Spora: A Journal of Biomathematics

Volume 10 Article 6

2024

Agent-Based Modeling for the Neophyte: An Application of NetLogo

Olcay Akman
Illinois State University, oakman@ilstu.edu

Siddharth Bhumpelli
Normal Community High School, Bhumpellis@gmail.com

Christopher Hay-Jahans
University of Alaska Southeast, cnhayjahans@alaska.edu

Recommended Citation

Akman, Olcay; Bhumpelli, Siddharth; and Hay-Jahans, Christopher (2024) "Agent-Based Modeling for the Neophyte: An Application of NetLogo," *Spora: A Journal of Biomathematics*: Vol. 10, 37–49.
DOI: https://doi.org/10.61403/2473-5493.1095
Available at: https://ir.library.illinoisstate.edu/spora/vol10/iss1/6

Intercollegiate Biomathematics Alliance

SPORA A Journal of Biomathematics

ISSN 2473-3067 (Print)
ISSN 2473-5493 (Online)

Agent-Based Modeling for the Neophyte: An Application of NetLogo

Olcay Akman[1,*], Siddharth Bhumpelli[2], Christopher Hay-Jahans[3]

*Correspondence:
Olcay Akman,
Center for Collaborative Studies in Biomathematics,
Campus Box 4520
Illinois State University
Normal, IL 61790-4520, USA
oakman@ilstu.edu

Abstract
Agent-based modeling has found applications in a wide range of fields including economics, sociology, ecology, epidemiology, transportation planning, and more. Its versatility allows researchers to investigate various "what-if" scenarios, test the effects of different policies or interventions, and gain insights into the underlying mechanisms driving complex systems. This article is intended for the curious student or researcher who is unfamiliar with agent-based modeling and is looking for a quick but reasonably informative exposure to the field.

Keywords: Individual-based model, predator-prey model, NetLogo

1 Introduction

It is evident from the literature that agent-based modeling (ABM) stands out as a powerful computational technique used across a wide range of disciplines to simulate complex systems comprised of autonomous agents. These agents, which could represent individuals, organizations, or even simple entities such as cells or molecules, operate within a defined environment and follow sets of rules governing their characteristics/attributes, behavior, and interactions. ABM offers a bottom-up approach to modeling, where emergent phenomena arise from the interactions of individual agents with each other and/or their environment rather than being explicitly programmed into the model.

The concept of agents in ABM draws inspiration from observations in social sciences, biology, and other fields where individual entities exhibit distinct behaviors that collectively shape the dynamics of the system as a whole. ABM provides a flexible framework for studying phenomena ranging from the behavior of financial markets to the spread of infectious diseases, ecological system dynamics, urban dynamics, and beyond.

In ABM, agents typically possess attributes, exhibit behaviors, and are subject to rules that govern their interactions with each other and the environment. These interactions often lead to the emergence of complex patterns, dynamics, and phenomena that would be difficult to predict solely from understanding the behavior of individual agents in isolation.

[1]Center for Collaborative Studies in Biomathematics, Illinois State University, Normal, IL, [2]Normal Community High School, Normal, IL, [3]Dept. of Natural Sciences, University of Alaska Southeast, Juneau, AK

One of the key strengths of ABM lies in its ability to capture heterogeneity, non-linearity, and feedback loops inherent in many real-world systems. By representing agents as autonomous entities with diverse characteristics and decision-making processes, ABM enables researchers to explore how micro-level interactions give rise to macro-level patterns and behaviors.

ABM has found applications in diverse fields including economics, sociology, ecology, epidemiology, transportation planning, and more. Its versatility allows researchers to investigate various "what-if" scenarios, test the effects of different policies or interventions, and gain insights into the underlying mechanisms driving complex systems.

This introductory exploration of ABM is intended for the curious student or researcher who is unfamiliar with ABM and is looking for a quick but reasonably informative exposure to the field. The exploration begins in Section 2 with a brief description of ABM and a simplistic view of the mechanics involved. This is followed in Section 3 by a preliminary overview of NetLogo, the preferred computational platform for ABM. After a NetLogo coding/application demonstration through two accessible examples in Sections 4 and 5, the interested reader is guided to resources leading to rigorous treatments of ABM and advanced applications of NetLogo in Section 6.

2 Agent-Based Models

While the title of this article refers to agent-based modeling, readers might also encounter the term individual-based modeling in the literature. It turns out that these terms are often used interchangeably and the protocol for designing and implementing such models is the same, see

[3] and [8]. In fact, the bibliographic analysis in [19] provides evidence in support of considering the terms IBM and ABM interchangeable. The subtle (maybe philosophical) difference between the focus of the two lies in the emphasis on decision making. So, in this article only the term agent-based model (or modeling) is used.

At this point consider a scenario in which a simple agent-based model could be used. In animal ecology the subjects of interest may be a collection of individual animals that may react to their environment through movement, through changes in their characteristics, or through changes in their behavior patterns.

Take, for example, the foxes and rabbits predator-prey model described in [1]. In the absence of all other factors, consider following the movement of foxes in response to the availability of one of their prey species, rabbits, and tracking the population sizes of the two species. If there are an adequate number of available rabbits, the foxes are not likely to move outside of their respective neighborhoods. However, if the food source for a particular fox becomes scarce, then it would be expected to explore further afield in search of rabbits. It is reasonable to hypothesize that the key determining factor as to whether a fox will survive and reproduce will be the presence or absence of rabbits. This scenario is modeled in Section 4.

In Section 5 the modeling scenario described above is enhanced a bit. Assuming that rabbits represent the only food source for the foxes, the food source for rabbits, say grass, is allowed to vary as well (be eaten and regrow). Thus, the rabbit population is made dependent on not only the presence of foxes, but also the availability of grass.

Of course, there is much more to a true foxes and rabbits system, and actually following foxes and rabbits in the wild to identify and record factors that influence their movement and/or characteristics of interest is at best difficult. Hence, there is still value in scientific hypotheses, probability driven computer simulations based on these hypotheses, and the comparison of simulation results with (potentially limited) field data. An analysis of how a collection of individuals (for example, foxes and rabbits) react to or are affected by their environment (food availability, competition, etc.) using computer simulations alongside field data serves as a simple application of agent-based modeling and its accompanying analyses.

3 NetLogo in a Nutshell

Developed by Uri Wilensky in 1999, NetLogo [21] is an open-source, multi-agent, programmable modeling environment that can be used to simulate natural and social phenomena as well as the dynamics of complex systems over time. The user-friendly interface of NetLogo makes it accessible to both novices and experienced users, and its flexibility makes it a versatile tool that can be used across disciplines. Its high-level programming language is specifically designed for simulating and modeling agent behaviors and interactions, and is easy to learn and implement. Another feature of NetLogo is the ease with which sliders can be used to observe immediate changes in simulation results. This enhances its value as a supporting tool for analysis and decision making, as well as making it an ideal teaching tool for demonstrating dynamical concepts to non-technical audiences.

To get started with NetLogo, visit the NetLogo website

`https://ccl.northwestern.edu/netlogo/`

for detailed instructions on downloading and installing the software. Also of particular use is the *NetLogo User Manual* and the *NetLogo Dictionary*, which contains the complete list of code syntax along with descriptions and examples [21].

