Casey Primozic
2018-12-13
Simulation and Modeling
Final Project Report, Final Draft

Ant Colony Simulation
Motivation, Design, Implementation, and Results
Casey Primozic

Introduction + Background

That natural world as it is today is the product of millions of years of evolution. The Earth, aside from events like asteroid impacts and solar activity, is a closed system that has had the chance to continuously develop over the millennia. Life, as part of its goals of survival and reproduction, self-optimizes through evolution in response to changes in its environment and in other species that co-exist with it. Along the way, it has developed many unique and elegant ways of adapting and becoming as efficient as possible. Many patterns developed in nature have been studied and adopted for use in computing. Neural networks, the basis of modern deep learning, evens bear the name of its inspiration (the networks of neurons that exist in the brains of countless species). Genetic algorithms mimic the behavior of DNA as it mutates from generation to generation, creating diversity and underpinning the process of evolution.

Ant colonies are another example of a natural system which has developed algorithmic behavior which was later adopted by computer science to solve problems as far flung as traffic routing and layout optimization. Individually, ants don't seem very impressive. However, as a group, they are capable of impressive feats of organization, construction, and coordination rivaling even those that humans are capable of with the aid of modern technology and spoken language. This behavior is known as *swam intelligence* – the process whereby a group of entities acts collectively in ways that are not possible individually. Directly or indirectly, they share information with each other and make more informed decisions based on the knowledge of other entities, even though that knowledge isn't directly available to them all individually.

As the goal of this simulation, I aim to replicate this swarm intelligence behavior. More specifically, I seek to produce a system capable of exhibiting the *ant colony optimization* – an example of swarm intelligence that boils down to finding the best paths through connected graphs. In order to do this, an agent-based simulation framework that I created called Minutiae[1] will be used and a set of rules defining the layout of the simulated world, its rules and constraints, and the behavior of ants will be defined. Specifically, the goal is to determine what the minimum amount of information made available to each ant is in order to see evidence of swarm intelligence via the ant colony optimization. By minimizing the amount of information that each ant has at any given point during the simulation, the colony's ability to operate more effectively together rather than singularly will be more clearly evident.

Simulation Framework Background - Minutiae

The Minutiae simulation framework was used to implement this simulation. Minutiae is agent-based, operating on a finite 2D **universe** (hereafter referred to as **U**) composed of **cells** (**C**) and **entities** (**E**). One cell exists at each coordinate within the universe, and any number of entities can exist at each of the universe's cells. Every cell has a state (**CS**), and each entity has a

[1] https://github.com/ameobea/minutiae

state as well (**ES**) along with a private, mutable state (**MES**). Each of these states can be defined by the user for each individual simulation. Since Minutiae is implemented in the Rust programming language, these states are strongly typed and verified at compile time to be valid. Other guarantees provided by Rust also apply, such as exhaustiveness checks, which contribute to the simulation's verification. In addition, Rust's **fat enums** are an excellent choice for representing these states. Fat enums are **tagged unions**, meaning a finite enumeration of possible values with optional corresponding state for each of them. For example, a car could be modeled as a fat enum of transmission gears as well as speed moving forward and reverse:

```rust
enum CarGear {
    Park,
    Neutral { speed: f32, },
    Drive { pedal_activation: f32, speed: f32, },
    Reverse { pedal_activation: f32, speed: f32, },
}
```

Both neutral, drive, and reverse have attached *speed* state which isn't shared by park since the car can't move while in park. Additionally, drive and reverse have pedal activation state to represent how far the pedal is depressed. This representation makes showing the relevant information about an entity's state extremely clear and helps to create robust handling functions.

Another important aspect of Minutiae is the universe's immutability. Each tick, the simulation can be thought of as atomically transitioning from one state to another, similar to a cellular automata. Rather than allowing entities to modify cells or other entities directly, they instead emit **actions** which are then processed by the simulation's **engine** and applied to the universe. There are three varieties of actions: Cell Actions which target a single cell (**CA**), Self Actions which target an entity's public state (**SA**), and Entity Actions which target another entity (**EA**). All of these action types are also created by the user per-simulation and enjoy the same static typing and verification guarantees as cell and entity states. The engine is a set of functions supplied by the user to handle driving entity behavior and handling dispatched actions.

