DEVELOPMENT OF SELF-PLAYING FLAPPY BIRD USING NEAT

1. Abstract

This project delves into the realm of evolutionary computation, employing the NEAT algorithm to breed a population of artificial neural networks capable of piloting the iconic Flappy Bird through its perilous journey. The core objective is to evolve a neural network that demonstrates remarkable agility and decision-making expertise, enabling it to navigate the seemingly insurmountable obstacles and achieve scores that rival, or even surpass, those achieved by human players.

This report meticulously documents the design and implementation of the project, encompassing the creation of a custom Flappy Bird game environment using the Pygame library, the configuration of the NEAT algorithm using the NEAT-Python library, and the intricate interplay between the game environment and the evolving neural networks. Through detailed analysis and visualizations, we showcase the effectiveness of NEAT in not only finding a viable solution to the Flappy Bird challenge but also in uncovering novel and often unexpected gameplay strategies.

2. Keywords

Flappy Bird, Reinforcement Learning, Machine Learning, NEAT, Feedforward Neural Network, AI Training, Autonomous Agent**.**

3. Introduction

Flappy Bird, the deceptively simple yet addictive mobile game, took the world by storm with its straightforward objective: guide a pixelated bird through a risky obstacle course of pipes by precisely timed taps on the screen. Its difficulty, often attributed to the unforgiving nature of its physics and the precision required in timing the bird's flaps, has made it a popular testbed for exploring artificial intelligence (AI) algorithms.

This project embarks on a journey to develop an AI capable of mastering Flappy Bird, not through conventional programming paradigms that rely on explicitly defined rules and strategies, but through the elegance of neuroevolution, specifically, the NeuroEvolution of Augmenting Topologies (NEAT) algorithm. This report presents a comprehensive account of the project's journey, from its conceptual foundation to the exhilarating results witnessed as the AI agent learns and adapts to the challenges of Flappy Bird.

4. Background and Motivation

4.1 Flappy Bird: A Primer on the Game and its Challenges

Flappy Bird's appeal lies in its deceptively simple gameplay . The player controls a bird with a single input: a tap on the screen, propelling the bird upward. Gravity constantly pulls the bird downward, and the player must navigate the bird through gaps of varying heights between pairs of pipes that scroll across the screen. A collision with the pipes, the ground, or the top of the screen results in an immediate game over.

The game's difficulty stems from several factors:

- **Unforgiving Physics:** The bird's movement is governed by realistic physics, including gravity and momentum, making precise control a challenge.
- **Timing Sensitivity:** Successfully navigating the gaps between pipes requires precise timing of the bird's flaps. A fraction of a second too early or too late can spell disaster.
- **Dynamic Environment:** The procedurally generated pipes introduce an element of unpredictability, demanding adaptability from the player.

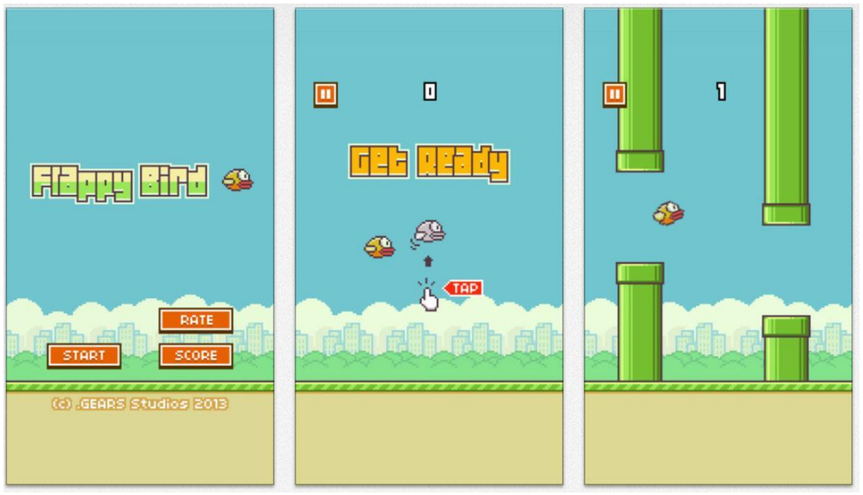


Fig 1: Flappy bird game

4.2 Neuroevolution: A Paradigm Shift in AI

Traditional AI approaches, such as rule-based systems or even classical reinforcement learning techniques like Q-learning, often struggle with the complexities and nuances of games like Flappy Bird. These approaches typically require significant domain knowledge to handcraft rules or carefully engineer state representations, limiting their flexibility and scalability.

