# Neuroevolution-Based Game Bot for T-Rex Runner

## Applying NEAT and Comparing with DDPG & PPO

J. Chenye W. Shaoyan W. Xiaoxu Q. Yonghui and W. Haoran

Yanshan University & Silesian University of Technology

Qinhuangdao / Gliwice, May 2025

# Table of Contents

# Authors' Contributions

## Overview

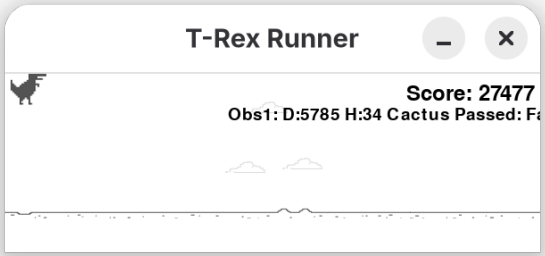
The following table summarizes the main contributions of each team member to the project:

| Author | Contribution |
|---|---|
| Jin Chenye | 30% Write NEAT code, front-end code, and AI explainability exploration, network evolution video (with ffmpeg) |
| Wang Shaoyan | 30% Write PPO, DDPG code, modify game code, and conceive artificial intelligence network design. |
| Wang Xiaoxu | 15% Design of PPO network, conception of NEAT artificial intelligence network design, and summary of achievements. |
| Qi Yonghui | 15% Design of DDPG network, conception of NEAT artificial intelligence network design, and summary of achievements. |
| Wu Haoran | 10% Conceived NEAT artificial intelligence network design, and proposed comparative ideas. |

# Project Overview: T-Rex Runner Automation

## Project Goal

To develop an intelligent agent capable of playing the Chrome T-Rex Runner game autonomously using biologically inspired AI techniques, primarily NeuroEvolution of Augmenting Topologies (NEAT).




T-Rex Runner Game Interface

## Motivation

- Explore the capabilities of neuroevolution in dynamic game environments.
- Understand the process of designing fitness functions for evolutionary algorithms.
- Compare NEAT's performance against other Reinforcement Learning (RL) methods like DDPG and PPO.

# Project Requirements

This project aligns with "Project 3: Neuroevolution-Based Game Bot".

## Core Requirements Met

- Create a simple game-playing bot (T-Rex Runner instead of Flappy Bird).
- Bot learns by evolving its neural network (brain) over time.
- Observe bot improvement over generations.
- Recommended Tools: Python, Pygame, NEAT-Python.

## Output Requirements Met

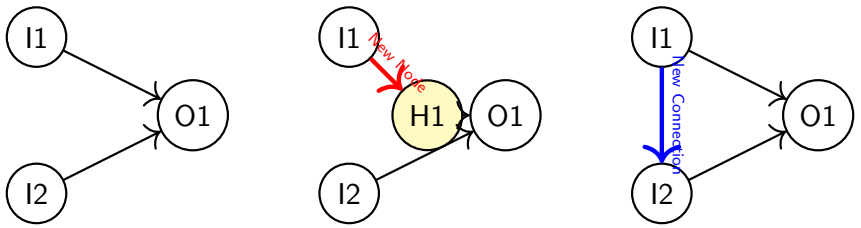
- ✔ Functional game bot that learns to play.
- ✔ Training performance curve showing improvement.
- ✔ Video recording or GIF of gameplay (planned for demo).
- ✔ Well-structured code using NEAT-Python.
- *Bonus:* Comparison with DDPG and PPO, detailed analysis of training

# NeuroEvolution of Augmenting Topologies (NEAT)

## What is NEAT?

A sophisticated GA that evolves not only the weights of a neural network but also its topology (structure).

