AUTONOMOUS DRIVING COMPARISON BETWEEN NEAT AND HYPERNEAT USING PYGAME

BY

Md. Faridul Islam
ID: 183-15-2304

AND

Md. Redoy Rajoan
ID: 183-15-2243

This Report Presented in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Computer Science and Engineering

Supervised By

Mohammad Monirul Islam
Senior Lecturer
Department of CSE
Daffodil International University

Co-Supervised By

Mohammad Jahangir Alam
Senior Lecturer
Department of CSE
Daffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY

DHAKA, BANGLADESH

SEPTEMBER 2022

APPROVAL

This Project titled “Autonomous Driving Comparison Between NEAT and HyperNEAT Using Pygame”, submitted by Md. Faridul Islam, ID: 183-15-2304 and Md. Redoy Rajoan, ID: 183-15-2243 to the Department of Computer Science and Engineering, Daffodil International University, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering and approved as to its style and contents. The presentation has been held on 13 September 2022.

DECLARATION

We hereby declare that, this project has been done by us under the supervision of **Mohammad Monirul Islam, Senior Lecturer, Department of CSE** Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere for award of any degree or diploma.

Supervised by:

[illegible]

Mohammad Monirul Islam
Senior Lecturer
Department of CSE
Daffodil International University

Co-Supervised by:

[illegible]

Mohammad Jahangir Alam
Lecturer
Department of CSE
Daffodil International University

Submitted by:

[illegible]

Md. Faridul Islam
ID: 183-15-2304
Department of CSE
Daffodil International University

[illegible]

Md. Redoy Rajoan
ID: 183-15-2243
Department of CSE
Daffodil International University

ACKNOWLEDGEMENT

First we express our heartiest thanks and gratefulness to almighty God for His divine blessing makes us possible to complete the final year project/internship successfully.

We really grateful and wish our profound our indebtedness to **Mohammad Monirul Islam**, **Senior Lecturer**, Department of CSE Daffodil International University, Dhaka. Deep Knowledge & keen interest of our supervisor in the field of "*Field name*" to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior draft and correcting them at all stage have made it possible to complete this project.

We would like to express our heartiest gratitude to Professor Dr. Touhid Bhuiyan, Head**,** Department of CSE, for his kind help to finish our project and also to other faculty member and the staff of CSE department of Daffodil International University.

We would like to thank our entire course mate in Daffodil International University, who took part in this discuss while completing the course work.

Finally, we must acknowledge with due respect the constant support and patients of our parents.

ABSTRACT

The use of autonomous driving is growing in popularity thanks to manufacturers like Tesla, Waymo, and numerous others. Regardless of autonomous driving, the application of the neuroevolution algorithm in neural networks today is without a doubt one of the most important and ground-breaking areas of research. A neural network's training, evolution, mutation, and crossover processes are both enjoyable and essential for the future. For the greatest results in this study, we created a neural network, trained it using the NEAT neuroevolution algorithm, performed mutation and crossover, and then managed the extinction of our genome. The NeuroEvolution of Augmenting Topologies (NEAT) and HyperNEAT brain evolution algorithms are both promising. We compare the performance of these two algorithms on a benchmark problem of a car driving simulator that includes strategic decision making. Our results demonstrate where HyperNEAT surpasses NEAT and where NEAT outperforms HyperNEAT. In this study, the neural network is primarily used to learn how to navigate a track on its own. To make it more challenging and learn how it performs on various tracks, we can test and train its performance on various tracks. That's when elitism and other mutational configurations come in help. Sometimes it works better to create a more random mutation, other times not so much. The same is true of elitism; it is beneficial to some extent but loses its significance beyond that. We discovered a good setup that allows our network to function at its optimum but varies slightly from track to track. There is always room for a program to perform additional tasks and to output intelligence in a real-time setting.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
Introduction

1.1 Introduction to NEAT and HyperNEAT

In recent years, there has been an increase in demand for the development of reliable and effective intelligent autonomous systems capable of making decisions similar to human behavior in a given task. Deep neural networks (DNNs), one of the AI family's most powerful members, have recently made significant progress in a variety of real-world situations due to their proficiency in difficult decision-making tasks. Convolutional neural networks (CNN), recursive neural networks (RvNN), recurrent neural networks (RNN), and deep generative networks are among the models associated with DNNs. CNNs are recognized as the most established method when compared to other DNN models because they delivered amazing results across a wide range of applications, particularly in the area of computer vision. [1]

While standard perception, path planning, and motion control approaches can address the majority of driving scenarios, the remaining unsolved driving scenarios are edge cases where traditional methods fall short. [2] CNNs are currently outperforming humans on common datasets, revolutionizing image and pattern recognition. [3] Driverless cars have progressed from "impossible" to "inevitable" in the last ten years. Since Google's self-driving car project began in 2009, the majority of automakers have implemented or are working to implement the technology. The technology's core concept is to use trained object identification algorithms to determine what actions a car should take. Deep learning, a branch of artificial intelligence, in particular, has proven to be effective in this area. [4]

We present NeuroEvolution of Augmenting Topologies (NEAT), a technique that outperforms the best fixed-topology approaches on a difficult benchmark reinforcement learning problem. We contend that the higher efficiency is due to the use of a methodical topological crossover strategy, the protection of structural innovation through speciation, and integral growth from minimal structure. A series of ablation investigations are used to test this claim, and the results show how important each element is to the system as a whole and to one another. As a result, learning moves much faster.

NEAT is also an important contribution to GAs because it shows how evolution can optimize and complicate solutions, allowing for the evolution of ever-more complex solutions over time and reinforcing the comparison to biological evolution. [5] There are a few NEAT additions. In rtNEAT, for example, a "lifetime" timer is placed on each member of the population, with the population being evaluated continuously. When the timer on a network expires, its current fitness score is checked to see if it ranks among the lowest in the population; if it does, the network is eliminated and replaced by a new network created from the offspring of two highly fit parents. The new network is assigned a timer and deployed into the population to participate in ongoing evaluations.

[6] Then, HyperNEAT is uniquely suited to the growth of massive structures. It was founded on the CPPN theory and remains an important field of study. [7] There's also odNEAT, a decentralized online version of NEAT designed for multi-robot systems. odNEAT is run onboard robots while tasks are being completed in order to continuously improve the parameters and topology of the artificial neural network-based controllers. Robots that use odNEAT have the ability to adapt to changing circumstances and learn new behaviors as they carry out their tasks.

The physical distribution of islands is used to implement the online evolutionary process. Each robot optimizes its own internal population of potential solutions (intra-island variation), and when two or more robots come together, they trade potential solutions (inter-island migration). [8]

The Hyper in the name comes from the word Hypercube. The entire name of this approach, which is long, is "Hypercube-based NeuroEvolution of Augmenting Topologies." Hypercube means the approach because a CPPN that describes a connection pattern is minimum of four-dimensional (for example from taking x1,y1,x2, and y2). We can see in some experiments, it can be more than four dimensional, i.e when the encoded connection pattern is 3 dimensions; for that particular case, the CPPN will be six dimensional (i.e. from inputs x1,y1,z1,x2,y2, and z2). These more than one dimensional spaces are commonly sampled in the range of a hypercube which begins at -1 and ends at 1 on each dimension.

Usually a connection weight is shown by each point on the hypercube. As a result, HyperNEAT is effectively designing a pattern into a hypercube. Later the same pattern is then interpreted as a neural network's connectivity pattern. [9] There are many categories of problems where you could really benefit: for example, problems with a huge input amount. Because of CPPNs are an indirect encoding in HyperNEAT, network size (for example, dimensionality) is not nearly the most important feature in HyperNEAT problem complexity (not similar to most other learning methods). This simply means that giving huge or even massive input list / arrays to the learning approach is not a problem (but it can cost more time for the Central Processing Unit to eventually run a giant network, depending on how gigantic it is).

Each and every position on the hypercube shows or presents a connection weight value. So, HyperNEAT is effectively designing a pattern on the inside of a hypercube. This pattern is then helps us understand the connectivity pattern of a neural network. [9] There are several types of problems where you could gain an upper hand: Problems involving a big number of inputs. All because CPPNs in HyperNEAT works in using indirect encoding, the volume of the network or its dimensionality is not the number one most important stat in HyperNEAT which results complexity (different from most other learning approaches). This shows that inputting the learning algorithm big or even gigantic input arrays is not a problem (although it may take longer for the Central Processing Unit to actually run through a giant network, depending how huge it actually is).

It is not impossible that the number of inputs needed for some problems is not so obvious. For example, the efficient resolution of a visual graph may be unknown in early stage. It can also be impossible to find out the suitable output resolution. In a multi section arm, for example, the suitable number of sections and joint muscles may not be found (each one requiring own output). Here, HyperNEAT can be called a fantastic candidate for such problems only cause its performance does not rely on anything virtually from the resolution given. You don't have to be worried about acquiring the resolution correctly the initial time, or even decreasing it as an outcome.

If you ever feel the need to repeat the test / simulation at a higher resolution later, anyone can rely that it will still work.
Additionally, after training phase ends, you may simply scale up the substrate's resolution and re-query the CPPN across. The solve might still be practical at a higher resolution even if it necessary wasn't trained or simulated on that resolution! You can train at a much higher resolution then before even if this is not that particular case and then bootstrap using the knowledge you got at a much lower resolution. The inputs values to a CPPN are sent into a function f and are represented as Cartesian coordinates in a space of n-dimension, where n determines the total number of inputs.
Value of f calculated at each coordinate tells whether a point in n-dimensional space is present, absent, or strongly expressed; as a result, the phenotype produced by f's assessment (which is thought of as the genotype) is a sequence (Stanley, 2007). ANNs and CPPNs differ in that CPPNs can work in a wide range of functions, whereas ANNs usually only perform sigmoid functions (Stanley et al, 2009). [10] The HyperNEAT method is an algorithm of evolution type that can help evolve huge-scale networks using generative indirect encoding. The NEAT and HyperNEAT are used to make neural networks in two steps that mix a number of functions into one function.
The transfer functions allow for the encoding of symmetry, imperfect symmetry, and variation in repetition. Compositional Pattern Producing Networks are the names given to the composed functions (CPPNs). In the second stage, the intended neurons are assigned spatial coordinates. The previously generated CPPN is then used to compute synaptic weights between all pairs of neurons (or a subset of pairs). Both neurons' coordinates are fed into the CPPN's inputs, and the CPPN then outputs their connection weight. If the weight's absolute value is less than a certain threshold, it is not expressed. The substrate is the connectivity pattern created by CPPN. The fundamental property of the HyperNEAT substrate is that it can be scaled to higher resolutions while retaining its core structure and function. [11]
We usually use HyperNEAT generative encoding method to generate artificial neural networks based on the NeuroEvolution of Augmented Topologies genetic algorithm. HyperNEAT creates a CPPN short for Compositional Pattern Producing Network, which is one category of artificial neural network. Not similar to traditional artificial neural networks that use sigmoid functions particularly, CPPNs can and do use a huge

range of methods. [12] The HyperNEAT algorithm is designed for a function or a set of functions that creates neural network weights. The neural network is usually built from neurons positioned in a rectangular mesh known as the substrate array. The substrate coordinates are used in the network as inputs for the function and the function outputs weight of the connection between two neurons.

