

Semester Project OOAD

Project Summary

Individual: Dieu My Nguyen

Project title: Agent-based model of firefly mating strategy

High-level overview: To make use of the concepts from this class for my PhD research, I aim to build an agent-based model to simulate firefly mating behaviors using an OOAD framework. Briefly, an agent-based model is a collection of interacting agents, each being an encapsulated bundle of data and methods that allow the agents to act in an environment. Notably, the agents typically have limited behaviors, i.e. they have knowledge of a small spatial area around them, but not of the global environment. Yet, complex global patterns emerge from the local interactions between individuals.

In the dark, during the process of firefly courtship and reproduction, males use flash signals as part of their courting ritual. A stationary female will need to perform some sort of tracking of different male flashes and select a mate, then respond to the chosen male's flash to signal her location. The environment contains dense number of male fireflies who are mating competitors. Further, to conserve energy, males likely do not send continuously signals and instead send a limited number of flashes to convey their location and trajectory. Then, how does a female firefly maintain contact with a male she is signaling in this noisy environment? What can male fireflies do to optimize their chances at being recognized and successfully tracked? These questions are the driving force behind my exploration of this problem via computational modeling (and later in my research, experimental observations). (The equivalent phenomenon in humans is the cocktail party problem, which describes how people can distinguish and identify one conversation when surrounded by many people.)

In my model, fireflies will be the individual agents divided into different types based on attributes and behaviors (i.e. female, male, and finer distinction of males with different types of flash patterns and movement strategies) and they exist in some discrete 2D environment (i.e. a grid). The 2-D flying motion of male fireflies will be modeled as a correlated random walk. The female will track an individual male using a "lookup table," a distribution of anticipated positions stored in memory, to evaluate which signal comes from the male she is attempting to track.

To accomplish & Expected system: For the scope of this course, I will keep the model minimal as long as core theoretical concepts are sufficiently modeled. My goal is to implement a working agent-based model with parameters a user can vary to observe various emergent properties of the whole system. Specifically, I would like to be able to make some preliminary observations about the optimal flashing and moving strategies male fireflies can employ to mate with the female. If time allows, I would like to also explore different strategies females can use for the challenging tracking task.



Project Requirements & Responsibilities

Functional capabilities:

- Model inputs: A set of model parameters from the user (e.g. simulation time, number of male fireflies, velocity/step size, random walk turning angle, resting interval between flashes)
- Model outputs: A (graphic if possible) simulation of the courting process, female's success/efficiency in tracking a single male of interest, numerical/text data (below)
- Data to store: Time data of each male's motion trajectory and flashing pattern, of female tracking and final decision

Constraints & Non-functional requirements:

- Python 3 code for easy sharing with computational biology community
- Reproducible parameter searches: Use a random seed throughout
- Maintainability and enhancement: Because this will be ongoing research, the model should be flexible for future changes, such as additional strategies each firefly type may employ.



Users and Tasks: Use Cases

Type of users, tasks they will accomplish with system:

1. Research Assistant: After building a working model, I will request assistance from an undergraduate mentee (or myself) to run parameter sweeps to heuristically explore the bounds and the optimal values of this biological-inspired model. The assistant will test various values and ranges and based on the model outputs, determine optimal values.
2. A Broad Audience (aka Browser, for lack of more creative name): I will share the finished model on Github with other researchers and a broad audience. The user can input various parameters to observe how the system changes. A set of default parameters and suggested ranges will be available. Rather than trying many parameters as the research assistant would for research purposes, this audience would simply run the model to reproduce certain parameter set results or to explore this biological phenomenon on a higher, curiosity-drive level.

The two types of users above also define the number of scenarios I will strive to make possible in the model for the scope of this class project. Below are the respective text use cases.

Use case 1:

- Name: Parameter Sweep/Research Scenario
- Brief Description: High level use case of a parameter sweep to explore bounds and optimal parameters of the model
- Actors: Research Assistant, Simulator (an abstract controller/observer of the model)
- Preconditions: To perform a Parameter Sweep, a set (Python dictionary) of parameters should be provided to the model's user interface.
- Basic Flow: Research Assistant will provide the Simulator with the parameters they wish to test and ask the Simulator to start the simulation. The Simulator will take in those given parameters, create an environment with fireflies, and start running the model to simulate the courtship event. The Simulator gives the output data of the model to the Research Assistant in the form of saving data files to a given local directory.
- Alternative Flows: N/A
- Exception Flow: If the parameters the Research Assistant provides are missing some values or not in the data type expected, the Simulator will send a message about the missing or incorrect values to the Assistant and request a new set of parameters.
- Post Conditions: The model needs not keep in memory the previous simulation, as data has been written to external files. Thus, the Simulator can clean up the experimental space (reset).

