



**NEW HORIZON
COLLEGE OF ENGINEERING**

Department of Artificial Intelligence & Machine Learning

Academic Year 2022-23(ODD)

Report for

Mini Project-I (21AIL38A)

On

“SIMULATION OF A SELF DRIVING CAR USING

REINFORCEMENT LEARNING”

By

Aatrey Kiran 1NH21AI001

Adetth Raju 1NH21AI007

C Srikar Reddy 1NH21AI018

Under the Guidance of

Dr. Prianka R R

Senior Assistant Professor,

Dept. of Artificial Intelligence & Machine Learning,

New Horizon College of Engineering,

Bangalore-560103



NEW HORIZON COLLEGE OF ENGINEERING

Department of Artificial Intelligence & Machine Learning

CERTIFICATE

Certified that the Mini Project- III with the subject code 21AIL38A work entitled **“SIMULATION OF A SELF DRIVING CAR USING REINFORCEMENT LEARNING”** carried out by Adetth Raju, Aatrey Kiran, C Srikar Reddy, USN: 1NH21AI007, 1NH21AI001, 1NH21AI018. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of Mini Project work.

Dr. Prianka R R

Internal Guide

Dr. N V Uma Reddy

Head of Department

External Viva :

EXAMINER

SIGNATURE WITH DATE

1.

2.

ABSTRACT

Autonomous driving in AI controlled vehicles has been implemented in the current day to day by many highly reputed car companies for example Tesla, Porsche, Ford, Volkswagen, Audi, and so on. Our goal in this research paper is to obtain the simulation of a vehicle in a user-built track without any collisions to the side of the track. This is being implemented using Reinforcement learning and Neural Evolution of Augmenting Topologies or NEAT for short. Deep learning is also implemented in this simulation to help the candidates perform in the next generations.

TABLE OF CONTENTS

SL.NO	CHAPTER	PAGE NUMBER
1	Introduction	
	1.1. Introduction	6
	1.2. Related Work	6
	1.3. Objective	7
	1.4. Literature Survey	7
	1.5. Proposed System	10
2	System Requirements	
	2.1. Hardware Requirements	12
	2.2. Software Requirements	12
3	System Design	
	3.1. System Architecture	13
	3.2. Modules	14
4	Implementation	
	4.1. Code Snippet	16
	4.2. Code	17
	4.3. Results	23
5	Conclusion	
	5.1. Conclusion	24
	5.2. Future Enhancement	24

TABLE OF FIGURES

SL.NO	FIGURE NAME	PAGE NUMBER
3.1	System Design Diagram	8
4.1	Code Snippet	10
4.2	Car under simulation	17

CHAPTER 1

1.1. INTRODUCTION

As far as means of transportation go, travelling by vehicles, or cars rather have been the staple of transportation across the globe. It might not be the safest of all the possible available transport methods but vehicular transportation is the cheapest, readily available and extremely convenient. To add another point on the positive pool of compliments this mean of transportation can also be implemented as a means of recreation, be it road trips or just long drives in general.

Here in this paper, we aim to create a 2D model of a vehicle and to put it in a test with many other similar vehicles and to successfully make them simulate efficient and fast transportation from one point to the other while being the safest by avoiding collisions with obstacles and other vehicles.

This is done by implementing neural networks and allowing the candidates(vehicles) to drive randomly while being rewarded/ penalized as they move around the test field and they learn to do the tasks that reward them.

1.2. RELATED WORK

The NEAT algorithm is based on the idea of using a genetic algorithm to evolve the topology of neural networks. The algorithm starts with a population of randomly generated neural networks, each with a small number of nodes and connections. These networks are then evaluated based on their performance on a given task (e.g. : a game or a control problem).

1)The Neuro-Evolution is the basis of this algorithm, which is the Reinforcement Learning part of the system. This Neuro-Evolution refers to the nodes (Neurons) in the “brain” of the system and whatever it learns, while the Evolution is when a Parent that is the best of the previous generation is taken and children are derived from this parent candidate.

2)The Augmented Topologies system is simply put, hidden traits like a new node in the layers of the given neural network or a new connection from one node in a certain layer to another node in a different layer of the network.

In each generation, the most fit networks are selected to breed, creating new networks with the same topology but with slight variations in their weights. In addition, the algorithm allows for the mutation of the topology of the networks, including the addition or deletion of nodes and connections. This allows for the exploration of new network structures that may perform better on the given task.