In the original HyperNEAT, the function is known as CPPN (Compositional Pattern Production Network) and is interpreted as a group of neural nodes with a scalar product on inputs and a nonlinear method on outputs. The structure of the network is identical to a typical neural network (that is why they named it network instead of a function). The NEAT method is mainly used to create the CPPN for the HyperNEAT. NEAT is one of many types of evolutionary algorithm that evolves a network generation by generation from an inferior form to something that has many complexity and niching in it. [13]

1.2 Motivation

We have long been captivated with neural networks and have yearned to work with them. It's still rather new to us, and there are many misconceptions regarding its potential here. Anyone finding it difficult to avoid learning more about artificial intelligence and

driverless vehicles is not alone. Even while driverless vehicles may be a thing of the far better future for us, it's hard to ignore their use in many other sectors. May this paper encourage others to learn more about neural networks and aid in our efforts to educate people about what little we do know about them. Additionally, we have always be curious about how autonomous driving functions in both the actual world and the virtual world. We also wanted to see how far a machine could advance in terms of learning and becoming intelligent. But as we are all aware, neural networks have much more potential than meets the eye. In order to make learning about neural networks more interesting for individuals who are unfamiliar with it, this paper will examine a very small piece of it, explain it, play with it, and play with it.

1.3 Rationale of the Study

There have been numerous studies comparing various approaches to autonomous driving and various developments in artificial intelligence. On the other hand, we attempted to go deeply into one particular neuroevolution algorithm that is regarded as one of the strongest and best. Basically, we tried to think of every way to modify N.E.A.T.

1.4 Research Questions

• How well do various NEAT configurations perform?

• What are the most popular neuroevolution algorithms?

• What are those algorithms' limitations?

1.5 Expected Output

In order to teach a car to drive intelligently and ultimately create autonomous driving, we used the NEAT neuroevolution method on a car in a track in a 2D environment. We created a track with a white exterior and a black interior that controls collision. PyGame was utilized to create the visualization. The application of neural networks and intelligence for those autos is then done using the NEAT library. Every time a particular quantity of a species' genome is created, a simulation is conducted with the goal of predicting the emergence of intelligence. The NEAT setup was then tweaked, and we documented the changes in our simulation across hundreds of generations, compared the various configurations, and observed if the network was made better or worse, which was our desired outcome.

1.6 Report Layout

This paper is organized as follows, in chapter one (Introduction) we gave an elaborate overview of our project we gave an overview of the project as Introduction in 1.1 followed by Motivation in 1.2, Rationale of the Study in 1.3, Research Question in 1.4, Expected Output in 1.5 and lastly Report Layout in 1.6. Then we have chapter 2 (Background) where we discuss about the origin of our problem starting with Terminologies in 2.1, Related Works in 2.2, Comparative Analysis and Summary in

2.3, Scope of the Problem in 2.4, Challenges in 2.5. Then we have chapter 3 (Research Methodology) where we briefly talk how we prepared for this project beginning with Research Subject and Instrumentation in 3.1 and Data Collection Procedure in 3.2, Statistical Analysis in 3.3 and lastly Applied Mechanism in 3.4. Beginning in chapter 4 (Experimental Results and Discussion) where we've shown how we executed our project practically in Experimental Setup in 4.1, Experimental Results & Analysis in 4.2 and Discussion in 4.3. Chapter 5 (Impact on Society, Environment and Sustainability) is for discussing every type of impact of our project in a society beginning with Impact on Society in 5.1, Impact on Environment in 5.2, Ethical Aspects in 5.3 and Sustainability Plan in 5.4. On the last chapter, chapter 6 (Summary, Conclusion, Recommendation and Implication for Future Research) we summarized our project and concluded it in Summary of the Study in 6.1, Conclusions in 6.2 and lastly Implication for Further Study in 6.3 followed by References.

CHAPTER 2

Background

2.1 Terminologies

Our project was to develop an autonomous driving project based on neural networks. We programmed in Python 3.10.0 with IDLE as our IDE. We used NEAT 0.92 for the autonomous algorithm. To visualize our game, we used the pygame 2.0 library. PIP was used to install all of these plugins. Following that, we used a game to simulate autonomous driving. As a genetic algorithm, NEAT learns from its mistakes and improves with each new generation. This works similarly to the survival of the fittest in nature.

2.2 Related Works

Table 2.1 Literature Review

Author(year)	Title	Description
Wilson, Akin. (2020)	Autonomous Driving: deep machine learning	This study demonstrates how deep learning can be used to train autonomous vehicles. Despite this, it appears unlikely that a single end-to-end convolutional neural network could provide a vehicle with complete autonomy.

Agnihotri, Akhil & Saraf, Prathamesh & Bapnad, Kriti. (2019)	A Convolutional Neural Network Approach Towards Self-Driving Cars	The best thing about CNNs is that they automatically select obvious features for the model by learning features from training cases rather than depending on humans to do it. CNNs are more accurate than flattened neural networks because they use the 2D structure of images. Another advantage of CNNs. It outperforms explicit feature decomposition techniques such as lane detection and nearby car recognition because the network selects which characteristics from the image are the best to extract.

Grigorescu, Sorin & Trasnea, Bogdan & Cocias, Tiberiu & Macesanu, Gigel. (2019).	A survey of deep learning techniques for autonomous driving	Advances in deep learning and artificial intelligence have hastened the development of self-driving car technologies over the last decade (AI). The goal of this research is to examine the most recent advancements in automated driving deep learning technologies. The first topics covered include deep reinforcement learning, convolutional and recurrent neural networks, and self-driving AI architectures. The driving scene perception, path planning, behavior arbitration, and motion control algorithms that were explored are on top of these technologies. Both the modular perception-planning-action pipeline, in which each module is created using deep learning techniques, and End2End systems, which instantaneously translate sensory data to steering commands, are investigated.

Jalali, Seyed Mohammad & Kebria, Parham & Khosravi, Abbas & Saleh, Khaled & Nahavandi, Darius & Nahavandi, Saeid. (2019).	Optimal Autonomous Driving Through Deep Imitation Learning and Neuroevolution	The necessity for developing dependable and efficient intelligent autonomous systems that can make decisions similar to human behavior in a certain activity has grown in recent years. Because the bulk of autonomous agent applications are dynamically complex and require several variables to construct policies, the problem persists. Using the imitation learning paradigm, a robot, agent, or autonomous system can replicate human behavior by observing how a task is accomplished during a supervisor's presentation. This fascinating subject is one of the subfields of artificial intelligence and will most likely play a big part in the development of intelligent automation systems in the future.

Kenneth O. Stanley,Risto Miikkulainen.(2002)	Efficient Evolution of Neural Network Topologies	Neuroevolution (NE), the artificial evolution of neural networks using genetic algorithms, has shown a lot of promise in reinforcement learning tasks. NE outperforms classic reinforcement learning strategies in several benchmark tasks. Neural networks are a good class of decision-making systems to evolve because they may represent responses to a wide range of issues and because the mapping from genotype to phenotype is often efficient. NE is particularly well suited to jobs involving reinforcement learning because to the lack of supervision required.

Buk, Zdenek & Koutník, Jan & Snorek, Miroslav. (2009).	NEAT in HyperNEAT substituted with genetic programming	They demonstrate the application of genetic programming (GP) in the indirect encoding of neural network weight evolution in this study. We compare the original HyperNEAT method to our own implementation, in which genetic programming was utilized instead of the NEAT on which it was based. The algorithm was called HyperGP. In simulation, the created neural networks were used to control mobile, independent entities (robots). The agents were trained to drive as swiftly as possible on average. As a result, they must learn how to drive a vehicle safely on public highways.

Haasdijk, Evert & Rusu, Andrei & Eiben, A. (2010).	HyperNEAT for Locomotion Control in Modular Robots.	Individual robots cannot predict in advance where they will end up in an organism in an application where autonomous robots can spontaneously combine into arbitrary organisms. A single genome that is sufficient for each individual robot can only be developed by an evolutionary algorithm that evolves the controllers if the organism is to be controlled autonomously by its component robots. However, the robots should behave differently depending on where they are in an organism, which means that their phenotype should vary. Using the HyperNEAT generative encoding technique with differential genome expression, they show how to solve this issue in this study.

Drchal, Jan & Koutník, Jan & Snorek, Miroslav. (2009).	HyperNEAT Controlled Robots Learn How to Drive on Roads in Simulated Environment.	Individual robots in an application where autonomous robots can spontaneously merge into arbitrary organisms cannot forecast where they will end up in an organism. If the organism is to be managed autonomously by its component robots, a single genome that is sufficient for each individual robot can only be generated through an evolutionary algorithm that evolves the controllers. However, the robots should behave differently based on their location within an organism, implying that their phenotype will vary. In this paper, they describe how to overcome this problem using the HyperNEAT generative encoding technique with differential genome expression.

2.3 Comparative Analysis

Table 2.2 Comparative Analysis of Evolutionary Algorithm

Method	Encoding	Evolutionary Algorithm	Aspects evolved
GNARL was made by Angeline et al.	Direct	Evolutionary programming	Parameters along with Structure (simultaneous, complexification)
EPNet was proposed by Yao and Liu,	Direct	Evolutionary in nature programming that is (mixed with simulated annealing and back propagation)	Structure and parameters (complexification, Combined and simplification)
NeuroEvolution of Augmenting Topologies (NEAT) from Miikkulainen and Stanley	Direct	The evolutionary algorithm Tracks genes with historical markers to allow for topological crossover and to protect innovation via speciation.	parameters along with Structure

Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) was made by D'Ambrosio, Gauci, Stanley	Spatial patterns formed by a compositional pattern-producing network (CPPN) inside a hypercube are interpreted as indirect and non-embryogenic connection patterns in a lower-dimensional space.	Algorithm based on genes. The NEAT algorithm is used to evolve the CPPN (before).	Fixed structure, Parameters (fully connected functionally)

Evolvable Substrate Hypercube-based NeuroEvolution of Augmenting Topologies (ES-HyperNEAT) was created Risi, Stanley	A hypercube's compositional pattern-producing network (CPPN) generates spatial patterns that are perceived as connection patterns in a lower-dimensional space in an indirect, non-embryogenic manner.	To evolve the CPPN. Genetic algorithm, the NEAT algorithm (described above) is used.	Network structure and Parameters
Evolutionary Acquisition of Neural Topologies (EANT/EANT2) was proposed by many people who are Sommer Siebel, Sommer and Kassahun	Both direct and indirect, maybe embryogenic (Common Genetic Encoding).	Evolution strategies , Evolutionary programming	Parameters and structure (complexification ,separately)

Interactively Constrained Neuro-Evolution (ICONE) was proposed by Rempis	Constraint masks are used to restrict the search to specific topologies and parameter manifolds.	Algorithmic evolution Using domain knowledge, constraint masks are used to dramatically decrease the search space.	Parameters and Structure (complexification separately, , interactive)
Deus Ex Neural Network (DXNN) was made by Gene Sher	Algorithm that is subject to change Constraint masks are used to dramatically reduce the search space by utilizing domain information.	formula for mathematics changes the structure and behavior of networks throughout a time span	Parameters and Structure (interactive , separately complexification,)

Nine different evolutionary algorithms are compared on "Table 2.2" in terms of how they encode data, the kind of evolutionary algorithm they are, and how far they have come.