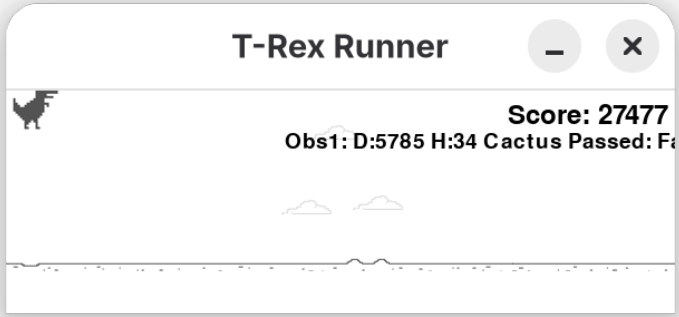
- **Starts Minimal:** Begins with simple networks and gradually adds complexity (nodes and connections).
- **Historical Markings (Innovation Numbers):** Tracks gene origins to solve the "competing conventions" problem in crossover. Allows meaningful crossover between different topologies.
- **Speciation:** Divides the population into species based on topological similarity. Protects innovation by allowing new structures time to optimize within their own niche.



# The T-Rex Runner Game Environment

## Environment ('aiGame.py')

- Implemented using **Pygame**.
- Player (T-Rex) must avoid cacti and birds.
- Game speed gradually increases.
- Provides state information to the agent and receives actions.




Game Sprites from 'resources.png'

## Key Features

- Obstacle generation logic.
- Collision detection.
- Scoring mechanism.
- Player can: Jump, Duck, No-op.

# State Representation & Action Space (NEAT)

## State Vector (Inputs to Neural Network)

The environment provides a 14-dimensional state vector, normalized for better NN performance:

1. Player Y position
2. Player vertical velocity
3. Is crouching (binary)
4. Obstacle 1: X distance
5. Obstacle 1: Y position
6. Obstacle 1: Width
7. Obstacle 1: Height
8. Obstacle 1: Is bird (binary)

(Normalization details in ‘DDPG_train.py‘ and ‘PPO_train.py‘, applied consistently for NEAT)

# State Representation & Action Space (NEAT)

## Action Space (Outputs from Neural Network)

The NEAT network outputs scores for 3 discrete actions:

- **Action 0**: No-op (continue running)
- **Action 1**: Jump
- **Action 2**: Duck / Fast-fall (if already jumping)

The action with the highest output score is chosen.

# Fitness Function Design (NEAT) - *Highlight*

The fitness function is crucial for guiding evolution. Our design:

## Base Fitness Components

- **Survival Reward**: ‘+0.1‘ for each game tick the T-Rex survives. Encourages longevity.
- **Obstacle Pass Reward**: ‘+10‘ for successfully passing an obstacle. Encourages progress.
- **Crash Penalty**: ‘-1‘ (implicitly, as game ends and fitness accumulation stops, also a fixed penalty can be added). Discourages dying.

# Bird Penalty Logic in eval_genomes

```
# state, reward, done = game.step(action)
# fitness += reward # Original part
bird_penalty = 0
overlap_threshold = 5 + (state["obs1_width"] if state["obs1_is_bird"] else 0)

if state["obs1_is_bird"] and (abs(state["obs1_dist_x"]) < overlap_threshold):
    bird_y = state["obs1_y"]
    if bird_y <= 85:  # High-flying bird (should duck)
        if action == 1: bird_penalty -= 20 # Penalty for jumping
        elif action == 0: bird_penalty -= 10 # Penalty for no-op
        # elif action == 2: bird_penalty += 0 # Correct action (duck)
    elif bird_y >= 125:  # Low-flying bird (should jump)
        if action == 2: bird_penalty -= 20 # Penalty for ducking
        elif action == 0: bird_penalty -= 10 # Penalty for no-op
        # elif action == 1: bird_penalty += 0 # Correct action (jump)
    else:  # Mid-height bird
        if action == 0: bird_penalty -= 10 # Penalty for no-op if action is better

state, reward, done = game.step(action) # Action performed AFTER penalty logic
fitness += reward + bird_penalty # Add bird penalty to fitness
```

Listing 1: Excerpt from ‘NEAT/train_neat.py‘ showing the bird penalty logic.

# NEAT Training Process & Seed Strategy - *Highlight*

## NEAT Configuration (‘neat-config.txt‘)

- Defines population size, fitness thresholds, mutation rates, speciation parameters, etc.
- Key parameters:
  - ‘pop_size = 100‘
  - ‘fitness_criterion = mean‘
  - ‘num_inputs = 14‘, ‘num_outputs = 3‘
  - Various mutation probabilities (node, connection, weight).