3.1 Key features

NetLogo is particularly popular in the fields of complex systems, social sciences, biology, ecology, and the environmental sciences due to its ease of use, flexibility, and powerful modeling capabilities. Some key features that contribute to this popularity include the following:

Agent-Based Modeling Environment NetLogo provides a user-friendly interface for creating and simulating agent-based models. Users can define different types of agents, such as *turtles* (representing individual entities) and *patches* (representing discrete spatial locations), and specify rules governing their behavior and interactions.

Turtle Graphics NetLogo features a built-in graphical display where agents, represented as turtles, move and interact within a two-dimensional grid, or *world*. Users can customize the appearance of turtles, patches, and other elements of the simulation environment.

Programming Language The programming language and syntax used by NetLogo are easy to learn and implement, and are accessible to users with varying levels of programming experience. The language allows users to define procedures, create variables, and control the behavior of agents and the simulation environment.

Definition of Agent Behaviors Users can define the behaviors of individual agents. This includes specifying how agents move, interact with other agents, perceive their environment, and respond to changes in their surroundings.

Model Library NetLogo provides a rich library of prebuilt sample models covering a wide range of topics, including biology, ecology, economics, sociology, and more. These models serve as templates for users to explore, modify, and extend, making it easier to get started with building complex simulations.

Tools for Experimentation and Analysis NetLogo supports experimentation and analysis tools that allow users to run multiple simulations with varying parameter values, collect data, visualize results, and analyze the behavior of the agents and environment being simulated by model over time. This enables users to test hypotheses, explore different scenarios, and gain insights into the dynamics of complex systems.

Community and Resources NetLogo has a large and active community of users, developers, and educators who contribute to the development of the software and share resources, tutorials, and best practices. This vibrant community (accessible via the NetLogo website) provides support and collaboration opportunities for users interested in agent-based modeling.

In summary, NetLogo is a free and well-supported platform that provides a powerful and intuitive tool for building, simulating, and analyzing agent-based models, making it an invaluable tool for students, educators, researchers, and practitioners interested in studying complex systems and emergent phenomena.

Actually getting started with the process of conducting ABM with NetLogo for the first time is fairly straightforward once the purposes of certain key components are understood.

3.2 Key components

Three tabs and a button that appear in the opening screen of NetLogo are worth mentioning, see Figure 1.

Interface. The model interface window is where the simulation is displayed. Also, it is here that buttons, sliders, and switches that control parameter values and the model simulation are placed. Additionally, dynamic graphs and more that can be used to monitor quantitative output associated with the model simulation can also be placed here. The option to insert such features using the `Add` button and drop-down menu next to it is contained in this window, see Figure 1. The blank square region in the center of the screen, which will be referred to as the canvas, is where the results of the simulation itself are displayed.

Settings. A NetLogo canvas has certain default settings that suffice for most purposes, but may be changed as needed, see Figure 2. In the default setting the canvas, named the *world*, is a coordinate plane that consists of a 32×32 grid centered at the origin, $(0, 0)$. This grid is a planar representation of a torus which results in a useful feature: the *world wrap*. If an agent travels off the screen on one side, it will reappear on the opposite side. Another setting that is of importance is the *view updates* option, see Figure 1. If this is set at `continuous`, the model will continue updating in between *ticks* (or time steps). This takes up a lot of processing space and may slow down the visual displays of the model. If the setting is on `ticks`, the model's visual display will only update after each tick. For more complex models this leads to a more efficient performance, the downside being that the model's visual display will update in steps rather than continuously.

Info. The `Info` tab opens the documentation window for the model. Suggested headers along with brief descriptions of what to place under the headers are provided for model creators to fill in useful information about their model, about possible applications, and on possible extensions. If shared with the NetLogo user community on the NetLogo website, this provides potential users of the model with valuable information that enables them to build off of others' work efficiently and to contribute to the continually growing database of developed ABM models in NetLogo.

Code. Code for a NetLogo program is entered in the `Code` window. The programming language of NetLogo allows for the writing of clear modular code to perform desired tasks. As with other programming languages, *procedures* (or functions) in NetLogo are self-contained sets of instructions for tasks to be performed by the program. These are written in the `Code` window, and every procedure within the code begins with the keyword `to`, followed by the name and then the body of the procedure (the instructions). The keyword `end` is used to signal the end of a procedure. Coding examples for the previously described simple foxes and rabbits scenarios along with explanations appear in Sections 4 and 5. Code for these two examples are adapted from the "Wolf Sheep Predation" sample model provided in "Tutorial #1: Models" of the *NetLogo User Manual Version 6.4.0* [21].

4 Foxes and Rabbits

Consider the simple foxes and rabbits scenario in which rabbits are the sole source of food for foxes and foxes are

