Shorelark Expanded: Predator-Prey Coevolution in an Evolutionary Simulation

Isaac Salzman

2026-04-13

Abstract

Shorelark is an interactive evolutionary simulation that uses a genetic algorithm and feed-forward neural networks to model agent behavior in a simple environment. Building on Patryk Wychowaniec's "Learning to Fly" project, this work extends the original food-seeking simulation into a more flexible experimental platform. The key additions are predator-prey coevolution, live statistics, runtime parameter configuration, and a batch analysis pipeline. Together, these features transform the simulation from a static demonstration into a tool for observing and measuring how competing populations adapt under different selection pressures.

1. Introduction

Evolutionary algorithms are easiest to understand when their effects can be seen directly. A simulation where agents move, compete, and improve over time makes natural selection more concrete than a purely mathematical description, but only if the simulation is flexible enough to support real experimentation. Shorelark was built with that goal in mind.

The project extends Patryk Wychowaniec's "Learning to Fly" foundation, which established neural-network-driven agents evolving through a genetic algorithm in a food-seeking environment. My additions focus on two main areas. The first is predator-prey coevolution: rather than a single population adapting to a static environment, two populations now place continuous pressure on each other. The second is usability. Runtime controls let users adjust population sizes, field of view, photoreceptor counts, hidden-layer sizes, and movement speeds, and live statistics make the simulation easier to read as it runs.

A separate batch execution tool and plotting pipeline complement the interactive interface by enabling repeatable analysis across many runs. This report covers how these additions affect the simulation's usefulness as both a learning tool and an experimental system, and how different parameter choices shape coevolutionary behavior over time.

2. Technical Background

Shorelark is built on two core ideas: genetic algorithms and feed-forward neural networks. The neural network gives each agent a way to convert sensory input into movement decisions, and the genetic algorithm is the mechanism by which those networks improve over time. Together they create a system where useful behavior emerges through selection rather than being hand-programmed.

In Shorelark, each agent's "genome" is the set of numerical weights in its neural network. Each generation, agents are evaluated by their fitness and a new population is produced through selection, crossover, and mutation. Specifically, the project uses roulette-wheel selection, meaning higher-fitness individuals are more likely to be chosen as parents. Their weights are then combined through uniform crossover and modified with Gaussian mutation. This balances stability with exploration, preserving traits that have already worked while still introducing variation that might lead to better strategies later.

The neural network itself is a simple feed-forward structure. Each agent perceives the world through a set of photoreceptors spread across its field of view, and those values are passed through a hidden layer before producing two outputs: forward/backward acceleration and rotational acceleration. The simulation never tells agents how to chase food or avoid danger. It only defines what they can sense and how their outputs affect movement. The weights handle the rest.

Coevolution adds another layer to this. In a basic foraging simulation, fitness is measured against a fixed environment and improvement is straightforward. When predators are introduced, part of the environment becomes adaptive. Prey that get better at surviving raise the bar for predators, and predators that get better at hunting raise the bar for prey. Neither population is improving toward a static goal, which makes the system more dynamic and less predictable than single-population evolution.

3. Method and System Design

My design changes focused on two things: adding more meaningful selection pressures to the world, and making those pressures easier to observe and adjust without touching source code.

3.1 Simulation World and Generation Cycle

The simulation world contains three entity types: prey, predators, and food. Prey try to find food and avoid being caught. Predators try to catch prey. Food keeps the prey population under pressure to navigate effectively, while predators add a second, more dynamic source of pressure on top of that.

Each generation runs for a fixed number of steps. During each step, agents sense their surroundings, run their neural networks, and update their movement. The

world then processes collisions, prey consuming food, and predators catching prey. When the steps run out, the simulation scores both populations and uses those fitness values to produce the next generation. Complex behavior emerges from this cycle because each generation inherits a biased sample of whatever worked best under the current conditions.

3.2 Agent Perception and Runtime Configuration

Agents have no symbolic knowledge of the world. Each one perceives its surroundings through a set of photoreceptors spread across its field of view, and those values are passed through the neural network to produce movement. The simulation never tells an agent what to do. It only defines what the agent can see and how its outputs translate to acceleration and turning.

All of the important perception and movement parameters are configurable at runtime. From inside the application, users can adjust photoreceptor count, field of view angle, hidden layer size, population sizes, food count, and speed multipliers for both prey and predators. This removes the need to edit code and recompile between experiments, which makes it much faster to test how specific changes affect the evolving populations. The interface also displays live statistics including generation number, prey fitness, predator fitness, and death counts, so the user can monitor measured performance alongside the visual simulation in real time.