1.3. OBJECTIVE

The objective of this paper is to simulate the behaviour of a self-driving car using computer models and algorithms. The simulation will focus on replicating the car's perception, decision making, and control systems in a virtual environment. The goal is to understand the performance and limitations of the self-driving car technology, as well as to identify the working of the NEAT algorithm and to have a deeper foundation on the machine learning model. Additionally, simulation will be run in different conditions like different road models.

1.4. LITERATURE SURVEY

- “Neuro Evolution of Augmented Topologies” by Kenneth O. Stanley and Risto Miikkulainen [2002]

In their study they found that the evolved controllers were able to perform well in the game, and that the NEAT algorithm was able to efficiently evolve the controllers' topologies and weights.

Advantages:

1. The NEAT algorithm was designed to be easy to understand and implement, which made it accessible to a wide range of researchers and developers.
2. The NEAT algorithm was highly adaptable and could be used to evolve neural networks for a variety of tasks.

Disadvantages:

1. While the NEAT algorithm was highly adaptable, it may not be the best approach for all neural network problems.
2. Despite the NEAT algorithm's apparent simplicity, it is still a relatively complex system that requires a significant number of computational resources to implement and optimize.
 - "A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks." by Kenneth O Stanley, David B D'Ambrosio and Jason Gauci [2009]

The paper presents an extension to the NEAT (Neuro Evolution of Augmenting Topologies) algorithm that allows it to evolve large-scale neural networks. The paper proposes a new encoding scheme for the NEAT algorithm called HyperNEAT, which represents the structure of a neural network using a high-dimensional hypercube. This encoding scheme enables the algorithm to exploit regularities in the problem domain to evolve large and complex neural networks more efficiently.

Advantages:

1. The paper proposed several modifications to the NEAT algorithm that significantly reduced the computational resources required to train neural networks. This made it more practical for use in real-world applications.
2. The paper introduced several new features to the NEAT algorithm, such as speciation control and fitness sharing, that increased its flexibility and made it more adaptable to a wider range of problems.

Disadvantages:

1. The paper did not compare the modified NEAT algorithm to other state-of-the-art neural network optimization techniques, which could provide a more complete picture of its relative performance.
2. The paper did not provide a detailed theoretical analysis of the modified NEAT algorithm, which could limit its ability to generalize to other domains or applications.

- “Evolving neural networks through augmenting topologies with a multi-objective approach” by W.H. van Willigen, Evert Haasdijk and L.K.H.M Kester [2013]

This paper proposes an extension to the NEAT algorithm that uses a multi-objective approach to optimize neural networks for both accuracy and complexity. The authors introduce a new fitness function and modified speciation method to enable the evolution of networks that balance accuracy and complexity more effectively.

Advantages:

1. The paper proposes a novel multi-objective approach to evolving neural networks that balance both accuracy and complexity, which is an important problem in many real-world applications.
2. The authors provide a detailed explanation of the proposed approach and its modifications to the NEAT algorithm, making it easy to replicate their work.

Disadvantages:

1. The paper lacks a thorough comparison with other state-of-the-art multi-objective neuroevolution methods, making it difficult to assess the relative performance of the proposed approach.
2. The paper does not provide a detailed analysis of the resulting neural networks, making it difficult to gain insights into how the proposed approach optimizes network architecture and connectivity.

- “Modularity in NEAT Reinforcement Learning Networks” by Humphrey Munn and Marcus Gallagher [2022]

This paper explores the concept of modularity in neural networks trained using the NEAT algorithm in the context of reinforcement learning, it proposes a modified version of the NEAT algorithm that encourages the development of modular neural networks.

Advantages:

1. Improved Performance: This paper demonstrated that their modified version of NEAT outperformed the original algorithm on several benchmark tasks
2. Expanded applicability: the effectiveness of the modified NEAT algorithm on a wider range of tasks, including image classification and reinforcement learning problems was demonstrated.

Disadvantages:

1. Lack of comparison: The paper did not compare the modified NEAT algorithm to other state-of-the-art neural network optimization techniques.
2. Lack of theoretical analysis: The paper did not provide a detailed theoretical analysis of the modified NEAT algorithm.

