CSCI-218 PROJECT

Topic 1: adding AI(NEAT) to a simple game(flappy-bird)

Group Members:

Mikaeel Faraz Safdar - 8074689
Mohammad Qudaih-7736356
Haifa Shkhedem - 7820197
Surya John prabhu -7830890
Abhinandan Ajit Kumar - 7837586

Table of Contents

1. Executive Summary

This study explores the application of artificial intelligence (AI) in teaching a neural network to play the game Flappy Bird using the NEAT approach (NeuroEvolution of Augmenting Topologies). By starting with random actions and gradually becoming an expert at the game through iterative evolution, the goal was to develop an AI agent that could learn and perform better on its own.

The key to solving Flappy Bird's dynamic issues was the NEAT algorithm's capacity to optimize neural network weights and structure. The AI gradually adjusted to avoid barriers and attain greater scores by changing the network with every generation.

The project's results show how effective evolutionary algorithms are at training AI to perform difficult jobs without the need for preset guidelines. The AI is able to learn how to overcome challenges, perform consistently, and get good grades. This project is a useful case study for investigating machine learning in gaming since it demonstrates the applicability of neuroevolutionary techniques in real-time decision-making tasks.

2. Motivation

The NEAT method was used for this project because it can change a neural network's weights and structure, which makes it extremely flexible to the changing demands of Flappy Bird. The game's environment is perfect for testing NEAT's evolutionary capabilities because it necessitates making decisions in real time based on the bird's position in relation to barriers.

NEAT begins with basic neural networks and gradually improves them, enabling the AI to learn and adapt more effectively than static algorithms. By choosing the top-performing networks in each generation, this neuroevolution approach mimics the process of natural selection, allowing the AI to gradually enhance its gameplay. The project's use of NEAT not only addresses the challenge of teaching AI to play Flappy Bird, but it also shows how the method may be used in more general contexts including dynamic problem-solving.

3. Technical Requirements

This section outlines the software, hardware, and setup instructions required to run the Flappy Bird AI using the NEAT algorithm. It also includes the necessary steps for installing dependencies and configuring the environment.

3.1 Software Requirements

To run the project, you will need the following software:

- **Python** (version 3.6 or higher): The primary programming language used for the project.
  - Download from Python's official website.
- **Visual Studio Code (VS Code)**: A code editor to write and run the Python scripts.
  - Download from VS Code website.
- **Pygame**: A Python library used for game development, handling game graphics, sounds, and user input.
  - Install via terminal using the command:
    bash
    Copy code
    pip install pygame
- **NEAT-Python**: A Python library used to implement the NEAT (NeuroEvolution of Augmenting Topologies) algorithm for evolving neural networks.
  - Install via terminal using the command:
    bash
    Copy code
    pip install neat-python

3.2 Hardware Requirements

- **Operating System**: Windows, macOS, or Linux
- **Minimum Hardware Specifications**:
  - **Processor**: Any modern processor (Intel Core i3 or equivalent).
  - **RAM**: At least 4GB of RAM (8GB recommended for smoother performance).
  - **Storage**: At least 500MB of free disk space for the project files and dependencies.
  - **Graphics**: A basic graphics card capable of running Pygame (any modern graphics card should suffice).

3.3 File Setup and Directory Structure

1. **Extracting Project Files**: After installing the project zip file, extract its contents into a single folder.
    - Ensure the folder contains all necessary files, including the Python scripts, configuration files, and image files.
2. **Image Files**: The following image files must be present in the specified directory:
    - bird1.png, bird2.png, bird3.png, pipe.png, base.png, bg.png.
    - These images are essential for rendering the game, and the paths to these files must be correctly referenced in the code.

3.4 Configuration File

The project uses a configuration file to specify the parameters for the NEAT algorithm. Some key parameters include:

- **Population size**: Number of neural networks in each generation.
- **Mutation rates**: Probability of mutation occurring for weights and network structure.
- **Fitness threshold**: The score at which the algorithm stops.
- **Number of inputs/outputs**: Defines the structure of the neural network used to control the bird.

The configuration file should be edited to fit the specific requirements of the project and the desired level of difficulty.

3.5 Setting Up the Environment

To set up the environment, follow these steps:

1. **Install Python** and ensure it's added to your system's PATH.
2. **Install VS Code** and open the project folder in it.
3. Install Dependencies: Open the terminal in VS Code and run the following commands to install the necessary libraries:
   bash
   Copy code
   pip install pygame
   pip install neat-python
4. Update Image File Paths: Modify the paths to the image files in the script. The code includes references to images, and these paths must be updated to match where the images are stored on your local machine. For example:
   python
   Copy code
   BIRD_IMGS =
   [pygame.transform.scale2x(pygame.image.load(os.path.join("C:/path/to/images/bird1.png"))),
   pygame.transform.scale2x(pygame.image.load(os.path.join("C:/path/to/images/bird2.png"))),
   pygame.transform.scale2x(pygame.image.load(os.path.join("C:/path/to/images/bird3.png")))]
5. **Running the Game**: Once the setup is complete, you can run the game by executing the following command in the VS Code terminal:
   bash
   Copy code
   python main.py
6. This will start the game, and the NEAT algorithm will begin evolving the neural networks to play Flappy Bird.

