# Predator–Prey Simulation with Swarm Intelligence

Akalya Sridharan
Washington State University
akalya.sridharan@wsu.edu

Hannah Garcia
Washington State University
hannah.k.garcia@wsu.edu

Trisha Teredesai
Washington State University
trisha.teredesai@wsu.edu

Khushi Panchal
Washington State University
khushi.panchal@wsu.edu

December 11, 2025

**Abstract**

Swarm intelligence, which models collective behavior through simple agent interactions, inspires algorithms capable of solving complex problems. In this project, we implemented Neuro Evolution of Augmenting Topologies (NEAT) to evolve neural network controllers for predator and prey agents in a simulated 2D ecosystem. Predators gradually learn efficient hunting strategies, while prey develop adaptive evasion behaviors, resulting in emergent and realistic dynamics. The NEAT concepts of steady network development, adaptability, and topology crossover enables agents to optimize their survival strategies independently across generations. Improvements in capture rates, survival durations, and strategy complexity were seen by both quantitative and qualitative evaluations, supporting the coevolutionary approach's efficacy. This approach provides a foundation for future research to represent more resilient and adaptable ecological behaviors, such as multi-agent cooperation, fitness shaping, and environment generalization.

## 1 Introduction

In an ecosystem, organisms coexist in the same environment, sharing resources and space. Ecological relationships are the interactions between organisms within an ecosystem. One of these interactions is Predation. Predation is any interaction in which one species benefits by obtaining resources from and to the detriment of the other [1]. This interaction most commonly occurs in the food chain, one of the most important factors affecting the natural world. For survival, organisms act as predator and prey, often taking on both roles. This project aims to analyze the interactions between predator and prey in an ecosystem and identify patterns with prey survival and predator efficiency, to see how organisms adapt to survive by learning from the environment. This is done by simulating an ecosystem environment with a swarm-like algorithm behavior using NeuroEvolution of Augmenting Topologies (NEAT) to learn the expected behaviors and evolve the neural controllers for both predator and prey agents. This approach combines principles of swarm intelligence with evolutionary machine learning, which allows agents to learn evasion strategies through the environment they are in.

This project topic was chosen because Swarm algorithms are a novel concept which are outside of the scope of this course, and we wanted to expand our horizons by researching a machine learning algorithm that we were not familiar with. Our goal with this project is to explore Swarm algorithms and apply it to an ecosystem-like environment consisting of predator and prey to discover how agents ensure their survival. This project was challenging to undertake due to the complexity of the algorithm, since our team had to familiarize ourselves with the concepts of Swarm and NEAT to implement the model and analyze our results. Additionally, we researched scholarly articles and the methods used to outline a simplified version of the algorithm that we could implement within the span of this semester. In the context of our project, understanding and analyzing predator and prey relationships is important in understanding ecological interactions and the heterogeneous dynamics of ecosystems. These relationships contribute to understanding key ecological areas such as food webs, ecosystem functioning and stability, and community organization. By improving our understanding of such dynamics, we can better predict species interactions that may otherwise be difficult to observe directly and develop more effective methods for analyzing and modeling ecosystems [2].

## 2 Background and Motivation

Swarm algorithms mimic the collective behavior of natural organisms. Swarm agents interact and make simple decisions, which are used to solve complex problems [3]. Swarm intelligence is defined as the "collective behavior of decentralized, self-organized systems, natural or artificial" [4]. Swarm intelligence algorithms make use of collective, group-like behavior that interact with each other and their surrounding environment, exploring complex behaviors that can only emerge through such interactive actions. With the evolution of swarm intelligence algorithms, starting with early computational models in the 1940s, to the introduction of swarm robotics in the 1990s, swarm intelligence has continuously evolved throughout the years to adapt to recent technological advances. Most notably, the optimization techniques based on social behavior of bird flocks and fish schools, plus foraging behaviors of ant colonies and artificial bee colonies, developed from the 1990s and 2000s, provide approaches to real-world problems that traditional algorithms often struggle with [5].