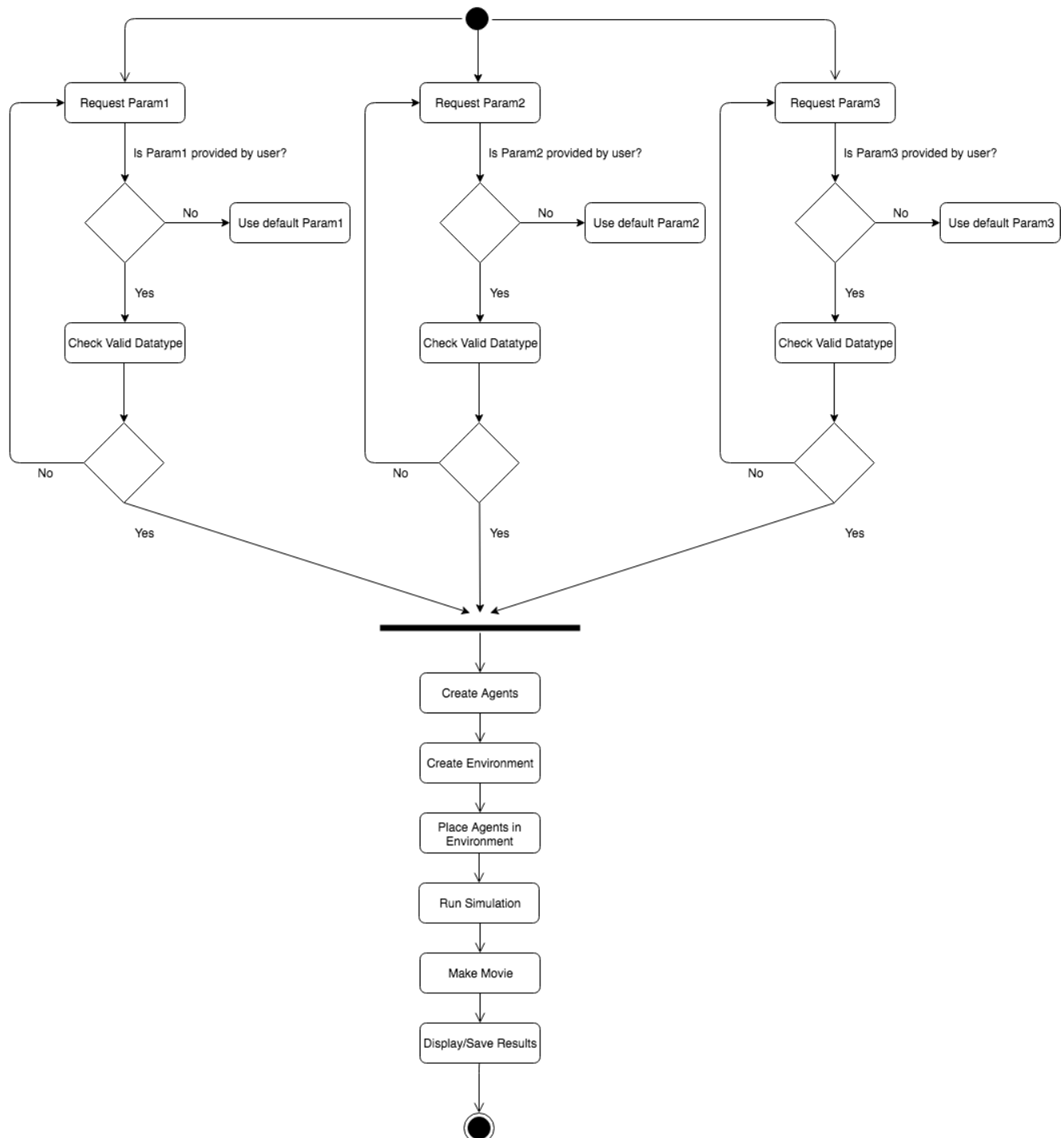
Use case 2:

- Name: Exploring the Model/Casual Use Scenario
- Brief Description: High level use case of a casual exploration of the model, for a broad audience type of user to quickly learn of the biological phenomenon
- Actors: Browser, Simulator
- Preconditions: This is for a casual user who would want to run the model once or twice to learn of the problem, so preconditions are only that optional parameters as input should be in a specified range (that perhaps we will have discovered via the Research Assistant's hard work).
- Basic Flow: The Simulator asks the Browser to optionally type in parameter values. If the Browser does not wish to provide a value for a parameter, the Simulator uses the default value. The Simulator then runs the model and creates a movie of flashing fireflies and shows the male firefly the female decides on. The Simulator also sends a message about the female's success in tracking her initially chosen male.
- Alternative Flows: As the Browser has the option to provide values of any number of parameters, we can have a mix of provided values, though this would not really change the flow of the Simulator's work except to fill in any values unprovided by the Browser.
- Exception Flow: If the data types of Browser-provided parameter values are incorrect, the Simulator will ask for new ones or ask the Browser if they want to use the defaults.
- Post Conditions: Same as Use case 1.



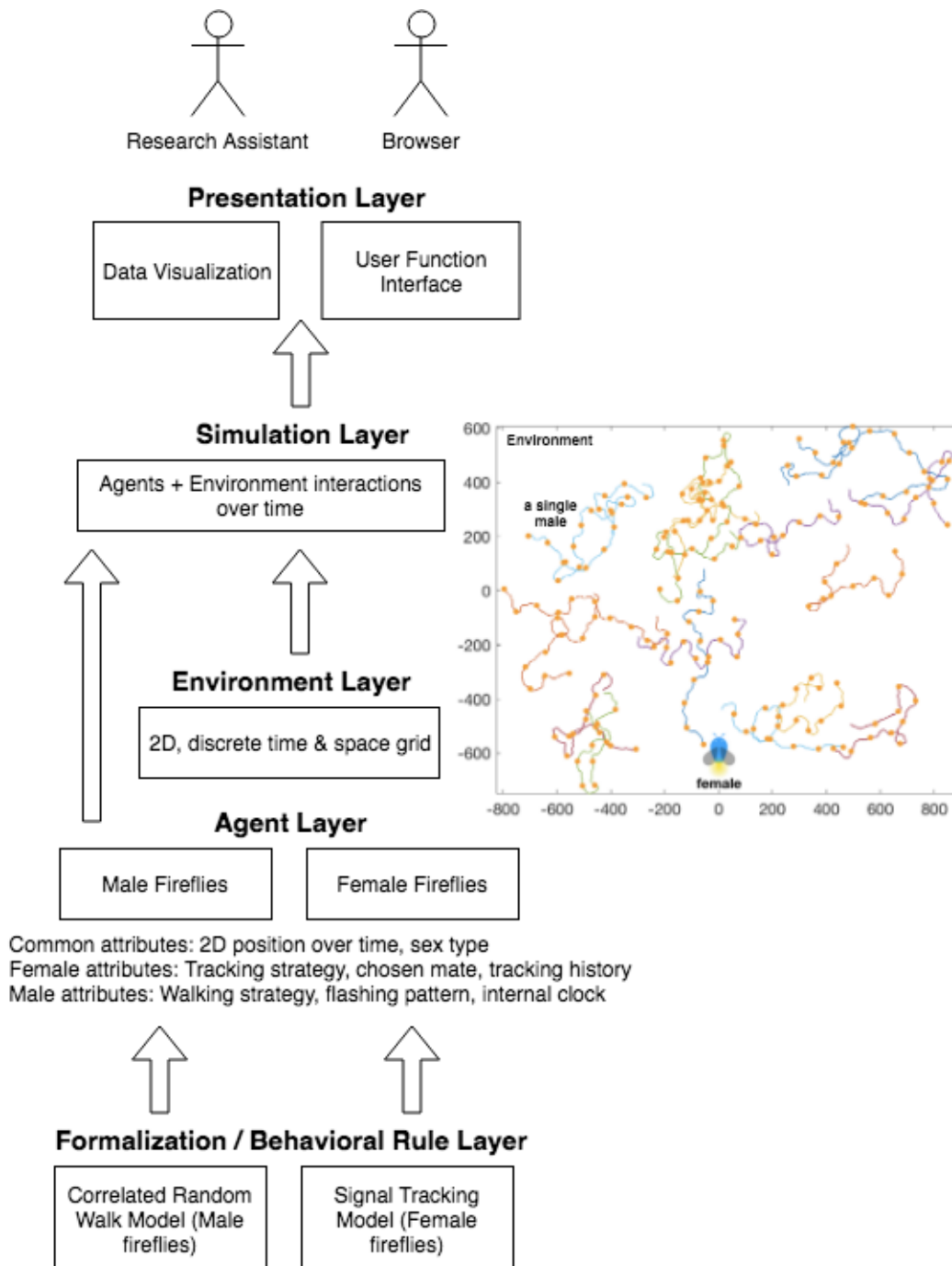
Activity Diagram

The Casual Use Scenario might be more involved with the optional input of parameters, so below is the activity diagram for User case 2. I have yet to decide on the final set of model parameters; this diagram documents an example case with 3 parameters.



Architecture Diagram

Attached to this diagram is an example environment with a single female firefly and many male fireflies, who form flying tracks (distinct lines) with inter-mitten flashes (orange dots). As we can see, with this dense population of competing males, it is quite difficult for the female to accurately track a single one.



Data Storage

How data will be persisted in this application: After a simulation of a particular set of parameters, the model will output text and numerical data above-described (probably will change slightly when I start to dive into the coding). This data will be persisted and updated during the simulation via common Python data structures, such as lists, dictionaries, data frames, or a combination of them.

Storage technology: To save the data, I plan to use storage technologies familiar to other computational biologists in my field for easy open access; those technologies include txt, JSON, and CSV file types.

Where the data will be stored: To keep a small scope for this project, I don't plan to build a web app to visualize my model unless time allows. Thus, data will be saved locally on the device on which the model is run.

