Optimizing Control Parameters of a Predator-Prey Simulation with an Evolutionary Algorithm

Ryan Slater
College of Engineering and Applied Sciences
University of Cincinnati
Cincinnati, Ohio
slaterrj@ucmail.uc.edu

***Abstract*—Understanding the workings of nature has been a constant endeavor of humanity. The interactions between organisms that dictate how species survive is of particular interest. While both micro and macroscopic interactions between organisms and species have been studied, the relationship between them remains a mystery. This paper proposes a novel method of simulating interactions between a predator and prey species that hopes to provide insight as to how the traits of individuals might impact the survival of the species as a whole.**

I. Introduction

Countless studies have been performed to attempt to understand how a population of individuals interacts with itself and with populations of other species to create large-scale group behavior from simple individual decisions. Great lengths have been taken to not just observe animals in their natural habitats, but to create simulated habitats for them to be observed under controlled environments. Additionally, computer simulations of animals and their environments have been created to attempt to replicate their behavior. Both approaches are far from perfect and are difficult to implement in a meaningful way.

One ongoing study by the Max Planck Institute of Animal Behavior in Germany is attempting to understand the behavior behind locust swarms. [3] By understanding how local interactions between individual locusts lead to the formation of swarms in the billions or trillions, how those swarms move through crops, and how they avoid predators, predictive models can be made to protect farmlands from dangerous swarms. To observe locust behavior, three different methods are employed. One experiment involves putting a single locust on a large ball that compensates for the movement of the locust, keeping it on the top of the ball at all times. A 360° screen is wrapped around the locust's field of vision, and a swarm projected onto that screen. The locust believes this projection to be a real swarm, which gives the researchers the ability to create their own virtual swarms and observe how a real locust reacts. A similar experiment involves attaching a neural probe to the locust to observe the brain signals produced by the locust while adapting to changes in virtual swarm behavior. The third experiment involves placing over 10,000 locusts in a 15 meter by 15 meter container, attaching retro-reflective tags to each of them, and using a high-precision motion capture system to track the location of each of the locusts at a framerate around 100 times per second. This allows for the study of a real swarm, albeit significantly smaller than those appear in nature.

Locust swarms affect around ten percent of the world's population by causing food shortages. Current methods of prevention are mostly brute-force: spray pesticide and hope. Studies like these provide invaluable insight that can lead to the protection of the world's crops. However, there are two main issues with performing studies like these. Firstly, there are ethical questions when it comes to experimenting on animals. Typically, experimenting on insects is acceptable [3], but debated when it comes to other animals. Collecting thousands of rats to analyze how they survive in a city's sewer system, for example, may be deemed unethical. Secondly, the process of collecting thousands of individuals and setting up the test environment is not only costly, but time-consuming. If computer simulations are able to be developed to represent these experiments, not only will time be saved, but experiments could be dynamically scaled to whatever size the researchers desire, hardware permitting.

The challenge is, however, actually creating these simulations. Most modern population simulations are simple and naive. [1] [2] [5]. However, simple representations of individuals reduce the computational load required to update a single individual in a simulation, which allows for larger simulations to be run. Thus, the objective when creating such simulations should be to create the simplest possible model of an individual that represents the animal being studied whilst maintaining a reasonably accurate portrayal of the real animal. Additionally, once the simulation is created, manual parameter tuning is required to create a stable environment for the individuals. This can be quite difficult and relies on developer knowledge and experience to choose suitable parameters [6]. This paper attempts to provide a solution for this problem, while keeping the objective of a simple representation of an individual in mind.

II. Background and Related Work

A. The Lotka-Volterra Model

In predator-prey simulations, two species are created: a predator species and a prey species. Individuals of the predator species must eat those of the prey species to survive and reproduce. Typically, there is some additional source of food specifically for prey, though sometimes, other methods are

employed. One of the most famous models for simulating the interaction between a predator and prey species is the Lotka-Volterra Model, designed by Alfred Lotka in 1910 and recreated by Vito Volterra in 1926. In this model, "the rate of increase of population is proportional to the size of population at any time." [1] The equation

$$\frac{dP}{dt} = KP \tag{1}$$

represents this statement, where k is some positive constant. By integrating, we can find the size of the population at an arbitrary time t

$$P(t) = P_0 e^{kt} \tag{2}$$

where P_0 is the initial size of the population. These equations are described by the Malthusian growth model, which "predicts an exponential growth in the population with time." [1] The Lotka-Volterra Model extends on this principle by introducing the notion of a predator species that impacts the size of the original population. To introduce the predator species to the Malthusian growth model, the following assumptions are made:

- There is no limit to the food source for the prey species.
- If there are no predators, the prey population x will increase as described by the Malthusian growth model, with the *coefficient of auto-increase* (as named by Volterra) of $\alpha > 0$.
- If there are no prey, the predator population y will decrease exponentially with coefficient $\gamma > 0$.
- If both x and y are nonzero (there are both predator and prey individuals present), the predator population will increase while the prey population decreases, both at a rate proportional to the multiple of the two populations (prey: $-\beta xy$, predators: δxy, where $\beta, \delta > 0$)

This leads to the creation of the following equations, modified from the original Malthusian equations to handle interactions between the predator and prey populations.

$$\frac{dx}{dt} = \alpha x - \beta xy \tag{3}$$

$$\frac{dy}{dt} = -\gamma y + \delta xy \tag{4}$$

Anisiu [1] used $\alpha = 2$, $\beta = 1.1$, $\gamma = 1$, and $\delta = 0.9$ to create the plot shown in Figure 1. This plot shows the cyclic and exponential nature of population growth and decay. As the prey population (blue) grows exponentially, the predator population (red) follows suit, delayed by some short time. After the predator population exceeds that of the prey, the prey consumption rate βxy outpaces the reproduction rate αx and the prey population begins to decline. Likewise, with a lack of food, the predator population follows suit.

This relationship is the primary principle defined by the Lotka-Volterra model. Other models for simulation predator and prey interactions tend to follow this principle, regardless of if they directly implement the Malthusian equations or not.

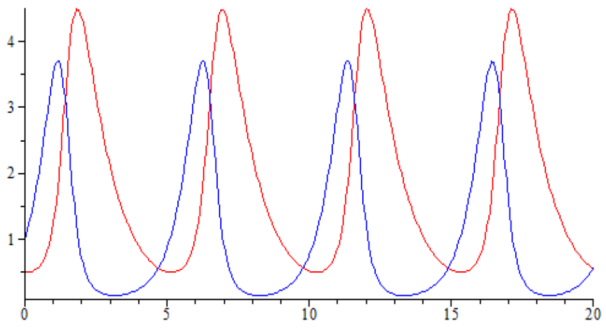

Figure 1. Blue - prey, red - predators

B. Emergent Patterns

The model proposed by this paper is inspired by two related works: one by Hawick et al. [2] and another by Tampon [5]. Hawick proposed an implicit implementation of the Lotka-Volterra Model in their 2006 paper: *A Zoology of Emergent Patterns in a Predator-Prey Simulation Model*. Their simulation implemented a population of foxes and rabbits on a flat plane, utilizing simple decision rules to dictate individual behavior. The objective of the paper was to investigate large-scale formations created by many individuals, as shown in Figure 2.

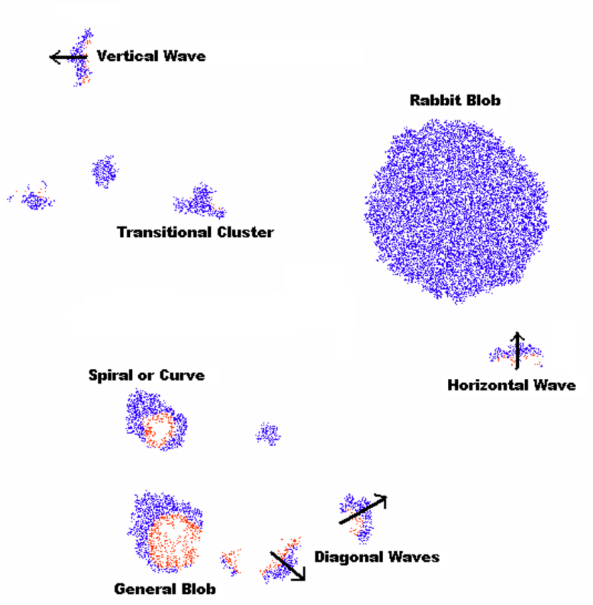

Figure 2. A set of the patterns created by a mass of individuals following simple rules. Red dots represent predators (foxes) and blue dots represent prey (rabbits)

Hawick does not describe details about how the individuals are implemented, how reproduction occurs, or if they even eat each other. However, the fact that simple decision rules such as "move towards food" and "move away from predators" can cause populations to exhibit behavior larger than any individual without any learning agent or inter-individual communication demonstrates that complex individuals with

learning capabilities are not required for basic predator-prey simulation. However, without complex decision rules or learning agents, the distinction between a predator-prey simulation and a particle simulation blurs.