Real-world applications include optimization problems (network routing, load balancing, function optimization, parameter tuning, and scheduling and resource allocation), collective decision-making (voting mechanisms, consensus formation for multi-agent systems, collective transport and object manipulation tasks, and task allocation algorithms in robotic swarms), and distributed sensing and exploration (environmental monitoring and data collection, search and rescue operations, planetary exploration missions, and mapping and localization in unknown environments) [5]. Today, swarm intelligence-based techniques are being explored by the U.S. military to control unmanned vehicles, by NASA for planetary mapping, by the medical field for locating and killing cancer tumors through stochastic diffusion search and nanobots, among many more applications [4].

When simulating the predator agent and the prey agent in our project, we observed how they would naturally behave, and discerned how it benefits their survival. The simulated environment follows certain parameters and interaction rules which the agents would abide by. The collective behavior of the agents emerges by evolving predator and prey controllers using NEAT principles, which get more efficient at survival-driven actions, increasing their fitness. Over generations, NEAT allows the agents to naturally optimize their decision-making strategies through feedback received from the environment they are placed in, with prey learning survival techniques and predators learning efficient hunting techniques [6].

### 2.1 NEAT Ideology

NEAT, or the NeuroEvolution of Augmenting Topologies, has increased efficiency compared to other fixed topology and neuro evolution methods. Neuroevolution is defined as the evolving artificial neural networks using genetic algorithms. It's highly effective in reinforcement learning tasks, like our project. NEAT is based on the concept of neuroevolution, or the method of evolving artificial neural networks with genetic algorithms, and its effectiveness in reinforcement tasks. NeuroEvolution of Augmenting Topologies (NEAT) specifically is a type of Topology and Weight Evolving Artificial Neural Network (TWEANN) that significantly outperforms any other fixed-topology neuroevolution methods. NEAT addresses three major challenges that other TWEANNs often face, and presents solutions to each of them. First, NEAT uses a "genetic representation that allows disparate topologies to crossover in a meaningful way." NEAT does this by using historical markings to line up genes with the same origin. This way, algorithms can observe two genes that have the same ancestral gene, although with possibly different weights. Thus, the system needs only to know which genes line up with which in order to keep track of the historical origin of every gene in the system [7].

Secondly, NEAT protects innovation through speciation. By separating each innovation into a different species, individuals compete primarily within their own niches instead of with the population at large. Topological innovations are then protected and have the time to optimize their structure before they have to compete with other niches in the population. Lastly, NEAT minimizes dimensionality by starting from a minimal structure and growing only when necessary. TWEANN algorithms usually start with an initial population of random topologies , due to their inability to protect innovation. New structures frequently do not survive in these methods; thus, they are introduced during initialization. In contrast, NEAT begins with a uniform population of networks with no hidden nodes. In the case of our predator-prey simulation, the neural network controllers for the agent are evolved over generations, and in each episode, the genome is scored for fitness in the defined environment. Once each genome is scored, NEAT performs selection, crossover, mutation and speciation to produce the next generation. High fitness is rewarded. Since NEAT protects innovation, the system is able to support new structures incrementally as structural mutations occur [7].

## 3 Machine Learning Task Definition

### 3.1 Task Definition

This project models predator and prey dynamics through an evolutionary machine learning system inspired by the methods detailed in "A Framework for Learning Predator-Prey Agents from Simulation to Real World" [8]. The multi-agent reinforcement learning task is defined by autonomous agents interacting in a simulated two-dimensional environment. The two primary agent roles include a predator agent, whose goal is to capture the prey, and a prey agent, whose objective is to successfully flee from the predator. The environment additionally contains obstacles to restrict movement and encourage strategic evolutions through survival mechanisms, such as obstacle avoidance and hiding behavior. The core problem is to learn policies for both agents that maximize each of their respective goals through repeated interactions within the learning setting. Agents must learn to adapt their movement and decision-making strategies over time to effectively respond to opposing agents' behavior and actions. Additionally, the task is episodic, ending when the prey is captured or when the maximum number of steps is reached.

