M20 - Introduction to Computer Programming with MATLAB
Instructor: Prof. Enrique López Droguett, Ph.D.
Teacher Assistants: M. Fidansoy, G. San Martín, M. Pishahang, V. Vela.
Fall 2023 – UCLA
Alex Lie, Alexander Vera
905901892, 505994435

FINAL PROJECT: ANT COLONY SIMULATION

Introduction

Being one of nature's most peculiar natural systems, ant colonies behave in intriguing ways. Despite being extremely small organisms, ants have incredible systems of communication during scavenging due to their use of different kinds of pheromones, which can be used as complex trails leading to a food source and back to the colony. In this final MATLAB project, students will utilize the skills they've developed in function design, control flow, and optimization to develop a simulation that closely depicts the behavior ants have in nature.

Model and Theory

Being a final project, not too much was initially provided to students in the hopes that they would explore different ways to optimize their program.

One of the few things provided included a set of five "skeleton" functions consisting of *CheckFoodProximity* which checks whether an ant can pick up food in its near vicinity, *CheckColonyProximity* which checks whether an ant is close enough to drop off food at the colony, *MovementValidationExecution* which determines whether an ant's next intended step is valid, *PheromonesUpdate* which keeps track of the ants' pheromones as they decay, and *ComputeNewAngle* which determines the angle at which an ant will be facing for its next step. The methodology of these functions will be described in more detail in later sections. Aside from these skeleton functions, students were provided with a few diagrams to help build intuition for this program. Figure 1 depicts an arbitrary map, consisting of the colony where ants will come out (depicted as the cyan circle) and the scattered food sources surrounding the area (depicted as magenta triangles). Figure 2 describes the ants' basic anatomy for this project, which consists of a facing angle (with respect to the positive x-axis), and a "smell area" describing how far an ant can detect objects (food, colony, and pheromones) in its area.

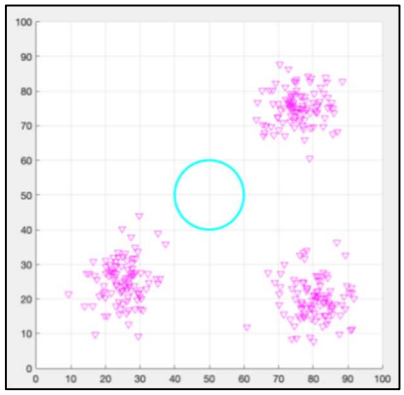

Figure 1. Arbitrary Simulation Map

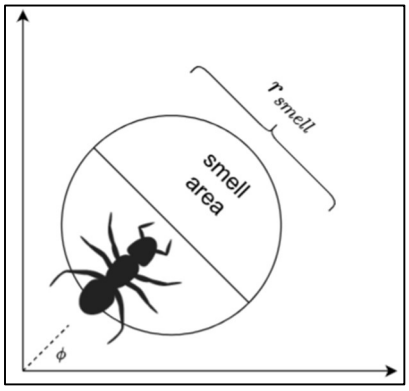

Figure 2. Simulated Ant Anatomy

Methodology

The main challenge of this project was getting multiple different parts working together in sync to mimic the behavior of an ant colony, so it was crucial to make sure that each function included in the program behaved properly.

To begin, *CheckFoodProximity* is crucial to the ants' ability to locate and pick up food around them. This function takes in as parameters the coordinates of the ant under consideration, a list of all available food sources on the map, and a threshold distance. With this information, the function calculates the distance between the ant and each of the food sources while keeping track of the minimum distance out of all the calculations. Once the minimum distance has been found, the function checks if it's less than the threshold distance, signifying that it's close enough for the ant to pick up. If the ant can pick up the closest food source, the food source list is updated with the removal of this food source, and an indicator is output signifying that the ant was successful in collecting food. If unsuccessful, the output indicator signifies the ant's failure.