3.6 Running the Game and Monitoring Output

When running the game, you will see output in the terminal, showing the progress of the AI training over multiple generations. Each generation's average fitness, the best fitness score, and the genetic distance will be displayed, helping you track the AI's evolution.

4. Main Constructs (Aspects/Parts)

4.1 Flappy Bird Game Mechanics

Flappy Bird is a side-scrolling game where the player controls a bird attempting to avoid obstacles. The objective is to stay airborne without hitting pipes or falling to the ground. The mechanics are straightforward but challenging, providing an excellent environment for testing AI solutions.

Game Setup:
The bird falls due to gravity, and the player must make it "flap" to stay airborne. It moves continuously forward and must navigate through vertically spaced pipes that appear from the right.

Obstacles:
Pipes are randomly generated with varying gaps, moving from right to left. The game ends if the bird collides with a pipe or falls to the ground.

Gameplay Objective:
The goal is to pass through as many pipes as possible. The score increases each time the bird successfully clears a gap, requiring real-time decision-making based on the bird's position, velocity, and the location of obstacles.

Pygame Implementation:
The game is implemented using the Pygame library, which handles graphics rendering, user input, and the game loop. The loop updates the bird’s position, checks for collisions, and renders the game state.

4.2 NEAT Algorithm Application

The NEAT (NeuroEvolution of Augmenting Topologies) algorithm evolves artificial neural networks to control the bird's actions. NEAT adjusts both the structure and weights of networks, making it ideal for learning and optimizing real-time behaviors.

How NEAT Works:

1. Population Initialization:
- The algorithm begins with a population of simple neural networks (genomes), which evolve over generations.

2. Fitness Evaluation:
- Each bird's performance is measured using a fitness function based on survival time and the number of pipes cleared.

3. Selection and Reproduction:
- High-performing genomes are selected to create the next generation through mutation and crossover.

4. Mutation and Evolution:
- Mutations involve changes to neural network weights, adding/removing nodes, or modifying connections, ensuring exploration of new strategies.

5. Stopping Criterion:
- The algorithm stops after a set number of generations (e.g., 50) or when a bird achieves a target fitness score (e.g., 100).

6. Saving the Best Genome:
- The highest-performing genome is saved for future use, representing the optimal AI model learned during evolution.

Key NEAT Features:
- Neural networks are feed-forward, with inputs processed sequentially to generate outputs.
- Default settings include mutation probabilities (e.g., 0.2 for adding nodes, 0.5 for modifying weights) and a tanh activation function.

4.3 Integration with Game Code

NEAT is integrated with the game code to evolve AI-driven behavior in real-time.

Inputs to Neural Network:
- The bird's vertical position relative to the ground.
- The distance to the top and bottom pipes.

Output:
- A decision on whether the bird should flap or not, based on the processed inputs.

Game Visualization:
- Pygame renders the bird, pipes, and score while reflecting the neural network's decisions. The game state updates dynamically, with fitness values adjusted based on performance.

Configuration File:
- A config-feedforward file specifies NEAT parameters such as population size, mutation rates, and network structure. These settings govern the evolution process and AI learning behavior.

4.4 Summary

Flappy Bird's mechanics provide a challenging yet straightforward environment for testing AI-based solutions. By evolving neural networks using NEAT, the AI learns to optimize decision-making over generations, improving its ability to navigate dynamic obstacles. This integration of simple game mechanics with advanced evolutionary algorithms highlights the potential of AI in real-time problem-solving.

5. Implementation

This section describes how the Flappy Bird game and the NEAT algorithm are implemented and integrated to create an AI that can learn to play the game. The implementation includes the setup of the game loop, the neural network integration using NEAT, fitness evaluation, and the training process.

5.1 Game Loop and NEAT Integration

The game loop, implemented using Pygame, continuously updates the game state and integrates the NEAT algorithm for decision-making. Key components include:

1. Game Initialization:
- Pygame sets up the window and initializes the bird, pipes, and background.

2. NEAT Initialization:
- A population of neural networks is generated to control the bird's actions.

3. Neural Network Inputs and Outputs:
- Inputs include the bird's position and the distances to the pipes.
- The network outputs whether the bird should flap or not.

