KANTIPUR ENGINEERING COLLEGE

(Affiliated to Tribhuvan University)

Dhapakhel, Lalitpur

[Subject Code: CT755]

A MAJOR PROJECT REPORT ON

# AUTONOMOUS CAR SIMULATOR USING ANN AND NEAT (REINFORCEMENT LEARNING)

**Submitted by:**

Ajay Chaudhary (KAN075BCT004)

Ashim Budha Chhetri (KAN075BCT016)

Ayush Niraula (KAN075BCT018)

Manoj Subedi (KAN075BCT033)

A MAJOR PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR IN COMPUTER ENGINEERING

**Submitted to:**

Department of Computer and Electronics Engineering

December, 2022

# AUTONOMOUS CAR SIMULATOR USING ANN AND NEAT (REINFORCEMENT LEARNING)

**Submitted by:**

**Ajay Chaudhary (KAN075BCT004)**

**Ashim Budha Chhetri (KAN075BCT016)**

**Ayush Niraula (KAN075BCT018)**

**Manoj Subedi (KAN075BCT033)**

**Supervised by:**

**Er. Mohan Bhandari**

**Machine Learning Engineer**

**A MAJOR PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR IN COMPUTER ENGINEERING**

**Submitted to:**

**Department of Computer and Electronics Engineering**

**Kantipur Engineering College**

**Dhapakhel, Lalitpur**

**December, 2022**

# ABSTRACT

A self-driving car ,also known as an autonomous car or a driver-less car, is a vehicle that travels between locations without the assistance of a human driver using a mix of sensors, cameras, radar, and artificial intelligence (AI). With the help of Artificial Neural Networks (ANNs) and Neuroevolution of Augmenting Topologies, we offer a simulator for autonomous vehicles (NEAT). This project's goal is to provide a simulation environment for testing and assessing the effectiveness of various ANN and NEAT models for driving an autonomous vehicle. NEAT algorithm is better algorithm when compared to other as NEAT is able to evolve the neural network in real-time, allowing it to adapt quickly to new situations and changing environments. This is particularly useful in an autonomous driving simulator, where the vehicle needs to respond to new and dynamic driving scenarios.

The simulation environment is built around a real-world representation of an autonomous vehicle and its surroundings, which includes angled and curvy roads. In order to teach the ANN and NEAT models how to steer the automobile based on inputs from sensors like ray casting, reinforcement learning techniques are used during training. Our simulation tests' findings indicate that the NEAT models perform better than the conventional ANN models in terms of driving accuracy and speed. The outcomes of the simulation also show NEAT's potential for real-time neural network evolution, which may be used in various robotics and control applications. This research makes a significant addition to the field of autonomous vehicle simulation and control and establishes the groundwork for future research in this sector.

***Keywords****– Neural network, Reinforcement learning, Self-driving.*

# ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all those who have contributed to the completion of this project. Our special thanks go to our constant source of encouragement Er. Mohan Bhandari sir for his invaluable guidance, support, and encouragement throughout the project from the inception to completion. Your support has been invaluable in helping us achieve our goals, and we are grateful for the opportunities you have provided us.

We also extend our thanks to HOD Er. Rabindra Khati sir for providing us with the necessary resources and facilities to carry out this project. We are grateful to Er. Bishal Thapa sir for his dedicated help and support. Finally, we would like to acknowledge the efforts and contributions of our team members, who have worked tirelessly to make this project a success. Thank you all for your support and contributions as this project would not have been possible without these assistance.

Ajay Chaudhary (Kan075Bct004)
Ashim Budha Chhetri (Kan075Bct016)
Ayush Niraula (Kan075Bct018)
Manoj Subedi (Kan075Bct033)

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF ABBREVIATION

| | |
|---|---|
| ANN | Artificial Neural Network |
| NEAT | NeuroEvolution of Augmenting Topologies |
| RL | Reinforcement Learning |
| AI | Artificial Intelligence |
| NN | Neural Network |

# CHAPTER 1
# INTRODUCTION

## 1.1 Background

Automated cars are becoming more common in today's society. Prototype automobiles have already been automated and found to be dependable enough to drive themselves under normal driving circumstances. Due to its capacity to learn from experience and make judgments based on sensory inputs, Artificial Neural Networks (ANNs) and Reinforcement Learning (RL) have been extensively applied in the field of autonomous vehicle control. Traditional ANNs, on the other hand, have a number of drawbacks, including a need of highly trained data. An evolutionary algorithm that has been demonstrated to be successful for developing neural networks is the NeuroEvolution of Augmenting Topologies (NEAT) algorithm. NEAT, in contrast to conventional ANNs, permits the development of both the network's weights and topology, leading to more adaptable and flexible network topologies. For testing and analyzing various control algorithms, simulation environments have grown in significance in the field of autonomous cars. A 2D autonomous automobile simulator can offer a practical and affordable testing and evaluation environment for various ANN and NEAT models for autonomous vehicle control.

Our project's objective is to create a 2D autonomous automobile simulator utilizing ANNs and NEAT and assess how well they perform in terms of driving an autonomous vehicle. The ANN and NEAT models will be trained using RL approaches, and the simulation environment will be based on a realistic representation of an autonomous automobile and its surrounding surroundings. For testing and analyzing various control algorithms, simulation environments have grown in significance in the field of autonomous cars. A 2D autonomous automobile simulator can offer a practical and affordable testing and evaluation environment for various ANN and NEAT models for autonomous vehicle control. Our project objective is to create a 2D autonomous automobile simulator utilizing ANNs and NEAT and assess how well they perform in terms of driving an autonomous vehicle. The ANN and NEAT models will be trained using RL approaches, and the simulation environment will be based on a realistic representation of an autonomous automobile and its surrounding surroundings[1].

## 1.2 Problem Statement

The importance of simulation environments for testing and analyzing various control algorithms for autonomous cars has grown. However, current autonomous vehicle simulation settings frequently fall short in their abilities to simulate intricate and varied driving situations and are constrained by the presumptions they make about the car and its surroundings. The objective of this project is to create a 2D autonomous automobile simulator using the NeuroEvolution of Augmenting Topologies (NEAT) algorithm and to assess how well it performs in a realistic simulation environment when managing an autonomous vehicle. The NEAT algorithm will be taught using Reinforcement Learning (RL) techniques, and the simulation environment will be based on a realistic model of an autonomous automobile and its surrounding environment. To illustrate the efficacy and efficiency of the NEAT algorithm for managing an autonomous automobile, the simulation results will be examined and contrasted with those from other systems currently in use[2].

## 1.3 Objective

Following listed are the major objective we are considering for our proposed Self-Driving Car Simulator:

1. To create a 2D simulation environment that accurately represents an autonomous vehicle's surroundings.
2. To put the NeuroEvolution of Augmenting Topologies (NEAT) method to use in the simulated environment for driving the self-driving automobile
3. To analyze and interpret the results of the simulation, and identify potential areas for improvement and future work.
4. To evaluate the performance of the NEAT algorithm on different tracks, and analyze the behavior of the autonomous car under different conditions.

## 1.4 Scope and Usability

### 1.4.1 Scope

1. The project provides a realistic simulation environment for an autonomous car and its surrounding environment, allowing for the testing and evaluation of control algorithms in a safe and controlled setting.
2. The project provides insights into the behavior and performance of the NEAT algorithm, and identifies potential areas for improvement and future work.

### 1.4.2 Usability

1. The 2D Autonomous Car Simulator can be used by researchers, engineers, and students in the field of autonomous vehicles, as a tool for testing and evaluating control algorithms for autonomous cars.
2. The project can be used as a teaching tool for students learning about autonomous vehicles, control systems, and simulation environments.
3. The project can be further developed and extended to include additional features, such as more complex driving scenarios, multiple cars, and different road conditions.

## 1.5 System Requirement

### 1.5.1 Software Requirement

This project is targeted for simulation purposes; therefore we cannot properly run this system on low-end devices or mobile platforms smoothly as it has some software requirements. Following listed are some of the requirements for its proper functioning:

1. Operating System: Windows, MacOS.
2. Web Library and Language Involved: Python, NEAT, ANN, Pygame
3. IDEs: Visual Studio Code, PyCharm

### 1.5.2 Hardware Requirement

Following listed are the hardware components required for the optimal performance of this system:

1. Laptop with Intel Core 2 Duo Processor (I series recommended), with minimum 4GB RAM (8GB recommended).

## 1.6 Project Features

1. Simulated/Custom tracks can be designed considering difficult curves and angles which will help in improvement of tracks and fixing potential errors
2. The project provides a user-friendly interface for interacting with the simulation environment and controlling the autonomous car, allowing for easy testing and evaluation of control algorithms.
3. Calculation of best fitness and generation for each custom tracks.
4. The project is designed to be flexible and customizable, allowing for easy extension and modification to include additional features, such as more complex driving scenarios, multiple cars, and different road conditions.

## 1.7 Feasibility Study

This project is ideal for any simulation based learning and training.We are intending to design this project in such a way that, the 2D simulation can be done before implementing different tracks for self driving car.

### 1.7.1 Economic Feasibility

The total cost of the project, including the cost of software development, hardware, and any other costs associated with the project was none as free version of software and tools was used through out the project development. Unless hardware oriented area is not involved, we are intending to build our simulation project that only uses a laptop with required programming for its smooth functioning.

### 1.7.2 Schedule Feasibility

Our project has been broken down into components, each of which includes assessing the availability of the necessary resources and the project schedule to see if the project can be finished within the allotted time limits. Both the model's training and the ray casting component's implementation took time, as seen in the Gantt chart. We also invested a lot of effort in research to locate the greatest supporting paper for our idea.

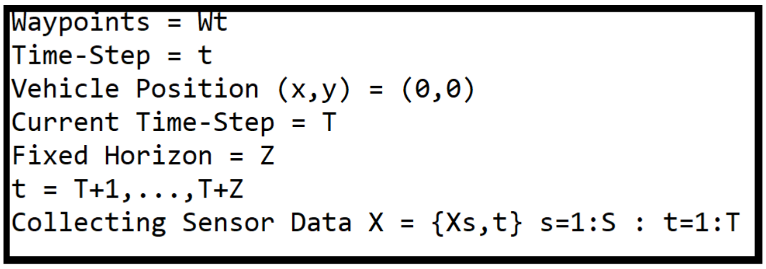
### 1.7.3 Technical Feasibility

As stated in the system requirements, many software components, libraries, and tools were required to complete this project. In contrast, the project's testing and training on laptops with Intel processors and both 4 and 8 GB of RAM went well.

# CHAPTER 2
# LITERATURE REVIEW

## 2.1 Related Works

From Kashyap Chitta and teams work we found that they presented the official CARLA Leaderboard stats for route completion(RC), infraction score(IS), and driving score (DS). RC is the portion of the route's distance that an agent completes for a particular route before deviating from it or hitting a roadblock. Every accident, lane infraction, red light infraction, and stop sign infraction has a cumulative multiplicative penalty known as IS. The RC weighted by the IS for that route is used to calculate the DS. Then they give the mean performance across all 42 routes after computing each measure for each route. For each model, They run their internal assessment three times, and for each measure, they give the mean and standard deviation.



```
Waypoints = Wt
Time-Step = t
Vehicle Position (x,y) = (0,0)
Current Time-Step = T
Fixed Horizon = Z
t = T+1,...,T+Z
Collecting Sensor Data X = {Xs,t} s=1:S : t=1:T
```

End-to-end trajectory planning, which employs waypoints as outputs, is a popular method for learning the driving job via professional examples. A waypoint is defined as the location of a vehicle in a BEV projection of the vehicle's local coordinate system at a given time step in an expert presentation. The vehicle's front is aligned along the positive y-axis, and the coordinate axes are set so that the vehicle is centered at the current time step. A trajectory that may be utilized to steer the vehicle is made up of waypoints from a series of future time-steps, where Z is a defined prediction horizon. They gather sensor data as our agent travels across the scene and store it in a fixed-length buffer of T time steps, where each parameter originates from a different sensor. They employed BEV semantic prediction in addition to waypoints as an adjunct task to enhance driving performance. BEV semantic prediction is a dense prediction job that aims to predict semantic labels at any spatiotemporal query location constrained to the spatial range and the time interval, in contrast to waypoints which are few in number and can be predicted discretely. A comprehensive comprehension of the scene dynamics may be obtained by forecasting both observed and future semantics. Since the location and orientation of vehicles contain information about their velocity, dynamics prediction from a single input frame is feasible.

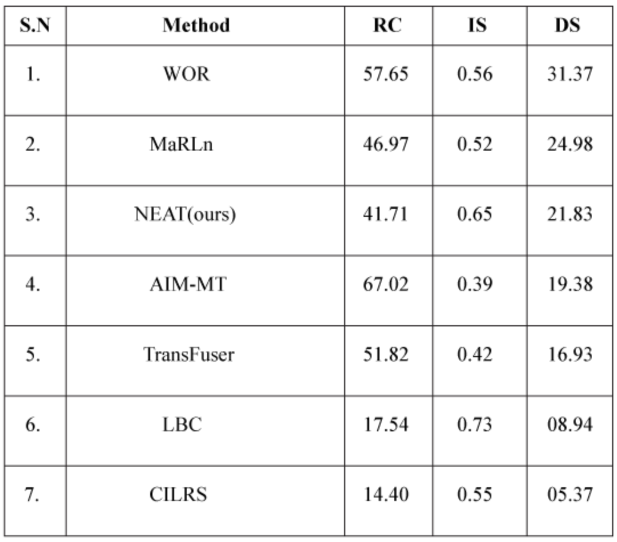
With their innovative NEAT feature representation, they advance the field of interpretable, high-performance, end-to-end autonomous driving in this study. The strategy uses the best level of safety among cutting-edge techniques on the CARLA simulator to handle the difficult challenge of combining BEV semantic prediction and vehicle trajectory planning from camera inputs. In terms of input modalities and output task/supervision, NEAT is general and adaptable, and they want to eventually integrate it with orthogonal concepts. In the paper by Kashyap and Prakash, they describe their experimental setup, compare the driving performance of NEAT to a number of baselines, present an ablation study to emphasize the significance of various architectural components, and demonstrate the interpretability of our approach through visualizations derived from our trained model. The comparison was made based on different parameters[3].