The next function is *CheckColonyProximity*, which is quite straightforward and similar to *CheckFoodProximity*. The input parameters of this function include the coordinates of the ant under consideration, the coordinates of the colony, and a different distance threshold. Following a similar logic as was described for the previous function, *CheckColonyProximity* simply calculates the distance between the ant and the colony and then checks whether this distance is less than the given threshold. If the function finds that the ant is close enough to the colony, it outputs an indicator signaling the ant's ability to drop off food if it is currently carrying any. If not close enough, the function output indicates that the ant is too far.

Following this was the *MovementValidationExecution* function, which again is straightforward in its behavior. For parameters, it takes in the coordinates of the ant under consideration, its current facing angle, its movement speed, and the allowed boundaries determined by the rectangular map (note that there is also a parameter for boundaries defining forbidden areas on the map, but this is an extra credit option that will not be considered here). This function first uses trigonometric functions to calculate this ant's anticipated position based on its current position, facing angle, and movement speed. Once this anticipated position is obtained, the function then determines if it is within the given boundaries defined by the map. If it is, the ant's new position is output with no change to its facing angle. If out of bounds, the ant keeps its previous position but gets its angle changed by 180 degrees.

The *PheromonesUpdate* function is responsible for keeping track of the pheromones throughout the map that will dictate the ants' behavior. For inputs, this function takes in a list of all the existing pheromones on the map, a list of all the concentrations corresponding to each of these pheromones, and a vector containing the decay rate for both the blue pheromones that ants with food follow and red pheromones that scavenging ants follow. With this information, the function iterates through each existing pheromone and decreases its concentration by the decay rate corresponding to its color. After this has been done, the function filters out all the pheromones with concentrations of zero or less. Once the decayed pheromones and concentrations have been removed, the function returns the updated list of pheromones and concentrations as outputs.

The last function, *ComputeNewAngle*, is possibly the most complex out of all the previously mentioned functions as it dictates the future movements of the ant based on its environment. As parameters, this function takes in the position of the ant, its facing angle, lists of all existing pheromones and concentrations, the smell radius of the ant, and values of slight angle randomness for when an ant does (*sigma_1*) or does not (*sigma_2*) sense pheromones around it. The function begins by checking whether there are any pheromones in the map, and if none are detected, the ant's facing angle is only changed slightly based on the degree of randomness given by *sigma_2*. Once this default case has been checked, the function proceeds with calculating the distances and angles between the ant and each of the pheromones. After these calculations are complete, the pheromones are filtered based on their distance and angle difference. Only the pheromones within the ant's smell radius and angle of detection (in this case ± 90 degrees) are considered for further calculations. If no pheromones remain viable for calculations, the ant's facing angle is only changed slightly as dictated by *sigma_2*. The next step is calculating the "weighted" average of all the pheromones under consideration by the ant, where the weight is dependent on the concentration of the pheromones. Then, a new angle is computed using the arctangent of the average y and x positions previously obtained from the weighted calculations. With this newly calculated angle, a degree of randomness is added to it as dictated by *sigma_1* so that it can finally be returned as an output.

Aside from all these functions, there's also the main testing environment *Ant_Simulation* which puts all the previously mentioned parts together to create a simulation of an ant colony, where scavenging (foodless) ants will leave their colony, drop blue pheromones until they find food, at which point they'll begin dropping red pheromones until they reach the colony to drop off the food they've collected. This cycle of following red pheromones leading to food sources and blue pheromones leading to the colony continues until the simulation is stopped. The initial setup includes loading the environment map and initializing customizable parameters (which will be altered for optimization) such as the decay rate, smell radius, and angles of randomness that are used by the functions. A specified number of ants are initialized, each containing information about their position, facing angle, and food status stating whether they're carrying food or not. An initial pheromones matrix is also created, containing information about their position, color, and concentration. Once all of these have been initialized, the colony points (perimeter) and food sources are initialized, holding information regarding their position, color (food acts as red pheromones, and colony points act as blue pheromones), and concentration (which will be infinitely high). The last step for program initialization is creating the figure, video file, and simulation variables which include a counter for food in the colony, a time step, and speed.