We can see that GNARL, created in 1994 by Angeline et al., utilized both a direct encoding technique and an evolutionary algorithm. This has guidelines and a framework (simultaneous, complexification).

Yao and Liu's EPNet, published in 1997, also employs the direct encoding technique. It uses programming that is based on evolution (combined with back propagation and simulated annealing). The evolution of this algorithm is composed of parameters and structure (mixed, complexification and simplification).

Neuro Evolution of Augmenting Topologies, or NEAT, was developed by Stanley and Miikkulainen in 2002. Additionally, NEAT benefits from direct encoding. The structure and parameters of NEAT are another aspect that changes.

2008 saw the development of Hypercube-based Neuro Evolution of Augmenting Topologies by Stanley, D'Ambrosio, and Gauci (Hyper NEAT). This algorithm uses a compositional pattern-producing network (CPPN) inside a hypercube to create spatial patterns that represent connection patterns in a lower-dimensional space. This method of encoding is indirect and non-embryogenic. Structure and parameters are additional evolving aspects.

Evolvable Substrate Hypercube-based Neuro Evolution of Augmenting Topologies, published by Risi and Stanley in 2012 (ES-Hyper NEAT). This algorithm uses indirect, non-embryogenic encoding, where spatial patterns are produced in a hypercube by a compositional pattern-producing network (CPPN), which are then converted into connectivity patterns in a lower-dimensional space. genetic programming. The NEAT algorithm is used to evolve the CPPN. The network structure and parameters are the element that have changed.

EANT/EANT2 (Evolutionary Acquisition of Neural Topologies) by Kassahun and Sommer, 2005 / Siebel and Sommer, 2007. It was both direct and indirect, and possibly embryogenic (Common Genetic Encoding). Structure and parameters (separately, the aspect of evolution's complexity. Rempis's Interactively Constrained Neuro-Evolution (ICONE) was published in 2012. The method of encoding was direct, which includes constraint masks to limit the search to specific topology / parameter manifolds. This is an evolving algorithm. Constraint masks are used to dramatically reduce the search space using domain knowledge. Structure and parameters (separately, complexification, interactive) were the evolution aspects of this algorithm.

Deus Ex Neural Network (DXNN), written by Gene Sher, was released in 2012. The direct/indirect strategy includes constraints, local adjustment, and evolutionary integration of new sensors and actuators. Over a variety of timescales, memetic algorithms alter the structure and properties of networks. The division, complexity, and interaction of structure and parameters was a feature of evolution.

Table 2.3 Comparative Analysis of NEAT and Hyper-NEAT Based on Fitness

Generation	**Species (**Using NEAT**)**	**Max Fitness (**Using NEAT**)**	**Species (**Using Hyper-NEAT**)**	**Max Fitness (**Using Hyper-NEAT**)**
1	1	162779.33	1	388.53
10	1	201551.26	5	17658.48
20	1	201551.26	5	169017.20
30	1	201683.80	4	190033.06
40	2	201683.80	4	190033.06
50	2	201683.80	4	190033.06
60	3	201683.80	2	190033.06
70	4	201683.80	4	190033.06
80	4	201683.80	4	190033.06
90	3	201683.80	3	198062.80
100	5	201683.80	3	201150.46
110	6	201683.80	3	201150.46
120	4	201683.80	2	201150.46
130	3	201683.80	2	201150.46
140	3	201683.80	2	201150.46
150	2	201683.80	2	201150.46
160	3	201683.80	2	201150.46
170	3	201683.80	2	201150.46
180	2	201683.80	2	201150.46
190	3	201683.80	2	222133.33
200	4	201683.80	2	222133.33
210	3	201683.80	2	222133.33
220	2	201683.80	2	222133.33
230	2	201683.80	3	222133.33
240	2	201683.80	3	222133.33
250	2	201683.80	3	222133.33
260	2	201683.80	4	222133.33
270	2	201683.80	4	222416.20

280	**2**	**201683.80**	**4**	**222416.20**
290	3	201683.80	2	222416.20
300	3	201683.80	2	222416.20
310	2	201683.80	2	222416.20
320	2	201683.80	2	222416.20
330	2	201683.80	2	222416.20
340	2	201683.80	2	222416.20
350	2	201683.80	2	222416.20
360	2	201749.73	2	222416.20
370	2	201749.73	2	222416.20
380	2	201749.73	3	222416.20
390	2	201749.73	3	222416.20
400	2	201749.73	2	222416.20
410	2	201749.73	2	222416.20
420	2	201749.73	2	222416.20
430	2	201749.73	2	222416.20
440	3	201749.73	2	222416.20
450	3	201749.73	2	222416.20
460	2	201749.73	2	222416.20
470	2	201749.73	2	222416.20
480	2	201749.73	2	222416.20
490	2	201749.73	2	222416.20
500	2	201749.73	2	222416.20

Here, we can observe how the fitness function value on NEAT and Hyper-NEAT has grown over time. As can be seen from the 500-generation study, in terms of fitness function, Hyper-NEAT outperformed NEAT.

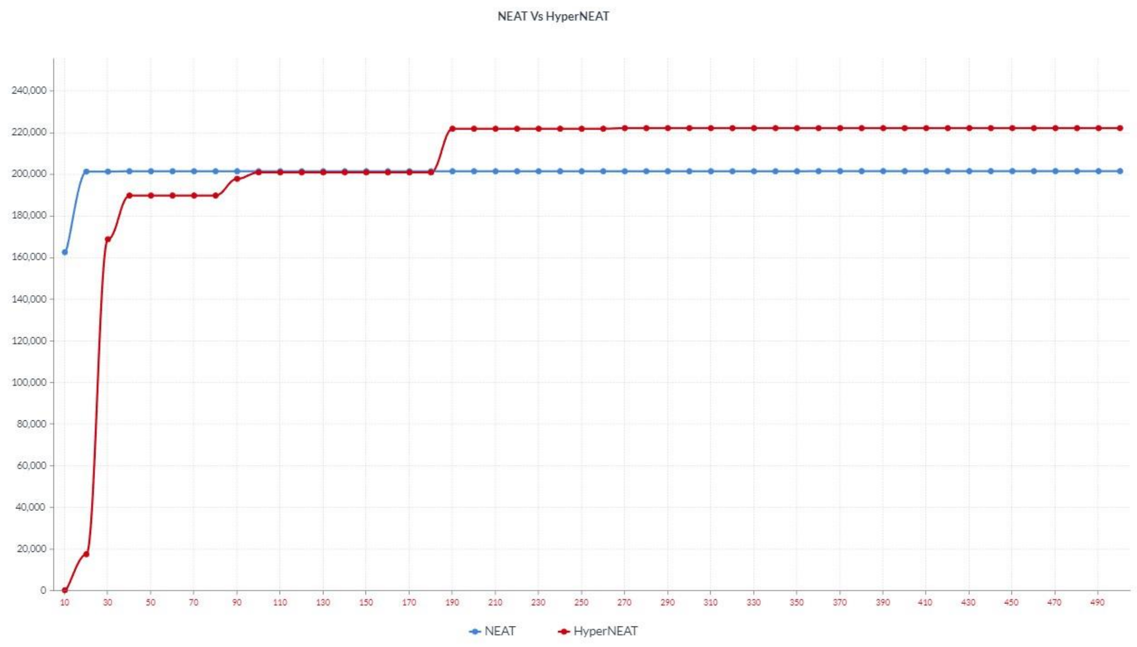
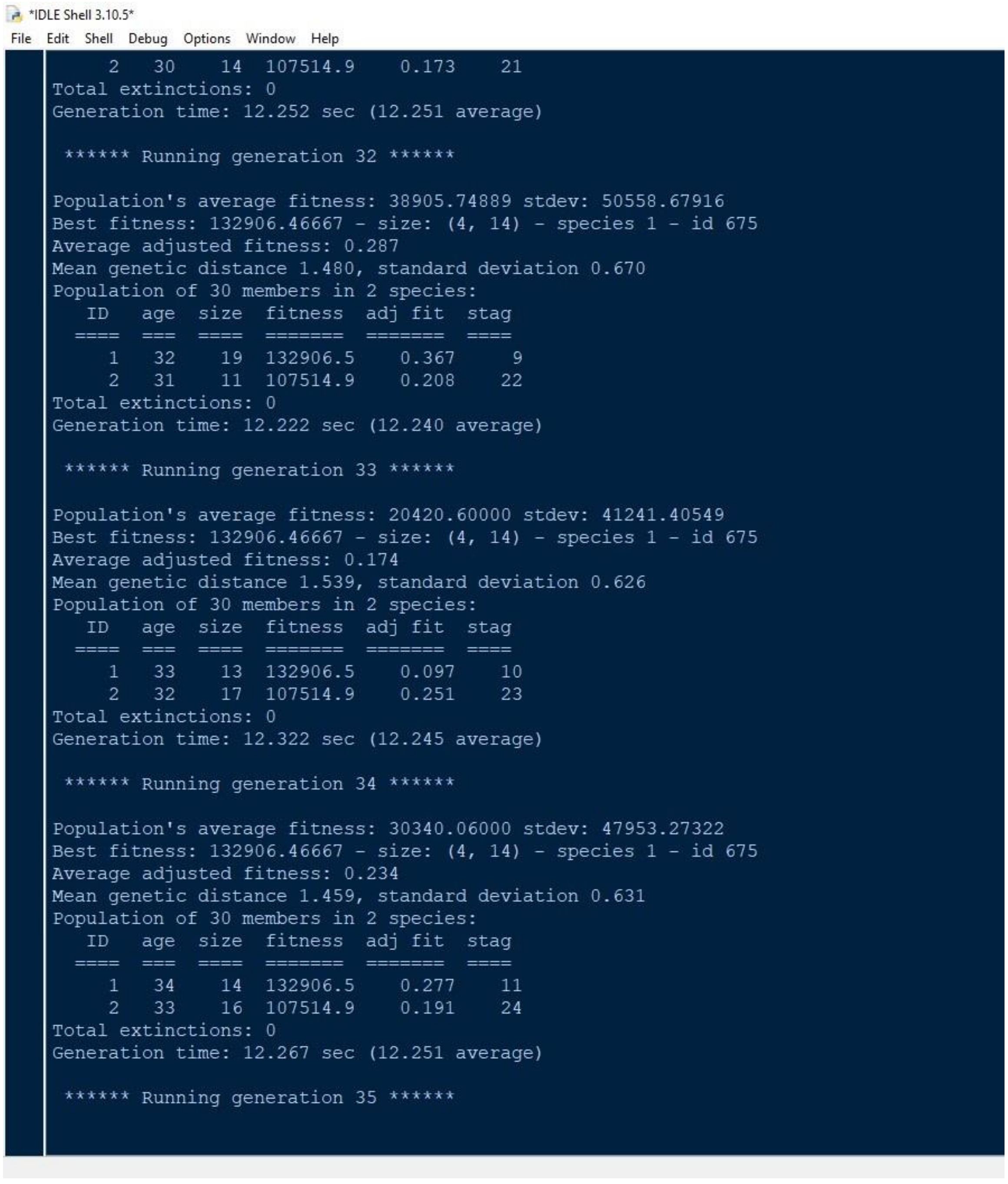


Fig. 2.1 Graph Chat of NEAT vs Hyper-Neat based on best fitness value.

