

FlightX

An Evolutionary AI-Based Flight Simulation System

A Project Report Submitted in Partial Fulfillment
of the Requirements for the Degree of

Bachelor of Science in Computer Science and Engineering

Submitted by
Farhan Ishraq

Department of Computer Science and Engineering

Submitted to
Department of Computer Science and Engineering

December 22, 2025

Abstract

FlightX is a two-dimensional autonomous flight simulation that demonstrates artificial intelligence learning through evolutionary computation. The system consists of multiple AI-controlled agents that learn to navigate through dynamically generated obstacles using a neural network whose weights evolve over generations. Unlike conventional machine learning systems, the neural networks in FlightX are implemented entirely from scratch without relying on any external AI or deep learning libraries. Learning emerges through mutation, selection, and survival-based fitness evaluation. The project integrates real-time visualization, interactive control panels, and performance analytics, making it suitable for academic study, experimentation, and demonstration of artificial intelligence principles.

Chapter 1

Introduction

Artificial Intelligence (AI) systems are increasingly required to operate in uncertain and dynamic environments. Traditional supervised learning techniques often depend on labeled datasets, which are unavailable in real-time simulations. Evolutionary algorithms provide an alternative approach where agents learn optimal behaviors through interaction with the environment rather than explicit instruction.

FlightX explores this paradigm by implementing an autonomous flight simulation inspired by obstacle-avoidance games. Each agent learns how to survive longer by evolving its internal decision-making mechanism over multiple generations. The system emphasizes transparency by exposing neural decisions, environmental sensing, and evolutionary progression through visual feedback and analytics.

Chapter 2

Problem Statement

The objective of this project is to design and implement a simulation environment in which autonomous agents can:

- Perceive their environment using limited sensory input
- Make real-time decisions based on neural computation
- Learn optimal behaviors without supervised data
- Improve performance over generations

The challenge lies in achieving meaningful learning using minimal neural structures while maintaining computational efficiency and interpretability.

Chapter 3

System Architecture

3.1 High-Level Design

The FlightX system is composed of four tightly integrated subsystems:

1. Simulation Environment
2. AI Agent and Neural Controller
3. Evolutionary Learning Engine
4. User Interface and Analytics



This loop executes continuously during runtime, allowing agents to react and adapt in real time.

Chapter 4

Simulation Environment

4.1 World Representation

The simulation consists of:

- A horizontal scrolling space
- Vertically aligned pipe obstacles
- A ground boundary and sky ceiling

Obstacles are procedurally generated to ensure variability and prevent memorization.

4.2 Physics and Collision

Agents are affected by gravity and velocity constraints. Collisions with pipes, ground, or the upper boundary immediately terminate the agent's lifespan.

Chapter 5

AI Agent Design

5.1 Agent Structure

Each agent represents a single autonomous entity and contains:

- Physical state (position, velocity)
- Sensory system (vision)
- Neural decision-maker (brain)
- Fitness evaluator

5.2 Vision System

The agent perceives only the closest obstacle using four normalized inputs:

1. Vertical offset from the pipe gap center
2. Horizontal distance to the next pipe
3. Reserved input for extensibility
4. Current vertical velocity

Normalization ensures scale invariance and stabilizes learning.

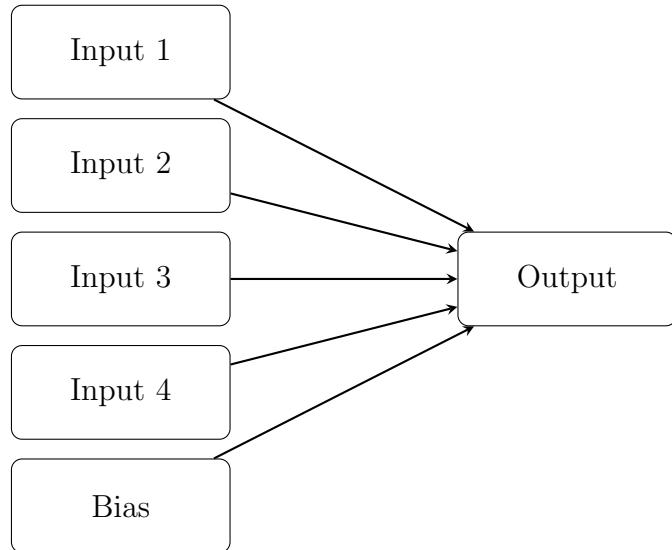
Chapter 6

Neural Network Implementation

6.1 Architecture

The neural network consists of:

- Input layer (4 neurons)
- Bias neuron
- Single output neuron



6.2 Activation Function

The output neuron uses a sigmoid activation function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

This bounds decisions between 0 and 1, simplifying action thresholds.

6.3 Decision Logic

- Output > 0.55 : upward flap
- Output < 0.45 : accelerated descent
- Otherwise: passive glide

Chapter 7

Evolutionary Learning System

7.1 Fitness Evaluation

Fitness is calculated as:

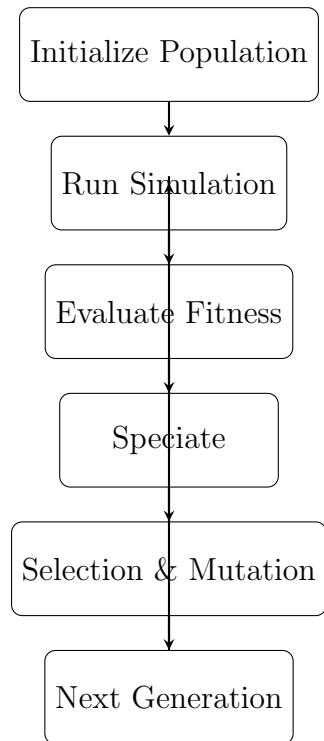
$$Fitness = 1.5 \times Lifespan - 100 \times |VerticalOffset|$$

This formulation rewards both survival time and flight stability.

