# AI Agent plays Flappy Bird using NeuroEvolution of Augmented Topologies (NEAT)

Course: **Artificial Intelligence**

Project Report by : *Ali Hassan - 232406*

Introduction

1. Background of the project

The project demonstrates the application of neuroevolution and genetic algorithm techniques in a Flappy Bird game environment to solve time-sensitive decision-making problems, thus maximizing survival fitness. This project implements the NEAT (NeuroEvolution of Augmenting Topologies) algorithm, a powerful evolutionary approach based on genetic transmission that simultaneously evolves both the topology and weights of neural networks.

Traditional machine learning approaches used for AI integration in games often require extensive training on a dataset and supervised learning techniques, resulting in high computational time and cost. In contrast, the use of evolutionary algorithms in this project can generate strategies to avoid pipe collisions through natural selection, mutation, and crossover operations without any prior knowledge of game mechanics or optimal strategies.

2. Problem statement

The primary challenge in training an artificial agent or neural network to successfully navigate a Flappy Bird game environment is complex, as it involves overlapping layers and simultaneous neurons. Following are some of the problems to tackle:

1. *Precise Timing Control:* The agent must learn when to jump and when to let gravity take effect
2. *Spatial Awareness:* Understanding the bird's position relative to pipes and window boundaries
3. *Dynamic Decision Making:* Real time responses to changing game frames
4. *Optimization Under Constraints:* Maximizing survival time and score through avoiding pipes

Traditional rule-based methods often need a lot of manual setup with conditional and logical programming for game strategies. On the other hand, supervised learning relies on large datasets filled with expert gameplay examples. That's where evolutionary algorithms come in as they can automatically and effectively help train neural networks, making the process smoother and more efficient.

3. Objectives

- *Implement NEAT Algorithm:* Create a complete implementation of the NEAT evolutionary algorithm for neural network evolution
- *Game Environment Integration:* Develop a Flappy Bird game environment suitable for AI training
- *Real-time Visualization:* Provide feedback showing the learning process and neural network decision-making
- *Performance Optimization:* Achieve consistent pipe passage within 5-10 generations

- *Comparative Analysis:* Develop both AI and human-playable versions for performance comparison

4. Scope & targeted users

The potential scope of the project was
- Complete implementation of the NEAT algorithm with adjustable parameters.
- Visual training environment featuring real-time neural network input visualization.
- A comprehensive fitness evaluation system that includes multiple reward components.
- A human-playable version designed for comparison and entertainment.

The targeted users are as following:
- Students and researchers who are studying evolutionary algorithms.
- AI enthusiasts seeking to understand practical applications of genetic algorithms.
- Game developers exploring artificial intelligence techniques for autonomous behavior of game characters.

System Requirements

a. Hardware Requirements

Minimum:
- **Processor:** Intel Core i3 or AMD equivalent (2.0 GHz)
- **Memory:** 4 GB RAM
- **Graphics:** Integrated graphics card
- **Storage:** 100 MB available space
- **Display:** 1024x768 resolution

Recommended:
- **Processor:** Intel Core i5 or AMD equivalent (3.0 GHz)
- **Memory:** 8 GB RAM
- **Graphics:** Dedicated graphics card (for smoother visualization)
- **Storage:** 500 MB available space (for logs and data)
- **Display:** 1920x1080 resolution

b. Software Requirements

1) **Operating System:**
   Windows 10/11
   macOS 10.14 or later
   Linux (Ubuntu 18.04 or equivalent)
2) **Programming Environment:**
   Python 3.7 or higher
   pip package manager

3) Required Libraries:

Library	Version	Purpose
pygame	2.6.1	Game development and graphics
neat-python	0.92	NEAT algorithm implementation
numpy	1.21.0	Numerical computations

4) Development Tools:
Visual Studio Code

Responsibilities

1. **Core Algorithm Implementation:**
   - NEAT algorithm integration and configuration
   - Neural network architecture design (4-input feedforward)
   - Fitness function development and optimization
   - Evolution process management (selection, crossover, mutation)

2. **Game Environment Development:**
   - Flappy Bird physics simulation
   - Collision detection system (pixel-perfect)
   - Pipe generation and movement mechanics
   - Visual rendering and user interface

3. **AI Training System**
   - Population management (20 genomes per generation)
   - Real-time fitness evaluation
   - Species formation and diversity maintenance
   - Performance monitoring and statistics

