**Auto-Playing Flappy Bird using Neuro Evolution of Augmented Topologies**

* **K S ASWATH RAMANA**

**DATA DESCRIPTION:**

This project does not utilize datasets in the traditional sense, as it involves creating an AI player for the game "Flappy Bird" using the NEAT (NeuroEvolution of Augmenting Topologies) algorithm. The game environment itself serves as the dataset, with the AI learning and evolving its behaviour through simulated gameplay.

**LIBRARIES USED:**

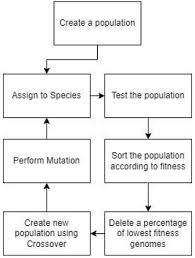
1. **PYGAME**
2. **NEAT**
3. **RANDOM, PANDAS, OS, TIME, VISUALIZE**

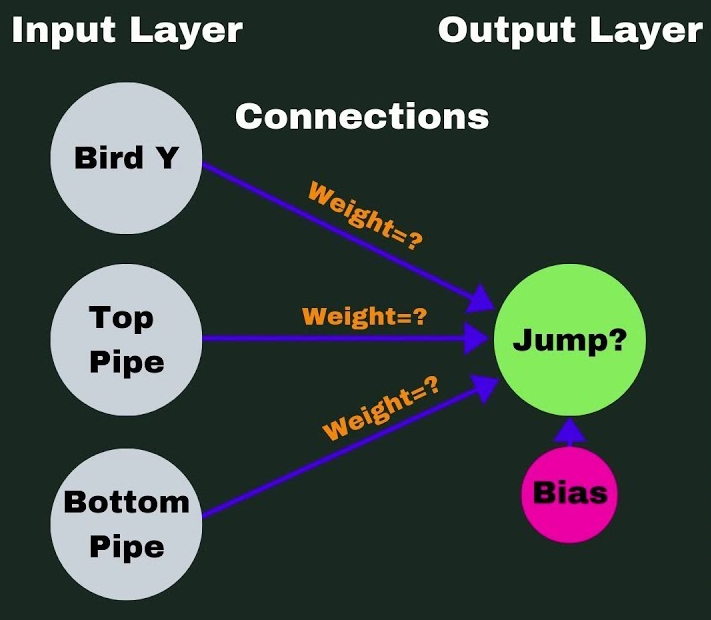
**HOW NEAT WORKS:**

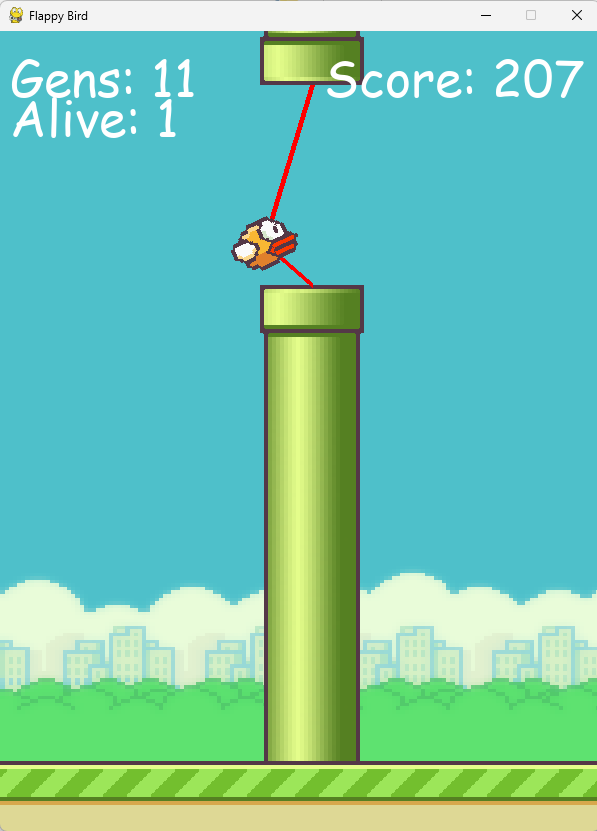
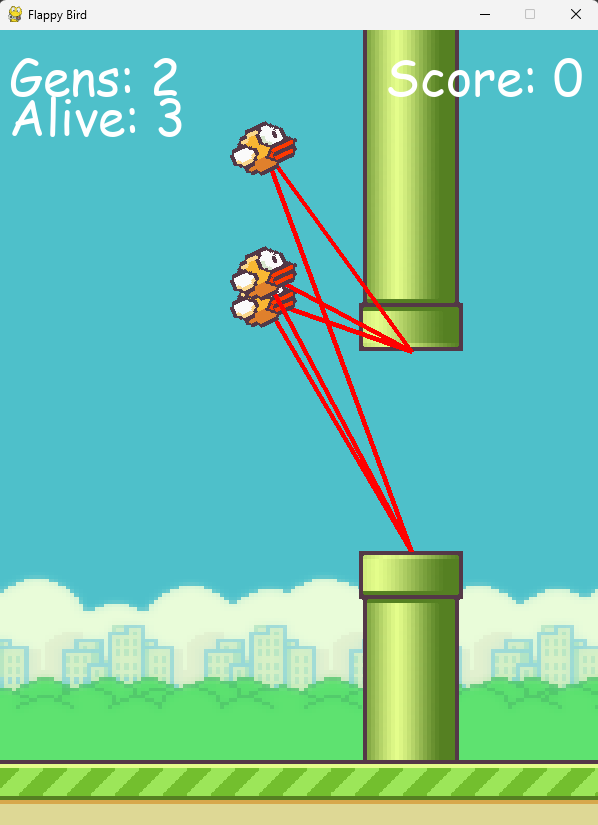
1. **Initialization:** NEAT starts with a population of simple neural networks, typically with a small number of neurons and connections.
2. **Evaluation:** Each neural network in the population is evaluated on a given task or set of tasks. The performance of each network is measured using a fitness function, which quantifies how well the network performs the given task(s).
3. **Selection:** After evaluation, networks are selected for reproduction based on their fitness. Networks with higher fitness scores are more likely to be selected for reproduction, while lower-performing networks are discarded.
4. **Reproduction:** NEAT employs two main genetic operators for reproduction: crossover and mutation.
   1. Crossover: Two parent networks are selected and their genomes (representations of neural network structures) are crossed over to create offspring networks. This helps in exchanging useful genetic information between parents.
   2. Mutation: Random changes are applied to the genomes of offspring networks, including adding new nodes (neurons) and new connections between nodes. This allows for the exploration of new network structures.
5. **Speciation:** NEAT maintains diversity in the population by organizing individuals into species based on their genetic similarity. Networks are grouped into species, and mating primarily occurs within species. This prevents premature convergence to suboptimal solutions.
6. **Fitness Sharing:** To encourage diversity within species, NEAT uses fitness sharing, which penalizes networks that are too similar to others within the same species. This prevents domination of the population by a few highly fit individuals.
7. **Iterative Evolution**: The process of evaluation, selection, reproduction, speciation, and fitness sharing is repeated for multiple generations. Over time, NEAT evolves increasingly complex neural networks that exhibit improved performance on the given tasks.

By combining the principles of genetic algorithms with neural networks, NEAT efficiently explores the vast search space of possible network topologies and parameters, ultimately producing highly adaptable and capable networks. NEAT has been applied successfully to various tasks, including control problems, game playing, and pattern recognition.