7.2 Speciation

Agents are grouped into species based on neural weight similarity. This preserves diversity and protects novel solutions from premature elimination.

7.3 Evolution Cycle



Chapter 8

Implementation Details

8.1 Core Modules

Each file in the project has a clear responsibility:

- **node.py**: Implements neural neurons
- **connection.py**: Implements weighted synapses
- **brain.py**: Feed-forward neural network
- **player.py**: AI agent logic
- **population.py**: Evolution control
- **species.py**: Diversity management
- **components.py**: Environment objects
- **menu_buttons.py**: UI system
- **config.py**: Shared configuration
- **main.py**: Main simulation loop

Chapter 9

Results and Discussion

Experimental results demonstrate:

- Progressive improvement in survival time
- Smoother flight trajectories
- Reduced collision frequency

Performance graphs show consistent learning trends, confirming effective evolution.

Chapter 10

Conclusion and Future Work

FlightX demonstrates that meaningful intelligent behavior can emerge from simple neural structures when combined with evolutionary pressure. The project successfully implements an AI learning system from first principles without external machine learning frameworks.

10.1 Future Enhancements

- Dynamic neural topology evolution
- Recurrent neural networks
- Replay-based visualization
- Multi-agent comparison dashboards

Chapter 11

Algorithmic Design and Implementation

This chapter presents the core algorithms used in FlightX. Each algorithm is described using formal pseudocode, followed by an explanation of its purpose and the corresponding source files responsible for its implementation. All algorithms are implemented from scratch without the use of external artificial intelligence or machine learning libraries.

11.1 Algorithm 1: Main Simulation Loop

Associated Files:

- `main.py`
- `population.py`
- `player.py`

Purpose: Controls the real-time execution of the simulation, including agent updates, rendering, collision detection, and generation transitions.

[1] Initialize simulation window
Initialize population with N agents
simulation is running
agent a in population a is alive
 a observes environment
 a computes neural decision
 a performs action
Update physics and position
collision detected
Mark a as dead
all agents are dead
Perform evolutionary selection
Generate next generation
Render environment and agents

11.2 Algorithm 2: Agent Environment Perception

Associated Files:

- `player.py`

- `components.py`

Purpose: Provides limited sensory input to the agent by detecting the nearest obstacle and extracting normalized environmental features.

[1] Agent position, obstacle positions Identify nearest upcoming obstacle Compute vertical distance to obstacle gap Compute horizontal distance to obstacle Read current vertical velocity Normalize all inputs Store values as neural inputs

11.3 Algorithm 3: Neural Network Forward Propagation

Associated Files:

- `brain.py`
- `node.py`
- `connection.py`

Purpose: Computes the agent's decision using a custom feed-forward neural network with weighted connections and sigmoid activation.

[1] Input vector $X = \{x_1, x_2, x_3, x_4\}$ Add bias input $x_0 = 1$ connections ($i \rightarrow o$)
 $sum \leftarrow sum + weight_{i,o} \times x_i$ $output \leftarrow \sigma(sum)$ $output > \theta_{jump}$ Perform upward flap
 Apply gravity

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

11.4 Algorithm 4: Fitness Evaluation

Associated Files:

- `player.py`

Purpose: Quantifies agent performance based on survival duration and flight stability to guide evolutionary selection.

[1] Agent lifespan, flight deviation $fitness \leftarrow lifespan \times \alpha$ $fitness \leftarrow fitness - |vertical_deviation| \times \beta$ Assign fitness score to agent

11.5 Algorithm 5: Speciation

Associated Files:

- `species.py`
- `population.py`

Purpose: Groups agents into species based on neural similarity to preserve genetic diversity and prevent premature convergence.

[1] agents in population $assigned \leftarrow false$ species s genetic similarity(agent, s) $>$ threshold Assign agent to species s $assigned \leftarrow true$ Break $assigned = false$ Create new species with agent

11.6 Algorithm 6: Evolutionary Selection and Reproduction

Associated Files:

- `population.py`
- `species.py`

Purpose: Controls survival, reproduction, extinction, and generation progression using evolutionary pressure.

[1] Group agents into species Compute average fitness per species Remove extinct species Remove stale species Sort species by fitness Preserve champion from each species population size $<$ target Select parent species Generate offspring via mutation Add offspring to population Increment generation count

11.7 Algorithm 7: Neural Mutation Operator

Associated Files:

- `brain.py`
- `connection.py`

Purpose: Introduces controlled randomness into neural weights to explore new behaviors while maintaining learned traits.

[1] connections in neural network $random() < mutation_rate$ $weight \leftarrow weight + random(-\delta, +\delta)$

Summary and Key Statement

The AI agent in FlightX operates using a fully custom neural network architecture. Learning does not rely on backpropagation, gradient descent, or labeled datasets. Instead, intelligent behavior emerges through evolutionary pressure, where agents with superior performance survive and propagate their neural parameters across generations. Every component of the artificial intelligence system — including neurons, synapses, fitness evaluation, mutation, and speciation — has been implemented entirely from scratch.

Acknowledgment

The author acknowledges the use of Python, Pygame, and Matplotlib for enabling rapid simulation development and visualization.