The simulation process proceeds each tick by first querying each entity (either serially in a well-defined order to allow for determinism and strong ordering or in parallel to maximize simulation performance) and running the entity driver function on it. A common pattern for implementing entity states is a **finite state machine**, which can represented as a function that transitions between a finite set of states. This function has three inputs and two outputs. The inputs include the previous **U** (or a subset of **U** such as the nearest *n* adjacent cells), the current **ES** for the entity, as well as the entity's **MES**. Both **U** and **ES** are guaranteed to be immutable by Rust, ensuring that they can't be accidentally modified during the entity driving process. **MES** is a potential source of impurity in the function, meaning that for the same **U** and **ES** the same function may exhibit different behavior. However, this is very important for things like entity-local pseudo-random number generation state that would be difficult to implement using **ES** and **SA**. The output of this function is a new **MES** (which in reality is simply mutable but can be thought of as being returned and assigned to the entity for the sake of a formal consideration of the system) and a list of actions to be applied to the universe.

Once all entities have been queried and their dispatched actions collected, the next step is to apply them to the universe. This is handled by three functions, one for each kind of action (**CA, SA,** and **EA**) which are defined by the user. It is the job of these actions to enforce the rules of the universe and ensure that entities behave within the bounds of its constraints. For example, translation actions (which represent an entity's desire to move to a different cell) must be checked to be within the bounds of the universe and within a specified movement range (if one is set). Also, actions targeting entities must be checked to ensure that the target entity still exists (entities can be deleted) or where it is expected to be (actions targeting entities that have moved can be ignored). Minutiae includes built-in functions to help with these cases and others. As a result of these handler functions, a new **U** is created.

The final component of is **middleware** (**M**). Middleware is simply a function that is given full, mutable access to the universe along with its cells and entities before and after each tick. Middleware allows behavior that doesn't fit within Minutiae's strict design to be implemented. A perfect example of this is modifications made to cells without being instigated by entities, which is used in this simulation in the form of pheromone evaporation. Also, middleware is used to implement visualization of the universe by taking a function mapping the (**CS, ES[]**) tuple for each cell in the universe and returning a color which can be plotted to visualize the simulation.

Following to this design, the simulation is run tick by tick with the result being plotted. As another result of Minutiae being written in Rust, the whole thing can be compiled to WebAssembly[2] and run in a web browser. WebAssembly is a binary bytecode format executed on a stack machine that is supported by all modern browsers, allowing near-native performance for code written in a variety of programming languages that support it as a compilation target. The Minutiae simulation can interoperate directly with JavaScript, allowing it to be seamlessly presented as a dynamic web application. For this simulation, a programmatically generated UI created using a library called `react-control-panel`[3] (a port of the `control-panel`[4] library that I implemented using the React framework) is used to enable dynamic control of the simulation's parameters. As the UI is adjusted, the corresponding simulation variables are updated live and the modified behavior is reflected in real time in the simulation and its visualization. The visualization is implemented using WebGL[5], a GPU-accelerated graphics API that allows for the color values produced by the simulation to be scaled and rendered very efficiently.

Simulation Design and Constraints

The primary goal of this model is to produce evidence of swarm evidence and replicate the ant colony optimization through an entity-based simulation of an ant colony while minimizing the information made available to each ant at any given time. The best way to outline the simulation's design is to show the selected **CS** and **ES**. **CS** consists of four possible states: Empty (with two continuous pheromone levels), Barrier (impassible terrain that ants must navigate around), Food along with an associated food amount, and Anthill which is the origin point of all ant entities and the location where collected food must be deposited. **ES** is implemented as a finite state machine consisting of three states: `Wandering` (ants navigate through the world in search of undiscovered food patches), `FollowingPheremonesToFood` (ants follow pheromone trails to previously discovered food patches), and `ReturningToNestWithFood` (ants follow pheromone trails to the anthill to deposit food that they are carrying). Both `FollowingPheremonesToFood` and `ReturningToNestWithFood` include associated state recording the last direction in which the ant moved.

[2] https://webassembly.org/
[3] https://github.com/Ameobea/react-control-panel
[4] https://github.com/freeman-lab/control-panel
[5] https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

There are two different pheromones which ants may deposit: Wandering pheromone (blue) which is deposited by wandering ants as they traverse the world in search of food and Returning pheromone (red) which is deposited by ants carrying food as they make their way back to the anthill. Each empty cell has continuous amounts of these, bounded by a configurable maximum amount.