C. Evolution of Individuals

Tampon took a very different approach with his simulation. The implementations of individuals in this simulation are more complex than that of Hawick's. In this simulation, each individual exists on a two dimensional plane and has direct control over its orientation on the plane and current speed. Both predators and prey have an "energy" property that depletes over time. If a predator's energy reaches 0, it dies. If a prey's energy reaches 0, it does not die, but is unable to move. For both, moving costs energy proportional to the speed of the movement. A predator can increase its energy level by eating prey, and a prey can increase its energy level by not moving. Obviously, a predator must move (and lower its energy level) to eat, and a prey risks being eaten while sitting still. The objective of each species is to reproduce. This is accomplished by the predators by eating enough (gaining enough gross energy to reach some threshold) and by the prey, by living for a set duration.

Each individual casts rays out to a fixed distance in front of them, and the distance of the first collision is returned for each ray. Akin to how predators in nature typically have forward-facing eyes and prey have eyes on the sides of their head, [3] rays are cast in a narrow cone for predators and a wide arc for prey. The maximum depth of each ray is longer for predators than it is for prey.

These properties imply the existence of some complex decision structure dictating the actions of each individual. As might be expected, Tampon utilized a small neural network for each individual. Though not explicitly stated, the networks appear to evolve in a similar manner as proposed by Stanley and Miikkulainen [4] in their paper *Evolving Neural Networks through Augmenting Topologies*. At each time step, each neural network takes in the list of collision distances and outputs a desired direction and speed. The network of each individual at the beginning of the population is not connected. Upon reproduction, the network of the parent is copied to the child and has a chance of mutating, either through weight modification, connection addition, or neuron addition.

Tampon expanded upon this simulation in [6]. In this video, he explained the difficulty in tuning parameters to create a stable simulation. However, once acceptable parameters were found, the Lotka-Volterra population growth dynamics as shown in Figure 1 can be observed. In [6], Tampon introduced the notion of group behavior. To implement this, each neural network was given an additional input for each ray cast, corresponding to the type of entity (predator or prey) that the ray hit. To encourage group behavior, predators were required to hit prey twice to eat them, and prey were given the ability to kill predators by hitting them four times. This change caused clusters of prey to form, defending each other from predators, which in turn caused predators to form lines to pressure prey with. The ability for simple individuals with no communication capabilities to form group tactics reaffirms the hypothesis that complex individuals are not required to simulate natural behavior of organisms. However, the issue of parameter tuning remains.

III. Experimental Setup

The simulation designed for this experiment involves a relatively simple predator-prey simulation involving plants, deer, and wolves. All individuals exist on a single two dimensional plane. Deer must eat plants to survive and wolves must eat deer. The novel development of this simulation is that while a learning algorithm is employed, it is not used to improve the performance of individuals. Instead, it is used to tune the world parameters that would otherwise need to be set by a human.

A. Animal Implementation

The deer and wolves are implemented in the exact same way. Each update tick of the simulation, every animal is provided with the location of the nearest food entity and nearest predator entity. For deer, this means plants and wolves, respectively. Wolves have no predator, and thus are always attempting to eat deer. Animals exist on a 1,500 x 1,500 grid with edge wrapping, meaning if an animal walks over the edge, it is moved to the same location on the opposite edge. Animals have a current orientation and location and each tick update can choose to turn and/or move forward. They also have an energy level, which is initialized to some positive amount at the start of the simulation and degrades over time. If this energy level reaches zero, the animal dies. Animals can increase their energy level by eating food.

1) Parameters: Table I lists all tunable parameters for animals. These parameters can be tuned individually for deer and wolves. Parameters decided for deer will be applied to all deer, and the same for wolves. That is, all deer in a simulation will be identical and all wolves will be identical, but deer are not guaranteed to have the same configuration as wolves (and likely will not). All values must at all times be positive and nonzero. Integers are initialized to a random value between 1 and 100 sampled from a uniform distribution. Initial float values are sampled from a uniform distribution between 0.01 and 1.0. Note that while plants are not animals, they do have their own energy value, though it does not degrade as they do not move and their $energy\_loss$ value is 0.

