VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CSE3013 – ARTIFICIAL INTELLIGENCE PROJECT

Winter Semester 2021-22

B1 + TB1 Slot

Project Report

Game Automation using NEAT Algorithm

Submitted by

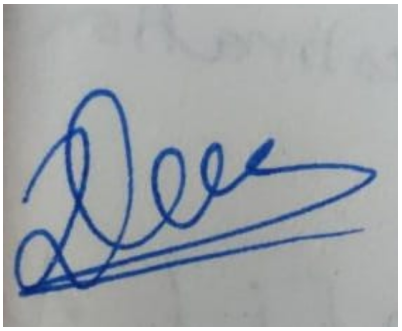
S.no	Name	Reg No
1	Dev Sinha	20BCE0723
2	Gaurab Kumar Rauniyar	20BCE2915
3	Vaishnav Sathiajith Menon	20BCI0014
4	Shantanu Patra	20BCI0088

Professor : Dr. Delhibabu R

School of Computer Science and Engineering

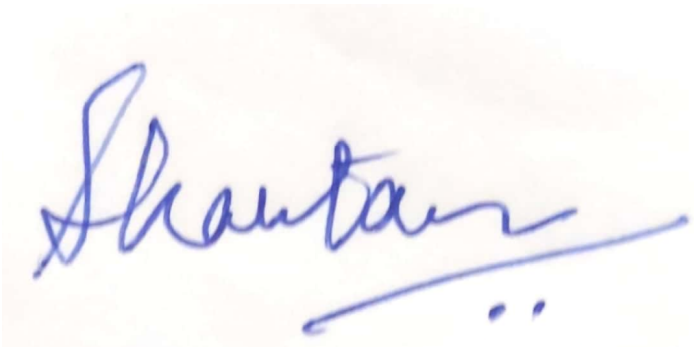
DECLARATION

I hereby declare that the J Component report entitled "TRAINING A.I TO PLAY COMPUTER GAMES USING NEAT ALGORITHM" submitted by me to VIT, Vellore-14 in partial fulfilment of the requirement for the award of the degree of B.Tech in **Computer science and engineering** is a record of bonafide undertaken by me under the supervision of **Dr. R. Delhi Babu** I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.



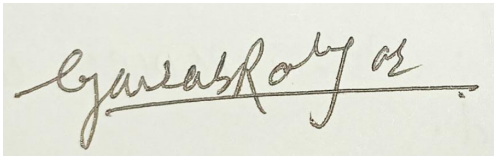
Dev Sinha

20BCE0723



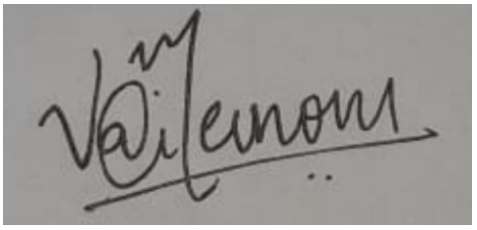
Shantanu Patra

20BCI0088



Gaurab Kumar Rauniyar

20BCE2915



Vaishnav Sathiajith Menon

20BCI0014

TABLE OF CONTENTS

Introduction

In this project, two very popular games – Ping-pong and Dino, have been automated using the NEAT algorithm. NeuroEvolution of Augmenting Topologies (NEAT) is a genetic algorithm (GA) for the generation of evolving artificial neural networks (a neuroevolution technique) developed by Ken Stanley in 2002 while at The University of Texas at Austin. We have further enhanced our project using Mediapipe and OpenCV for hand gesture control. MediaPipe is a Framework for building machine learning pipelines for processing time-series data like video, audio, etc.

This cross-platform Framework works in Desktop/Server, Android, iOS, and embedded devices like Raspberry Pi and Jetson Nano. OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products.

Abstract

NEAT (NeuroEvolution of Augmenting Topologies) is an evolutionary algorithm that creates artificial neural networks. To evolve a solution to a problem, the user must provide a fitness function which computes a single real number indicating the quality of an individual genome: better ability to solve the problem means a higher score. The algorithm progresses through a user-specified number of generations, with each generation being produced by reproduction (either sexual or asexual) and mutation of the most fit individuals of the previous generation.

The reproduction and mutation operations may add nodes and/or connections to genomes, so as the algorithm proceeds, genomes (and the neural networks they produce) may become more and more complex. When the preset number of generations is reached, or when at least one individual (for a fitness criterion function of max; others are configurable) exceeds the user-specified fitness threshold, the algorithm terminates.

It uses a combination of the number of non-homologous nodes and connections with measures of how many homologous nodes and connections have diverged since their common origin.