1. Route Completion (RC): RC is the percentage of the route distance completed by the agent before it deviates from the route or gets blocked.
2. Infraction Score (IS): IS is a cumulative multiplicative penalty for every collision, lane infraction, vehicle collision and edge collision.
3. Driving Score (DS): DS is computed as the RC weighted by the IS for that route.

## 2.2 Existing System

### 2.2.1 Waabi

Waabi, a Toronto-based AI startup developed an advanced simulator that can train autonomous vehicles to handle nearly limitless types of driving conditions–in a virtual world and do so faster and more thoroughly than self-driving rivals that prioritize road tests. The Waabi World platform is more comprehensive than any used by competi-



| S.N | Method | RC | IS | DS |
|---|---|---|---|---|
| 1. | WOR | 57.65 | 0.56 | 31.37 |
| 2. | MaRLn | 46.97 | 0.52 | 24.98 |
| 3. | NEAT(ours) | 41.71 | 0.65 | 21.83 |
| 4. | AIM-MT | 67.02 | 0.39 | 19.38 |
| 5. | TransFuser | 51.82 | 0.42 | 16.93 |
| 6. | LBC | 17.54 | 0.73 | 08.94 |
| 7. | CILRS | 14.40 | 0.55 | 05.37 |

Figure 2.1: Comparison

tors as it can more accurately mimic real-world scenarios and create the types of rare, challenging "edge cases" that occur on the road only rarely. There are some technical similarities between Waabi and our project, as both uses machine learning models and both have the features of inputting various parameter as input neurons. Given that, there are some differences that makes our project better than Waabi itself, for example simulated/Custom tracks can be designed considering difficult curves and angles which will help in improvement of tracks and fixing potential errors . Our project provides a user-friendly interface for interacting with the simulation environment and controlling the autonomous car, allowing for easy testing and evaluation of control algorithms. Calculation of best fitness and generation for each custom tracks and project itself is designed to be flexible and customizable, allowing for easy extension and modification to include additional features, such as more complex driving scenarios, multiple cars, and different road conditions[4].

Pros

1. Complex use of machine learning algorithm and inputs for better output
2. Large amount of datasets

3. More efficient and effective hardwares and tools

Cons:

1. Ability to change the testing environment by changing tracks
2. Calculation of best fitness and generation for each custom tracks
3. No user-friendly interface for interacting with the simulation environment

# CHAPTER 3
# METHODOLOGY

## 3.1 Block Diagram

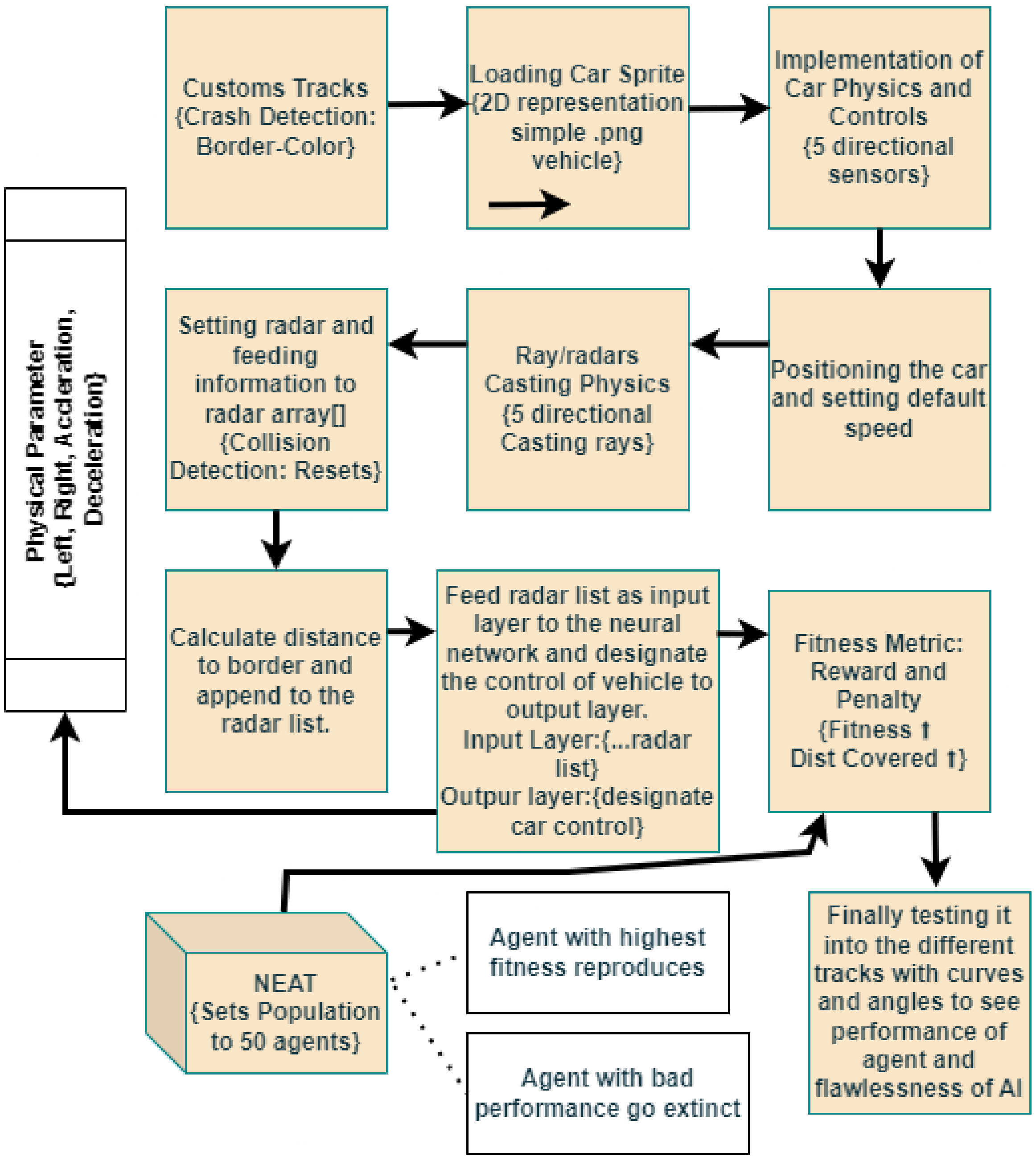

Customs Tracks {Crash Detection: Border-Color}
Loading Car Sprite {2D representation simple .png vehicle}
Implementation of Car Physics and Controls {5 directional sensors}
Positioning the car and setting default speed
Ray/radars Casting Physics {5 directional Casting rays}
Setting radar and feeding information to radar array[] {Collision Detection: Resets}
Physical Parameter {Left, Right, Accleration, Deceleration}
Calculate distance to border and append to the radar list.
Feed radar list as input layer to the neural network and designate the control of vehicle to output layer. Input Layer:{...radar list} Outpur layer:{designate car control}
Fitness Metric: Reward and Penalty {Fitness ↑ Dist Covered ↑}
NEAT {Sets Population to 50 agents}
Agent with highest fitness reproduces
Agent with bad performance go extinct
Finally testing it into the different tracks with curves and angles to see performance of agent and flawlessness of AI

Figure 3.1: Block Diagram

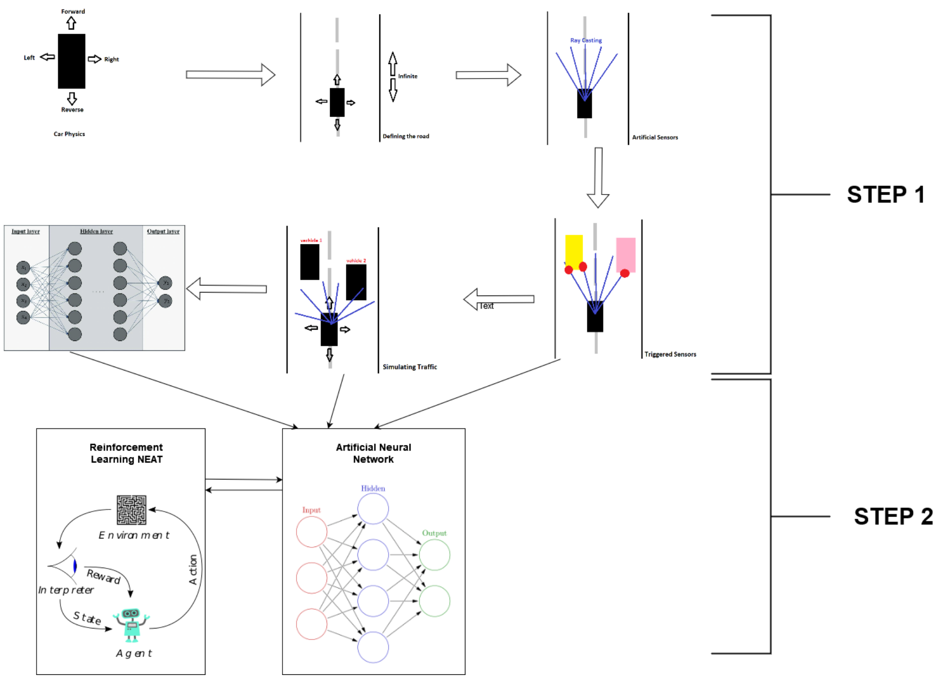


Figure 3.2: Simulation Diagram

## 3.2 Building the Simulator

Following are the block by block components which ultimately results in a self-driving car:

### 3.2.1 Physical Component and Environment Setup

Building the physical component is simply the process of building the environment of simulation including the physical component/motion. This project involves the process of using python and neat library to make a agent/vehicle race around in a custom tracks. We use different approaches including pygame, os ,math and NEAT for creating a environment in order to create an environment.

1. Importing the pygame and NEAT library for building simulation
2. Setting the default pygame environment with appropriate width and height that determines the entire sumulation environment.
3. Loading the customs tracks into the working environment

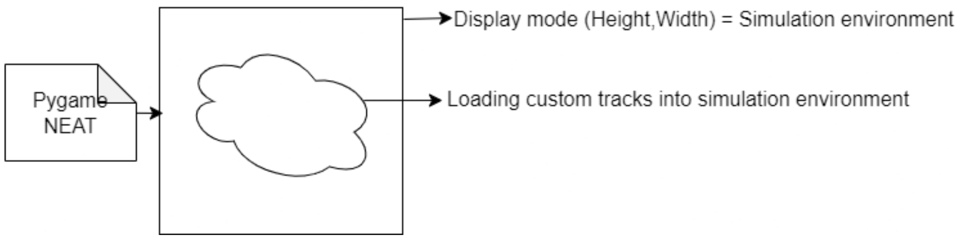

Display mode (Height,Width) = Simulation environment
Pygame
NEAT
Loading custom tracks into simulation environment

Figure 3.3: Environment Setup

### 3.2.2 Loading agent Sprite and Car Physics

This component implements physics to move our 2D car around the screen. For the first part we move the car using arrow keys. This component uses basic physics and trigonometry. We move the car using the keyboard listeners. Simply adjusting the x and y coordinates on both sides of the car we can make the car move which responds to the button pressed.

1. Loading the car sprite with the help of os path into the simulation environment
2. Setting the center and vector with angle , velocity, direction , alive rate and radars to zero and empty array.
3. Key listeners are use to move the agent, whose control is later on designated to neural network.

### 3.2.3 Defining The Road/Custom Tracks

. The defined edges of the road helps us to make the self-driving agent always take the right path. Once the agent is trained, it can self drive infinitely or upto defined generation. Defining the road and lanes will later help us to navigate the car properly and control the car in care of any obstacle ahead.