Classes used to access data at run-time: My system will have the responsibility of *generating* data, especially for the research use case. I will consider building an additional data visualization system that uses the model output data, if time allows. With this, we could have users viewing simulations that have already run and saved, in form of a movie.




UI Mockups/Sketches

I'd love to create a Python version of NetLogo, but it probably can't happen in a month. Thus, for now, I aim to keep the UI aspect rather simple, while the primary focus will be to create a working ABM that will yield some interesting insights about fireflies.

I will develop most/all of my code in Jupyter Notebook. This means the user will be expected to run the model this way and they will see *some* code, rather than only seeing a graphic interface (until I can make a nice visualization app). There will be a notebook that acts as a UI, and notebooks with low level model implementations will be imported into that notebook so the user will not need to see the detailed code. Below are generalized screen mockups for all users (final version will probably be more nuanced). A user will input parameter values. The simulation will start. Then, data and a simulation movie will be outputted when the simulation is completed (or the user will be given a choice to save the data or not). Some built-in Jupyter Notebook interactive control packages to use: IPywidgets, cufflinks.

(This screenshot below is a cumulative of all the mockups for my system. Some comments are left in it for your information.)

jupyter Firefly Courtship ABM Main Last Checkpoint: 17 minutes ago (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run

Notebook description:

This notebook acts as the user interface to run the firefly courtship agent-based model.

```
In [5]: # Import statements for necessary packages
```

```
In [ ]: # The code here will import the other Jupyter notebooks
# containing the low level implementation of the model,
# allowing the user to not need to be exposed to those
# details and only interact with this notebook

import ipynb.fs.full.modules.Firefly as Firefly
import ipynb.fs.full.modules.Environment as Environment
import ipynb.fs.full.modules.Simulator as Simulator
import ipynb.fs.full.modules.Utils as Utils
```

Choosing model parameters

```
In [81]: # The ipywidgets package is used to allow users to select parameters
# Another way to consider is to have the user simply enter the values,
# but this requires the additional step of checking for valid types
# as depicted in the activity diagram

print("Please choose values for the following parameters:")
print("[Unchanged parameters will use default values.]")

# Concise code here to display the ipywidget sliders/buttons
```

Please choose values for the following parameters:
[Unchanged parameters will use default values.]