The NEAT algorithm chooses a direct encoding methodology because of this. Their representation is a little more complex than a simple graph or binary encoding, however, it is still straightforward to understand. It simply has two lists of genes, a series of nodes and a series of connections.

Using OpenCV, one can process images and videos to identify objects, faces, or even handwriting of a human. When integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis.

MediaPipe provides cornerstone Machine Learning models for common tasks like hand tracking, therefore removing the same developmental bottleneck that exists for a host of Machine Learning applications. These models, along with their excessively easy-to-use APIs, in turn streamline the development process and reduce project lifetime for many applications that rely on Computer Vision.

Requirements

Software Specifications :

- Python 3.5.0 or above
- Windows 8 or above
- Compiler that supports

Hardware Specifications :

- 4GB RAM
- 4GB ROM
- GPU
- Any modern PC processor (Intel i3.Apple M1 etc.)

Libraries :

- **neat-python**
- **pygame**
- Pip
- WebOp
- setuptools
- wheel
- **Open cv**
- **Media pipe**
- Random
- **Keyboard**
- Time

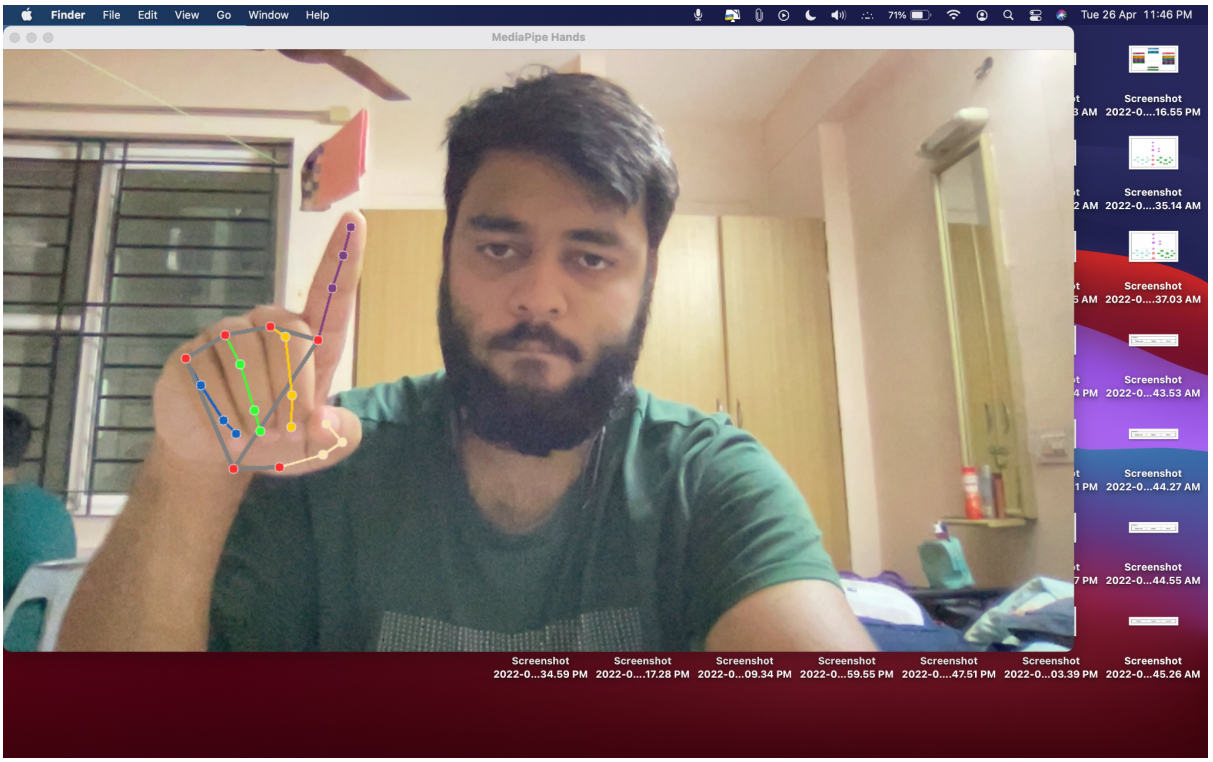
INPUT:

To train the AI model using Neat Algorithm:

Using NEAT we set up an algorithm to evolve minimal networks by starting all networks with no hidden nodes. Each individual in the initial population is simple input nodes, output nodes and a series of connection genes between them. When combined with speciation, this proves to be a powerful idea in evolving minimal, yet high-performance networks.