4. **Visualization and Debugging:**
   - Neural network input visualization (red line system)
   - Real-time training progress display
   - Debug controls and user interaction
   - Console output for generation statistics
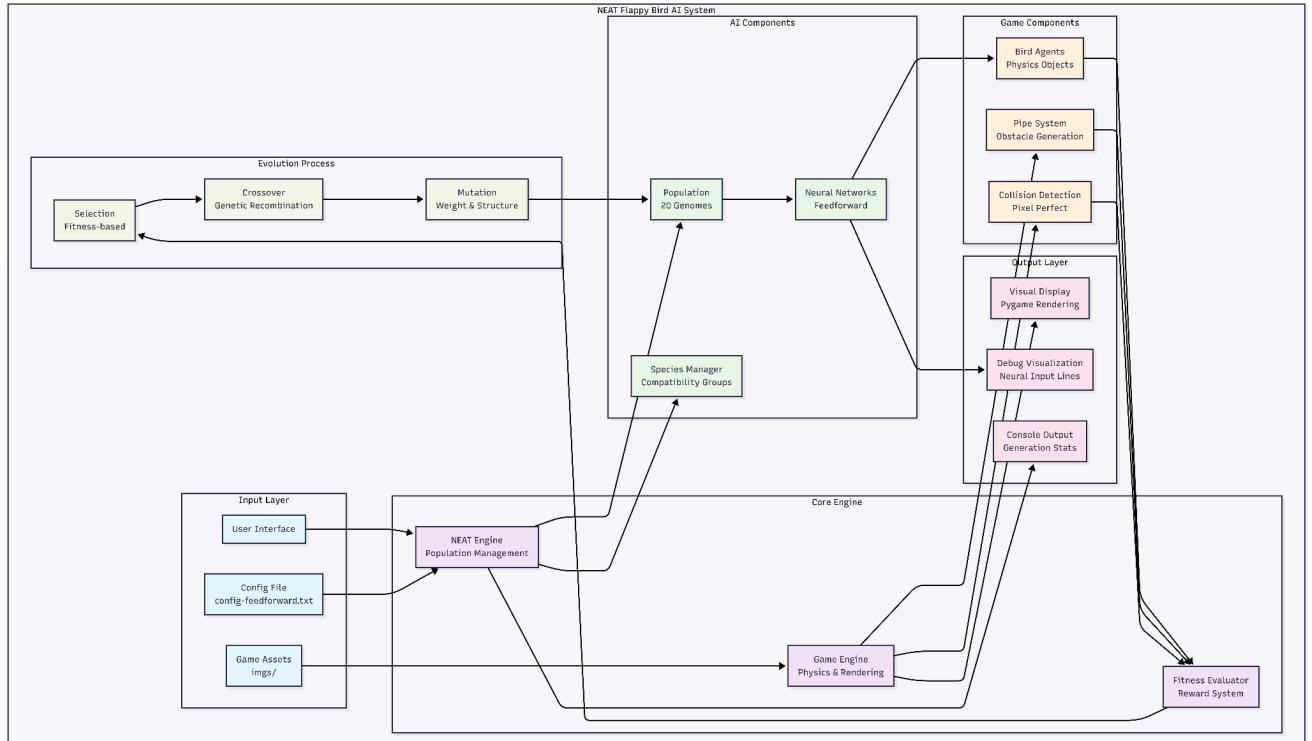
5. **Documentation and Testing:**
   - Comprehensive system documentation
   - Architecture and data flow diagrams
   - Performance testing and evaluation
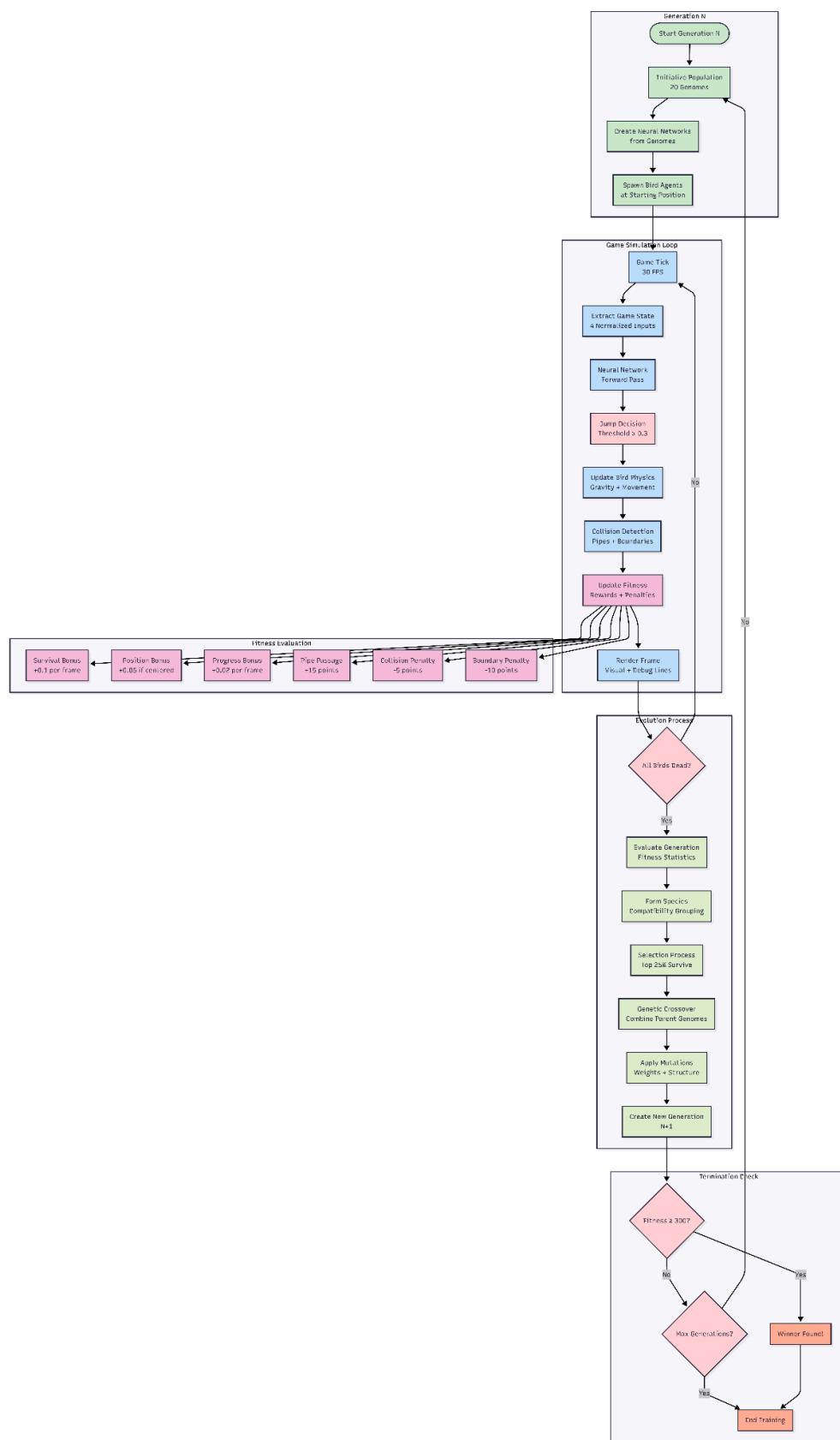
System Design & Architecture
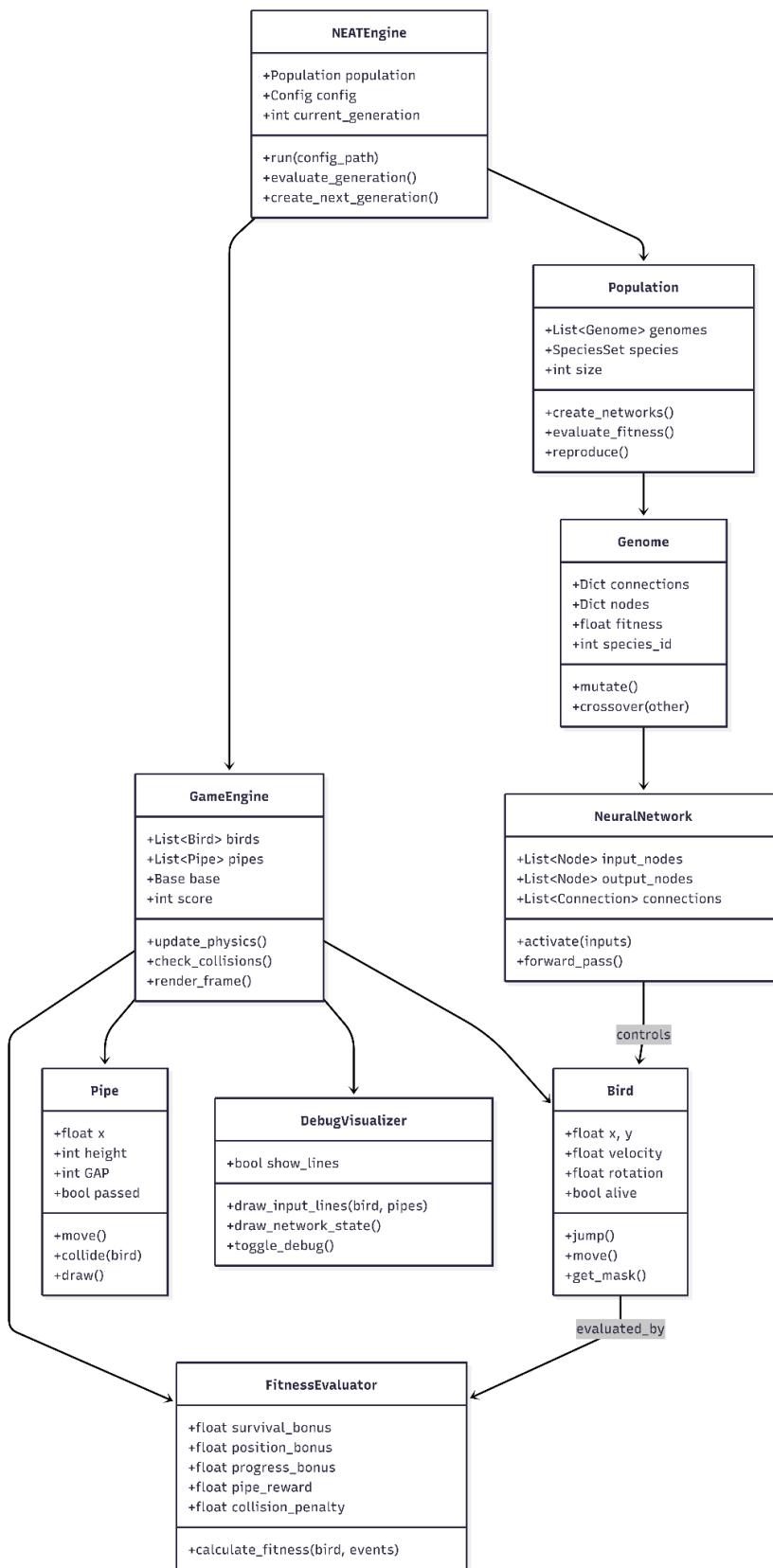
a. System Block Diagram / Architecture
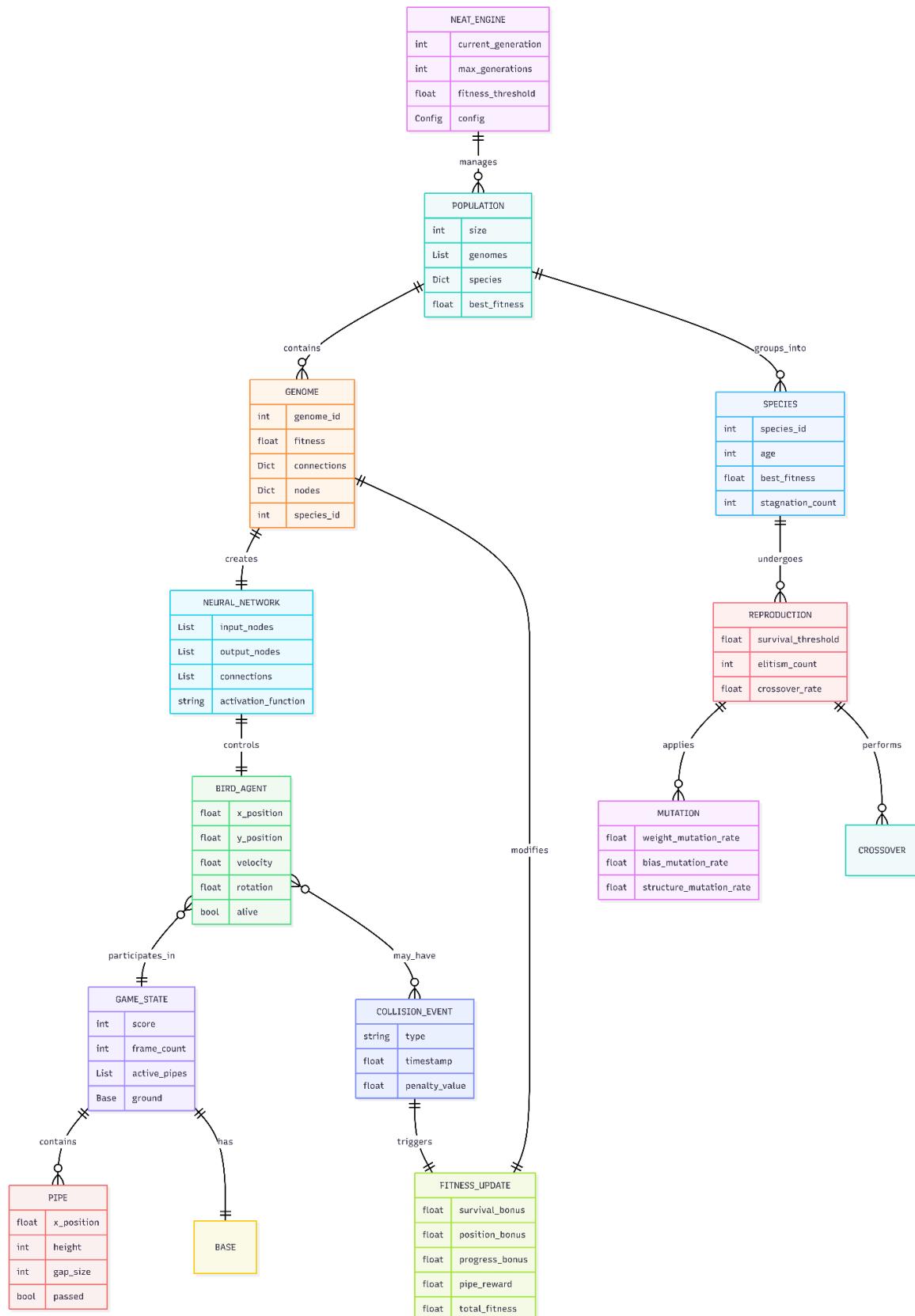
b. Data Flow Diagram