# NEAT Training Process & Seed Strategy - *Highlight*

## Training Loop (‘train_neat.py‘)

- Iterates for a specified number of generations.
- In each generation, ‘eval_genomes‘ evaluates all genomes.
- NEAT library handles selection, crossover, mutation, and speciation.
- Checkpoints and best genomes are saved periodically and on new best fitness.
- Detailed statistics (best/avg/std fitness, species count, genome complexity) are logged and saved.

# NEAT Training Process & Seed Strategy - *Highlight*

## Seed Strategy - *Key for Reproducibility & Exploration*

A two-phase seed strategy was implemented:

- **Phase 1 (Fixed Seed):** For the initial ‘fixed_seed_generations‘ (e.g., first 100-200 generations) OR until a ‘min_fitness_for_random‘ (e.g., 3000) is achieved.
  - *Purpose:* Provides a consistent environment for early learning, allowing promising basic structures to emerge without being overly penalized by random difficult game sequences. Helps in comparing initial learning trajectories.
- **Phase 2 (Random Seed):** After the conditions for Phase 1 are met.
  - *Purpose:* Exposes the evolving agents to a wider variety of game scenarios, promoting robustness and generalization. Prevents overfitting to a single game seed.

# DDPG and PPO for Benchmarking

To compare NEAT with mainstream DRL methods, DDPG and PPO were adapted to the T-Rex game:

## DDPG

- Actor-critic, model-free.
- Adapted for discrete actions (Gumbel-Softmax).
- Replay buffer + target networks.
- Files: `DDPG/DDPG_agent.py`

## PPO

- Actor-critic, model-free.
- Clipped surrogate objective for stable updates.
- Batching experience updates.
- Files: `PPO/ppo_agent.py`

## Common Features

- Use same 14-dim state as NEAT.
- Output discrete actions (0, 1, 2).
- Trained in the same environment.

# Project Directory Structure

```
.
 DDPG/
    aiGame.py
    DDPG_agent.py
    DDPG_train.py
 NEAT/  (Primary Focus)
    aiGame.py (T-Rex Environment)
    neat-config.txt (NEAT Parameters)
    train_neat.py (Training Script)
    play_trex_neat.py (Run Best Agent)
    visualize.py (Network Plotting)
    server.py (Web Visualization Interface)
    saved_models/ (Checkpoints, Genomes, Stats)
       neat-checkpoint-X
       neat-best-genome-overall.pkl
       neat_stats_data.pkl
    resources.png (Game Assets)
 PPO/
    aiGame.py
    ppo_agent.py
    ppo_train.py
 ... (Other files like LICENSE)
```

# Challenges & Solutions

## Challenge 1: Slow Initial Learning / Stagnation

- **Problem:** Agents struggled to learn basic survival or make meaningful progress.
- **Solution:**
  - *Refined Fitness Function:* Added specific rewards for passing obstacles and the strategic bird penalty to guide learning more effectively.
  - *Seed Strategy:* Used fixed seed initially to provide a consistent learning environment, reducing early randomness.

# Challenges & Solutions

## Challenge 1: Slow Initial Learning / Stagnation

- **Problem:** Agents struggled to learn basic survival or make meaningful progress.
- **Solution:**
  - *Refined Fitness Function:* Added specific rewards for passing obstacles and the strategic bird penalty to guide learning more effectively.
  - *Seed Strategy:* Used fixed seed initially to provide a consistent learning environment, reducing early randomness.