Inputs for the agent controllers include its (x, y) position, movement speed, and radius for collision detection. Additionally, the agents don't see the whole board. Instead, each agent receives a small observation vector towards the relative direction of its opposing agent. This method mimics real swarms and their local perception. Outputs of the agent controllers include the direction of movement through direction vectors, multiplied by each agent's movement speed, in order to update the agent's position.

### 3.2 Data and External Resources

The external resources used consist mainly of Python libraries, including NumPy (2.3.5) for vector math, Matplotlib (3.10.7) for visualizations, and the NEAT-Python library (1.0.0) for evolutionary optimization.

The NEAT library uses genomes containing two sets of genes that describe how to build the artificial neural network: node genes, specifying a single neuron, and connection genes, specifying a single connection between the neurons. For evolution to occur, a fitness function for each agent's survival will be determined to compute a value indicating the quality of an individual genome. The best ability and methodology to solve a problem and reach a goal increases the fitness score.

The evolutionary process is defined by the parameters and initial settings through the inputted NEAT configuration. The following parameters, directly referenced from NEAT-Python 1.1.0's documentation and Stanley and Miikkulainen's research, include evolution settings such as population size, allowed resets on extinction, enabled fitness termination, and a fixed seed to ensure fixed randomization and reproducibility [7][9]. Neural network structure and mutation rules are additionally inputted to determine how each agent's brain evolves. Lastly, specification settings, species stagnation, and reproduction settings are configured according to developer input to control genomes and their species.

The training process does not use datasets or any pre-labeled training data. Instead, through reinforcement learning, agents learn through repeated interactions in the simulated environment to maximize rewards.

### 3.3 Research Questions

Our project investigates the following research questions: How can a two-dimensional environment be represented to effectively demonstrate agent interaction and learning behavior? How can we best model predator and prey agents in our implementation? How does evolutionary growth encourage strategic evolution strategies for the prey and predators? How do swarm-based interactions and mechanics affect any emergent behavior or survival outcomes?

## 4 Technical Approach

### 4.1 System Overview

In our system, we model a continuous two-agent predator-prey pursuit problem with static circular obstacles in a 2D environment with bounded coordinates. Both agents operate under partial observability: at each timestep the environment returns a compact sensory vector comprised of normalized direction vectors, a scaled Euclidean distance, a bias term, and obstacle-direction features that point toward the nearest obstacle. The compact features are processed by the controller, which produces a continuous two-dimensional velocity vector, clamped to the agents' respective maximum speed. The environment integrates the dynamics, accounts for the boundaries, obstacles, and computes a capture radius condition, also storing the episode trace of positions, velocities, and distances between the two agents. The system is customizable, allowing the use of interchangeable, callable controller strategies with the same interface for direct comparison of predefined strategies, randomly initialized neural networks, and NEAT-evolved strategies.

### 4.2 Baseline Prey Strategy

Before performing neuro-evolution, we provide a deterministic prey baseline that encodes several intuitive strategies for escaping the predator and helps stabilize the evaluation. The simplest baseline simply executes a minimalist greedy escape strategy, where the prey calculates the vector opposite the predator's position and moves that direction at a constant speed. This strategy is purely reactive and simply checks the integration. Furthermore, we provide a refined, manually designed strategy that linearly combines several other strategies: escaping the predator, avoiding obstacles (by moving away from the nearest obstacle), a hide-behind-obstacle vector provided by an environment helper, a lenient wall avoidance strategy near the edge of the arena, and a stochastic jitter that prevents it from being strictly deterministic.

### 4.3 Evolved Prey Strategy

We use genomes evolved using the NEAT algorithm. Each of these genomes is wrapped around a controller that maps the reduced observation space used by the baseline methods to continuous velocities. During the prey evolution phase, the learning goals are episodic, focusing on maximizing survival time and the distance attained at the end of the episode, trying to survive and avoid capture, with the implicit goal of using whatever advantage the obstacles and the arena geometry may provide. Finally, the best evolved prey is stored for use as an opponent for the predator.

