Assignment Sheet

EES 4760/5760: Agent- and Individual-Based Computational Modeling

Spring, 2018

General instructions for reading assignments

- Do the assigned reading *before* you come to class on the date for which it is assigned. If you have questions or find the ideas presented in the readings confusing, I encourage you to ask questions in class.
- Download the list of errata for the textbook from https://ees4760.jgilligan.org/files/Railsback_Grimm_2012_Errata.pdf.
- Questions in the "Reading Notes" sections of the assignments are for you to think about to make sure you understand the material, but you do not have to write up your answers or turn them in.

  You are responsible for all the assigned readings, but topics I have highlighted in the reading notes are particularly important.
- In addition to the questions I ask in the reading notes, look over the "Conclusions" section at the end of each chapter in *Agent-Based Modeling* to check whether you understand the key facts and concepts from the chapter.

Class #1, Tue, Jan. 9: Introduction

Reading:

No new reading for today.

Class #2, Thu, Jan. 11: The computer modeling cycle

Homework #1: Set up Box and NetLogo

There is nothing for you to turn in, but do the following two tasks to prepare for next week:

- Download NetLogo version 6.0.2 from https://ccl.northwestern.edu/netlogo/ and install it on your computer.
- Set up your Box account. Make a Box folder for your homework for this course (call it `Lastname_EES_4760` (undergraduates) or `Lastname_EES_5760` (graduate students), substituting your own last name), and share it with me, giving me "Editor" role.

Reading:

- Railsback & Grimm, Ch. 1.
- Handout: Peter Tyson, "Artificial Societies" MIT Technology Review **100**(3) 15–17 (Apr. 1, 1997) (posted online).

READING NOTES:

This reading sets the stage for answering the questions:

1. What is computational modeling and why is it useful in social and natural science research?
2. What are agent based models? How are they different from other kinds of models? What makes them useful for scientific research?

The reading introduces the idea of a **modeling cycle**. You should understand the different steps in the modeling cycle. You should also think about why Railsback and Grimm describe modeling as a cycle, as opposed to a linear process with a start and stop.

As to what makes agent-based modeling special, Steven Railsback and Volker Grimm are ecologists and *Agent-Based Modeling* emphasizes aspects of agent-based modeling that are well suited for studying ecological systems. Others, such as social scientists, emphasize the aspects of agent-based modeling that are well suited for problems in social science. And still others, such as computer scientists, emphasize aspects of automated and autonomous things (ranging from packets of data on a network to swarms of robots or flying drones that need to coordinate their activities and avoid collisions). What all of these approaches have in common are their use of individuals or **agents** (what is an agent?), which inhabit some kind of space or **environment** (this could be physical space or an abstract space, such as a computer network). Agents **interact** with each other and with the environment, and they make **decisions** according to rules.

The article "Artificial Life" gives you a feel for how an early agent-based model called "Sugarscape" was used as part of a very influential research project in the 1990s. Joshua Epstein and Robert Axtell who wrote Sugarscape are highly respected pioneers in agent-based modeling and the Sugarscape model set off a revolution in agent-based modeling by showing that a very simple model could reproduce complex phenomena that are observed in real societies. As you read through this article, think about what the different applications of agent-based models have in common. Do these suggest questions that you might be interested in exploring with agent-based models. Do you have questions, as you read this, about whether computer modeling can really tell you about real societies?

Agent-based models are often used to examine **emergent** phenomena. Neither reading describes clearly what *emergence* means. There is no simple definition, but during the semester we will pay a lot of attention to learning about emergence and trying to understand it. Do not worry if you don't understand emergence at this point. Emergence is difficult to put into words, and it's much easier to understand from experience. Over the course of the semester, we will work together to understand what emergence is and how to study it.

You can download "Artificial Societies" from Brightspace or https://ees4760.jgilligan.org/files/reading/Tyson_1997_Artificial_Societies.pdf.

Class #3, Tue, Jan. 16: Introduction to NetLogo

Homework #2: Introducing NetLogo

Turn in the following homework on Box by 11:59 pm:

- As you read along with Section 2.3, follow along on your computer and build the *Mushroom Hunt* model by typing in the code shown in the textbook. The whole program is shown on pages 28–29. **Save your model as `mushroom_hunt.nlogo`**.

  This exercise may seem very simple, but it is the first step toward learning how to program NetLogo and it will be an important first step toward writing your own models.

  After you are done typing in your model, try running it.

When you are done, make a subfolder called `HW_2` in your Box folder for this class, and upload your model.

