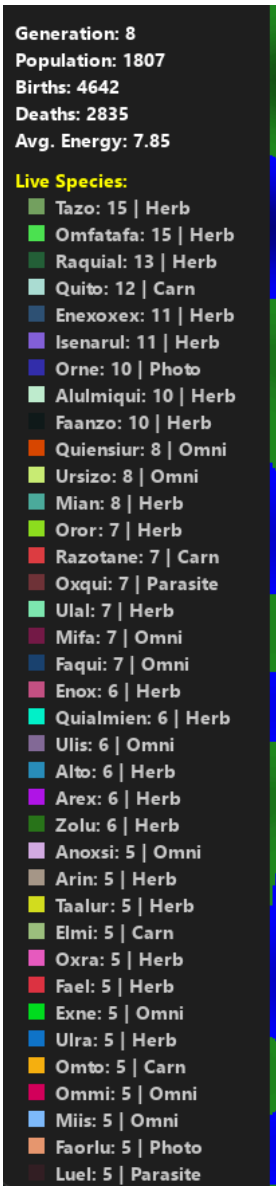
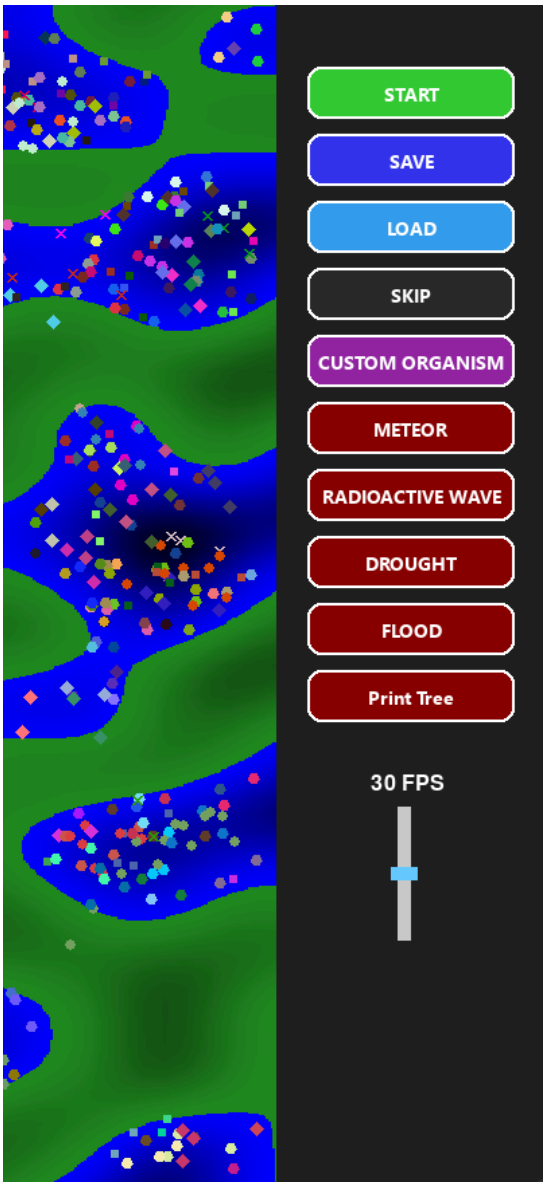
A-Life Challenge Team Final Report

Tallent Hagan, Paris Zhou, Anthony Prudent
CS 467 Spring 2025



Simulation display with statistics tracking and species legend



Simulation display with buttons for controls and hazards

Table of Contents

I. Introduction

This project is an artificial life simulator built with Python, NumPy, and Pygame. Autonomous organisms interact, adapt, and evolve within a 2D environment. Each organism is driven by a genome, encoding traits like size, speed, vision, diet type, and abilities such as swimming or flying. These genetic instructions shape how they move, survive, and compete.

The simulation features a procedurally generated terrain with a mix of land and bodies of water. Organisms explore, flee predators, chase prey, and reproduce with mutations introducing evolutionary change over time. Traits like camouflage, metabolism, and fertility influence survival in the ecosystem.

Users can see real-time statistics, species divergence, and population trends. A color-coded display shows active species and current statistics, while a right-side panel includes sliders and buttons that allow for control of and changes to the simulation.

II. Software Description

The program begins with a main menu that allows the user to set the initial parameters, this includes the number of organisms, the size of the environment, and the base mutation rate. In the current implementation, the simulation initializes by creating the specified number of organisms with randomly generated genomes.