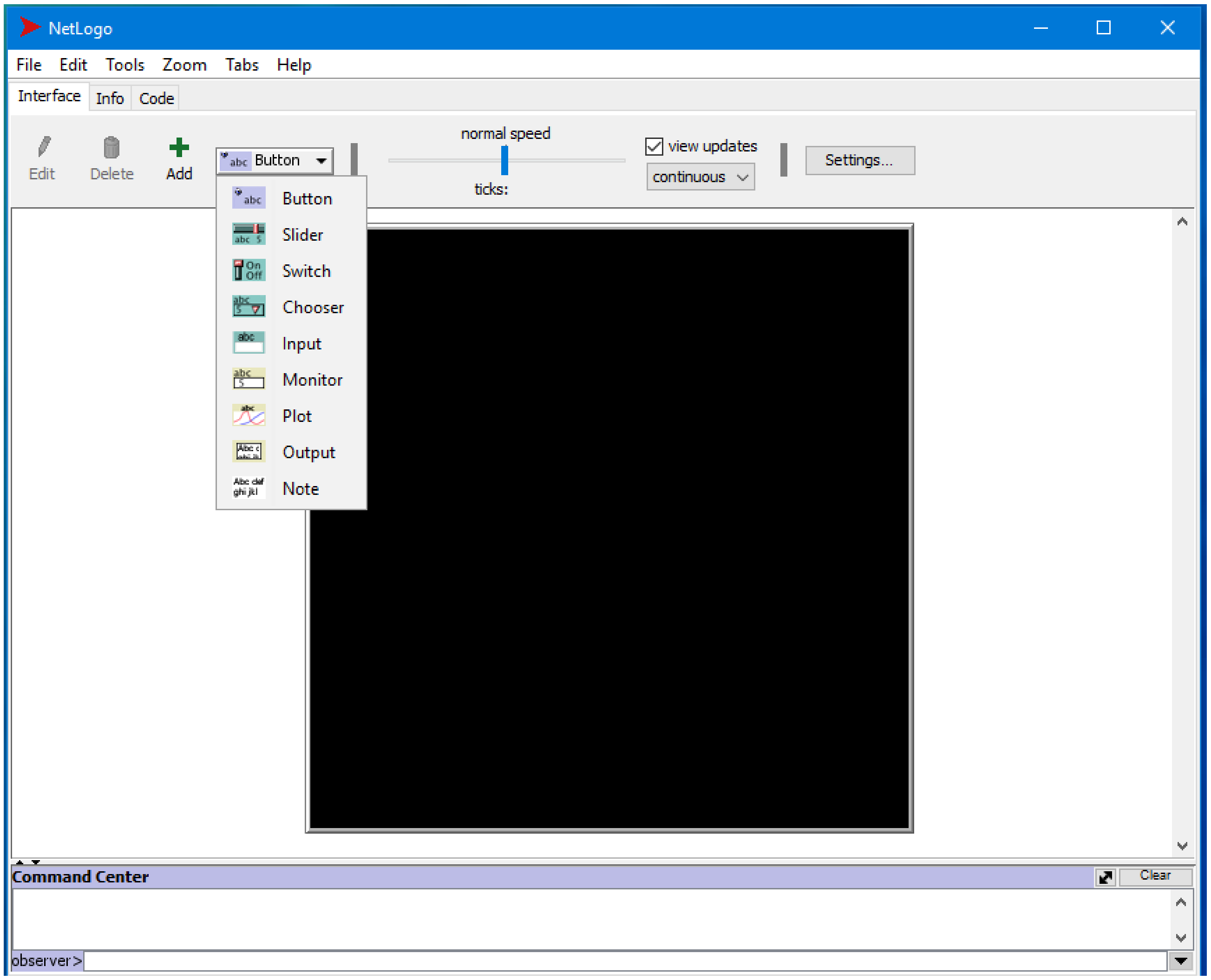


Figure 1: Opening view of the NetLogo window. The tabs of interest are `Interface` (the current view), `Info`, and `Code`. The drop-down menu next to the `Add` button will find use in this article, but the `Command Center` at the bottom of this window is better left for when users have achieved a suitable level of comfort with NetLogo. The `view updates` option is left on the `continuous` setting, and the `Settings...` button to the right of this opens the popup window shown in Figure 2.

the only predator that feed on rabbits. Also assume that the food source for rabbits is unlimited.

4.1 Preparing the canvas and the code

For this example the *world*, see Figure 2, is left in the default setting and *view updates*, see Figure 1, is set on `continuous`. The next task is to identify the central players in the program. This is done through the `Code` tab, with sliders providing initial values and other attributes as follows.

First, insert a slider for the initial rabbit population, see Figure 3. Then, just as is done for the rabbit population, additional sliders are inserted to set each of the following: the amount of energy gained by a rabbit from food (named, for example, `rabbit-gain-from-food` with values ranging from 0 to 10 with a starting value of 4); the reproduction rate for rabbits (`rabbit-reproduce`, with % units ranging from 0% to 100% with a starting value of 4%); the initial fox population (`initial-number-foxes`, ranging from 0 to 10 with a starting value of 4); the energy gained by a fox from food (`fox-gain-from-food`, ranging from 0 to 50 with a starting value of 15); and the reproduction rate for foxes (`fox-reproduce`, ranging from 0% to 100% with a starting value of 3%). The initial values from these sliders and certain global definitions are used by the `Setup` procedure which contains instructions on how to set up the starting canvas, see Figure 4.

The next task is to define several procedures that control the attributes, movement, interactions, and survival of the rabbits and foxes, see Figure 5. These are placed below the `Setup` procedure. Briefly, and in order of appearance, the `move` procedure causes agents to move one step in a randomly selected direction. The `reproduce-rabbits` procedure randomly models the reproduction of rabbits at a rate provided by the `rabbit-reproduce` slider. If the randomly generated number is less than the reproduction rate, the rabbit reproduces by hatching a new rabbit and then shares its energy with its offspring. Similarly, the `reproduce-foxes` procedure follows the same logic as `reproduce-rabbits`, governing the reproduction of foxes, with the reproductive rate being provided by the `fox-reproduce` slider.

Moving on to the predation part of the model, the `eat-rabbits` procedure simulates the predation behavior of foxes. If there is at least one rabbit in a fox's patch (`prey != nobody`), then one rabbit (`prey`) is selected randomly and is eaten, and the rabbit is asked to `die`. Then the fox gains energy according to the `fox-gain-from-food` slider. Finally, the `death` procedure checks the energy level of an agent. If an agent's energy is less than or equal to zero, it dies. In the current simple setting this mimics natural mortality of the foxes and rabbits due to starvation or other causes. These procedures are then called in the `Go` procedure to perform one iteration of a simulation of the model from start to finish, see Figure 6.

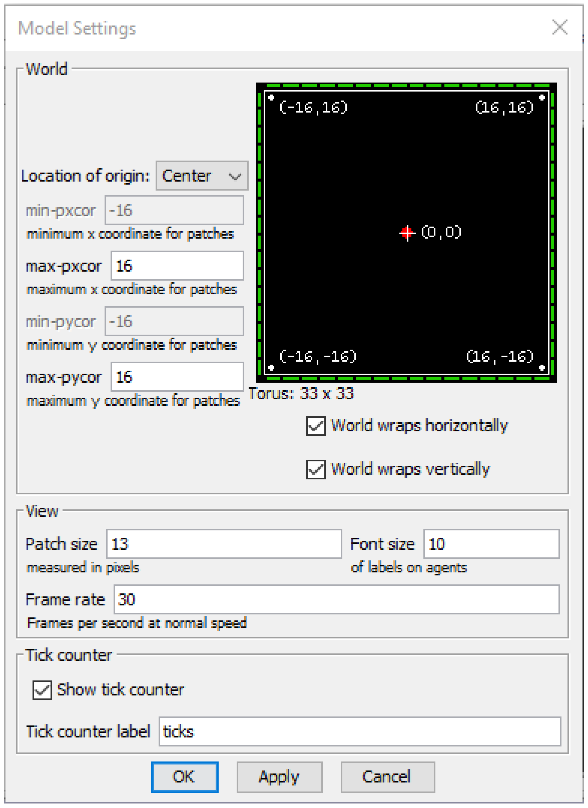
In a full simulation, the `Go` procedure is called repeatedly, each iteration representing one time-step. The procedure checks for the existence of foxes and rabbits, keeps track of their population sizes, and governs their movements from patch to patch, their reproduction, and their deaths. At the end of each iteration the time-step counter (`tick`) is advanced by one step.

With the `Setup` and `Go` procedures defined, buttons to run the code in these procedures are inserted in the interface window, see Figures 7 and 8.

A matter of interest in traditional predator-prey models is how the predator and prey populations vary with time. To insert a plot that monitors the sizes of these populations in the interface window, first click on the `Add` button; next select `Plot` from the drop-down menu; and finally click on a blank space to the left or right of the canvas to get the `Plot` popup window, see Figure 9.

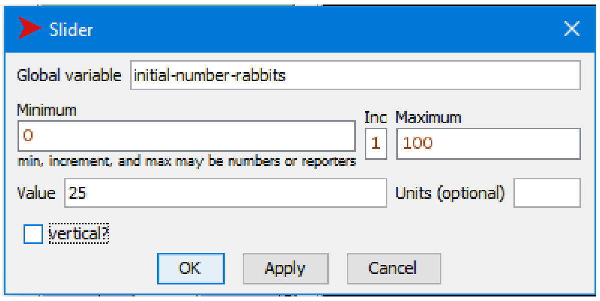
4.2 Coding side notes

As mentioned previously, an efficient and effective way to get started with coding in NetLogo is to take advantage of code in freely available and functioning NetLogo models that can be adapted to a different, but equivalent story. The `Check` tab in the `Code` window is very help-



Location of origin: Center
min-pxcor -16
minimum x coordinate for patches
max-pxcor 16
maximum x coordinate for patches
min-pycor -16
minimum y coordinate for patches
max-pycor 16
maximum y coordinate for patches
Torus: 33 x 33
☑ World wraps horizontally
☑ World wraps vertically
View
Patch size 13
measured in pixels
Font size 10
of labels on agents
Frame rate 30
Frames per second at normal speed
Tick counter
☑ Show tick counter
Tick counter label ticks

Figure 2: The `Model Settings` window shown here is where the settings are defined for the canvas on which the simulations are displayed. For most simple scenarios the displayed default settings suffice.