3.3 Batch Analysis

The interactive interface is useful for observation, but a single run can be misleading since evolutionary simulations are stochastic. The same configuration can produce different outcomes depending on random initialization and mutation. To account for this, the project includes a batch runner that executes many independent simulations and exports the results to a CSV file. A separate plotting script then generates four summary views: mean fitness over time with confidence intervals, percentile bands for prey fitness, per-generation improvement, and a prey-predator tradeoff phase plot. Together these give a clearer picture of how the system typically behaves than any single run could show on its own.

4. Results

The batch results provided a clear picture of the system's behavior over time. The main experiment used 500 independent runs, each executed for 300 generations. Under default parameters, 122 of those 500 runs (24.40%) ended in complete prey die-out. That result alone shows the system is not converging toward one stable outcome. Instead it has a meaningful failure mode where predator pressure becomes strong enough that prey cannot recover, which highlights how coevolution can produce both successful adaptation and collapse depending on how competing pressures unfold.

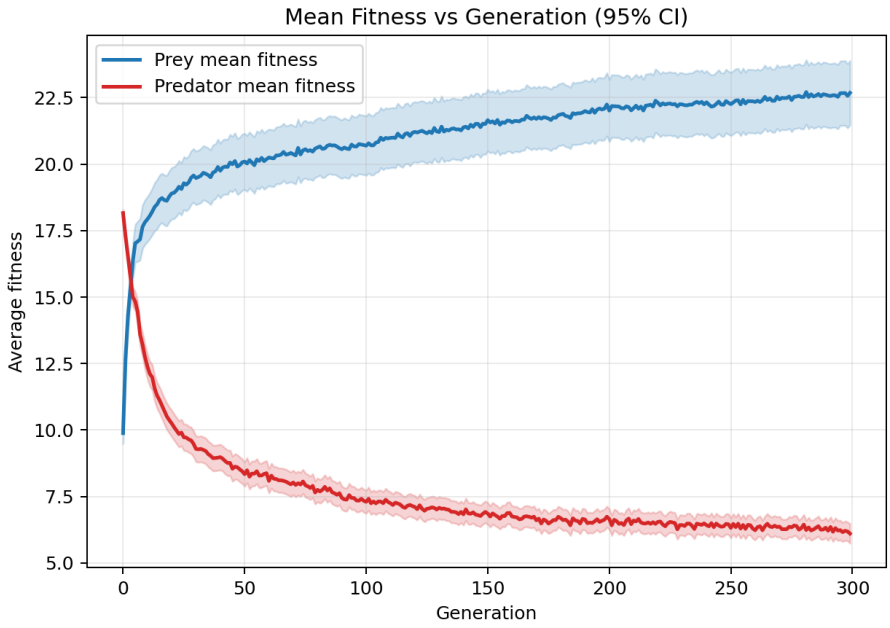

Figure 1: Mean Fitness

Figure 1 shows mean fitness trends over time. Predator mean fitness starts above prey mean fitness, suggesting an early advantage for predators. That relationship reverses fairly quickly, and prey fitness overtakes predators in the early part of training before continuing to climb slowly while predator fitness gradually declines. This is a good example of how coevolution does not guarantee balanced long-term gains. In this parameter setting, prey appear to discover survival and food-collection strategies faster than predators improve their hunting.