1.5. PROPOSED SYSTEM

In this proposed system, we aim to simulate the behaviour of a self-driving car using the NEAT (Neuro-Evolution of Augmenting Topologies) algorithm in a 2D environment. The simulation will focus on replicating the car's perception, decision making, and control systems in a virtual environment and evaluate its performance in different scenarios.

The proposed system will have following components:

Simulation Environment: A 2D simulation of a city or road network will be created, including the layout of roads, traffic rules, and obstacles. The environment will be designed to mimic the real-world conditions and provide the self-driving car with realistic sensory inputs.

Neural Network Controllers: An initial population of neural network controllers will be created, each with a small number of nodes and connections. These controllers will be used to control the behaviour of the self-driving car in the simulation.

NEAT Algorithm: The NEAT algorithm will be used to evolve the controllers. It will evaluate the performance of the controllers in the simulation environment, select the fittest controllers for breeding, and perform mutations to explore new network structures.

Performance Metrics: A set of predefined metrics will be used to evaluate the performance of the controllers in the simulation environment. These metrics will include, but not limited to, time to complete a route, collisions, adherence to traffic rules, etc.

Data Analysis: The results of the simulation will be analysed to understand the performance and limitations of the self-driving car technology, as well as to identify potential issues and challenges that need to be addressed before deployment on public roads.

User Interface: A user interface will be provided to the users to monitor the simulation. The interface will allow users to access the results of the simulation.

The proposed system will be implemented using programming languages such as Python and using libraries such as NEAT-python. The system will be able to simulate self-driving car's behaviour in a 2D environment, evaluate its performance and identify areas for improvement in the car's design and control algorithms

CHAPTER 2

2.1. HARDWARE REQUIREMENTS

The hardware requirements for a system varies as the program keeps getting updated, and depends on the operating system that is being used, but the basic requirements are:

Processor: A 1 GHz processor or higher

RAM: 1 GB or more

Storage: A hard drive with at least 100 MB of free space

2.2. SOFTWARE REQUIREMENTS

Python Interpreter: You need to have the Python interpreter installed on your computer.

Required Libraries: You may need to install additional libraries and packages. These can be installed using the **Python Package Manager (pip)**. For this program, the libraries needed are:

neat-python

Operating System: Python is compatible with all major operating systems

CHAPTER 3

3.1. SYSTEM ARCHITECTURE

Given below is the architecture diagram for the NEAT neural network architecture.

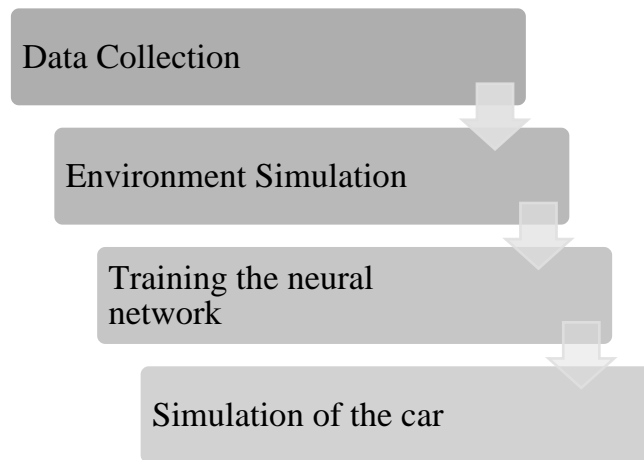


Fig 3.1: System Architecture

Data Collection: The data is collected from the sensors that are cast from the car and the calculation for the collision has been set so that when the car touches the edges of the black track the candidate dies, and the sensors also help the car to navigate through the track.

Environment Simulation: A 2D environment where the code is executed. This simulation environment can be modified before and after running the simulation and the change is immediate. The simulation environment can be edited with any kind of image editor program. In this simulation GIMP is used. GIMP stands for GNU Image Manipulation Program.

Training the neural network: The neural network starts off as a basic neural network with no hidden layers. This neural network then progressively gets more and more complex as more and more generations pass and the algorithm learns a more efficient way to solve the current problem, in this case the simulation of the car. More nodes are connected and new hidden layers are formed in the neural network.

Simulation of the car: The user sees the car running in the current track that is drawn on the map that is either predefined or which the user has created. This simulation contains data like the current generation, the genome data, the fitness size, the execution current time, etc.