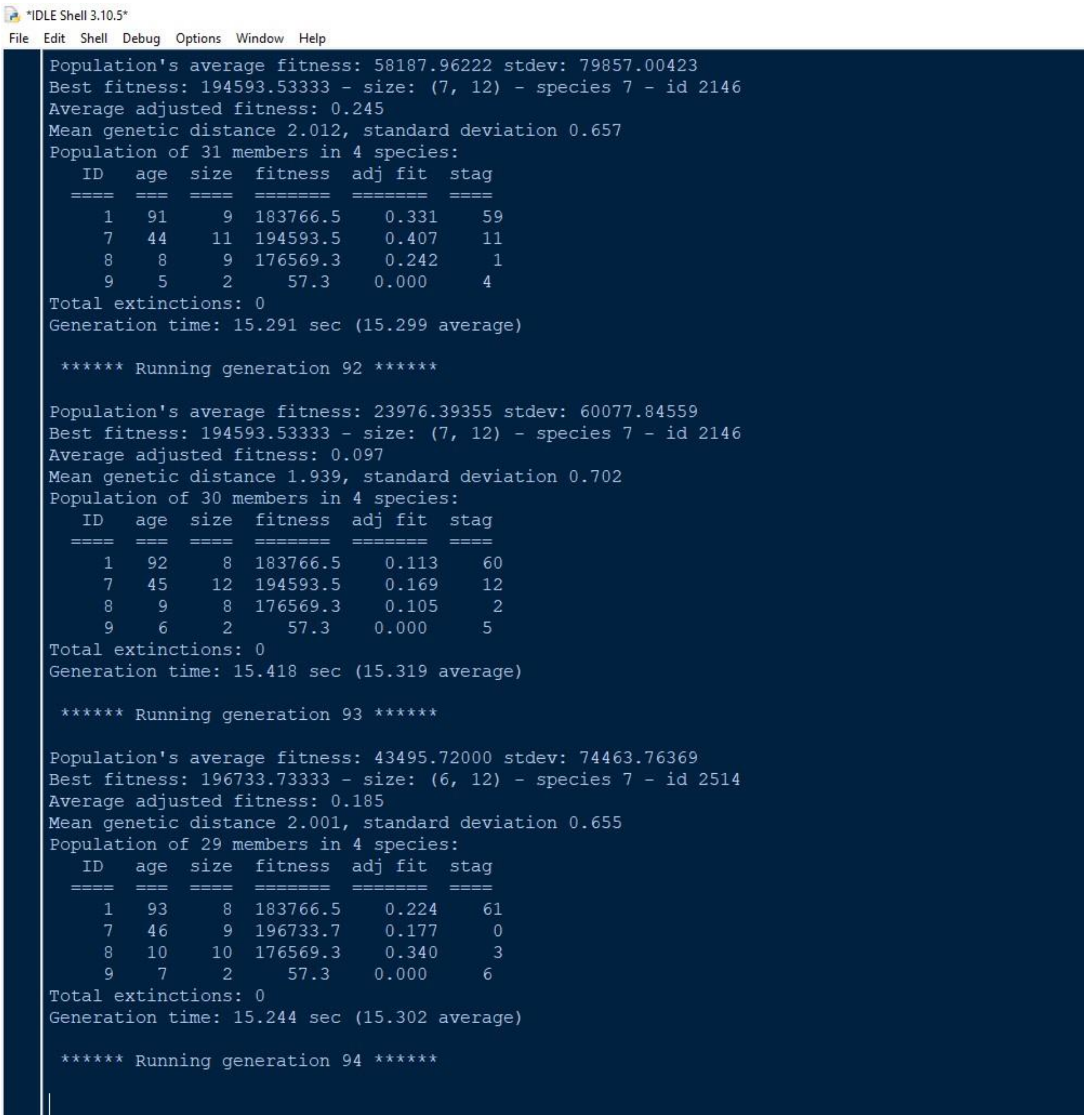
This graph also demonstrates how in terms of fitness function, Hyper-NEAT outperformed NEAT.



```
      2   30    14  107514.9    0.173    21
Total extinctions: 0
Generation time: 12.252 sec (12.251 average)

 ****** Running generation 32 ******

Population's average fitness: 38905.74889 stdev: 50558.67916
Best fitness: 132906.46667 - size: (4, 14) - species 1 - id 675
Average adjusted fitness: 0.287
Mean genetic distance 1.480, standard deviation 0.670
Population of 30 members in 2 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1   32    19  132906.5    0.367     9
     2   31    11  107514.9    0.208    22
Total extinctions: 0
Generation time: 12.222 sec (12.240 average)

 ****** Running generation 33 ******

Population's average fitness: 20420.60000 stdev: 41241.40549
Best fitness: 132906.46667 - size: (4, 14) - species 1 - id 675
Average adjusted fitness: 0.174
Mean genetic distance 1.539, standard deviation 0.626
Population of 30 members in 2 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1   33    13  132906.5    0.097    10
     2   32    17  107514.9    0.251    23
Total extinctions: 0
Generation time: 12.322 sec (12.245 average)

 ****** Running generation 34 ******

Population's average fitness: 30340.06000 stdev: 47953.27322
Best fitness: 132906.46667 - size: (4, 14) - species 1 - id 675
Average adjusted fitness: 0.234
Mean genetic distance 1.459, standard deviation 0.631
Population of 30 members in 2 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1   34    14  132906.5    0.277    11
     2   33    16  107514.9    0.191    24
Total extinctions: 0
Generation time: 12.267 sec (12.251 average)

 ****** Running generation 35 ******
```

Fig. 2.2 NEAT SHELL log during 500-generation simulation.

In this figure, we can see the number of species, their age, size, fitness, adjacent fitness, stagnation, running generation, average fitness, generation time, total extinctions, best fitness and standard deviation of each generation while using NEAT.



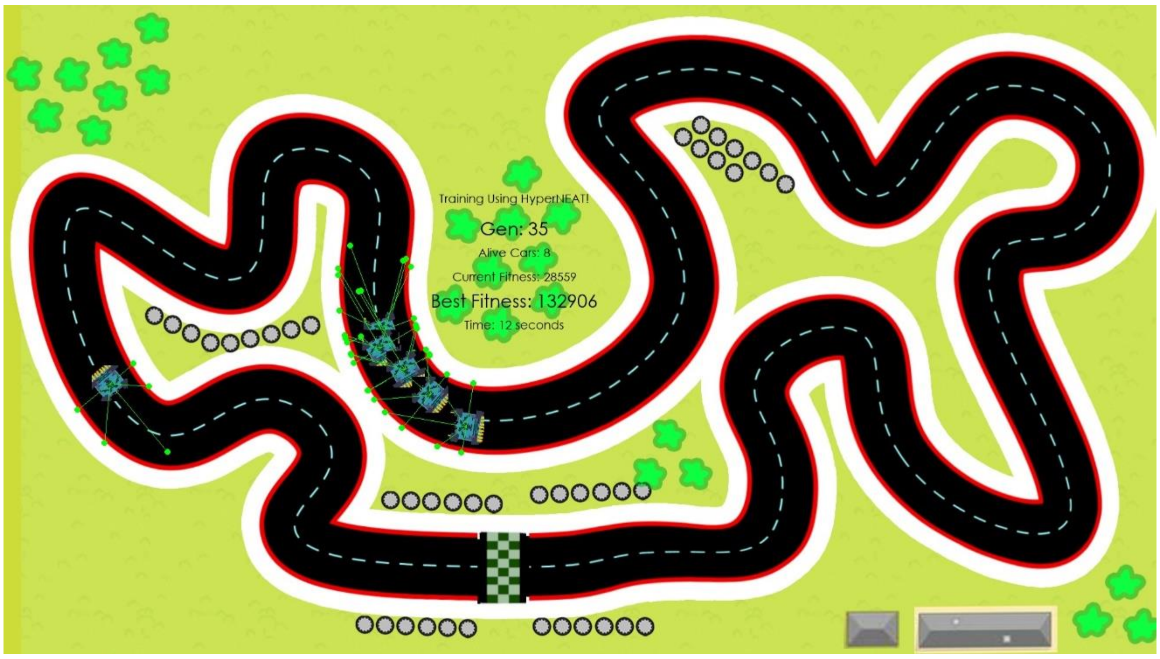
```
Population's average fitness: 58187.96222 stdev: 79857.00423
Best fitness: 194593.53333 - size: (7, 12) - species 7 - id 2146
Average adjusted fitness: 0.245
Mean genetic distance 2.012, standard deviation 0.657
Population of 31 members in 4 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1   91     9  183766.5    0.331    59
     7   44    11  194593.5    0.407    11
     8    8     9  176569.3    0.242     1
     9    5     2      57.3    0.000     4
Total extinctions: 0
Generation time: 15.291 sec (15.299 average)

 ****** Running generation 92 ******

Population's average fitness: 23976.39355 stdev: 60077.84559
Best fitness: 194593.53333 - size: (7, 12) - species 7 - id 2146
Average adjusted fitness: 0.097
Mean genetic distance 1.939, standard deviation 0.702
Population of 30 members in 4 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1   92     8  183766.5    0.113    60
     7   45    12  194593.5    0.169    12
     8    9     8  176569.3    0.105     2
     9    6     2      57.3    0.000     5
Total extinctions: 0
Generation time: 15.418 sec (15.319 average)

 ****** Running generation 93 ******

Population's average fitness: 43495.72000 stdev: 74463.76369
Best fitness: 196733.73333 - size: (6, 12) - species 7 - id 2514
Average adjusted fitness: 0.185
Mean genetic distance 2.001, standard deviation 0.655
Population of 29 members in 4 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1   93     8  183766.5    0.224    61
     7   46     9  196733.7    0.177     0
     8   10    10  176569.3    0.340     3
     9    7     2      57.3    0.000     6
Total extinctions: 0
Generation time: 15.244 sec (15.302 average)

 ****** Running generation 94 ******
```

Fig. 2.3 HyperNEAT SHELL log during 500-generation simulation

In this figure, we can see the number of species, their age, size, fitness, adjacent fitness, stagnation, running generation, average fitness, generation time, total extinctions, best fitness and standard deviation of each generation while using HyperNEAT.