Global variable initial-number-rabbits
Minimum 0
Inc 1
Maximum 100
min, increment, and max may be numbers or reporters
Value 25
Units (optional)
☐ vertical?

Figure 3: Slider for initializing the rabbit population. The range of values from which to choose is arbitrary and the starting value, set at 25, is also a matter of choice.

```
;------------------------------------------------------------
; Define globals
;-----------------------------------------------------------
; Define the parameter to hold rabbit upper bound
globals [max-rabbits]
; Define variables involved
breed [rabbits rabbit]
breed [foxes fox]
; Give each turtle an energy attribute
turtles-own [energy]
;------------------------------------------------------------
; Define the Setup procedure
;------------------------------------------------------------
to Setup
  ; Clear all previous contents
  clear-all
  ; Adjust maximum possible number of rabbits depending on
  ; if the online version of NetLogo is being used or not
  ifelse netlogo-web? [set max-rabbits 10000]
                      [set max-rabbits 20000]
  ; Color all patches green
  ask patches [set pcolor green]
  ;------------------------------------------------------------
  ; Set the stage for, and placement of rabbits on canvas
  ;------------------------------------------------------------
  create-rabbits initial-number-rabbits
  [
    set shape "circle"
    set color white
    set size .5
    ; Give each rabbit some initial energy
    set energy random (2 * rabbit-gain-from-food)
    ; Place initial rabbits in random locations
    setxy random-xcor random-ycor
  ]
  ;------------------------------------------------------------
  ; Set the stage for, and placement of foxes on canvas
  ;------------------------------------------------------------
  create-foxes initial-number-foxes
  [
    set color black
    set size 1
    ; Give each fox some initial energy
    set energy random (2 * fox-gain-from-food)
    ; Place initial foxes in random locations
    setxy random-xcor random-ycor
  ]
  ; Reset ticks counter to zero
  reset-ticks
; End the Setup procedure
end
;------------------------------------------------------------
```

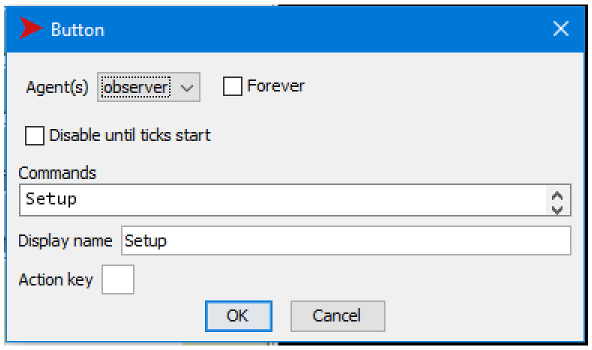
Figure 4: Code to define global variables and the `Setup` procedure is placed at the very top of the coding window. The `Setup` procedure initializes the canvas on which the model simulation will be displayed and also places an initial number of rabbits and foxes in random locations. Readers are encouraged to refer to the *NetLogo Dictionary* for definitions of keywords used here and descriptions of what they do.

```
;------------------------------------------------------------
; Miscellaneous procedures to be used by the simulation
;------------------------------------------------------------
; Define the move procedure
;------------------------------------------------------------
to move
  ; Randomly assign left or right turn angle between 0
  ; and 360 degrees, and then one step forward
  ifelse random 2 < 1 [lt random 360] [rt random 360]
  fd 1
end
;------------------------------------------------------------
; Define the rabbit reproduction procedure, share energy,
;  and make offspring move one step in a random direction
;------------------------------------------------------------
to reproduce-rabbits
  if random-float 100 < rabbit-reproduce
  [
    set energy (energy / 2)
    hatch 1 [rt random-float 360 fd 1]
  ]
end
;------------------------------------------------------------
; Define the fox reproduction procedure, share energy,
; and make offspring move one step in a random direction
;------------------------------------------------------------
to reproduce-foxes
  if random-float 100 < fox-reproduce
  [
    set energy (energy / 2)
    hatch 1 [rt random-float 360 fd 1]
  ]
end
;------------------------------------------------------------
; Define the predation procedure
;------------------------------------------------------------
to eat-rabbits
  let prey one-of rabbits-here
  if prey != nobody
  [
    ask prey [die]
    set energy energy + fox-gain-from-food
  ]
end
;------------------------------------------------------------
; Define the death procedure. Agents die if their energy
; level is less than or equal to zero
;------------------------------------------------------------
to death
  if energy <= 0 [die]
end
;------------------------------------------------------------
```

Figure 5: Code for miscellaneous procedures needed to run a simulation of the model are placed after the end of the `Setup` procedure. Again, readers are encouraged to refer to the *NetLogo Dictionary* for definitions of keywords used here and descriptions of what they do.

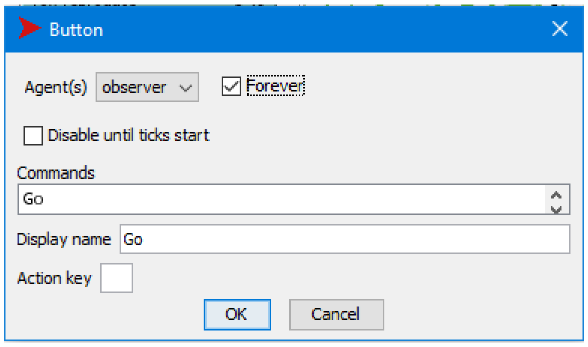
```
;-----------------------------------------------------------
; Define the Go procedure, this runs the simulation
;-----------------------------------------------------------
to Go
 ; Set exit conditions
 ; If there are no foxes and no rabbits, stop
 if not any? turtles [stop]
 ; If there are no foxes and the number of rabbits is too
 ; high, stop
 if not any? foxes and count rabbits > max-rabbits
  [user-message "Rabbits rule!" stop]
 ;----------------------------------------------------------
 ; Ask the rabbits to move and reproduce
 ;----------------------------------------------------------
 ask rabbits
  [
    move
    reproduce-rabbits
  ]
 ;----------------------------------------------------------
 ; Ask the foxes to move, lose energy, eat rabbits, die,
 ; or reproduce
 ;----------------------------------------------------------
 ask foxes
  [
    move
    set energy energy - 1
    eat-rabbits
    death
    reproduce-foxes
  ]
 ;----------------------------------------------------------
 ; Advance the tick counter
 ;----------------------------------------------------------
 tick
; End the Go procedure
end
;-----------------------------------------------------------
```

Figure 6: Code for the Go procedure is placed at the very end of the program. This procedure calls the previously defined procedures shown in Figure 5 in an appropriate order to perform all tasks associated with a single iteration of a simulation of the ABM.



Agent(s) observer
☐ Forever
☐ Disable until ticks start
Commands
Setup
Display name Setup
Action key

Figure 7: The Setup button provides a convenient means of running code in the Setup procedure. This button calls the Setup procedure to perform the initialization process.