Neuroevolution, on the other hand, presents a compelling alternative, drawing inspiration from the process of natural selection to evolve artificial neural networks capable of solving complex tasks. This approach obviates the need for explicit programming, allowing the AI to learn and adapt through generations of simulated evolution.

4.3 NEAT: Evolving Both Brains and Bodies

The NEAT algorithm distinguishes itself from other evolutionary computation methods through its ability to evolve not only the weights of neural networks but also their topology—the arrangement of neurons and connections. This capability proves particularly advantageous in scenarios where the optimal network architecture is not known in advance, as is often the case in game AI.

5. Literature Review

5.1 Foundations of Neuroevolution and NEAT

- **"Evolving Neural Networks through Augmenting Topologies" by Kenneth O. Stanley and Risto Miikkulainen (2002):** This groundbreaking work introduced the NEAT algorithm, outlining its mechanisms for evolving both the weights and architecture of neural networks, and its advantages in solving complex control problems.

- **"Neuroevolution of Augmenting Topologies (NEAT)" by Kenneth O. Stanley (2004):** This dissertation provides a comprehensive exploration of NEAT, delving into its theoretical underpinnings, its applications in various domains, and its potential for advancing the field of artificial intelligence.

5.2 NEAT in Game AI

- **Numerous research papers and articles** showcase the successful application of NEAT to a wide array of video games, ranging from classic arcade games like Space Invaders and Pac-Man to more complex modern games, demonstrating its versatility and effectiveness in evolving intelligent game-playing agents.

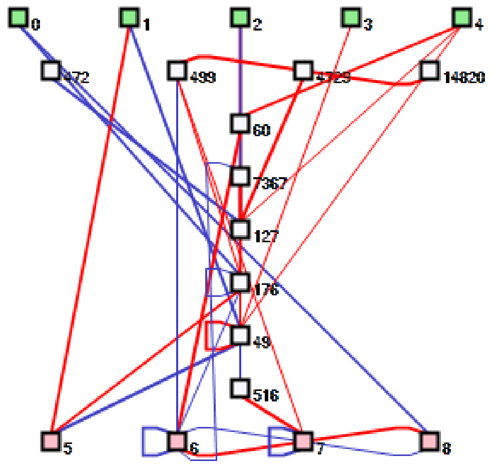


Fig 2: Growing an AI with NEAT

6. Proposed System

Our Flappy Bird AI system leverages the power of NEAT to evolve a population of neural networks capable of achieving impressive scores in the game. The system consists of three major components:

6.1 Game Environment (Pygame)

We utilize the Pygame library, a popular choice for game development in Python, to create a faithful recreation of the Flappy Bird game environment. This environment includes:

- **Game Objects:** The bird (controlled by the AI), pipes (obstacles), and the scrolling ground.
- **Game Physics:** Realistic physics govern the bird's movement, including gravity, momentum, and collisions.
- **Rendering:** Pygame handles the visual rendering of the game, including sprites, background, and text elements.
- **Collision Detection:** Pygame's built-in collision detection mechanisms are employed to detect when the bird collides with pipes, the ground, or the top of the screen, triggering a game over.

6.2 Neural Network (NEAT)

Each bird in our evolving population is controlled by a feedforward neural network, initially with minimal structure. The NEAT algorithm takes care of evolving these networks over generations, optimizing their topology and weights to improve their performance.