## Challenge 2: Optimal Action Against Birds

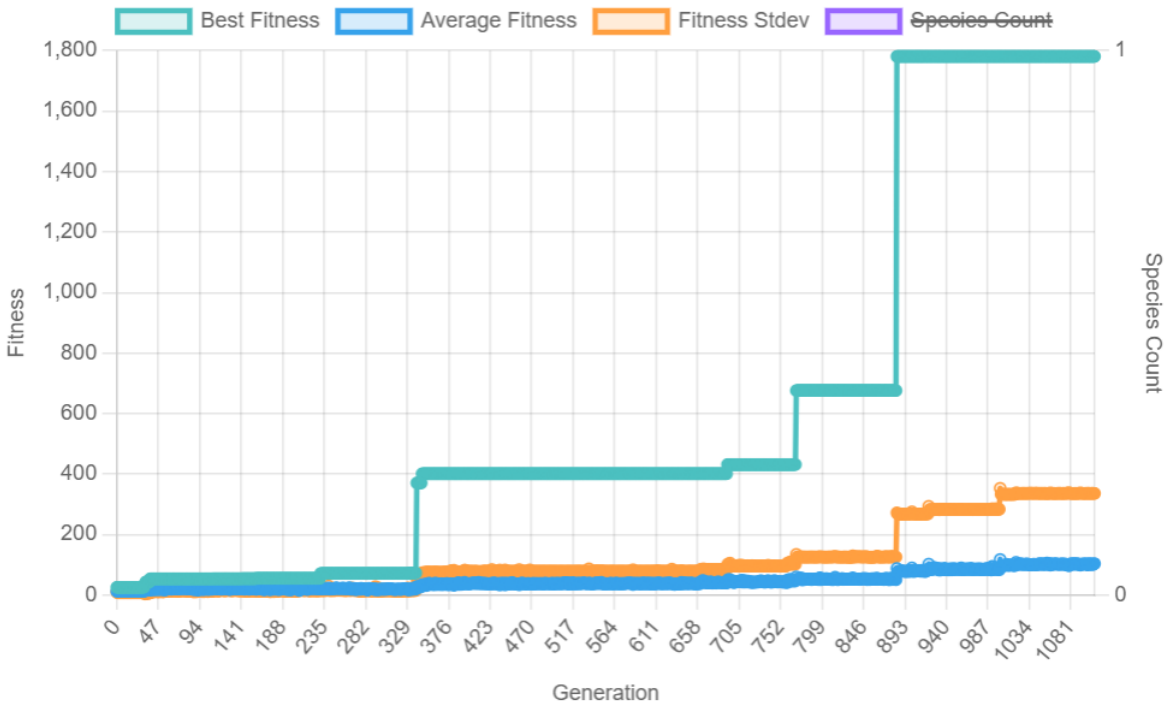
- **Problem:** Agents often failed to distinguish between high and low birds, leading to suboptimal actions (e.g., jumping into a high bird).
- **Solution:** The ‘bird_penalty‘ system in the fitness function directly addresses this by penalizing incorrect actions based on bird altitude.

# Challenges & Solutions

## Challenge 3: Long Training Times & Resource Management

- **Problem:** NEAT can be computationally intensive, requiring many generations.
- **Solution:**
  - Implemented robust checkpointing to resume training.
  - Optimized ‘aiGame.py‘ for faster simulation where possible.
  - Focused fitness function to accelerate learning of crucial behaviors.