To control player paddle using hand gestures:

Using OpenCV library, we access the webcam of our PC and using media pipes hand landmarks we use the x - coordinate of the tip of the index finger to control the ping pong paddle used by the player, while the other paddle is being controlled by the neural networks trained using the NEAT algorithm.



OUTPUT:

→ We have created an unbeatable AI model using NEAT algorithm for various classic arcade games.
→ Neural network that has reached fitness threshold criteria for pong.

→ We have successfully automated the dino game using NEAT algorithm.

→ We are able to control a paddle using the simple hand gestures of the player.

Algorithm (Dino Game):

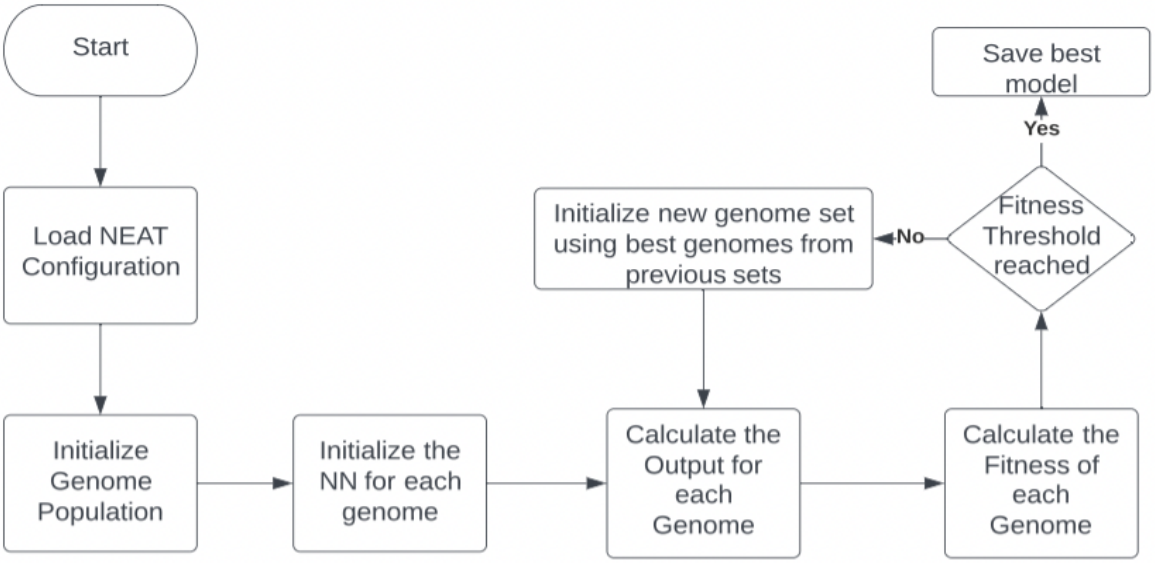
```
if (!dino.jumping) {
  let action = 0;// variable for action 1 for jump 0 for not
  const prediction =
dino.model.predict(tf.tensor2d([convertStateToVector(state)]));
  const predictionPromise = prediction.data();
  predictionPromise.then((result) => {
  if (result[1] > result[0]) {
  // we want to jump
  action = 1;
  // set last jumping state to current state
dino.lastJumpingState = state;
  } else {
  // set running state to current state
  dino.lastRunningState = state;
  }
resolve(action);});
```

Algorithm(PONG):