**NEAT:**



****

****

**CONFIGURATION FILE :**

**[NEAT]**

**fitness\_criterion = max**

**fitness\_threshold = 100**

**pop\_size = 10**

**reset\_on\_extinction = False**

**[DefaultGenome]**

**# node activation options**

**activation\_default = tanh**

**activation\_mutate\_rate = 0.0**

**activation\_options = tanh**

**# node aggregation options**

**aggregation\_default = sum**

**aggregation\_mutate\_rate = 0.0**

**aggregation\_options = sum**

**# node bias options**

**bias\_init\_mean = 0.0**

**bias\_init\_stdev = 1.0**

**bias\_max\_value = 30.0**

**bias\_min\_value = -30.0**

**bias\_mutate\_power = 0.5**

**bias\_mutate\_rate = 0.7**

**bias\_replace\_rate = 0.1**

**# genome compatibility options**

**compatibility\_disjoint\_coefficient = 1.0**

**compatibility\_weight\_coefficient = 0.5**

**# connection add/remove rates**

**conn\_add\_prob = 0.5**

**conn\_delete\_prob = 0.5**

**# connection enable options**

**enabled\_default = True**

**enabled\_mutate\_rate = 0.01**

**feed\_forward = True**

**initial\_connection = full**

**# node add/remove rates**

**node\_add\_prob = 0.2**

**node\_delete\_prob = 0.2**

**# network parameters**

**num\_hidden = 0**

**num\_inputs = 3**

**num\_outputs = 1**

**# node response options**

**response\_init\_mean = 1.0**

**response\_init\_stdev = 0.0**

**response\_max\_value = 30.0**

**response\_min\_value = -30.0**

**response\_mutate\_power = 0.0**

**response\_mutate\_rate = 0.0**

**response\_replace\_rate = 0.0**

**# connection weight options**

**weight\_init\_mean = 0.0**

**weight\_init\_stdev = 1.0**

**weight\_max\_value = 30**

**weight\_min\_value = -30**

**weight\_mutate\_power = 0.5**

**weight\_mutate\_rate = 0.8**

**weight\_replace\_rate = 0.1**

**[DefaultSpeciesSet]**

**compatibility\_threshold = 3.0**

**[DefaultStagnation]**

**species\_fitness\_func = max**

**max\_stagnation = 20**

**species\_elitism = 2**

**[DefaultReproduction]**

**elitism = 2**

**survival\_threshold = 0.2**

**CODE :**

import csv

import pandas as pd

import pygame

import random

import os

import time

import neat

import visualize

import pickle

pygame.font.init() # init font

WIN\_WIDTH = 600

WIN\_HEIGHT = 800

FLOOR = 730

STAT\_FONT = pygame.font.SysFont("comicsans", 50)

END\_FONT = pygame.font.SysFont("comicsans", 70)

DRAW\_LINES = True

WIN = pygame.display.set\_mode((WIN\_WIDTH, WIN\_HEIGHT))

pygame.display.set\_caption("Flappy Bird")

pipe\_img = pygame.transform.scale2x(pygame.image.load(os.path.join("imgs","pipe.png")).convert\_alpha())

bg\_img = pygame.transform.scale(pygame.image.load(os.path.join("imgs","bg.png")).convert\_alpha(), (600, 900))

bird\_images = [pygame.transform.scale2x(pygame.image.load(os.path.join("imgs","bird" + str(x) + ".png"))) for x in range(1,4)]

base\_img = pygame.transform.scale2x(pygame.image.load(os.path.join("imgs","base.png")).convert\_alpha())

gen = 0

global solved\_flag

solved\_flag = False

generation\_limit = 30

current\_generation = 0

class Bird:

MAX\_ROTATION = 25

IMGS = bird\_images

ROT\_VEL = 20

ANIMATION\_TIME = 5

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

self.tilt = 0 # degrees to tilt

self.tick\_count = 0

self.vel = 0

self.height = self.y

self.img\_count = 0

self.img = self.IMGS[0]

def jump(self):

self.vel = -10.5

self.tick\_count = 0

self.height = self.y

def move(self):

self.tick\_count += 1

# for downward acceleration

displacement = self.vel\*(self.tick\_count) + 0.5\*(3)\*(self.tick\_count)\*\*2 # calculate displacement

# terminal velocity

if displacement >= 16:

displacement = (displacement/abs(displacement)) \* 16

if displacement < 0:

displacement -= 2

self.y = self.y + displacement

if displacement < 0 or self.y < self.height + 50: # tilt up

if self.tilt < self.MAX\_ROTATION:

self.tilt = self.MAX\_ROTATION

else: # tilt down

if self.tilt > -90:

self.tilt -= self.ROT\_VEL

def draw(self, win):

self.img\_count += 1

# For animation of bird, loop through three images

if self.img\_count <= self.ANIMATION\_TIME:

self.img = self.IMGS[0]

elif self.img\_count <= self.ANIMATION\_TIME\*2:

self.img = self.IMGS[1]

elif self.img\_count <= self.ANIMATION\_TIME\*3:

self.img = self.IMGS[2]

elif self.img\_count <= self.ANIMATION\_TIME\*4:

self.img = self.IMGS[1]

elif self.img\_count == self.ANIMATION\_TIME\*4 + 1:

self.img = self.IMGS[0]

self.img\_count = 0

# so when bird is nose diving it isn't flapping

if self.tilt <= -80:

self.img = self.IMGS[1]

self.img\_count = self.ANIMATION\_TIME\*2

# tilt the bird

blitRotateCenter(win, self.img, (self.x, self.y), self.tilt)

def get\_mask(self):

return pygame.mask.from\_surface(self.img)

class Pipe():

GAP = 200

VEL = 5

def \_\_init\_\_(self, x):

self.x = x

self.height = 0

# where the top and bottom of the pipe is

self.top = 0

self.bottom = 0

self.PIPE\_TOP = pygame.transform.flip(pipe\_img, False, True)

self.PIPE\_BOTTOM = pipe\_img

self.passed = False

self.set\_height()

def set\_height(self):

self.height = random.randrange(50, 450)

self.top = self.height - self.PIPE\_TOP.get\_height()

self.bottom = self.height + self.GAP

def move(self):

self.x -= self.VEL

def draw(self, win):