c. Component Architecture Diagram

NEATEngine

+Population population
+Config config
+int current_generation

+run(config_path)
+evaluate_generation()
+create_next_generation()

Population

+List<Genome> genomes
+SpeciesSet species
+int size

+create_networks()
+evaluate_fitness()
+reproduce()

Genome

+Dict connections
+Dict nodes
+float fitness
+int species_id

+mutate()
+crossover(other)

GameEngine

+List<Bird> birds
+List<Pipe> pipes
+Base base
+int score

+update_physics()
+check_collisions()
+render_frame()

NeuralNetwork

+List<Node> input_nodes
+List<Node> output_nodes
+List<Connection> connections

+activate(inputs)
+forward_pass()

controls

Pipe

+float x
+int height
+int GAP
+bool passed

+move()
+collide(bird)
+draw()

DebugVisualizer

+bool show_lines

+draw_input_lines(bird, pipes)
+draw_network_state()
+toggle_debug()

Bird

+float x, y
+float velocity
+float rotation
+bool alive

+jump()
+move()
+get_mask()

evaluated_by

FitnessEvaluator

+float survival_bonus
+float position_bonus
+float progress_bonus
+float pipe_reward
+float collision_penalty

+calculate_fitness(bird, events)

d. ERD/Flow Diagram

Explanation of Modules

1. ## NEAT Engine Module

   This module wraps the entire evolutionary process by initializing the population, managing generations, and determining termination conditions. It acts as the central controller that coordinates all other modules and ensures the genetic training cycle progresses correctly from evaluation to the next evolution.

2. ## Game Engine Module

   The physics simulation and core game mechanics, including object movement, collision detection, and frame rendering are the responsibilities of the Game Engine Module. It provides the interactive environment in which the AI agent operates and supplies visual feedback during gameplay.

3. ## Neural Network Module

   The feedforward neural networks derived from NEAT genomes are concerned in the Neural Network Module that is responsible for forward propagation of inputs, activation function processing, and decision-making based on fitness scores. It receives game state inputs from the environment and produces control outputs that drive the agent's next action ie. whether to jump or not jump.

4. ## Fitness Evaluator Module

   Calculation metrics for species in the generation are handled by The Fitness Evaluator Module, which tracks in-game progress, assigns rewards or penalties, and accumulates fitness scores. These scores are then used to assess each genome's performance during evolution.

5. ## Population Manager Module

   The Population Manager Module manages the collection of genomes and species by organizing species formation, handling selection, and coordinating reproduction. It works closely with the NEAT Engine to apply evolutionary operations, including mutation and crossover.

6. ## Visualization Module

   Real-time visual lines display information about the safe area of the bird and dynamic neurons being fed into the network, updating with each frame and producing console outputs. This helps monitor neural network behavior, agent inputs, and the overall training progress.

AI Concepts Used

The project implements NeuroEvolution of Augmenting Topologies (NEAT) to train an artificial agent to play Flappy Bird. NEAT combines evolutionary algorithms with neural networks, allowing both the network weights and topology to evolve over time. The following sections describe the neural network design, the NEAT algorithm configuration, the fitness function, and the training process.

1. Neural Network Architecture

The agent is controlled by a **feedforward neural network** generated from NEAT genomes. The network begins with a minimal structure and is allowed to grow in complexity through evolutionary processes.

1.1. Network Topology

The initial network consists of four input neurons, no hidden neurons, and one output neuron. Hidden neurons are not predefined and may emerge during evolution as required. The output neuron produces a continuous value that determines whether the bird performs a jump or not.

Input Layer	4 neurons (normalized game state inputs)
Hidden Layer	0 neurons initially (can evolve)
Output Layer	1 neuron (jump decision)
Architecture Type	Feedforward

1.2. Input Processing

Input 1: Bird Y Position (normalized)
├── Range: [0.0, 1.0]
├── Calculation: bird.y / WINDOW_HEIGHT
└── Purpose: Vertical position awareness

Input 2: Distance to Pipe Gap Center (normalized)
├── Range: [-1.0, 1.0]
├── Calculation: (bird.y - pipe_center) / (WINDOW_HEIGHT / 2)
└── Purpose: Gap alignment measurement

Input 3: Bird Velocity (normalized)
├── Range: [-1.0, 1.0]
├── Calculation: bird.velocity / 20.0
└── Purpose: Movement direction and speed

Input 4: Horizontal Distance to Pipe (normalized)
├── Range: [0.0, 1.0]
├── Calculation: (pipe.x - bird.x) / WINDOW_WIDTH
└── Purpose: Timing for decision making

1.3. Activation Function:

Function: Hyperbolic Tangent (tanh)
Range: [-1, 1]

1.4. Decision Logic:

if output[0] > 0.3: # Threshold-based decision
    bird.jump()
Else: # Let gravity take effect (no action)

1.5. Output Layer:

Output: Jump Decision
├── Activation: tanh function
├── Range: [-1.0, 1.0]
├── Threshold: 0.3
└── Action: if output > 0.3 then bird.jump()

2. NEAT Algorithm Details

The NEAT algorithm evolves both the structure and parameters of neural networks through population-based genetic optimization. Each generation comprises multiple genomes, each representing a distinct neural network.

2.1. Population-Based Evolution

The system keeps a population of 20 genomes for each generation. Genomes are automatically classified into species based on their structural similarity, using a compatibility threshold of 3.5 to define species boundaries. This speciation process helps safeguard innovation by enabling new structures to develop without facing immediate competition from more advanced networks. The evolutionary cycle continues for a configurable/custom number of generations.

2.2. Selection Mechanism

Selection is performed using fitness-proportionate selection combined with elitism. The top 25% of individuals in each generation survive to reproduce, and the two best genomes per species are preserved unchanged. This strategy ensures retention of high-performing solutions while maintaining genetic diversity and preventing premature convergence.

2.3. Genetic Operations

New offspring are generated through crossover and mutation. Crossover combines genetic material from two parent genomes while preserving valid network structures using innovation numbers to align genes correctly.

Mutation introduces controlled randomness into the population. Weight and bias mutations allow the addition of new nodes or connections. Connections may also be enabled or disabled, allowing networks to refine their topology over time.

2.4. Fitness Function Design

The fitness function is designed as a multi-component reward system that balances survival, positional stability, forward progress, and task completion. Agents receive small continuous rewards for staying alive, maintaining a safe vertical position, and moving forward. Successfully passing a pipe yields a substantial reward, reinforcing the game's primary objective.

Penalties are applied for collisions with pipes or game boundaries, providing strong negative reinforcement for failures. This shaping of the fitness landscape encourages safe flight behavior, longer gameplay, and progressive fitness score gain.