1. Environment here simply means the like and unlikeness between different tracks.
2. Each tracks are independent of each other in the sense that each track has different curves , different angles, different length and different narrow width.
3. Simulation in each track is different and independent to determine the result differently

4. Agent independently train in different tracks and each track record can be analyzed.

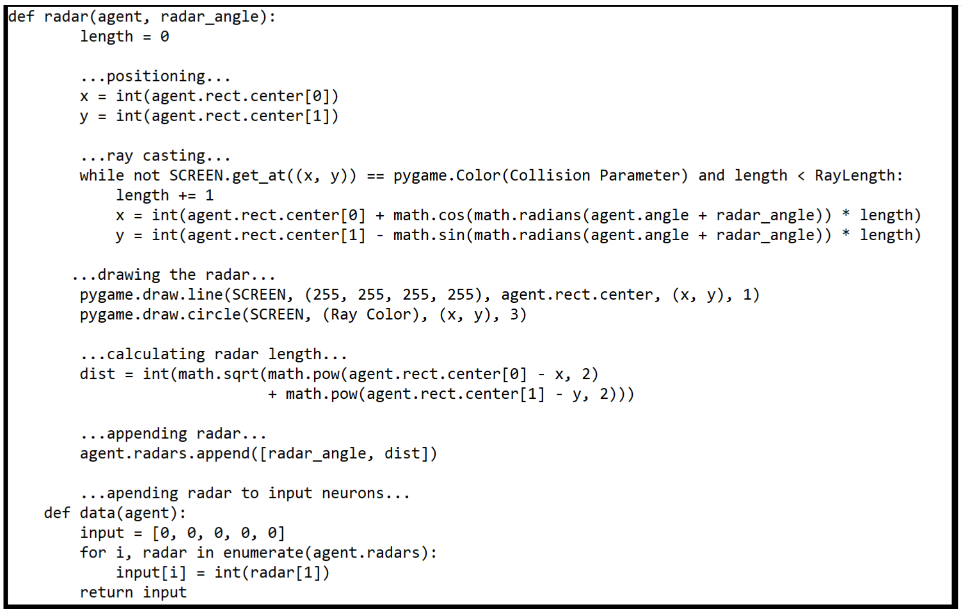
### 3.2.4 Ray Segmentation/ Ray Casting

Artificial sensors for our car are the most important component of our project. For this component we use ray casting. In computer graphics and computational geometry, ray casting is a rendering technique. It can turn a two-dimensional map into a three-dimensional viewpoint. Ray casting is a technique for converting a two-dimensional image into a three-dimensional projection by tracing rays from the viewing point into the viewing volume. Ray casting works on the idea that rays may be cast and traced in groups depending on geometric limitations. Ray casting involves obtaining a ray from the pixel via the camera and computing the intersection of all objects in the image. The pixel value from the nearest intersection is then retrieved and used as the projection's basis.

1. Determining the ray count in different direction , for our project we are estimating 3 or 5 rays.
2. Determining the ray length
3. Determining the ray spread. This is the angle in which the ray spread (pi/5) –¿ 45 degree
4. Ray spread change according to number of rays
5. Update ray angle
6. For x coordinate , ray length and sine ray angle parameter is used
7. For y coordinate , ray length and cosine ray angle parameter is used

### 3.2.5 Triggered Sensors

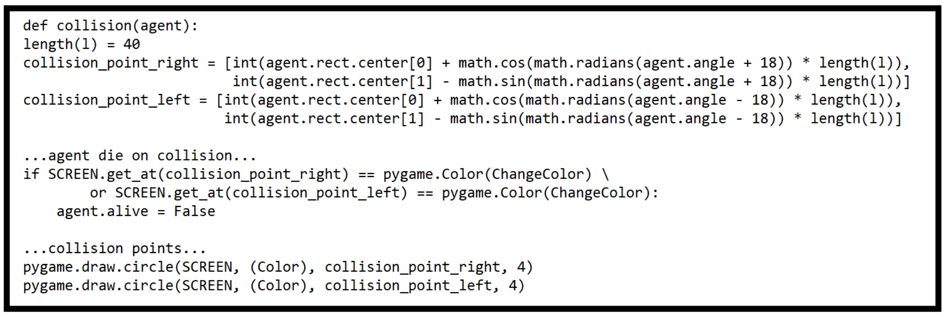
Self driving car makes the decision based on the output received from the sensors which in this case is triggering the minimum ray length. The car should be able to judge the edge of the road and finally make a decision mimicking the human brain. Therefore instead of assigning any task to the sensor trigger we send the output to the neural network.



```
def radar(agent, radar_angle):
        length = 0

        ...positioning...
        x = int(agent.rect.center[0])
        y = int(agent.rect.center[1])

        ...ray casting...
        while not SCREEN.get_at((x, y)) == pygame.Color(Collision Parameter) and length < RayLength:
            length += 1
            x = int(agent.rect.center[0] + math.cos(math.radians(agent.angle + radar_angle)) * length)
            y = int(agent.rect.center[1] - math.sin(math.radians(agent.angle + radar_angle)) * length)

       ...drawing the radar...
        pygame.draw.line(SCREEN, (255, 255, 255, 255), agent.rect.center, (x, y), 1)
        pygame.draw.circle(SCREEN, (Ray Color), (x, y), 3)

        ...calculating radar length...
        dist = int(math.sqrt(math.pow(agent.rect.center[0] - x, 2)
                             + math.pow(agent.rect.center[1] - y, 2)))

        ...appending radar...
        agent.radars.append([radar_angle, dist])

        ...apending radar to input neurons...
    def data(agent):
        input = [0, 0, 0, 0, 0]
        for i, radar in enumerate(agent.radars):
            input[i] = int(radar[1])
        return input
```

### 3.2.6 Collision Detection

To avoid accidents and collisions, the automobile should be able to recognize impediments for smooth and effective operation. It should also be able to determine the distance between the automobile and the barrier. Similarly, Track Detection is critical since the autonomous vehicle must stay on a predetermined path and stay inside the yellow lines on both sides of the road. For this project the car deforms upon collision while training.



```
def collision(agent):
length(l) = 40
collision_point_right = [int(agent.rect.center[0] + math.cos(math.radians(agent.angle + 18)) * length(l)),
                         int(agent.rect.center[1] - math.sin(math.radians(agent.angle + 18)) * length(l))]
collision_point_left = [int(agent.rect.center[0] + math.cos(math.radians(agent.angle - 18)) * length(l)),
                        int(agent.rect.center[1] - math.sin(math.radians(agent.angle - 18)) * length(l))]

...agent die on collision...
if SCREEN.get_at(collision_point_right) == pygame.Color(ChangeColor) \
        or SCREEN.get_at(collision_point_left) == pygame.Color(ChangeColor):
    agent.alive = False

...collision points...
pygame.draw.circle(SCREEN, (Color), collision_point_right, 4)
pygame.draw.circle(SCREEN, (Color), collision_point_left, 4)
```

### 3.2.7 Artificial Neural Network

1. Artificial neural networks (ANNs), also known as neural networks (NNs) or neural nets, are computer systems that are inspired by the biological neural networks that make up animal brains.

2. Artificial neurons are a set of linked units or nodes in an ANN that loosely replicate the neurons in a biological brain. Each link may send a signal to other neurons, just like synapses in a human brain.
3. An artificial neuron receives impulses, analyses them, and can send messages to other neurons. Each neuron's output is generated by some non-linear function of the sum of its inputs, and the ”signal” at a connection is a real number.
4. Edges are the terms for the connections. The weight of neurons and edges is frequently adjusted as learning progresses. The signal strength at a connection is increased or decreased by the weight. Neurons may have a threshold that allows them to send a signal only if the aggregate signal exceeds it.
5. Neurons are usually grouped into layers. On their inputs, separate layers may apply different transformations. Signals go from the first layer (input layer) to the last layer (output layer), maybe many times.
6. We now have a complete simulation of a drivable car with sensors that can see the road for other cars and traffic. But so far we are doing the driving which is about to change. This is called an artificial neural network where we make the car simulate and drive itself in a more human-like fashion.
7. Neural networks are computing systems inspired by the biological neural networks in our brain. Neurons are simply cells and different branch like structures receive signals. When stimulated enough, a neuron fires a signal through an axon.
8. Neural networks work well when in a team. Our brain has almost 86 billion neurons. For humans, different sensory organs like eyes and ears send signals to the brain for processing. The processing happens like a chain reaction where some neurons fire and some don't. And through the spinal cord and motor neurons it makes some muscles crack in as a reflex. This all happens in a split second.
9. Our car neural network will do something similar to that. Neurons on the first layer will be connected to sensors. They will send the signal forward sometimes while the last layer(output layer or third layer) will be connected to car controls.
10. Our project will require relatively medium neural networks which are divided into three levels.

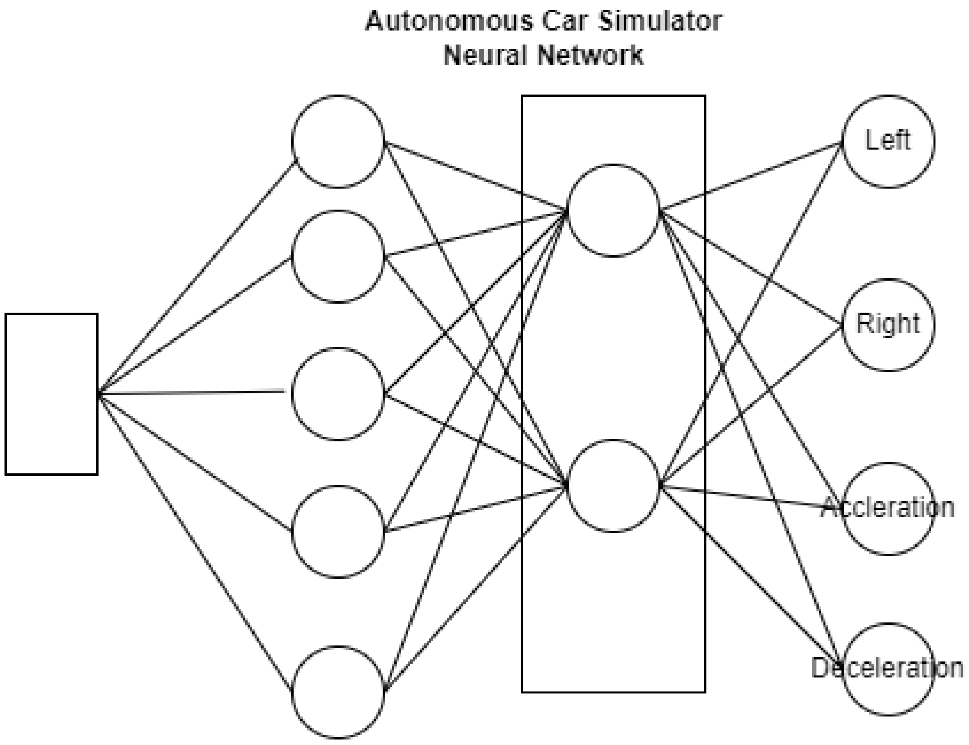


Figure 3.4: Artificial Neural Network

### 3.2.8 Calculation of neural network

1. We have basically around 5/6 ray casting sensors for our car model which is the input node and around 4 nodes as the output node.
2. The output node enables the car to take certain actions i.e, forward, reverse, left and right. Neural network's job is to train itself and learn how to drive a car on a randomly generated path.
3. The car should be modeled in such a way that it avoids every collision with other cars and edges.
4. Neural network is simply a collection of nodes called neurons. Neurons are linked together by weight between -1 to 1.
5. These neurons are placed together in layers namely input layer, hidden layer and output layer. Each node can have a value between 0 to 1.
6. Neuron linked with weight = -1 to 1 and Each node can have value = 0 to 1
7. Each node in input layer = data received for the ray casting
8. Node value = 1 (if collision) and Node value= 0 (Out of sensor range)
9. Node of hidden layer = Weight * Node value

10. Node of hidden layer = Summation of Input layer = Function Generation
11. Output layer = Trigger or action if crossed threshold.

### 3.2.9 Analyzing Result

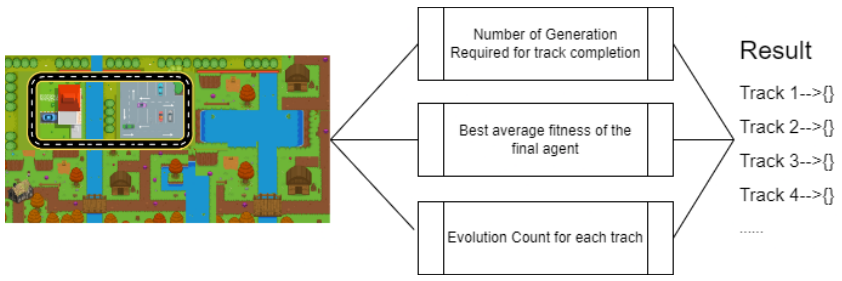

Number of Generation Required for track completion
Best average fitness of the final agent
Evolution Count for each trach
Result
Track 1-->{}
Track 2-->{}
Track 3-->{}
Track 4-->{}
......

Figure 3.5: Analyzing Result

### 3.2.10 Reinforcement Learning

Reinforcement learning is a machine learning training strategy that rewards desirable actions while penalizing undesirable ones. A reinforcement learning agent can sense and comprehend its surroundings, act, and learn via trial and error in general. Developers establish a way of rewarding desired actions and punishing bad ones in reinforcement learning. To motivate the agent, this strategy provides positive values to desired acts and negative values to undesirable behaviors. To obtain an ideal answer, the agent is programmed to seek long-term and greatest overall return. These long-term objectives keep the agent from stagnating on smaller objectives. The agent eventually learns to shun the unpleasant and focus on the good [5].