# draw top

win.blit(self.PIPE\_TOP, (self.x, self.top))

# draw bottom

win.blit(self.PIPE\_BOTTOM, (self.x, self.bottom))

def collide(self, bird, win):

bird\_mask = bird.get\_mask()

top\_mask = pygame.mask.from\_surface(self.PIPE\_TOP)

bottom\_mask = pygame.mask.from\_surface(self.PIPE\_BOTTOM)

top\_offset = (self.x - bird.x, self.top - round(bird.y))

bottom\_offset = (self.x - bird.x, self.bottom - round(bird.y))

b\_point = bird\_mask.overlap(bottom\_mask, bottom\_offset)

t\_point = bird\_mask.overlap(top\_mask,top\_offset)

if b\_point or t\_point:

return True

return False

class Base:

VEL = 5

WIDTH = base\_img.get\_width()

IMG = base\_img

def \_\_init\_\_(self, y):

self.y = y

self.x1 = 0

self.x2 = self.WIDTH

def move(self):

self.x1 -= self.VEL

self.x2 -= self.VEL

if self.x1 + self.WIDTH < 0:

self.x1 = self.x2 + self.WIDTH

if self.x2 + self.WIDTH < 0:

self.x2 = self.x1 + self.WIDTH

def draw(self, win):

win.blit(self.IMG, (self.x1, self.y))

win.blit(self.IMG, (self.x2, self.y))

def blitRotateCenter(surf, image, topleft, angle):

rotated\_image = pygame.transform.rotate(image, angle)

new\_rect = rotated\_image.get\_rect(center = image.get\_rect(topleft = topleft).center)

surf.blit(rotated\_image, new\_rect.topleft)

def draw\_window(win, birds, pipes, base, score, gen, pipe\_ind):

if gen == 0:

gen = 1

win.blit(bg\_img, (0,0))

for pipe in pipes:

pipe.draw(win)

base.draw(win)

for bird in birds:

# draw lines from bird to pipe

if DRAW\_LINES:

try:

pygame.draw.line(win, (255,0,0), (bird.x+bird.img.get\_width()/2, bird.y + bird.img.get\_height()/2), (pipes[pipe\_ind].x + pipes[pipe\_ind].PIPE\_TOP.get\_width()/2, pipes[pipe\_ind].height), 5)

pygame.draw.line(win, (255,0,0), (bird.x+bird.img.get\_width()/2, bird.y + bird.img.get\_height()/2), (pipes[pipe\_ind].x + pipes[pipe\_ind].PIPE\_BOTTOM.get\_width()/2, pipes[pipe\_ind].bottom), 5)

except:

pass

# draw bird

bird.draw(win)

# score

score\_label = STAT\_FONT.render("Score: " + str(score),1,(255,255,255))

win.blit(score\_label, (WIN\_WIDTH - score\_label.get\_width() - 15, 10))

# generations

score\_label = STAT\_FONT.render("Gens: " + str(gen-1),1,(255,255,255))

win.blit(score\_label, (10, 10))

# alive

score\_label = STAT\_FONT.render("Alive: " + str(len(birds)),1,(255,255,255))

win.blit(score\_label, (10, 50))

pygame.display.update()

def eval\_genomes(genomes, config):

global WIN, gen

win = WIN

gen += 1

# start by creating lists holding the genome itself, the

# neural network associated with the genome and the

# bird object that uses that network to play

nets = []

birds = []

ge = []

for genome\_id, genome in genomes:

genome.fitness = 0 # start with fitness level of 0

net = neat.nn.FeedForwardNetwork.create(genome, config)

nets.append(net)

birds.append(Bird(230,350))

ge.append(genome)

base = Base(FLOOR)

pipes = [Pipe(700)]

score = 0

clock = pygame.time.Clock()

run = True

while run and len(birds) > 0:

clock.tick(30)

for event in pygame.event.get():

if event.type == pygame.QUIT:

run = False

pygame.quit()

quit()

break

pipe\_ind = 0

if len(birds) > 0:

if len(pipes) > 1 and birds[0].x > pipes[0].x + pipes[0].PIPE\_TOP.get\_width():

pipe\_ind = 1

for x, bird in enumerate(birds):

ge[x].fitness += 0.1

bird.move()

output = nets[birds.index(bird)].activate((bird.y, abs(bird.y - pipes[pipe\_ind].height),abs(bird.y - pipes[pipe\_ind].bottom)))

