Evolving Neural Agents in Simulated Ecosystems

Marija Ćetković
UP FAMNIT
Koper, Slovenia
marijacetkovic03@gmail.com

Aleksandar Tošić
UP FAMNIT
Koper, Slovenia
aleksandar.tosic@upr.si

Domen Vake
UP FAMNIT
Koper, Slovenia
domen.vake@famnit.upr.si

Abstract

This paper explores how adaptive behaviors can emerge in artificial agents through neuroevolution in a dynamic 2D ecosystem. Using the NeuroEvolution of Augmenting Topologies (NEAT) algorithm both the neural network structure and weights evolved over time without predefined architectures or behaviors. The system models two agent types: herbivores and carnivores that compete for limited food resources in a simulated environment. From the beginning, it was evident that environment design, input encoding, and reward shaping had a major impact on agent behavior. Poorly tuned conditions led to exploitation, overfitting, or meaningless patterns. But when the system was carefully balanced, the agents began developing survival strategies such as movement efficiency, food seeking, and attacking. Herbivores evolved plant consumption behaviors, while carnivores built on this base to prioritize attacks and meat consumption. Some behaviors generalized well to larger environments, showing that agents were not just memorizing patterns. We observed how NEAT's speciation and innovation mechanisms were crucial for maintaining diversity and avoiding premature convergence. At the same time, challenges like catastrophic forgetting revealed the limitations of neural networks in long-term skill retention. Ultimately, this work demonstrates how intelligent, adaptive behavior can emerge from simple evolutionary principles and offers a foundation for future research into co-evolution, agent roles, and artificial life.

Keywords

neuroevolution, NEAT, evolutionary algorithms, artificial life, simulated ecosystems, co-evolution, neural networks

1 Introduction

This research explores neuroevolution for adaptive agent behaviors in a dynamic ecosystem. Agents are controlled by feedforward neural networks that map sensory inputs to actions [4], and their structure and weights evolve incrementally using the NEAT algorithm [5]. Unlike static or predefined tasks, this simulation presents agents with a changing environment where no explicit 'correct' behavior exists.

Dynamic environments without fixed objectives require exploratory and adaptable approaches. Gradient-based optimization relies on differentiable fitness functions and fixed topologies, while reinforcement learning can struggle under sparse rewards. Evolutionary algorithms, by evaluating populations of agents directly on survival and performance, provide a natural solution for such open-ended scenarios [1]. Neural networks allow agents to flexibly map sensory input to actions, and NEAT enables both the network topology and weights to evolve over time in comparison to fixed-topology weight-evolving evolutionary methods. We implemented NEAT from scratch to have full control over mutation, crossover, and fitness evaluation, ensuring that the system could support our experimental goals and to potentially build a controllable and extensible evolutionary framework.

While NEAT has been previously applied to multi-agent systems, many studies focus on performance in a specific task. This paper addresses whether an agent-based NEAT framework can produce ecological equilibrium without an explicit objective. Our primary contribution is the demonstration and analysis of stable, co-adaptive predator-prey dynamics, showing how specific evolved behaviors arise from the underlying neural network topologies of the agents.

2 Methods

2.1 Environment Model

The ecosystem is a discrete 2D grid populated with food resources and agents. Herbivores consume plants, carnivores consume meat, and all agents perceive their surroundings through a limited sensory range.

2.2 Evolutionary Framework

Agents (creatures) interact with the world and are controlled by neural networks (genomes) evolved using NEAT. Initial populations start with minimal structures (fully connected input/output layers), and complexity increases through structural mutations. Genomes consist of genes, which are lists of nodes (with ID and type: input, hidden, output) and connections (with ID, nodes they connect, weight and enabled flag). Each tick, each agent receives a snapshot of the world state as input, to ensure stable input for everyone. Inputs include diet type, hunger level, local 3x3 neighborhood scan for food, neighbors (type and health level), and direction toward the nearest food source. Based on that agents choose actions as softmax output of their neural networks. The outputs correspond to discrete actions: move (up, down, left, right), eat, attack, or stay. The actions become events that are handled in a deferred manner. First, the invalid actions are filtered out, then the EventManager processes all queued events at once sequentially: applying changes in the world, updating fitness, and health of agents, which can be seen in the Algorithm 1.