### 4.4 Predator Model

The predator controller can be evaluated both as a hand-crafted baseline and as an evolved policy. The baseline predator controller is a greedy/avoidance controller that moves towards the prey, with a mild, negative-weighted obstacle avoidance term added, and the vector is normalized and scaled to the predator's top speed. For the learned predators, the method uses NEAT again, learning controllers that take observations as input and output continuous velocities. For the predators, the fitness objective is formulated to meet several goals at once: successively shortening the distance between the predator and the prey, avoiding behaviors where the agent gets stuck near walls, and, most importantly, successful capture early on. Naively, a distance-based fitness measure alone can suffer from degenerate solutions (e.g., freezing, which occurs when stationary points result in zero penalties for progress). To rectify this, the proposed model uses a multi-term fitness objective that optimizes progress towards the prey, regression, and rewards prey capture episodes, where the weights were adjusted for emerging interception/cutting off strategies instead of mere direct approaches.

### 4.5 Swarm Mechanics

Although our experiments focus on single-predator, single-prey interactions for clarity, the simulator and controller interface are intentionally general and readily extensible to multi-agent swarms. The same compact observation format can be used with neighbor-relative positions and velocities (e.g., k-nearest neighbors) or a local occupancy map for coordination. For multi-agent systems, sharing parameters among symmetric agents (reducing the dimensionality of the search space) and the use of centralized training and decentralized control (CTDE) or evolutionary approaches with precise credit assignment will be valuable. By abstraction, the hardcoded primitives used within the baseline approaches could be combined to support group behaviors like flanking attacks, and these will also emerge when using NEAT for the multi-predator cases.

### 4.6 Training Protocol and Evolution Loop

We use a staged evolutionary protocol, where robust opponents are evolved while maintaining tractability in our runs. Firstly, the prey evolved would develop a strong evasive strategy, and then the best evolved strategy would be stored as the evolved prey. Secondly, a predator population would be evolved for this evolved prey. Here, for a large number of randomized episodes of interactions, the score of every predator genome would be used as its fitness function. NEAT would be used for selection, speciation, and structural mutation for the evolved population. Moreover, evaluation of a population would be done for several instantiations of a given environment, which would minimize variability of characteristic values. The points of convergences, along with the best strategy, would also be stored. Further, optionally, this cycle would continue, where the strategy for prey would be evolved for this evolved predator strategy, expecting to develop an arms race. This, however, would require a structure that supports a two-phase co-evolution.

In practice, this is accomplished using the following: a fixed episode horizon, a parameterized capture radius, and evaluation average per genome. Random seeds and config JSONs are stored for reproducibility, and learned dynamics are analyzed using qualitative data, specifically the visualization of trajectories, in combination with other quantitative measures. These design choices, along with appropriate fitness shaping, are critical for avoiding pathologies and achieving high-quality pursuit and evasion strategies.

## 5 Evaluation Methodology

### 5.1 Simulation Environment

Predator and prey agents interact in a simulated environment that has well-defined parameters that control the world state. Each parameter is modifiable to update the configuration of the environment setup, but our implementation follows the environment definition as follows. Agent interaction controls are also defined by the environment, which governs the starting position of the agents as well as capture measures. Training occurs in a 2D bounded domain with dimensions 100 units x 100 units, with 3 randomly placed circular obstacles. The radii of the obstacles range from 4-12 units. Once the environment is set up with obstacles, the agents are placed within the boundary. Predator spawns in the free space, in a random position on the board, avoiding corners and obstacles. The predator agent's speed is set to 2 units per timestep. The prey spawns within 40 units of the prey with a speed of 1.1 units per timestep. The capture radius is also defined, when a predator gets within a 5-unit radius of the prey, it is considered that the prey has been captured. The simulation environment also keeps track of observations of each agent in a feature vector. These features pertain to the distance of one agent from the other, as well as the distance from the agent to its nearest obstacle. These parameters are used by NEAT to evaluate the genome. Along with the feature vector, the environment also returns the agents' actions, which includes the Euclidean distance between the predator and prey, whether or not the prey has been captured and the position coordinates of each agent on the board.