Continuous Rewards (per frame)
fitness += 0.1   # Base survival reward
fitness += 0.05   # Center position bonus (if |bird.y - center| < 100)
fitness += 0.02   # Forward progress reward

Event-Based Rewards
fitness += 15   # Pipe passage (major achievement)

Penalties
fitness -= 5   # Pipe collision
fitness -= 10   # Boundary collision (ground/ceiling)
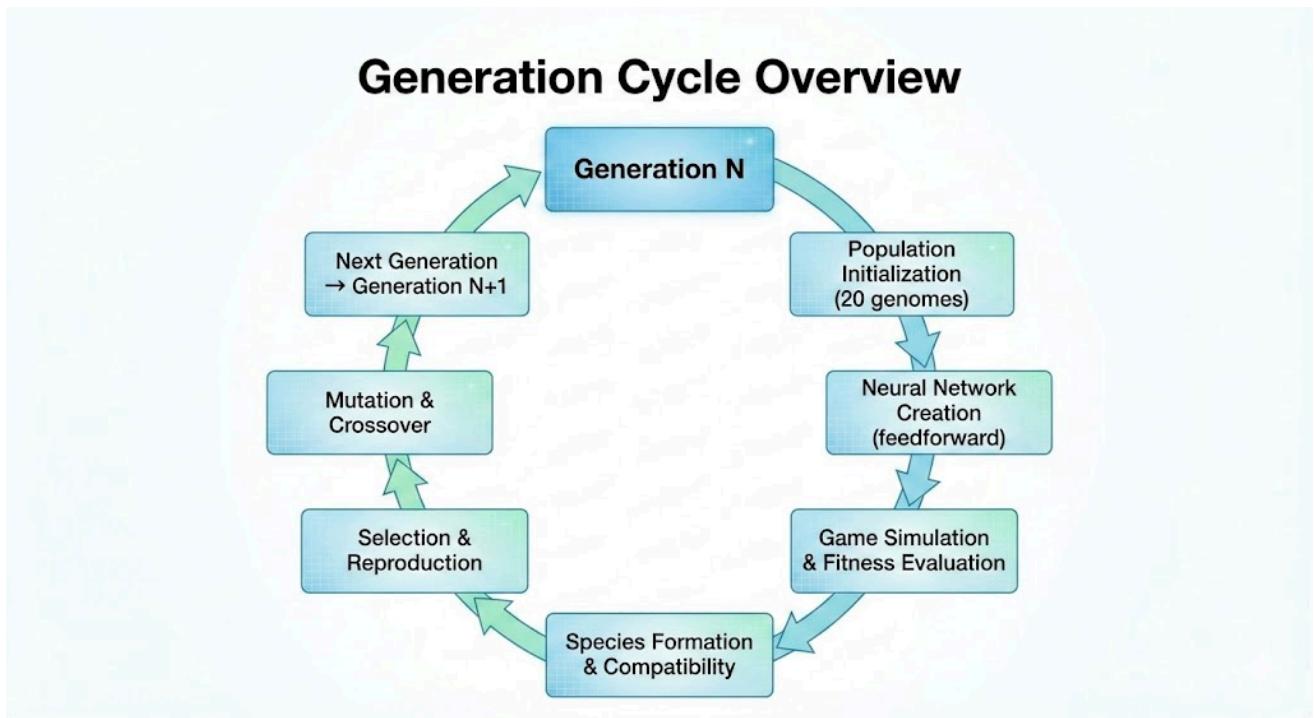
3. Model Training Process

The training process follows a standard NEAT evolutionary cycle in which neural networks are evaluated, selected, and evolved across generations.

3.1. Evolutionary Cycle

Each generation begins with the initialization or inheritance of genomes, followed by evaluation in the game environment. Fitness scores are computed for all individuals, and then selection, reproduction, and mutation are applied to form the next generation. This cycle repeats until convergence or a stopping condition is met.

1. *Initialization: Create a random population with minimal structure*
2. *Evaluation: Run each genome through a game simulation*
3. *Selection: Rank by fitness, select top performers*
4. *Reproduction: Crossover between high-fitness parents*
5. *Mutation: Apply random adjustments to offspring*
6. *Replacement: Form a new generation, repeat the cycle*

Generation Cycle Overview

- Generation N
- Population Initialization (20 genomes)
- Neural Network Creation (feedforward)
- Game Simulation & Fitness Evaluation
- Species Formation & Compatibility
- Selection & Reproduction
- Mutation & Crossover
- Next Generation → Generation N+1

3.2. Learning Progression

Early generations exhibit rudimentary behavior, primarily learning the distinction between jumping and falling. As evolution progresses, agents develop obstacle awareness and consistently avoid pipes. Later generations demonstrate optimized strategies, maintaining stable flight paths and achieving higher scores through refined timing and positioning.

3.3. Convergence Criteria

Training terminates when one of several conditions is satisfied: the best-performing genome achieves a fitness score of **300 or higher**, the maximum number of generations is reached, or population improvement stagnates. NEAT's built-in speciation mechanisms help manage stagnation by automatically disregarding.

Success: Best fitness ≥ 300 points
Timeout: Maximum generations reached
Stagnation: Species improvement (handled automatically)

Implementation details

a. Programming languages & tools

Language: Python 3.10

Dev Environment: Visual Studio Code with Python extensions and Git for source code management while pip for dependency installation

Library	Description
pygame	Game development and graphics rendering
neat	NEAT algorithm implementation
random	Random number generation for mutations
os	File system operations
time	Performance timing and delays

b. Modules description

1. Main Training Module (flappy_bird.py)

Component	Type	Description
Bird	Class	Physics-based bird agent
Pipe	Class	Obstacle generation and collision handling
Base	Class	Scrolling ground representation
main(genomes, config)	Function	NEAT evaluation function for genome fitness
run(config_path)	Function	Training orchestration and execution control

2. Human Game Module (game_py)

Component	Type	Description
Game	Class	Complete game state management
Bird	Class	Human-controlled bird physics

Pipe	Class	Obstacle system for human gameplay
main_game()	Function	Game loop with UI/UX handling

3. **Configuration Module (config-feedforward.txt)**

Section	Purpose
[NEAT]	Population size and evolutionary settings
[DefaultGenome]	Neural network topology and mutation rates
[DefaultSpeciesSet]	Species formation and compatibility parameters
[DefaultStagnation]	Detection of stagnant species
[DefaultReproduction]	Selection, crossover, and reproduction rules

c. **Code structure overview**

```
ai-agent-game-neat/
        ├── flappy_bird.py # Main AI training implementation
        ├── game.py          # Human-playable version
        ├── config-feedforward.txt     # NEAT algorithm configuration
        ├── imgs/                # Game assets directory
        │   ├── bird1.png, bird2.png, bird3.png
        │   ├── pipe.png, base.png, bg.png
        ├── README.md            # Project overview and setup
        └── high_score.json       # Human game high score storage
```