These components are the basis of the simulation of the self-driving car that come together to provide us with a seamless experience with the running of the program. This program can be converted to an application executable format which can be run independent of a python interpreter or a compiler too.

3.2. MODULES

1. Simulation Environment:

The simulation environment used in this following program is a track of black colour that the car moves on.

This simulation environment can be modified before and after running the simulation and the change is immediate.

The simulation environment can be edited using any image editor program, in this case a third-party program called GIMP (gnu image manipulation program) is used.

2. Controllers:

A base of neural networks with no hidden layers is established.

As the program progresses new mutations are formed and new nodes are created and connected in the hidden layer.

The main nodes we need to focus on are the Input layer, i.e. the sensors of the car that detect the white of the background

And the output layers here is the speed and direction control, i.e., move left, move right , increase or decrease speed.

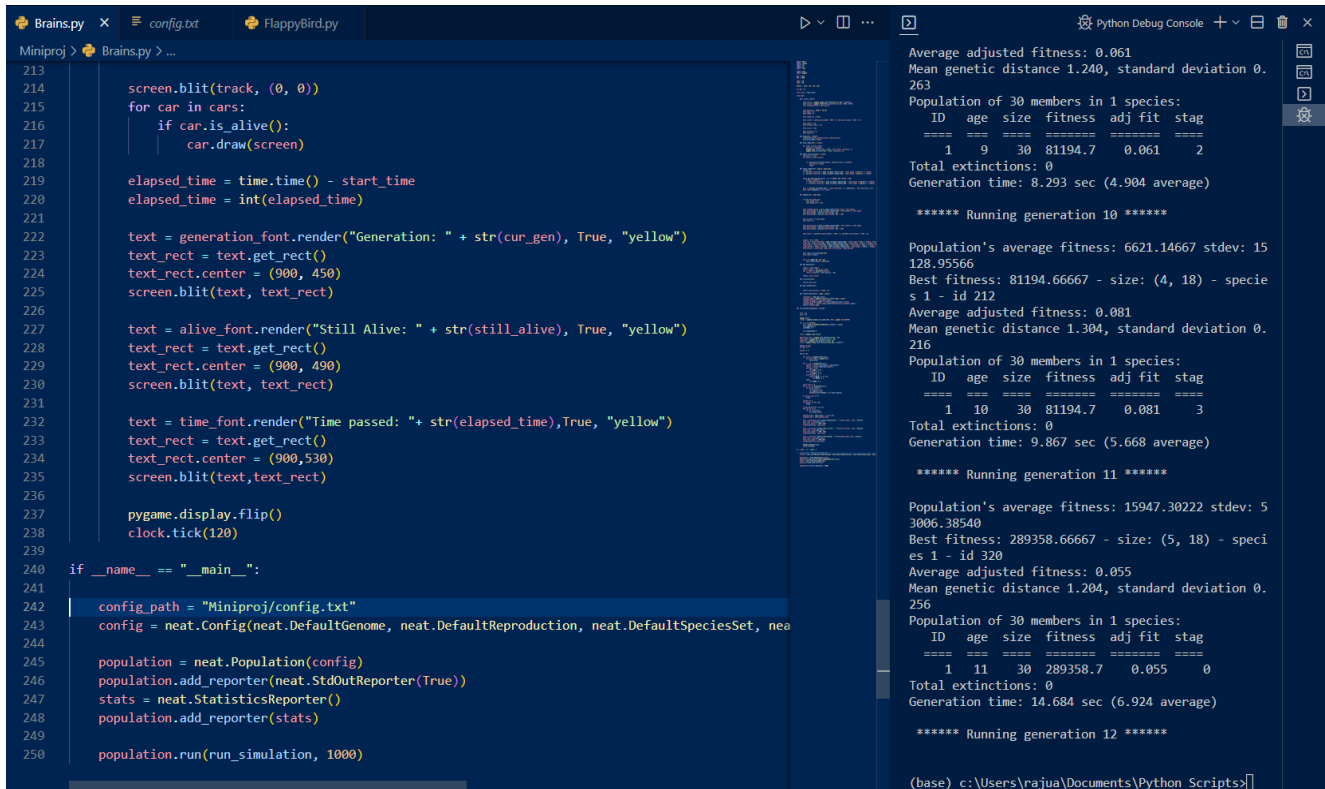
3. Metrics:

A set of metrics or constraints are added, as in the dampening of the genome using a damping function (in this case : \tanh), or other damping functions like sigmoid also can be used.

