Project Documentation Outline: Flappy Bird NEAT AI

1. Introduction

- Brief description of the project.
- What is Flappy Bird?
- Why AI (NEAT) is used.
- Project goal: *train AI birds using NeuroEvolution to play Flappy Bird automatically.*

2. Tools & Technologies

- **Unity Engine** (for game environment, physics, rendering).
- **C#** (game scripts, AI integration).
- **NEAT-inspired Genetic Algorithm** (custom implementation).
- **Visual Studio Code / Rider / Visual Studio** (IDE).
- **Version Control (Git/GitHub)** *(optional but recommended)*.

3. Game Setup

3.1 Core Mechanics

- Bird physics (gravity, flap strength).
- Pipes spawning & scrolling.
- Collision detection (obstacle vs scoring trigger).
- Scoring system.

3.2 Player Control

- Human Player script (Player.cs).
- Transition to AI-controlled birds.

4. AI Design

4.1 Neural Network

- Input layer (Bird Y, distance to pipe, gap height, velocity).
- Hidden layer(s).

- Output layer (flap or not).
- FeedForward function.

4.2 Genetic Algorithm

- Population initialization.
- Fitness function (pipes passed + survival time).
- Selection (elitism, roulette selection).
- Crossover.
- Mutation (rate, strength).

4.3 BirdAI Integration

- Bird controlled by NN instead of player input.
- Data flow: Environment → Inputs → NN → Flap decision.

5. Evolution Manager

- Population management.
- Spawning birds.
- Tracking deaths & fitness.
- Evolving new generation.
- Saving/Loading best networks (bestBrain.json).

6. UI & Visualization

- Canvas elements:
    - Score (pipes passed).
    - Generation number.
    - Alive count.
- Replay Mode (show the best bird).
- Training speed control (timeScale slider).

7. Training Process

7.1 Training Parameters

- Population size.
- Mutation rate.
- Hidden neurons.
- Elite count.

7.2 Observations

- Generations evolve over time.
- Birds learn to hover, then to pass first pipe, eventually master the game.

7.3 Challenges

- Balancing fitness (timeAlive vs pipesPassed).
- Preventing “lazy birds” (hovering forever).
- Mutation tuning.

8. Results

- Screenshots / videos of training progress.
- Fitness graph per generation.
- Replay best bird performance.
- Discussion of how many generations it took to get good performance.

9. Future Improvements

- Better visualization (real-time graph of fitness).
- More advanced AI (Unity ML-Agents, Deep Q-Learning).
- Extend to other games (Snake, Dino, Pong).
- Optimize training speed (headless mode, batch runs).

10. Conclusion

- Summary of what was achieved.
- NEAT’s effectiveness for simple control tasks.
- Value of Unity as a simulation environment for AI.

1. Introduction

This project is a recreation of the classic *Flappy Bird* game, enhanced with an Artificial Intelligence system that learns to play automatically. The original Flappy Bird was a simple but highly addictive mobile game where the player controls a bird, tapping to make it flap upwards while avoiding pipes.

In this version, the challenge is not just playing the game but teaching a computer agent to **learn the game mechanics** through **NeuroEvolution of Augmenting Topologies (NEAT)** principles. Instead of being explicitly programmed with the rules to win, the AI evolves its own strategy by trial and error, improving over multiple generations.

The main goal of the project is to:

- Build the Flappy Bird environment in Unity.
- Implement a neural network–based bird controller.
- Train birds using evolutionary algorithms (selection, crossover, mutation).
- Visualize how the AI gradually learns to survive longer and pass more pipes.

This makes the project both a **fun game clone** and an **AI experiment**, showing how evolutionary computation can produce intelligent behavior in dynamic environments.

---

2. Tools & Technologies

Unity Engine

Unity is the main development platform used to build the game environment. It provides:

- **Physics simulation** (gravity, collisions).
- **2D rendering** (sprites, animations, background).
- **Scene management** (pipes, ground, game loop).
- **UI system** for showing score, generation, and alive bird count.

Unity's flexibility makes it perfect for both game logic and embedding AI systems.

C# Programming Language

All game scripts and AI code are written in **C#**, Unity's primary scripting language.
C# is used for:

- Controlling bird physics (gravity, flapping).
- Managing pipes and scoring.
- Implementing the neural network and genetic algorithm logic.
- Connecting AI decisions to the Unity game world.