The fitness function evolved through experimentation. Early versions rewarded survival, but later iterations combined survival time, food consumption, and for carnivores, attack behavior.

2.3 NEAT Mechanisms

Innovation Tracking is the process of tracking structural mutations globally to keep genomes aligned during crossover. A singleton class assigns a unique ID to each new connection or node. If a structural change already exists, it reuses the same ID; if not, it creates a new one. This ensures a consistent identification of identical innovations in all genomes [5].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
Information Society 2025, Ljubljana, Slovenia
© 2025 Copyright held by the owner/author(s).
https://doi.org/10.70314/is.2025.sikdd.10

```
while generation limit not reached do
    // Simulation Phase
    while creatures alive do
        foreach creature c in population do
            c.observe(world) ;            // perception
            c.chooseAction() ;             // NN policy
            c.queueAction() ;        // check validity,
             enqueue
        end
        eventManager.process() ;       // apply action
         effects
        foreach creature c in population do
            c.updateHealth() ;   // starvation, death
            c.evaluateFitness() ;     // assign fitness
        end
    end
    // Evolution Phase
    assignGenomesToSpecies() ;          // speciation
    createOffspring() ;   // apply GA within species
    resetWorld() ; // spawn new creatures and food
end
```

Algorithm 1: High-Level Evolutionary Simulation Loop

NEAT preserves evolutionary innovation by speciation (niching) [5]. Each generation, evaluated genomes are reassigned to species based on structural compatibility distance, which is calculated as a weighted sum of the number of disjoint and excess connections (present in one parent, within and beyond other's genome region respectively), and weight difference averages between the matching (present in both parents) ones, given by: $\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot W$. Existing species are cleared and each genome is compared to species representatives; if no match is found, a new species is created. Representatives are updated every generation to maintain diversity. Fitness is shared within species (adjusted by species size) to balance selection pressure. The compatibility threshold strongly affects stability: low thresholds create many narrow species, high thresholds create broader but less distinct species, requiring careful tuning.

To prevent the population from maintaining one dominant species and limiting the exploration of the algorithm, NEAT uses adjusted fitness [5]. Instead of assigning raw fitness scores, the individual's fitness is adjusted by the number of individuals who are within its distance delta, given by: $f_i' = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))}$.

Evolution is achieved through genetic operators within species:

Mutation: Weight changes (random reset 5–10% or small Gaussian perturbation) and structural changes (adding nodes or edges, toggling connections). Resulting genome is checked for acyclicity.

Crossover: Offspring inherit connections aligned by innovation number; matching ones are inherited from the first parent, while disjoint and excess come from the parent with greater fitness score (or random if equal). Invalid (cyclic) offspring are replaced by mutated fitter parent.

Selection: Parent selection uses tournament selection: a subset of individuals (size 5) is sampled and the fittest is chosen. With 3% probability, a random individual is selected to maintain diversity. This setup provides moderate selection pressure - avoiding premature convergence while keeping implementation simple, efficient, and robust across different fitness functions.

2.4 Graphical User Interface

Figure 1 shows an example of the GUI which serves to visually track the simulation in real time, making the evolutionary process observable and interpretable, as analyzing logs alone could be misleading. It allowed following the population changes over generations, spotting emerging behaviors such as movement patterns or interactions, and understand whether agents are actually evolving. It helps detect issues such as creatures moving in the same direction or wandering aimlessly.