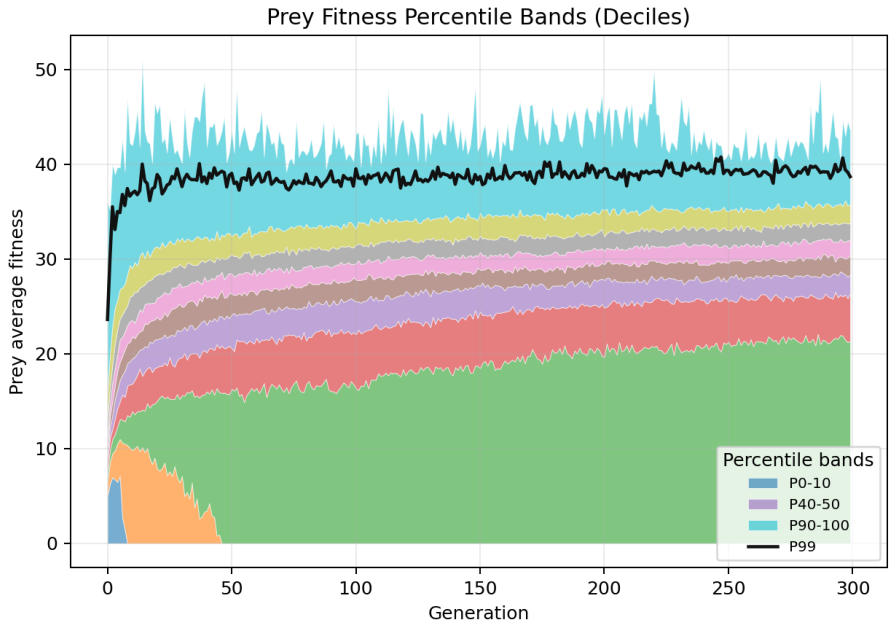

Figure 2: Prey Fitness Percentile Bands

Figure 2 shows prey fitness percentile bands, making clear that improvement is not uniform across runs. The lowest two deciles drop out quickly while the 99th percentile reaches an average of roughly 38 to 40 foods per generation, showing that some runs produce very successful prey populations even though many others do not. One of the more interesting details is that the run with the highest recorded average prey fitness still died out completely about twenty generations later, suggesting that strong performance at one point in training does not guarantee long-term stability.

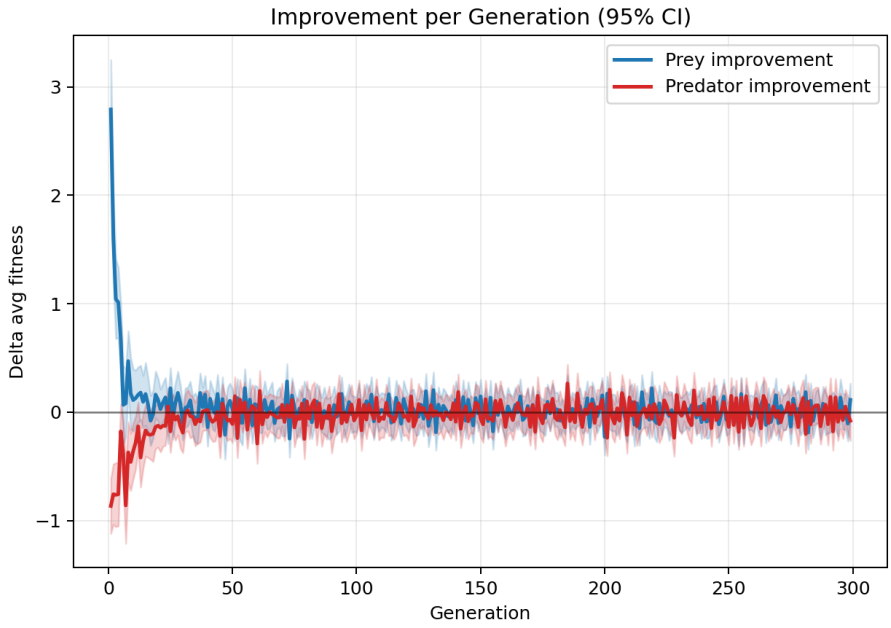

Figure 3: Improvement per Generation

Figure 3 shows improvement per generation and reinforces the idea of diminishing returns. Prey show strong early improvement that quickly falls close to zero, after which both populations oscillate around zero rather than continuing upward. Predator improvement begins negative, then also stabilizes near zero. Most of the large behavioral changes happen early, and the rest of training is spent making smaller adjustments in response to each other.