Another metric like data analysis for the standard deviation for the rate of learning for the neural network, or the fitness threshold where the program terminates.

CHAPTER 4

4.1. CODE SNIPPET



```
213
214     screen.blit(track, (0, 0))
215     for car in cars:
216         if car.is_alive():
217             car.draw(screen)
218
219     elapsed_time = time.time() - start_time
220     elapsed_time = int(elapsed_time)
221
222     text = generation_font.render("Generation: " + str(cur_gen), True, "yellow")
223     text_rect = text.get_rect()
224     text_rect.center = (900, 450)
225     screen.blit(text, text_rect)
226
227     text = alive_font.render("Still Alive: " + str(still_alive), True, "yellow")
228     text_rect = text.get_rect()
229     text_rect.center = (900, 490)
230     screen.blit(text, text_rect)
231
232     text = time_font.render("Time passed: " + str(elapsed_time), True, "yellow")
233     text_rect = text.get_rect()
234     text_rect.center = (900, 530)
235     screen.blit(text, text_rect)
236
237     pygame.display.flip()
238     clock.tick(120)
239
240 if __name__ == "__main__":
241
242     config_path = "Miniproj/config.txt"
243     config = neat.Config(neat.DefaultGenome, neat.DefaultReproduction, neat.DefaultSpeciesSet, neat.DefaultStagnation)
244
245     population = neat.Population(config)
246     population.add_reporter(neat.StdOutReporter(True))
247     stats = neat.StatisticsReporter()
248     population.add_reporter(stats)
249
250     population.run(run_simulation, 1000)
```

Average adjusted fitness: 0.061
Mean genetic distance 1.240, standard deviation 0.263
Population of 30 members in 1 species:

ID	age	size	fitness	adj fit	stag
1	9	30	81194.7	0.061	2

Total extinctions: 0
Generation time: 8.293 sec (4.904 average)

***** Running generation 10 *****

Population's average fitness: 6621.14667 stdev: 15128.95566
Best fitness: 81194.66667 - size: (4, 18) - species 1 - id 212
Average adjusted fitness: 0.081
Mean genetic distance 1.304, standard deviation 0.216
Population of 30 members in 1 species:

ID	age	size	fitness	adj fit	stag
1	10	30	81194.7	0.081	3

Total extinctions: 0
Generation time: 9.867 sec (5.668 average)

***** Running generation 11 *****

Population's average fitness: 15947.30222 stdev: 53006.38540
Best fitness: 289358.66667 - size: (5, 18) - species 1 - id 320
Average adjusted fitness: 0.055
Mean genetic distance 1.204, standard deviation 0.256
Population of 30 members in 1 species:

ID	age	size	fitness	adj fit	stag
1	11	30	289358.7	0.055	0

Total extinctions: 0
Generation time: 14.684 sec (6.924 average)

***** Running generation 12 *****

(base) c:\Users\rajua\Documents\Python Scripts>

Fig 4.1: Code Snippet

Here we can observe the function that draws the car on the screen and if the car is alive it continues to do so, this particular section of the code was chosen as the snippet since the “brains” of the code is housed in the last few lines, i.e. the NEAT algorithm that has been implemented has been used in the last “if” call.

This shows us how the NEAT module has been implemented and we see the file location of the configuration file as well as the default pre-set parameters defined in the module beforehand.

The last line calls the run function that executes the simulation for 1000 generations and the right side shows us the statistics of the data.