2.4 Scope of the Problem

There have been few comparisons of NEAT and HyperNEAT in the same depth as this paper, and no other paper compares these two genetic algorithms with such a graphic interface. NEAT and HyperNEAT have been subjected to extensive analytical and mathematical scrutiny. We compared these two algorithms not just with a track and automobiles from hundreds of generations, but we also changed a few configurations to see how they performed. We can also examine how these run and perform in real time, with results and scores updated in real time. We also built a player mode in which the player is given the same amount of time as the algorithms and can practice and try to score as many points as possible like the simulations, as well as see the current and best scores in real time. This is possible with a simple keyboard (WASD) input. In this paper, we not only created a graphical interface that displays a variety of relevant information, but we also made this project simple to alter and play with. With the work we've done, the possibilities for analysis of this project are virtually limitless. Even graphics components are simple to create, maintain, and alter.



```
Training Using HyperNEAT!
Gen: 35
Alive Cars: 8
Current Fitness: 28559
Best Fitness: 132906
Time: 12 seconds
```

Fig. 2.4 Simulating using HyperNEAT

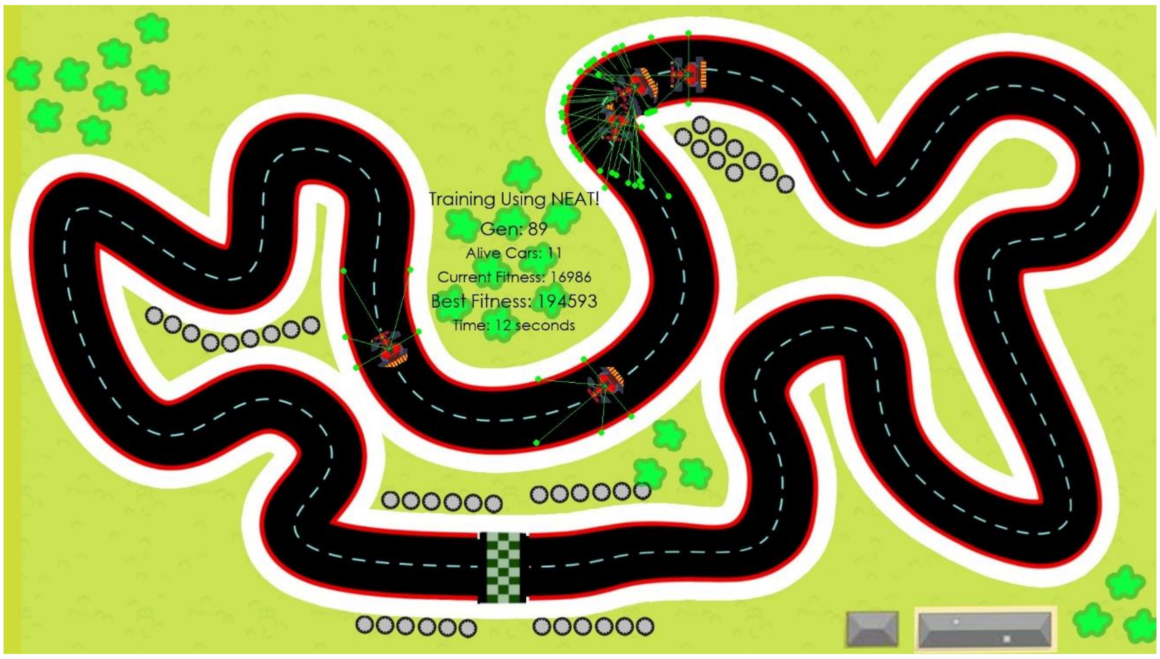


Fig. 2.5 Simulating using NEAT

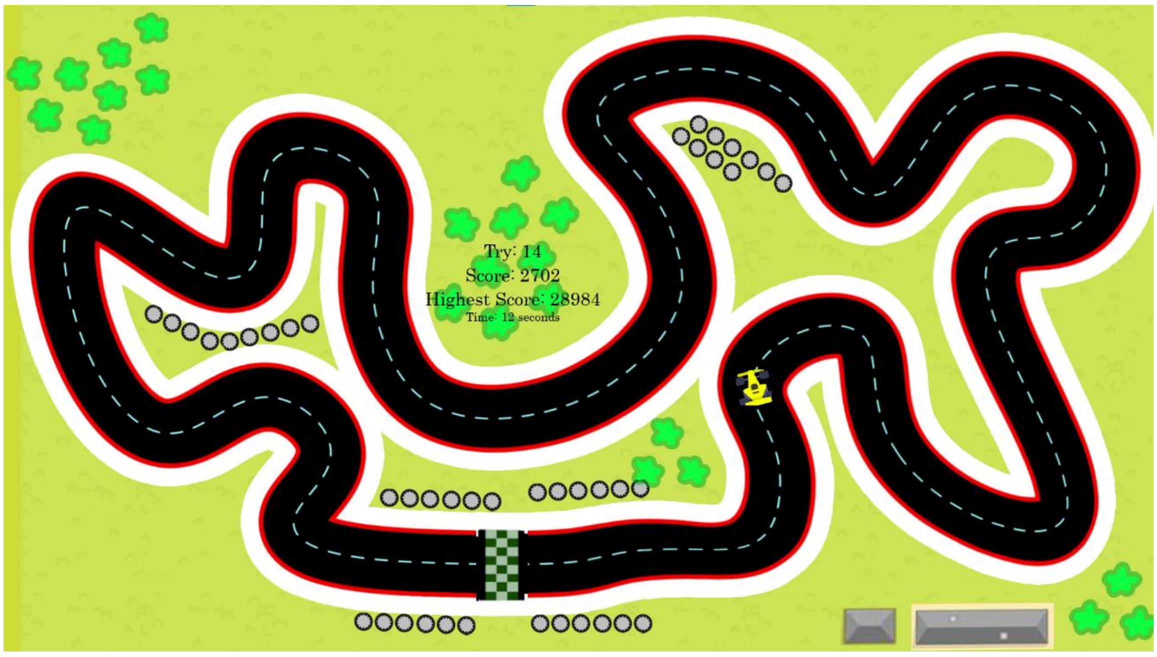
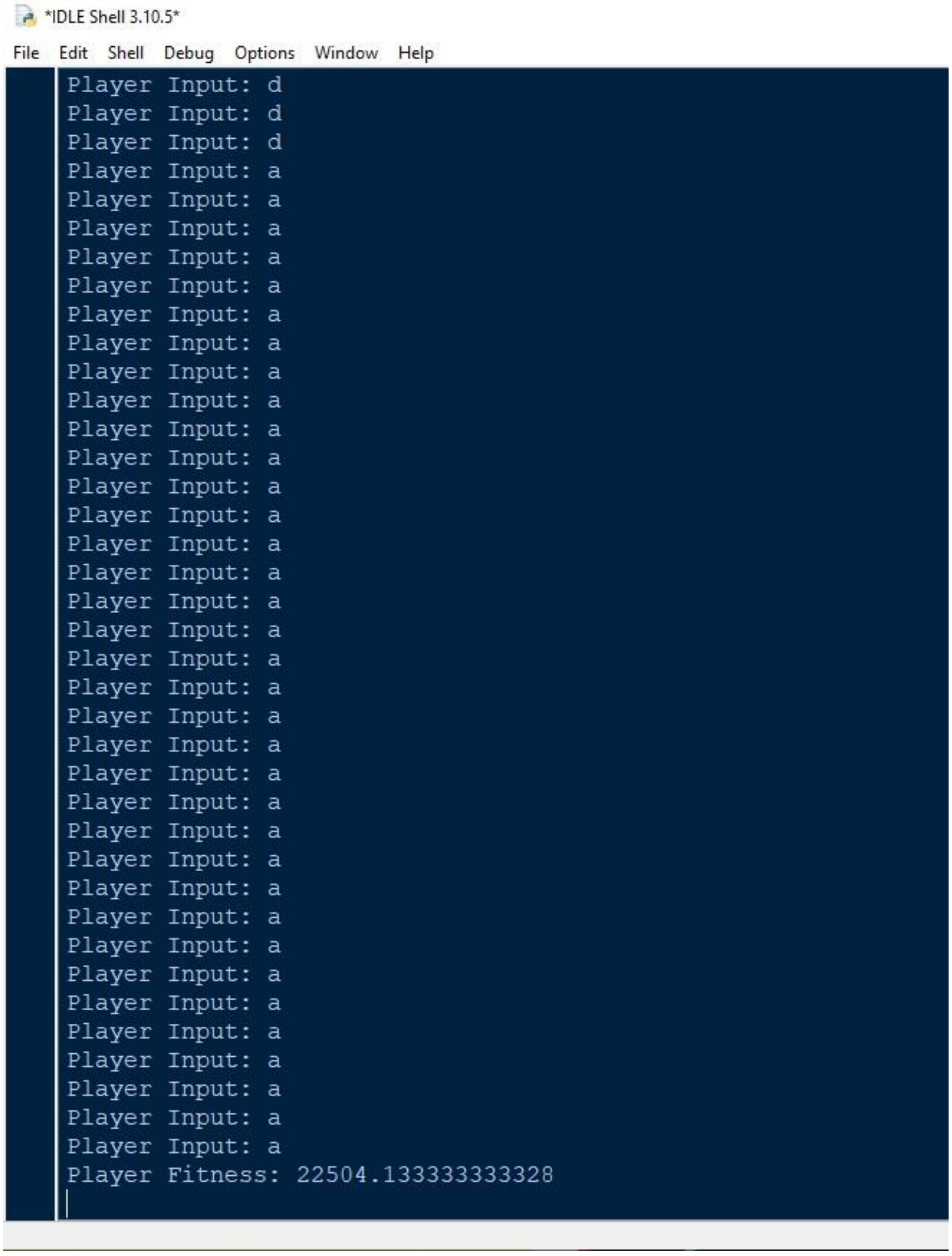


Fig. 2.6 Player Mode



```
Player Input: d
Player Input: d
Player Input: d
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Input: a
Player Fitness: 22504.133333333328
```

Fig. 2.7 Player Mode terminal log.

The visual interface of our project can be seen here, where you can view the generation, vehicles alive, current fitness, best fitness, and time given for each run. On the player mode, you can see how many times the player tried, his or her score, the highest score, and the amount of time allotted to score. It should be noted that the methods for calculating the score for the player and the fitness for the simulation are completely identical.

2.5 Challenges

Since the goal of our project was to simulate a self-driving AI car, we had to determine the optimal strategy and algorithms and put them into practice without any prior knowledge. With simpler tracks, we are getting good results, but as the complexity of the track rose, we struggled to get the best results. Because of this, we attempted to put certain ideal algorithms in place. We find NEAT to be simple to implement, and many projects have used it.

CHAPTER 3
Research Methodology

3.1 Research Subject and Instrumentation

Our research focuses on teaching a neural network the autonomous driving intelligence. The demand for AI in the modern world is endless. Additionally, conducting research in this area will encourage and inspire newcomers to continue working on the advancement of autonomous and AI systems. The application of this research is essential for both physical projects and any virtual projects.