Neural Network & NEAT-inspired Algorithm

The AI birds are controlled by a lightweight **feedforward neural network**.
Inputs → [Hidden Layer] → Output

- **Inputs**: Bird Y position, distance to next pipe, gap center difference, bird's vertical velocity.
- **Output**: Whether the bird should flap or not.

A **Genetic Algorithm (GA)** is used to evolve better neural networks across generations:

- **Initialization**: Random networks for the first population.
- **Fitness function**: Score based on pipes passed + survival time.
- **Selection**: Best-performing birds are chosen as parents.
- **Crossover**: Combine weights from two parents.
- **Mutation**: Randomly tweak weights for diversity.

This process mimics natural selection and allows the AI to gradually improve.

Visual Studio Code (or Visual Studio/Rider)

The IDE is used for:

- Writing and debugging C# scripts.
- Integrating Unity game objects with AI logic.
- Managing project files.

Version Control (Optional)

Using **GitHub or Git** is highly recommended for:

- Tracking changes.
- Backing up work.
- Sharing the project with others.

So in short:

- **Unity** → game environment.
- **C#** → logic + AI scripts.
- **Neural Network + GA (NEAT-inspired)** → learning brains for birds.
- **IDE (VS Code / Visual Studio)** → coding.

---

3. Game Setup

The first step in the project was to recreate the **Flappy Bird environment** inside Unity. This provides the world where both humans (initially) and AI birds (later) can interact and learn. The main components are **core mechanics**, **obstacles**, and **player control**.

3.1 Core Mechanics

Flappy Bird is a simple game with just a few mechanics:

- **Gravity** pulls the bird downward continuously.
- **Flap** (a jump impulse) pushes the bird upward.
- **Pipes** scroll from right to left, creating obstacles.
- **Collision detection** determines when the bird hits a pipe or the ground (resulting in death).
- **Score** increases whenever the bird successfully passes a pair of pipes.

In Unity:

- Gravity was implemented manually in the Player and BirdAI scripts, by reducing the bird's Y velocity each frame.
- Flap was a positive force applied when the player taps or when the AI activates the output neuron.
- Pipes were spawned using a **spawner script** that creates new pipe pairs at fixed intervals with randomized gaps.
- The background and ground were placed as static sprites to simulate continuous movement.

3.2 Bird Physics

The bird's movement is defined by two main variables:

- **Direction (velocity)** → updated every frame with gravity.
- **Position** → updated by adding velocity × deltaTime.

In code (simplified from Player.cs / BirdAI.cs):

direction.y += gravity * Time.deltaTime;

transform.position += direction * Time.deltaTime;

- When the bird flaps, direction.y is set to a positive strength value.
- To make visuals more natural, the bird is also rotated slightly upward when moving up and downward when falling.

3.3 Pipe Spawning

The **Spawner script** generates pipes:

- New pipes appear every few seconds.
- Each pipe pair has a random vertical offset so that gaps are different.
- Once pipes move off-screen to the left, they are destroyed to save memory.

This creates a continuous challenge for the bird to navigate through.

3.4 Collision Detection

The bird has a **2D Collider**. Pipes and ground also have colliders with the tag "Obstacle".

- When the bird collides with an obstacle, the OnTriggerEnter2D method in the bird's script detects the hit.
- At that moment, the bird is marked as dead (isAlive = false).
- For AI, the **EvolutionManager** is notified so that the bird's fitness can be recorded.

3.5 Player Control (Initial Testing)

Before introducing AI, a simple **Player script** was created for human testing:

- Spacebar or screen tap triggers a flap.
- Gravity pulls the bird down otherwise.
- This ensured that the environment was working correctly (pipes, collisions, scoring) before AI took over.

This stage validated that the **core gameplay loop** was functional. Once the game worked manually, the same environment was used for the **AI-controlled birds**.

So at this stage, the game environment had:

- A working flappy bird physics system.
- Spawning pipes with randomized gaps.
- Scoring system.
- Collision + death detection.
- Player-controlled prototype (before AI).