4.2. CODE

```
import math
import random
import sys
import os
import time
import NEAT
import pygame

WID = 1920
HEI = 1080

CARX = 60
CARY = 60

BODCOL = (255, 255, 255, 255)

cur_gen = 0
start_time = time.time()

class Car:
    def __init__(self):
        self.sprite = pygame.image.load('Miniproj/car.png').convert()
        self.sprite = pygame.transform.scale(self.sprite,
        (CARX, CARY))
        self.rotated_sprite = self.sprite

        self.position = [830.0, 920.0]
        self.angle = 0
        self.speed = 0
        self.speed_set = False

        self.center = [self.position[0] + CARX / 2,
self.position[1] + CARY / 2]
        self.radars = []
        self.drawing_radars = []
        self.alive = True
        self.distance = 0
        self.time = 0

    def draw(self, screen):
        screen.blit(self.rotated_sprite, self.position)
        self.draw_radar(screen)
```



```

def draw_radar(self, screen):
    for radar in self.radars:
        position = radar[0]
        pygame.draw.line(screen, "blue", self.center,
position, 1)
        pygame.draw.circle(screen, "blue", position, 5)

def check_collision(self, track):
    self.alive = True
    for point in self.corners:
        if track.get_at((int(point[0]), int(point[1]))) ==
BODCOL:
            self.alive = False
            break

def check_radar(self, degree, game_map):
    length = 0
    x = int(self.center[0] + math.cos(math.radians(360 -
(self.angle + degree))) * length)
    y = int(self.center[1] + math.sin(math.radians(360 -
(self.angle + degree))) * length)

    while not game_map.get_at((x, y)) == BODCOL and length
< 300:
        length = length + 1
        x = int(self.center[0] + math.cos(math.radians(360
- (self.angle + degree))) * length)
        y = int(self.center[1] + math.sin(math.radians(360
- (self.angle + degree))) * length)

        dist = int(math.sqrt(math.pow(x - self.center[0], 2) +
math.pow(y - self.center[1], 2)))
        self.radars.append([(x, y), dist])

def update(self, game_map):

    if not self.speed_set:
        self.speed = 20
        self.speed_set = True

    self.rotated_sprite = self.rotate_center(self.sprite,
self.angle)
    self.position[0] += math.cos(math.radians(360 -
self.angle)) * self.speed
    self.position[0] = max(self.position[0], 20)
    self.position[0] = min(self.position[0], WID - 120)

```

```

        self.distance += self.speed
        self.time += 1

        self.position[1] += math.sin(math.radians(360 -
self.angle)) * self.speed
        self.position[1] = max(self.position[1], 20)
        self.position[1] = min(self.position[1], WID - 120)

        self.center = [int(self.position[0]) + CARX / 2,
int(self.position[1]) + CARY / 2]

        length = 0.5 * CARX
        left_top = [self.center[0] + math.cos(math.radians(360
- (self.angle + 30))) * length, self.center[1] +
math.sin(math.radians(360 - (self.angle + 30))) * length]
        right_top = [self.center[0] + math.cos(math.radians(360
- (self.angle + 150))) * length, self.center[1] +
math.sin(math.radians(360 - (self.angle + 150))) * length]
        left_bottom = [self.center[0] +
math.cos(math.radians(360 - (self.angle + 210))) * length,
self.center[1] + math.sin(math.radians(360 - (self.angle +
210))) * length]
        right_bottom = [self.center[0] +
math.cos(math.radians(360 - (self.angle + 330))) * length,
self.center[1] + math.sin(math.radians(360 - (self.angle +
330))) * length]
        self.corners = [left_top, right_top, left_bottom,
right_bottom]
        self.check_collision(game_map)
        self.radars.clear()

        for d in range(-90, 120, 45):
            self.check_radar(d, game_map)

    def get_data(self):
        radars = self.radars
        return_values = [0, 0, 0, 0, 0]
        for i, radar in enumerate(radars):
            return_values[i] = int(radar[1] / 30)
        return return_values

    def is_alive(self):
        return self.alive

    def get_reward(self):

```

```

        return self.distance / (CARX / 2)

    def rotate_center(self, image, angle):
        rectangle = image.get_rect()
        rotated_image = pygame.transform.rotate(image, angle)
        rotated_rectangle = rectangle.copy()
        rotated_rectangle.center =
rotated_image.get_rect().center
        rotated_image =
rotated_image.subsurface(rotated_rectangle).copy()
        return rotated_image

def run_simulation(genomes, config):
    nets = []
    cars = []
    pygame.init()
    screen = pygame.display.set_mode((WID, HEI),
pygame.FULLSCREEN)

    for i, g in genomes:
        net = NEAT.nn.FeedForwardNetwork.create(g, config)
        nets.append(net)
        g.fitness = 0
        cars.append(Car())

    clock = pygame.time.Clock()
    generation_font = pygame.font.SysFont("Arial", 30)
    alive_font = pygame.font.SysFont("Arial", 20)
    time_font = pygame.font.SysFont("Arial", 20)
    track = pygame.image.load('Miniproj/map.png').convert()
    global cur_gen
    cur_gen += 1
    counter = 0

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit(0)

        for i, car in enumerate(cars):
            output = nets[i].activate(car.get_data())
            choice = output.index(max(output))
            if choice == 0:
                car.angle += 10
            elif choice == 1:
                car.angle -= 10