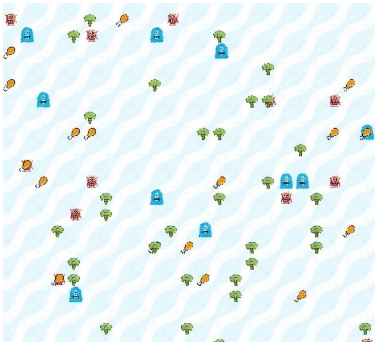


Figure 1: GUI close-up

2.5 Implementation Notes

The simulation was implemented in Java with LibGDX [2] for visualization. NEAT logic included custom classes for genomes, species management, and innovation tracking. The evolutionary loop evaluated agents in the world, assigned fitness, reproduced genomes, and reset the environment for subsequent generations.

2.6 Setup

After every run around 10 percent of the population is saved and loaded for the next run, with that part of population unchanged and the rest filled with mutations of it. This is done to speed up the evolution process. In early runs, we disabled the perception of other agents to prevent confusion and help them learn basic eating behavior. Once they consistently moved and consumed food, perception was turned on to allow them to adapt to a more complex environment. We also tested this logic on other inputs such as the food direction vector left agents essentially 'blind' to non-local food. So, during early iterations, we spawned food in random concentrated areas rather than spreading it widely, to help them learn to use this vector.

3 Results

3.1 Herbivore Evolution

Herbivores initially explored aimlessly but gradually developed stable food-seeking strategies. Over 800 generations, their action distribution stabilized, with movement actions dominating and eating consistently rewarded. In larger environments, agents prioritized exploration to reach scarcer resources, showing emergent adaptation beyond memorized patterns.

We can see from Fig.2 that the initial fitness highly oscillates, with great difference in average and maximal fitness, as well as some outliers with high fitness who end up consuming a lot of food. This is expected to some extent, as when one creature consumes food, it reduces the available resources for others in the population.

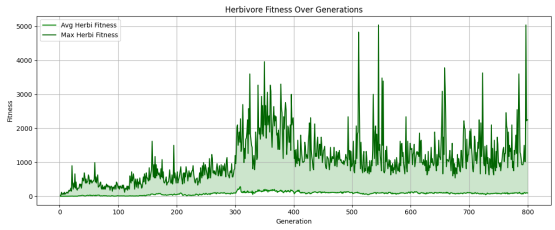


Figure 2: Herbivore fitness

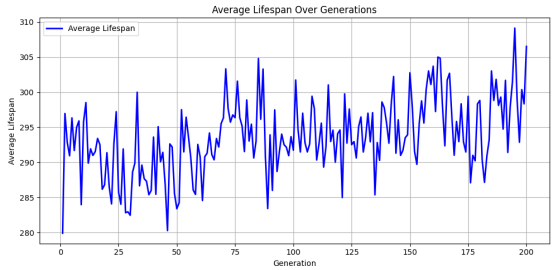


Figure 3: Average creature lifespan

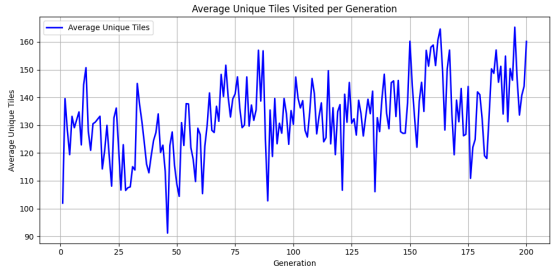


Figure 4: Average number of unique tiles visited

Figures 3 and 4 show agents that survive longer naturally explore more of the environment. The observed correlation between emerges from adaptive behavior under environmental constraints, rather than from any explicit exploration rewarding.

3.2 Carnivore Adaptation and Catastrophic Forgetting

Carnivores were evolved by reusing herbivore topologies and adjusting weights, transferring eating behavior to meat sources. This transfer was successful in just 200 generations, but the agents showed catastrophic forgetting when switched back to herbivore roles, losing previous behaviors [3]. This showed us that we needed more general pretraining to make sure that agents were using their role, food and food vector inputs, and not overfitting to the food type.