NOTES ON HOMEWORK:

I recommend getting together with a classmate and working together on this assignment. Since the assignment consists of typing code in and running it, do not worry about your work being identical to your partner's. However, I strongly recommend that you type everything in yourself because you will not learn if you just copy someone else's code or download the code from a source on the web.

If you run into trouble and cannot make your model work, do not worry. Ask a classmate for help, or email me (and attach your `.nlogo` model file), or simply come to class on Tuesday with questions about the problems you had getting your model to work. Since your model file will be on Box, we will be able to discuss the problems in class.

Reading:

- Railsback & Grimm, Ch. 2.

READING NOTES:

Familiarize yourself with NetLogo. I recommend that you read through the chapter with NetLogo open on your computer. Feel free to play around with NetLogo and try things out. The homework consists of following the step-by-step creation of a model in section 2.3.

Errata for chapter 2:

- On page 20, after you add the `setup` and `go` buttons to the interface, but before you write the `go` procedure, when you click the "Check" button, it will take you to the interface page because the `go` button has an eror: the procedure `go` does not exist yet. After you write the `go` procedure in the Code tab, then the Check button will work as described in the book.
- The first bullet on page 21 should read:
  - Just as we created the `setup` procedure (with two lines saying `to setup` and `end`, write the "skeleton" of the `go` procedure.

Class #4, Thu, Jan. 18: Specifying models: The ODD protocol

Homework #3: Becoming familiar with NetLogo

This homework consists of reading and working through tutorials, so there is nothing to turn in.

- Everyone should do exercises 1–2 in Chapter 2 of Railsback & Grimm. This consists of reading and working through tutorials, so there is nothing to turn in.

Reading:

- Railsback & Grimm, Ch. 3.

READING NOTES:

You can download several useful documents related to the ODD protocol from the class web site:

- The journal article, V. Grimm *et al.* (2010). "The ODD protocol: A review and first update" *Ecological Modeling* **221**, 2760–68.. https://ees4760.jgilligan.org/files/odd/Grimm_2010_ODD_update.pdf
- A Word document that provides a template for writing ODDs: https://ees4760.jgilligan.org/files/odd/Grimm_2010_odd_template.docx
- Lists of scientific publications using agent-based and individual-based models that either do or don't use the ODD protocol (this appeared as Appendix 1 of the Grimm *et al.* paper): https://ees4760.jgilligan.org/files/odd/Grimm_2010_appendix_1.pdf, https://ees4760.jgilligan.org/files/odd/ch3_ex1_pubs_with_no_ODD.pdf, https://ees4760.jgilligan.org/files/odd/ch3_ex2_pubs_with_ODD.pdf.

Class #5, Tue, Jan. 23: Your first model

Homework #4: Experimenting with NetLogo

Make a folder called "HW_4" in your Box folder and upload your work when you're done (Word or text files for the descriptions and ODD document, and `.nlogo` files for your NetLogo models).

- Railsback & Grimm, Chapter 2, exercises Exercises 3–4.

  For exercise 4 in chapter 2, you will make seven sequential modifications of the basic mushroom hunt model. Each step modifies the previous one, so the last model will have all the modifications from the bulleted list in Ex. 2.4. Save each model with a new name, such as `ex_2_4a.nlogo`, `ex_2_4b.nlogo`, ..., `ex_2_4g.nlogo`

- Railsback & Grimm, Chapter 3 exercise 3.

  Write your answers in any convenient text format (a simple text file, a Word document, a `.pdf` file, or whatever suits you). Call the file `ex_3_3.docx` (or `ex_3_3.pdf`, etc.).

NOTES ON HOMEWORK:

My advice for Chapter 3, Exercise 3 is don't be too ambitious with your model, but keep it very simple. Don't worry about getting everything right. If there are things you don't feel sure about or don't know how to express, you should just write a parenthetical note in your ODD document commenting on your difficulty. Come to class prepared to talk about how this exercise went and where you felt confused about trying to specify your model.

As always, I recommend working with another classmate or in a group, comparing your different ways of approaching the exercises, and discussing any difficulties together.

Reading:

- Railsback & Grimm, Ch. 4.

READING NOTES:

For the reading, read Chapter 4 in *Agent-Based Modeling* first and focus mostly on this chapter.