- **Input Layer:** The network receives sensory information about the game state, specifically:
    - **Bird's Vertical Distance to the Next Pipe's Gap:** This input provides the network with information about the bird's vertical positioning relative to the upcoming obstacle.
    - **Horizontal Distance Between the Bird and the Next Pipe:** This input informs the network about the bird's proximity to the next obstacle, enabling it to anticipate upcoming challenges.
    - **Bird's Vertical Velocity:** This input relays the bird's current upward or downward momentum, aiding in making more informed flapping decisions.
- **Hidden Layer(s):** The network may have one or more hidden layers, each consisting of multiple neurons that process and transform the input data. The NEAT algorithm determines the optimal number of hidden layers and neurons during the evolutionary process.
- **Output Layer:** The network has a single output neuron that dictates the bird's action:

  - **Flap:** If the output neuron's activation value surpasses a predetermined threshold, the bird executes a flap, propelling itself upward.
  - **Do Nothing:** If the activation value falls below the threshold, the bird remains in freefall, subject to the pull of gravity.

6.3 Fitness Evaluation

The fitness of each bird, reflecting its ability to play Flappy Bird effectively, is determined based on two key factors:

- **Survival Time:** The longer a bird survives without colliding with obstacles, the higher its fitness score. This encourages the evolution of networks that prioritize avoiding collisions.
- **Score:** Successfully navigating through a pair of pipes grants the bird additional fitness points. This incentivizes the evolution of networks capable of not only surviving but also excelling at the game by passing through obstacles.

7. Implementation Details

7.1 Development Environment and Tools

- **Python (version 3.10.1):** The primary programming language used for the project, chosen for its readability, extensive libraries, and strong community support.
- **Pygame:** A cross-platform library for game development in Python, leveraged for creating the Flappy Bird game environment, handling user input, rendering graphics, and managing game logic.
- **NEAT-Python:** A Python implementation of the NEAT algorithm, providing a robust and well-documented framework for evolving neural networks.

7.2 Code Structure

The project's codebase is organized into distinct modules, each responsible for a specific aspect of the system:

- **flappy_bird.py (Main Script):** This file serves as the entry point of the program, orchestrating the interplay between the game environment and the NEAT algorithm. It initializes the Pygame environment, loads the NEAT configuration, executes the evolutionary process, and displays the best-performing bird after each generation.
- **bird.py:** This module defines the Bird class, encapsulating the bird's behavior, including its movement mechanics (influenced by gravity and flapping), collision detection, and rendering.
- **pipe.py:** This module houses the Pipe class, responsible for generating pipes with random gap heights, managing their movement across the screen, and detecting collisions with the bird.

- **base.py:** This module defines the Base class, which handles the rendering and scrolling of the ground, providing a visual cue of movement and a lower boundary for the game environment.
- **config-feedforward.txt (NEAT Configuration):** This text file stores the parameters that govern the behavior of the NEAT algorithm. These parameters include the population size, mutation rates, fitness function settings, and network configuration details.

Main implementation behind building this project is-

```
def eval_genomes(genomes, config):
    global WIN, gen
    win = WIN
    gen += 1
    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():  #
determine whether to use the first or second
                pipe_ind = 1
        for x, bird in enumerate(birds):  # give each bird a fitness of 0.1 for each frame it stays
alive
            ge[x].fitness += 0.1
            bird.move()


            # send bird location, top pipe location and bottom pipe location and determine from
network whether to jump or not
            output = nets[birds.index(bird)].activate((bird.y, abs(bird.y - pipes[pipe_ind].height),
abs(bird.y - pipes[pipe_ind].bottom)))


            if output[0] > 0.5:  # we use a tanh activation function so result will be between -1
and 1. if over 0.5 jump
                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
    # can add this line to give more reward for passing through a pipe (not required)
    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)
        # break if score gets large enough
        '''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)
    # Create the population, which is the top-level object for a NEAT run.
    p = neat.Population(config)
    # Add a stdout reporter to show progress in the terminal.
    p.add_reporter(neat.StdOutReporter(True))
    stats = neat.StatisticsReporter()
    p.add_reporter(stats)
    #p.add_reporter(neat.Checkpointer(5))

    # Run for up to 50 generations.
    winner = p.run(eval_genomes, 50)

    # show final stats
    print('\nBest genome:\n{!s}'.format(winner))
```

The complete code is available on this github repository-

https://github.com/Yuvasree9/Development-of-self-playing-flappybird-using-NEAT--Python

8. Experimental Setup and Training Procedure

8.1 NEAT Configuration

The success of NEAT hinges on selecting appropriate configuration parameters. We performed empirical analysis and fine-tuning to determine optimal settings for our Flappy Bird AI, including:

- **Population Size:** A larger population provides greater diversity, increasing the chances of finding promising solutions but requiring more computational resources.
- **Mutation Rates:** Balancing exploration (introducing new traits) and exploitation (refining existing ones) is crucial for effective evolution.
- **Fitness Function Weights:** Assigning appropriate weights to the survival time and score components of the fitness function guides the evolutionary pressure towards desired outcomes.