2) Update Procedure: Each update, every animal performs some action. They first set an objective, either food or predator, based on which is closer. If the nearest food entity is closer to the animal than the nearest predator entity, the animal will attempt to move closer to the food. If the nearest predator is closer than the nearest food entity, the animal will attempt to move away from the predator. Because wolves have no predator, their objective will always be food. The animal will then decide if it wishes to turn and/or move. If the objective is food, the animal will turn $turn\_speed$ degrees towards the food entity and move forward if not within $eating\_range$. If

Table I
TUNABLE PARAMETERS FOR ANIMALS

Parameter	Details	Type
turn_speed	How much an animal can turn per tick (in degrees)	float
movement_speed	How much an animal can move per tick (in pixels)	float
eating_range	Radius (in pixels) for how close food has to be for an animal to be able to eat it	int
energy	Initial energy level of the animal	int
energy_loss	How much energy is lost per update, regardless of movement	float
energy_loss_from_turning	How much energy is lost per update if the animal turns	float
energy_loss_from_moving	How much energy is lost per update if the animal moves	float
energy_from_food	What proportion of the food entity's energy the animal gains when eating	float
reproduction_energy	The threshold for how much energy an animal must have to reproduce	int

the animal is within $eating\_range$ of the food entity, it will eat it and gain energy in accordance with Equation 5.

$$self.energy \mathrel{+}= food.energy * self.energy\_from\_food \tag{5}$$

If the objective is to escape from a predator, a similar procedure happens. The animal will attempt to turn to the direction opposing the vector between its location and the predator's location and will always elect to move. This will guarantee the animal attempts to achieve the ideal escape vector. It may also place the animal on a path that brings food closer to it than the predator, in which case, the objective will switch back to moving towards food.

Regardless of objective, after all movement is completed, the animal's energy value will be updated. If the animal turned, its $energy\_loss\_from\_turning$ value will be subtracted from its energy value. If the animal moved, its $energy\_loss\_from\_moving$ value will be subtracted from its energy value. Regardless of decisions made, $energy\_loss$ will be subtracted as well. If the animal's energy value is ≤ 0 at this point, it dies and is removed from the simulation.

3) Reproduction Procedure: An animal is deemed ready to reproduce when $energy > reproduction\_energy$. This check is performed after the animal moves and potentially eats. If this condition is met, the animal's energy is decreased by its initial value and a clone is created at the exact same location as the parent. There is a hard population cap for each species of 200 individuals, so if the population for a species is at 200, no individual can reproduce, regardless of its current energy level. This means that deer can stockpile energy beyond what would normally be possible, which allows the wolf population to gain more energy and reproduce faster to compensate for the surplus of deer.

B. Simulation

There are three additional world parameters that must be configured: the initial population sizes. Obviously, the population of deer and wolves will fluctuate in size as the simulation runs. If either reach zero, the simulation is terminated. To account for the Lotka-Volterra assumption of infinite food for the prey, the plant population is kept constant. Because plant energy remains constant through their lifetime, they are incapable of dying (unless eaten) or reproducing. To compensate, when a plant is eaten, a new one is randomly spawned in. The simulation loop operates as follows, where p, x, and y are the plant, deer, and wolf populations respectively:

initialize p, x, y
for $i = 0;\ i < 20000;\ i\ ++$ **do**
    remove dead entities
    if $p < p_0$ **then**
        spawn $p_0 - p$ new plants
    end if
    for $entity \in x \cup y$ **do**
        update $entity$
        if $entity$ can reproduce **then**
            spawn child and reduce $entity$'s energy
        end if
    end for
    if $x == 0 \parallel y == 0$ **then**
        exit
    end if
end for

C. Learning Algorithm

As stated earlier, the objective of this model is not to use a learning engine to improve the performance of individuals in a population, but rather to optimize the typically manually-tuned parameters that control the simulation. An evolutionary algorithm (EA) was used to perform this optimization, the details of which are laid out in this section.