It is worth noting that the Sugarscape model you read about for Jan. 11 is very similar to the hilltopping model. Sugarscape was part of a very influential research project in the 1990s, in which Joshua Epstein and Robert Axtell showed that a very simple model could reproduce complex biological and economic phenomena that are observed in real societies and ecosystems. You may find it interesting at this point to look back at the "Artificial Societies" article and to play with the Sugarscape model in NetLogo.

You can downlload an ODD for the butterfly model, which is suitable to paste into the "Info" tab in NetLogo, from the class web site: https://ees4760.jgilligan.org/files/models/chapter_04/butterfly_model_ODD.txt

Class #6, Thu, Jan. 25: Using models for science

Reading:

- Railsback & Grimm, Ch. 5.

READING NOTES:

This reading sets the stage for answering the big question, "How can we use agent-based models to do science?" There are several aspects to this question, which this chapter will introduce:

1. How can we produce quantitative output from our models?
2. How can your models read and write data to and from files? (This is important for connecting your model to other parts of your project)
3. How should we test our models to make sure they do what we think they do? (More on this in Chapter 6)
4. Making your research reproducible by using version control and documentation.

A number of you like to use Excel or statistical analysis tools, such as R, SPSS, or Stata. The material in this chapter about importing and exporting data using text or `.csv` files will be very useful for this. By default, NetLogo only allows you to read in data in simple text files. However, if comes with some extensions that you can use to read in data from other common file formats, including `.csv` and ArcGIS shapefiles and raster (grid) files.

If you want to read in data from csv files, you may want to look at the documentation for the `csv` extension to NetLogo. To use it, you just put the line `includes [csv]` as the first line of your model, and then use functions from the extension, such as `let data csv:from-file "myfile.csv"`.

To read in date from ArcGIS files, look at the documentation for the `GIS` extension. You would put the line `extensions [ gis ]` as the first line of your model, and then use functions, such as `gis:load-dataset`, which can load vector shape files (`.shp`) and raster grid files (`.grd` or `.asc`). The GIS extension offers a lot of functions for working with vector and raster GIS data. If you're interested in using GIS data in your models, take a look at the GIS examples in the NetLogo model library.

You can download the data file with the elevations for the realistic butterfly model from https://ees4760.jgilligan.org/files/models/chapter_05/ElevationData.txt

Errata for Chapter 5:

- Page 68: Replace the statement

```
export-plot "Corridor width"
    word "Corridor-output-for-q-" q
```

with this:

```
export-plot "Corridor width"
    (word "Corridor-output-for-q-" q ".csv")
```

The revised statement appends the file extension "`.csv`" to the file name created via the `word` primitive. Note that when `word` is used to concatenate more than two values tovether, it and all the values must be inside parentheses.

Class #7, Tue, Jan. 30: Testing and validating models

Homework #5: Science with models: Butterfly mating

Turn in the following homework on Box by 11:59 pm:

- **Everyone:**
  - Railsback & Grimm, Ch. 4, Ex. 4.2, 4.4
  - Railsback & Grimm, Ch. 5, Ex. 5.1–5.5.
- **Graduate Students:**
  - Railsback & Grimm, Ch. 5, exercises 5.6–5.7.

Reading:

- Railsback & Grimm, Ch. 6.

READING NOTES:

No one writes perfect programs. Errors in programs controlling medical equipment have killed people. Errors in computer models and data analysis code have not had such dire results, but have wasted lots of time for researchers and have caused public policy to proceed on incorrect assumptions. In many cases, these errors were uncovered only after a great deal of frustration because the original researchers would not share their computer codes with others who were suspicious of their results.

You can never be certain that your model is correct, but the more aggressively you check for errors the more confident you can be that it does not have major problems.

Two very important things you can do to ensure that your research does not suffer a similar fate are:

1. Test your code. Assume your program has errors in it and make the search for those errors a priority in your programming process. Some things you can do in this regard are:
   - Write your code with tests that will help you find errors.
   - Work with a partner: after one of you writes code, the other should read it and check for errors.
   - Break your program up into small chunks. It is easier to test and find bugs if you are looking at a short block of code than if you are looking at hundreds of lines of code.
   - Independently reimplement submodels and check whether they agree with the submodel you are using.
2. Publish your code. If you trust your results and believe they are important enough to publish in a book or journal, then you should make your code available (there are many free sites, such as github.com and openabm.org where people can publish their models and other computer code).

   The more that other researchers can read your code, the greater the probability that they will find any errors, and if you make it easy for others to use your code, it will help science because other

people can build on your work, and it will help your reputation because when other people use your model or other code they are likely to cite the publication in which you first announced it, so your work will get attention.