4. Game Loop:
- Updates the game screen, processes bird movements, checks collisions, and adjusts fitness values based on performance. This loop evolves the population over generations.

5.2 Fitness Evaluation and Neural Network Training

Fitness evaluation drives the evolution of neural networks. The process includes:

1. Fitness Calculation:
- Scores increase with survival time and pipes cleared, rewarding effective navigation.

2. Selection of Parents:
- High-fitness genomes are chosen for reproduction to generate the next generation.

3. Mutation and Crossover:
- Offspring genomes inherit successful traits with additional mutations to explore new strategies.

4. Stopping Criteria:
- Evolution halts after reaching the fitness threshold or a maximum number of generations.

5. Saving the Best Genome:
- The top-performing genome is saved as a file, ready to demonstrate the trained AI.

This implementation combines the simplicity of Flappy Bird with the adaptability of NEAT to create an engaging and dynamic AI training environment.

5.3 Code Implementation

The core logic of the game and the NEAT algorithm is integrated in the main() function. Below is a breakdown of how the game and NEAT algorithm work together in the code.

1. **Game Setup and Pygame Initialization**:
   The Pygame library is used to initialize the game window, set up the game environment, and handle user input. The bird's image and pipes are loaded into the game, and their initial positions are set.
2. **Neural Network Decision-Making**:
   For each bird, the neural network takes the following inputs:
   - The vertical position of the bird.
   - The distance to the top pipe.
   - The distance to the bottom pipe.
3. Based on these inputs, the neural network outputs whether the bird should flap or not. The output is then used to control the bird's behavior in the game. This is done by feeding the current game state into the neural network, which decides the next action.
4. **Fitness Evaluation and Updates**:
   The fitness function updates each bird's fitness based on how long it survives and how many pipes it clears. If the bird collides with a pipe or the ground, its fitness score is updated to reflect its failure, and the game restarts for that bird. The fitness values are stored and used for selection in the next generation.
5. **Pygame Rendering**:
   The game state is updated in each iteration, and the Pygame library is used to render the current state of the game to the screen. The bird's position, the pipes, and the current score are drawn on the screen. The game continues until the stopping criteria are met.

5.4 Final Demonstration and Saved Genome

Once the NEAT algorithm has evolved the population and selected the best-performing genome, this genome is saved as winner_genome.pkl. This saved genome can then be loaded into the game to demonstrate how the AI plays the game autonomously.

- **AI Mode**:
  The AI_mode.py script can be used to load the winner_genome.pkl file and run the Flappy Bird game with the trained AI. The AI will control the bird based on the neural network learned through the NEAT process, and the game will play automatically, showcasing how well the AI has learned to avoid pipes and survive.

5.5 Summary

The implementation of the Flappy Bird AI with NEAT involves integrating the NEAT algorithm with the game logic. The game continuously trains neural networks through evolution, where networks that perform better (by surviving longer and clearing more pipes) are selected to produce offspring. Over generations, the neural networks evolve, allowing the AI to improve its gameplay. The combination of Pygame for game rendering and NEAT for AI evolution results in an AI that can autonomously play Flappy Bird, demonstrating the effectiveness of neuroevolution in real-time decision-making tasks.

6. Difficulties and Constraints

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

1. Computational Challenges

- **Resource Intensity**: Evolving neural networks through NEAT is computationally demanding, especially as networks become more complex over generations. This complexity can lead to increased processing time and memory usage, potentially hindering real-time application.

2. Algorithmic Constraints

- **High-Dimensional Input Limitations**: NEAT may struggle with high-dimensional inputs, as it can be challenging to create well-tuned networks in such scenarios.
- **Fitness Function Design**: Crafting an appropriate fitness function that accurately measures the AI agent's performance is complex. The function must balance short-term and long-term goals, as well as exploration and exploitation, to guide the evolutionary process effectively.

3. Environmental Challenges

- **Dynamic Game Environment**: Flappy Bird's environment is dynamic, with obstacles appearing at varying intervals and positions. This variability requires the AI to adapt to changing conditions, complicating the learning process.
- **Real-Time Decision Making**: The AI must make quick decisions to navigate obstacles, necessitating efficient neural network architectures capable of processing inputs and producing outputs within tight time constraints.

4. Implementation Constraints

- **Parameter Tuning**: Determining optimal values for parameters such as population size, mutation rates, and crossover rates is crucial. Improper tuning can lead to premature convergence or excessive computational load.
- **Reproducibility and Randomness**: The stochastic nature of evolutionary algorithms introduces challenges in reproducing results consistently, which can complicate debugging and performance evaluation.