d. **Screenshots of code & output**

github.com/whizali/neat-flappy-bird-ai/ for the complete project

1. **Main Python Code**

```
"""
NEAT Flappy Bird AI Implementation
=================================
```

```python
This implementation uses NEAT (NeuroEvolution of Augmenting Topologies)
to train
AI agents to play Flappy Bird. The neural networks evolve over
generations to
learn optimal timing and decision-making strategies.

Key Features:
- Normalized neural network inputs for better learning
- Optimized fitness function with proper reward/penalty system
- Visual training to watch birds learn in real-time
- Robust error handling and resource management
"""


import pygame
import random
import os
import neat
import time


# Initialize pygame
pygame.init()
pygame.font.init()


# Game constants
WINDOW_WIDTH = 600
WINDOW_HEIGHT = 800


# Load game assets with error handling
try:
    BIRD_IMGS = [
        pygame.transform.scale2x(pygame.image.load(os.path.join("imgs",
"bird1.png"))),
        pygame.transform.scale2x(pygame.image.load(os.path.join("imgs",
"bird2.png"))),
        pygame.transform.scale2x(pygame.image.load(os.path.join("imgs",
"bird3.png")))
    ]
    PIPE_IMG =
pygame.transform.scale2x(pygame.image.load(os.path.join("imgs",
"pipe.png")))
```

```python
    BASE_IMG =
pygame.transform.scale2x(pygame.image.load(os.path.join("imgs",
"base.png")))
    BG_IMG =
pygame.transform.scale2x(pygame.image.load(os.path.join("imgs",
"bg.png")))
    STAT_FONT = pygame.font.SysFont("comicsans", 50)
except pygame.error as e:
    print(f"Error loading game assets: {e}")
    print("Make sure the 'imgs' folder contains: bird1.png, bird2.png,
bird3.png, pipe.png, base.png, bg.png")
    exit(1)


# Global settings - Visual training mode only
current_generation = 0
SHOW_DEBUG_LINES = True  # Set to False to hide neural network input
visualization


class Bird:
    """
    Bird class representing the player/AI agent.
    Handles physics, animation, and collision detection.
    """
    IMGS = BIRD_IMGS
    MAX_ROTATION = 25
    ROTATION_VELOCITY = 20
    ANIMATION_TIME = 5

    def __init__(self, x, y):
        """Initialize bird at given position."""
        self.x = x
        self.y = y
        self.tilt = 0
        self.tick_count = 0
        self.velocity = 0
        self.height = y
        self.img_count = 0
        self.img = self.IMGS[0]
```

```python
    def jump(self):
        """Make the bird jump (negative velocity = upward movement)."""
        self.velocity = -10.5
        self.tick_count = 0
        self.height = self.y


    def move(self):
        """Update bird position based on physics."""
        self.tick_count += 1

        # Physics calculation: velocity + gravity acceleration
        displacement = self.velocity * self.tick_count + 1.5 * \
self.tick_count**2

        # Terminal velocity (max fall speed)
        if displacement >= 16:
            displacement = 16
        # Reduce upward movement slightly for balance
        if displacement < 0:
            displacement -= 2

        self.y = self.y + displacement

        # Handle bird rotation based on movement direction
        if displacement < 0 or self.y < self.height + 50:
            if self.tilt < self.MAX_ROTATION:
                self.tilt = self.MAX_ROTATION
        else:
            if self.tilt > -90:
                self.tilt -= self.ROTATION_VELOCITY


    def draw(self, win):
        """Draw the bird with animation and rotation."""
        self.img_count += 1

        # Cycle through bird animation frames
        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]
```

```python
        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

        # Special case for steep dive
        if self.tilt <= -80:
            self.img = self.IMGS[1]
            self.img_count = self.ANIMATION_TIME * 2

        # Rotate and draw the bird
        rotated_image = pygame.transform.rotate(self.img, self.tilt)
        new_rect =
rotated_image.get_rect(center=self.img.get_rect(topleft=(self.x,
self.y)).center)
        win.blit(rotated_image, new_rect.topleft)

    def get_mask(self):
        """Get collision mask for pixel-perfect collision detection."""
        return pygame.mask.from_surface(self.img)


class Pipe:
    """
    Pipe class representing obstacles in the game.
    Handles movement, collision detection, and gap positioning.
    """
    GAP = 200   # Gap size between top and bottom pipes
    VEL = 5     # Horizontal movement speed

    def __init__(self, x):
        """Initialize pipe at given x position with random gap
height."""
        self.x = x
        self.height = 0
        self.top = 0
        self.bottom = 0
```

```python
        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):
        """Set random height for the pipe gap."""
        self.height = random.randrange(50, 450)
        self.top = self.height - self.PIPE_TOP.get_height()
        self.bottom = self.height + self.GAP

    def move(self):
        """Move pipe horizontally across the screen."""
        self.x -= self.VEL

    def draw(self, win):
        """Draw both top and bottom pipe segments."""
        win.blit(self.PIPE_TOP, (self.x, self.top))
        win.blit(self.PIPE_BOTTOM, (self.x, self.bottom))

    def collide(self, bird):
        """Check if bird collides with this pipe using pixel-perfect
detection."""
        bird_mask = bird.get_mask()
        top_mask = pygame.mask.from_surface(self.PIPE_TOP)
        bottom_mask = pygame.mask.from_surface(self.PIPE_BOTTOM)

        # Calculate offset positions for collision detection
        top_offset = (self.x - bird.x, self.top - round(bird.y))
        bottom_offset = (self.x - bird.x, self.bottom - round(bird.y))

        # Check for overlap with either pipe segment
        top_overlap = bird_mask.overlap(top_mask, top_offset)
        bottom_overlap = bird_mask.overlap(bottom_mask, bottom_offset)

        return top_overlap or bottom_overlap


class Base:
    """
```

```python
    Base class representing the scrolling ground.
    Creates illusion of continuous movement.
    """
    VEL = 5
    WIDTH = BASE_IMG.get_width()
    IMG = BASE_IMG

    def __init__(self, y):
        """Initialize base at given y position."""
        self.y = y
        self.x1 = 0
        self.x2 = self.WIDTH

    def move(self):
        """Move base segments to create scrolling effect."""
        self.x1 -= self.VEL
        self.x2 -= self.VEL

        # Reset positions when segments move off screen
        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):
        """Draw both base segments."""
        win.blit(self.IMG, (self.x1, self.y))
        win.blit(self.IMG, (self.x2, self.y))


def draw_debug_lines(win, bird, pipes, pipe_ind):
    """
    Draw red debug lines showing neural network inputs for a bird.
    """
    if not SHOW_DEBUG_LINES or not pipes or pipe_ind >= len(pipes):
        return

    # Colors for different input lines
    RED = (255, 0, 0)
    ORANGE = (255, 165, 0)
```

```python
    YELLOW = (255, 255, 0)
    GREEN = (0, 255, 0)

    current_pipe = pipes[pipe_ind]

    # Input 1: Bird Y position - horizontal line across screen
    pygame.draw.line(win, RED, (0, bird.y), (WINDOW_WIDTH, bird.y), 2)

    # Input 2: Distance to pipe gap center
    pipe_center_y = current_pipe.height + current_pipe.GAP / 2
    pygame.draw.line(win, ORANGE, (bird.x, bird.y), (bird.x,
pipe_center_y), 3)

    # Input 3: Bird velocity - vertical line showing velocity direction
    velocity_end_y = bird.y + (bird.velocity * 10)  # Scale for
visibility
    if bird.velocity < 0:  # Going up
        pygame.draw.line(win, YELLOW, (bird.x - 20, bird.y), (bird.x -
20, velocity_end_y), 3)
    else:  # Going down
        pygame.draw.line(win, YELLOW, (bird.x - 20, bird.y), (bird.x -
20, velocity_end_y), 3)

    # Input 4: Horizontal distance to pipe
    pygame.draw.line(win, GREEN, (bird.x, bird.y), (current_pipe.x,
bird.y), 2)

    # Draw pipe gap boundaries for reference
    pygame.draw.line(win, (100, 100, 255), (current_pipe.x,
current_pipe.height),
                     (current_pipe.x +
current_pipe.PIPE_TOP.get_width(), current_pipe.height), 2)
    pygame.draw.line(win, (100, 100, 255), (current_pipe.x,
current_pipe.height + current_pipe.GAP),
                     (current_pipe.x +
current_pipe.PIPE_TOP.get_width(), current_pipe.height +
current_pipe.GAP), 2)
```

```python
def draw_window(win, birds, pipes, base, score, generation=0,
pipe_ind=0):
    """
    Render the complete game window with all elements and debug
visualization.
    """
    # Draw background
    win.blit(BG_IMG, (0, 0))

    # Draw pipes
    for pipe in pipes:
        pipe.draw(win)

    # Draw debug lines for each bird (only show for first few birds to
avoid clutter)
    if SHOW_DEBUG_LINES and pipes:
        for i, bird in enumerate(birds[:5]):  # Show debug lines for
first 5 birds only
            draw_debug_lines(win, bird, pipes, pipe_ind)

    # Draw score
    text = STAT_FONT.render("Score: " + str(score), 1, (255, 255, 255))
    win.blit(text, (WINDOW_WIDTH - text.get_width() - 10, 10))

    # Draw generation and bird count
    gen_text = STAT_FONT.render("Gen: " + str(generation), 1, (255,
255, 255))
    win.blit(gen_text, (10, 10))

    bird_text = STAT_FONT.render("Birds: " + str(len(birds)), 1, (255,
255, 255))
    win.blit(bird_text, (10, 60))

    # Debug info
    if SHOW_DEBUG_LINES:
        debug_text = pygame.font.Font(None, 24).render("Debug Lines:
Red=Y pos, Orange=Gap dist, Yellow=Velocity, Green=Pipe dist", 1, (255,
255, 255))
        win.blit(debug_text, (10, 110))
```

```python
        # Draw base and birds
        base.draw(win)
        for bird in birds:
            bird.draw(win)

        pygame.display.update()


def main(genomes, config):
    """
    Main training function called by NEAT for each generation.
    Handles the complete game simulation and fitness evaluation.
    """
    global current_generation
    current_generation += 1

    # Initialize neural networks and birds for each genome
    nets = []
    ge = []
    birds = []

    for _, g in genomes:
        net = neat.nn.FeedForwardNetwork.create(g, config)
        nets.append(net)
        birds.append(Bird(230, 300))  # Start birds higher up
        g.fitness = 0
        ge.append(g)

    # Initialize game objects
    base = Base(730)
    pipes = [Pipe(600)]

    # Initialize display
    win = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))
    pygame.display.set_caption("NEAT Flappy Bird AI!")
    clock = pygame.time.Clock()
    score = 0

    # Main game loop
    run = True
```

```python
    while run and len(birds) > 0:
        clock.tick(30)  # 30 FPS for smooth visual learning

        # Handle pygame events
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_d:  # Press 'D' to toggle
debug lines
                    global SHOW_DEBUG_LINES
                    SHOW_DEBUG_LINES = not SHOW_DEBUG_LINES
                    print(f"Debug lines: {'ON' if SHOW_DEBUG_LINES else
'OFF'}")

        # Determine which pipe to focus on for AI input
        pipe_ind = 0
        if len(pipes) > 1 and birds[0].x > pipes[0].x +
pipes[0].PIPE_TOP.get_width():
            pipe_ind = 1

        # Update each bird and get AI decision
        for x, bird in enumerate(birds):
            bird.move()

            # Fitness rewards for survival and good positioning
            ge[x].fitness += 0.1  # Base survival reward

            # Bonus for staying near middle (avoid ground/ceiling)
            middle_y = WINDOW_HEIGHT / 2
            distance_from_middle = abs(bird.y - middle_y)
            if distance_from_middle < 100:
                ge[x].fitness += 0.05

            # Small bonus for forward progress
            ge[x].fitness += 0.02

            # Prepare normalized inputs for neural network
            bird_y_norm = bird.y / WINDOW_HEIGHT
```

```python
            pipe_center = pipes[pipe_ind].height + pipes[pipe_ind].GAP
/ 2
            distance_to_center = (bird.y - pipe_center) /
(WINDOW_HEIGHT / 2)
            velocity_norm = bird.velocity / 20.0
            horizontal_distance = (pipes[pipe_ind].x - bird.x) /
WINDOW_WIDTH

            # Get AI decision
            output = nets[x].activate((bird_y_norm, distance_to_center,
velocity_norm, horizontal_distance))

            # Jump if output exceeds threshold
            if output[0] > 0.3:
                bird.jump()

        # Handle pipe collision and passing
        add_pipe = False
        removed = []
        birds_to_remove = []

        for pipe in pipes:
            pipe_passed_by_any_bird = False

            for x, bird in enumerate(birds):
                # Check collision
                if pipe.collide(bird):
                    ge[x].fitness -= 5  # Collision penalty
                    birds_to_remove.append(x)

                # Check if bird passed pipe
                if not pipe.passed and pipe.x < bird.x:
                    pipe_passed_by_any_bird = True

            # Mark pipe as passed and trigger new pipe generation
            if pipe_passed_by_any_bird and not pipe.passed:
                pipe.passed = True
                add_pipe = True

            # Mark pipes for removal when off screen
```

```python
        if pipe.x + pipe.PIPE_TOP.get_width() < 0:
            removed.append(pipe)

    # Remove collided birds
    for x in reversed(birds_to_remove):
        birds.pop(x)
        nets.pop(x)
        ge.pop(x)

    # Move all pipes
    for pipe in pipes:
        pipe.move()

    # Add new pipe and reward all surviving birds
    if add_pipe:
        score += 1
        for g in ge:
            g.fitness += 15  # Big reward for passing pipe
        pipes.append(Pipe(600))

    # Remove off-screen pipes
    for r in removed:
        pipes.remove(r)

    # Check for boundary collisions (ground/ceiling)
    boundary_removals = []
    for x, bird in enumerate(birds):
        if bird.y + bird.img.get_height() >= 730 or bird.y < 0:
            ge[x].fitness -= 10  # Boundary collision penalty
            boundary_removals.append(x)

    # Remove birds that hit boundaries
    for x in reversed(boundary_removals):
            birds.pop(x)
            nets.pop(x)
            ge.pop(x)

    # Update base and render
    base.move()
```

```python
        draw_window(win, birds, pipes, base, score, current_generation,
pipe_ind)


def run(config_path):
    """
    Initialize and run the NEAT evolution process.
    """
    # Load NEAT configuration
    config = neat.config.Config(
        neat.DefaultGenome,
        neat.DefaultReproduction,
        neat.DefaultSpeciesSet,
        neat.DefaultStagnation,
        config_path
    )

    # Create population and add reporters
    p = neat.Population(config)
    p.add_reporter(neat.StdOutReporter(True))
    stats = neat.StatisticsReporter()
    p.add_reporter(stats)

    try:
        # Run evolution for 50 generations
        winner = p.run(main, 50)
        print(f"\nTraining completed! Best genome: {winner}")
    finally:
        # Clean up pygame resources
        if pygame.get_init():
            pygame.quit()


if __name__ == "__main__":
    local_dir = os.path.dirname(__file__)
    config_path = os.path.join(local_dir, "config-feedforward.txt")
    run(config_path)
```