### 3.2.11 NEAT for Reinforcement Learning

Reinforcement learning differs from unsupervised learning in terms of objectives. In unsupervised learning, the aim is to detect similarities and differences between data points; in reinforcement learning, the goal is to develop an appropriate action model that maximizes the agent's total cumulative reward. Some key terms that describe the basic elements of an RL problem are:

1. Environment — Physical world in which the agent operates

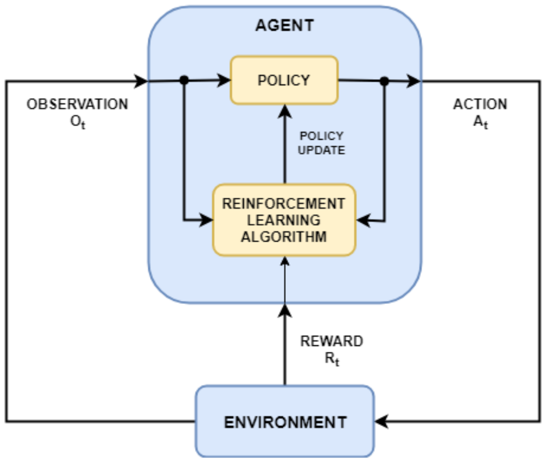


Figure 3.6: Reinforcement Learning

2. State — Current situation of the agent
3. Reward — Feedback from the environment
4. Policy — Method to map agent's state to actions
5. Value — Future reward that an agent would receive by taking an action in a particular state

### 3.2.12 Importance of using NEAT

1. The learning process is expensive for the agent, specially, in the beginning steps for most of the algorithm. The learning process in NEAT is comparatively more effective.
2. Fitness is expected to be a Python float value. The absolute magnitude and signs of these fitnesses are not important, only their relative values. These values can be easily seperated by use of NEAT.
3. Upon implementing a fitness function, Following steps is carried out:
   (a) Create a neat.config.Config object from the configuration file .
   (b) Create a neat.population.Population object using the Config object created above.
   (c) Call the epoch method on the Population object, giving it your fitness function and the maximum number of generations you want NEAT to run.
4. After these three things are completed, NEAT will run until either they reach the

specified number of generations, or at least one genome achieves the max fitness threshold value that is specified in the config file

5. Functions are available in the neat.visualize module to plot the best and average fitness vs. generation, plot the change in species vs. generation, and to show the structure of a network described by a genome
6. NEAT-Python allows the user to provide drop-in replacements for some parts of the NEAT algorithm, and attempts to allow easily implementing common variations of the algorithm
7. The default reproduction scheme uses explicit fitness sharing and a fixed species stagnation limit. This behavior is encapsulated in the DefaultReproduction class [6].

### 3.2.13 Implementation of neat into simulation

In NEAT, a population of neural networks is evolved over generations through mutation and crossover operations, guided by a fitness function that measures how well each network performs on a given task. The networks are evolved to become increasingly complex, with new nodes and connections added over time, in order to better model the task at hand.

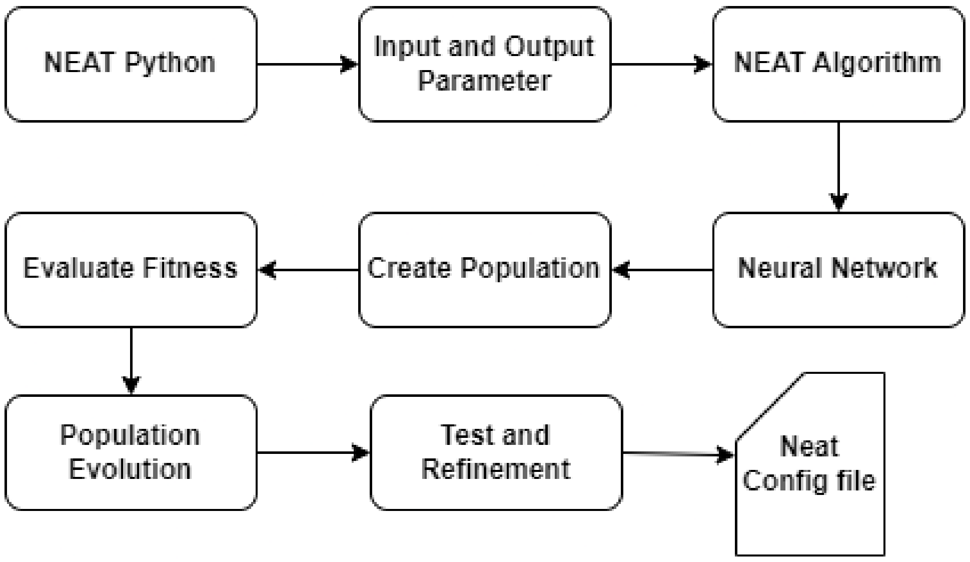


Figure 3.7: NEAT Algorithm

The NEAT configuration file is a text file that contains various parameters and settings that control the behavior of the NEAT algorithm in ”neat-python”. The configuration file is typically used to specify the population size, mutation rates, and other important parameters of the algorithm. Here are some of the key settings that are typically included in a NEAT configuration file:

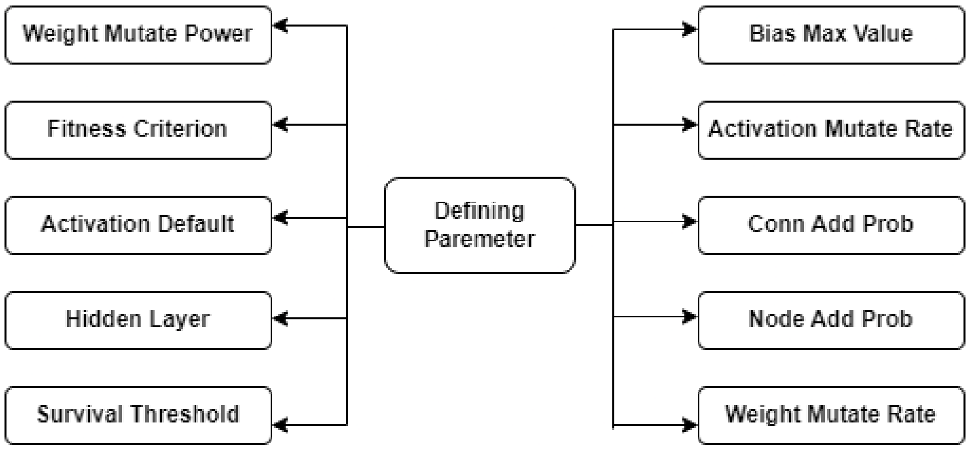


Figure 3.8: Defining Parameter

1. **Pop Size:** Set the population size to a value that is appropriate for your task and computational resources. For example, we can start with a population size of 100.
2. **Fitness Criterion:** Set the fitness criterion to ”maximize”, since we want to maximize the car’s performance on the task.
3. **Fitness Threshold:** Set the fitness threshold to a value that represents an acceptable level of performance for your task. For example, we can set the threshold to a score that represents completing the task with a certain level of efficiency.
4. **Mutate Prob:** Set the mutation probability to a value that is appropriate for the task and network architecture. For example, we can start with a mutation probability of 0.1.
5. **Add Node Prob:** Set the probability of adding a new node to a value that is appropriate for your task and network architecture. For example, we can start with a probability of 0.03.
6. **Add Conn Prob:** Set the probability of adding a new connection to a value that is appropriate for your task and network architecture. For example, we can start with a probability of 0.05.
7. **Compatibility Threshold:** Set the compatibility threshold to a value that is appropriate for your task and network architecture. For example, we can start with

a compatibility threshold of 3.0.

8. **Species Threshold:** Set the species threshold to a value that is appropriate for our task and network architecture. For example, we can start with a species threshold of 2.
9. **Max Stagnation:** Set the maximum number of generations that a species can go without improving its fitness score to a value that is appropriate for our task. For example, we can start with a value of 15.
10. **Node Activation:** Set the activation function used by the nodes in the neural network to a value that is appropriate for our task and network architecture. For example, we can use the sigmoid activation function.

### 3.2.14 Use Case

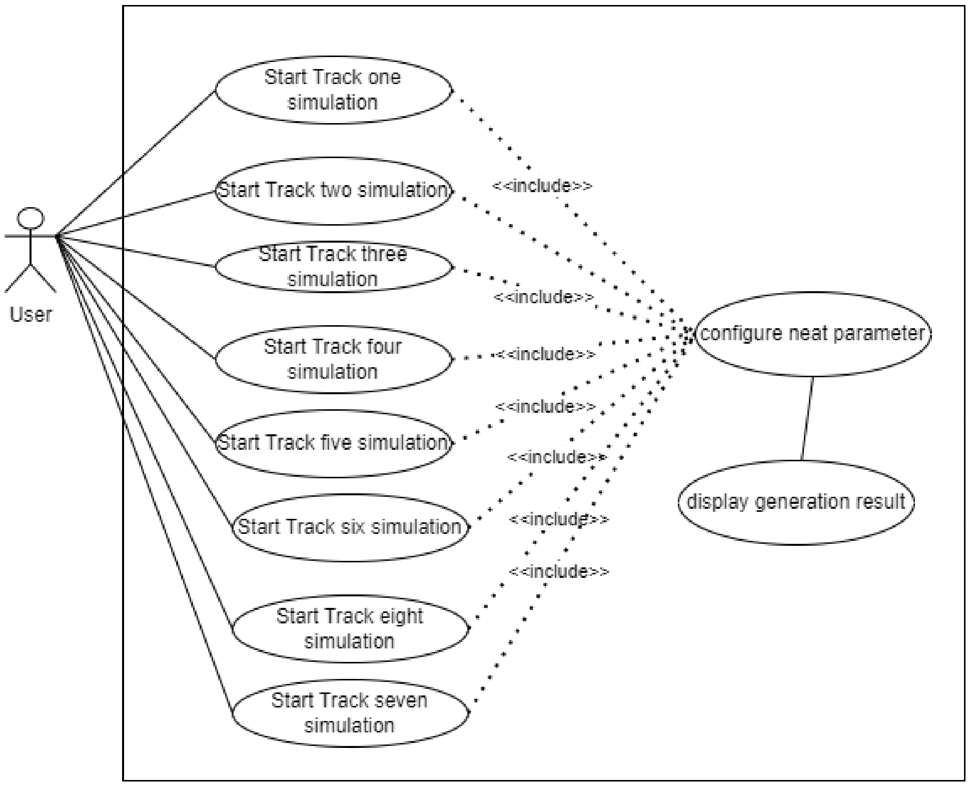

User
Start Track one simulation
Start Track two simulation
Start Track three simulation
Start Track four simulation
Start Track five simulation
Start Track six simulation
Start Track eight simulation
Start Track seven simulation
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
configure neat parameter
display generation result

Figure 3.9: Track Eight

### 3.2.15 Increment Model

The incremental model is a software development technique that interactively integrates the parts of the waterfall model. It entails both the creation and the upkeep of a system. The needs are divided into many components in this model. Analysis, design, implementation, testing/verification, and maintenance are all stages of incremental development. Every iteration goes through the steps of requirements, design, coding, and testing. The initial increment is frequently a core product that addresses the essential criteria, with additional features added in subsequent increments. The client receives the primary product. Following the client's analysis of the core product, the strategy for the next increment is developed. A Successive version model is another name for this concept. The incremental model is similar to the waterfall model in that it iteratively incorporates the features of the waterfall model. In each increment, the waterfall model is used again and again. In addition, the incremental approach uses linear sequences in a certain manner.

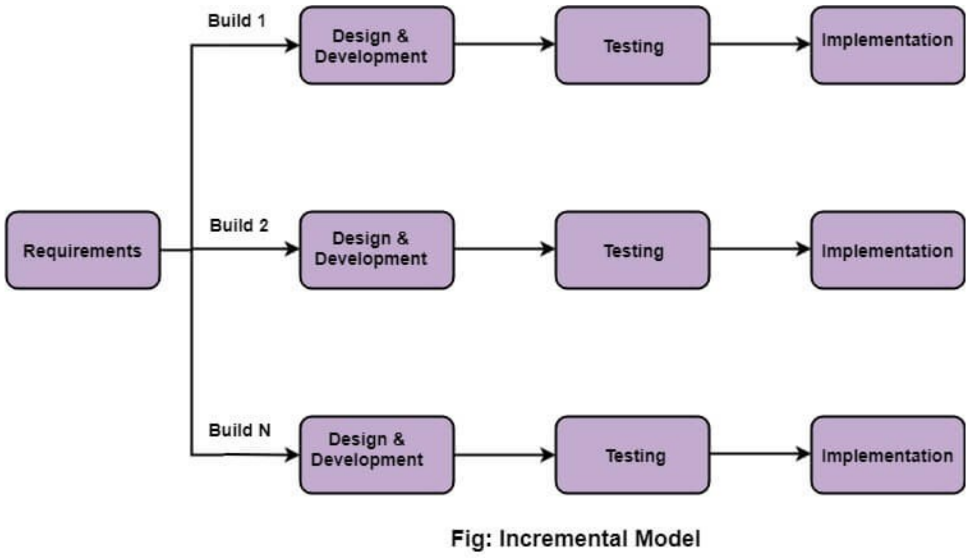


Figure 3.10: Incremental Model

### 3.2.16 Custom Tracks

Following shown are the different types of tracks we have used for out testing environment. Each of the following tracks is different in one way or another. Some of the tracks have a logic of taking decision only after obstacle detection. Tracks includes cars/trucks as the obstacle within the road. Green color code is set to be the defining collision parameter for the agent and through out the training process, the agent try to avoid driving over the greens color code.