### 5.2 Metrics

Once the environment is defined, we can begin training the controllers. For each genome, a controller is created which feeds into the NEAT network. Each genome is evaluated by running multiple episodes, in our implementation we run 5 episodes, each with a time period of 500 units. In each episode, we start by initializing the environment with randomly placed obstacles and agents to increase robustness. At each timestep, the actions for each agent are decided by the controllers using the current observations (the feature vector defined in the environment) and the environment is updated accordingly. A trace of each timestep is also recorded. If the prey is captured, the episode ends early, otherwise it will run for the whole period. Each episode is summarized by the number of steps taken, if the prey was captured or not and the final distance between the two agents. With the trace, each episode is given a fitness score and the overall genomes fitness is determined by calculating by the mean of the episode fitnesses.

Due to the nature of the desired behaviors for each agent, the fitness for the predator and prey are calculated with unique factors. For both agents, they are incentivized to move by providing a movement bonus, which prevents them from freezing. For the predator, there is a strong incentive to capture the prey quickly, which is represented by a large positive reward for capture as well as a reward for closing the distance between itself and the prey. Since the predator's objective is to chase the prey, it is rewarded for aligning its movement towards the prey and concurrently penalized for "running away" or increasing the distance between itself and the prey and remaining close to the world borders. Conversely, the prey is awarded for extending its survival time, meaning the longer the episode runs, higher the reward. The prey also receives a bonus for increasing the distance between itself and the predator. Lastly, the prey is heavily penalized for getting captured. The final fitness for the episode is the weighted sum of these factors. NEAT uses the fitness score to refine the agents for further generations. High-fitness genomes are selected to produce the next generation of controllers.

## 5.3 Hyperparameters

The hyperparameters of the NEAT algorithm used for predator-prey training were mainly based on the default configuration from the documentation, but certain parameters were modified to better fit our implementation and provide us with the desired results [9]. First, the fitness threshold was increased to ensure the evolution does not end early and to make sure the controllers are appropriately trained. A default genome was defined, which controls how each agent's neural network evolves and mutates. Bias and weight parameters are also defined to control how NEAT modifies them over generations. The structural mutation, which is how NEAT evolves the architecture of the neural network over generations, is configured to increase topological growth and increase topological cleanup, causing the predator and prey to improve faster and develop robust behavioral strategies.

The default reproduction parameters determine how new genomes (the offspring of the current generation) are created. Our implementation has a higher elitism threshold, which specifies how many of the high-fitness genomes remain unchanged in the next generation. Preserving the top two genomes makes evolution stable by reducing best-fit characteristics through mutation. The survival threshold is also increased, which allows more of the current generation to reproduce. This allows for more diverse and niche behaviors to emerge. The minimum species size is also set to make sure a population is not wiped out. In unison, these factors guarantee that evolution is stable with agents preserving beneficial traits across generations and building strong behavioral architectures.

## 5.4 NEAT Evolution Method

With each generation, NEAT uses selection, speciation, crossover, and mutation to generate the genomes for the next generation. Selection is handled by picking out the genomes with the best fitness for reproduction, as well as using elitism to preserve a certain number of genomes. Next, NEAT groups genomes into species according to compatibility. Doing this prevents younger genomes from being outperformed by older, stronger genomes since they would be classified into separate species. When creating the new generation, apart from the genomes unchanged by elitism, genomes are formed by crossing over traits from parents from previous generations. Mutations are controlled by weight, bias, as well as structural mutation parameters. Over numerous generations, NEAT employs these principles to evolve the controllers to be more efficient and obtain higher average fitness.