2. Console Output Example

```
$ python flappy_bird.py
pygame 2.6.1 (SDL 2.28.4, Python 3.10.7)
Hello from the pygame community. https://www.pygame.org/contrib

 ****** Running generation 0 ******

Population's average fitness: -1.53200 stdev: 8.35240
Best fitness: 21.71000 - size: (1, 4) - species 1 - id 8
Average adjusted fitness: 0.175
Mean genetic distance 1.069, standard deviation 0.347
Population of 20 members in 1 species (after reproduction):
   ID   age  size   fitness   adj fit  stag
  ====  ===  ====  =========  =======  ====
    1    0    20    21.710     0.175     0
Total extinctions: 0
Generation time: 3.491 sec

 ****** Running generation 1 ******

Population's average fitness: 1.17300 stdev: 8.31431
Best fitness: 29.11000 - size: (1, 4) - species 1 - id 35
Average adjusted fitness: 0.215
Mean genetic distance 1.002, standard deviation 0.424
Population of 20 members in 1 species (after reproduction):
   ID   age  size   fitness   adj fit  stag
  ====  ===  ====  =========  =======  ====
    1    1    20    29.110     0.215     0
Total extinctions: 0
Generation time: 4.650 sec (4.071 average)

 ****** Running generation 2 ******

Population's average fitness: 3.23100 stdev: 10.23533
Best fitness: 35.50000 - size: (2, 5) - species 1 - id 48
Average adjusted fitness: 0.231
Mean genetic distance 0.998, standard deviation 0.374
Population of 20 members in 1 species (after reproduction):
   ID   age  size   fitness   adj fit  stag
  ====  ===  ====  =========  =======  ====
    1    2    20    35.500     0.231     0
Total extinctions: 0
Generation time: 5.061 sec (4.401 average)

 ****** Running generation 3 ******

Population's average fitness: 1.18400 stdev: 7.77500
Best fitness: 27.69000 - size: (1, 3) - species 1 - id 74
Average adjusted fitness: 0.232
Mean genetic distance 1.051, standard deviation 0.444
```
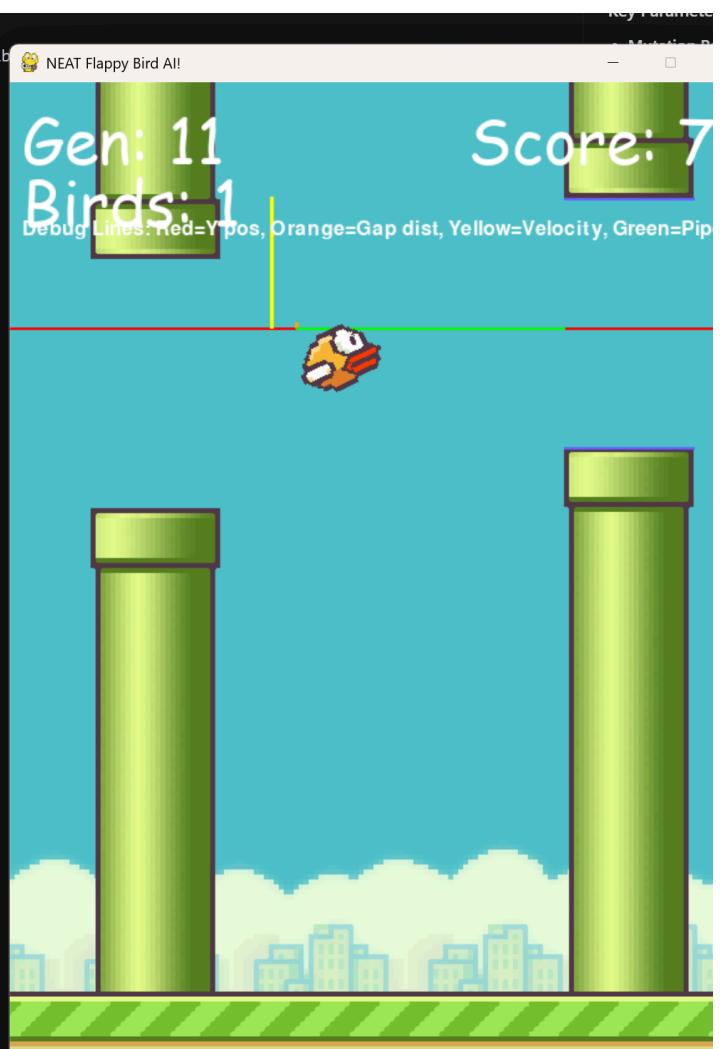

NEAT Flappy Bird AI!