Agent(s) observer
☑ Forever
☐ Disable until ticks start
Commands
Go
Display name Go
Action key

Figure 8: The Go button. Checking the Forever box instructs NetLogo to perform repeated runs of this procedure until one of the exit conditions in the Go procedure is met, or the user stops the process by clicking on the Go button a second time.

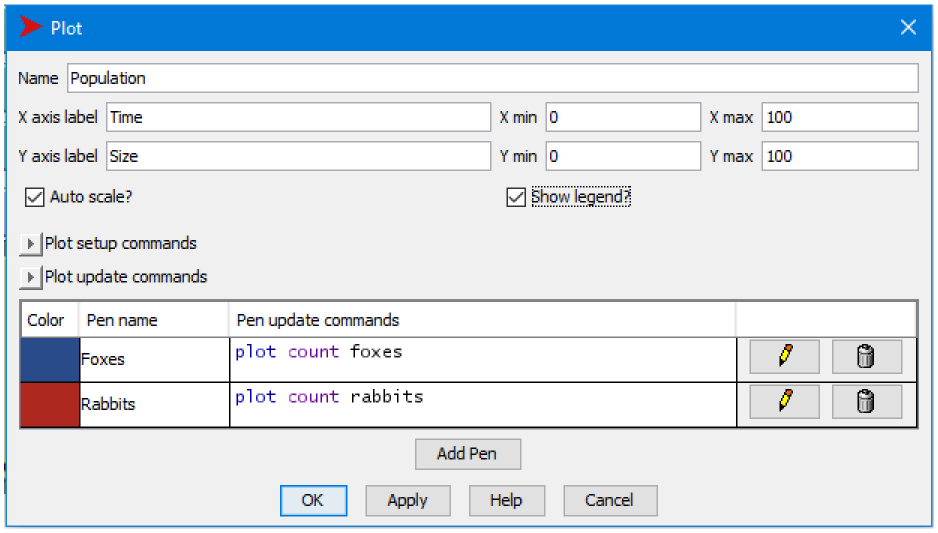
ful in catching syntax errors (a red check-mark indicates problems); however, detecting logical errors is the responsibility of the modeler. Using a *modular* approach in coding with plenty of procedures for specific tasks rather than one long program makes tracking down logical errors more efficient. To avoid having to track down large numbers of logical errors, it is best to check the soundness of each procedure as and when it is written.

4.3 Simulations and observations

To perform a simulation, begin by clicking on the Setup button. This will initialize the canvas, see Figure 10. Notice that if the Setup button is clicked again (and again) the initial placement of the foxes and rabbits populations will change—sometimes foxes will be among the rabbits, and sometimes one or more foxes will be isolated from their prey. Then, clicking on the Go button will start a simulation, using the initial scenario provided by the Setup procedure. The speed at which the simulation is displayed can be adjusted using the slider located above the canvas. Typically, the normal speed suffices, but sometimes slowing things down allows a clearer view of what is happening. To stop a simulation, click on the Go button again. To run a new simulation, click on the Setup button to create a fresh starting canvas, and then click on the Go button.

When a simulation of this model is initiated, two things start happening: The rabbits and foxes move around on the canvas, reproduce, and die; and the plot of the population sizes continually updates.

On running a few simulations it will be noticed that the plot of the population sizes of foxes and rabbits does not necessarily behave like its counterpart from a compartmental predator-prey model, see [1]. While the fox population size does tend to lag behind the rabbit pop-



Name: Population

X axis label: Time X min: 0 X max: 100

Y axis label: Size Y min: 0 Y max: 100

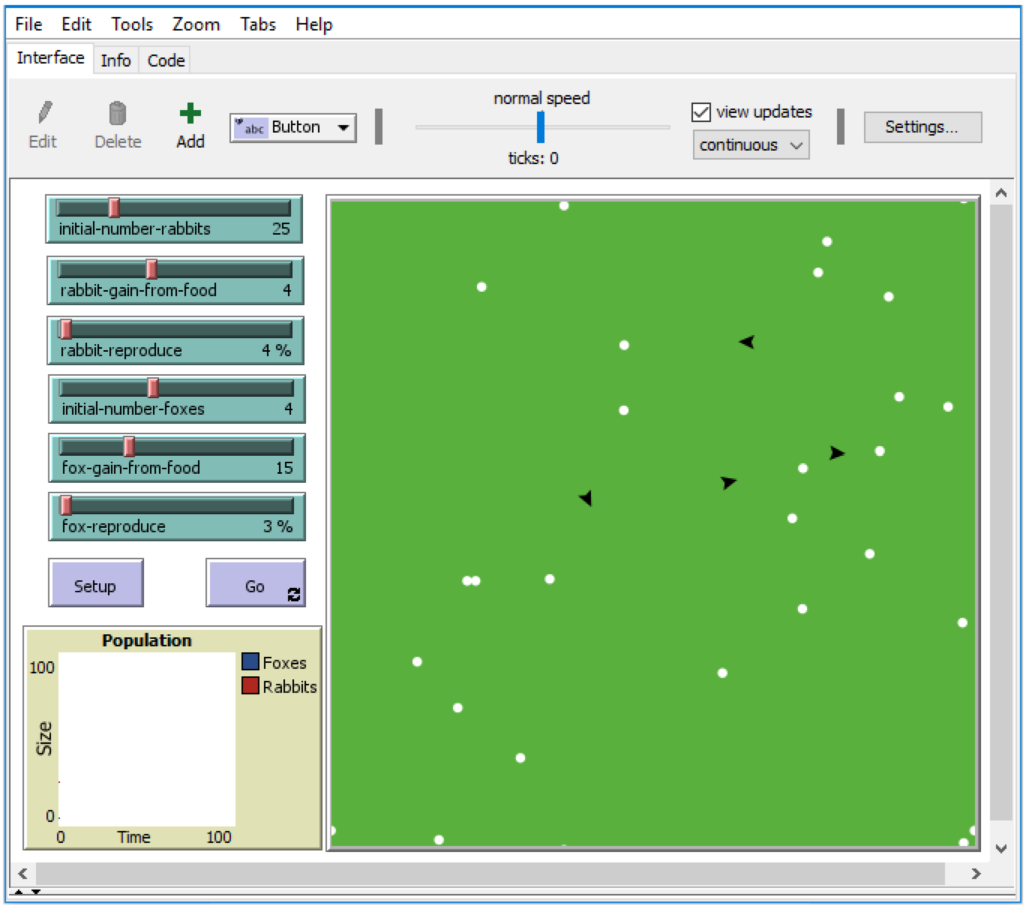
☑ Auto scale? ☑ Show legend?

Plot setup commands

Plot update commands

Color	Pen name	Pen update commands
🟦	Foxes	`plot count foxes`
🟥	Rabbits	`plot count rabbits`

Figure 9: Selecting `Plot` from the drop-down menu and clicking on a blank space opens this popup window. Then, user-filled-in commands, appropriate plot color selections, and preferred labels (pen names) produce the desired plots.



normal speed

ticks: 0

☑ view updates

continuous

initial-number-rabbits 25

rabbit-gain-from-food 4

rabbit-reproduce 4 %

initial-number-foxes 4

fox-gain-from-food 15

fox-reproduce 3 %

Figure 10: The sliders, buttons, and the plot can be moved around as desired—right-click and select the object to be moved, then drag it to a preferred location. This figure shows the interface window after a rearrangement, and after clicking the `Setup` button. The white dots in the grid represent rabbits, and the black spear-tips represent foxes.

ulation size, tidy and consistent periodic curves do not seem to appear—more often than not, one or the other of the two species ends up dying out. This seems to be the case even if the sliders are used to change parameter values for simulations. One might wonder why this is the case.