3.3 Co-Evolution Dynamics

To try to avoid the problem of forgetting mentioned, we saved agents of both types that evolved their basic skills independently. When carnivores were alone we gave them no motivation to use the attack action, to wire the logic later to herbivores. The attacking behavior was rewarded only for carnivores, but as shown below, some role interference was inevitable.

In smaller worlds, herbivores focused on eating, carnivores split between eating and attacking; in larger worlds, carnivores prioritized attacking, herbivores balanced movement and eating.

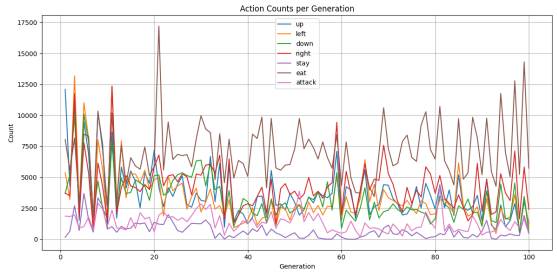


Figure 5: Herbivore action distribution 100x100 world

In the beginning, the actions chosen were randomized, but Figure 5 shows how herbivores learned to prioritize the eating action, although initial interference is evident. The usage of stay and attack actions is low.

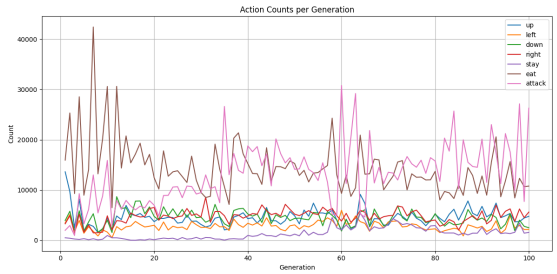


Figure 6: Carnivore action distribution 100x100 world

In Fig.6 we can spot how carnivores experience problems in balancing the eating and attacking action, but the attacking action slightly dominates after some time.

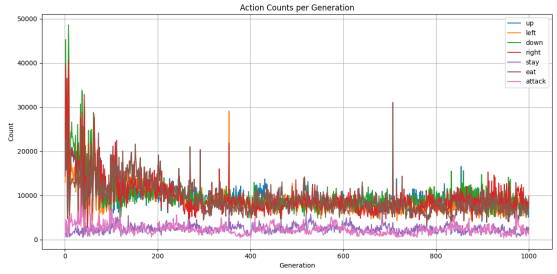


Figure 7: Herbivore action distribution 200x200 world

Figure 7 displays herbivore behavior, where the action distribution is more stable and there is a clear evolved balance of eating and moving actions, which is expected in a larger world.

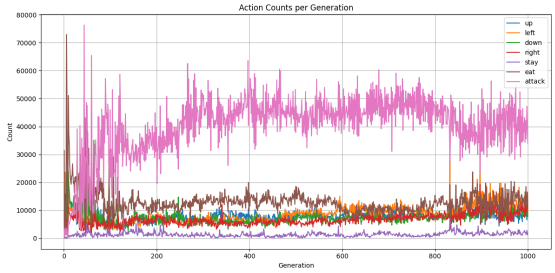


Figure 8: Carnivore action distribution 200x200 world

Figure 8 displays carnivore behavior in the larger world, where they were given a greater incentive to attack. From the distribution, we can see that they indeed attacked more, with the other actions being balanced out, and the staying action was rarely chosen.

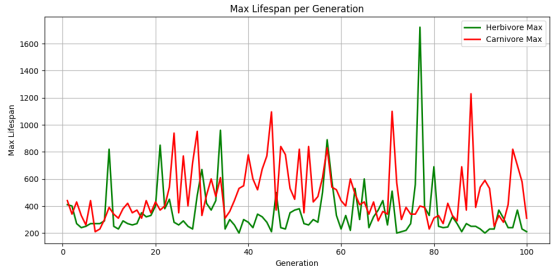


Figure 9: Maximum lifespan in the coevolutionary setting

Fitness (as well as the lifespan depicted in Figure 9) fluctuated in an "arms race" pattern with no dominant winner. This outcome is expected, as the rise in one role's performance lowered the performance of the other. This shows that the system tended toward balance, which aligns with the objective of testing whether coevolution with NEAT agents could produce equilibrium.