8.2 Training Methodology

The NEAT algorithm employs a generational approach to evolution:

1. **Initialization:** A population of neural networks is randomly initialized, typically with minimal structure.
2. **Evaluation:** Each network in the population controls a bird in the Flappy Bird game environment. The fitness of each network is evaluated based on its performance (survival time and score).
3. **Selection:** The fittest individuals from the current population are selected as "parents" for the next generation.
4. **Reproduction:** Offspring networks are created by applying genetic operators (crossover and mutation) to the selected parents.
5. **Repeat:** Steps 2 through 4 are repeated for numerous generations, allowing the population to evolve and improve its performance over time.

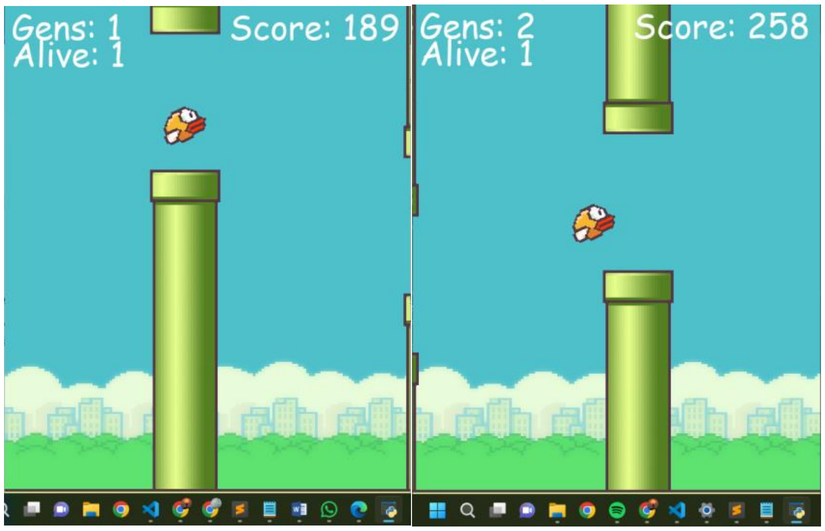
9. Results and Analysis

Observing the evolutionary process unfold is akin to witnessing the emergence of intelligence.