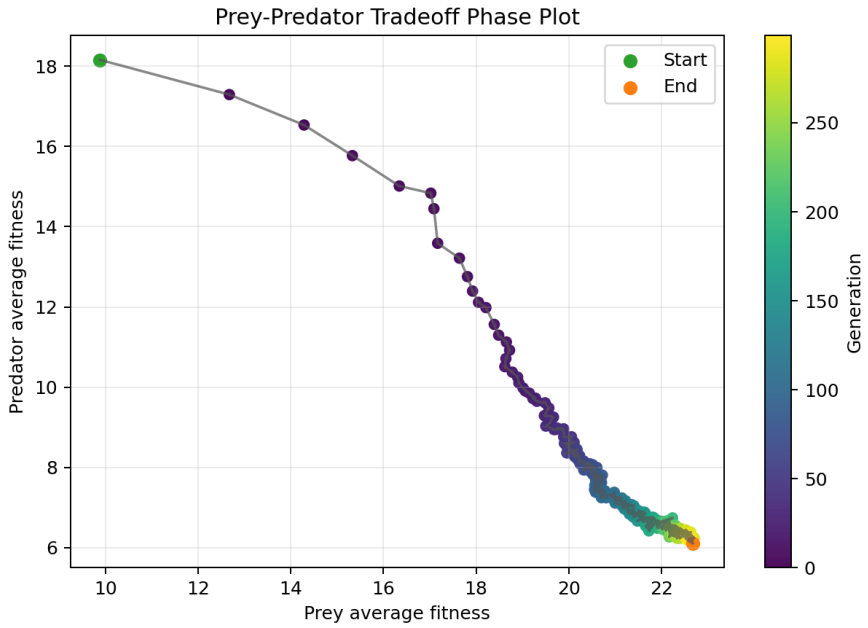

Figure 4: Prey-Predator Tradeoff

Figure 4 is the prey-predator tradeoff phase plot, which captures why coevolution was such an important addition. Rather than a simple story of one population improving, it shows how prey and predator fitness move together over training, with gains in one population reshaping the landscape for the other. The tradeoffs are most visible in early generations, but the plot still shows noticeable movement after 300 generations, meaning the system has not fully stabilized by the end of the experiment.

5. Discussion

The most important outcome of this project is that the additions changed the simulation from a fixed demonstration into a useful environment for experimentation. Predator-prey coevolution made the environment more dynamic and produced more complex behavior than a single-population system would. The runtime controls and live statistics made that complexity easier to observe without interrupting the workflow to edit code and rebuild.

The batch results suggest the system behaves less like a straightforward optimizer and more like a shifting competitive process. Prey improvement is strong early but does not continue upward smoothly. The fact that the highest-performing prey population later died out entirely is especially notable because it shows that success in one generation does not guarantee long-term survival.

In a coevolutionary setting a strong strategy may only be temporarily strong, and as the opposing population changes, that same strategy can become a liability. That instability is not a flaw in the simulation; it is part of what makes it worth studying.

The usability improvements are also not separate from the scientific value of the project. Changing parameters inside the application made it easier to compare different conditions and observe how populations responded. Live statistics reduced the gap between visual intuition and measured performance, and the batch tools helped determine whether a pattern from one run was actually representative. The interface work and the analysis work support the same goal: making the simulation easier to learn from.

The results also show that this is still a simplified model. Agents perceive the world through a limited sensor system, move in an abstract environment, and are scored with narrow fitness measures. Even so, the system produces meaningful tradeoffs, instability, and long-term behavioral shifts. That suggests the project succeeded in creating a sandbox where small design changes lead to visibly and measurably different evolutionary outcomes.

6. Limitations and Future Work

Shorelark is a simplified model, and some of that simplification creates real constraints on what the system can demonstrate. The sensor system is coarse, the environment is abstract, and the fitness measures are narrow. Prey are only rewarded for collecting food and surviving, and predators are only rewarded for catching prey. This means the agents cannot evolve more nuanced strategies like group behavior, territory, or deception, since nothing in the fitness function selects for those things.

The coevolution system also has a balance problem. Under default parameters, nearly a quarter of runs end in complete prey die-out, which means a large portion of batch runs do not produce stable long-term coevolution at all. Different default parameters or an adaptive difficulty system that adjusts predator pressure based on prey population health could make the system more robust and produce more consistent results across runs.

A few directions stand out for future work. The most interesting would be spatial structure, giving the world distinct regions or resources so that agents could evolve location-based strategies. Another useful addition would be to add further competition between predators. As of now, they cannot see and do not interact with each other, which leaves out a meaningful dimension of real-world competitive dynamics.

7. Conclusion

This project set out to extend an existing evolutionary simulation into a system that was more interactive, more configurable, and more useful for analy-

sis. Rather than replacing Patryk Wychowaniec's original foundation, Shorelark built on it by adding predator-prey coevolution, live statistics, an improved interface, and runtime parameter controls. Those additions changed the project from a food-seeking demonstration into a broader experimental platform where competing populations can be observed under a range of different conditions.

The results show that these changes were worthwhile. The interactive application makes it easy to test new configurations and immediately observe their effects, while the batch pipeline makes it possible to identify trends across many independent runs. The batch figures show that the system produces meaningful coevolutionary behavior: prey and predator performance shift in response to each other, early gains tend to be larger than later ones, and even highly successful populations can collapse under changing competitive pressure. These patterns would have been much harder to recognize without the runtime controls, live statistics, and batch tools added in this project.

Shorelark demonstrates that simple evolutionary components can produce rich behavior when placed in a configurable and competitive setting. More importantly, it shows that the value of a simulation like this comes not only from the agents it evolves, but from the tools built around it to make that evolution visible, measurable, and easy to explore.

References

- Patryk Wychowaniec. "Learning to Fly" series. https://pwy.io/posts/learning-to-fly-pt1/
- Isaac Salzman. Shorelark Expanded. GitHub, 2026. https://github.com/Czensored/shorelark-expanded