Time:

Num males:

Step size:

Flash interval:

Turning angle:

☐ Save data

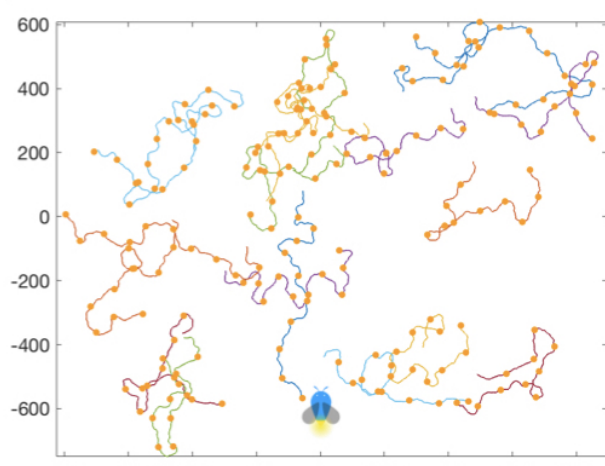
Simulation in progress

```
In [110]: # Code here to show progress bar
```

Loading:

Display simulation movie

```
In [111]: # Code here to show movie once simulation finishes running
```

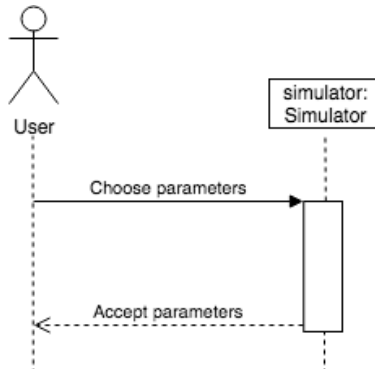


50

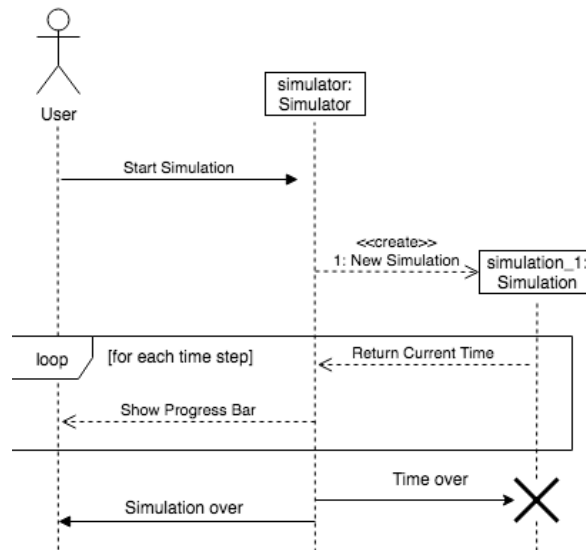
User Interactions

Below will be descriptions for a general user.

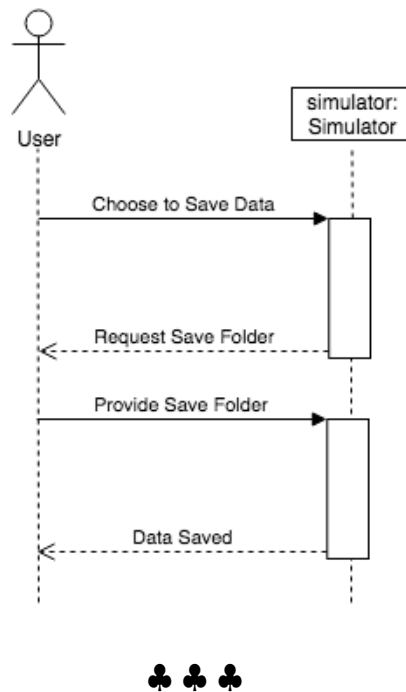
1. Choose model parameters: The system will request parameter inputs from the user in form of entering values at the prompt or using widget sliders as shown in the mockup. The Simulator (abstract controller of model) will save these values to run a new simulation.



2. Start and await simulation: The user can press a Start button. While the Simulator starts and runs the simulation, the user can see its progress with a visual progress bar.



3. View results: Once the simulation finishes running, the user can choose to save the data to a folder.



Class Diagram

I will attempt to keep it simple for our scope, and add more if time allows. Below is my initial class diagram. I expect it to change in the iterative process of implementation and further thinking abstractly about the problem.