Ants have access to the state of all immediately surrounding (including diagonally) cells and their state as well as the cell they are standing on. They can move to any one of these visible cells as long as it is empty or an anthill. Multiple ants can exist on the same cell, and ants can not jump diagonally across barrier entities. Ants can deposit (configurably) variable amounts of pheromone onto the cell they currently reside on. **MES** is initialized to the coordinates of the nest when ants are created, but it is only accessed when ants are in the `ReturningToNestWithFood` state.

The world is generated with an initial set of fully blank cells with no pheromones deposited anywhere. To this, food patches and barriers are generated according to the simulation's world generation parameters. The frequency and size of both food patches and barriers are configurable independently. Finally, an anthill is generated at a random location in the world and all ants are spawned on top of it. Ants share a global PRNG which they use as a source of randomness when making decisions. Pheromones are periodically evaporated by a configurable amount until they evaporate completely.

Simulation Design Basis + Justification

All aspects of the simulation were chosen to align with the makeup and behavior of real ant colonies as closely as possible while taking a reductionist approach in order to meet the requirements of the simulation objectives. In nature, ants make use of complicated systems of pheromonal communication in many aspects of their behavior including breeding, defense, and the collection of food. This system is "[...] extremely complicated and uses at least five different systems [of pheromones]." [3]. A wide range of chemicals, glands, and dispensing mechanisms are used to deposit these pheromones and transmit them to other ants. Different types, quantities, and combinations of these chemicals allow ants to coordinate very complex activities while, in some cases, being completely blind [3].

The rationale behind selecting the wandering and returning pheromones for the simulation are based in the biphasic behavior exhibited in the foraging patterns of some ant species. Ants such as the army ant (most species belonging to the subfamily Ecitoninae) switch between nomadic wandering and coordinated group behavior: "Army-ant (legionary) behavior is characterized by the combination of two discrete features, nomadism and group-predation, both of which are maximally developed in the subfamily Dorylinae" [4]. This behavior is regulated almost entirely by pheromonal coordination. By choosing two pheromones to represent these two modes of behavior, this behavior can be effectively modeled while keeping the model's complexity to a minimum in order to more clearly see the underlying behavior and reduce possible sources of information leakage.

When returning to the anthill, ants are given access to their location relative to it. This is the only piece of information that is available to them outside of their current state machine state (the direction of their previous step is more of a model of their physical orientation than some kind of information they hold internally, so it isn't counted here). The reason for this addition is that, given the limited pheromonal framework and behavioral system that the simulation sets in place, keeping track of direction and orientation is very difficult. In the real world, ants use a variety of techniques of keeping track of their position by using such techniques as visual recognition of landmarks and physically following other ants [3]. Ants even make use of the number of steps they take as a method of returning to previous locations: "These results support the hypothesis that desert ants use a pedometer for distance measurement, or a step integrator [...]" [5]. As an approximation of these and similar navigation systems, the ants' offset from the anthill is used to aid their return, and only when they are in the `ReturningToNestWithFood` state. All other behavior is orchestrated entirely based on the concentrations of the two simulated pheromones and simulation parameters.

Ant Behavior

Ants make probabilistic choices about their movement through the world based on their current state as well as the levels of pheromones that are visible to them. When choosing what cell to move to, they use an algorithm to calculate weights of all of the cells surrounding them and then pick between them based on their weights.

When ants are in the `FollowingPheremonesToFood` state, cells are weighted higher based on their concentration of Returning pheromone as well as if they are in the general direction that the ant moved in on the last tick. This models ants following trails that ants carrying food deposited on their return voyage to the nest as well as a bias towards moving in a more or less straight direction. Ants in this state transition to a `Wandering` state if surrounding Returning pheromone levels drop low enough.

When ants are in the `ReturningToNestWithFood` state, they have the same bias to move in the same direction as they were moving, but they also gain a bias towards moving in the direction of the nest with regards to its position offset from their current location. They also move towards higher concentrations of the Wandering pheromone, operating under the assumption that wandering ants laying the trail originated at the nest and that there will be higher concentrations of that pheromone near the nest as it is deposited by ants in the `FollowingPheremonesToFood` state as they make their way to food sources from the nest. Ants in this state transition into a state where they move randomly if visible Wandering pheromones drop below a threshold, attempting to find a new trail that they can follow back to the anthill.