In the display, each organism is rendered using a set shape based on their diet type with a color unique to their species. General simulation statistics are shown on the left side menu, including current population and number of new births. Below the statistics, is a list of all current species with species' name, population, diet-type, and color legend. On the right side menu is the list of simulation controls, this includes start/stop, save/load, skip, and a slider for simulation speed. The custom species button, will generate new organisms based on input from the user. There are also a number of environmental disaster options, such as a meteor strike, or a radioactive wave that mutates all organisms.

III. Development Efforts Description

A. Tallent Hagan

My initial development was on setting up PyGame for rendering and UI controls. This included learning how visuals work, how to create the GUI, and how to handle input. My original prototype was a colored-variant of Conway's Game of Life.

PyGame was also a great tool for not just rendering the simulation, but also for creating/handling the functionality of the GUI. The only GUI element developed outside PyGame was the custom organism, where we used tkinter, as creating a long list of input fields was cumbersome to implement in PyGame.

Once everything was connected, I worked on the controls. Using PyGame's event handler options allowed us to connect mouse clicks (and drags) on our visual elements such as the buttons to the various functions for the controls, such as starting and stopping the simulation. For saving and loading, we made the environment also hold all information on current organisms, so that we could pickle the environment object then reload it from that pickle file.

My last efforts were on custom organism functionality and the meteor hazard. I used tkinter to handle user inputs. I refactored our code for initial spawning of organisms, to also generate organisms mid-simulation when passed a dictionary from custom organisms. For the meteor, I created animations that allowed it to move across the screen and create a crater at a randomized impact location. Damage to surrounding organisms was done through a modified version of the organism attack function.

Throughout, we also held multiple weekly meetings and working sessions. Deviations occurred from problems implementing our original plan.The Organism and Genome were to be completely separate classes. However, handling individual decision-making was not being feasible given thousands of organisms to track with thousands of genomes. We refactored several times before deciding on a numpy dtype array to track genome values and physical values such as positioning. This, and a spatial index algorithm, allowed for much more efficient calculations over a large population.

B. Paris Zhou

Initially, I was charged with the objective of creating the environment that would house our organisms. It was decided we would start with an NxN matrix with discrete cells to represent our simulation. This was following in the steps of Conway's game of life, which had discrete cells with simple rules governing the life and death of occupied cells. We quickly realized that it would be exceptionally difficult to use discrete cells to create a simulation that facilitates emergent behavior alongside our longer term stretch goals such as biomes, hazards, sexual reproduction, genetic mutation, and other features. I decided to switch our environment into a 2d plane implementation as it would better allow for us to create unique organisms with emergent behavior without copying Conway. The use of a 2d plane would not interfere with rendering of organisms.

I fortunately found a great video on YouTube about how to create realistic 3d terrain. Using principles from this video and a blog post in the description, I was able to create randomly generated 2d terrain for more organic organism movement and for a more interesting simulation.

Once I had finished the 2d plane, I suggested we unpack our organism representation from python class objects and instead represent our organisms in a more abstract fashion. Organisms would instead be represented across separated arrays rather than all packed into separate python objects. The logic behind this decision is elaborated upon more in the major technologies section of this report.

The most complicated and time-consuming portion of implementation was organism behavior and species lineage tracking.

Organism behavior was based on 17 different genes and the terrain. Morphological genes such as attack, defense, vision, and camouflage impacted movement because organisms would "chase" other organisms with lower net attacks

{ max((our attack - their defense), (their attack - our defense)) }

and could only attack organisms if their vision gene was greater than the camouflage gene of their target.

- Speed impacted the distance an organism can move within a frame.
- Metabolism rate impacted the costs incurred by displacement per frame.
- Nutrient efficiency impacts how much energy is gained from net attacks performed.
- Diet type impacts organism interactions, photosynthesizing plants cannot attack anyone, carnivores cannot attack photosynthesizers but could attack herbivores and omnivores.
- Fertility rate impacts the threshold of energy required for reproduction.
- Offspring count was never fully implemented but would have increased offspring per reproducing event.
- Pack behavior determined whether an organisms would move in a pack to hunt.
- Symbiotic genes determined whether organisms would share energy with other symbiotic organisms.
- Swim, Walk, Fly genes would determine what terrain could be traversed.
- Current age and Max age were used to prevent runaway reproduction and would determine for how many frames an organism could live.

After finishing organism movement and behavior, I worked on a method for tracking lineage and speciation. I settled on using a dictionary to map speciation events and tracking parent and child id's with a p_id and c_id field for organisms. This allowed the use of DFS to explore the evolutionary tree of a simulation and subsequently allowed storage of simulation data.

C. *Anthony Prudent*

At the beginning of development, my efforts were not as drastically different from the project plan due to the need of implementing rudimentary ideas. As such, I worked on setting up organism classes and designing the genes.