# NEAT Training Performance

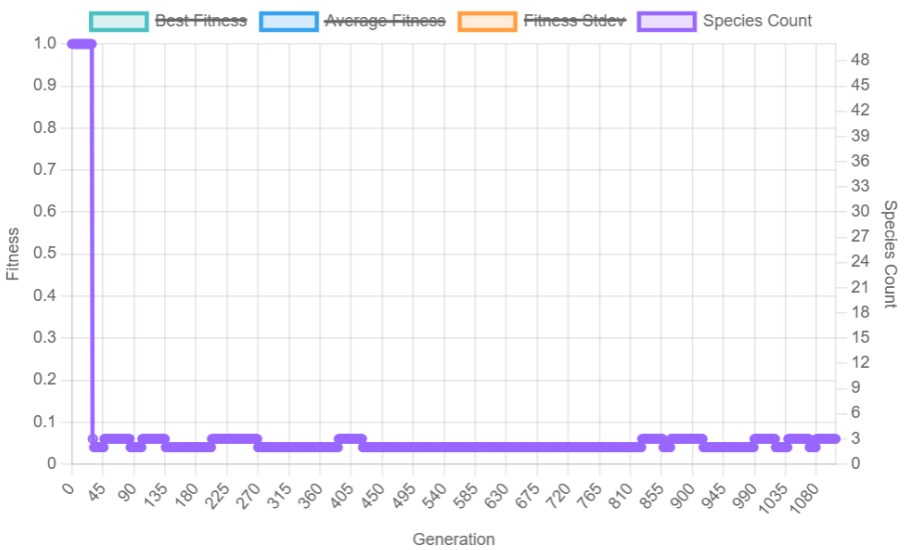



Best, Average, and Std Dev Fitness over Generations.

## Observations

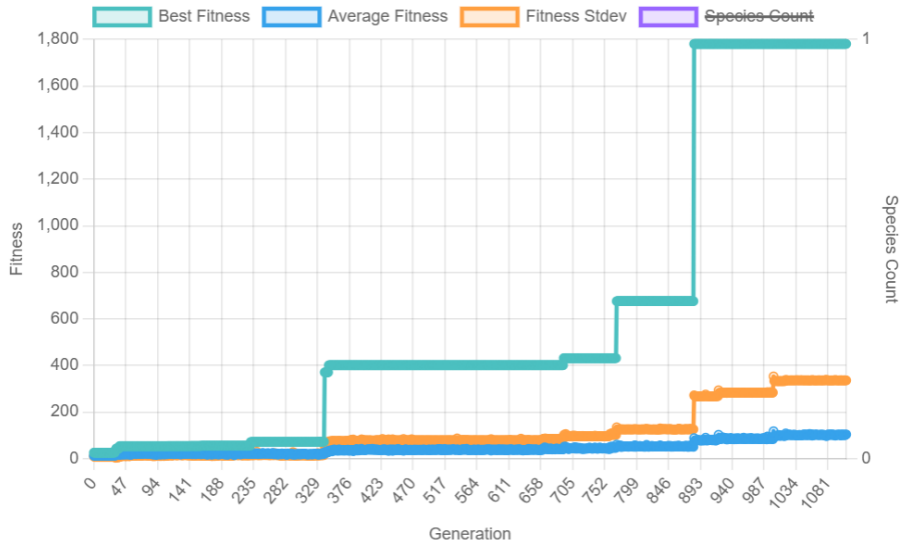
- Steady increase in best fitness.
- Average fitness also shows upward trend.
- Standard deviation indicates diversity in population.
- Note periods of stagnation followed by breakthroughs (typical of EAs).
- (Mention when seed strategy switched if visible effect).

# NEAT Speciation and Genome Complexity




Number of Species Over Generations.

- Shows dynamic nature of speciation.
- New species emerge, some die out.
- Helps protect innovation.




Complexity (Nodes & Connections) of Best Genome.

- Generally, complexity increases over time as NEAT finds more sophisticated solutions.

# Further Insights into NEAT Evolution

## Network Evolution Trends

Observations from analyzing training process videos (e.g., ‘complexity_evolution.mp4‘, ‘fitness_evolution.mp4‘, ‘network_rankX_evolution.mp4‘):

- Network evolution primarily converged towards two main structural directions, often based around specific persistent hidden nodes (e.g., nodes analogous to 47 and 12 in certain successful lineages).
- The fitness distribution among the population tended to become more uniform over extended training, indicating a convergence towards high-performing solutions or niches.

## Genome Complexity Observations

- Genome complexity (number of nodes and connections) for high-performing individuals often stabilized within a range of approximately 5-12 nodes and 15-23 connections.
- There wasn't always a strong direct correlation between higher complexity and higher fitness in later stages; effective, relatively simpler structures often persisted.

*These insights are based on detailed analysis of recorded training data and videos.*

# Key Moments in Training Evolution

## "Cretaceous Extinction" Event (Generations 29-30)

- Around generations 29-30, a significant reduction in species diversity was observed.
- This suggests a period where many less-fit genetic lines were unable to adapt and died out, a sort of "survival of the fittest" bottleneck.
- This evolutionary pressure likely paved the way for more robust strategies to emerge from the surviving species.

