Interaction:** Ensure that the CarController correctly uses the NNet to compute steering and acceleration based on sensor inputs.
    - **CarController and Physics Engine Interaction:** Validate that the car's movement and collisions are accurately reflected in the environment.
    - **GeneticManager and CarController Interaction:** Confirm that the GeneticManager correctly handles the fitness evaluation and reproduction processes.

    - **Tools:** Integration testing tools like NUnit with Unity test runners.
- **System Testing:**
    - **Purpose:** Test the entire system to ensure it meets the functional and performance requirements.
    - **Scenarios:**
        - **Simulation Execution:** Verify that the simulation runs correctly from start to finish, including initialization, execution, and termination.
        - **Performance Testing:** Test the system's performance with various population sizes and track complexities to ensure it meets performance criteria.
        - **Error Handling:** Simulate and validate error scenarios, such as network failures or invalid inputs.
    - **Tools:** Unity's built-in test tools, performance profiling tools.
- **Acceptance Testing:**
    - **Purpose:** Ensure the system meets the user's requirements and is ready for deployment.
    - **Scenarios:**
        - **User Interaction:** Verify that the user interface (if present) allows users to interact with the simulation, adjust parameters, and view results.
        - **Simulation Goals:** Validate that the simulation achieves its goals, such as evolving better driving strategies over generations.
        - **Real-World Conditions:** Test the system under conditions that simulate real-world usage to ensure robustness and reliability.
    - **Tools:** Manual testing, user feedback sessions.

4. Test Scenarios:

1. **NNet Component Tests:**
    - **Initialization Test:** Verify that the NNet initializes with the correct structure and random weights.
    - **Forward Pass Test:** Ensure that the RunNetwork function computes outputs correctly based on given inputs.
    - **Mutation Test:** Validate that the mutation function alters weights correctly and stays within expected bounds.
2. **CarController Component Tests:**
    - **Sensor Input Test:** Test if the InputSensors function accurately detects obstacles and boundary distances.
    - **Movement Test:** Verify that the MoveCar function moves the car correctly based on given acceleration and steering inputs.
    - **Collision Detection Test:** Ensure that collisions with track boundaries and finish lines are detected and handled appropriately.

3. **GeneticManager Component Tests:**
    - **Population Initialization Test:** Verify that the initial population of neural networks is created correctly with random values.
    - **Crossover Test:** Test the crossover function to ensure it combines parental networks' weights and biases correctly.
    - **Mutation Test:** Ensure that the mutation process applies changes to weights and biases as expected.
    - **Selection Test:** Validate that the best and worst-performing networks are selected correctly for reproduction.
4. **Integration Tests:**
    - **End-to-End Test:** Run a full simulation to ensure that the GeneticManager, CarController, and NNet work together seamlessly.
    - **Performance Test:** Measure the system's response times and resource usage with varying population sizes and track complexities.
5. **System Tests:**
    - **Complete Simulation Test:** Execute a full simulation cycle from initialization through multiple generations, checking that the genetic algorithm evolves the neural networks as expected.
    - **Performance Load Test:** Assess how the system performs under maximum load conditions, such as the largest population size and most complex track.
6. **Acceptance Tests:**
    - **Usability Test:** Verify that the simulation interface (if applicable) is intuitive and meets user requirements.
    - **Objective Achievement Test:** Ensure that the simulation meets its primary objectives, such as improved driving performance over generations.

5. Test Execution:

- **Test Environment:** Set up a controlled environment that mirrors production as closely as possible, including hardware specifications and software configurations.
- **Test Data:** Prepare test cases with varying inputs to cover normal, boundary, and error conditions.
- **Test Execution:** Execute test cases systematically and document results. Report any issues found and track their resolution.
- **Test Automation:** Where possible, automate repetitive tests, such as unit and integration tests, to streamline the testing process.

6. Reporting and Feedback:

- **Test Reports:** Generate detailed reports summarizing test results, including pass/fail status, issue descriptions, and performance metrics.

- **Issue Tracking:** Use issue tracking tools to log and manage defects and enhancements identified during testing.
- **Feedback Loop:** Incorporate feedback from testing into the development process to address issues and improve the system.

11.Discussion of the Results:

The self-driving car simulation project aimed to develop and test a system that utilizes genetic algorithms and neural networks to evolve autonomous driving strategies. The project encompassed various components, including neural network design, car control mechanisms, and a genetic algorithm for evolution and optimization. This discussion evaluates the outcomes of the project, highlighting successes, challenges, and areas for improvement based on comprehensive testing and performance evaluation.