---

4. AI Design

The core of this project is making the birds "learn" how to play Flappy Bird without being told *how*. Instead of hard coding a set of rules ("if pipe is near, flap"), we let the birds evolve strategies themselves through a **neural network** and a **genetic algorithm** inspired by NEAT (NeuroEvolution of Augmenting Topologies).

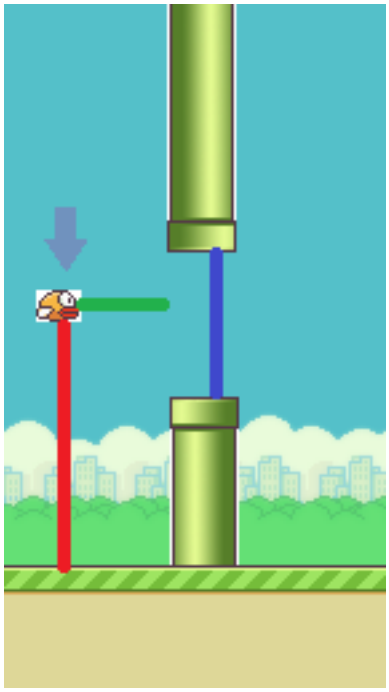
4.1 Neural Network (The Bird's Brain)

A **neural network** is a simplified model of how a brain processes information.
It takes **inputs** (senses), passes them through **neurons** (decision units), and produces **outputs** (actions).

Inputs (Bird's Sensors)

Each bird sees only limited information about the world:

1. Bird's **current Y position**.
2. **Horizontal distance** to the next pipe.
3. **Vertical distance** from the bird to the pipe gap center.
4. Bird's **vertical velocity**.



Analogy: Imagine you are blindfolded but someone tells you:

- "You're this high up."
- "The pipe is this far ahead."
- "The gap is this much above/below you."
- "You're moving up/down this fast."

From this, you decide: *Do I flap or not?*

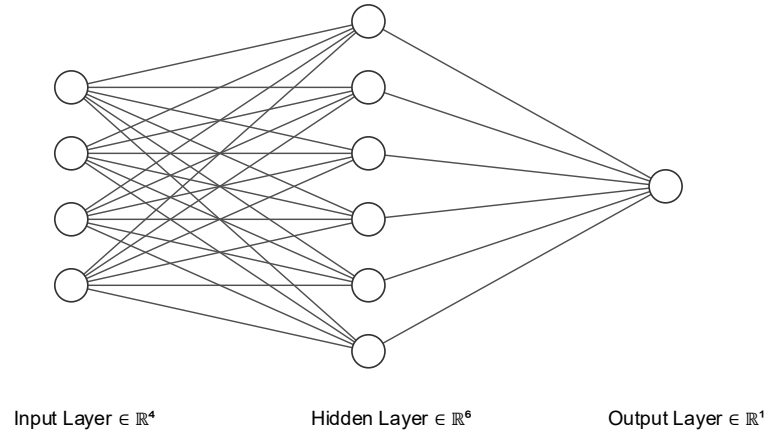
Outputs (Bird's Decision)

- **1 neuron** → produces a number between 0 and 1.
- If output > 0.5 → flap.
- If output ≤ 0.5 → do nothing (fall).

Hidden Neurons

- Hidden neurons process information in between input and output.
- They allow the bird to learn complex patterns (e.g., "If I'm falling and the pipe is close, flap earlier").

Neural Network:



Feedforward Process (How Bird Thinks)

1. Inputs (4 values) go into the input layer.
2. Each input is multiplied by a **weight** (connection strength).
3. Values are summed up and adjusted by a **bias** (like a mood setting).
4. The result passes through an **activation function** (usually sigmoid), squashing it into a 0–1 range.
5. The final number decides whether the bird flaps.

Equation:

$$output = \sigma(\sum(input * weight) + base)$$

Where σ = sigmoid function.

Unity Connection

- Every frame, the bird gathers inputs from the game (pipe position, velocity).
- Passes them into its neural network.
- Gets an output → decides flap/no flap.
- Updates its position accordingly.

4.2 Genetic Algorithm (The Evolution Process)

Instead of programming the neural network weights manually, we use **evolution**:

- Just like animals evolve over generations.
- Birds that perform better (survive longer, pass more pipes) reproduce more.
- Over many generations, birds adapt to master the game.