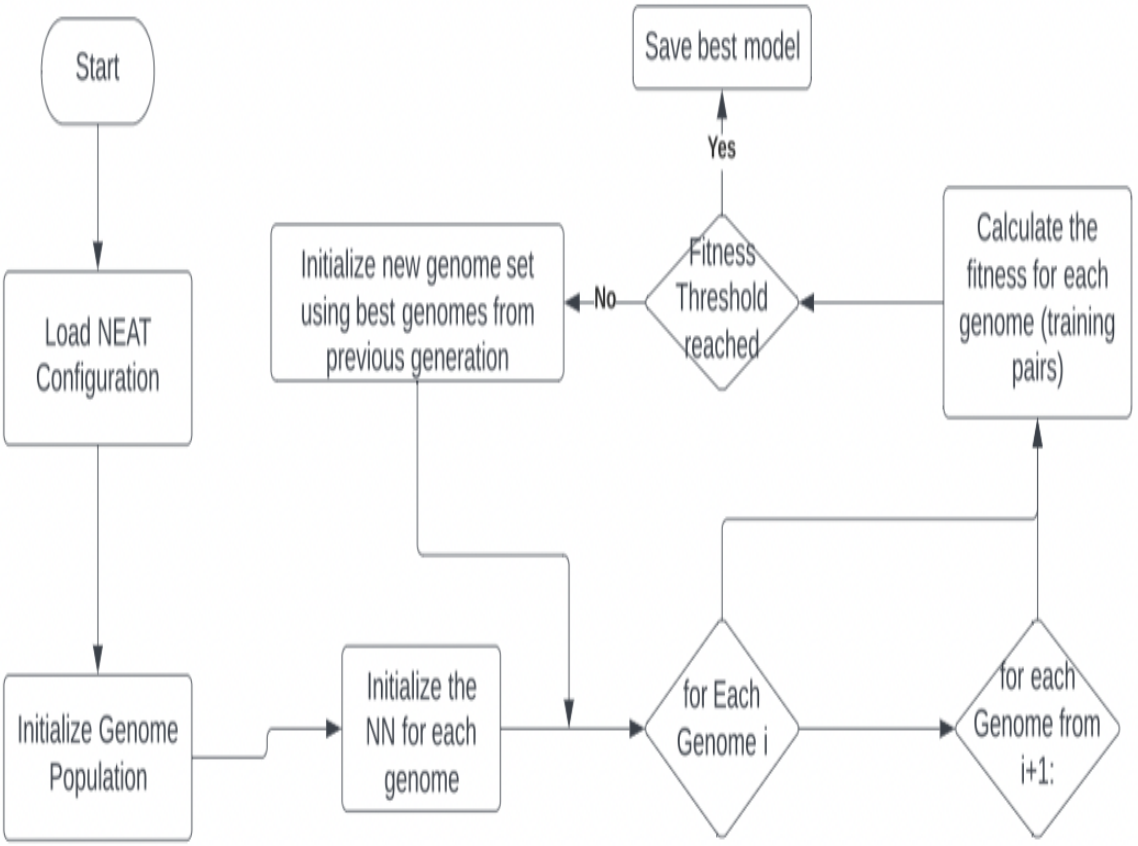
```
start:
    config = neat.abstract(config.txt)
    p = neat.initializePopulation(config)
    winnerNN = p.run(eval_genome, 50)
    save winnnerNN into best.pickle
    eval_genome:
        for genome1 in enumerate(gemones):
            genome1.fitness = 0
            for genome2 in genomes[min(i+1, len(genomes) - 1):]:
                train_ai(genome1, genome2, config)
    train_ai(genome1, genome2, config):
        net1 = neat.nn.FeedForwardNetwork.create(genome1, config)
        net2 = neat.nn.FeedForwardNetwork.create(genome2, config)
        while true:
            game_info = self.game.loop()
            self.move_ai_paddles(net1, net2)
   move_ai_paddle(net1, net2):
            output1 = net1.activate((left_paddle.y, ball.y, dist(left_paddle,
ball)))
            decision1 = output1.index(max(output1))

            if decision1 == 0:
                pass
            elif decision1 == 1:
                self.game.move_paddle(left=True, up=True)
            else:
                self.game.move_paddle(left=True, up=False)

            output2 = net2.activate((right_paddle.y, ball.y, dist(right_paddle,
ball)))
            decision2 = output2.index(max(output2))

            if decision2 == 0:
                pass
```

```
elif decision2 == 1:
    self.game.move_paddle(left=False, up=True)
else:
    self.game.move_paddle(left=False, up=False)

game_info = game.loop()

game.draw(draw_score=False, draw_hits=True)
pygame.display.update()
```

Flow Chart (Dino Game):



Flow Chart(PONG):



SCREEN SHOTS:
CONFIGURATION(.txt) OF NEAT ALGORITHM:

The NEAT section **specifies parameters particular to the generic NEAT algorithm or the experiment itself**. This section is always required, and is handled by the Config class itself.

Fitness_criterion:
The function used to compute the termination criterion from the set of genome fitnesses.Allowable values are: min, max, and mean

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 400
pop_size              = 50
reset_on_extinction   = False
```

Max_stagnation:
Species that have not shown improvement in more than this number of generations will be considered stagnant and removed. This defaults to 15.

Species_elitism:
The number of species that will be protected from stagnation; mainly intended to prevent total extinctions caused by all species becoming stagnant before new species arise.

```
[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 20
species_elitism      = 2
```

Feed_forward: If this evaluates to True, generated networks will not be allowed to have recurrent connections (they will be feedforward). Otherwise they may be (but are not forced to be) recurrent.