1) Representation: The evolutionary algorithm must be able to learn the three initial population counts as well as all parameters for each species. As mentioned in Table I, the deer and wolves each have nine learnable parameters. This means the EA will need to learn a total of 22 parameters (plants also have an initial energy parameter). Because float and int parameters are encapsulated by very different value ranges, they must be treated independently. To make for easy management in code, the ten integers are stored in one array, and the twelve floats in another. To allow for smoother convergence, two arrays of mutation strengths (the maximum magnitude of values added to parameters during mutation) were also created, each containing a value that corresponds to an individual parameter. By learning parameter values as well as mutation strengths, peaks in the fitness hyperspace can be climbed more smoothly. Table II describes how each variable is initialized. It should be noted that $reproduction\_energy$ was made to the maximum between the itself and three times

the initial energy value. This ensures that animals will have to consume at least a few food sources before reproducing, which ideally reduces the chance of a population growing too rapidly at the beginning of the simulation.

Table II
INITIAL VALUES OF GENOME VARIABLES

Array	Distribution	Min Value	Max Value
int parameters	uniform	1	100
int mutation strengths	gaussian	1	10
float parameters	uniform	0.01	0.1
float mutation strengths	gaussian	0.01	0.1

2) Crossover & Mutation: The crossover method used is a modified version of uniform crossover, where each array is treated as a separate genome. That is, integer parameters may only cross over with integer parameters, integer mutation strengths may only cross over with integer mutation strengths, etc. This was done to prevent values of differing scales from mixing. A manually-controlled crossover rate was used to determine the likelihood of two values switching between parent genomes.

Mutation was done, again, independently on each array. For each value in both parameter arrays, a random value was sampled from a gaussian distribution centered at zero with a spread of the mutation strength corresponding to that parameter. Then, $reproduction\_energy$ was modified in the same manner that it was initialized, ensuring it is reasonably higher than the initial energy of each species. Finally, mutation strengths were modified with values sampled from a gaussian distribution, centered at zero, with a spread of ten for integers and 0.1 for floats.

3) Evaluation & Selection: The fitness criteria selected for this problem was the number of updates the simulation was able to run for. Simulations that have stable populations should theoretically be able to run indefinitely and as such, longer simulations are deemed "better." The problem with this approach is that the ideal simulation will not be able to be evaluated, as it will never terminate. To compensate for this, a hard limit of 20,000 updates was imposed on all simulations. Each genome was run five times, and the best run taken to be considered for its fitness score. This was done to compensate for the fact that because each animal is placed randomly initially, it is conceivable that wolves spawn in an ideal location to eat all deer quickly, or that the wolf population vastly outnumbers that of the deer population at the beginning. Either would cause the deer population to die off quickly, leading to an early termination of the simulation due to an unfortunate initial configuration.

Survivor selection was done by taking the best half of the population. These individuals were then used as parents. Crossover was performed on randomly selected parents until the genome population reached the same size as it originally was. This was done to intentionally keep the population size down, as evaluation can take multiple hours per individual if the individual is highly fit. All individuals were mutated before being evaluated.

IV. RESULTS

On the surface the algorithm appears to be learning. As shown by Figure 3, the general trend of fitness by generations learned is positive, with an individual hitting the maximum fitness of 20,000 after three four generations. By the tenth generation, the fitnesses of all individuals are fairly evenly spread around 10,000, which was used as the termination condition. The genome of the fittest individual is shown in Table IV.