For the main portion of this testing environment, the script opens a loop that iterates over *T* timestamps. This loop begins by separating the blue and red pheromones from the original matrix containing all pheromones' information. The script then iterates over each ant to compute their new angles, check movement validity, and set new positions, all while making sure to pick up or drop off food is possible. Once this ant iteration is complete, the pheromones matrix is updated using the *PheromonesUpdate* function as well as through the addition of newly created pheromones dropped by each and at the start of each timestamp. With all of this done, all the pheromones (with proper opacity depending on their concentration), ants, and colony points are plotted. Finally, the loop's last step is to capture a frame to add to the simulation video and update the display with a slight pause. The loop eventually ends, causing the video to stop.

Calculations and Results

For Map 1, the ant colony was placed at the center with coordinates (50, 50) with the food sources being located northwest of this location. The ants were successful in collecting all 500 food sources and returning them to the colony by timestamp 442. Overall, this simulation was highly efficient and greatly demonstrated the behavior of this natural system in the real world. Figure 3 shows the simulation before any food was brought to the colony, Figure 4 shows the simulation in progress, and Figure 5 shows the exact time stamp at which all food had been successfully collected (note that Figure 5 looks slightly different visually—this was to obtain the exact frame at which the simulation was completed).

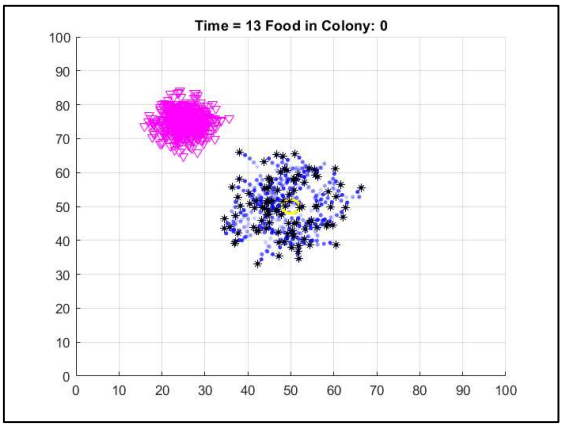

Figure 3. Map 1 Simulation - No Food

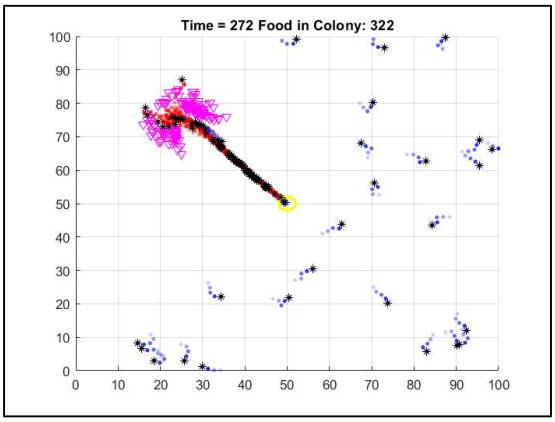

Figure 4. Map 1 Simulation - In Progress

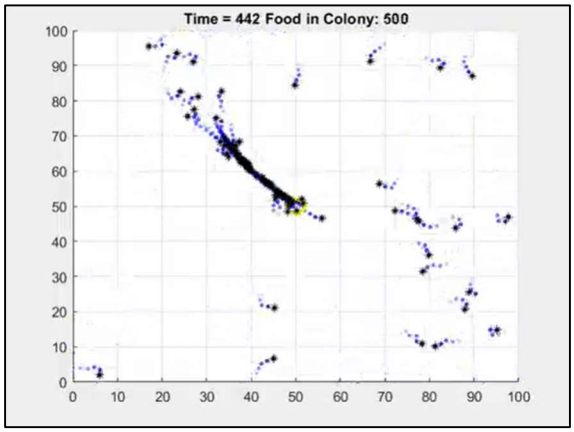

Figure 5. Map 1 Simulation – Complete

For Map 2, the ant colony was placed at the center of a larger environment with the coordinates (75, 75) and with the food sources being located north, east, south, and east of this location. This time, the ants were unsuccessful in collecting all food sources but still performed exceptionally well in obtaining 525 of them, which is most of the total. Figure 6 shows the simulation before any food was brought to the colony, Figure 7 shows the simulation in progress, and Figure 8 shows the simulation right before reaching maximum iterations.