```
# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.01

feed_forward            = True
initial_connection      = full_direct
```

Initial_connection: Specifies the initial connectivity of newly-created genomes. (Note the effects on settings other than unconnected of the enabled_default parameter.)

node_add_prob: The probability that mutation will add a new node (essentially replacing an existing connection, the enabled status of which will be set to False). Valid values are in [0.0, 1.0].

node_delete_prob: The probability that mutation will delete an existing node (and all connections to it). Valid values are in [0.0, 1.0].

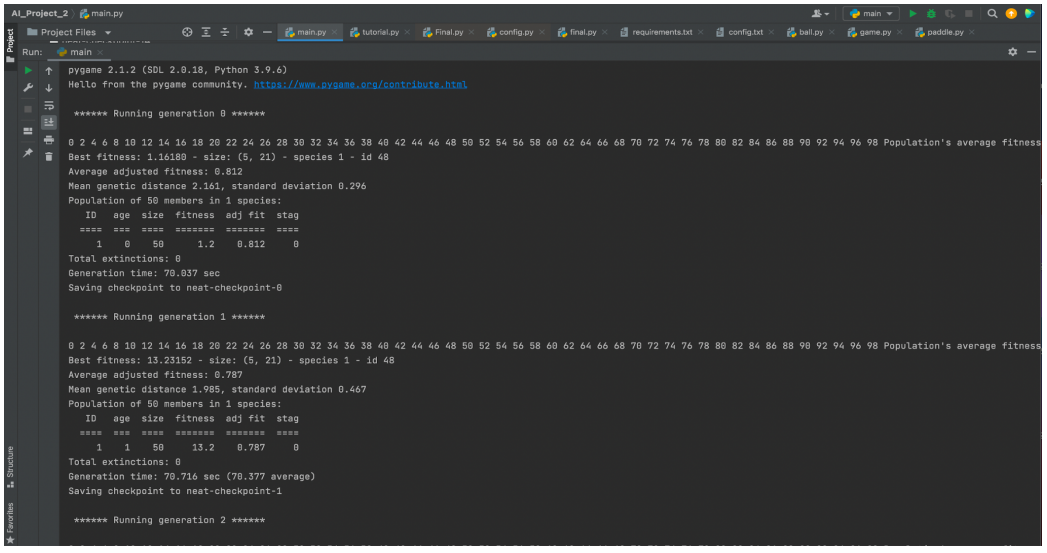
num_hidden: The number of hidden nodes to add to each genome in the initial population.

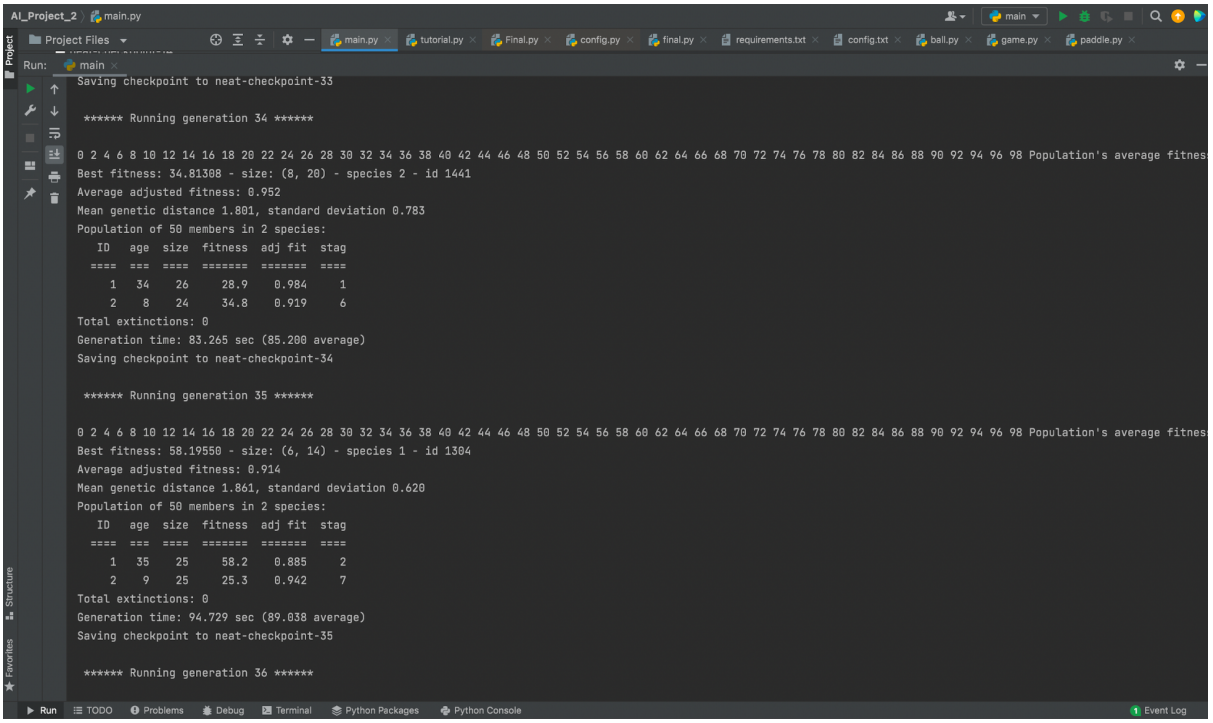
num_inputs: The number of input nodes, through which the network receives inputs. **num_outputs:** The number of output nodes, to which the network delivers outputs.