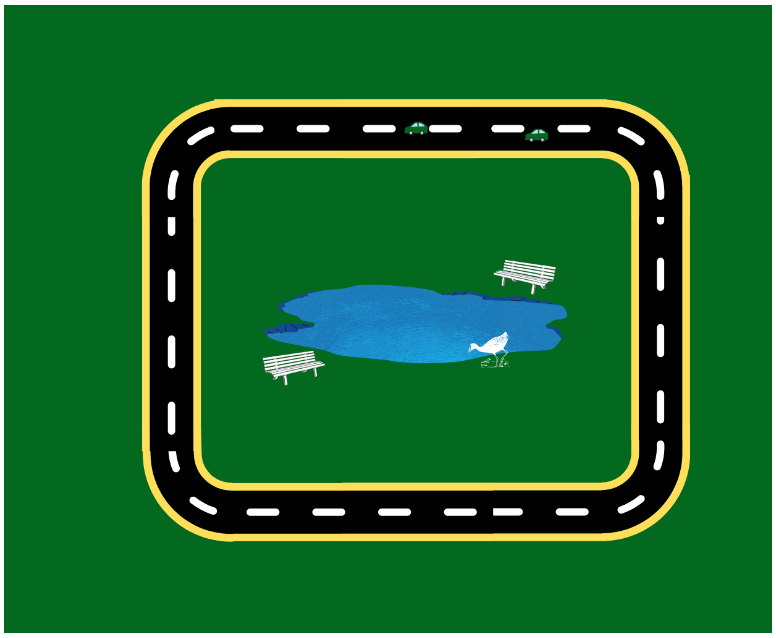


Figure 3.11: Track One

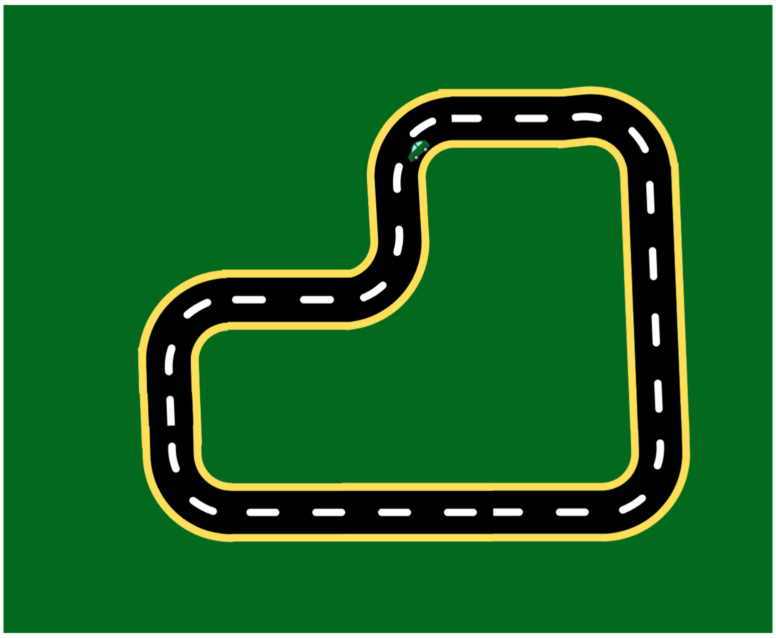


Figure 3.12: Track Two

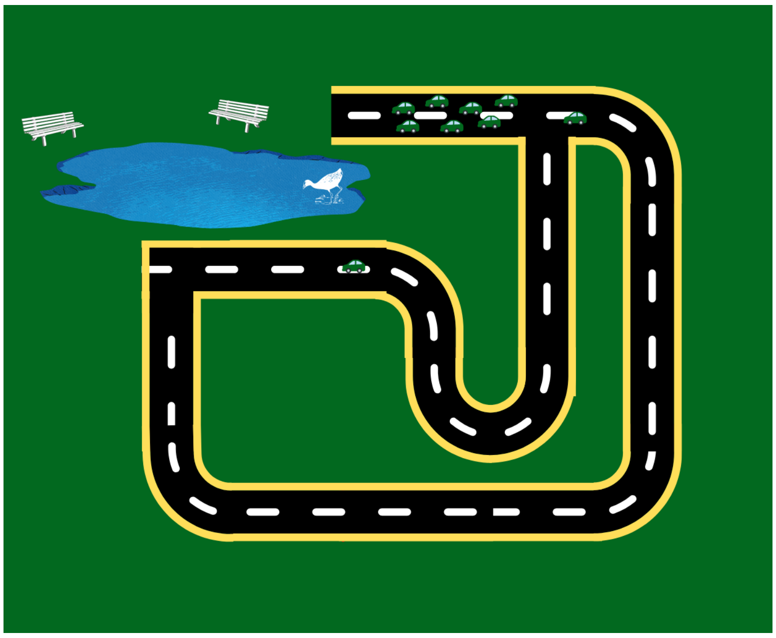


Figure 3.13: Track Three

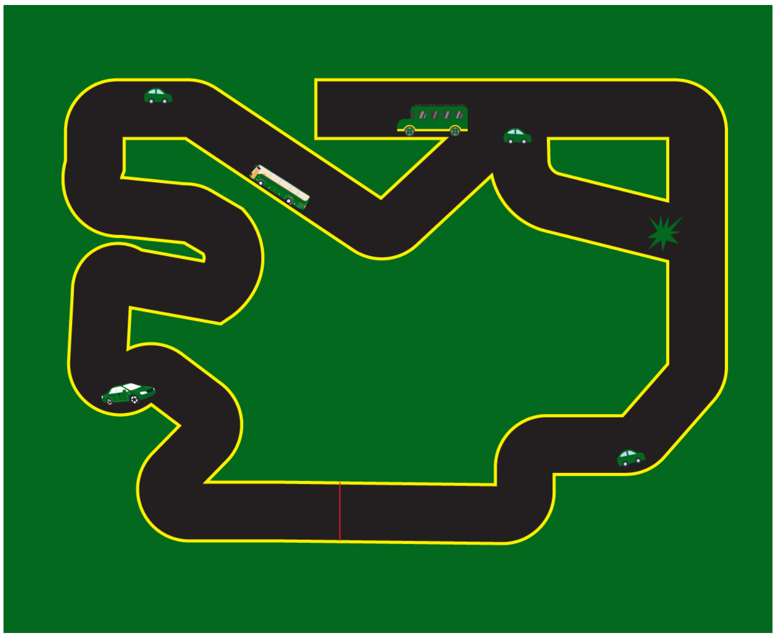


Figure 3.14: Track Four

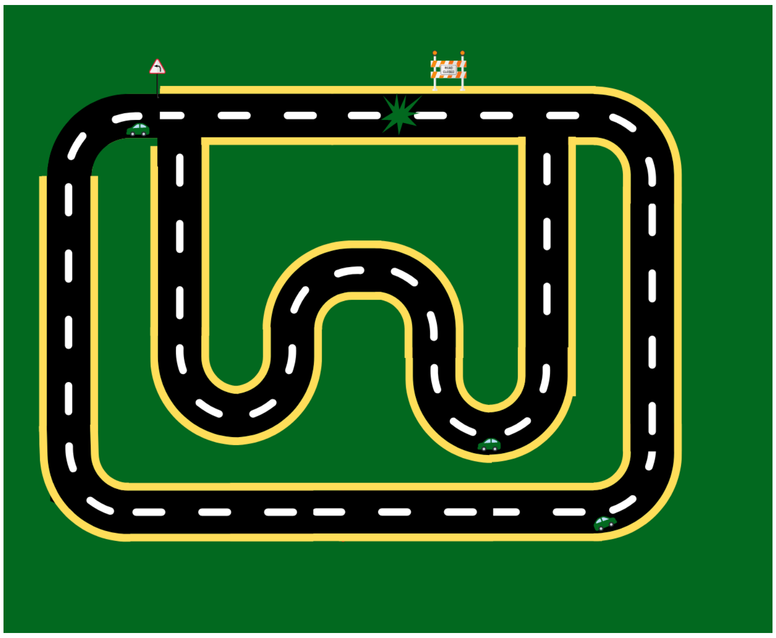


Figure 3.15: Track Five

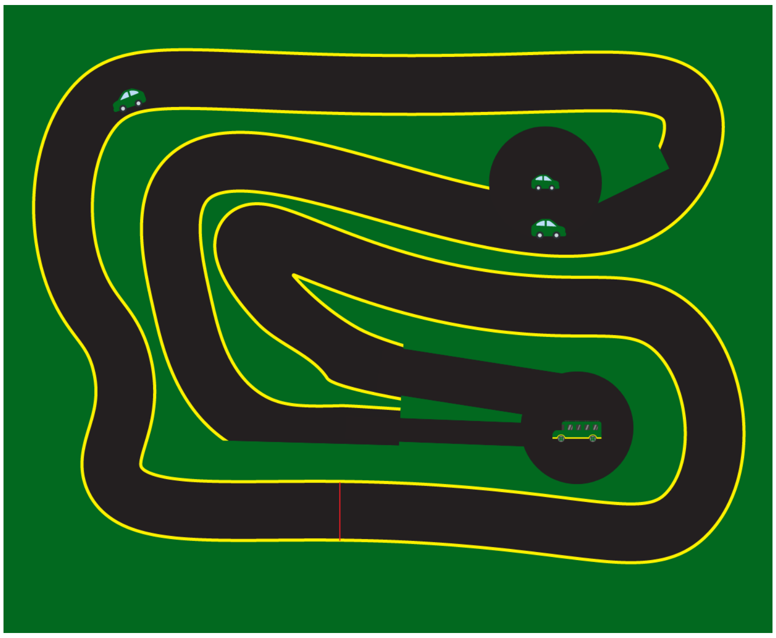


Figure 3.16: Track Six

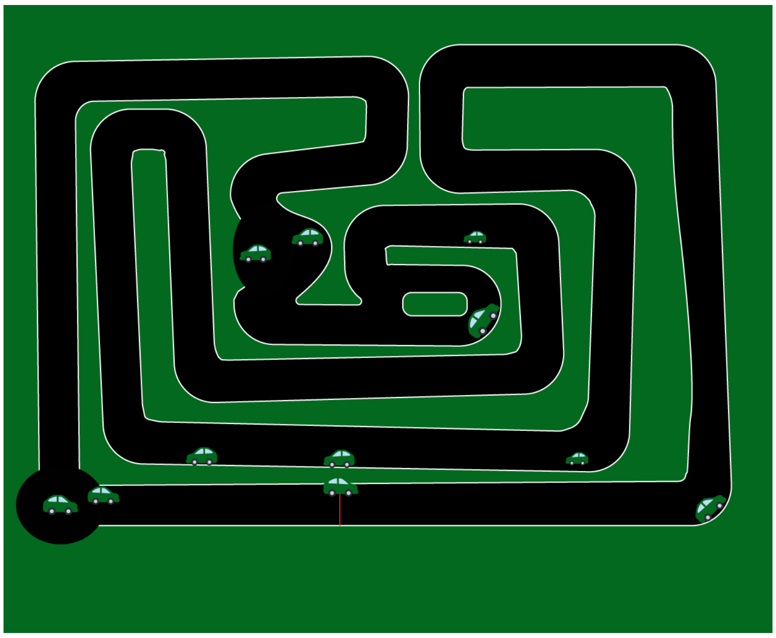


Figure 3.17: Track Seven

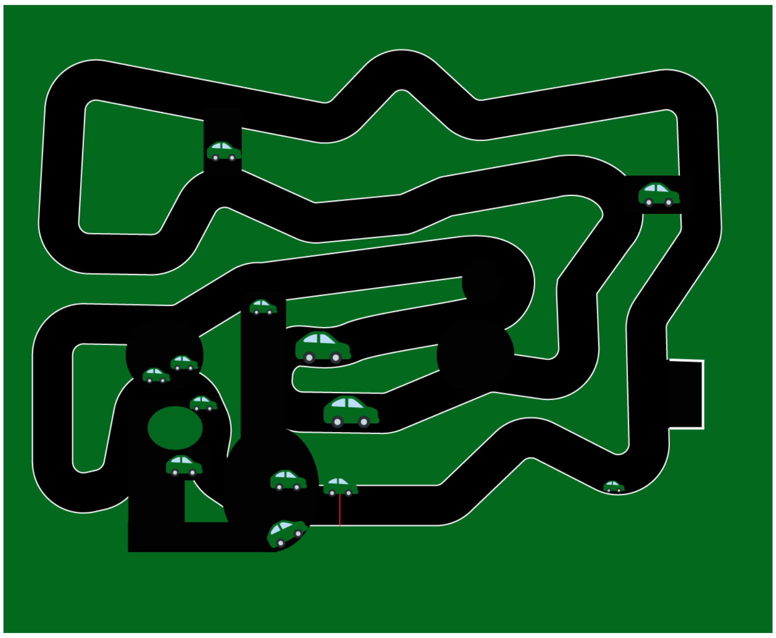


Figure 3.18: Track Eight

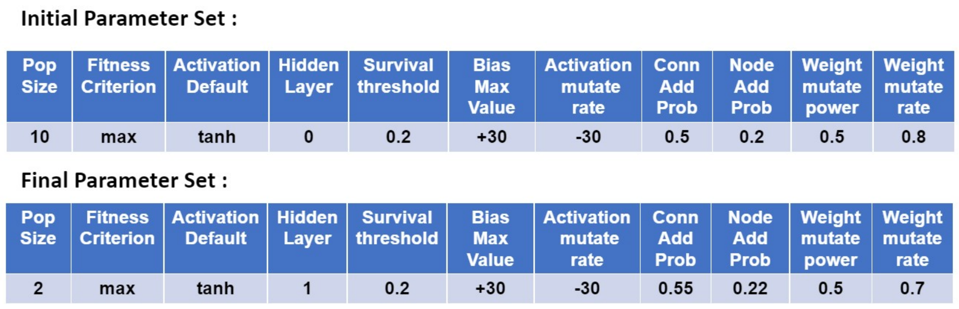
# CHAPTER 4
# EPILOGUE

## 4.1 Result And Findings

Following mention are the results and graphs of our finding that was obtained by training the agent into eight different custom environment.

