Matthew Schieber, mschieber3

CSE/ECE 6730 Project 1: Predator and Prey Populations With Variable Grass Species

Final Report

Introduction:

For this project, I have implemented an event-driven simulator for modeling a field of predators (wolves) and preys (rabbits). The customer of the simulation is a conservation agency that is concerned with which type of grass to plant a field with. Different grass grow at different rates and therefore sustain different populations of predators and preys. The customer will use my simulation in order to predict population sustainability and make a decision about which grass seed to plant the field with.

Implementation Details:

I implemented the event-driven simulator using the Python programming language. There are two main files. The first file is called `engine.py`, which contains the general machinery for the simulator and is agnostic to the application. Here, you will find the main loop of the engine, `RunSim()`, as well as event structures and the future event list (FEL) implementation. For event structures, I used a named tuple of the form:

```
Event = namedtuple('Event', 'timestamp event_name global_data event_data')
```

First, the timestamp was necessary in order to ensure that events are processed in timestamp order. The event name was necessary in order to emulate the pointer callback method in the original C implementation. Importantly, the `global_data` and `event_data` were crucial to pass around information necessary for each event as well as to avoid true write-read global variables.

For the FEL, I implemented a linear list. Since my simulation processes 100,000s of events, this structure yielded some slowdowns, so I also incorporated the `heapq` module in order to use a very fast tree-based priority queue. I included a toggle that would switch between the two implementations.

The second file in my simulator is `field.py`, which contains all data structures, event scheduling and processing functions. Note that all code in this file is specific to my simulation. At the top of this file, you will find all read-only global variables used in the simulation. This includes all model constants as well as the model parameter dictating grass species. I tuned the model parameters to my liking in the verification stage.

Next, I implemented a generic `Population` class which maintains both rabbit and wolf populations. This class keeps track of all unique organisms in a population and when initiated, it schedules all feeding events. Moreover, this class takes care of killing off organisms that have died as well as repopulations to emulate breeding.

After that, you will find all scheduling and event utilities. The major events include `RepopulateRabbits`, `RepopulateWolves`, `EatGrass`, `HuntRabbit`, and `GrowGrass`. I also included a null event called `RecordState`, which occurs in discrete intervals and logs the population numbers throughout the simulation. Moreover, there were also some miscellaneous utilities, such as a rerouting function to emulate pointer callback, normal sample, and update globals function.

Finally, the main loop of my simulation begins by initializing the wolf and rabbit populations. Then, a dictionary containing global data is created, all initial events are schedules, and the simulation begins with a call to `RunSim()`. Throughout the simulation, I use data collected in the `RecordState` to print to stdout the amount of grass in the field as well as the current number of rabbits and wolves. When the simulation ends after a prechosen number of days, the recorded population data can then be plotted using the MatPlotLib python module. There is a toggle at the top of the `field.py` file to turn the MatPlotLib functionality off or on. By default, it is off.

Verification:

To verify my simulation machinery, I confirmed efficacy of my FEL by printing timestamps and used the `RecordState` printouts to confirm each event was behaving correctly. Verification of the population behaviors is fairly intuitive. For instance, if the rabbit population is increasing, the wolf population should go up as well, and visa versa.

Validation - Sample Inputs/Outputs:

Here I present the efficacy of my event-driven simulator for modeling the behavior of rabbit and wolf populations. First, it is important to note that my simulator requires several different model constants, and tuning these constants was crucial for meaningful model performance. I compared some of my model constants with relative relationships and the resulting behaviors of my models with data from https://en.wikipedia.org/wiki/Lotka%E2%80%93Volterra_equations and https://en.wikipedia.org/wiki/Population_dynamics. The behaviors of my simulation are consistent with what is found in these sources.

Model constant tuning was fairly intuitive. For example, if I allow the rabbits to reproduce at too fast of a rate, something like this arises:

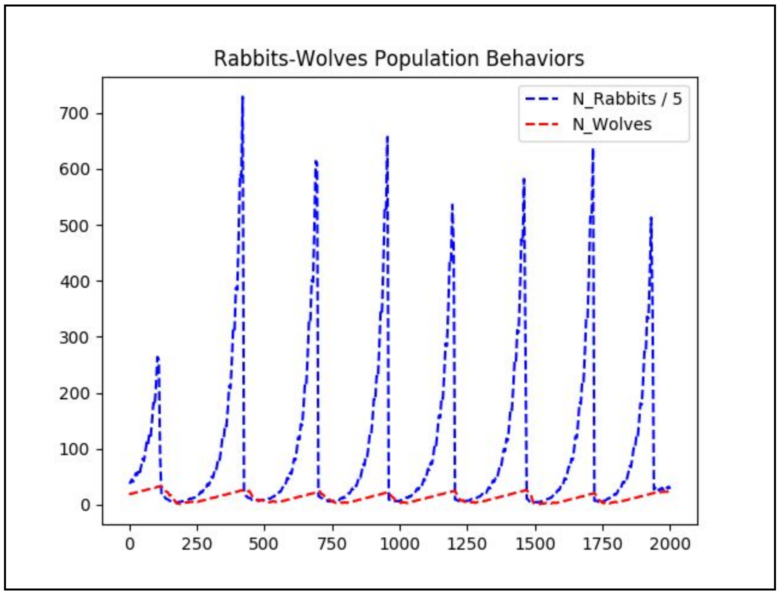

Figure 1: Model behavior when rabbits can reproduce too quickly.

While these behaviors might be realistic, I do not think it is a good idea to allow the rabbit population to spike this dramatically. Moreover, if the wolves are too good at hunting the rabbits (WolfCatchingRabbitRate), then the simulation will result in something like this:

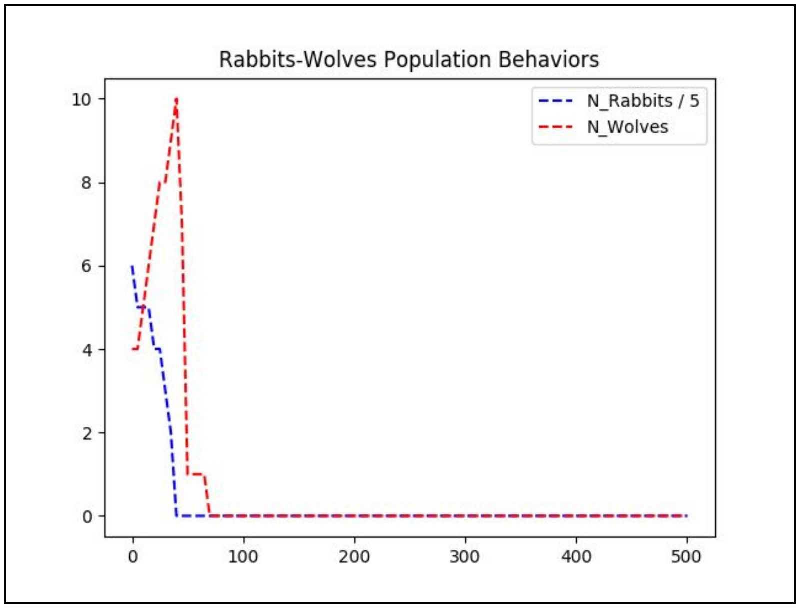

Figure 2: Model behavior when wolves can catch rabbits too efficiently.

Here, the wolves are too good of hunters, so the wolf population quickly spikes and overhunts the rabbit population. With that, here are the model constants that I found to be meaningful:

Table 1: Example of stable and good model constants.

InitialRabbitCount	30
InitialWolfCount	4
R_Starvation	2
W_Starvation	6
RabbitBreedingRate	0.10
WolfBreedingRate	0.02
R_Repopulation	6
RabbitsToHunt	0.10
WolfCatchingRabbitRate	0.25
SimulationLength	2000
FieldSize	250

Sample Inputs and Outputs:

The purpose of my simulator is to reveal the population behaviors when different grass species are sued. The model parameters are tuples, were the first element dictates how many units the grass grows, and the second elements is how often the grass grows. Note that this applies to the entire field of grass, so that every

G[1] days, N_Grass is incremented by G[0]*FieldSize. As a sample simulation, I will use the following model parameters:

Table 2: Example model parameters.

Model Parameter, G
[1, 2]
[1, 4]
[1, 6]
[1, 8]

Moreover, I used the same model constants as shown in Table 1. Here are the results:

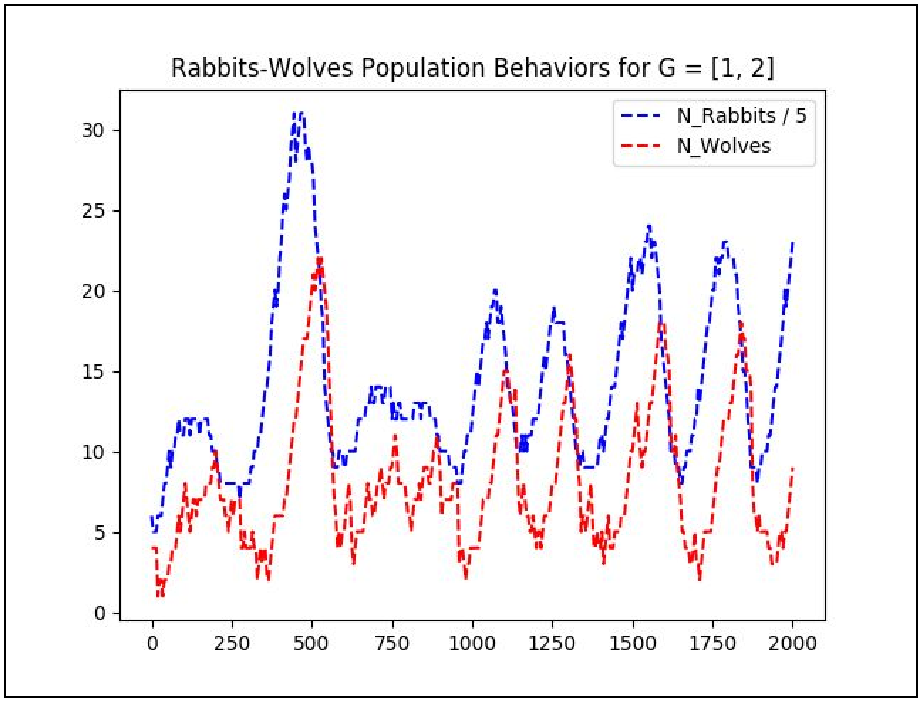

Figure 3: Model behavior when G = [1, 2].

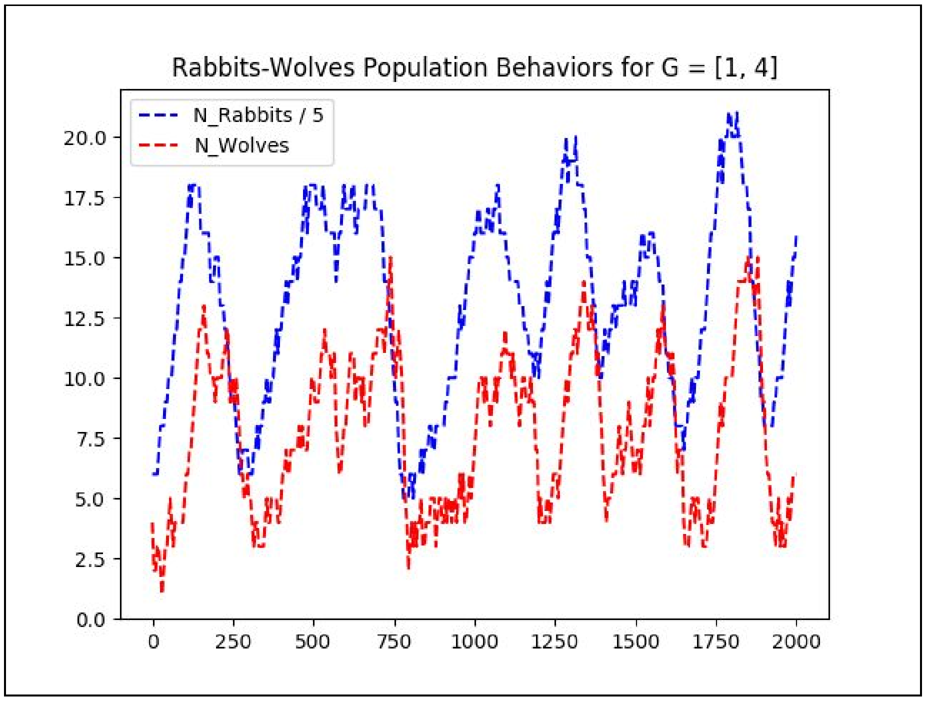

Figure 4: Model behavior when G = [1, 4].