Many scholarly journals demand that you make your code available as a condition for publishing your paper, and federal funding agencies are increasingly requiring that any research funded by their grants must make its code and data available to other researchers and the public.

The Marriage Model

The ODD for the marriage model and a NetLogo model that implements the ODD, but with many errors, can be downloaded from the class web site: https://ees4760.jgilligan.org/files/models/chapter_06/Marriage_age_model_ODD.pdf, and https://ees4760.jgilligan.org/files/models/chapter_06/Marriage_model_with_errors.nlogo.

Errata for chapter 6:

- Section 6.3.7, in the example `test` procesure: The syntax of the `foreach` statement changed, starting with NetLogo version 6.0. This was because NetLogo introduced a new feature, called "anonymous procedures," which you can read about in the NetLogo User Manual. This means that we need to change the code in the book for the `go-back` test procedure. The new procedure should read:

```
ask turtles
 [
    set color blue ; Make the return path a different color
    foreach path   ; See "foreach: executes once for each
                   ; patch on the list, where next-patch is the
                   ; patch currently being executed
    [ next-patch ->
      set heading towards next-patch
      fd 1
    ]
 ]
```

- Section 6.3.10, Figure 6.4: The column labels in the very top row of the fiture's spreadsheet are wrong: what is labeled col. B should be col. A, the column labeled O should be N, etc.. In the figure caption, reference to col. N should be to the corrected col. M (which is currently labeled N); and references to col. M should be to the corrected col. N (currently labeled O).
- Section 6.5, p. 91: Note that the "social network" of young people can potentially include other people below the marriage age of 16. Hence, it is possible for a person less than 16 to be selected as a spouse and married.

Class #8, Thu, Feb. 1: Choosing Research Projects

Homework #6: Reproducing a model from its ODD

Turn in the following homework on Box by 11:59 pm:

- **Graduate Students:**
  - Graduate students should do Railsback & Grimm, Ex. 5.11

NOTES ON HOMEWORK:

- **Everyone:**

- **Undergraduates:**
- **Graduate Students:**You can download the journal article for this exercise, R. Jovani & V. Grimm. (2008) "Breeding synchrony in colonial birds: From local stress to global harmony", *Proc. Royal Soc. London B* **275**, 1567–63 from the class web site, https://ees4760.jgilligan.org/files/models/chapter_05/Jovani_Grimm_2008_Breeding.pdf.

  Errata for homework exercise:

  - The parameter SD should have a value of 10.0, not 1.0.

Reading:

- Railsback & Grimm, Ch. 7.

READING NOTES:

There is not very much reading for today. Read Chapter 7 of *Agent-Based Modeling* carefully (it's very short) The point of this chapter is to help you get ideas for your term research project.

Before class, I want you to read through the research project assignment and think about what you might want to do for a research project. We will spend class talking about possible research projects.

Class #9, Tue, Feb. 6: Emergence

Homework #7: Research project proposal

Turn in the following homework on Box by 11:59 pm:

- Turn in a one-page research project proposal

Reading:

- Railsback & Grimm, Ch. 8.

READING NOTES:

This is a major chapter. Emergence is one of the most important concepts in agent-based modeling, so pay close attention to the discussion in this chapter and think about how you can measure and assess emergence.

This chapter also introduces a very important tool for doing experiments in NetLogo: **BehaviorSpace**. BehaviorSpace lets you repeatedly run a NetLogo model while varying the settings of any of the controls on your user interface. Where there is randomness (stochasticity) in the model, you can perform many runs at each set of control settings. This will let us perform **sensitivity analysis** to determine whether a certain emergent phenomenon we are investigating happens only for values of the parameters within a narrow range, or whether it happens over a wide range of the parameters. It will let us determine which parameters are most important for the phenomenon.

For homework and your modeling projects you will use BehaviorSpace extensively. BehaviorSpace outputs large amounts of data to `.csv` files, which you can read into Excel, R, SPSS, or another tool where you can do statistical analysis and generate plots such as the ones in figures 8.3, 8.5, and 8.6.

The format in which BehaviorSpace saves its data is very annoying to deal with in many tools. Indeed, it's almost impossible to do anything useful with it in Excel. Because of this, I have written a tool called `analyzeBehaviorspace` that can read the output of a BehaviorSpace run and allow you to interactively graph it and re-organize the data to make it more useful.

You can either use this tool online in a web browser at https://analyze-behaviorspace.jgilligan.org or install it on your own computer. For details, see the description of `analyzeBehaviorspace` on the "Reading Resources and Computing Tools for Research" handout.