## 6 Results and Discussion

### 6.1 Quantitative Results

Over 30 generations of evolution, we observed a clear improvement in both predator efficiency and prey survival strategies. Early generations showed highly unstable results characterized by erratic movement, minimal learning, and low capture rates. As evolution progressed into mid to late generations, predators demonstrated consistent performance gain, such as smoother trajectories, faster distance closure, and higher capture rates, while prey developed more sophisticated evasion strategies as the NEAT algorithm refined network topologies and weights. The primary metrics were capture rate, mean time-to-capture, final distance and fitness distribution. The capture rate is calculated as the proportion of episodes in which the predator successfully captured the prey. The mean time-to-capture is the average number of timesteps required for a successful capture of the prey by the predator. In the episodes where the prey was not captured, the final distance is the distance between predator and prey at completion. The fitness distribution is the average and maximum evolutionary fitness per generation.

Across evaluation runs with randomized initial conditions, the trained predator achieved a high capture frequency, often securing captures in under 20-40 steps when the prey's escape trajectory intersected with environmental bottlenecks. In cases where the predator was not captured, the predator still demonstrated improved pursuit behavior, lowering final distances compared to early-generation agents. During training, fitness values showed a clear upward trend, with the best genome increasing from early-generation scores around 4,000-6,000 to final best scores exceeding 14,000. Population fitness variance decreased as the predator converged toward consistent, high-quality survival strategies. This trend indicates that the fitness function successfully shaped meaningful improvements in agent behavior, showing that NEAT's evolution methods resulted in insightful improvements in efficiency and performance over successive generations.

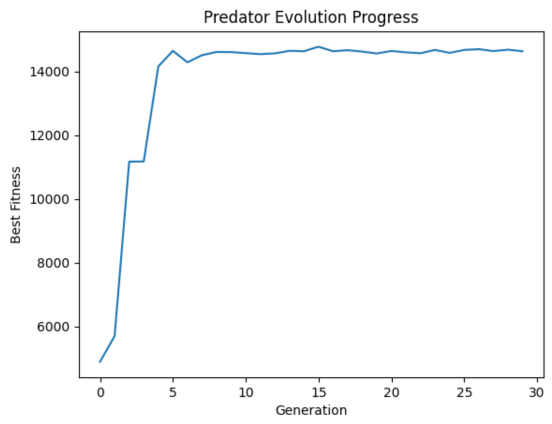

Figure 1: The predator's fitness curve increases rapidly over early generations before converging to a stable high-performance policy.

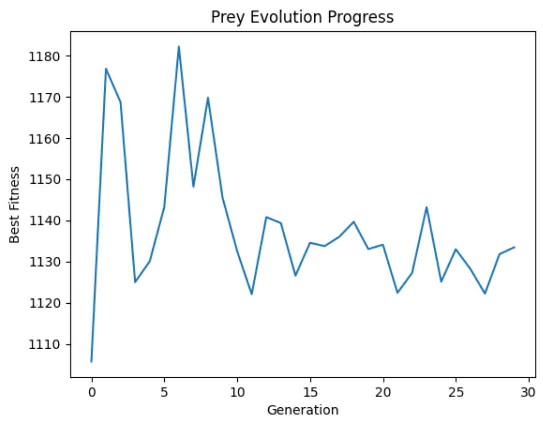

Figure 2: Prey fitness over generations, fluctuating as the population adapts to an improving predator.

## 6.2 Qualitative Results

To evaluate our agent's behavior qualitatively, we visualized their trajectories from several predator-prey chase episodes using custom animation and path-plotting tools. These visualizations allowed us to observe how each agent navigated throughout the environment, interacted with various obstacles, and responded to the opposing agent's behaviors and patterns. During successful episodes, the learned predator demonstrated pursuit behavior, where it adjusted its heading towards the prey, cut angles to decrease distance, and maintained momentum without freezing in place. NEAT's evolutionary process allowed these behaviors to emerge organically.

Additionally, the prey dummy controller provided a strong baseline to assess the predator's skills and abilities. In several runs, the prey could initiate escape trajectories by combining flee vectors, obstacle avoidance, and wall-shaping behavior. However, the predator consistently demonstrated the ability to close the distance when the prey's avoidance path intersected with any environmental constraints. Visualizations assisted with highlighting these scenarios, showing how the predator successfully exploited the prey's turn. In other cases, the prey escaped by using its surrounding obstacles or reaching the periphery of the world, showing both the strengths and limitations of this learned policy.