```

```

        elif choice == 2:
            if(car.speed - 2 >= 12):
                car.speed -= 2
            else:
                car.speed += 2

    still_alive = 0
    for i, car in enumerate(cars):
        if car.is_alive():
            still_alive += 1
            car.update(track)
            genomes[i][1].fitness += car.get_reward()

    if still_alive == 0:
        break

    counter += 1
    if counter == 30 * 40:
        break

    screen.blit(track, (0, 0))
    for car in cars:
        if car.is_alive():
            car.draw(screen)

    elapsed_time = time.time() - start_time
    elapsed_time = int(elapsed_time)

    text = generation_font.render("Generation: " +
str(cur_gen), True, "black")
    text_rect = text.get_rect()
    text_rect.center = (900, 450)
    screen.blit(text, text_rect)

    text = alive_font.render("Still Alive: " +
str(still_alive), True, "black")
    text_rect = text.get_rect()
    text_rect.center = (900, 490)
    screen.blit(text, text_rect)

    text = time_font.render("Time passed: "+
str(elapsed_time), True, "black")
    text_rect = text.get_rect()
    text_rect.center = (900, 530)
    screen.blit(text, text_rect)
    pygame.display.flip()

```

```

        clock.tick(120)

if __name__ == "__main__":
    config_path = "Miniproj/config.txt"
    config = NEAT.Config(NEAT.DefaultGenome,
NEAT.DefaultReproduction, NEAT.DefaultSpeciesSet,
NEAT.DefaultStagnation, config_path)
    population = NEAT.Population(config)
    population.add_reporter(NEAT.StdOutReporter(True))
    stats = NEAT.StatisticsReporter()
    population.add_reporter(stats)
    population.run(run_simulation, 1000)