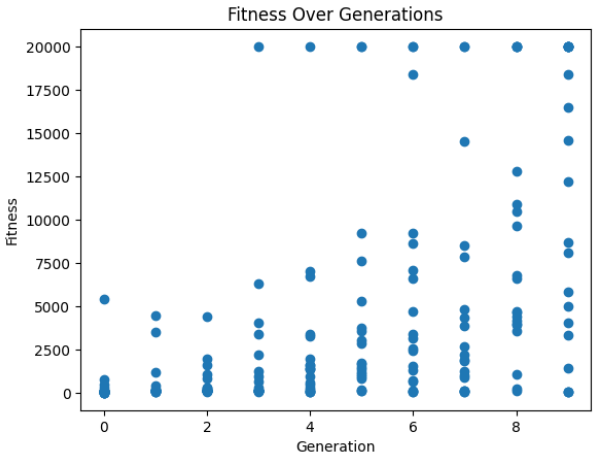

Figure 3. Fitness tends to improve given more learning time

Parameter	Value
plant_count	48
deer_count	50
wolf_count	37
plant_energy	49
deer_turn_speed	0.77
deer_movement_speed	0.67
deer_eating_range	15
deer_energy	68
deer_energy_loss	0.16
deer_energy_loss_from_turning	0.02
deer_energy_loss_from_moving	0.71
deer_energy_from_food	0.41
deer_reproduction_energy	204
wolf_turn_speed	0.91
wolf_movement_speed	0.28
wolf_eating_range	35
wolf_energy	17
wolf_energy_loss	0.39
wolf_energy_loss_from_turning	0.62
wolf_energy_loss_from_moving	0.43
wolf_energy_from_food	0.64
wolf_reproduction_energy	51

This genome gives deer a lifespan of about 76 ticks and wolves a lifespan of about 11.8 ticks, assuming neither eat. Wolves also have a significantly lower movement speed than deer. These two facts would lead one to assume that the wolf population is likely doomed from the beginning. However, this is not the case. The EA revealed some bug in the simulation that allows wolves to reproduce at a rapid rate, right at the beginning of the simulation. This phenomenon is shown in Figure 4.

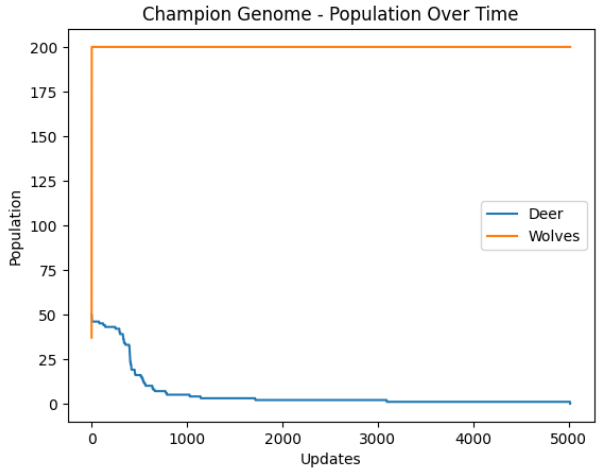

Figure 4. This individual exploits a bug that allows the wolf population to reproduce uncontrollably

Not all individuals are able to exploit this bug. The individual shown in Figure 5 has a decent-looking population curve.

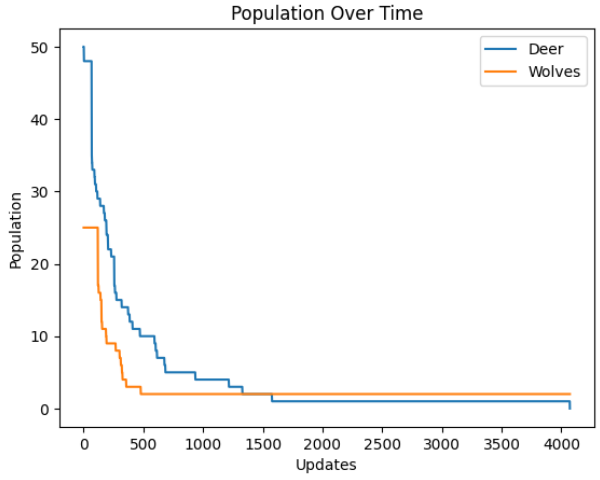

Figure 5. This individual shows a decent population curve

This implies that while the champion genomes are most likely all garbage, the entire population may not be. The trend in Figure 3 demonstrates that the EA is in fact learning. It is for this reason that it is reasonable to do further tests. Ablation tests were done to determine the importance of crossover and mutation. Both were able to improve the model independently, but nowhere near the magnitude of the tests combining them. However, crossover appears to have a more significant impact on learning than mutation.