Of course, it should be noted that an ABM does not rely on differential equations for which instantaneous and uniform mixing of the predator and prey populations is assumed. This means that a fox will not necessarily be placed near enough to a rabbit to ensure it gets food in time to survive and/or reproduce. Also, the manner in which foxes gain from food through predation, and the manner in which the two species are made to reproduce in this ABM differs from the compartmental model equivalent—here these actions are performed in a random manner.

5 Foxes, Rabbits, and Grass

Consider a small enhancement to the previously described model. Suppose the rabbits are able to eat all the grass in a patch, resulting in the possibility of a rabbit ending up in a patch with no food. Suppose further that grass in a depleted patch regrows after a certain time. So, while rabbits in some areas may run out of food, there will be other areas where the grass has regrown.

5.1 Setup and code

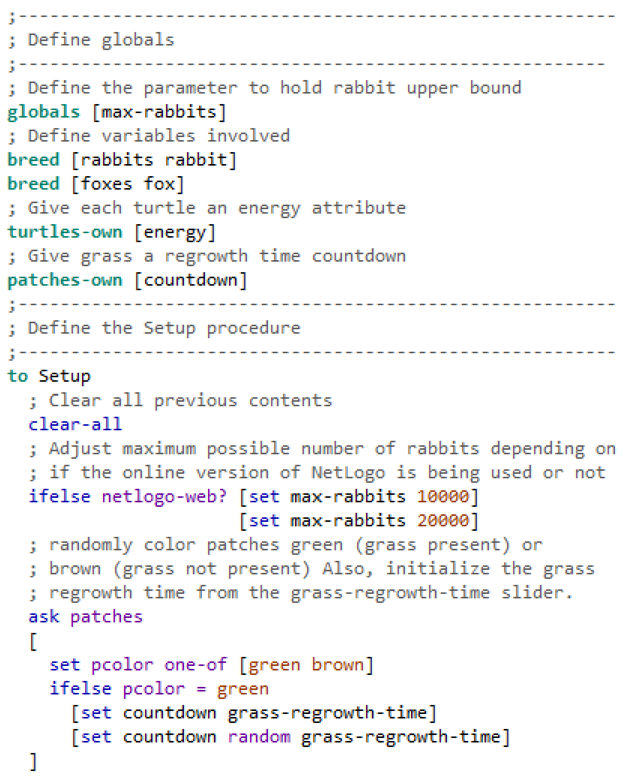
The code from the previous example, see Figures 4–6, can be adapted to accommodate these changes quite easily as follows.

First, a slider (`grass-regrowth-time`, ranging from 0 to 10 with a starting value of 3) is inserted in the interface window. Then, in the `globals` definitions section of the code, see Figure 11, the grass (the patches) are given a regrowth `countdown` variable. The start of the `Setup` function is then altered so that patches with grass and without grass are scattered randomly about the canvas.

Next, two additional procedures, `grow-grass` and `eat-grass`, are included at the end of the miscellaneous procedures portion of the code, see Figure 12.

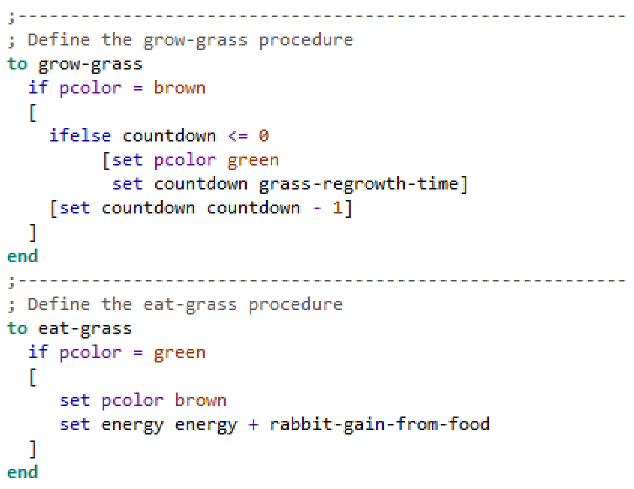
One might be curious about the paths rabbits and foxes follow throughout a simulation, but it would be nice to be able to switch this feature on and off. Here is where a `Switch` comes in use, see Figure 13.

Some small changes to the `Go` procedure shown in Figure 6 cover the scenarios posed, see Figure 14. Notice the inclusion of an additional exit condition in the beginning: a simulation will stop if the number of ticks exceeds 10,000. Then, an `ifelse` command is included at the start of both the `ask rabbits` and the `ask foxes`



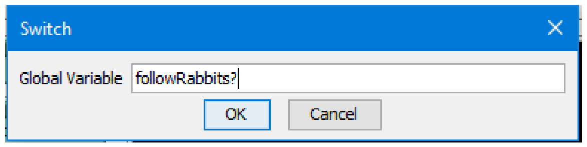
```
;----------------------------------------------------------
; Define globals
;---------------------------------------------------------
; Define the parameter to hold rabbit upper bound
globals [max-rabbits]
; Define variables involved
breed [rabbits rabbit]
breed [foxes fox]
; Give each turtle an energy attribute
turtles-own [energy]
; Give grass a regrowth time countdown
patches-own [countdown]
;----------------------------------------------------------
; Define the Setup procedure
;----------------------------------------------------------
to Setup
  ; Clear all previous contents
  clear-all
  ; Adjust maximum possible number of rabbits depending on
  ; if the online version of NetLogo is being used or not
  ifelse netlogo-web? [set max-rabbits 10000]
                      [set max-rabbits 20000]
  ; randomly color patches green (grass present) or
  ; brown (grass not present) Also, initialize the grass
  ; regrowth time from the grass-regrowth-time slider.
  ask patches
  [
    set pcolor one-of [green brown]
    ifelse pcolor = green
      [set countdown grass-regrowth-time]
      [set countdown random grass-regrowth-time]
  ]
```

Figure 11: Altered portions of the global variables and the `Setup` procedure from Figure 4. The remaining code for the `Setup` procedure is as shown in Figure 4.



```
;----------------------------------------------------------
; Define the grow-grass procedure
to grow-grass
  if pcolor = brown
  [
    ifelse countdown <= 0
         [set pcolor green
          set countdown grass-regrowth-time]
    [set countdown countdown - 1]
  ]
end
;----------------------------------------------------------
; Define the eat-grass procedure
to eat-grass
  if pcolor = green
  [
     set pcolor brown
     set energy energy + rabbit-gain-from-food
  ]
end
```

Figure 12: Additional miscellaneous procedures added to the end of those shown in Figure 5. The remaining procedures are as shown in Figure 5.


Switch

Global Variable followRabbits?