### 4.1.1 Track One

Following are the findings obtained from the track one. Following information shows the changes made in the neat parameter that helped us to obtained the desired generation and smooth functioning of the simulation.



**Initial Parameter Set :**

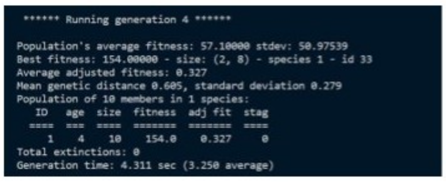
| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 10 | max | tanh | 0 | 0.2 | +30 | -30 | 0.5 | 0.2 | 0.5 | 0.8 |

**Final Parameter Set :**

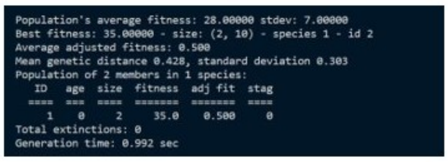
| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | max | tanh | 1 | 0.2 | +30 | -30 | 0.55 | 0.22 | 0.5 | 0.7 |

Following information shows the final output made after the change in neat parameter. The generation improvement was made by changing small aspect of the algorithm and number of generation was improved from 4 to 1 which displays neat evolution simulation.

1. Initial Generation : 4
2. Final Generation : 1



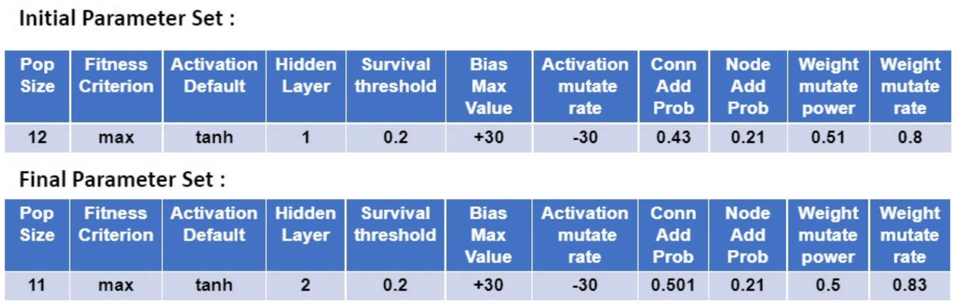
```
 ****** Running generation 4 ******

Population's average fitness: 57.10000 stdev: 50.97539
Best fitness: 154.00000 - size: (2, 8) - species 1 - id 33
Average adjusted fitness: 0.327
Mean genetic distance 0.605, standard deviation 0.279
Population of 10 members in 1 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    4    10    154.0    0.327     0
Total extinctions: 0
Generation time: 4.311 sec (3.250 average)
```



```
Population's average fitness: 28.00000 stdev: 7.00000
Best fitness: 35.00000 - size: (2, 10) - species 1 - id 2
Average adjusted fitness: 0.500
Mean genetic distance 0.428, standard deviation 0.303
Population of 2 members in 1 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    0     2     35.0    0.500     0
Total extinctions: 0
Generation time: 0.992 sec
```

### 4.1.2 Track Two

Following are the findings obtained from the track two. Following information shows the changes made in the neat parameter that helped us to obtained the desired generation and smooth functioning of the simulation.


Initial Parameter Set :

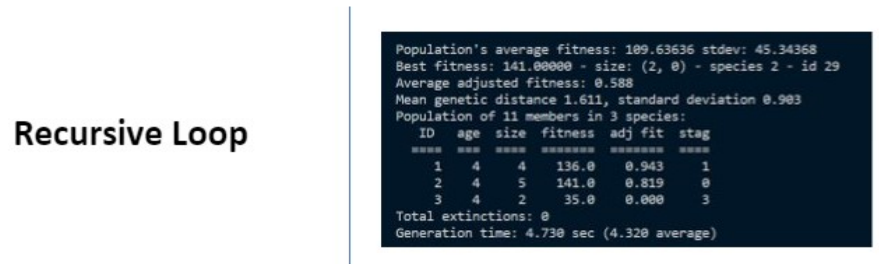
| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 12 | max | tanh | 1 | 0.2 | +30 | -30 | 0.43 | 0.21 | 0.51 | 0.8 |

Final Parameter Set :

| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 11 | max | tanh | 2 | 0.2 | +30 | -30 | 0.501 | 0.21 | 0.5 | 0.83 |

Following information shows the final output made after the change in neat parameter. The generation improvement was made by changing small aspect of the algorithm and number of generation was improved from recursive loop to 2 which displays neat evolution simulation.

1. Initial Generation : Recursive Loop
2. Final Generation : 2

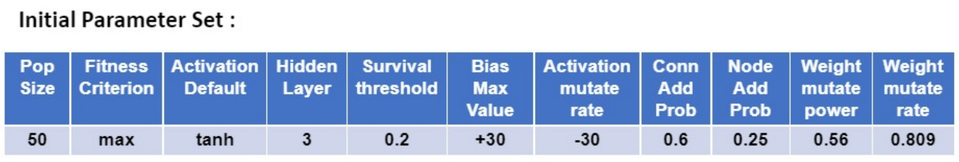


```
Population's average fitness: 109.63636 stdev: 45.34368
Best fitness: 141.00000 - size: (2, 0) - species 2 - id 29
Average adjusted fitness: 0.588
Mean genetic distance 1.611, standard deviation 0.903
Population of 11 members in 3 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    4     4    136.0    0.943     1
     2    4     5    141.0    0.819     0
     3    4     2     35.0    0.000     3
Total extinctions: 0
Generation time: 4.730 sec (4.320 average)
```

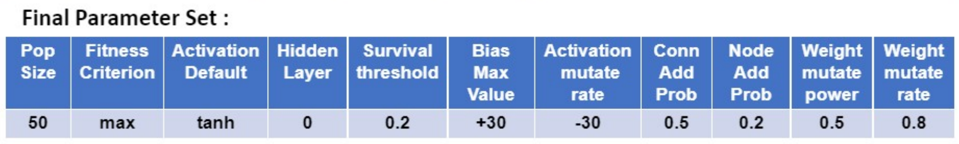
### 4.1.3 Track Three

Following are the findings obtained from the track three. Following information shows the changes made in the neat parameter that helped us to obtained the desired generation and smooth functioning of the simulation.

Following information shows the final output made after the change in neat parame-


Initial Parameter Set :

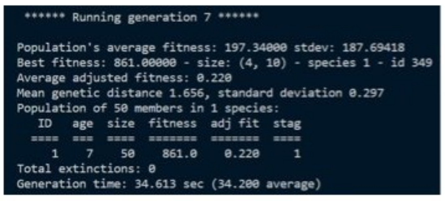
| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 50 | max | tanh | 3 | 0.2 | +30 | -30 | 0.6 | 0.25 | 0.56 | 0.809 |


Final Parameter Set :

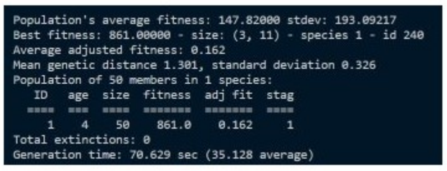
| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 50 | max | tanh | 0 | 0.2 | +30 | -30 | 0.5 | 0.2 | 0.5 | 0.8 |

ter. The generation improvement was made by changing small aspect of the algorithm and number of generation was improved from 23 to 7 which displays neat evolution simulation.

1. Initial Generation : 23
2. Final Generation : 7



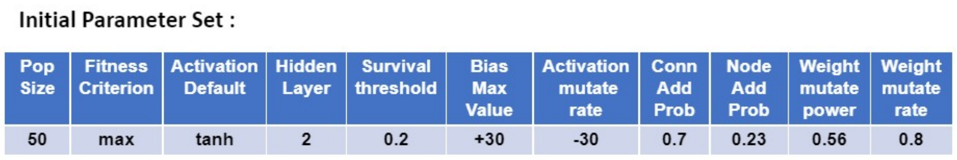
```
 ****** Running generation 7 ******

Population's average fitness: 197.34000 stdev: 187.69418
Best fitness: 861.00000 - size: (4, 10) - species 1 - id 349
Average adjusted fitness: 0.220
Mean genetic distance 1.656, standard deviation 0.297
Population of 50 members in 1 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    7    50    861.0    0.220     1
Total extinctions: 0
Generation time: 34.613 sec (34.200 average)
```



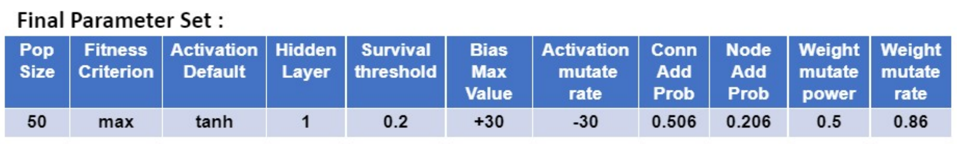
```
Population's average fitness: 147.82000 stdev: 193.09217
Best fitness: 861.00000 - size: (3, 11) - species 1 - id 240
Average adjusted fitness: 0.162
Mean genetic distance 1.301, standard deviation 0.326
Population of 50 members in 1 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    4    50    861.0    0.162     1
Total extinctions: 0
Generation time: 70.629 sec (35.128 average)
```

### 4.1.4 Track Four

Following are the findings obtained from the track four. Following information shows the changes made in the neat parameter that helped us to obtained the desired generation and smooth functioning of the simulation.


Initial Parameter Set :

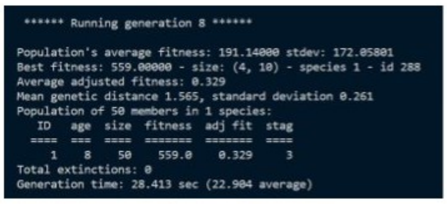
| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 50 | max | tanh | 2 | 0.2 | +30 | -30 | 0.7 | 0.23 | 0.56 | 0.8 |


Final Parameter Set :

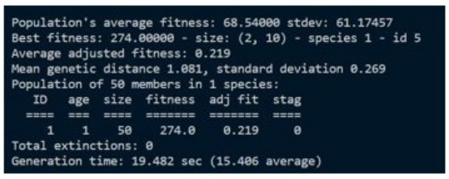
| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 50 | max | tanh | 1 | 0.2 | +30 | -30 | 0.506 | 0.206 | 0.5 | 0.86 |

Following information shows the final output made after the change in neat parameter. The generation improvement was made by changing small aspect of the algorithm and number of generation was improved from 22 to 9 which displays neat evolution simulation.

1. Initial Generation : 22
2. Final Generation : 9



```
 ****** Running generation 8 ******

Population's average fitness: 191.14000 stdev: 172.05801
Best fitness: 559.00000 - size: (4, 10) - species 1 - id 288
Average adjusted fitness: 0.329
Mean genetic distance 1.565, standard deviation 0.261
Population of 50 members in 1 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    8    50    559.0    0.329     3
Total extinctions: 0
Generation time: 28.413 sec (22.904 average)
```

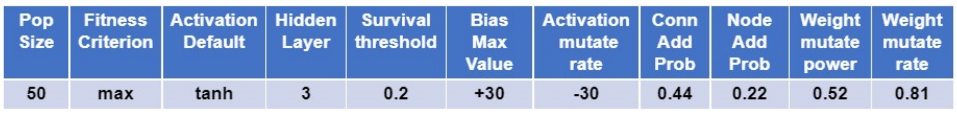


```
Population's average fitness: 68.54000 stdev: 61.17457
Best fitness: 274.00000 - size: (2, 10) - species 1 - id 5
Average adjusted fitness: 0.219
Mean genetic distance 1.081, standard deviation 0.269
Population of 50 members in 1 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    1    50    274.0    0.219     0
Total extinctions: 0
Generation time: 19.482 sec (15.406 average)
```

### 4.1.5 Track Five

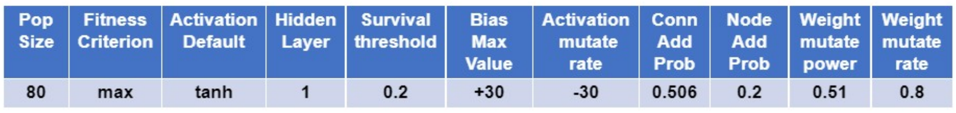
Following are the findings obtained from the track five. Following information shows the changes made in the neat parameter that helped us to obtained the desired generation and smooth functioning of the simulation.

Initial Parameter Set :



| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 50 | max | tanh | 3 | 0.2 | +30 | -30 | 0.44 | 0.22 | 0.52 | 0.81 |

Final Parameter Set :

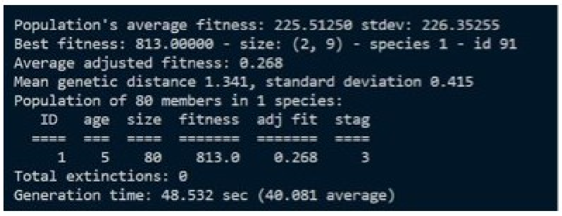


| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 80 | max | tanh | 1 | 0.2 | +30 | -30 | 0.506 | 0.2 | 0.51 | 0.8 |

Following information shows the final output made after the change in neat parameter. The generation improvement was made by changing small aspect of the algorithm and number of generation was improved from recursive loop to 14 which displays neat evolution simulation.