3.2 Data collection procedure

With 5 input nodes, 4 output nodes, and a variety of hidden nodes, we may apply the NEAT (NeuroEvolution of Augmenting Topologies) method and determine whether or not and how much it improves. The NEAT algorithm's configuration will then be adjusted to work with a range of test tracks. For better comparison between our implementation and the total NEAT algorithm in autonomous driving, we will receive several mutations during the mutation process depending on the setup.

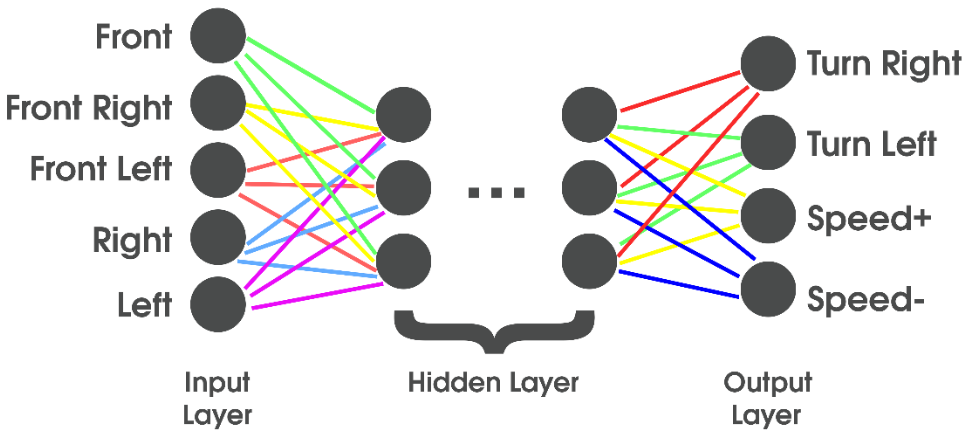


Fig. 3.1 The Input, Output and Hidden layers of our neural network with respected names of the inputs and the outputs.

3.3 Statistical Analysis

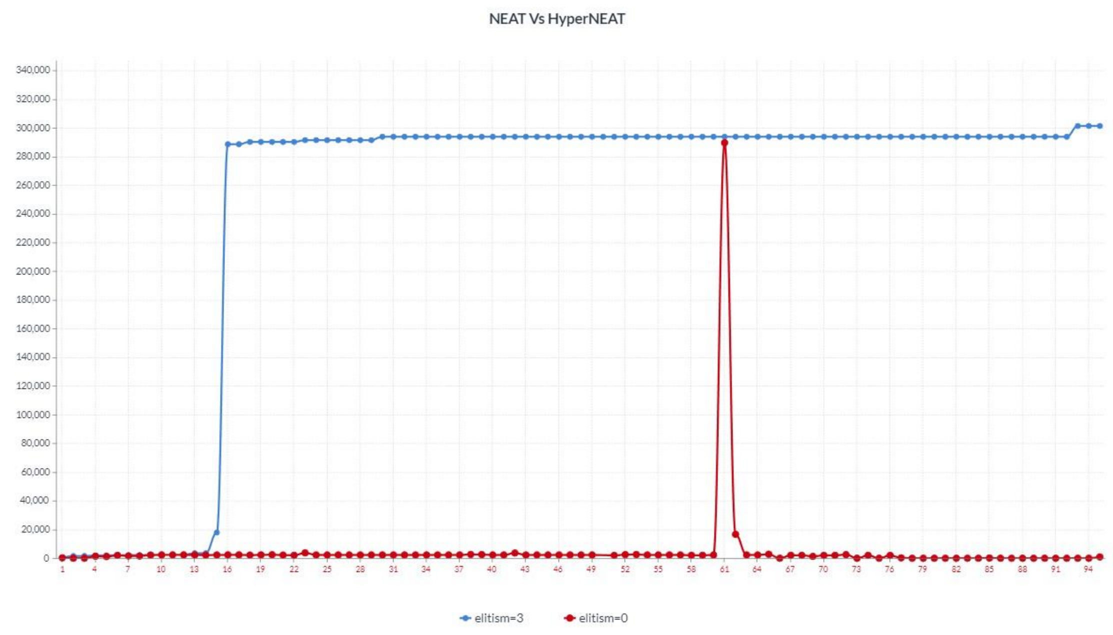


Fig 3.2 Comparison of our neural net based on Elitism between 0 and 3.

The built-in Default Reproduction class's parameters are specified in the Default Reproduction section. When establishing the Config instance, you must designate this class as the reproduction implementation; otherwise, you must include whatever configuration (if any) is necessary for your specific implementation. Elitism is the proportion of each species' fittest individuals who will be carried over from one generation to the next. 0 is the default value.

Table: 3.1 Showing number of species and max fitness over 100 generations between 0.1 and 0.9 weight mutate rate

Generation	Species (0.1 wmr)	Max Fitness (0.1 wmr)	Species (0.9 wmr)	Max Fitness (0.9 wmr)
10	1	2791.1	2	289039.3
20	1	53490.7	6	94163.3
30	2	290474.1	7	290076.1
40	2	291903.7	13	289358.7
50	2	291903.7	23	291348.3
60	2	291903.7	28	1588.5
70	3	291903.7	43	305.2
80	3	300007.5	47	305.2
90	2	300007.5	54	305.2
100	2	301980.7	60	305.2

.

Here, we can see how the neural network and the course of evolution are completely different due to the difference in weight mutation rates between 0.1 and 0.9. Here are the top 100 generation results after running the simulation for hundreds of iterations. WMR, or weight mutate rate, refers to the probability that a mutation will modify a link's weight by appending a random number. We compared the weight modify rate value, which modifies the weight of a connection by adding a random value, between 10% and 90% probability.

3.4 Applied Mechanism

We set out to train a neural network to drive autonomously without crashing, and with NEAT's assistance, we succeeded. By comprehending those, we additionally adjusted its configuration, noting the difference it makes over several generations of test runs.

Like any other neural network, the NEAT algorithm requires input, output, and a few hidden nodes. Then, using our car's radar value and the forward propagation procedure, we transmit a value to obtain one of the four outputs. Each node has a value that ranges from 1 to 0. The car's distance and likelihood of collision are determined by input nodes (here, color white). 1 denotes proximity, whereas 0 denotes a secure distance. After that, this value is transferred, and an output is produced. If you get the wrong output, you'll crash, but if you get the appropriate output, you'll survive.

Then we keep the best ones and mutate them, changing slightly their weight value and node value with a probability, which is also what we do for replacing the value, based on the output and depending on the fitness value established by the fitness calculation method of NEAT. We regard connection nodes and weight responses to be from the same species if there is a specific difference between their weight values, and we consider them to be from distinct species if there is a difference more than that. We cross across 2 species for a newer genome if any genome exhibits improvement after a particular generation, and as always, the genome with the best fitness remains.

Additionally, there are elitism and a certain threshold that keep the network from going extinct and are crucial for giving some genomes a chance to develop as some genomes do not evolve or anything else that really helps to eliminate the worst ways to move forward and forces the neural network to choose a different path in order to try to achieve a better fitness genome.

CHAPTER 4

Experimental Results and Discussion

4.1 Experimental setup

For this project we used Python 3.10.0 for programing using IDLE as our IDE. For autonomous algorithm we used NEAT 0.92. We used pygame 2.0 library to visualize our game. All these plugins were installed using PIP. We used Photoshop to make the graphical assest for the visualization of this project.