V. Discussion

Obviously, the fact that the EA is able to somehow cause a population to jump to 200 individuals in the first tick means that the results of this experiment cannot be taken too seriously. Given the inconsistent nature of the phenomenon, it is reasonable to assume that bug occurs not from the

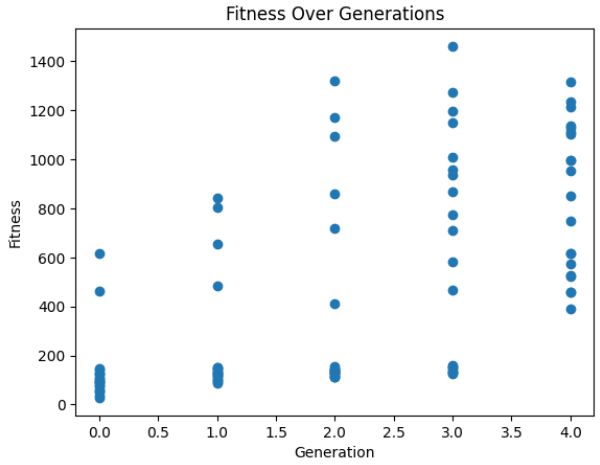

Figure 6. Fitness trends by generation with just crossover

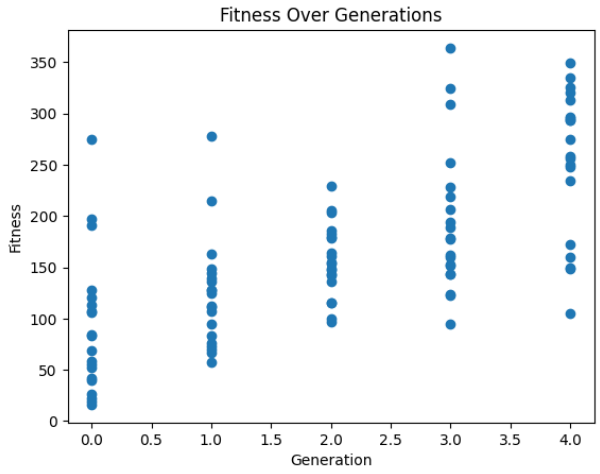

Figure 7. Fitness trends by generation with just mutation

EA, but by random chance in the simulation. If this is true, then the general trend of the EA improving the average performance of the simulations over time may hold some merit. If so, then it is reasonable to say that the objective of automating the tuning of simulation control parameters was successfully accomplished. Additionally, the individual animal representations are extremely simple, with just a couple inputs to a basic logic engine.

If we assume that the EA works as intended and are able to fix the population explosion bug, there are a few avenues to pursue for future work. Firstly, there is some optimization to be done. The most easy optimization is to better parallelize simulations. Currently, each batch of 5 simulations is done by a single thread. However, each trial in a batch is independent of all others, so each of the simulations could be performed by a separate thread. Current runs show that most threads finish early, anywhere from a few seconds to a few minutes, while high-performing genomes can take multiple hours. Also, surrogate functions could be employed to reduce

the CPU time of evaluation, which would allow for larger genome populations (and animal populations, for that matter), as well as allowing for longer simulations. Current simulations are arbitrarily capped at 20,000 updates. Though the ideal infinitely long simulation will never be possible to evaluate, higher update limits will allow for time for populations to grow and shrink, possibly in a fashion similar to that show in Figure 1.

Two reasonable modifications to the model include simplifying the representation and adding a more advanced vision system to the animals. Currently, the genome representation is overly complex to allow for different kinds of variables to be stored easily. However, the genome could just as easily be a single array of positive floats that are scaled as needed during the creation of a species of animal. This would allow for crossover between attributes not supported by the current genome. Additionally, the vision system employed by Tampon [6] is relatively cheap to compute, but provides a more accurate representation of how deer and wolves behave in the wild. Implementing such a system would add some computational overhead, but potentially be worth the trade-off for a more accurate simulation.

References

[1] Mira-Cristiana Anisiu. Lotka, volterra and their model. *Didactica Mathematica*, 32:9–17, 01 2014.

[2] K. A. Hawick, H. A. James, and C. J. Scogings. A zoology of emergent patterns in a predator-prey simulation model. In H. Nyongesa, editor, *Proceedings of the Sixth IASTED International Conference on Modelling, Simulation, and Optimization*, pages 84–89, Gabarone, Botswana, September 2006.

[3] Tom Scott. It's the matrix, but for locusts., Apr 2023. URL https://www.youtube.com/watch?v=KRIBVykhpC4&amp.

[4] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002. doi: 10.1162/106365602320169811.

[5] Jean Tampon. Evolving ais - predator vs prey, who will win?, May 2022. URL https://www.youtube.com/watch?v=qwrp3lB-jkQ.

[6] Jean Tampon. Much bigger simulation, ais learn phalanx, Nov 2022. URL https://www.youtube.com/watch?v=tVNoetVLuQg.