# Agent-Based Modeling and Simulation

## Implementing a First Agent-Based Model

Dr. Alejandro Guerra-Hernández

**Universidad Veracruzana**
Centro de Investigación en Inteligencia Artificial
*Sebastián Camacho No. 5, Xalapa, Ver., México 91000*

mailto:aguerra@uv.mx
http://www.uv.mx/personal/aguerra

August 2019 - January 2020

# Credits

- These slides are completely based on the book of Railsback and Grimm [1], chapter 5.
- Any difference with this source is my responsibility.

Universidad Veracruzana

# Programming in NetLogo

- We continue your lessons in NetLogo, but from now on the focus will be on programming and using real ABMs that address real scientific questions.
- The Mushroom Hunt model of session 2 was neither very agent-based nor scientific, in ways we discuss in this chapter.
- You are going to start actually using an ABM to produce and analyze meaningful output and address scientific questions.

Universidad Veracruzana

# Learning Objectives

- Understand how to translate a model from its written description in ODD format into NetLogo code.
- Understand how to define global, turtle, and patch variables.
- Become familiar with NetLogo's most important primitives, such as ask, set, let, create-turtles, ifelse, and one-of.
- Start learning good programming practices, such as making very small changes and constantly checking them, and writing comments in your code.
- Produce your own software for the Butterfly model described in session 4.

Universidad Veracruzana

# ODD Protocol

- We have introduce de ODD Protocol for descibing an ABM, and as an example provided the ODD formulation of a butterfly hilltopping model.
- What do we do when it is time to make a model described in ODD actually run in NegLogo?
- It turns out to be quiet straightforward because the organizations of ODD and NetLogo correspond closely.
- The major elements of an ODD formulation have corresponding elements in NetLogo.

Universidad Veracruzana

# Purpose

- From now on, we will include the ODD descriptions of our ABMs on NetLogo's Information tab.
- These descriptions will the start with a short statement of the model's overall purpose.

Universidad Veracruzana

# Entities, State Variables, and Scales

- Basic entities for ABMs are built into NetLogo: the World of square patches, turtles as mobile agents, and the observer.
- The state variables of the turtles and patches, and perhaps other types of agents, are defined via **turtles−own** [ ] and **patches−own** [ ] statements.
- The variables characterizing the global environment are defined in the **globals** [ ] statement.
- In NetLogo, as in ODD, these variables are defined right at the start.

Universidad Veracruzana

# Process Overview and Scheduling

- This, exactly, is represented in the go procedure.
- Because a well designed go simply calls other procedures that implement all the submodels, it provides an overview (but not the detailed implementation) of all processes, and
- specifies their schedule, that is, the sequence in which they are executed each tick.

Universidad Veracruzana

# Design Concepts

- These concepts describe the decisions made in designing a model and so do not appear directly in the NetLogo code.
- However, NetLogo provides many primitives and interface tools to support these concepts.

Universidad Veracruzana

# Initialization

- This corresponds to an element of every NetLogo program, the setup procedure.
- Pushing the setup button should do everything described in the Initialization element of ODD.

Universidad Veracruzana

# Input Data

- If the model uses a time series of data to describe the environment, the program can use NetLogo's input primitives to read the data from a file.

Universidad Veracruzana

# Submodels

- The submodels of ODD correspond closely but not exactly to procedures in NetLogo.
- Each of a model's submodels should be coded in a separate NetLogo procedure that is then called from the go procedure.
- Sometimes, though, it is convenient to break a complex submodel into several smaller procedures.
- These correspondences between ODD and NetLogo make writing a program from a model's ODD formulation easy and straightforward.

Universidad Veracruzana

# Hierarchical, step-by-step Development

- Program the overall structure of a model first, before starting any of the details. This keeps you from getting lost in the details early. Once the overall structure is in place, add the details one at a time.
- Before adding each new element (a procedure, a variable, an algorithm requiring complex code), conduct some basic tests of the existing code and save the file. This way, you always proceed from firm ground: if a problem suddenly arises, it very likely (although not always) was caused by the last little change you made.
- First, let us create a new NetLogo program, save it, and include the ODD description of the model on the Information tab.

Universidad Veracruzana

# Excercise I

- Start NetLogo and use File/New to create a new NetLogo program. Use File/Save to save the program under the name `butterfly-corridors.nlogo` in an appropriate folder.
- Go to the page of Grimm's book:

  `http://www.railsback-grimm-abm-book.com`

- Download de ODD description of the butterfly model (chapter 4).
- Go to the information tab in NetLogo, click the Edit button, and paste in the model description accordingly.

Universidad Veracruzana

# Excercise II

- Let us implement first the entities, state variables, and scales part of the model.
- Go to the procedure tab and insert:

```
globals [ ]
patches-own [ ]
turtles-own [ ]
```

- Click on the check button, there should be no error message.
- Since butterflies have no other state variables other than their location, we do not need to define new variables for the turtles.

Universidad Veracruzana

# Excercise III

- But patches have a variable for elevation, insert:

```
patches-own [ elevation ]
```

- NetLogo infers the type of the variables from the first value assigned via the primitive `set`.
- Now, go to the interface tab, click the Settings button, and change "Location of origin" to Corner and Bottom Left; change the number of columns and rows to 149. Turn off the two world wrap tick boxes, so that our model has closed boundaries.
- If the world is too big, click again the Settings button, and set the "Patch size" to 3 or so.
- Save the changes.

Universidad Veracruzana

# Excercise IV

- At the end of the existing program, insert this:

```
to setup
  ca
  ask patches
  [

  ]
  reset-ticks
end
```

- Click the Check button again to make sure the syntax of this code is correct.
- Assigning elevations to the patches will create a topographical landscape for the butterflies to move in. What should the landscape look like?