Steps in Evolution

1. **Population Initialization**
    - Start with 100 birds.
    - Each bird's neural network weights are random.
    - At first, they flap randomly (most die immediately).
2. **Fitness Function**
    - Defines how "good" a bird is.
    - Formula:

      $$\text{Fitness} = (\text{Pipes passed} * 1000) + \text{timealive}$$

    - Birds that survive and pass pipes score higher.
3. **Selection**
    - Pick the best-performing birds (the elite).
    - Some parents are chosen randomly using **roulette wheel selection** (probability proportional to fitness).
4. **Crossover**
    - Combine two parent birds' neural networks into a child.
    - Example: half the weights from parent A, half from parent B.
5. **Mutation**
    - Randomly tweak some weights (like genetic mutation).
    - Controlled by mutationRate and mutationStrength.

- Keeps population diverse (prevents everyone from being the same).

6. **Next Generation**
    - Old birds die.
    - New birds (children + elites) are spawned.
    - Repeat the cycle.

Analogy

Think of it like a **classroom exam**:

- At first, 100 students guess answers randomly (most fail).
- The top scorers are allowed to teach the next class.
- Some answers are copied, some are modified (mutations).
- Over many classes (generations), the students learn the correct answers (flappy strategy).

4.3 NEAT Principles Used

The original NEAT algorithm is very advanced (it evolves network structure too).
In this project, we use a **simplified NEAT-inspired GA**:

- Fixed network structure (inputs → hidden → output).
- Only weights & biases evolve.
- Still powerful enough for Flappy Bird.

4.4 BirdAI Integration

Each bird in Unity is linked to a **NeuralNetwork object**:

1. **Initialize()** assigns a cloned brain from the population.
2. **Update()** each frame:
    - Collects inputs (bird position, pipe distance).
    - Runs brain.FeedForward(inputs).
    - If output > 0.5 → flap.

3. **OnTriggerEnter2D()** detects death → reports fitness to EvolutionManager.

So each bird is an **agent**:

- The world is the teacher.
- Evolution is the grading system.
- Fitness is the exam result.

4.5 Evolution Manager (The Conductor)

The **EvolutionManager** script controls the whole process:

- Spawns 100 birds at the start of each generation.
- Tracks when each bird dies.
- Stores fitness values.
- Evolves a new population when all birds are dead.
- Keeps track of:
    - **Current generation**.
    - **Alive count**.
    - **Best fitness ever**.

This script is the "mother nature" of the simulation — it ensures the population survives and improves.

4.6 Why This Works

Although each bird is very simple (just one decision: flap or not), evolution makes them **increasingly smart**:

- Gen 1: Random flapping → almost all die at first pipe.
- Gen 10: Some survive past 1–2 pipes.
- Gen 50+: Birds learn timing, hover in gaps, and play consistently well.

This demonstrates a powerful principle:
Even without explicit rules, **simple learning + evolution = intelligent behavior.**

5. Evolution Manager

The **Evolution Manager** is the *"conductor"* of the entire AI training process. While each bird only knows how to flap or not flap, the Evolution Manager oversees **spawning, tracking, scoring, and evolving** the whole population.

Think of it as **Mother Nature** in the world of Flappy Bird.

5.1 Responsibilities of Evolution Manager

The Evolution Manager has 6 key roles:

1. **Initialize Population** → Create the first generation of random birds (brains).
2. **Spawn Birds** → Place each bird into the game world with its brain.
3. **Track Progress** → Monitor which birds are alive, how long they survive, and how many pipes they pass.
4. **Calculate Fitness** → Assign a score to each bird when it dies.
5. **Evolve New Generation** → Select the best, breed, and mutate to create new birds.
6. **Restart Simulation** → Clear old objects, spawn new birds, and continue training.

5.2 Step-by-Step Cycle

1. Initialization

- At the start of the game, the manager creates a **population** of neural networks.
- Each network has random weights and biases.
- Example: 100 birds = 100 neural networks.

```
for (int i = 0; i < populationSize; i++) {
  population.Add(new NeuralNetwork(inputCount, hiddenCount, outputCount));
}
```

2. Start a Generation

- All pipes from the last run are cleared.