As you read the chapter, be sure to try out the experiments with the birth-and-death model and the flocking model. Try reading the output of the behaviorspace runs into analyzeBehaviorspace (the web version or a local version installed on your computer), or your favorite statistical analysis software and try to generate plots similar to figures 8.3, 8.5, and 8.6.

If you have time, try to play around with BehaviorSpace using those models (varying different parameters) or other models from the NetLogo library to explore the ways that changing parameters affects the models' behavior.

Class #10, Thu, Feb. 8: Observation

Homework #8: Testing and Debugging

Turn in the following homework on Box by 11:59 pm:

- **Everyone:**
  - Railsback & Grimm, Ch. 6, Ex. 6.2, 6.3, 6.5
- **Graduate Students:**
  - Railsback & Grimm, Ch. 6, Ex. 6.4, 6.7

Reading:

- Railsback & Grimm, Ch. 9.

READING NOTES:

In this chapter, we examine how to detect and record the properties of a model that we want to study.

The article, D. Kornhauser, U. Wilensky, and W. Rand. (2009). "Design guidelines for agent-based model visualization," *Journal of Artificial Societies and Social Simulation* **12**, 1 is available online at http://jasss.soc.surrey.ac.uk/12/2/1.html.

I have posted a refresher guide for NetLogo programming on the class web site at https://ees4760.jgilligan.org/files/models/chapter_09/ch9_ex8_Netlogo_exercises.pdf

Errata for Chapter 9:

- Section 9.3, p. 119: The discussion of histograms mentions potential difficulties setting the *x* axis to a useful range, so all the bars are visible. One solution is to use the `set-plot-x-range` primitive. For example, add this statement:

  ```
  set-plot-x-range (min [elevation] of turtles)
      (max [elevation] of turtles)
  ```

  just before the statement:

  ```
  histogram [elevation] of turtles
  ```

Or, if the histogram should always start at zero:

```
set-plot-x-range 0 max [elevation] of turtles
```

(You may need to use primitives such as `ceiling` and `precision` to convert the inputs to `set-plot-x-range` to nice round numbers.)

Class #11, Tue, Feb. 13: Sensing

Homework #9: Analyzing model experiments

Turn in the following homework on Box by 11:59 pm:

- **Everyone:**
  - Railsback & Grimm, Ch. 8, Ex. 8.1, 8.2
  - Railsback & Grimm, Ch. 9, Ex. 9.1, 9.3, 9.4
- **Graduate Students:**
  - Railsback & Grimm, Ch. 8, Ex. 8.3, 8.4
  - Railsback & Grimm, Ch. 9, Ex. 9.6

Reading:

- Railsback & Grimm, Ch. 10.

READING NOTES:

Important programming concepts in this chapter include:

Links: Agents interact with their physical environment (patches around them) and with other agents nearby, but they can also interact in social networks, which can be represented by links.

Variable scope: Understand the differences between global variables, local variables, patch variables, agent variables, and link variables. Understand how an agent can get the value of a global variable or variables belonging to a certain patch or link or another agent.

Entity detection: Understand different ways to detect which agents or patches meet certain conditions (e.g., within a certain distance, have a certain color, have the largest or smallest values of some variable, etc.).

The agents' interactions, both with their environment and with each other through sensing. Part of the design concepts section of a model's ODD consists of specifying what the agents can sense: They might be able to sense other agents within a certain distance. They might only be able to detect other agents if they are within a certain angle (e.g., the agent might be able to look forward, but might not be able to see behind itself unless it turns around). Agents might be able to detect certain qualities of one another (e.g., I can see how tall you are, but I can't see how much money you have).

Agents can interact both spatially and through networks of links. You can create many kinds of links so that agents can belong to many networks (e.g., family, co-workers, members of a church congregation, etc.).

Sensing involves two steps:

1. Detect which entities your agent (or patch) will sense.
2. Get the values of the sensed variables from those entities.

Be sure to code the Business Investor model as you read section 10.4. You will use it in Chapter 11 and it will form the basis for one of the team projects you will present. The other team project will use the Telemarketer model from section 13.3. This would be a good time for you and your teammates to get started on your team projects.

Errata for Chapter 10:

- Section 10.4, p. 140: The first statement in the first block of code near the top of p. 140 should be: `let potential-destinations ...` instead of `set potential-destinations ...`.