1. Initial Generation : Recursive loop
2. Final Generation : 14

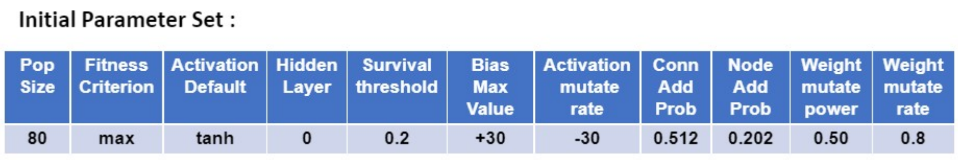
**Recursive Loop**



```
Population's average fitness: 225.51250 stdev: 226.35255
Best fitness: 813.00000 - size: (2, 9) - species 1 - id 91
Average adjusted fitness: 0.268
Mean genetic distance 1.341, standard deviation 0.415
Population of 80 members in 1 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    5    80    813.0    0.268     3
Total extinctions: 0
Generation time: 48.532 sec (40.081 average)
```

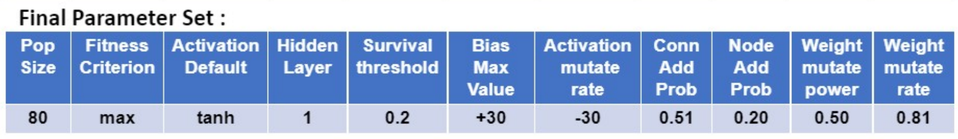
### 4.1.6 Track Six

Following are the findings obtained from the track six. Following information shows the changes made in the neat parameter that helped us to obtained the desired generation and smooth functioning of the simulation.



**Initial Parameter Set :**

| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 80 | max | tanh | 0 | 0.2 | +30 | -30 | 0.512 | 0.202 | 0.50 | 0.8 |

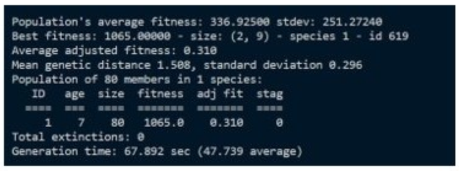


**Final Parameter Set :**

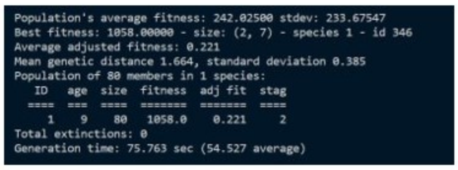
| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 80 | max | tanh | 1 | 0.2 | +30 | -30 | 0.51 | 0.20 | 0.50 | 0.81 |

Following information shows the final output made after the change in neat parameter. The generation improvement was made by changing small aspect of the algorithm and number of generation was improved from 31 to 16 which displays neat evolution simulation.

1. Initial Generation : 31
2. Final Generation : 16



```
Population's average fitness: 336.92500 stdev: 251.27240
Best fitness: 1065.00000 - size: (2, 9) - species 1 - id 619
Average adjusted fitness: 0.310
Mean genetic distance 1.508, standard deviation 0.296
Population of 80 members in 1 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    7    80   1065.0    0.310     0
Total extinctions: 0
Generation time: 67.892 sec (47.739 average)
```



```
Population's average fitness: 242.02500 stdev: 233.67547
Best fitness: 1058.00000 - size: (2, 7) - species 1 - id 346
Average adjusted fitness: 0.221
Mean genetic distance 1.664, standard deviation 0.385
Population of 80 members in 1 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    9    80   1058.0    0.221     2
Total extinctions: 0
Generation time: 75.763 sec (54.527 average)
```

### 4.1.7 Track Seven

Following are the findings obtained from the track seven. Following information shows the changes made in the neat parameter that helped us to obtained the desired generation and smooth functioning of the simulation.

**Initial Parameter Set :**

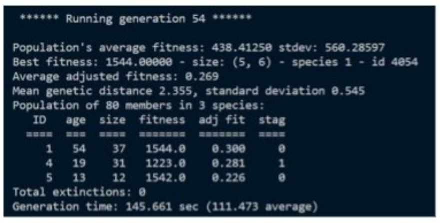
| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 80 | max | tanh | 3 | 0.2 | +30 | -30 | 0.562 | 0.206 | 0.501 | 0.801 |

**Final Parameter Set :**

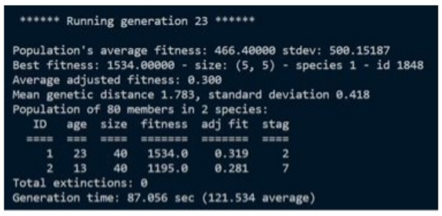
| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 80 | max | tanh | 0 | 0.2 | +30 | -30 | 0.51 | 0.21 | 0.51 | 0.81 |

Following information shows the final output made after the change in neat parameter. The generation improvement was made by changing small aspect of the algorithm and number of generation was improved from 54 to 23 which displays neat evolution simulation.

1. Initial Generation : 54
2. Final Generation : 23



```
 ****** Running generation 54 ******

Population's average fitness: 438.41250 stdev: 560.28597
Best fitness: 1544.00000 - size: (5, 6) - species 1 - id 4054
Average adjusted fitness: 0.269
Mean genetic distance 2.355, standard deviation 0.545
Population of 80 members in 3 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1   54    37   1544.0    0.300     0
     4   19    31   1223.0    0.281     1
     5   13    12   1542.0    0.226     0
Total extinctions: 0
Generation time: 145.661 sec (111.473 average)
```

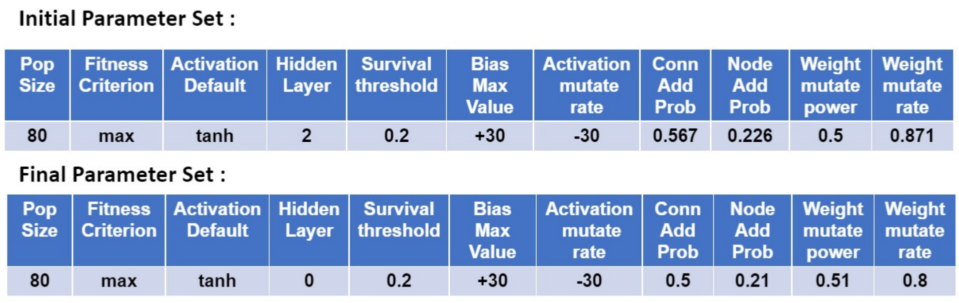


```
 ****** Running generation 23 ******

Population's average fitness: 466.40000 stdev: 500.15187
Best fitness: 1534.00000 - size: (5, 5) - species 1 - id 1848
Average adjusted fitness: 0.300
Mean genetic distance 1.783, standard deviation 0.418
Population of 80 members in 2 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1   23    40   1534.0    0.319     2
     2   13    40   1195.0    0.281     7
Total extinctions: 0
Generation time: 87.056 sec (121.534 average)
```

### 4.1.8 Track Eight

Following are the findings obtained from the track eight. Following information shows the changes made in the neat parameter that helped us to obtained the desired generation and smooth functioning of the simulation.

Following information shows the final output made after the change in neat parameter. The generation improvement was made by changing small aspect of the algorithm



Initial Parameter Set :

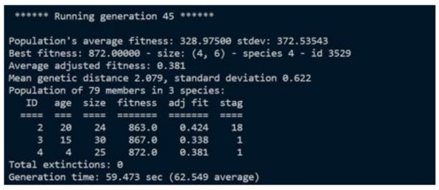
| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 80 | max | tanh | 2 | 0.2 | +30 | -30 | 0.567 | 0.226 | 0.5 | 0.871 |

Final Parameter Set :

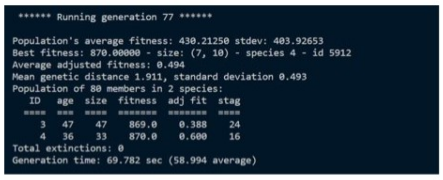
| Pop Size | Fitness Criterion | Activation Default | Hidden Layer | Survival threshold | Bias Max Value | Activation mutate rate | Conn Add Prob | Node Add Prob | Weight mutate power | Weight mutate rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 80 | max | tanh | 0 | 0.2 | +30 | -30 | 0.5 | 0.21 | 0.51 | 0.8 |

and number of generation was improved from 77 to 45 which displays neat evolution simulation.

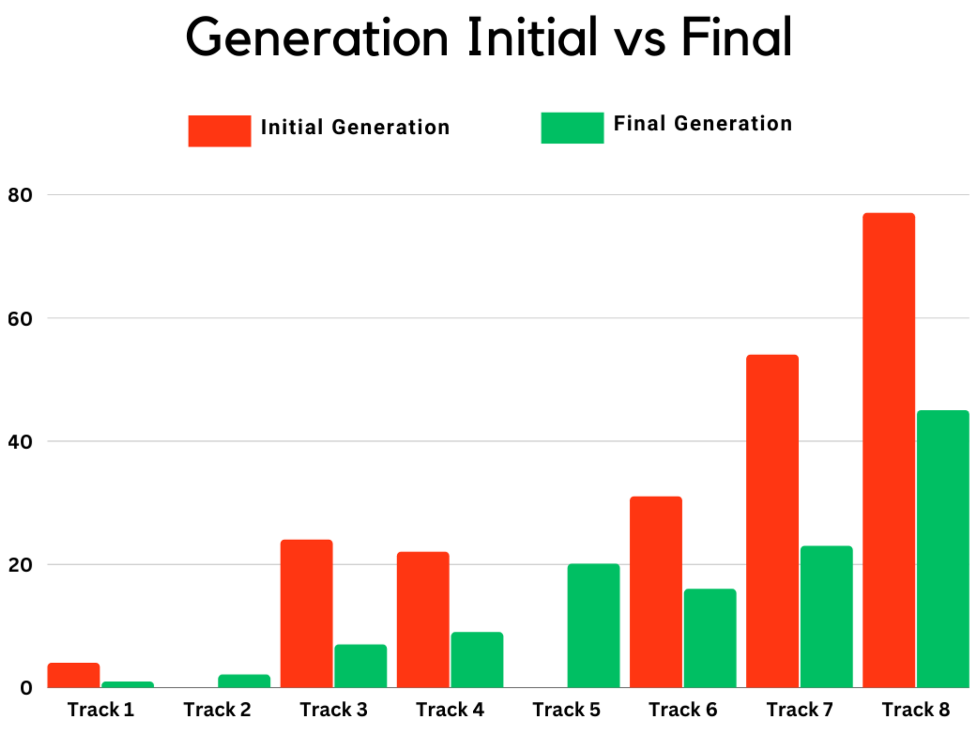
1. Initial Generation : 77
2. Final Generation : 45