# Key Moments in Training Evolution

## "Cretaceous Extinction" Event (Generations 29-30)

- Around generations 29-30, a significant reduction in species diversity was observed.
- This suggests a period where many less-fit genetic lines were unable to adapt and died out, a sort of "survival of the fittest" bottleneck.
- This evolutionary pressure likely paved the way for more robust strategies to emerge from the surviving species.

## The "Aha Moment" (Generations 246-247)

- A dramatic jump in maximum fitness occurred between generations 246 and 247.
- The average fitness also showed a corresponding oscillating increase.
- Analysis revealed that at this point, the agent learned to overcome a specific, previously insurmountable small cactus obstacle. This breakthrough represents a critical learning milestone.

# The 'Aha Moment': Conquering a Critical Obstacle (Gen 246 vs. 247)


**Generation 246:** Agent consistently fails at this specific small cactus.


**Generation 247:** Agent successfully learns to navigate the same obstacle.

## Significance

This transition clearly demonstrates a specific learning event where the evolutionary process discovered a strategy (likely a subtle change in network weights or topology) to overcome a persistent challenge, leading to a significant performance boost.

# Analysis of Best Evolved NEAT Agent

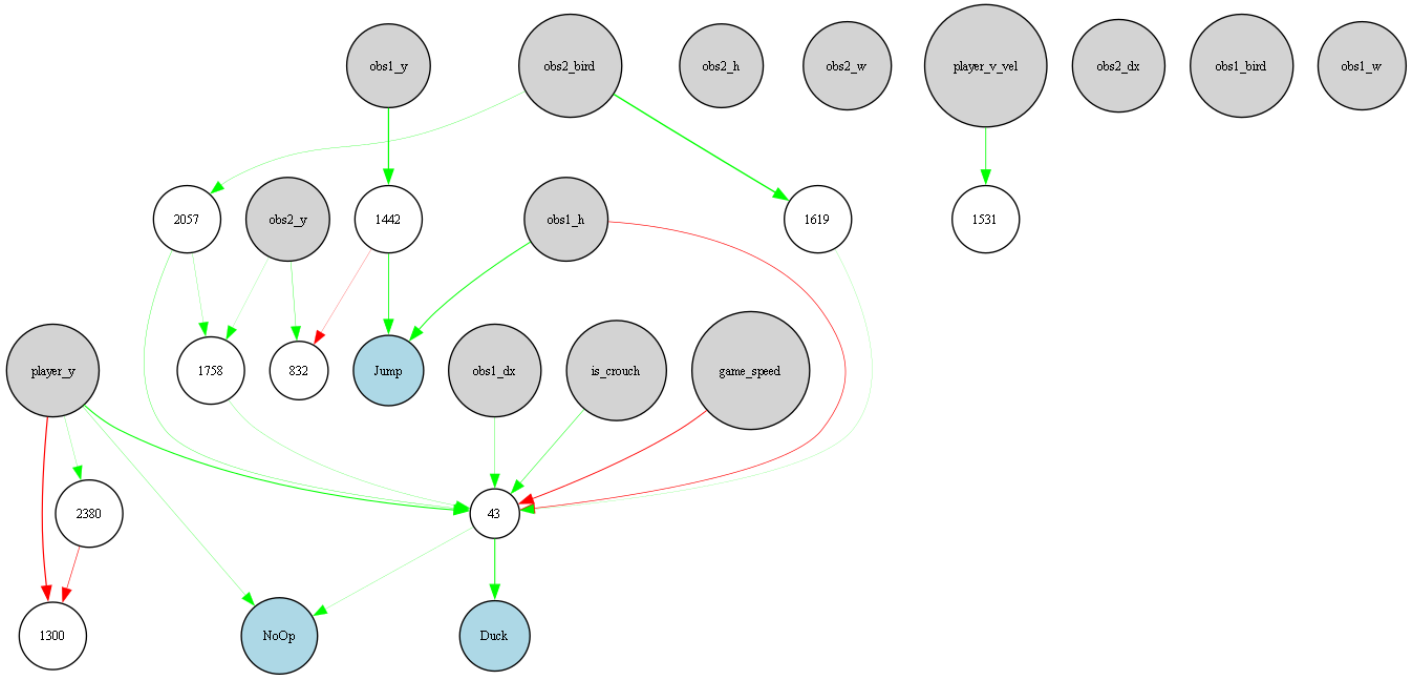



Figure: Network Topology of the Best Evolved Genome.