Ants in the `Wandering` state do not focus on pheromones, instead being strongly biased towards walking in a straight line until they encounter an obstacle or find food. Along the way, they lay a Wandering pheromone wherever they walk and transition to the `FollowingPheremonesToFood` state if they encounter a strong enough Returning pheromone trail.

All of the weights and biases for these different modes of behavior are adjustable via the UI and can be altered dynamically while the simulation is running. In addition to using pheromone weights, a configurable "base" weight can be added to cells which helps encourage ants to explore new territory and find more effective routes than the ones already mapped.

Verification

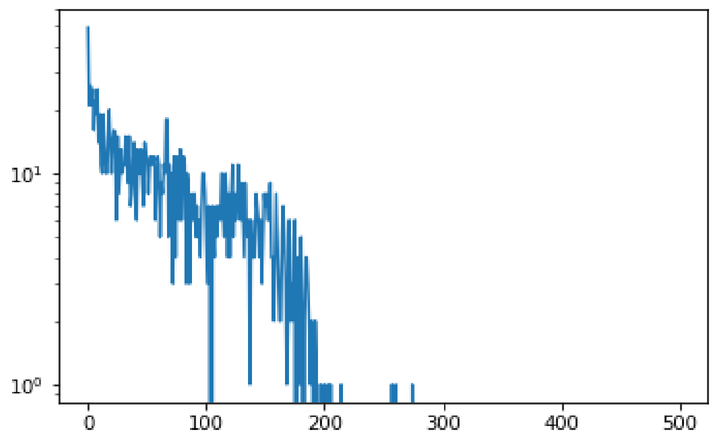
A large part of the verification and validation of the simulation is implemented via the guarantees of the Minutiae platform. One of the strongest bases of the simulation is the limitation of information made available to the ant entities. In order to ensure that no data such as the state of cells out of range of the entity or the location of the anthill when the ant isn't returning is available, the strict structure of the entity driver function and the data available through its arguments act as protection against data leakage. In addition, Minutiae's strictly enforced policy of immutability except through explicit actions goes a long way in preventing bugs in the simulation's implementation.

The other part of the verification for the simulation was carried out through visual analysis of the simulation's performance in the visualization. Due to the large amount of data available through the visualization and the characteristics of the simulation's design, it is possible to ascertain a great deal of information about how the simulation is performing in real time simply by watching the visualization. Obvious problems such as ants violating simulation rules are immediately evident, and the results of parameter modifications become apparent in real time.

Results + Validation

Depending on the properties of the initial world generated by the simulation and the parameters chose, dramatically different behavior can come to be exhibited. Although obvious factors such as the quantity of food and number of ants play a huge part, more nuanced settings such as the maximum amount of pheromone that can be deposited on a cell also play huge roles in simulation performance. Even tiny tweaks such as changing the base weight for cells when moving by single percents can completely alter the way a simulation progresses.

As a method of evaluating the performance of different parameters for the simulation, the total amount of food collected per 100-tick period was recorded for the first 50,000 ticks of the simulation[6]. This allows for both the overall performance of the simulation as well as changes in the simulation's performance over time to be evaluated. To find the optimal set of parameters to maximize the colony's average food collected per tick, settings were changed one or two at a time in response to observations made via the visualization.
The first configuration started with default settings used during development. They were not well tuned, and after ~20,000 iterations the ants were all stuck on “islands” of pheromones with no connection to the anthill and ceased food collection completely:

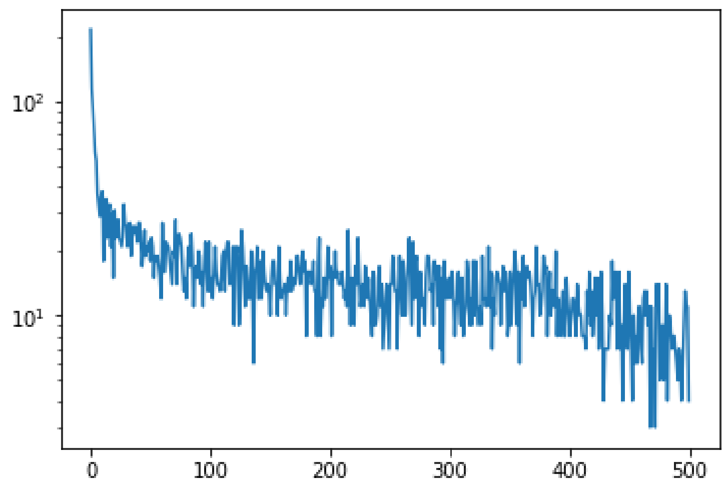


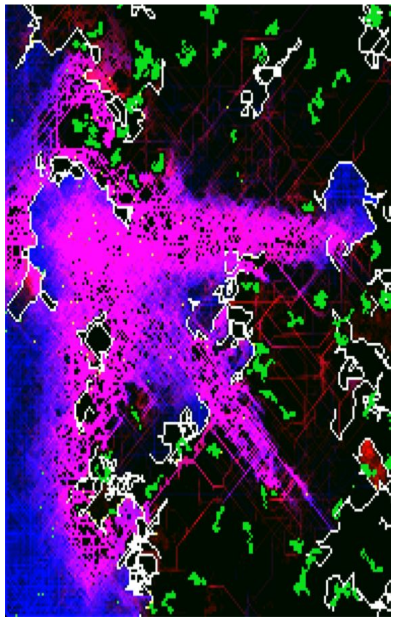
This problem was resolved by turning up the innate bias of all cells from zero (meaning that when returning to the colony, ants will never step off of cells with at least some wandering pheromone) to allow for ants to discover routes back.

Over several iterations, the bias to moving towards the nest while returning with food was increased by 70%, the pheromone evaporation rate was reduced by 30%, and the maximum

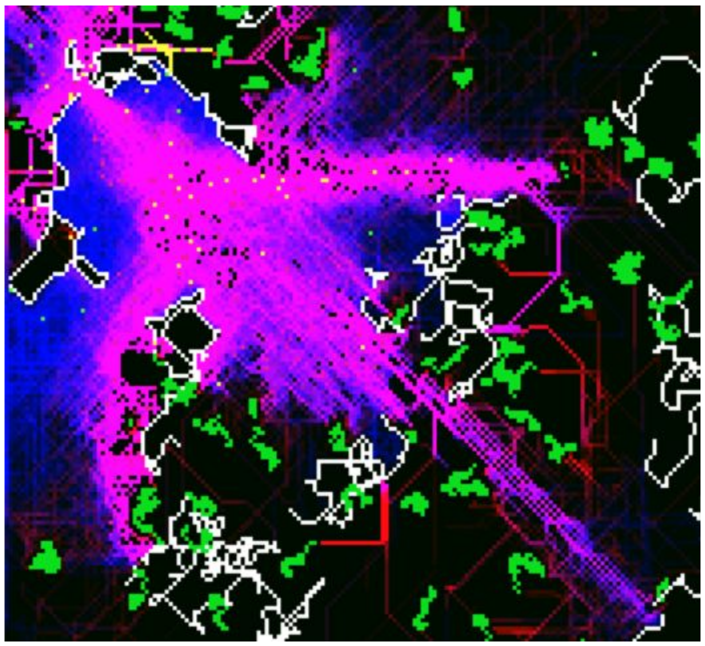
[6] Jupyter Notebook with statistics and graphs: https://gist.github.com/Ameobea/424b9dbc33cbbcb77b793ec39c5c68e0

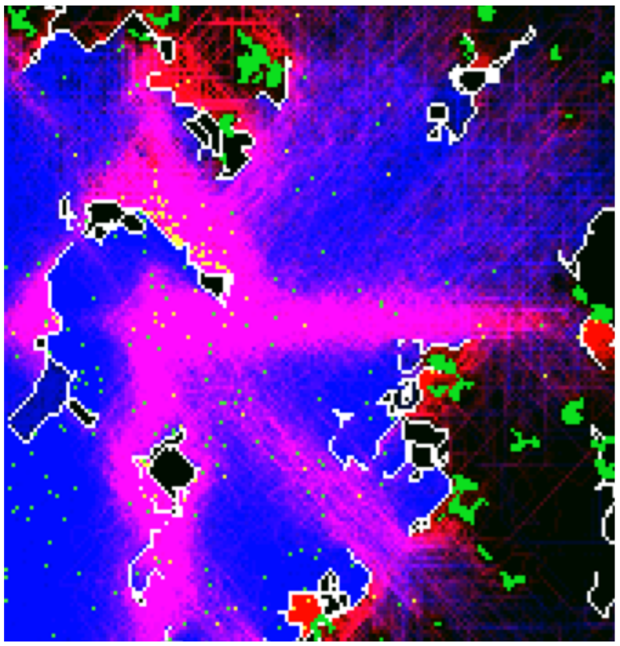
pheromone per cell reduced from 15.0 to 10.0. As a result, a stable colony developed with an average food per tick of 0.15486:



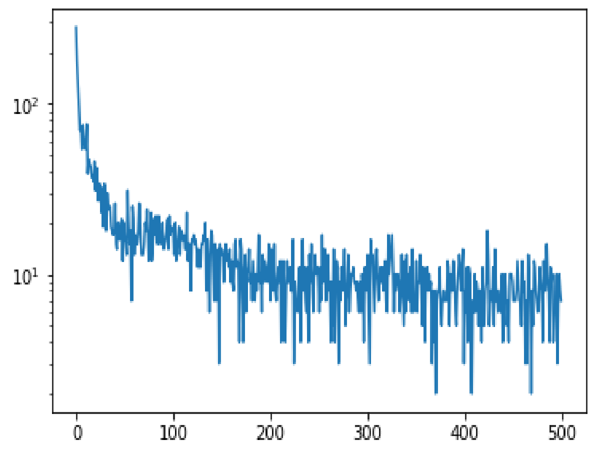


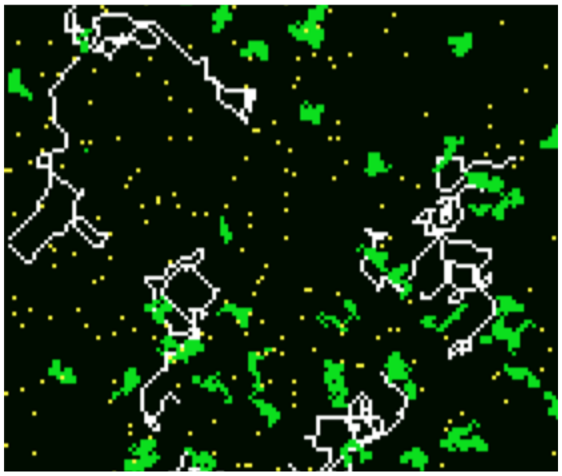
After expanding the ranges of some settings past the limits I had previously set for them, I found that efficiency could be increased by upping the bias towards moving in the same direction as the previous tick dramatically. After increasing that bias from 2.5 all the way up to 50 and increasing the bias towards the anthill as well, efficiency went up an additional 40% to a maximum average of 0.2578 food per tick.





As a base case to verify that the pheromones actually did have a positive impact on the ants' ability to effectively find and gather food, the simulation's settings were modified to have a maximum pheromone level of 0. The bias of moving towards the anthill was retained, as was the bias to continue moving in the same direction as the previous tick. The results (average food per tick of 0.13874) clearly show that ants were at a significant disadvantage working alone without the benefit of pheromones. This decrease in efficiency would be even more dramatic if the world had more barriers in it, forcing ants to wander for even longer before finding their targets. Although they were able to fairly easily gather the nearby resources through mostly random movement and the returning bias, ants quickly became stuck in concave barriers due to the lack of pheromones to guide them around them. They also failed to explore beyond stretches of barrier in due to a lack of pheromones to follow long distances away from the anthill.





As is clearly demonstrated by the results, the cooperative behavior made possible by pheromonal communication greatly increases the ants' ability to locate and gather food. The fact that small changes to simulation parameters and ant behavior variables can have drastic impacts on their performance is also evident.

Challenges + Problems

Although Minutiae went a long way to help with the plumbing and correctness of the simulation, there were several problems encountered during the simulation's development. One of the first issues that I came across was entities getting stuck in various parts of the world. Initial experimentation with simulation parameters led me to find that using a high attraction bias to the anthill helped a lot with getting ants home and collecting food faster. However, it also caused ants to ignore the systems that were designed to help with getting stuck, namely the pheromones. It got to the point that ants would end up entirely sealed off in pheromone "islands" wedged in a corner somewhere, unable to wiggle around enough to create a connecting pheromone trail out. To resolve this, I added a small amount of innate bias to cells that don't have any pheromones on them at all. I also added in the bias for continuing in the the direction of previous movement. Together with well-turned pheromone weights and pheromone evaporation / max settings, ants were able to much more effectively gather food.