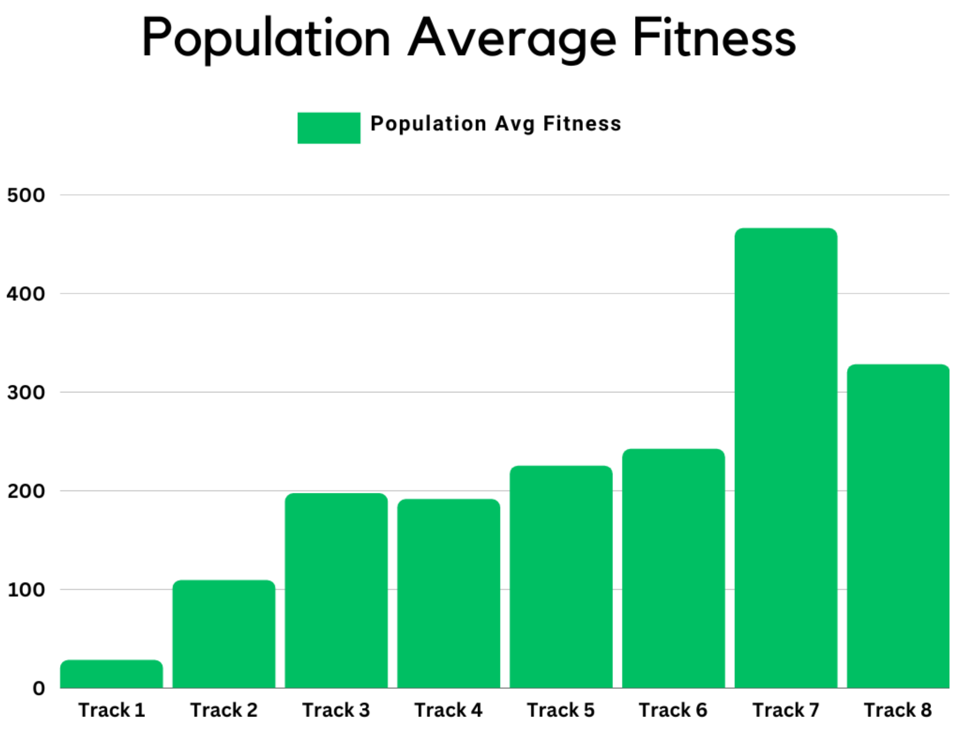


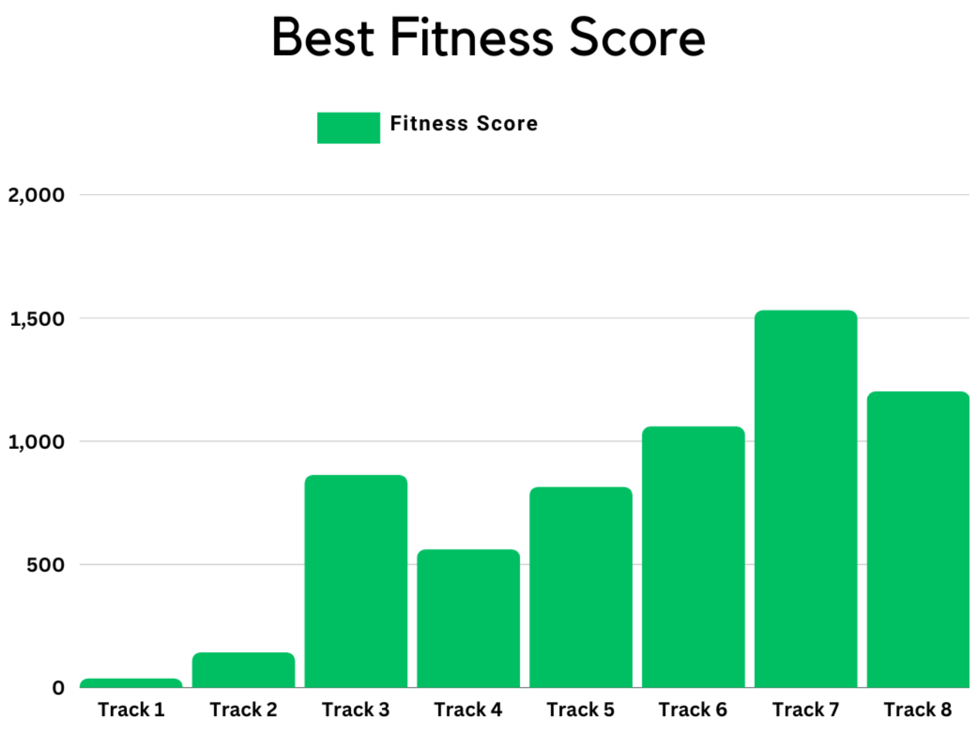
```
 ****** Running generation 45 ******

Population's average fitness: 328.97500 stdev: 372.53543
Best fitness: 872.00000 - size: (4, 6) - species 4 - id 3529
Average adjusted fitness: 0.381
Mean genetic distance 2.079, standard deviation 0.622
Population of 79 members in 3 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     2   20    24    863.0    0.424    18
     3   15    30    867.0    0.338     1
     4    4    25    872.0    0.381     1
Total extinctions: 0
Generation time: 59.473 sec (62.549 average)
```

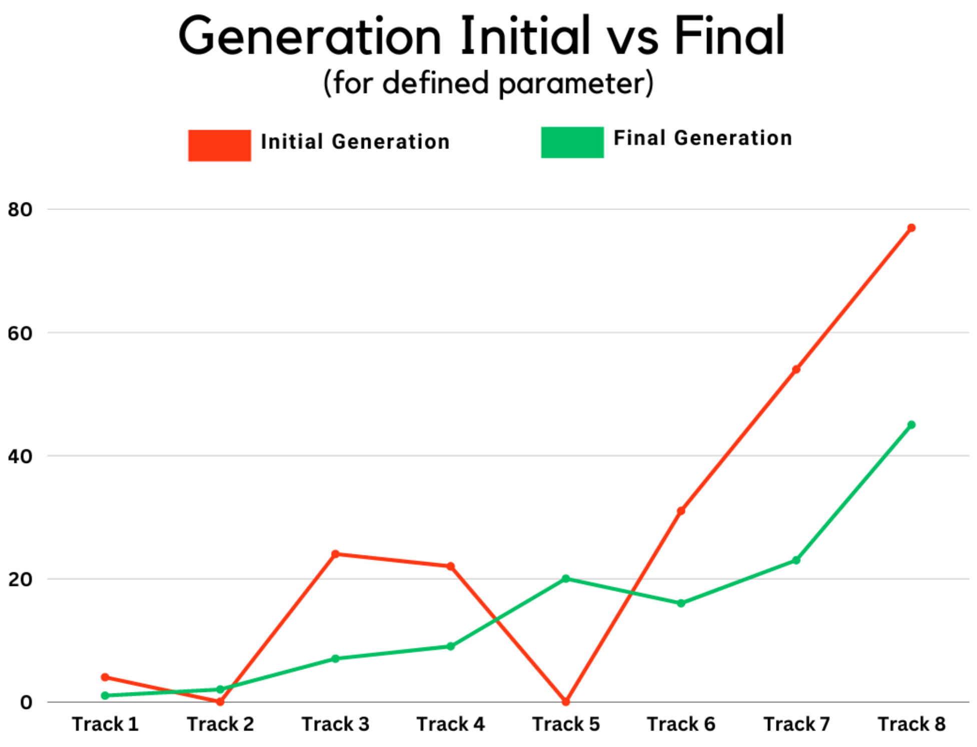


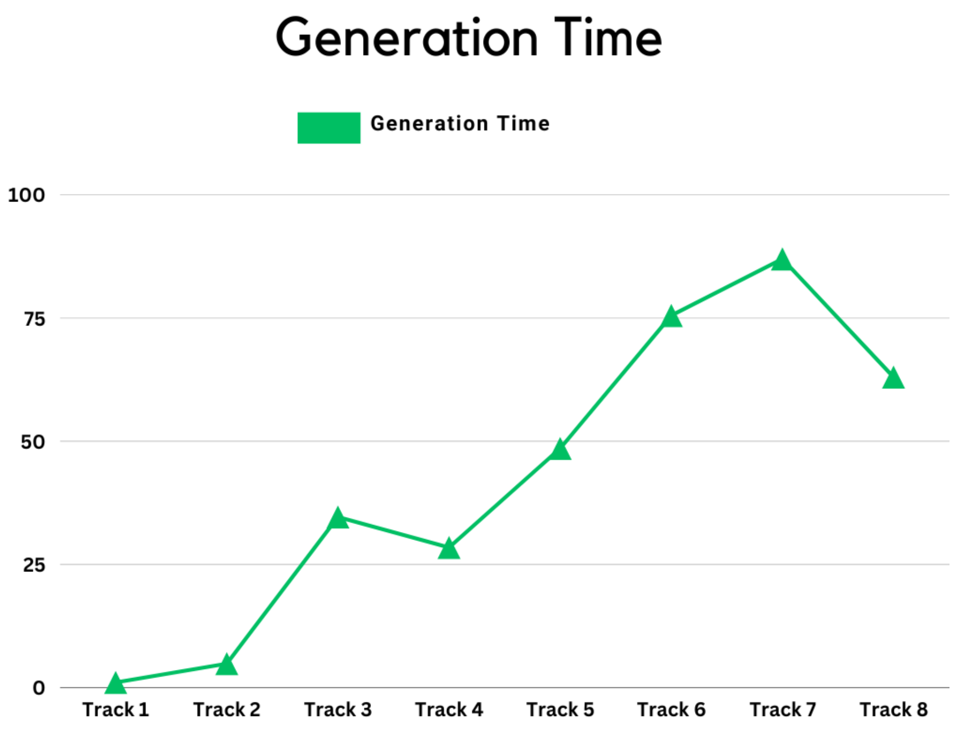
```
 ****** Running generation 77 ******

Population's average fitness: 430.21250 stdev: 403.92653
Best fitness: 870.00000 - size: (7, 10) - species 4 - id 5912
Average adjusted fitness: 0.494
Mean genetic distance 1.911, standard deviation 0.493
Population of 80 members in 2 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     3   47    47    869.0    0.388    24
     4   36    33    870.0    0.600    16
Total extinctions: 0
Generation time: 69.782 sec (58.994 average)
```

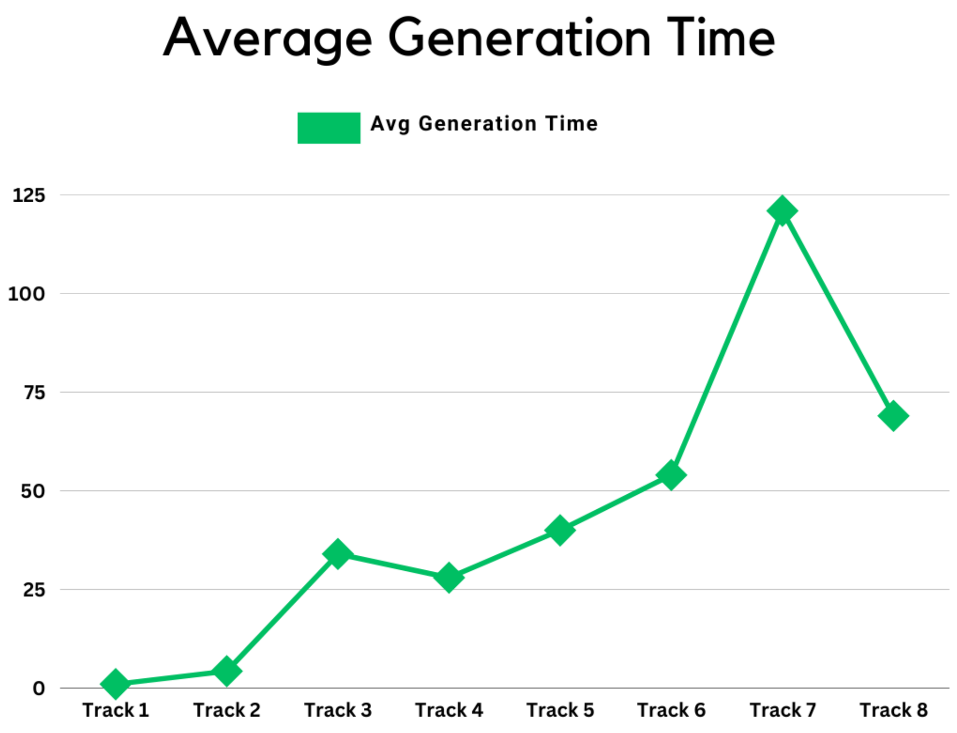

Generation Initial vs Final
Initial Generation
Final Generation
80
60
40
20
0
Track 1
Track 2
Track 3
Track 4
Track 5
Track 6
Track 7
Track 8

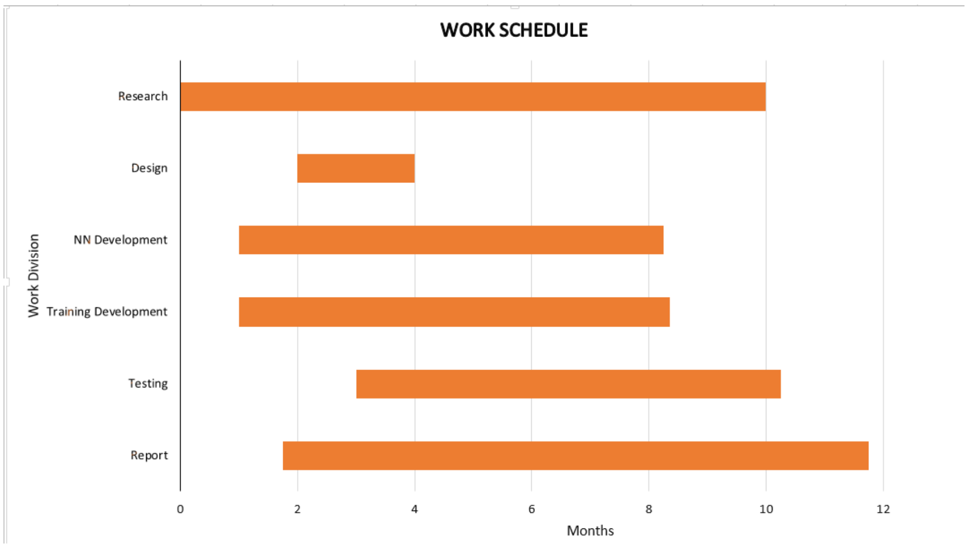

Population Average Fitness
Population Avg Fitness
500
400
300
200
100
0
Track 1
Track 2
Track 3
Track 4
Track 5
Track 6
Track 7
Track 8


Best Fitness Score
Fitness Score
2,000
1,500
1,000
500
0
Track 1
Track 2
Track 3
Track 4
Track 5
Track 6
Track 7
Track 8


Generation Initial vs Final
(for defined parameter)
Initial Generation
Final Generation
80
60
40
20
0
Track 1
Track 2
Track 3
Track 4
Track 5
Track 6
Track 7
Track 8


Generation Time
Generation Time
100
75
50
25
0
Track 1
Track 2
Track 3
Track 4
Track 5
Track 6
Track 7
Track 8


Average Generation Time
Avg Generation Time
125
100
75
50
25
0
Track 1
Track 2
Track 3
Track 4
Track 5
Track 6
Track 7
Track 8


WORK SCHEDULE
Work Division
Research
Design
NN Development
Training Development
Testing
Report
0
2
4
6
8
10
12
Months

# REFERENCES

[1] K. O. Stanley and R. Miikkulainen, ”Evolving neural networks through augmenting topologies,” Evolutionary Computation, vol. 10, no. 2, pp. 99-127, 2002.”

[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore, ”Reinforcement learning: a survey,” Journal of Artificial Intelligence Research, vol. 4, pp. 237-285, 1996.

[3] K. Chitta, A. Prakash, and A. Geiger, “Neat: Neural attention fields for end-to-end autonomous driving,” in Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), October 2021, pp. 15 793–15 803.

[4] Safe. scalable. A new paradigm. Waabi. (n.d.). Retrieved January 29, 2023, from https://waabi.ai/

[5] Lutkevich, B. (2023, January 23). self-driving car (autonomous car or driverless car). Enterprise AI. https://www.techtarget.com/searchenterpriseai/definition/driverless-car

[6] NEAT-Python’s documentation! — NEAT-Python 0.92 documentation. (n.d.). https://neat-python.readthedocs.io/en/latest/