Network Topology of the Best Evolved Genome.

## Best Genome Details

- Achieved a best fitness $> 10000$
- Generation found: 1099
- Number of nodes: 20
- Number of connections: 17

# Analysis of Best Evolved NEAT Agent

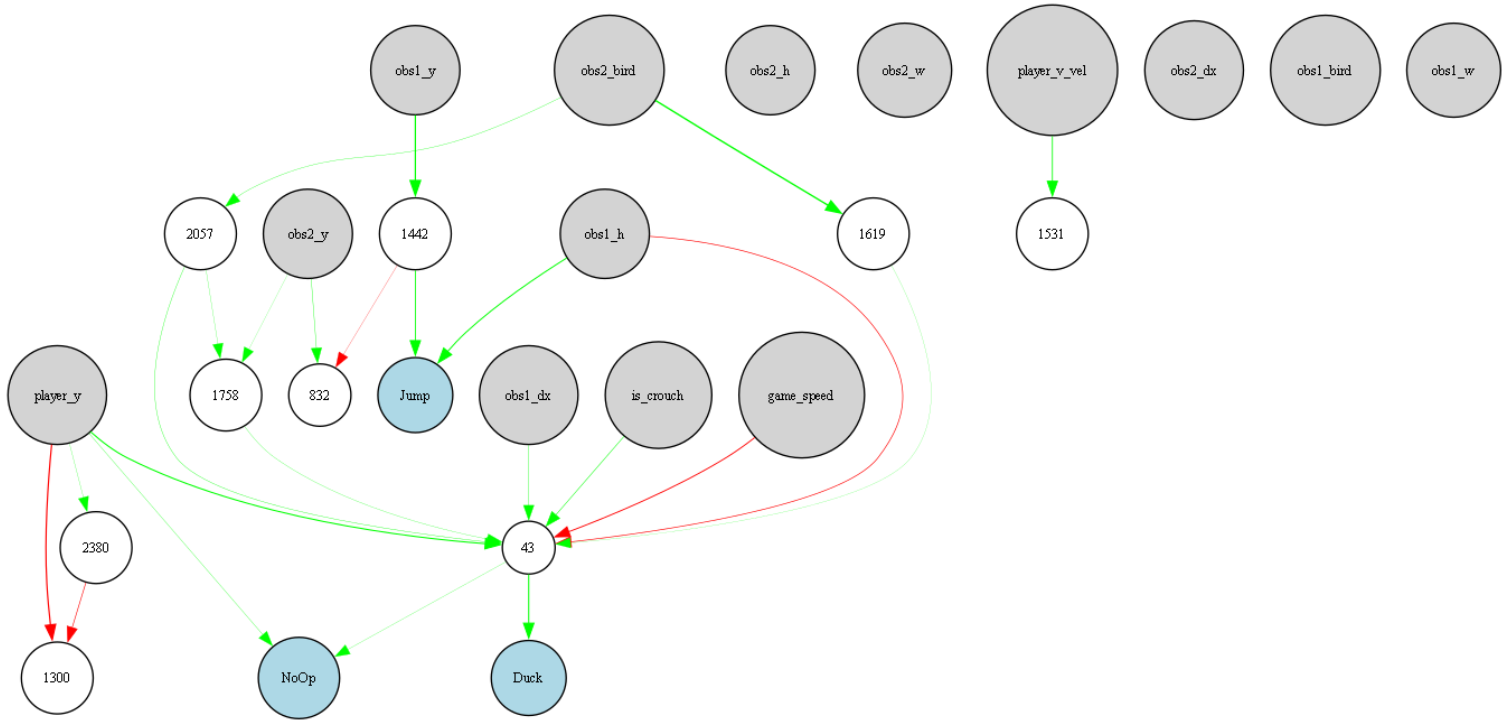



Figure: Network Topology of the Best Evolved Genome.