- Birds are spawned at the starting position.
- Each bird gets a **clone of its assigned brain**.

```
for (int i = 0; i < populationSize; i++) {
   var go = Instantiate(birdAIPrefab, spawnPoint.position, Quaternion.identity);
   BirdAI ai = go.GetComponent<BirdAI>();
   ai.Initialize(population[i].Clone(), i);
}
```

3. Monitoring Birds

- Every frame, each bird runs its brain (FeedForward) and decides whether to flap.
- The Evolution Manager tracks:
    - **Alive count** (how many birds are still alive).
    - **Generation number**.
    - **Best fitness achieved** so far.

UI can display these values to visualize progress.

4. Death Reporting

When a bird dies (hits a pipe or ground), it calls:

EvolutionManager.Instance.ReportDeath(genomeIndex, pipesPassed, timeAlive);

This allows the manager to:

- Calculate **fitness** using the formula:

$$\text{Fitness} = (\text{Pipes passed} * 1000) + \text{timealive}$$

- Save the result in the fitnesses array.
- Update the **best bird ever** if this bird beat all previous records.

5. Evolution

Once all birds in the generation are dead:

1. **Sort by Fitness** → Rank birds from best to worst.
2. **Elitism** → Copy the top eliteCount brains unchanged to the new generation.
3. **Roulette Selection** → Select parents based on probability proportional to fitness.

4. **Crossover** → Create children by mixing two parent brains.
5. **Mutation** → Randomly tweak child weights to maintain diversity.
6. Replace the old population with the new one.

6. Restart Simulation

- All old bird GameObjects are destroyed.
- A new generation of birds is spawned with their evolved brains.
- The cycle repeats (Steps 2 → 6).

5.3 Key Features

- **Generation Counter** → Keeps track of how many times evolution has run.
- **Best Fitness Tracker** → Records the all-time best performer.
- **Brain Saving** → Best bird's brain is saved in JSON (bestBrain.json) for replay.
- **Alive Count** → Reports how many birds are still alive at any moment.
- **Time Scaling** → Training speed can be increased (e.g., timeScale = 5x) to evolve faster.

5.4 Analogy

Imagine a **school for birds**:

- Generation 1: 100 students take the exam (Flappy Bird).
- The teacher (EvolutionManager) records their scores.
- Top students (elites) get copied to the next class.
- Some mix notes (crossover), some make mistakes but discover better answers (mutation).
- Every new class gets smarter.

Over time, the school produces a bird genius that can fly perfectly.

5.5 Why It's Important

Without the Evolution Manager, the AI would be static — birds would never improve. This script transforms the project from a simple game into a **learning system**, allowing intelligence to emerge from randomness.

It's the central piece that bridges **neural networks (brains)** and the **game environment (world)**.

---

6. UI & Visualization

While the AI is training, it's crucial to show what's happening in the game. The user interface (UI) helps humans (players, researchers, or professors) understand how the AI is learning over generations.

6.1 Goals of the UI

1. Show **progress** (generation number, alive birds, score).
2. Make the training process **visible** and fun to watch.
3. Allow **speed control** (e.g., train at 5× speed).

6.2 Implemented UI Elements

- **Canvas with Text elements** for:
    - **Score** → how many pipes the leading bird passed.
    - **Generation** → which generation is currently running.
    - **Alive** → how many birds are alive in the current generation.

Representation:

Score: 3

Gen: 1

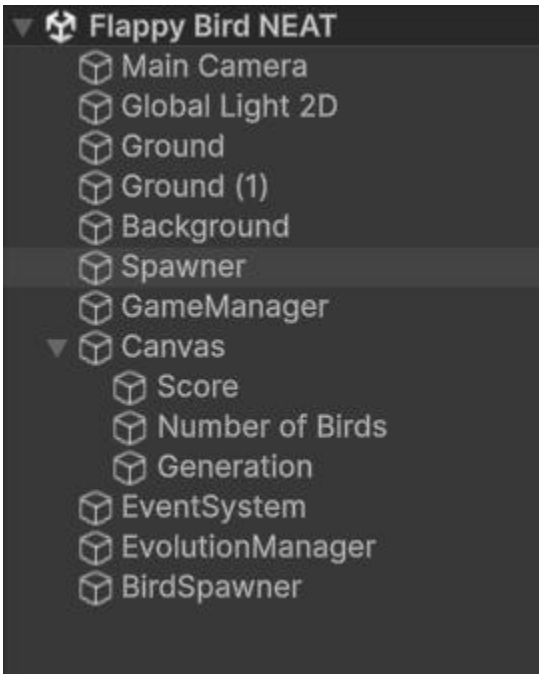
Alive: 97

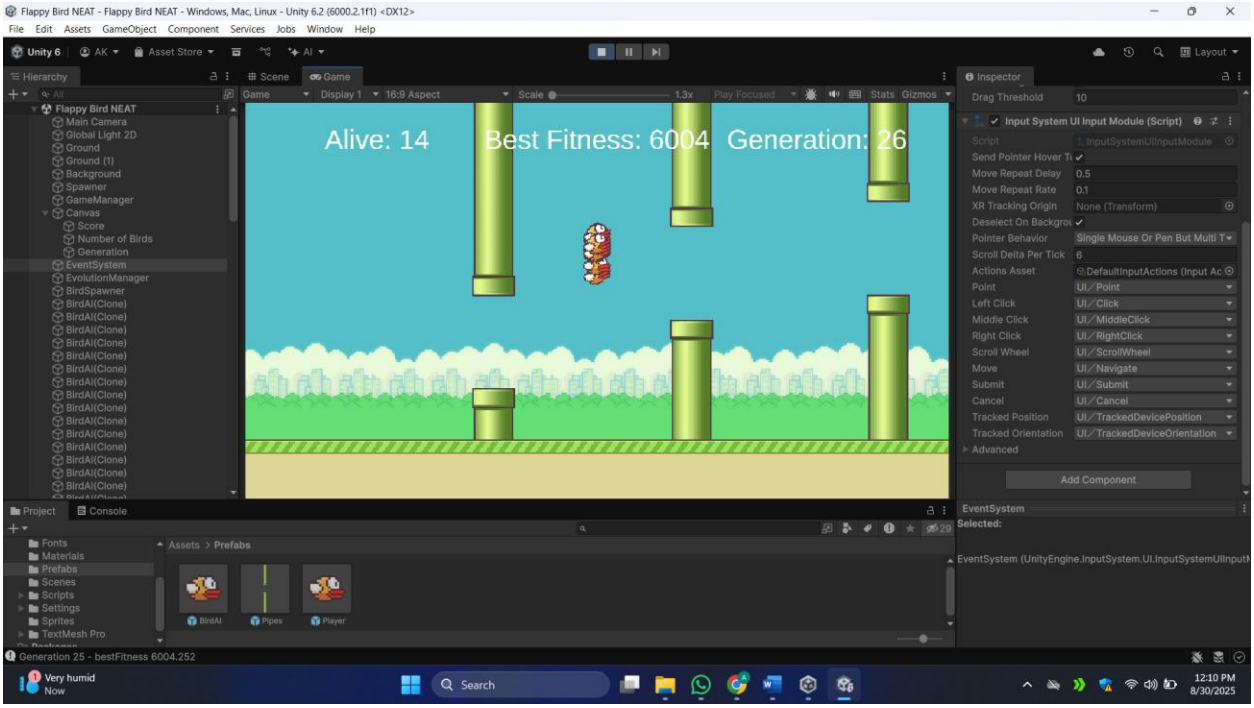
- **Training Speed Slider (Optional):**
  Adjusts timeScale to speed up/slow down training.

6.3 Unity Implementation

- Created a **Canvas** in Unity.
- Added a **Text (UI → TextMeshPro)** element.
- Updated the text every frame in a UIManager script:

```
uiText.text = $"Score: {currentScore}\nGen:
{EvolutionManager.Instance.Generation}\nAlive: {EvolutionManager.Instance.AliveCount}";
```





---

7. Results & Analysis

This section documents what happened during the AI's training process. It answers: *"Did the AI actually learn?"*

7.1 Early Generations (Random Behavior)

- In **Generation 1**, all birds flap randomly because their brains are initialized with random weights.
- Most die at the first or second pipe.
- Score rarely exceeds 1.

7.2 Mid Generations (Improvement Begins)

- By **Generation 10–20**, some birds start surviving longer.
- The evolution process selects the best performers and combines their traits.
- Score improves to 3–5 pipes consistently.
- Alive count drops slower compared to Generation 1.