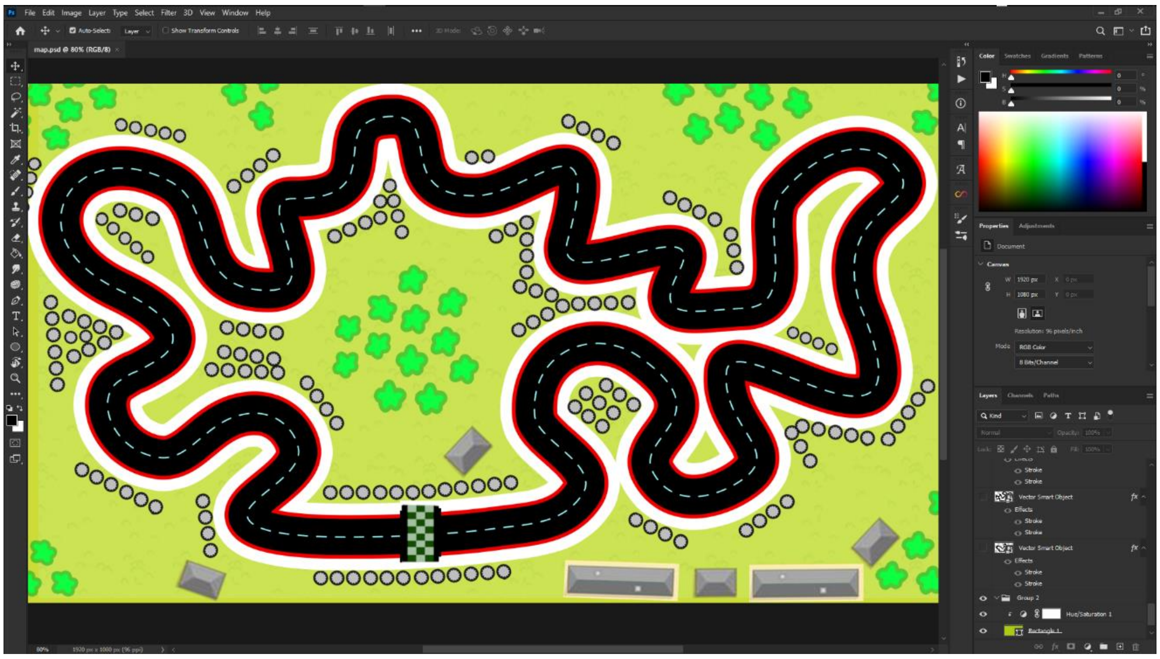


Fig 4.1 graphical asset, MapXii.png

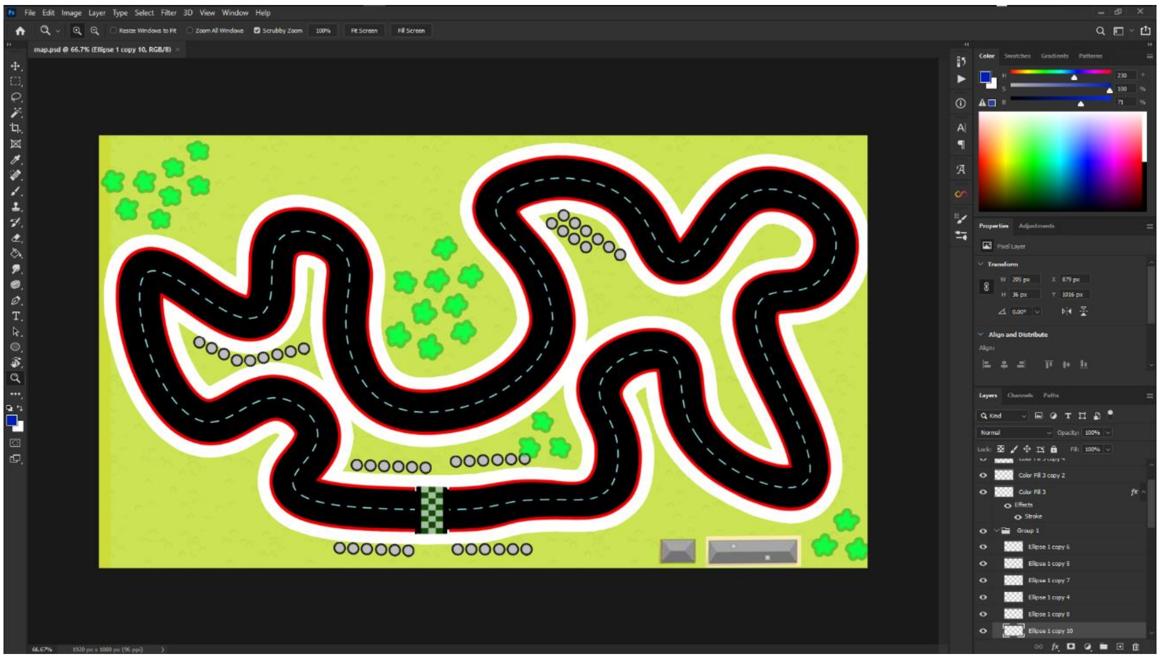


Fig 4.2 graphical asset, MapX.png

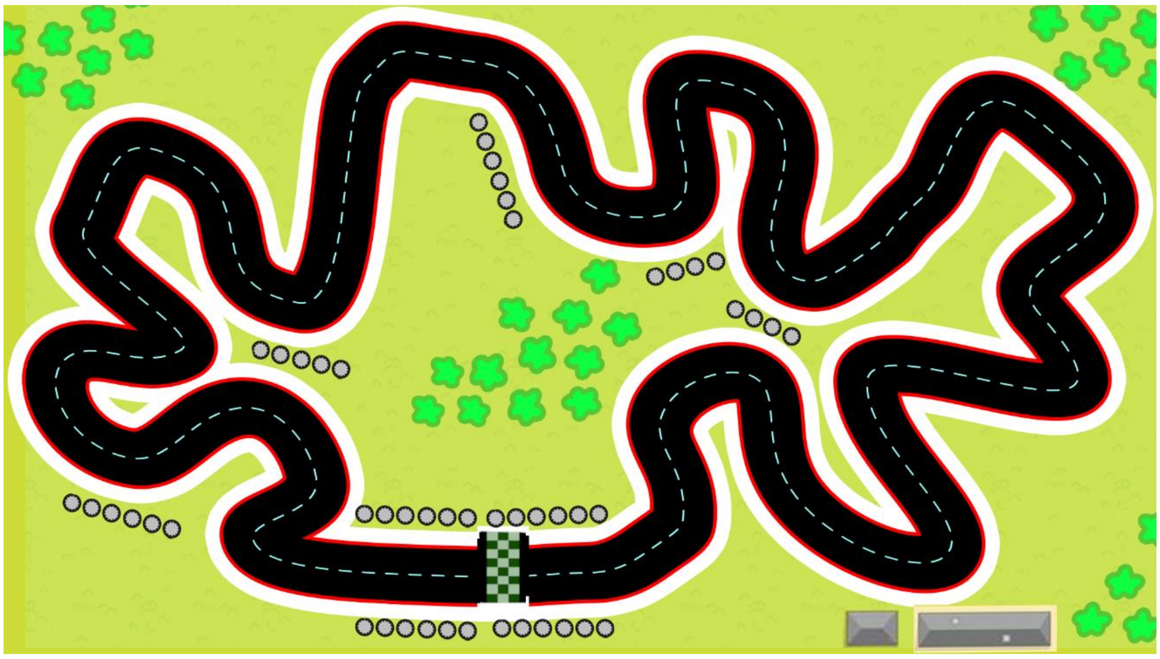


Fig 4.3 graphical asset, MapXi.png

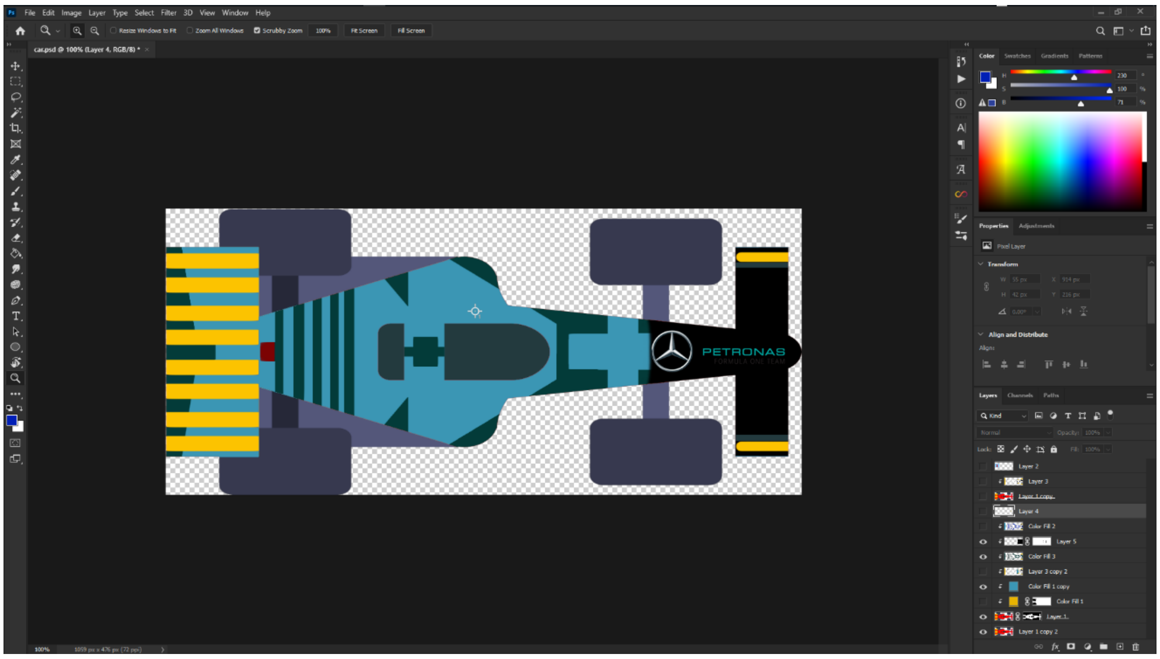


Fig 4.4 graphical assest, Car1.png

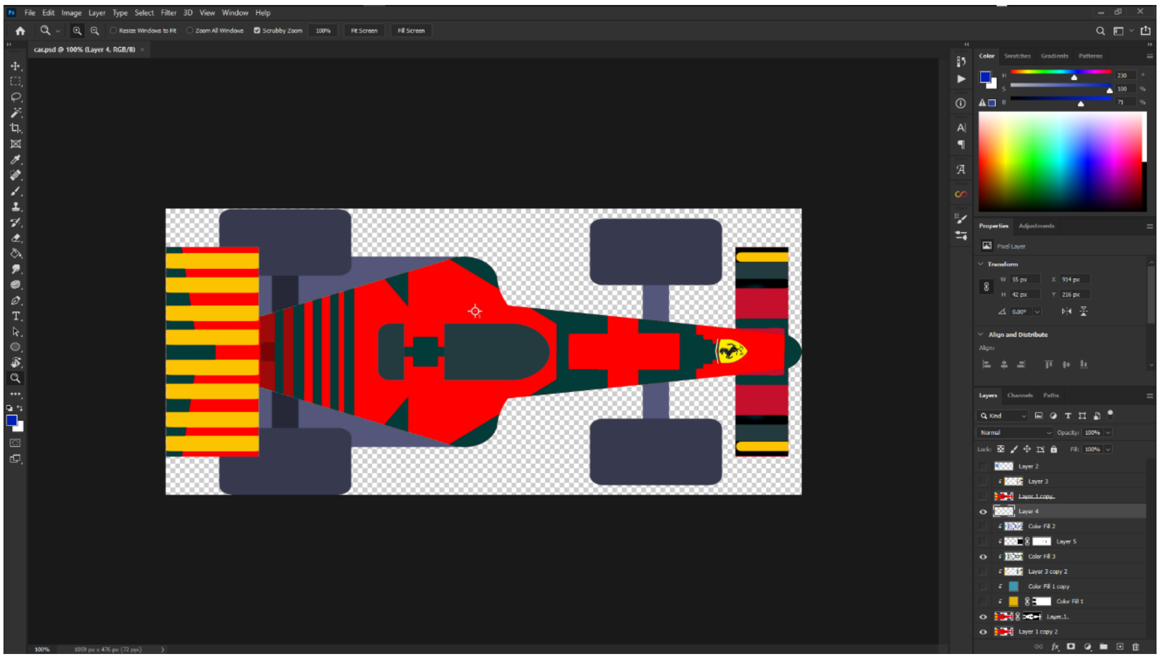


Fig 4.5 graphical assest, Car2.png

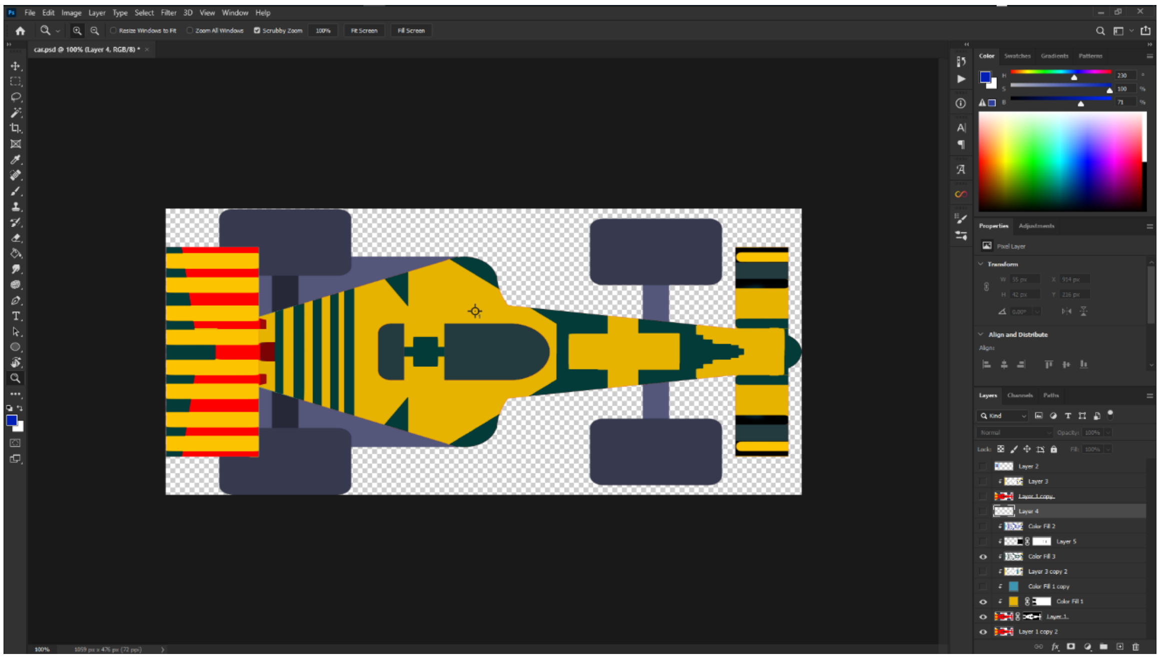


Fig 4.6 graphical assest, Car 3.png

4.2 Experimental Results & Analysis

Table: 4.1 Showing number of species and max fitness over 100 generations between No hidden Layer and with hidden Layer.

