Wolf Sheep Predation: Reimplementing a Predator-Prey Ecosystem Model as an Instructional Exercise in Agent-Based Modeling

E. Michael Bearss, Walter Alan Cantrell, Christian W. Hall, Joy E. Pinckard, and *Mikel D. Petty*

University of Alabama in Huntsville
Computer Science Department
emb0034@uah.edu, wac0010@uah.edu, ch0164@uah.edu, jep0037@uah.edu, pettym@uah.edu
*Corresponding author

Keywords: Agent-based modeling, experiential learning, Mesa, population dynamics

Abstract

The NetLogo Wolf Sheep Predation model is well suited for instructional use due to its use of familiar agents, concise source code, and simple graphics. As part of graduate course surveying modeling and simulation methods, a team of four students reimplemented the Wolf Sheep Predation model, treating the NetLogo version as the simuland, or system their model should simulate. The reimplementation used the Python programming language and the Mesa framework. Mesa is a framework for agent-based modeling written in Python, with features that are roughly equivalent to NetLogo. This paper documents their implementation process, compares the reimplemented Python/Mesa version to the original NetLogo version, and reports the students' and instructor's assessments of Wolf Sheep Predation as an instructional exercise.

1 Introduction

2 Background

2.1 Agent-Based Modeling

The agent-based modeling methodology has matured and its applications expanded in recent years [Collins, 2015] and software tools for agent-based modeling are available [Wilensky, 2015]. Agent-based modeling models systems that are composed of multiple distinct agents that interact with each other and their environment. Agents sense the environment, adapt their behaviors to the environment, and may alter their environment. According to [Macal, 2010], agents are modular and autonomous entities that have a state that varies with time and that interact with other agents in manners that affect behavior. Agents' behaviors may result in changes to the environment which, in turn, may influence the later behaviors of other agents. Simulations using agent-based models may reveal unanticipated relationships and configurations. Modeling the agents as independent units facilitates ar-

rangements that emerge from the autonomous decisions and behaviors of those agents.

2.2 Population Dynamics

When modeling population dynamics, differential equations can be used to approximate the behavior of species with large populations. Using one or more equations, we can produce an analytical solution to the population model that is easy to compute and will provide good accuracy if the population is sufficiently large. Using an analytical solution, however, introduces a few obvious issues. First, to use a system of differential equations, real numbers must be used. It becomes problematic to have fractional population members. These analytic solutions also disregard any biological constraints, such as requiring two of a species to reproduce. If these issues can be tolerated, it becomes very simple to model a population. To model a single population without any constraining factors, we can use the Malthusian growth model. Using a constant growth and death rate, we can use the following equation to predict the population at time t using an initial population P_0 and a growth rate of r.

$$P(t) = P_0 \cdot e^{rt} \tag{1}$$

In any realistic environment, 1 will not work, as the population will continue to grow exponentially with time. To address the issue of limited resources we can use a logistic growth model. Here we will introduce a saturation point that the model will eventually converge. Here we choose $a \gg b > 0$, this causes the rate of change of p to become 0 when $p = a/b$ or when $p = 0$. [Bungartz 2014]

$$p(t) = \frac{a \cdot p_0}{b \cdot p_0 + (a - b \cdot p_0) \cdot e^{-at}} \tag{2}$$

Systems of differential equations can also be used to model populations of two or more species. With populations of two species, there can be various relationships. The species can either be in competition for the same resource, exist as a predator-prey relationship, or non-interactive. Non-interacting species are typically not interesting since they can be modeled independently. The other two relationships will experience different behaviors. We will focus on the predator-prey relationship since it is the focus of our experiment. To model two species populations, we can choose values for a_1, a_2, b_1, b_2, c_1, and c_2 such that:

$$\frac{b_1}{c_2} > \frac{a_1}{a_2} > \frac{c_1}{b_2} \tag{3}$$

For these values, the population of the predator species $\overline{p}$ and prey species $\overline{q}$ will converge to the following stable values. [Bungartz 2014]

$$\overline{p} = \frac{a_1 b_2 - c_1 a_2}{b_1 b_2 - c_1 c_2},\ \overline{q} = \frac{b_1 a_2 - a_1 c_2}{b_1 b_2 - c_1 c_2} \tag{4}$$

When modeling predator-prey relationships, we can choose $b_1 = b_2 = 0$ corresponding to no restriction to the population of one's own species. This will cause an oscillating pattern to occur. As the prey's species becomes large, the predator's population will also grow. This will then cause the prey's species to quickly die off, followed by the predator's, as resources become constrained.

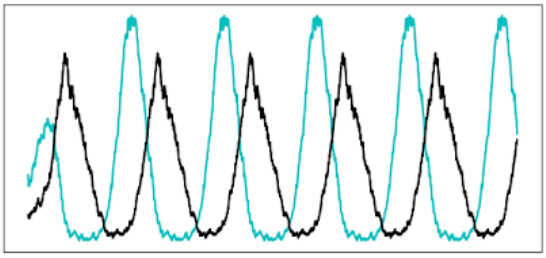

Figure 1: *Model of a predator-prey relationship demonstrating the oscillating behavior. As the prey species (blue) increases, it triggers an increase in the predator (black) species. This will cause the prey to die off, followed by the predator. This cycle repeats indefinitely in the continuous case.*

In 1 the population will continue to oscillate forever. As noted earlier, however, this is only true in the continuous case. When used to model a discrete population, the predator-prey case is often unstable with only two species.