Class #12, Thu, Feb. 15: Adaptive Behavior and Objectives

Homework #10: Programming agent sensing

Turn in the following homework on Box by 11:59 pm:

- **Everyone:**
  - Railsback & Grimm, Ch. 10, Ex. 10.1, 10.2
  - Railsback & Grimm, Ch. 11, Ex. 11.1, 11.2
- **Graduate Students:**
  - Railsback & Grimm, Ch. 11, Ex. 11.3

Reading:

- Railsback & Grimm, Ch. 11.

READING NOTES:

Agents' behavior often consists of trying to achieve some **objective**.

I have discussed the way that Adam Smith's "invisible hand of the market" is a kind of agent-based view of a nation's economy: Each person (agent) has an objective of trying to maximize his or her own wealth (that's the agent's **micromotive**), and in doing so, the population of agents manages unintentionally to maximize the total wealth of the nation (an emergent **macrobehavior** that results from the collective interactions of the agents and their micromotives).

For Darwin, agents whose objectives are to survive and reproduce under changing environmental conditions achieve emergent phenomena of evolution and speciation.

If we are going to program an agent-based model to simulate such an economy (we saw this in the Sugarscape models), you need to program your agents to try to achieve their objective (maximize their wealth). There are two approaches to this:

1. You could program a sophisticated strategy into your agents.
2. You could program a simple strategy into your agents, but give them the ability to learn from their experience and adapt their behavior according to what they learn. (see section 11.3 for details and an example)

This chapter discusses different kinds of objectives you might have your agents employ. An important concept from decision theory and behavioral economics that might be new to you is **satisficing**. This

term, introduced by Herbert Simon[1] in 1956, refers to making decisions by choosing a "good-enough" option when it would take too much time and effort to determine which option is the absolute best. See section 11.4 for details and an example.

Class #13, Tue, Feb. 20: Prediction

Reading:

- Railsback & Grimm, Ch. 12.

READING NOTES:

In class, we will discuss the telemarketer model.

For the teams working on the telemarketer model, the steps are:

1. Build the model as described in the ODD.
2. Do the analyses in section 13.3.2
3. Next work on two extensions:
    1. Mergers (section 13.5)
    2. Customers remember (section 13.6)

Before class on Thursday, everyone should read Chapter 13 and the ODD for the telemarketer model and be ready to discuss the model in class. This will be a chance for the teams working on the model to ask questions.

Class #14, Thu, Feb. 22: Interaction

Reading:

- Railsback & Grimm, Ch. 13.

READING NOTES:

Errata for Chapter 13:

- Section 13.6, p. 180: Starting with version 5.0 of NetLogo, the Programming Guide no longer says that putting items on a list using `fput` is better (faster) than putting them on the end of the list using `lput`. NetLogo's lists were reprogrammed to eliminate this difference and improve the speed of adding and accessing list items.

[1] Herbert Simon (1916–2001) was a fascinating intellectual. Kind of a renaissance scholar, he made major contributions to political science, economics, cognitive psychology, and artificial intelligence. He won the Nobel Prize for economics in 1978. His publications have been cited more than 250,000 times and even fifteen years after his death, they are still cited more than 10,000 times per year.

Class #15, Tue, Feb. 27: Team Presentations

Reading:

No new reading for today.

Class #16, Thu, Mar. 1: Research Project ODDs

Reading:

No new reading for today.

Class #17, Tue, Mar. 13: Prediction

Reading:

- Railsback & Grimm, Ch. 14.

READING NOTES:

A good example of asynchronous updating is the model of breeding synchrony, described in Jovani and Grimm (2008) and in Chapter 23, which I have posted on the class web site: https://ees4760.jgilligan.org/files/models/chapter_23/Ch_23_4_breeding_synchrony.nlogo

Errata for Chapter 14:

- Section 14.2.3, p. 187: The `foreach` statement in the example code for executing turtles in size order is not compatible with NetLogo versions 6.0 and higher. For version 6.0.2, the statement should be:

  ```
  foreach sort-on [size] turtles
  [
      next-turtle -> ask next-turtle [do-sales]
  ]
  ```

- Section 14.2.5, p. 189: This code statement:

  ```
  while [count patches with [trigger-time > ticks] > 0]
  ```

  should of course be:

  ```
  while [any? patches with [trigger-time > ticks] ]
  ```

  (This Mousetrap model also has a small flaw: it does not represent the possibility that a trap scheduled to be triggered by one ball could be triggered sooner by a different ball that is launched later but has a shorter trajectory.)

Class #18, Thu, Mar. 15: Stochasticity

Reading:

- Railsback & Grimm, Ch. 15.