Generation	**Species (no hidden layer)**	**Max Fitness (no hidden layer)**	**Species (with 2 hidden layer)**	**Max Fitness (with 2 hidden layer)**
10	1	58473.73	30	210.26
20	2	59333.73	17	210.26
30	2	291186.73	2	1850.00000
40	2	291186.73333	3	2716.86667
50	2	291186.73333	3	2852.00000
60	2	308974.73333	3	2914.00000
70	2	308974.73333	3	2914.00000
80	2	308974.73333	4	2953.33333
90	2	308974.73333	4	8345.13333

100	2	308974.73333	3	8345.13333

Here, we can observe how the neural network and the course of evolution varies significantly depending on whether there is a hidden layer or not. Here is the top 100-generation outcome after running the simulation for hundreds of generations. Here, it is clear that the model without the hidden layer is better fit than the model with the hidden layer. As a result, on the model without hidden layers, more automobiles crossed the finish line.

As far as we are aware, hidden layers are just layers of mathematical operations, each of which is meant to give an output that is particular to the intended outcome. For instance, squashing functions are a name for several types of hidden layers. Because they take an input and output a value between 0 and 1, which is the range for determining probability, these functions are especially helpful when the algorithm's desired result is a probability.
For instance, ResNet handles some of these issues. One of the most well-known deep learning models is the residual network (ResNet), which was first presented in a paper by Shaoqing Ren, Kaiming He, Jian Sun, and Xiangyu Zhang. The design, which is based on VGG-19, has a 34-layer plain network where the shortcut or skip connections are added.

4.3 Discussion

Simply described, hidden layers are layers of mathematical operations, each one intended to produce an output specific to the desired outcome. For instance, squashing functions are a name for several types of hidden layers. Because they take an input and output a value between 0 and 1, which is the range for determining probability, these functions are especially helpful when the algorithm's desired result is a probability.
The operation of a neural network can be decomposed into particular data

transformations thanks to hidden layers. Each function in a hidden layer is tailored to deliver a certain result. For instance, later layers may combine functions from a hidden layer that are used to recognize human eyes and ears to recognize faces in photos. Even if the functions to recognize eyes on their own are insufficient to distinguish things on their own, they can work together in a neural network.

CHAPTER 5

Impact on Society, Environment and Sustainability

5.1 Impact on Society

Over the past several years, there has been a rise in interest in the quick development of the technologies that enable automated vehicles (AVs). It is realistic to assume that vehicles with higher levels of automation will be on the road in the near future, affecting numerous facets of our society along the way, as more manufacturers start to test AV prototypes and sell cars with some automation features. To maximize the benefits of AVs and lessen the issues they pose to society, it is crucial to understand their possible effects. AV consequences extend successively onto various facets of society, much like ripple effects. Traffic and travel expenses are primarily affected on a first-order basis. For instance, AVs can minimize the inefficiency of journey time and lower parking expenses by independently self-parking in less expensive locations. Second-order effects that have an impact on travel habits and the transportation system are produced as a result of the first-order effects. AVs may increase car miles driven while decreasing demand for non-motorized and public transit (VMT). Less lanes, narrower lanes, smaller medians, and fewer street signs and signals may also be brought about by AVs. Finally, equality, the economy, and the environment are all significantly impacted by first- and second-order consequences. system. AVs may reduce the demand for public transit and non-motorized transportation and increase vehicle miles traveled (VMT). AVs may also result in fewer lanes, narrower lanes, smaller medians, and fewer street signs and signals. Finally, the first- and second-order impacts have profound influences on equity, economy, and the environment.

5.2 Impact on Environment

Despite the advantages, it's feasible that AV penetration will harm non-motorized and public transportation. People may become more dependent on their cars as AV technology makes them more usable and accessible. Smaller parking and operating spaces are needed for AVs.

Driving costs drop when AVs relieve congestion and parking space constraints, which could then unleash latent driving demand, probably at the expense of cheaper modes of transportation like transit and non-motorized mobility. Additionally, since linked AVs don't need as many traffic signals or signs to function, there can be more uncontrolled intersections, which make it harder for pedestrians and cyclists to cross the street and put them in more danger. Additionally, as more people start using AVs, transportation investments may start to lean more in favor of cars, further marginalizing infrastructures for transit and non-motorized transportation.

Climate change is greatly impacted by the increase in automated machinery. Even while it can result in the loss of manual labor jobs, the environmental impact is positive overall. Compared to electronic machines, heavy duty manual machinery releases more CO2 into the atmosphere. Automated equipment can help cut carbon emissions in half, clearing the air as a result.

5.3 Ethical Aspects

In order to provide a comprehensive response, we must first define what is immoral. Is it conceivable for a thing, a product, or an object to be unethical? The definition of ethics that is most frequently used has to do with morals and is much more appropriate for behavior than for things. Since automation is a tool rather than a behavior, the topic of "is automation ethical" really refers to the invention and application of automation software.

Therefore, the development of business process automation is entirely ethical because it aims to enhance working conditions for thousands of people worldwide. When automation is used improperly or unethically, this becomes unethical. This, fortunately, makes those ethical concerns about automation completely avoidable.

5.4 Sustainability Plan

There were no costs as NEAT is an open-source algorithm. Since we ran our simulation without actually implementing the autonomous driving intelligence on any hardware, there was no hardware expense. We may therefore require only a small number of resources, which could be expensive, for further investigation on this issue.

CHAPTER 6

Summary, Conclusion, Recommendation and Implication for Future Research

6.1 Summary of the Study

Artificial systems called neural networks were influenced by biological neural networks. These systems acquire task-specific knowledge by being exposed to a variety of datasets and examples.

In this research, we wanted to create a neural network that could grow an intelligence for autonomous vehicle driving on any course.

For better results and understanding, we can also modify the NEAT algorithm's configuration, which is a project unto itself. We can also learn the theory underlying the NEAT algorithm and draw lessons from it.

6.2 Conclusions

AI and neural networks are particularly well-liked in the modern world. This makes our lives safer and simpler while also assisting us in solving a variety of issues. The future's solution, autonomous driving is safe and good to the environment. The ability to interact with the environment is provided for disabled persons and people with limited mobility, as well as new career opportunities. Since clean energy will be used to power sustainable automobiles, carbon and greenhouse gas emissions will be close to zero. As a result, our study involved researching and using simulation to help develop autonomous driving. This makes it possible to provide more data sets for automated driving while minimizing danger. Which can be put into practice in real life with greater assurance and security.

6.3 Implication for Further Study

NEAT works effectively as the "brains" of virtual creatures because it is already an algorithm inspired by evolution.

Limitations:

NEAT's key drawback is its sluggish convergence to optimal outcomes, particularly in complex or difficult contexts. Gradient approaches have the potential to be significantly faster, and recent developments in Deep RL Policy Gradients (algorithms like A3C and DDPG) suggest that RL can handle settings that are far more complicated than NEAT.

Future Scope:

As we create and investigate autonomous driving through simulation. This will produce some high-quality data. It will make it possible to produce larger data sets for automated driving without taking any chances. Which can be put into practice in real life with greater assurance and security.

References

[1] S. M. J. Jalali, P. M. Kebria, A. Khosravi, K. Saleh, D. Nahavandi and S. Nahavandi, "Optimal Autonomous Driving Through Deep Imitation Learning and Neuroevolution" 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC) Bari, Italy. pp. 2, October 6-9, 2019

[2] S. Grigorescu, B. Trasnea, T. Cocias, G. Macesanu, "A Survey of Deep Learning Techniques for Autonomous Driving", Article in Journal of Field Robotics· November 2019, pp. 4

[3] A. Agnihotri, P. Saraf, K. R. Bapnad, "A Convolutional Neural Network Approach Towards Self-Driving Cars", 2019 IEEE 16th India Council International Conference (INDICON), 13-15 December 2019, pp. 2

[4] A. Wilson, "Autonomous Driving: deep machine learning

approaches", University of Nottingham: MSc Machine Learning in Science, 2020, pp. 2

[5] K. O. Stanley and R. Miikkulainen, "Efficient Evolution of Neural Network Topologies", 2002 Congress on Evolutionary Computation (CEC ’02). Piscataway, NJ: IEEE, pp. 1

[6] Neuroevolution of Augmenting Topologies, available at <<https://en.wikipedia.org/wiki/Neuroevolution_of_augmenting_topologies#Extensions>>, last accessed on 15-07-22 at 10:00 PM

[7] HyperNEAT, available at <<https://en.wikipedia.org/wiki/HyperNEAT>>, last accessed at 16-07-22 at 9:00 PM

[8] Compositional pattern-producing network, available at <<https://en.wikipedia.org/wiki/Compositional_pattern-producing_network>>, last accessed on 16-07-22 at 11:00PM

[9] HyperNEAT User’s Page, available at <<http://eplex.cs.ucf.edu/hyperNEATpage/>>, last accessed on 07-08-22 at 10:15PM

[10] Comparison of NEAT and HyperNEAT on a Strategic Decision-Making Problem Jessica Lowell, Kir Birger, and Sergey Grabkovsky CS 6140 Final Project

[11] J. Drchal, M. Snorek “HyperNEAT Controlled Robots Learn How to Drive on Roads in Simulated Environment”, Conference Paper May 2009, DOI: 10.1109/CEC.2009.4983067 · Source: DBLP

[12] E. Haasdijk, A. A. Rusu, A.E. Eiben, “HyperNEAT for Locomotion Control in Modular Robots”, Conference Paper September 2010, DOI: 10.1007/978-3-642-15323-5_15 · Source: DBLP

[13] Z. Buk, J. koutn´ık, and M. Snorek, “NEAT in HyperNEAT substituted with genetic programming”, Conference Paper September 2009, DOI: 10.1007/978-3-642-04921-7_25 · Source: dx.doi.org

AUTONOMOUS DRIVING COMPARISON BETWEEN NEAT AND HYPERNEAT USING PYGAME

ORIGINALITY REPORT

26%	19%	13%	12%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	dspace.daffodilvarsity.edu.bd:8080 Internet Source	7%
2	en.wikipedia.org Internet Source	3%
3	Submitted to Daffodil International University Student Paper	3%
4	Seyed Mohammad Jafar Jalali, Parham M. Kebria, Abbas Khosravi, Khaled Saleh, Darius Nahavandi, Saeid Nahavandi. "Optimal Autonomous Driving Through Deep Imitation Learning and Neuroevolution", 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), 2019 Publication	1%
5	nn.cs.utexas.edu Internet Source	1%
6	Evert Haasdijk. "HyperNEAT for Locomotion Control in Modular Robots", Lecture Notes in Computer Science, 2010 Publication	1%