As development furthered, deviations from the plan occurred. The main problem we were facing was a rework of most of the codebase due to performance issues. To solve this issue, we had to quickly learn how to use NumPy, which I had to carefully read documentation and example code to

fully understand. Despite my lack of experience, I was able to clean up the new codebase into a readable and more maintainable format.

During this time, species were also supposed to be named using AI, but as I looked deeper into the implementation, it was not feasible due to potential efficiency issues and financial costs. An additional deviation from the plan was integrating the organisms with the environment instead of developing the organism action system due to major changes in the code using NumPy.

From here, development loosely followed the project plan due to time constraints and solidification of team roles. During this time, I became less focused on specific parts of the program, but rather filling in the gaps where needed. Instead of focusing on features such as sexual reproduction and biomes, I instead focused on implementing environmental hazards to meet project needs, allowing me to gain a better understanding of the program as a whole.

IV. Major Software Tools and Libraries

Our decision to use python was based on a combination of factors. Between all of us, we were all best at using python so logically having the majority of development within python would cause the least amount of friction. Python also has excellent library support. Though there could have been more computationally efficient languages to incorporate within the stack, the increased prototyping speed afforded by python more than made up for any losses in performance. With python we were able to rapidly implement ideas we had and vet any design within a week's time. Similarly we chose pytest to test our simulation during development due to the group's familiarity with the testing framework.

A subsequent major decision we had to make was determining the amount of numpy we wanted to use in the simulation. Since Python is an interpreted language and is limited in execution by the Global Interpreter Lock (GIL), frequent computations such as matrix operations were going to be severely bottlenecked by our choice of language. We decided to circumvent these computational limitations by leveraging NumPy and SciPy python library tools frequently. These libraries were built to interoperate with C to circumvent GIL limitations in execution speed via the Basic Linear Algebra Subprogram (BLAS) library and Single Instruction Multiple Data (SIMD) capabilities of C and Fortran [1]. These tools helped significantly in speeding up cumbersome matrix operations while maintaining pythonic syntax within our source code.

The use of numpy in our code added a tradeoff we had to consider because python class objects could not naturally leverage the benefits of BLAS and SIMD. Using native python classes to represent our organisms requires us to unpack data from python objects into BLAS/SIMD capable data, perform necessary calculations, and then repack into python objects. This seemed too cumbersome. It follows then that rather than having organisms stored within a singular object and operated upon by the simulation, it would be more efficient to store our organisms spread about multiple arrays within a singular numpy data type called organism_dtype. Though this may appear to be an inconsequential differentiation from the prior design decision it has widespread implications for performance improvement. The packing of our data into numpy's dtype object rather than a native python class allows for more efficient calculations while preserving some pythonic syntax.

It seems strange to write out Github as a technology that we leveraged because it is so integral and so extremely idiomatic when it comes to making a decision on what VCS to use. There is a reason it is so widely adopted. The use of Github for our version control system (VCS) was pivotal for feature development speed and prototyping. Github branching and workflow options helped us iterate quickly as our team could work on multiple features at once independently without having to worry about compatibility issues while prototyping and simultaneously relieved our worries during development about creating an unusable program and erasing prior work.

Though brief, the use of the noise library and blog posts by Inigo Quilez [2] were critical when determining how to create terrain. Initially the terrain was discrete and represented with a boolean mask over our 2-D plane. This was fine for a while until we implemented non-random movement for our organisms. Using movement only based on organisms nearby created strange movement patterns as organisms might choose to nonsensically chase other organisms onto a terrain they cannot traverse and subsequently die. We then changed the terrain to a continuous over our 2d plane area to implement more realistic movement. The terrain nudging would be unnoticeable for an organism on traversable terrain but increase asymptotically as an organism would approach untraversable terrain.

V. Conclusion

Our collaborative effort allowed us to bring together biological modeling, procedural graphics, and interactive design within a real-time simulation. Built with Python, NumPy, and Pygame, the project evolved significantly from its original plan in response to technical challenges and performance constraints. By shifting to efficient data structures like NumPy dtype arrays and implementing a spatial index system, our team successfully scaled the simulation to support thousands of dynamically interacting organisms.

While not all planned feature were implemented, such as AI-driven species naming or complex reproductive systems, the core objectives were met and expanded with additions like natural disasters, a custom organism generator, and visual statistics tracking. Our software offers an engaging, controllable ecosystem with emergent behaviors and evolutionary dynamics.

References:

[1] NumPy Developers, "NumPy Documentation," NumPy, [Online]. Available: https://numpy.org/doc/. [Accessed: 4-Jun-2025].

[2] I. Quilez, "More Noise," iquilezles.org, [Online]. Available: https://iquilezles.org/articles/morenoise/. [Accessed: 4-Jun-2025].