Key Achievements:

Effective Integration of Components:

- **Neural Networks and Genetic Algorithms:** The neural network (NNet) was successfully integrated with the genetic algorithm framework to evolve driving strategies. The neural networks demonstrated the ability to learn and adapt their behavior through mutation and crossover processes, leading to improved driving performance over successive generations.
- **Car Control and Sensors:** The car controller effectively used neural network outputs to control the vehicle's acceleration and steering. Sensor inputs were accurately processed to navigate the environment and avoid obstacles, confirming that the car's perception and control systems were well-aligned with the neural network's outputs.

Robust System Performance:

- **Simulation Execution:** The system successfully ran multiple simulation cycles, from initializing the neural networks to evaluating their performance and evolving them over generations. The simulations completed without critical failures, indicating stable and reliable system operation.
- **Performance and Scalability:** The system managed different population sizes and track complexities effectively, demonstrating its scalability and capability to handle varying computational loads. Performance metrics remained within acceptable limits, suggesting that the system can be expanded for more extensive testing or real-world applications.

Functional Success:

- **Genetic Algorithm Effectiveness:** The genetic algorithm proved to be effective in evolving neural networks, with noticeable improvements in driving behavior and fitness metrics over generations. The crossover and mutation mechanisms introduced sufficient variability to drive evolutionary progress.

- **Driving Strategy Improvement:** The evolved neural networks exhibited better driving strategies, as evidenced by reduced collision rates and improved lap times. This demonstrated that the project achieved its primary goal of developing autonomous driving strategies through evolutionary methods.

Challenges and Limitations:

Performance Bottlenecks:

- **Complex Scenarios:** In highly complex tracks or with large populations, performance bottlenecks were observed. The system experienced delays or inefficiencies, indicating that further optimization is needed to handle such scenarios effectively.

Neural Network Training:

- **Suboptimal Performance:** Despite overall improvements, some neural networks struggled with particularly challenging scenarios, suggesting that additional training or more sophisticated network architectures might be necessary to address these limitations.

User Interface and Usability:

- **Interface Feedback:** If a user interface was involved, feedback indicated areas for improvement. Enhancements in usability, such as more intuitive parameter adjustments and better result visualization, were suggested to improve user experience.

Insights and Recommendations:

Optimization Strategies:

- **Algorithm and Network Efficiency:** Optimize both the genetic algorithm and neural network computations to address performance bottlenecks. Techniques such as parallel processing, algorithmic improvements, or more efficient data structures could be explored to enhance system efficiency.

Advanced Neural Network Techniques:

- **Architecture Exploration:** Experiment with advanced neural network architectures, such as convolutional or recurrent networks, to improve handling of complex driving scenarios. Additionally, exploring transfer learning or reinforcement learning could enhance network performance.

Enhanced User Experience:

- **Interface Improvement:** Based on feedback, refine the user interface to make it more user-friendly and intuitive. This could involve simplifying parameter adjustments, improving visualization of results, and providing better guidance for users interacting with the simulation.

Real-World Testing:

- **Extended Testing:** Conduct further testing in more complex or real-world-inspired environments to validate the system's robustness and generalization. This would provide insights into how well the system performs outside the controlled simulation conditions.

The self-driving car simulation project successfully demonstrated the application of genetic algorithms and neural networks to evolve autonomous driving strategies. The project achieved its objectives of developing and optimizing neural networks for improved driving performance, with effective integration and performance under various conditions. While challenges such as performance bottlenecks and training limitations were encountered, the project provided valuable insights and a solid foundation for future enhancements and real-world applications. The recommendations for optimization, advanced techniques, and user experience improvements offer a pathway to further refine and expand the system's capabilities.

12.References:

Medium https://medium.com/ Stackoverflow https://stackoverflow.com/ ChatGPT AI https://chat.openai.com/

Github link of the project: https://github.com/ibrahimsudas/Genetic-Algorithm-Self-Driving-Car.git

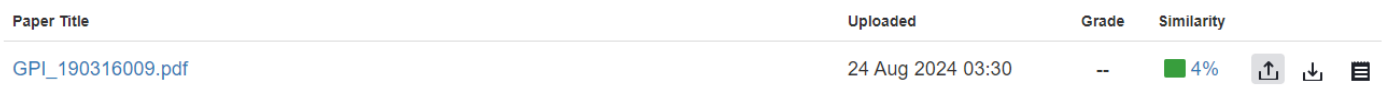
13.Interdisciplinary Domain of Our Study

Automative

14.Sustainability Development Goal of Our Project

9.Industry/ Innovation / Infrastructure

15.Similarity Report



Paper Title	Uploaded	Grade	Similarity
GPI_190316009.pdf	24 Aug 2024 03:30	--	🟩 4%