'''

input\_data\_df = pd.DataFrame(columns=["bird\_y", "bird\_pipe\_top\_diff", "bird\_pipe\_bottom\_diff", "output"])

input\_data\_df = input\_data\_df.append({"bird\_y": bird.y,

"bird\_pipe\_top\_diff": abs(bird.y - pipes[pipe\_ind].height),

"bird\_pipe\_bottom\_diff": abs(bird.y - pipes[pipe\_ind].bottom),

"output": output[0]},

ignore\_index=True)

input\_data\_df.to\_csv('neat\_input\_data.csv', mode='a', header=False, index=False)

'''

if output[0] > 0.5:

bird.jump()

base.move()

rem = []

add\_pipe = False

for pipe in pipes:

pipe.move()

# check for collision

for bird in birds:

if pipe.collide(bird, win):

ge[birds.index(bird)].fitness -= 1

nets.pop(birds.index(bird))

ge.pop(birds.index(bird))

birds.pop(birds.index(bird))

if pipe.x + pipe.PIPE\_TOP.get\_width() < 0:

rem.append(pipe)

if not pipe.passed and pipe.x < bird.x:

pipe.passed = True

add\_pipe = True

if add\_pipe:

score += 1

for genome in ge:

genome.fitness += 5

pipes.append(Pipe(WIN\_WIDTH))

for r in rem:

pipes.remove(r)

for bird in birds:

if bird.y + bird.img.get\_height() - 10 >= FLOOR or bird.y < -50:

nets.pop(birds.index(bird))

ge.pop(birds.index(bird))

birds.pop(birds.index(bird))

draw\_window(WIN, birds, pipes, base, score, gen, pipe\_ind)

if score > 10:

solved\_flag = True

'''if score > 20:

pickle.dump(nets[0],open("best.pickle", "wb"))

break'''

def run(config\_file):

config = neat.config.Config(neat.DefaultGenome, neat.DefaultReproduction,

neat.DefaultSpeciesSet, neat.DefaultStagnation,

config\_file)

p = neat.Population(config)

p.add\_reporter(neat.StdOutReporter(True))

stats = neat.StatisticsReporter()

p.add\_reporter(stats)

while not solved\_flag:

winner = p.run(eval\_genomes, 1)

print('\nBest genome:\n{!s}'.format(winner))

if \_\_name\_\_ == '\_\_main\_\_':

local\_dir = os.getcwd()

config\_path = os.path.join(local\_dir, 'config-feedforward.txt')

run(config\_path)

**COMPARISON :**

**Comparison are done with reference to**

1. **Batch Size**
   1. **As Batch size increases, the number of generations required to become accurate Decreases**
2. **Activation Function**
   1. **Not Much of a difference by changing Activation Function**
3. **Tuning Fitness Parameter**
   1. **Can be advantageous or Disadvantageous, will not work all the time.**
4. **Number of Inputs**
   1. **Can help in Increasing accuracy and reduce Training period.**

**NEAT vs OTHER APPROACH:**

Complex Environments: NEAT excels in evolving neural networks for complex environments with high-dimensional input spaces and non-linear dynamics. RL may struggle to learn in such environments due to the large state space and intricate relationships between actions and rewards.

No Need for Reward Design: In NEAT, fitness functions drive the evolution of neural networks based on performance metrics, without the need for explicitly defined reward structures. RL, on the other hand, often requires careful design of reward functions, which can be challenging and time-consuming, especially in complex tasks.

Minimal Hyperparameter Tuning: NEAT typically requires minimal hyperparameter tuning compared to RL algorithms, which often involve tuning parameters such as learning rates, exploration strategies, and discount factors. NEAT's genetic algorithm handles much of the optimization process automatically.

**CONCLUSION :**

NEAT (NeuroEvolution of Augmenting Topologies) is a powerful evolutionary algorithm used for evolving artificial neural networks (ANNs) with complex structures and behaviours. It introduces a genetic algorithm to evolve both the weights and architectures of neural networks simultaneously, allowing for the generation of increasingly sophisticated solutions to various problems.

In this project, NEAT was applied to create an AI player for the popular game Flappy Bird. Through simulated gameplay and evolutionary processes, the AI learned to navigate the game environment, avoiding obstacles and maximizing its score. NEAT proved effective in this context, showcasing its capability to evolve neural networks capable of exhibiting intelligent behaviours in dynamic and challenging environments.