Modeling discrete populations becomes significantly more difficult as birth and death rates produce integer changes in the population and must be modeled as random events. Rather than determining the value of the population directly, we will model the probability of the population being of specific size.

If we assume the growth and death rates are constant we can derive the following equation:

$$\pi_x(t+\delta t) = \pi_x(t) - \gamma x \delta \pi_x(t) + \gamma(x-1)\delta t \pi_{x-1}(t) \quad (5)$$

with γ representing the combination of birth and death rates, and δt representing a tiny step in time that approaches 0. With this we can use an infinite series of equations to model a discrete population. By choosing a small fixed value of δt we can then approximate the population at time t numerically. [Bungartz 2014]

Modeling discrete populations, especially ones with multiple species, becomes extremely difficult. In these cases it is often helpful to use an agent-based simulation to find a possible solution to the desired problem.

2.3 NetLogo and the Wolf Sheep Predation model

NetLogo's Wolf Sheep Predation model is a highly abstracted simulation of a predator-prey ecosystem consisting only of wolves, sheep, and grass. The populations of these three species are interrelated, as wolves consume sheep, sheep consume grass, and grass is depleted and restored over time. In this model, shown in 2 animals have a chance to reproduce asexually each time-step based on the animal's reproductive rate. When an animal reproduces, its energy is split between the parent and the child. When an animal's energy is depleted, it perishes and is removed from the simulation.

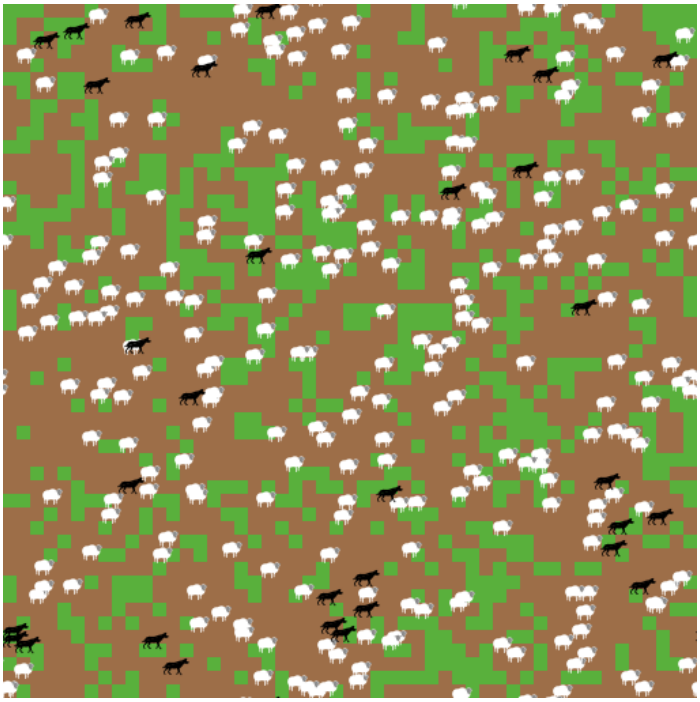

Figure 2: *A screenshot of NetLogo's Wolf Sheep Predation model [Wilensky 1997].*

The simulation terrain is a square grid of green or brown squares, each of which represents a patch of grass in one of two states: an incomplete growth cycle, or a completed growth cycle. Sheep can only eat a patch of grass if it is in the green state. After being consumed, the patch becomes brown, and then a certain amount of time passes before it finishes its growth cycle and becomes green again. Wolves and sheep follow random paths on the grid and do not seek out food or run from predators. When a wolf encounters a sheep, it will eat that sheep and gain some energy. When a sheep eats grass, the sheep gains some energy, and the grass restarts its growth cycle. Each turn, all wolves and sheep lose one unit of energy.

3 Implementation

3.1 Wolf Sheep Predation model as simuland

The simuland was studied through observation of the simuland's code, which was written in the NetLogo programming language. Unlike Python, NetLogo is procedural, so we reevaluated the logic from an object-oriented perspective. We scrutinized all the methods and data in the simuland and found that it could be classified based on the agent it was associated with: (a wolf, a sheep, or a patch of grass or dirt). We then documented the simuland's logic, organizing information by its associated agent.

3.2 Python and Mesa

Our team decided to use the Python programming language to implement our model of the NetLogo simulation for three main reasons: 1) all team members were familiar with Python already; 2) Python has powerful third-party libraries such as numpy, pandas, and arcade; and 3) it is an object-oriented language.