Class #19, Tue, Mar. 20: Collectives

Reading:

- Railsback & Grimm, Ch. 16.

READING NOTES:

Errata for Chapter 16:

- Section 16.4.1, p. 213: The description of social status is not clear: it assumes ages are real numbers instead of integers. It should read:

  _The social status of a dog can be (a) "pup," meaning its age is less than one;

  (b) "yearling," with age of 1 year; (c) "subordinate," meaning age is 2 or greater but the dog is not an alpha; (d) "alpha," the dominant individual of its sex in a pack; and (e) "disperser," meaning the dog currently belongs to a disperser group, not a pack._

- Section 16.4.1, p. 213: The first bullet of the description of pack formation should start "Determine how many times the disperser group meets another. . . "
- Section 16.4.1, Fig. 16.1: Be aware that this figure does not depict the logistic function parameters in the Wild Dog model description. The figure shows a probability of 0.5 when the number of dogs in the population is 67% of carrying capacity (40 out of 60), while the model description says the probability is 0.5 when the population is 50% of carrying capacity. To match the model description, the figure's curve should go through X = 30, Y = 0.5.
- Section 16.4.2, p. 217: Setting the dog's variable `my-pack` to the pack that created it (via the statement `set my-pack myself`) can cause a subtle error in the pack formation submodel. Disperser groups cannot form a pack with other disperser groups that originated in the same pack. In most people's code, disperser groups use the value of `my-pack` from one of their dogs to determine which pack they came from. The error can occur when two disperser groups came from packs that have since died; when a disperser dog's pack dies, NetLogo automatically changes its value of `my-pack` to the keyword `nobody`. Because disperser groups cannot form a new pack with another group that has the same value of `my-pack`, groups whose packs have died cannot combine with each other, even if they really came from two different packs. A solution is to set `my-pack` to the unique `who` number of the dog's pack. Instead of `set my-pack myself`, use `set my-pack [who] of myself`. Use a similar statement when new dogs are created in the reproduction submodel. Then change the statements where a dog's value of `my-pack` is compared to a pack, from (for example):

  ```
  set pack-members dogs with [my-pack = myself]
  ```