3.4 Species Diversity

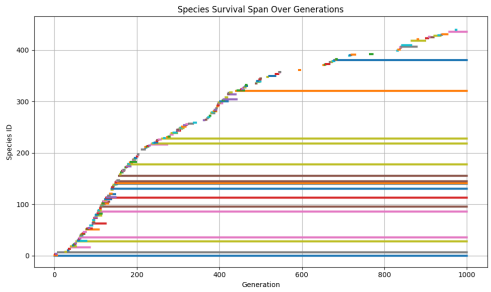


Figure 10: Species diversity over generations.

The survival plot of emerging species in Figure 10 shows an important aspect of the NEAT algorithm. The initial drop means that a few very successful topologies dominated the population, but using a lower compatibility threshold prevents the total loss of diversity. The number of species stabilized after some time, while many smaller species died out quickly.

4 Design Observations

Agent behavior is highly sensitive to design choices in fitness functions, environment setup, and input representation. Poorly designed fitness functions can lead to inefficient or trivial behaviors, such as flickering near food, which highlight the exploration-exploitation trade-off and the role of environmental influence in shaping behavior. Static or predictable resource spawn locations can cause overfitting, where agents memorize positions instead of learning general strategies. Dynamic and unpredictable environments are necessary to evolve general food-seeking behaviors and to observe which patterns emerge due to evolutionary pressures versus environmental conditions. However, environments that are too unpredictable can hinder learning and obscure the distinction between inherited tendencies and environment-induced behaviors.

Input scaling and initial placement also influence behavioral emergence. Unlimited health input caused agents to idle, while spawning agents too close together and awarding them for food consumption led to aimless wandering when neighbors died, showing correlations learned from the environment. These observations demonstrate how neural networks may pick up coincidental patterns that influence both relearning across generations and the adopted strategies.

Metrics did not always reflect consistent progress, as dynamic food spots and starting points introduced noise. Dips or peaks in performance do not necessarily indicate genuine failure or success. Adjustments to fitness, food rewards, and environmental parameters were required to guide learning, prevent reward hacking, and allow behavioral adaptation. Comparing herbivore and carnivore roles shows that behaviors are shaped by both environmental pressures and the interactions between agent strategies and resource availability. Agents adjust their actions based on the resources they encounter, and these actions influence which resources remain available, creating a feedback loop between behavior and the environment.

5 Conclusion and Future Work

This paper demonstrates that nature-like behaviors can emerge from relatively simple principles when agents evolve in dynamic, open-ended environments without predefined goals. By evolving herbivores and predators both separately and in co-evolution, we showed that evolutionary pressures can produce adaptive behaviors and predator-prey equilibria, highlighting how role-specific dynamics shape ecosystem stability. This work lays the foundation for future experiments that involve more complex behaviors, survival strategies, and deeper coevolutionary dynamics. Future directions could include investigating the potential of refined role awareness mechanisms, improved memory or learning retention, and more complex agent inputs and actions, enabling us to push the boundaries of what these agents can learn over time.

References

[1] A.E. Eiben and J.E. Smith. 2003. *Introduction to Evolutionary Computing. Natural Computing*. Springer-Verlag, Berlin.

[2] LibGDX. [n. d.] Libgdx game development framework. https://libgdx.com/. ().

[3] Michael Mccloskey and Neil J. Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, 24, 104–169.

[4] Michael A. Nielsen. 2018. Neural networks and deep learning. misc. (2018). http://neuralnetworksanddeeplearning.com/.

[5] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10, 2, 99–127. http://nn.cs.utexas.edu/?stanley:ec02.