Soon after deciding upon a programming language for the model, we discovered the Mesa framework, of which had many desirable features which replicate our simuland. Mesa defines base classes for the model and agent, and has many built-in components which make agent-based modeling easier [Kazil, 2020]. One component which is crucial in our model is the agent scheduler; our model uses the RandomActivation scheduler, which is roughly equivalent to NetLogo's ask procedure which reads in each agent in an agentset in a random order. [Wilensky 1997] Like the NetLogo implementation, one scheduler is instantiated for each of the three

agent types (wolves, sheep, and patches). Another critical component is the grid, which is needed to represent the positions of each agent over the course of the simulation; for this, our model uses a single MultiGrid object, which has useful methods for placing, moving, and removing agents. ContinuousSpace is another spatial component which was initially considered, as wolves and sheep actually move with floating-point coordinates; however, complications arose since sheep and wolves are supposed to eat whatever resources are available on their current patch (which are represented by discrete locations) – so the former spatial component was chosen instead.

One of the most important aspects of Mesa lies in its powerful built-in visualization tools, which allow for real-time visualization and data analysis of the model using a browser-based interface. [Project Mesa Team 2016] This was particularly useful when debugging any errors which existed in the implementation. Two visualization modules used within our model are the CanvasGrid to visualize the individual agents, as well as the ChartModule to visualize their populations over time as a line graph. Mesa also includes UI features such as sliders to allow for easy parameterization of the model's initial state.

Figure 3: *Real-time visualization of Wolf Sheep Predation model using Mesa framework.*

3.3 Reimplemented Model

The terrain is modeled as a 51 by 51 grid of 15 square pixels. A pixel is green when grass exists and brown when sheep have eaten the grass. The wolves and sheep move about on the grid exploring the environment to find food and maintain their health.

3 depicts a real-time visualization of the reimplemented model. White circles represent sheep, while black squares represent wolves. Green and brown squares represent grass and dirt patches, respectively. Additionally, newly reproduced wolves and sheep are colored blue, while wolves and sheep that are on the verge of starvation are colored red. The graph located beneath the grid shows the popu-

lation of wolves, sheep, and grass patches over each step or tick. The "Frames Per Second" slider at the top adjusts the tick rate of the model, while the sliders on the left allow for customization of initial parameters. Finally, the buttons at top right allow for the user to start and stop the model, proceed to the next step, and reset the model.

Code Excerpt 1 shows the Python implementation of the constructor for the Wolf-Sheep-Grass model. Line 1 gives the class definition of WolfSheepGrass, which inherits from the base class as defined in mesa.Model. Line 3 defines its constructor and its parameters. width and height define the dimensions of the grid-based world. grass_regrowth_rate is an integer between 0 and 100 (inclusive) which determines when dirt grows to grass. initial_wolves and initial_sheep are integers between 0 and 250 (inclusive) to spawn the initial number of animals in the simulation. wolf_food_gain and sheep_food_gain are integers which determine how much energy the animal gains from eating food; possible values range from 0–100 for the former and 0–50 for the latter. wolf_reproduce and sheep_reproduce determine the percent chance of the respective animal spawning a child of its type, ranging from 0% to 20%. Finally, max_sheep defines the maximum number of sheep allowed; if no wolves are alive, and the count of sheep exceeds this number, the simulation ends.

Line 5 calls the constructor of the Model class (which WolfSheepGrass derives from). Line 7–8 saves the width, height, and maximum number of sheep allowed in the simulation for later use. Line 10 and lines 12–14 instantiate the MultiGrid and RandomActivation objects as discussed earlier, respectively.

Lines 16–23 initialize sheep_schedule and the shared grid with the initial number of Sheep objects at random xy-coordinates; lines 25–32 follow similarly, initializing Wolf objects to the wolf_schedule and adding them to the same grid. The Wolf and Sheep classes only differ in their eating behaviors, so in the implementation, they derive from an Animal class (which itself derives from the mesa.Agent class). Each animal is initialized with a random direction between 0 and 360 degrees, as well as a random amount of energy between 0 and 2 times that animal's food gain.

Finally, lines 34–40 initialize the environment by creating Patch objects to patch_schedule and to the shared grid. Each Patch object has a 50% probability to be generated as grass upon instantiation (likewise, it has an equal probability to be generated as dirt).

Our model comes close to replicating the NetLogo model, but it is different in one notable way: NetLogo uses turtles to move the wolves and sheep around the environment. It does so by rotating the turtle between 0 and 50 degrees once towards the right, then again towards the left, then finally moving the turtle forward by one step. Our model does not utilize turtles; while a turtle library exists for Python, it was preferable instead to use trigonometric functions for calculating movement by assigning a direction and xy-coordinates to wolves and sheep. However, this ultimately should achieve the same result with regard to an agent's movement.

3.4 Visualization and Validation

4 Instructional Assessment

4.1 Students' assessments

4.2 Instructor's assessment

5 Conclusion

[1]

2

References

[1] Jackie Kazil, David Masad, and Andrew Crooks. Utilizing python for agent-based modeling: The mesa framework. In Robert Thomson, Halil Bisgin, Christopher Dancy, Ayaz Hyder, and Muhammad Hussain, editors, *Social, Cultural, and Behavioral Modeling*, pages 308–317, Cham, 2020. Springer International Publishing.