7. Conclusion

The project successfully demonstrated the application of the NEAT algorithm to train an AI agent for dynamic, real-time decision-making. Over successive generations, the AI improved its ability to navigate obstacles, achieving higher fitness scores and consistently better performance.

The project highlighted the strengths of NEAT in evolving both the structure and weights of neural networks, allowing the AI to adapt to varying challenges in the game environment. While the implementation faced challenges, such as computational demands and the need for careful parameter tuning, the outcomes affirmed the algorithm’s effectiveness in this application.

This project provided valuable insights into combining AI with game development and demonstrated the potential of neuroevolution for tasks requiring adaptability. Future improvements could include optimizing parameters, introducing more complex game mechanics, and exploring multi-agent scenarios.

In conclusion, the project not only met its objectives but also served as a stepping stone for further exploration of AI-driven game applications.

8. References

https://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf?

http://arxiv.org/pdf/2112.03670

https://www.toolify.ai/ai-news/mastering-ai-with-neat-algorithm-implementation-training-and-results-2381620?

9. Appendix

bg.png

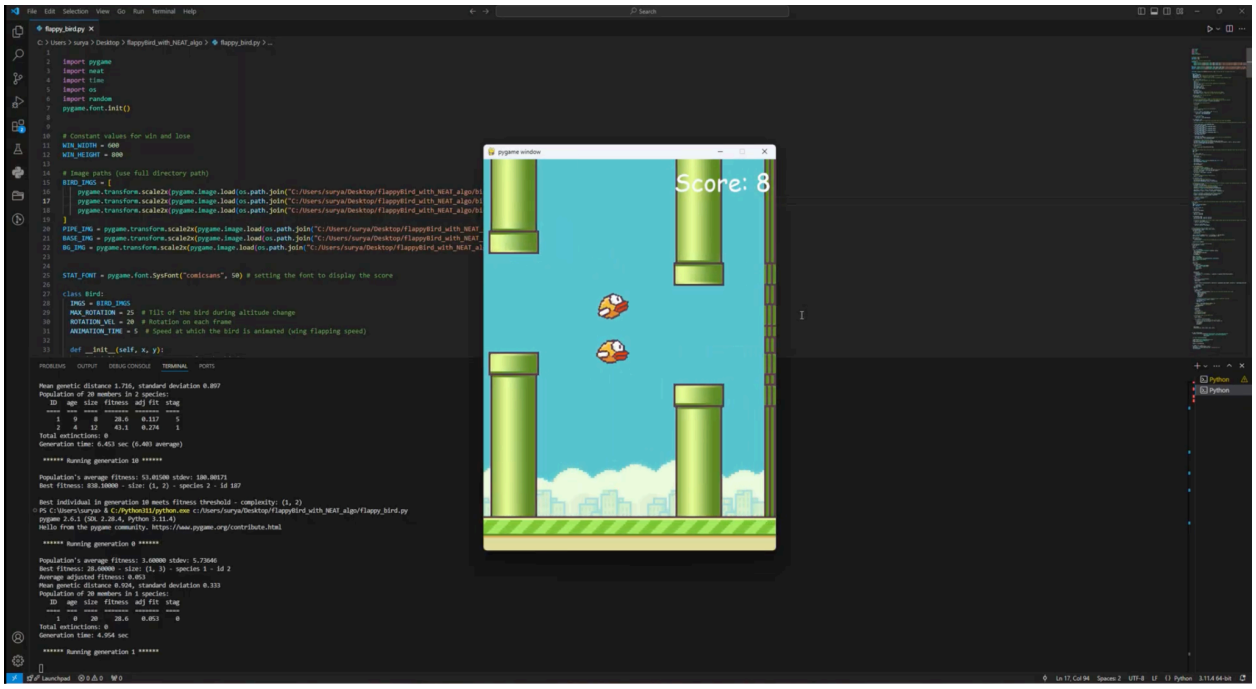
Base.png

bird1.png and bird2.png

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 100
pop_size              = 20
reset_on_extinction   = False

[DefaultGenome]
# node activation options
activation_default      = tanh
activation_mutate_rate  = 0.0
activation_options      = tanh

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 30.0
bias_min_value          = -30.0
bias_mutate_power       = 0.5
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.1

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.5
conn_delete_prob        = 0.5

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.01

feed_forward            = True
initial_connection      = full

# node add/remove rates
node_add_prob           = 0.2
node_delete_prob        = 0.2
```

```
# network parameters
num_hidden              = 0
num_inputs              = 3
num_outputs             = 1

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 30.0
response_min_value      = -30.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 30
weight_min_value        = -30
weight_mutate_power     = 0.5
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 20
species_elitism      = 2

[DefaultReproduction]
elitism           = 2
survival_threshold = 0.2
```

Config_file.txt



Game demo