Overall, such qualitative inspection shows that the evolved predator learned meaningful strategies consistent with pursuit behavior rather than random motion. These qualitative visualizations serve as evidence that the training pipeline generated emergent behaviors.

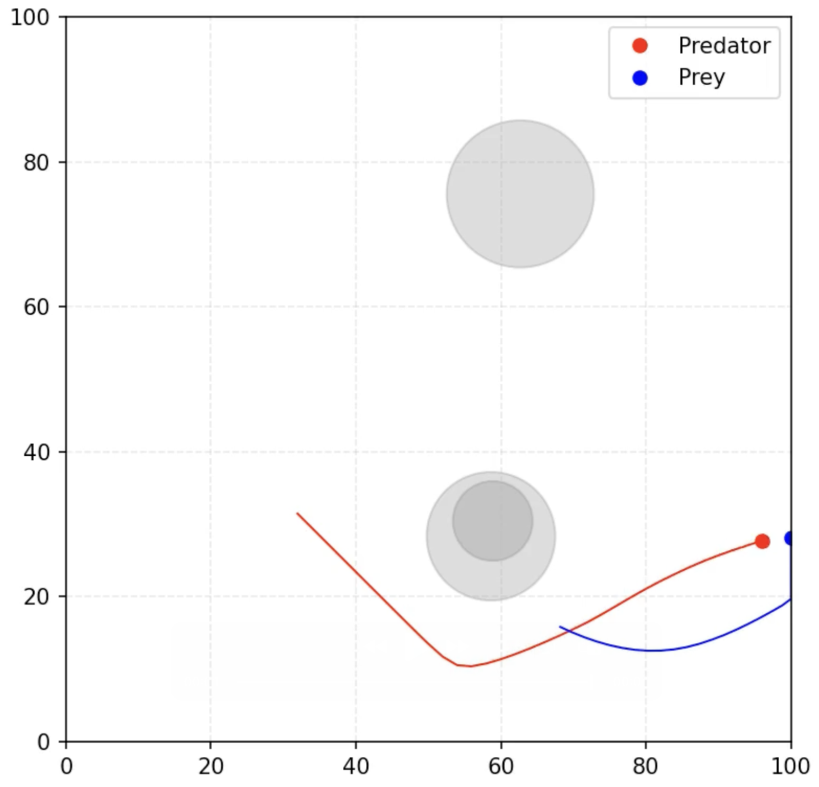

Figure 3: Environment simulation with predator chasing prey

### 6.3 Key Technical Challenges

Several technical challenges arose over the course of our project's implementation, significantly affecting the system's mechanics. One major challenge occurred when we attempted to train the predator against the previously evolved prey. Since the prey had been trained much longer, it made it practically impossible for the predator to capture. In turn, each predator genome gained uniformly low fitness, causing NEAT to have no gradient to reference and improve upon. In order to restore the desired learning dynamics, we replaced the advanced prey with a dummy controller. The dummy controller behaved intelligently, but not to the point of shutting down predator evolution. Implementing this change allowed the predator to correctly acquire consistent chasing behavior.

We additionally discovered that the environment's elements interacted poorly with early versions of the fitness function. Random spawn locations, obstacle arrangements, and world sizes would cause one episode evaluation to produce extremely noisy fitness values, and genomes could receive unrealistically high scores purely because of a lucky configuration. Additionally, collision logic originally allowed agents to partially enter obstacle boundaries. This logic, combined with the prey's obstacle avoidance and hiding vectors, resulted in the prey occasionally being pushed towards the predator instead of away from the predator. We addressed these issues by adding guards, type checks, and fallback behavior. Additionally, increasing to thirty randomized episodes per genome instead of one, and adjusting the world initialization settings, assisted with stabilizing training.

## 7 Limitations and Future Work

The first major limitation is the project's use of evolution with a single predator. Adding two or three predators would allow for the discovery of new behaviors such as flanking, interception, or even herding the prey towards an obstacle. The possibility of four or more predators could also be explored.