Network Topology of the Best Evolved Genome.

## Qualitative Behavior

- Successfully navigates most cacti.
- Demonstrates improved ability to handle birds due to the specialized fitness component.
- Shows adaptive behavior as game speed increases.

# Comparison with DDPG and PPO

To benchmark NEAT's performance, two common Deep Reinforcement Learning algorithms were implemented for the T-Rex game:

Table: Comparative Performance

| **Metric** | **NEAT** | **DDPG** | **PPO** |
|---|---|---|---|
| Best Score (Avg over N runs) | **>10000** | 25 | 38 |
| Training Time (Generations/Steps) | ∼2000 Gens | ∼2000 Steps | ∼2000 Steps |
| Stability | Fluctuating | More Stable | Quite Stable |
| Interpretability of Policy | High (Network visible) | Low (Deep NN) | Low (Deep NN) |
| Hyperparameter Tuning Effort | Moderate | High | Moderate-High |

- NEAT often finds novel, interpretable solutions.
- DDPG/PPO require more tuning, but may outperform NEAT with time.

# Live Demonstration / Video

## Showcasing the Trained Agent

- We proposed a video (or a live demo, if you like ^^) to show the best NEAT-trained T-Rex agent in action.
- Highlights:
  - Consistent obstacle avoidance.
  - Handling of increasing game speed.
  - Reactions to different bird types (if clearly visible).
- We can also showcase the ‘server.py‘ interface for model analysis if time permits.

▶ Gameplay Video

# NEAT Evolution / Video

## Showcasing the Trained Agent

- We also proposed some videos to show how the NEAT-trained T-Rex agent evolves in time.
- Choose one or some:
  - complexity_evolution.mp4
  - fitness_evolution.mp4
  - network_rank1_evolution.mp4
  - network_rank2_evolution.mp4
  - network_rank3_evolution.mp4
  - network_rank4_evolution.mp4
  - network_rank5_evolution.mp4
- We pay much efforts on explainable solution, if time permits please let us show those videos.

Video Time

# Conclusion

## Key Achievements

- Successfully applied NEAT to train an autonomous agent for the T-Rex Runner game.
- Developed a sophisticated fitness function, including strategic penalties/rewards for bird encounters, which significantly guided learning.
- Implemented a two-phase seed strategy, balancing initial learning consistency with later generalization.
- The NEAT agent demonstrated competent gameplay, adapting to increasing difficulty, including learning specific challenging obstacles.
- Identified key evolutionary events like species bottlenecks and learning breakthroughs ("Aha moments").
- Implemented DDPG and PPO for comparison, providing insights into different AI approaches for this task.
- Created a Python/Flask/HTML based tool for checkpoint analysis

# Conclusion

## Lessons Learned

- Fitness function design is paramount in evolutionary algorithms.
- NEAT's ability to evolve topology is powerful for tasks where the optimal network structure is unknown.
- Careful management of training parameters (like seeding) can impact learning efficiency and outcome.
- Observing detailed evolutionary dynamics (like species changes and fitness jumps) provides deeper understanding of the learning process.

# Future Work

## Short-term Goals

- **Extensive Hyperparameter Tuning:** For NEAT, DDPG, and PPO to achieve peak performance.
- **More Sophisticated State Normalization:** Explore adaptive normalization techniques.
- **Advanced Fitness Shaping:** Introduce more nuanced rewards/penalties (e.g., for near misses, or for efficient energy use if applicable).

## Long-term Plans

- **More Complex Games:** Apply the learned principles to games with larger state/action spaces. **Hybrid Approaches:** Explore combining NEAT with other learning techniques (e.g., using NEAT to find a good network topology for an RL agent).
- **Multi-objective NEAT:** Optimize for multiple criteria simultaneously (e.g., score and network simplicity).

# Thank You!

Questions?