Figure 13: Switch to track rabbit movements. A switch to track foxes movements is also included.

```
;-----------------------------------------------------------
; Define the Go procedure, this runs the simulation
;-----------------------------------------------------------
to Go
 ; Set exit conditions
 ; If there are no foxes and no rabbits, stop
 if not any? turtles [stop]
 ; If there are no foxes and the number of rabbits is too
 ; high, stop
 if not any? foxes and count rabbits > max-rabbits
  [user-message "Rabbits rule!" stop]
 ; Stop the simulation if ticks exceed 10000
  if ticks > 10000 [user-message "Time is up!" stop]
 ;-----------------------------------------------------------
 ; Make the rabbits move, lose energy with movement, die,
 ; reproduce, and eat grass
 ;-----------------------------------------------------------
 ask rabbits
  [
    ifelse followRabbits? [pen-down] [pen-up]
    move
    set energy energy - 1
    eat-grass
    death
    reproduce-rabbits
  ]
 ;-----------------------------------------------------------
 ; Make the foxes move, lose energy, eat rabbits, die,
 ; or reproduce
 ;-----------------------------------------------------------
 ask foxes
  [
    ifelse followFoxes? [pen-down] [pen-up]
    move
    set energy energy - 1
    eat-rabbits
    death
    reproduce-foxes
  ]
 ;-----------------------------------------------------------
 ; Grow new grass and advance tick counter
 ;-----------------------------------------------------------
  ask patches [grow-grass]
  tick
; End the Go procedure
end
;-----------------------------------------------------------
```

Figure 14: The `Go` procedure shown in Figure 6 is altered to include the grass eating and growing procedures, as well as the tracking (or not tracking) of the movements of the two species.

instructions. Depending on the switch setting, these instruct NetLogo to trace (or not trace) the movements of rabbits and/or foxes. Also in the `ask rabbits` instructions, rabbits are made to lose energy with movement, and gain energy by eating grass. Finally, patches are asked to grow grass if the prescribed regrowth time has passed.

5.2 Simulations and observations

Just like the previous model, results of simulations for this model using the initial setup shown in Figure 15 also depend on the placement of foxes. Sometimes the foxes disappear early in a simulation, and sometimes they last a little longer. Playing around with the `gain-from-food` sliders make simulations more interesting. For example, if the foxes gain 30 units from food and the rabbits gain 5 units, the foxes end up decimating the rabbit population and then themselves. Using the sliders to increase the rabbit reproduction rate to 20% and lowering the fox reproduction rate to 5% seems to yield longer simulations.

An interesting exercise for the reader might be to include instructions that make the rabbits move in search of food and away from foxes, and then have foxes move in search of rabbits and away from other foxes.

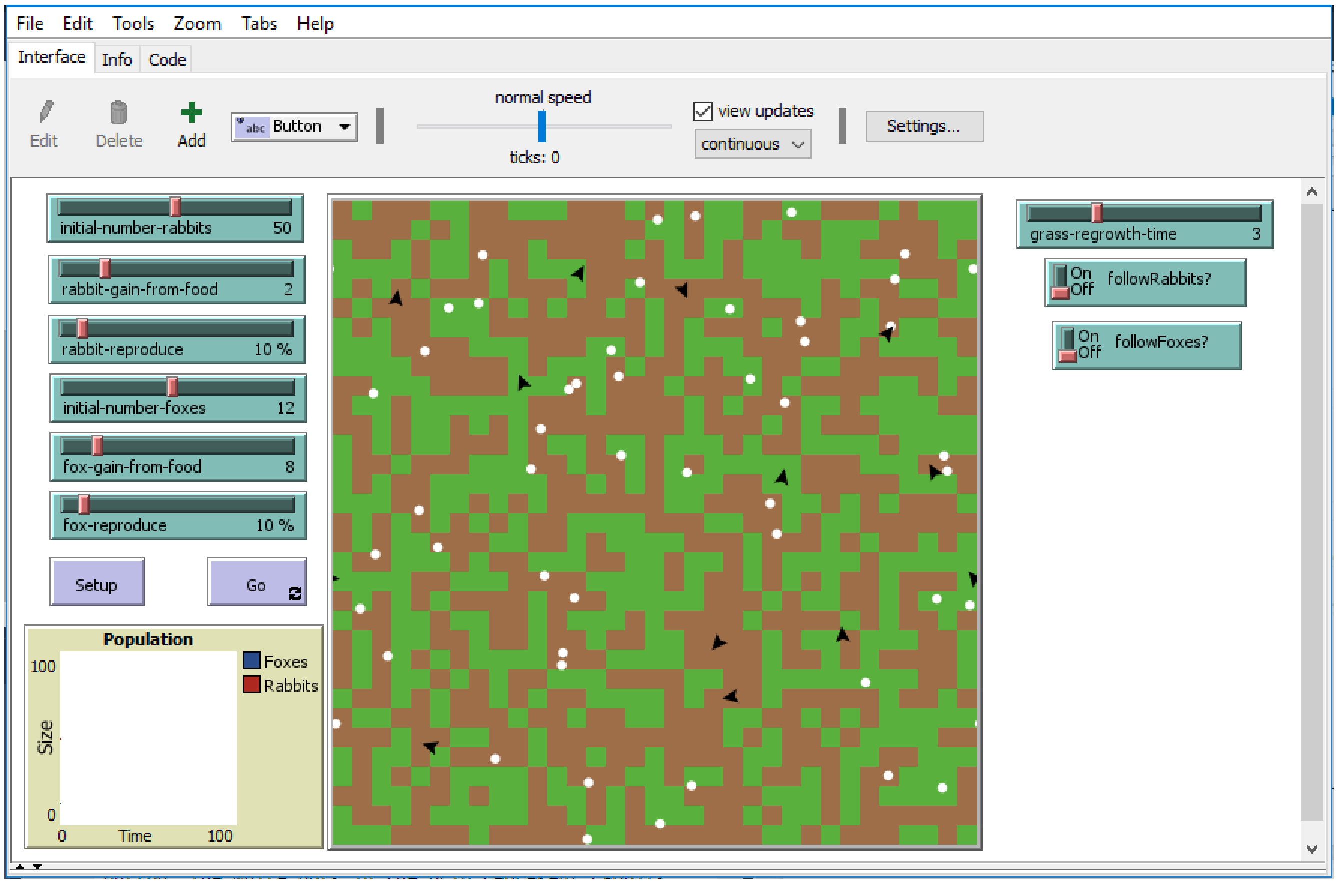
6 For the Interested

This section is for those who wish to dive into the world of ABM and NetLogo to explore their rich applications. The articles and books cited here represent just a fraction of the literature in this field, and bibliographies of the articles and books cited provide further sources.

The literature reviews in [3] and [6] provide a range of applications of ABM beginning in the early years with selections from around 1972 to around 2014. Earlier applications did not involve NetLogo, which was first created in 1999, see the FAQs in [21].

Beginning with modeling infectious diseases, for example, the spread of the Zika virus at the 2016 Olympics is modeled using agent-based modeling in [11]. Another example of modeling infectious diseases is given in [18] where the authors use agent-based modeling in NetLogo to simulate the spread of COVID-19 in a traditional college classroom. One of the many applications of ABM in medicine appears in [2], where the firing of neurons over time in response to painful bladder stimulation is analyzed through simulations with an agent-based model.