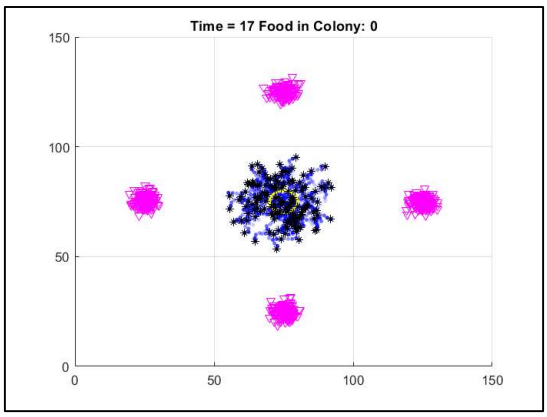

Figure 6. Map 2 Simulation - No Food

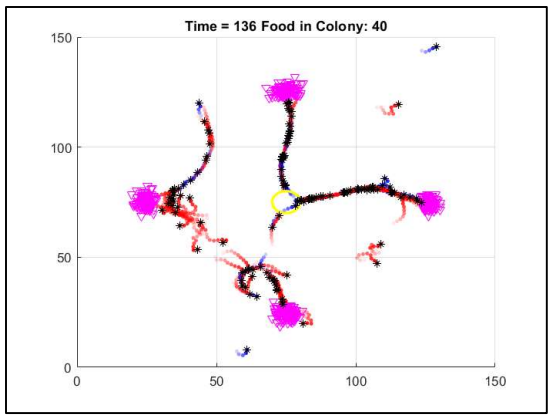

Figure 7. Map 2 Simulation - In Progress

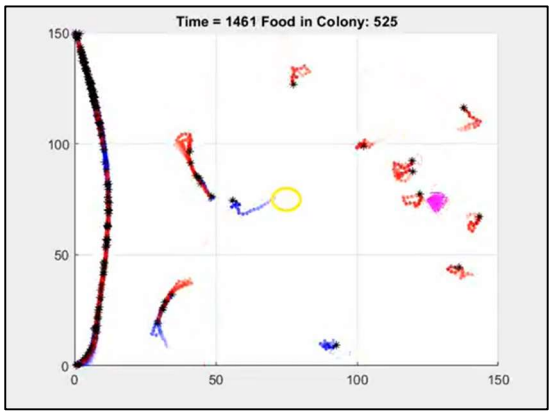

Figure 8. Map 2 Simulation – Maximum Iterations Nearly Reached

Discussion and Conclusion

For Map 1, the decay values *delta_blue* and *delta_red* were given values of 0.2 and 0.1 (which are quite large compared to Map 2 values) so that the pheromone paths would be smaller, ensuring that ants don't steer off from the main trail between the food sources and colony that would eventually be created. The parameter *sigma_phi_1* was given a value of 0.1, which was small compared to *sigma_phi_2* with a value of 1.0 because testing showed that it was best to encourage more random map exploration when no pheromones were detected while also making sure ants only focused on reaching their destination if relevant pheromones were detected. Finally, *r_smell* was given the maximum value allowable so that ants could have the largest possible area of detection. Map 2 was a little more challenging due to the different areas of the environment that needed to be explored. Because of this need for more exploration, *delta_blue* and *delta_red* were both given values of 0.05 so that pheromone trails would be larger, therefore giving ants a better chance of finding their required destination. The parameter *sigma_phi_1* was given a value of 0.5 to allow ants to find better pathing towards their destination while also making sure they don't lose focus of it, and *sigma_phi_2* was given a value of 1.5 to give ants a better chance of finding the food, which was more spread out this time. Again, *r_smell* was maximized to 10 for the same reasons previously stated.