Other future work includes fitness shaping. Most of the training stability in the project's current system comes from its reward structure. However, adding explicit bonuses for using obstacles strategically, such as hiding behind one before attacking, or penalties for entering unsafe regions, can nudge evolution toward higher-level behaviors rather than blindly chasing prey.

Finally, a major step forward would be generalization. Currently, predators are evolving in a fixed-size environment with a limited number of obstacles. Future versions of the code implementation could assist with randomizing the map layout and the obstacle shapes. This would train predators that are robust to new environments instead of overfitting in a single world. Overall, these future steps push our system closer to real swarm intelligence, where cooperation, strategy, and adaptation emerge together.

## 8 Conclusion

Our study was successful in implementing a predator-prey simulation that used the NEAT method to develop neural network controllers for predator and prey agents. As prey evolved adaptive evasive tactics and predators developed increasingly effective tailing strategies over generations, complex emergent dynamics reflecting genuine predator-prey interactions emerged. Agents were able to learn and optimize survival strategies independently thanks to NEAT's coevolution, adaptability, and gradual network extension, demonstrating the algorithm's capacity to generate complex, non-programmed behaviors.

The results show how neuroevolutionary techniques may be used to replicate adaptive behaviors in ecological systems, even though our implementation was restricted to a single predator and a set habitat. Future improvements like fitness shaping, multi-agent collaboration, and generalized environments might increase simulation robustness and realism. Overall, the project emphasizes NEAT's effectiveness in nurturing emergent intelligence and establishing an adequate platform to evaluate more complicated swarm-based interactions and adaptive strategies in simulated ecosystems.

## 9 References

[1] A. Ryczkowski, “Five Types Of Ecological Relationships,” *Sciencing*, Mar. 24, 2022. `https://www.sciencing.com/five-types-ecological-relationships-7786/`

[2] E. Mizsei et al., “A trait-based framework for understanding predator–prey relationships: Trait matching between a specialist snake and its insect prey,” *Functional Ecology*, vol. 33, no. 12, pp. 2354–2368, Sep. 2019, doi: `https://doi.org/10.1111/1365-2435.13446.`

[3] “Can swarm intelligence integrate with AI and machine learning?,” *Milvus.io*, 2025. `https://milvus.io/ai-quick-reference/can-swarm-intelligence-integrate-with-ai-and-machine-learning` (accessed Oct. 08, 2025).

[4] Wikipedia Contributors, “Swarm intelligence,” *Wikipedia*, Nov. 24, 2019. `https://en.wikipedia.org/wiki/Swarm_intelligence` (accessed Oct. 09, 2025)

[5] “Historical development of swarm intelligence — Swarm Intelligence and Robotics Class Notes — Fiveable,” *Fiveable*, 2025. `https://fiveable.me/swarm-intelligence-and-robotics/unit-1/historical-development-swarm-intelligence/study-guide/LNOm1YBPH6z9gbVM(accessedOct.09,2025)`

[6] A. McKee, “Swarm Intelligence Algorithms: Three Python Implementations,” *Datacamp.com*, Oct. 10, 2024. `https://www.datacamp.com/tutorial/swarm-intelligence`

[7] K. O. Stanley and R. Miikkulainen, “Evolving Neural Networks through Augmenting Topologies,” *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, Jun. 2002, doi: `https://doi.org/10.1162/106365602320169811.`

[8] J. Chen and Z. Gao, “A Framework for Learning Predator-prey Agents from Simulation to Real World,” *arXiv (Cornell University)*, Oct. 2020, doi: `https://doi.org/10.48550/arxiv.2010.15792.`

[9] “NEAT-Python 1.1.0 Documentation,” *Readthedocs.io*, Dec. 06, 2025. `https://neat-python.readthedocs.io/en/latest/` (accessed Dec. 9, 2025).

## 10 Appendix

Link to GitHub repository: `https://github.com/akalyasri/Predator-Prey-Simulation-with-Swarm`