```
# node add/remove rates
node_add_prob             = 0.2
node_delete_prob          = 0.2

# network parameters
num_hidden                = 2
num_inputs                = 3
num_outputs               = 3
```

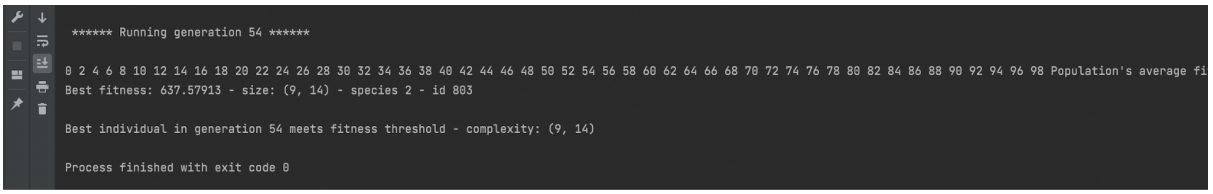
TRAINING MODEL:

Here we train the model, we initialise each genome to compete with other genomes to give a fitness value to each genome and by that way the algorithm decides which genomes will become a part of elitism, and be used for initialization of the next set of genomes for training the next generation.



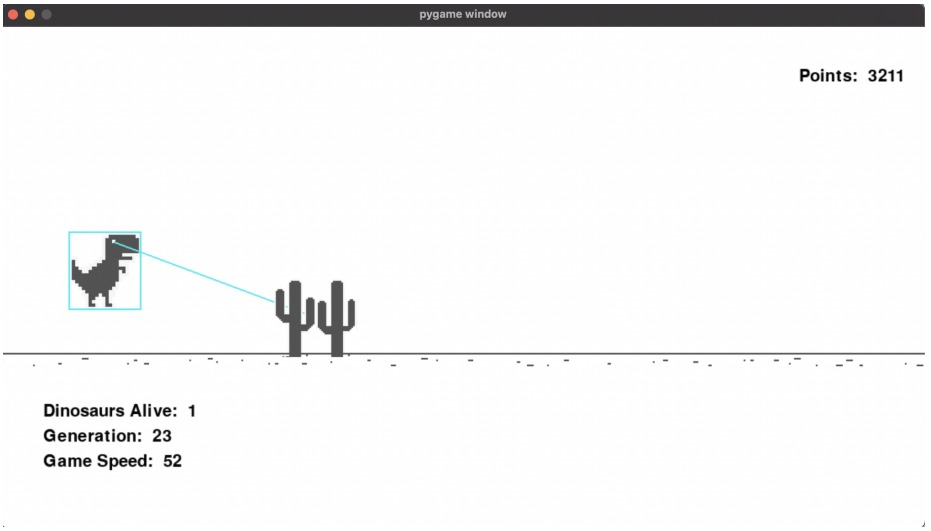


As we can see in generation 54, one of the genomes has reached the fitness threshold, so the training phase has completed and has returned the most suited genome(NN) as our winner and will be used to play against the human opponent.

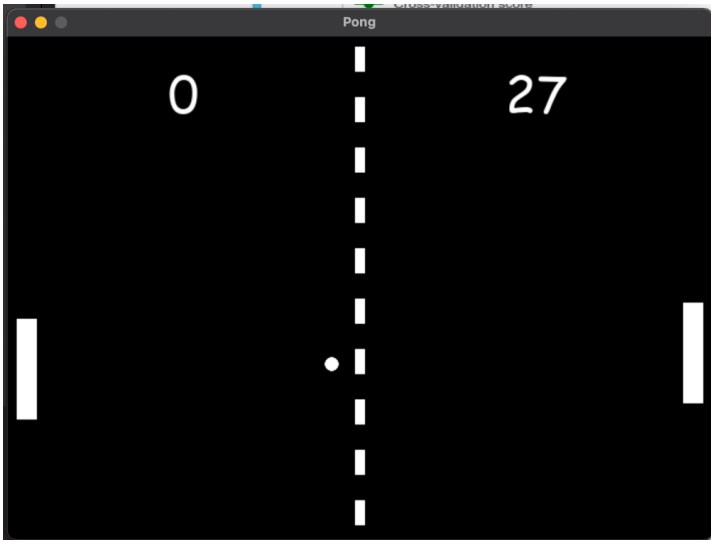


OUTPUT MODEL:

This dino game is fully being controlled by the Neural Net that has been trained by our AI model.

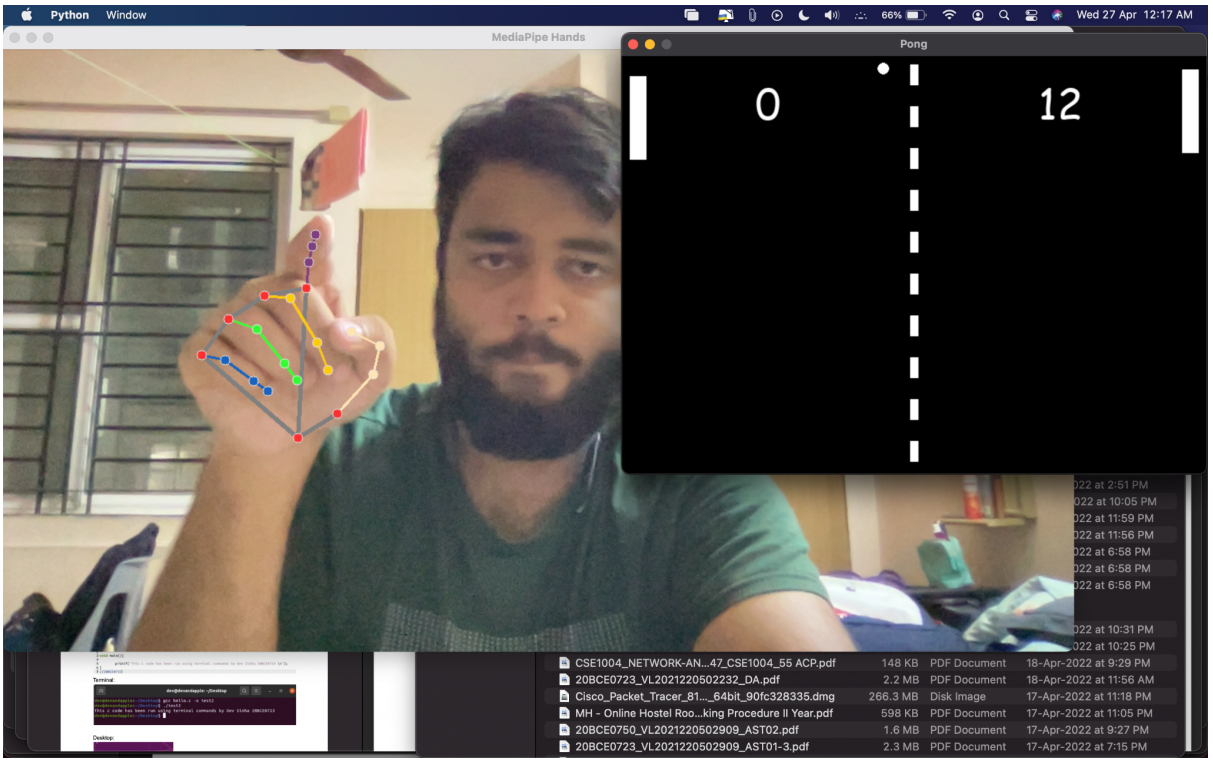


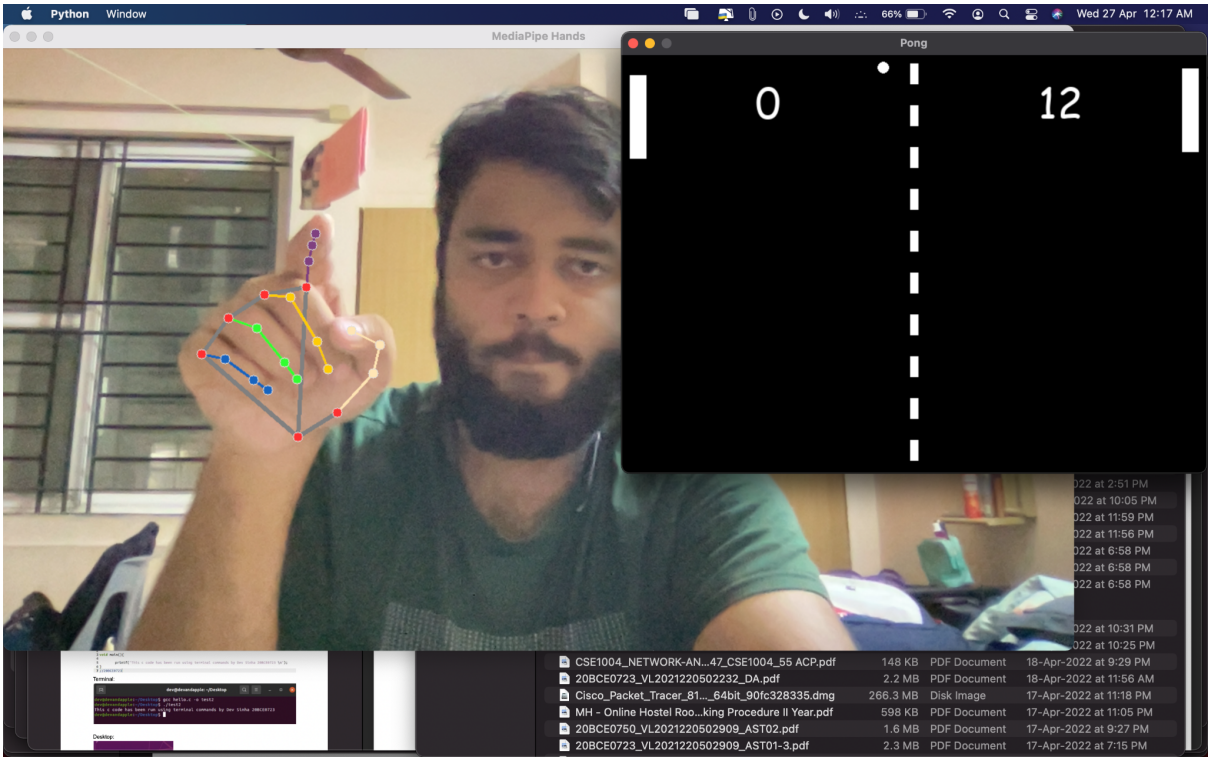
Here we can see the clear dominance of our AI model over the human player



HAND GESTURE CONTROL:

Here we have used media pipes pre trained hand landmarks to get the coordinates of the fingertips of our index finger and we have implemented it in such a way that if our hand is in the upper half of the screen the user controlled paddle moves up and if the hand is in the lower part of the screen then the paddle moves down, and accordingly the AI model moves the opponent paddle.





Conclusion

Neuroevolution (NE) is the artificial evolution of neural networks using genetic algorithms. Here we have implemented the Neuroevolution of Augmenting Topologies (NEAT) algorithm in training the PONG and DINO game. Firstly, Fitness_criterion, Fitness_Threshold, Population_size, Extinction_threshold was initialised. Then, depending on the parameters all the corresponding

genome values were recorded. Similarly, a Configuration list with the MAX fitness value was obtained. Using this elitism list we decide which genomes to choose for the crossover to form the genomes of the next generation.

After a few generations of slowly learning, mutating, speciation and crossover, one genome from a particular species and generation reaches the pre-defined fitness threshold, hence returning the "winner" neural network.

Once the score reaches the fitness threshold we set, then we can exit the training by saving the neural network associated with that ball using picke. We have also initiated the hand gesture for controlling the paddle from the user side.

Despite the inconclusive empirical results, NEAT might still be a viable heuristic process for finding a solution for optimization tasks, especially when there is only little to no information about the optimal dimension of controllers. NEAT can also be used to explore dimension space.

APPENDICES

Appendix 1: NEAT-Python 0.92

NEAT is a method for evolving arbitrary neural networks. NEAT-Python is a pure Python implementation of NEAT, with no dependencies other than the Python standard library.

NeuroEvolution of Augmenting Topologies (NEAT) is a genetic algorithm (GA)

for the generation of evolving artificial neural networks. It alters both the weighting parameters and structures of networks, attempting to find a balance between the fitness of evolved solutions and their diversity. It is based on applying three key techniques: tracking genes with history markers to allow crossover among topologies, applying speciation (the evolution of species) to preserve innovations, and developing topologies incrementally from simple initial structures ("complexifying").

Appendix 2: mediapipe 0.8.9.1

MediaPipe Hands is a high-fidelity hand and finger tracking solution. It employs machine learning (ML) to infer 21 3D landmarks of a hand from just a single frame.

Whereas current state-of-the-art approaches rely primarily on powerful desktop environments for inference, this method achieves real-time performance on a mobile phone, and even scales to multiple hands.