7.3 Advanced Generations (Learned Strategy)

- By **Generation 50+**, the AI develops consistent strategies:
    - Flapping near the bottom of gaps.
    - Avoiding unnecessary flaps (smoother flight).
- High scores of **20+ pipes** become possible.

7.4 Quantitative Analysis

- **Fitness Growth:** Best fitness increases steadily across generations.
- **Score Trend:** Max score per generation rises as evolution progresses.
- **Survival Time:** Average survival time increases over time.

👉 You can also record **graphs** of:

- Best fitness vs generation.
- Average fitness vs generation.

7.5 Observations

- Evolution works even with a simple network (1 hidden layer).
- Diversity from mutation is critical → without it, population stagnates.
- Increasing population size leads to faster learning (but requires more computation).

---

8. Future Work

While the current AI successfully learns to play Flappy Bird using a simple neural network + genetic algorithm, there are many directions for extending and improving the project.

8.1 Algorithm Improvements

- **Deep Neural Networks:**
  Instead of a single hidden layer (6 neurons), future versions could use multiple hidden layers or more neurons for richer decision-making.
- **NEAT (NeuroEvolution of Augmenting Topologies):**
  The current network topology is fixed. NEAT would evolve both **weights** and **network structure** (adding/removing neurons and connections). This would allow the AI to discover more complex strategies.
- **Hybrid Learning:**
  Combine genetic algorithms with **gradient-based learning** (e.g., backpropagation) for faster convergence.

8.2 Training Enhancements

- **Parallel Training:**
  Run multiple generations simultaneously (multi-threading or cluster computing) to accelerate evolution.
- **Adaptive Mutation Rate:**
  Dynamically adjust mutation strength based on how stagnant or diverse the population is.
- **Replay Buffer:**
  Save past successful birds and occasionally reintroduce them into the population to maintain strong genetic material.

8.3 Visualization & UI

- **Live Neural Network Visualizer:**
  Show the active bird's brain (inputs, hidden neurons, output) in real time.
- **Graphs in Unity:**
  Display generation vs fitness/score trend directly inside the Unity editor.
- **Bird Highlighting:**
  Mark the *current best-performing bird* visually (e.g., with a golden outline).

8.4 Applications Beyond Flappy Bird

- **Obstacle Navigation Drones:**
  Adapt the algorithm to train drones or robots to pass through obstacles.
- **Self-driving Car Simulation:**
  Replace pipes with traffic rules and obstacles.
- **Other Games:**
  Extend to Pong, Snake, or platformers using the same evolution framework.

---

9. Conclusion

This project successfully demonstrated how **evolutionary algorithms** can be used to train an artificial intelligence agent inside a game environment.

- The **Flappy Bird environment** was implemented in Unity, with custom physics, obstacles, and scoring.
- A **simple feedforward neural network** (4-6-1 architecture) was used as the bird's brain.
- The **Evolution Manager** applied genetic algorithm principles (selection, crossover, mutation) to improve performance across generations.
- Over time, the AI progressed from **random flapping (Gen 1)** to **learned strategies (Gen 50+)**, surviving for much longer and achieving higher scores.

This work proves that even **simple AI techniques** can create intelligent behavior in games, providing both entertainment and an educational demonstration of machine learning.

---

10. References

1. **Unity Documentation** – Game Engine fundamentals and scripting.
   https://docs.unity3d.com/
2. **Code Bullet (YouTube)** – Inspiration for evolving Flappy Bird AI using NEAT.
   https://www.youtube.com/watch?v=WSW-5m8lRMs
3. **Stanley, K. O., & Miikkulainen, R. (2002).**
   Evolving Neural Networks through Augmenting Topologies (NEAT). *Evolutionary Computation, 10(2)*.

4. **Sebastian Lague (YouTube)** – Tutorials on AI, neural networks, and genetic algorithms in Unity.
   https://www.youtube.com/user/Cercopithecan
5. **NEAT-Python Library** – Implementation details for NEAT algorithm in Python.
   https://neat-python.readthedocs.io/
6. **Goodfellow, I., Bengio, Y., & Courville, A. (2016).**
   Deep Learning. MIT Press.