Gen: 11
Birds: 1

Score: 7

Debug Lines: Red=Y pos, Orange=Gap dist, Yellow=Velocity, Green=Pip

```
 ****** Running generation 3 ******

Population's average fitness: 1.18400 stdev: 7.77500
Best fitness: 27.69000 - size: (1, 3) - species 1 - id 74
Average adjusted fitness: 0.232
Mean genetic distance 1.051, standard deviation 0.444
Population of 20 members in 1 species (after reproduction):
   ID   age  size   fitness   adj fit  stag
  ====  ===  ====  =========  =======  ====
    1    3    20     27.690    0.232     1
Total extinctions: 0
Generation time: 4.957 sec (4.540 average)

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

Population's average fitness: 7.19300 stdev: 25.32099
Best fitness: 111.14000 - size: (2, 4) - species 1 - id 77
Average adjusted fitness: 0.116
Mean genetic distance 0.963, standard deviation 0.470
Population of 20 members in 1 species (after reproduction):
   ID   age  size   fitness   adj fit  stag
  ====  ===  ====  =========  =======  ====
    1    4    20    111.140    0.116     0
Total extinctions: 0
Generation time: 12.351 sec (6.102 average)

 ****** Running generation 5 ******

Population's average fitness: 5.51800 stdev: 10.57639
Best fitness: 28.53000 - size: (1, 3) - species 1 - id 108
Average adjusted fitness: 0.349
Mean genetic distance 0.858, standard deviation 0.560
Population of 20 members in 1 species (after reproduction):
   ID   age  size   fitness   adj fit  stag
  ====  ===  ====  =========  =======  ====
    1    5    20     28.530    0.349     1
Total extinctions: 0
Generation time: 4.746 sec (5.876 average)

 ****** Running generation 6 ******

Population's average fitness: 8.02800 stdev: 19.29221
Best fitness: 52.97000 - size: (1, 3) - species 1 - id 108
Average adjusted fitness: 0.240
Mean genetic distance 0.728, standard deviation 0.325
Population of 20 members in 1 species (after reproduction):
   ID   age  size   fitness   adj fit  stag
  ====  ===  ====  =========  =======  ====
```
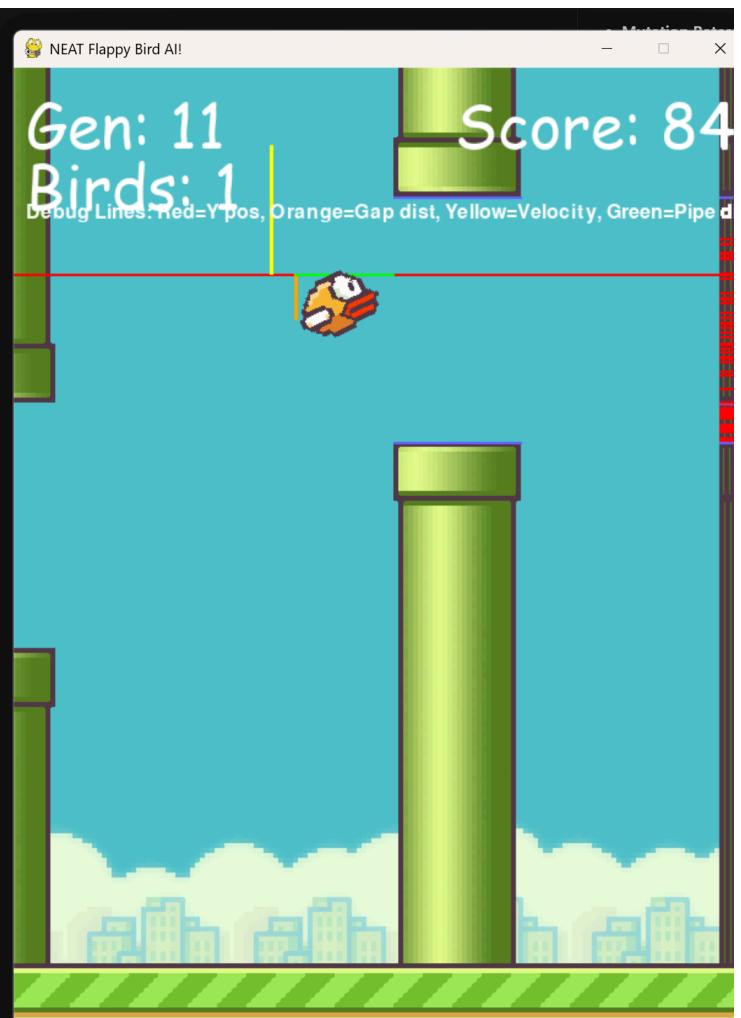
NEAT Flappy Bird AI!

Gen: 11
Birds: 1

Score: 84

Debug Lines: Red=Y pos, Orange=Gap dist, Yellow=Velocity, Green=Pipe d

```
 ****** Running generation 6 ******

Population's average fitness: 8.02800 stdev: 19.29221
Best fitness: 52.97000 - size: (1, 3) - species 1 - id 108
Average adjusted fitness: 0.240
Mean genetic distance 0.728, standard deviation 0.325
Population of 20 members in 1 species (after reproduction):
   ID   age  size   fitness   adj fit  stag
  ====  ===  ====  =========  =======  ====
    1    6    20     52.970    0.240     2
Total extinctions: 0
Generation time: 7.275 sec (6.076 average)

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

Population's average fitness: 2.12750 stdev: 8.71465
Best fitness: 33.08000 - size: (1, 3) - species 1 - id 131
Average adjusted fitness: 0.217
Mean genetic distance 0.626, standard deviation 0.226
Population of 20 members in 1 species (after reproduction):
   ID   age  size   fitness   adj fit  stag
  ====  ===  ====  =========  =======  ====
    1    7    20     33.080    0.217     3
Total extinctions: 0
Generation time: 4.680 sec (5.901 average)

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

Population's average fitness: 7.38650 stdev: 9.41124
Best fitness: 30.32000 - size: (1, 3) - species 1 - id 150
Average adjusted fitness: 0.344
Mean genetic distance 0.680, standard deviation 0.313
Population of 20 members in 1 species (after reproduction):
   ID   age  size   fitness   adj fit  stag
  ====  ===  ====  =========  =======  ====
    1    8    20     30.320    0.344     4
Total extinctions: 0
Generation time: 5.089 sec (5.811 average)

 ****** Running generation 9 ******

Population's average fitness: 4.51750 stdev: 7.89456
Best fitness: 23.77000 - size: (2, 4) - species 1 - id 180
Average adjusted fitness: 0.363
Mean genetic distance 0.934, standard deviation 0.457
Population of 20 members in 1 species (after reproduction):
   ID   age  size   fitness   adj fit  stag
  ====  ===  ====  =========  =======  ====
    1    9    20     23.770    0.363     5
```

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

Population's average fitness: 145.17400 stdev: 585.94621
Best fitness: 2697.82000 - size: (1, 3) - species 1 - id 191

Best individual in generation 10 meets fitness threshold - complexity: (1, 3)

Training completed! Best genome: Key: 191
Fitness: 2697.819999999571
Nodes:
        0 DefaultNodeGene(key=0, bias=-0.16865887198244095, response=0.9864401827580658, activation=tanh, aggregation=sum
Connections:
        DefaultConnectionGene(key=(-3, 0), innovation=3, weight=-0.0611468717269415, enabled=True)
        DefaultConnectionGene(key=(-2, 0), innovation=2, weight=1.9285157706626794, enabled=True)
        DefaultConnectionGene(key=(-1, 0), innovation=1, weight=0.4363809576376245, enabled=True)
(venv)
aliha@DESKTOP-QNENFCL MINGW64 ~/OneDrive/Desktop/ai-agent-game-neat (main)
$
```

Explanation of Important Functions

1. NEAT Integration Functions

Function: main(genomes, config)
Acts as the core NEAT evaluation function, executed once per generation. It initializes neural networks from genomes, simulates the game, evaluates agent performance, and updates fitness scores.

a. Initialization Phase

Neural networks are created from each genome, and corresponding birds are spawned at a fixed starting position. Fitness values are initialized for evolutionary tracking.

```
# Create neural networks from genomes
for _, g in genomes:
    net = neat.nn.FeedForwardNetwork.create(g, config)
    nets.append(net)
    birds.append(Bird(230, 300))  # Starting position
    g.fitness = 0             # Initialize fitness
    ge.append(g)                # Store genome reference
```

b. Game Objects Setup

Initial game entities are created, including the base (ground), the first pipe, and the score counter.

```
base = Base(730)        # Ground at y = 730
pipes = [Pipe(600)]     # Initial pipe at x = 600
score = 0             # Score counter
```

c. Main Game Loop (30 FPS)

The simulation runs at a fixed frame rate while handling user input, updating game physics, and evaluating neural network outputs.

I. Event Handling
```
# Quit detection
if event.type == pygame.QUIT:
    run = False

# Debug toggle (D key)
if event.key == pygame.K_d:
    SHOW_DEBUG_LINES = not SHOW_DEBUG_LINES
```

II. Pipe Selection Logic

The active pipe is selected based on the bird's horizontal position to ensure correct distance calculations.