  to:

  ```
  set pack-members dogs with [my-pack = [who] of myself]
  ```

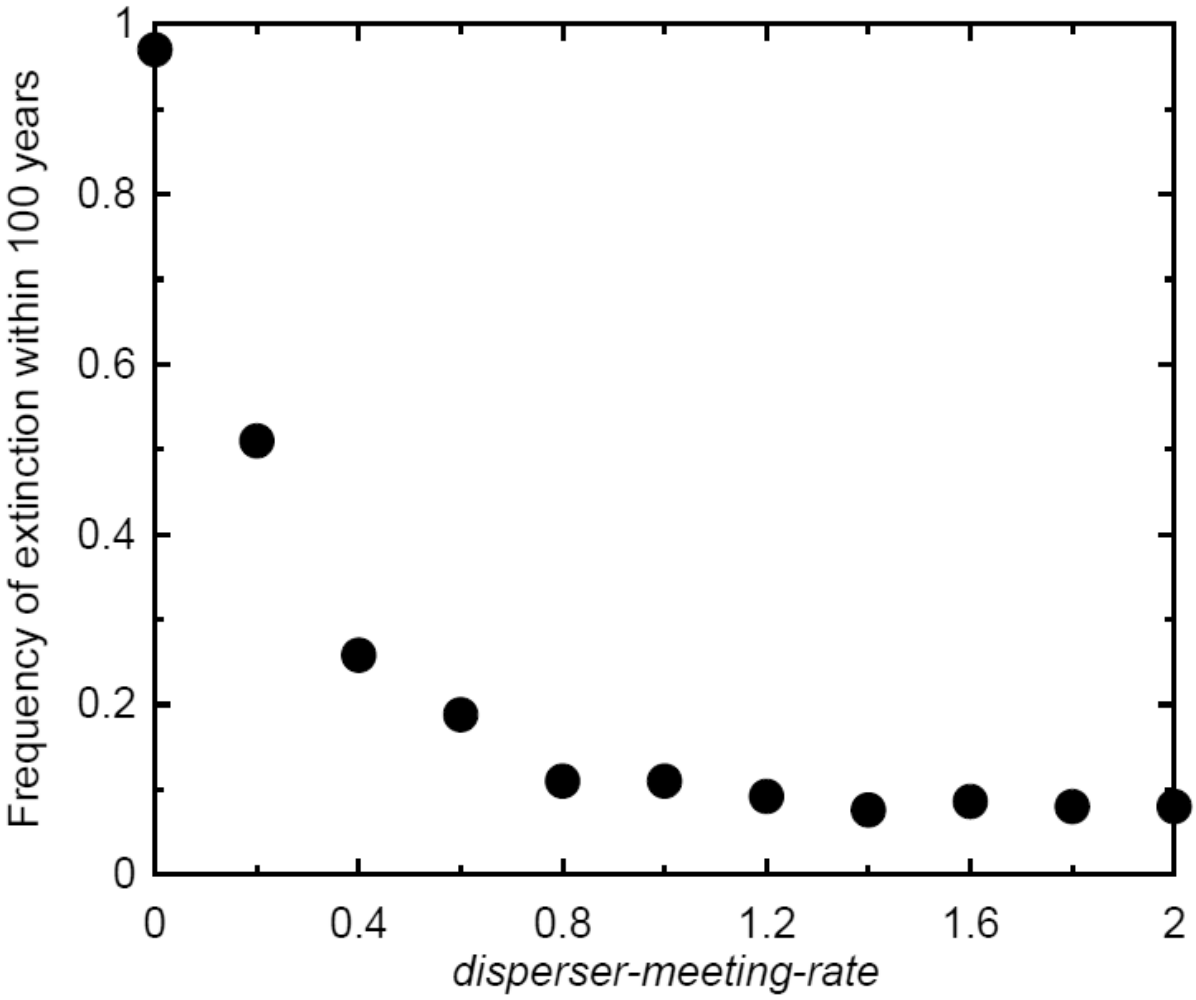

Figure 1: Corrected Fig. 16.2

And because `my-pack` now contains a who number instead of a pack, one of the last lines in the `create-disperser-group-from` code (page 220) needs to be changed, from:

```
ask dogs-former-pack [set pack-members ...
```

to:

```
ask pack dogs-former-pack [set pack-members ...
```

This error appears to have little effect on model results.

- Section 16.4.3, Figure 16.2: The figure should look like

Class #20, Thu, Mar. 22: Patterns

Reading:

- Railsback & Grimm, Ch. 17–18.

READING NOTES:

You can download the ODD for the BEFORE beech forest model, which is described in section 18.3, from the class web site: https://ees4760.jgilligan.org/files/models/chapter_18/ch18_before_ODD.pdf.

I have also posted a list of published models in which observed patterns are important: https://ees4760.jgilligan.org/files/models/chapter_18/ch18_ex1_models_list.pdf

Class #21, Tue, Mar. 27: Theory Development

Reading:

- Railsback & Grimm, Ch. 19.

READING NOTES:

There is an implementation of the wood hoopoe model, suitable for Exercise 2, on the class web site: https://ees4760.jgilligan.org/files/models/chapter_19/ch19_ex2_wood_hoopoes.nlogo

Class #22, Thu, Mar. 29: Parameterization and Calibration

Reading:

- Railsback & Grimm, Ch. 20.

READING NOTES:

Errata for Chapter 20:

- Section 20.5, p. 264: The first bullet of the description of the simple trait for how subordinate adults decide whether to scout should say, "If there are no *older* subordinate adults" instead of "other subordinate adults".
- Section 20.7, Exercise 2: The model equation has parentheses in the wrong place. It should be:

$$N_t = N_{t-1} + rN_{t-1}[1 - (N_{t-1}/K)] - H_{t-1}$$

Class #23, Tue, Apr. 3: Parameterization and Calibration 2

Reading:

No new reading for today.

Class #24, Thu, Apr. 5: Analyzing ABMs

Reading:

- Railsback & Grimm, Ch. 22.

Class #25, Tue, Apr. 10: Sensitivity and Robustness

Reading:

- Railsback & Grimm, Ch. 23.

READING NOTES:

You can download the model of breeding synchronyCh_23_4_breeding_synchrony.nlogo), described in Jovani and Grimm (2008) and in section 23.4, from the class web site: https://ees4760.jgilligan.org/files/models/chapter_23/)Ch_23_4_breeding_synchrony.nlogo

Class #26, Thu, Apr. 12: Looking Ahead: ABMs Beyond this Course

Reading:

- Railsback & Grimm, Ch. 24.

Class #27, Tue, Apr. 17: Presentations

Reading:

No new reading for today.

Class #28, Thu, Apr. 19: Presentations

Reading:

No new reading for today.