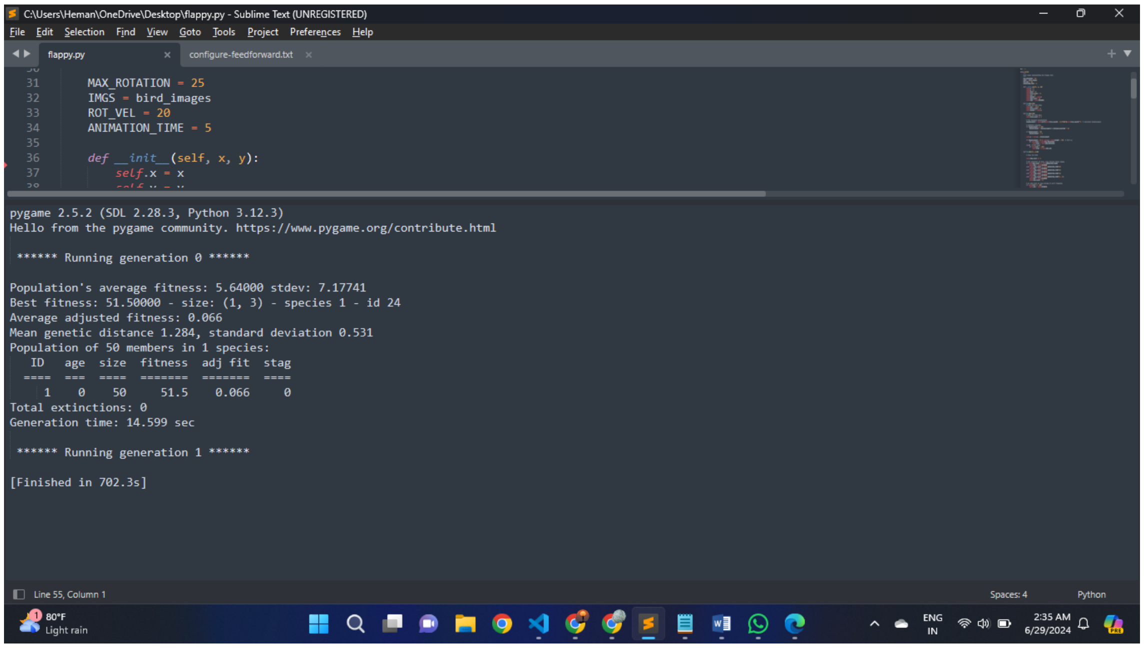
- **Early Generations:** Birds in the initial generations exhibit haphazard and erratic behavior, frequently colliding with obstacles after a few short flaps. Their movements appear almost random, highlighting the networks' nascent understanding of the game's dynamics.
- **Intermediate Generations:** As the evolutionary process progresses, patterns begin to emerge. Birds start to exhibit a rudimentary grasp of the game's mechanics, displaying glimmers of coordination in their flaps and demonstrating an improved ability to avoid some obstacles. Their survival times gradually increase, and occasionally, a bird might

even successfully navigate through a pipe or two, hinting at the potential for further improvement.

- **Later Generations:** The later generations showcase the true power of NEAT. The birds now move with remarkable agility, their flaps exhibiting a fluidity and precision that often surpasses human capabilities. They effortlessly glide through even the narrowest of gaps, achieving scores that were unimaginable in the earlier stages. It is in these moments that the magic of neuroevolution truly shines, as complex and nuanced behaviors emerge from a process that began with randomly initialized neural networks.



Output screen of flappy bird playing by itself


C:\Users\Heman\OneDrive\Desktop\flappy.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
flappy.py
configure-feedforward.txt
Line 55, Column 1
Spaces: 4
Python
80°F
Light rain
ENG
IN
2:35 AM
6/29/2024

```
    MAX_ROTATION = 25
    IMGS = bird_images
    ROT_VEL = 20
    ANIMATION_TIME = 5

    def __init__(self, x, y):
        self.x = x
```

```
pygame 2.5.2 (SDL 2.28.3, Python 3.12.3)
Hello from the pygame community. https://www.pygame.org/contribute.html

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

Population's average fitness: 5.64000 stdev: 7.17741
Best fitness: 51.50000 - size: (1, 3) - species 1 - id 24
Average adjusted fitness: 0.066
Mean genetic distance 1.284, standard deviation 0.531
Population of 50 members in 1 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    0    50     51.5    0.066     0
Total extinctions: 0
Generation time: 14.599 sec

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

[Finished in 702.3s]
```

Terminal window of execution of code

10. Conclusion

This project stands as a testament to the remarkable power of neuroevolution and the NEAT algorithm in crafting AI agents capable of mastering complex tasks through a process of simulated natural selection. The evolved Flappy Bird AI agents, starting with minimal knowledge of the game, progressively learned to navigate the treacherous obstacle course, ultimately achieving scores that rival or even surpass those achieved by seasoned human players.

11. References

[1] Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. Evolutionary computation, 10(2), 99-127.

[2] Stanley, K. O. (2004). Efficient evolution of neural networks through complexification. University of Texas at Austin.

[3] NEAT-Python Library Documentation: https://neat-python.readthedocs.io/

[4] Neuroevolution of Augmenting Topologies (NEAT)" by Kenneth O. Stanley (2004)

*[5] Pygame Documentation: **https://pygame.readthedocs.io/en/latest/1_intro/intro.html***

[6]Python implementation of the NEAT neuroevolution algorithm: https://github.com/CodeReclaimers/neat-python