```
pipe_ind = 0  # Default to first pipe

# Switch to second pipe once the bird passes the first
if len(pipes) > 1 and birds[0].x > pipes[0].x + pipes[0].PIPE_TOP.get_width():
    pipe_ind = 1
```

d. **Neural Network Processing**

Input Normalization: All inputs are normalized to improve training stability and convergence.

```
bird_y_norm = bird.y / WINDOW_HEIGHT
pipe_center = pipes[pipe_ind].height + pipes[pipe_ind].GAP / 2
distance_to_center = (bird.y - pipe_center) / (WINDOW_HEIGHT / 2)
velocity_norm = bird.velocity / 20.0
horizontal_distance = (pipes[pipe_ind].x - bird.x) / WINDOW_WIDTH
```

Input Ranges:

- bird_y_norm: [0, 1]
- distance_to_center: [-1, 1]
- velocity_norm: approximately [-1, 1]
- horizontal_distance: [0, 1]

Decision-making: The neural network output determines whether the bird jumps.

```
output = nets[x].activate(
    (bird_y_norm, distance_to_center, velocity_norm, horizontal_distance)
)
if output[0] > 0.3:  # Jump threshold
    bird.jump()
```

2. Fitness Function Implementation

Fitness values guide the evolutionary process by rewarding survival, progress, and successful navigation.

Continuous Rewards (Per Frame)

```
ge[x].fitness += 0.1  # Survival reward
# Center alignment bonus
middle_y = WINDOW_HEIGHT / 2
distance_from_middle = abs(bird.y - middle_y)
if distance_from_middle < 100:
    ge[x].fitness += 0.05

ge[x].fitness += 0.02  # Forward movement reward
```

Event-Based Rewards and Penalties

```
# Reward for passing a pipe
if pipe_passed_by_any_bird:
```

```
for g in ge:
    g.fitness += 15

# Collision penalties
if pipe.collide(bird):
    ge[x].fitness -= 5

# Boundary collision penalty
if bird.y >= 730 or bird.y < 0:
    ge[x].fitness -= 10
```

3. Bird Management

Birds that collide with obstacles or boundaries are safely removed from all tracking structures.

```
# Collect indices of birds to remove
birds_to_remove = []
boundary_removals = []

# Remove in reverse order to preserve indexing
for x in reversed(birds_to_remove):
    birds.pop(x)
    nets.pop(x)
    ge.pop(x)
```

4. NEAT Configuration Loading

run(config_path) Initializes NEAT configuration, manages population evolution, and executes the training process.

```
config = neat.config.Config(
    neat.DefaultGenome,
    neat.DefaultReproduction,
    neat.DefaultSpeciesSet,
    neat.DefaultStagnation,
    config_path)
```

5. Population Management

```
p = neat.Population(config)
p.add_reporter(neat.StdOutReporter(True))

stats = neat.StatisticsReporter()
p.add_reporter(stats)
```

6. Evolution Execution

The NEAT algorithm is executed for a maximum of 50 generations.

```
winner = p.run(main, 50)
```

7. Resource Cleanup

Ensures graceful shutdown of game resources.

```
finally:
    if pygame.get_init():
        pygame.quit()
```

Testing & Evaluation

Successful Training upto (11 Generations (0-10))
Final Best Fitness: 2,697.82 (Generation 10)
Total Runtime: ~23.5 seconds
Outcome: Fitness threshold (300) exceeded — **winner found!!**

Generation Statistics Table

Gen	Avg Fitness	Std Dev	Best Fitness	Best Genome ID	Network Size	Species	Species Details	Adj. Fitness	Genetic Distance (Mean ± SD)	Gen Time	Total Time
0	-1.532	8.352	21.710	8	(1, 4)	1	ID:1, Age:0, Size:20, Fit:21.710, Adj:0.175, Stag:0	0.175	1.069 ± 0.347	3.491s	3.491s
1	1.173	8.314	29.110	35	(1, 4)	1	ID:1, Age:1, Size:20, Fit:29.110, Adj:0.215, Stag:0	0.215	1.002 ± 0.424	4.650s	8.141s
2	3.231	10.235	35.500	48	(2, 5)	1	ID:1, Age:2, Size:20, Fit:35.500,	0.231	0.998 ± 0.374	5.061s	13.202s

							Adj:0.231, Stag:0				
3	1.184	7.775	27.690	74	(1, 3)	1	ID:1, Age:3, Size:20, Fit:27.690, Adj:0.232, Stag:1	0.232	1.051 ± 0.444	4.957s	18.159s
4	7.193	25.320	111.140	77	(2, 4)	1	ID:1, Age:4, Size:20, Fit:111.140, Adj:0.116, Stag:0	0.116	0.963 ± 0.470	12.351s	30.510s
5	5.518	10.576	28.530	108	(1, 3)	1	ID:1, Age:5, Size:20, Fit:28.530, Adj:0.349, Stag:1	0.349	0.858 ± 0.560	4.746s	35.256s
6	8.028	19.292	52.970	108	(1, 3)	1	ID:1, Age:6, Size:20, Fit:52.970, Adj:0.240, Stag:2	0.240	0.728 ± 0.325	7.275s	42.531s
7	2.127	8.714	33.080	131	(1, 3)	1	ID:1, Age:7, Size:20, Fit:33.080, Adj:0.217, Stag:3	0.217	0.626 ± 0.226	4.680s	47.211s
8	7.386	9.411	30.320	150	(1, 3)	1	ID:1, Age:8, Size:20, Fit:30.320, Adj:0.344, Stag:4	0.344	0.680 ± 0.313	5.089s	52.300s

| 9 | 4.517 | 7.894 | 23.770 | 180 | (2, 4) | 1 | ID:1, Age:9, Size:20, Fit:23.770, Adj:0.363, Stag:5 | 0.363 | 0.934 ± 0.457 | 2.751s | 55.051s |
| 10 | **145.174** | **585.946** | **2,697.820** | **191** | (1, 3) | 1 | ID:1, Age:10, Size:20, Fit:2697.820, Adj:N/A, Stag:N/A | N/A | N/A | 2.751s | **57.802s** |

Limitations

Implementing the NEAT algorithm to train an AI agent for Flappy Bird presents several challenges and constraints, which can be broadly categorized into computational, algorithmic, environmental, and implementation factors.

1. **Computational Challenges due to Resource Intensity:**

   Evolving neural networks using NEAT is computationally demanding, particularly as networks become more complex across generations. This can result in longer processing times and higher memory usage, potentially bringing performance overhead.

2. **Algorithmic Constraints**

   NEAT may struggle when handling high-dimensional inputs, as designing well-tuned networks becomes increasingly complex in such scenarios. Developing an effective fitness function that accurately evaluates the AI agent's performance is challenging. The function must balance short-term rewards with long-term objectives to guide the evolutionary process effectively.

3. **Environmental Challenges**

   The Flappy Bird environment is highly dynamic, with pipes appearing at varying positions and intervals. The AI must make rapid decisions to navigate obstacles successfully, necessitating efficient neural network architectures that can process inputs and produce outputs within quick time constraints.

4. **FeedForward Implementation Constraints**

   Selecting optimal parameter values, such as population size, mutation rate, and crossover rate, is critical. Improper tuning can lead to premature convergence or high computational cost.

Conclusion

The project successfully uses NEAT, a genetic evolutionary algorithm, to train an AI agent for dynamic, real-time decision-making in the game environment. Over successive generations, the neural network improved its ability to navigate the game's pipes without colliding, achieving higher fitness scores and better performance. This shows NEAT's strengths by allowing the AI to manage collisions without supervision or large prior datasets. Despite challenges like high computational demands and the need for careful tuning of the feedforward setup, the results show that the algorithm is effective for autonomous game playing.

References

[1] NEAT-Python Documentation, *Python implementation of NEAT*:
https://neat-python.readthedocs.io/

[2] Pygame Documentation, *Python game development library*: https://www.pygame.org/docs/

[3] Python Software Foundation, *Python 3.10 Documentation*: https://docs.python.org/3.10/

[4] K. O. Stanley, *NEAT (NeuroEvolution of Augmenting Topologies)*
https://www.cs.ucf.edu/~kstanley/neat.html

[5] Neuroevolution of Augmenting Topologies Demos (2003) https://nn.cs.utexas.edu/?neatdemo

Appendix / Source Code

Repository URL	https://github.com/whizali/neat-flappy-bird-ai
Branch	main

Project Landing Page / Site

Live Demo	https://whizali.github.io/neat-flappy-bird-ai

END