```

4.3. RESULT

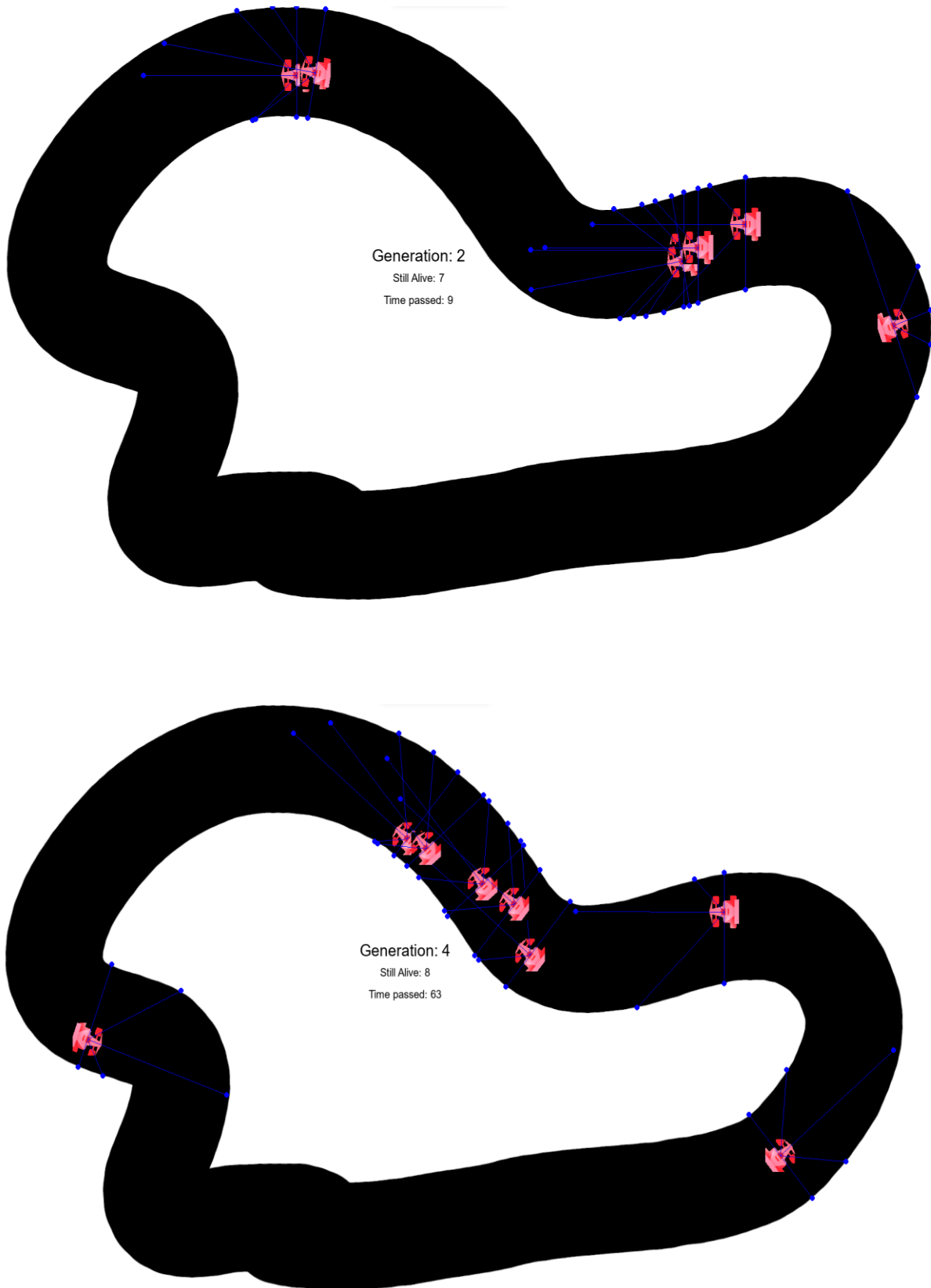


Fig 4.2: The cars running under simulation

CHAPTER 5

5.1. CONCLUSION

The algorithm helps in identifying the problems and other bugs before the deployment of self-driving car to the public. Further, the NEAT algorithm helps in understanding the performance better and evaluate the limitations of the technology. The power efficiency of the AI algorithm in learning is gauged.

In this paper, we have presented a simulation of a self-driving car using reinforcement learning techniques. Through our experiments, we have demonstrated the effectiveness of the reinforcement learning approach in training an autonomous vehicle to navigate complex driving scenarios.

Our study suggests that reinforcement learning is a promising technique for developing autonomous driving systems, as it enables the vehicle to learn from its own experience and improve its performance over time. Further research can explore the integration of additional sensory data and more complex decision-making algorithms to enhance the performance of our self-driving car in real-world scenarios.

Overall, our work contributes to the development of autonomous driving technology and highlights the potential of reinforcement learning as a key approach for enabling self-driving vehicles in the future.

5.2. FUTURE ENHANCEMENT

The code can be used in unity and 3D environment when scaled up to a large scale with more complexity. Major Industries like Tesla, Mercedes etc., can implement a map of the cities/towns and can use machine learning to efficiently and quickly traverse the map. Collision avoidance can be achieved when implemented with many other agents in the map.

REFERENCES

- [1] Kenneth O. Stanley and Risto Miikkulainen, “Evolving Neural Networks Through Augmenting Topologies”, Journal – Evolutionary Computation , 2002
- [2]BD Bryant, R Miikkulainen,” Evolving Neural network agents in the NERO videogame”, Journal – IEEE Transactions on Computational Intelligence and AI in games, 2005
- [3] Kenneth O Stanley, David B D'Ambrosio, Jason Gauci, “A Hypercube based encoding for evolving large scale neural network” ,Conference – Genetic and Evolutionary Computational Conference (GECCO), 2009
- [4] W.H. van Willigen, Evert Haasdijk, L.K.H.M Kester, “Evolving intelligent vehicle control using multi-objective NEAT”, Journal – IEEE Transactions on Computational Intelligence and AI in games, 2013
- [5] Humphrey Munn, Marcus Gallagher, “Modularity in NEAT reinforcement Learning Networks”, Journal – Evolutionary Computation, 2017
- [6]V.Madhavan, FP Such, ”Deep Neuroevolution: Genetic Algorithms are a competitive alternative for Training deep neural networks for reinforcement learning”, Journal – Journal of Machine Learning Research, 2017
- [7] Christian P, Michael Sinapius, Marco Brysch, Naser Al Natsheh, “A NEAT based two stage neural network approach to generate a control algorithm for a pultrusion system”, Journal - NeuroComputing,2021