In ecological research, ABM serves as a powerful tool in enabling scientists to simulate complex interactions within ecosystems at an individual level. The article [17] provides a foundational introduction to how individual-based models can be used to better predict individual-



ticks: 0
initial-number-rabbits 50
rabbit-gain-from-food 2
rabbit-reproduce 10 %
initial-number-foxes 12
fox-gain-from-food 8
fox-reproduce 10 %
grass-regrowth-time 3
followRabbits?
followFoxes?
Population: Size 0–100, Time 0–100

Figure 15: The starting screen for the altered model. Brown patches of the canvas represent grass-less areas, and green patches represent areas with grass.

based ecology. An example of an agent-based modeling approach to examine winter survival rates of American robins and eastern bluebirds with discussions on the relationship between the two species of birds and their survival appears in [9]. More broadly, [13] provides an example of applying ABM to wildlife ecology and management through an exploration of how different species respond to potential changes in environmental conditions.

In a similar vein, the authors of [16] use a behavior-based ABM to simulate the underlying behavioral mechanisms of caribou and explore strategies they use in navigating winter landscapes in search of a preferred winter habitat. An application to fisheries appears in the unpublished article [12], which explores the use of an agent-based model in studying the return of Pacific salmon to their freshwater spawning grounds in the Snake River.

Moving to the social sciences, one of the earliest applications of the agent-based approach to modeling is introduced in [15], see also [5] and [20]. These explore how individuals in a population tend to distribute themselves into self-segregated neighborhoods. An application of agent-based modeling to interpersonal violence and the effectiveness of the *Green Dot Bystander Training Program* is explored in [10].

Finally, good resources to get started with Netlogo are the website [4] and the article [8]. Then, the books [7] and [14] are excellent resources for gaining a strong foundation in ABM and the use of NetLogo.

Acknowledgments

The idea leading to this article was motivated at the 2023 Cross-Institutional Undergraduate Research Experience (CURE) workshop. This annual workshop is sponsored by the Intercollegiate Biomathematics Alliance (IBA), a subsidiary of the Center for Collaborative Studies in Mathematical Biology at Illinois State University. The authors are indebted to the IBA for providing academic and other relevant resources starting with the inception of the project up to its conclusion. The authors would like to thank the reviewer for suggestions on improving the readability of this article.

Author Contributions

The idea for this project was posed by Olcay Akman at the 2023 CURE Workshop. Siddharth Bhumpelli researched the application of NetLogo to ABM and decided on the examples used. Olcay Akman and Christopher Hay-Jahans provided guidance and motivation when and as needed in writing and persisting with the project.

References

[1] Akman, O., Bhumpelli, S., Cline, C., and Hay-Jahans, C. (2024). Compartmental modeling for the neophyte: An application of Berkeley Madonna. *Spora: A Journal of Biomathematics*, 10(1):7–14. DOI: 10.61403/2473-5493.1089.

[2] Baktay, J., Neilan, R. M., Behun, M., McQuaid, N., and Kolber, B. (2019). Modeling neural behavior and pain during bladder distention using an agent-based model of the central nucleus of the amygdala. *Spora: A Journal of Biomathematics*, 5(1):1–13. DOI: 10.30707/SPORA5.1Baktay.

[3] DeAngelis, D. L. and Grimm, V. (2014). Individual-based models in ecology after four decades. *F1000-Prime Reports*, 6:39. DOI: 10.12703/P6-39.

[4] Getting started with NetLogo (n.d.). `https://ccl.northwestern.edu/netlogo/bind/article/getting-started-with-netlogo.html`

[5] Grauwin, S., Goffette-Nagot, F., and Jensen, P. (2012). Dynamic models of residential segregation: An analytical solution. *Journal of Public Economics*, 96:124–141. DOI: 10.1016/j.jpubeco.2011.08.011.

[6] Grimm, V. (1999). Ten years of individual-based modelling in ecology: What have we learned and what could we learn in the future? *Ecological Modelling*, 115(2–3):129–148. DOI: 10.1016/S0304-3800(98)00188-4.

[7] Grimm, V. and Railsback, S. F. (2005). *Individual-based modeling and ecology*. Princeton University Press, Princeton, NJ.

[8] Grimm, V., Berger, U., Bastiansen, F., Eliassen, S., Ginot, V., Giske, J., . . . , DeAngelis, D. L. (2006). A standard protocol for describing individual-based and agent-based models. *Ecological Modelling*, 198: 115–126. DOI: 10.1016/j.ecolmodel.2006.04.023.

[9] Iselin, S., Segin, S., and Capaldi, A. (2016). An agent-based modeling approach to determine winter survival rates of American robins and eastern bluebirds. *Spora: A Journal of Biomathematics*, 2(1):27–34. DOI: 10.30707/SPORA2.1Iselin.

[10] Kendrick, P., Apenyo, T., and Callender Highlander, H. (2017). Agent-based models of the Green Dot bystander violence prevention program on college campuses. *Spora: A Journal of Biomathematics*, 3(1): 15–28. DOI: 10.30707/SPORA3.1Kendrick.

[11] Matheson, T. S., Satterthwaite, B., and Callender Highlander, H. (2017). Modeling the spread of the Zika virus at the 2016 Olympics. *Spora: A Journal of Biomathematics*, 3(1):29–44. DOI: 10.30707/SPORA3.1Matheson.

[12] McCarthy, C. (2017). Agent based model of salmon migration in the Snake River. Available at `https://www.whitman.edu/Documents/Academics/Mathematics/2017/McCarthy.pdf`

[13] McLane, A. J., Semeniuk, C., McDermid, G. J., and Marceau, D. J. (2011). The role of agent-based models in wildlife ecology and management. *Ecological Modelling*, 222(8):1544–1556. DOI: 10.1016/j.ecolmodel.2011.01.020.

[14] Railsback, S. F., and Grimm, V. (2019). *Agent-based and individual-based modeling: A practical introduction* (2nd ed.). Princeton University Press, Princeton, NJ.

[15] Schelling, T. C. (1971). Dynamic models of segregation. *The Journal of Mathematical Sociology*, 1(2): 143–186. DOI: 10.1080/0022250X.1971.9989794.

[16] Semeniuk, C. A., Musiani, M., Hebblewhite, M., Grindal, S., and Marceau, D. J. (2012). Incorporating behavioral–ecological strategies in pattern-oriented modeling of caribou habitat use in a highly industrialized landscape. *Ecological Modelling*, 243: 18–32. DOI: 10.1016/j.ecolmodel.2012.06.004.

[17] Stillman, R. A., Railsback, S. F., Giske, J., Berger, U. T. A., and Grimm, V. (2015). Making predictions in a changing world: The benefits of individual-based ecology. *BioScience*, 65(2):140–150. DOI: 10.1093/biosci/biu192.

[18] Talwar, S., and Yust, A. E. (2023). An investigation of mitigation measures on the spread of COVID-19 in a college classroom using agent-based modeling. *Spora: A Journal of Biomathematics*, 9(1):60–68. DOI: 10.61403/2473-5493.1086.

[19] Vincenot, C. E. (2018). How new concepts become universal scientific approaches: Insights from citation network analysis of agent-based complex systems science. *Proceedings of the Royal Society B*, 285: 20172360. DOI: 10.1098/rspb.2017.2360.

[20] Wilensky, U. (1997). NetLogo segregation model. Available at `http://ccl.northwestern.edu/netlogo/models/Segregation`. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

[21] Wilensky, U. (1999). NetLogo. Available at `http://ccl.northwestern.edu/netlogo/`. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.