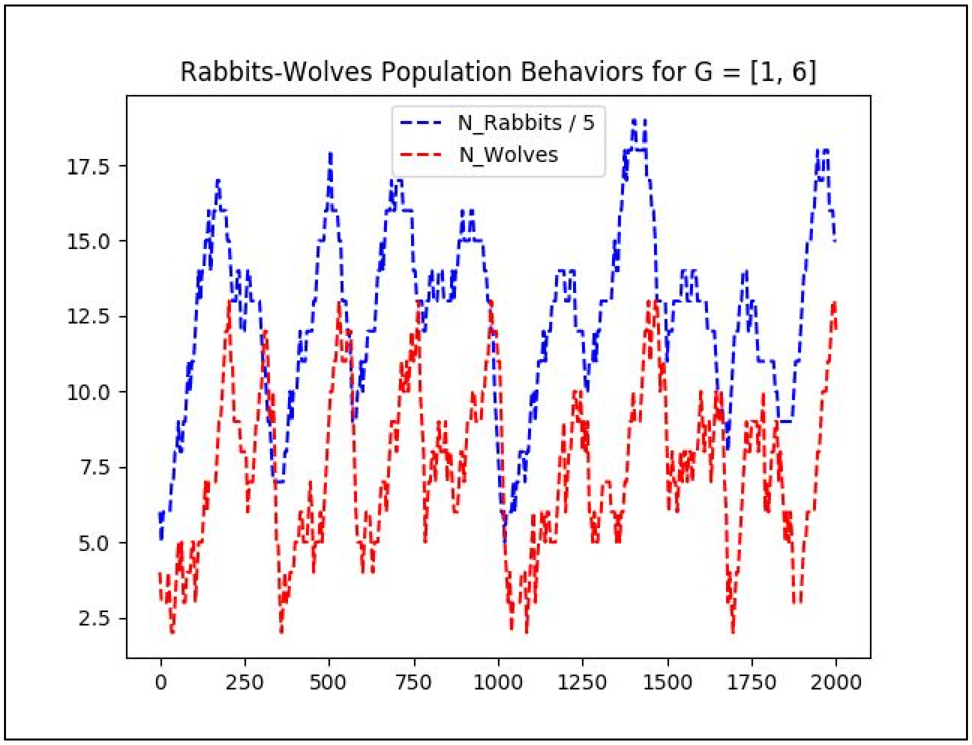

Figure 5: Model behavior when G = [1, 6].

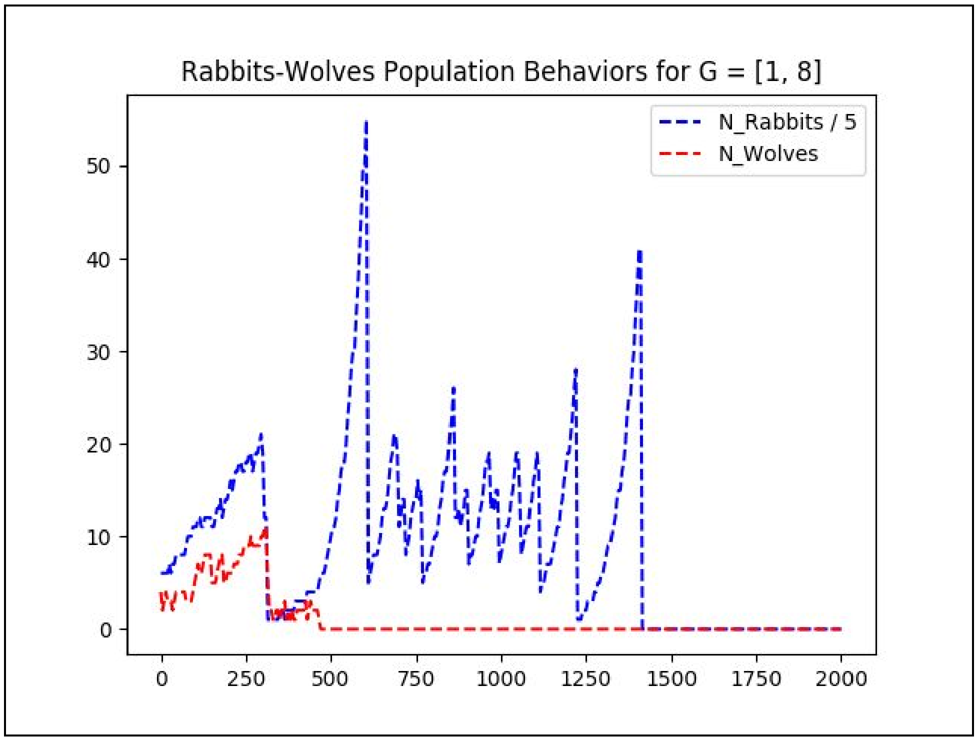

Figure 6: Model behavior when G = [1, 8].

Discussion:

These results validate the efficacy of my simulator. As the grass grows more and more slowly from G = [1, 2] to G = [1, 4] to G = [1, 6], one can see the population behaviors are similar, however, the average amount of organisms in the field systematically decreases. For the customer of this simulation, this means that purchasing grass seed that grows faster will allow them to sustain higher population numbers, on average. In the last case, G = [1, 8], the grass grows too slow, which results in both the rabbit and wolf populations going extinct. The customer of the simulation should be advised to not purchase this grass seed.

Sources:

1. Wikipedia contributors. (2018, March 3). Lotka–Volterra equations. In *Wikipedia, The Free Encyclopedia*. Retrieved 03:37, March 5, 2018, from https://en.wikipedia.org/w/index.php?title=Lotka%E2%80%93Volterra_equations&oldid=828624027
2. Wikipedia contributors. (2018, January 31). Population dynamics. In *Wikipedia, The Free Encyclopedia*. Retrieved 03:38, March 5, 2018, from https://en.wikipedia.org/w/index.php?title=Population_dynamics&oldid=823249272