An additional area of difficulty while implementing the early stages of the simulation involved the development cycle. The output of Minutiae's visualization function was an array of RGBA pixel data. Rendering this to a canvas using built-in 2D canvas APIs was trivial, but the resulting image was extremely small (each ant was only a single pixel) and the visualization was next to useless. Manually scaling the visualization by copying pixels or using the built-in image scaling APIs was very slow as it was executed on the CPU and blocked the simulation loop while it processed. As a solution, WebGL shaders were written which handled sampling the small image created by Minutiae and rendering it larger on the canvas. The implementation of this was not trivial – WebGL is an extensive and low-level API that requires the programmer to handle many of the details involved with shipping textures to the GPU, transforming them, and displaying the result on a canvas.

The only other considerable issue encountered had to do with dealing with the large number and extreme sensitivity of simulation parameters. It was very easy during testing to encounter combinations of parameters that led to undesirable behavior and then be unable to tweak settings back to how they were in order to get rid of it. The complex interactions between the different variables coupled with the fact that many of them were heavily dependent on existing simulation state often led to the best choice being a hard reset to defaults.

Future Work

One area of future work that I would be interested in pursuing is dynamically modifying simulation parameters over time, in response to different events in the simulation or according to ants' positions in the world. One problem that the simulation encounters is with pheromone evaporation with regards to the size of the universe. As ants exhaust food resources near the colony and must walk further and further to find viable food supplies, pheromone evaporation can lead to serious problems with ants' ability to coordinate and collect food. Allowing for the pheromone intensity left by ants to increase the further away it is from the anthill may prove useful for allowing ants to continue effective gathering at longer distances. Additionally, varying the attraction to the anthill depending on how far away ants are from it would likely greatly improve ants' ability to successfully return to the nest. As demonstrated in the results, higher attraction factors proved extremely influential in allowing ants to find the anthill faster, but they resulted in serious issues with ants getting stuck behind concave barriers.

Additionally, it may prove interesting to add economic aspects to the simulation. Allowing ants to reproduce based on the amount of food they collect and allowing food to repopulate may give rise to self-regulating behavior and allow another aspect of ant life to be modeled.

I would be interested in pursuing more advanced methods of parameter selection. Gradient descent, an optimization algorithm for functions with large numbers of variables that is extremely popular in machine learning, may be a good fit for automated parameter optimization in this simulation, I foresee problems with high simulation simulation time being a factor, but it may be possible to at least partially alleviate them by paralleling the simulation using web workers.

Conclusion

The behavior of ant colonies is very complex, involving many interconnected systems and interdependent communications systems. Swarm intelligence allows individual ants to work together as a collective in order to create non-linear gains in efficiency through cooperation. This simulation demonstrates those same characteristics, allowing users to visualize the ant colony optimization and observe the virtual ants' behavior in real time. Minutiae's strengths are well exhibited, and the end product is made accessible to a wide audience through its availability in a web browser with no installation or technical requirements at all.

In addition to providing a meaningful model of its targeted phenomena, the simulation provides a good example of an interactive and accessible tool which people can use to experiment for themselves and discover dynamically. Rather than statically replicating something and spitting out results, the entire process unfolds live, giving the user a chance to pause and look closer along the way. Together with appropriate background information, it can help people understand the core concepts of swarm intelligence and collective decision making while also demonstrating the possibilities of agent-based simulations.

Live link to the simulation itself: https://ants.ameo.design/

Full source code for the simulation: https://github.com/Ameobea/sketches/tree/ants

Minutiae's Source Code: https://github.com/Ameobea/minutiae

Works Cited

[1] David Hutchison et al. *Ant Colony Optimization and Swarm Intelligence*, Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

[2] Jason Brownlee. 2012. *Clever algorithms: nature-inspired programming recipes*, United Kingdom: LuLu.com.

[3] Robert K. Vander Meer. 1998. *Pheromone communication in social insects: ants, wasps, bees, and termites*, Boulder, CO: Westview Pr.

[4] E.O. Wilson. 1958. The Beginnings of Nomadic and Group-Predatory Behavior in the Ponerine Ants. *Evolution* 12, 1 (1958), 24. DOI:http://dx.doi.org/10.2307/2405901

[5] Matthias Wittlinger, Rüdiger Wehner, and Harald Wolf. 2006. The ant odometer: stepping on stilts and stumps. *e-Neuroforum* 12, 3 (January 2006). DOI:http://dx.doi.org/10.1515/nf-2006-0307