Universidad Veracruzana

# Excercise V

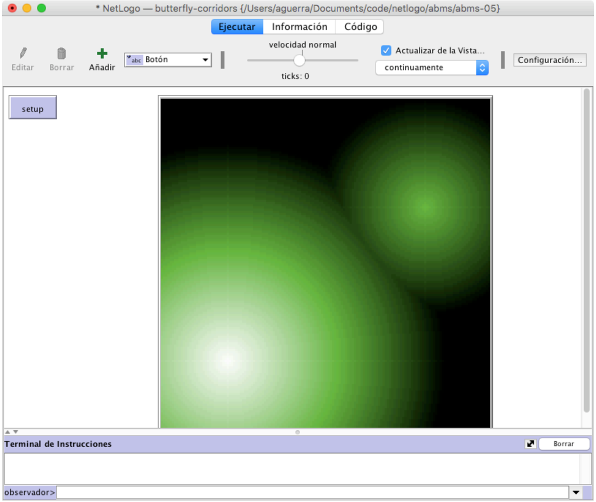
- The ODD description is incomplete: it simply says we start with a simple artificial topography.
- To start it is a good idea to create simple scenarios for easily predicting what should happen. Creating two hills will do:

```
ask patches
[
  let elev1 100 - distancexy 30 30
  let elev2 50 - distancexy 120 100

  ifelse elev1 > elev2
  [ set elevation elev1 ]
  [ set elevation elev2 ]

  set pcolor scale-color green elevation 0 100
]
```

Universidad Veracruzana

# Excercise VI

- Add the corresponding button to setup the model:




Universidad Veracruzana

# Excercise VII

- Now, lets create some agents. Enter the following code after the ask patches statement:

```
crt 1
[
 set size 2
 setxy 85 95
]
```

- Setup the model again to check if everything is fine.
- Let us implement the schedule:

```
to go
  ask turtles [ move ]
end
```

Universidad Veracruzana

# Excercise VIII

- Add the skeleton for `move` to avoid errors:

```
to move

end
```

- The go procedure must be enhanced to stop after 1000 steps, accordingly to the ODD description:

```
to go
  ask turtles [ move ]
  tick
  if ticks >= 1000 [ stop ]
end
```

- Add the go button.

# Excercise IX

- Let us implement the submodel for moving:

```
to move
  ifelse random-float 1 < q
  [ uphill elevation ]
  [ move-to one-of neighbors ]
end
```

- Of course, you need to add *q* to the globals:

```
globals [ q ]
```

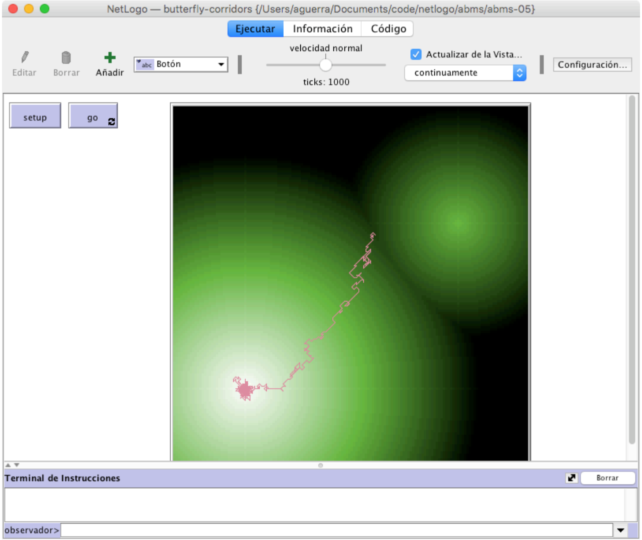
- and set it to, e.g., 0.4, in the `setup`:

```
set q 0.4
```

- Try your model, using **pen−down**

Universidad Veracruzana

# Excercise X




Universidad Veracruzana

# Missing issues

1. Comments.
2. Observations.
3. A realistic landscape.
4. An analysis of the model.

- The last three are adressed in the next session.

Universidad Veracruzana

# Comments

- Comments are any text following a semicolon (;) on the same line in the code.
- Such text is ignored by NetLogo and instead is for people.
- Comments are needed to make code easier for others to understand, but they are also very useful to ourselves: after a few days or weeks go by, you might not remember why you wrote some part of your program as you did instead of in some other way.
- Putting a comment at the start of each procedure saying whether the procedure is in turtle, patch, or observer context helps you write the procedures by making you think about their context, and it makes revisions easier.

Universidad Veracruzana

# Use Comments For

- Briefly describe what each procedure or nontrivial piece of the code is supposed to do;
- Explain the meaning of variables;
- Document the context of each procedure; Keep track of what code block or procedure is ended by "]" or end;
- and In long programs, visually separate procedures from each other by using comment lines like this:

```
; -------------------------------------
```

- To temporarily deactivate code statements.

Universidad Veracruzana

# Don't Use For

- Detailed and lengthy comments are no substitute for code that is clearly written and easy to read!
- Especially in NetLogo, you should strive to write your code so you do not need many comments to understand it.
- Use names for variables and procedures that are descriptive and make your code statements read like human language.
- Use tabs and blank lines to show the code's organization.

Universidad Veracruzana

# Example

```
to move ; The butterfly move procedure
  ifelse random-float 1 < q ; q is the probability of moving uphill straighfordwardly.
  [ uphill elevation ] ; move deterministically uphill
  [ move-to one-of neighbors ] ; move randolmly around current location.
end ; end of move
```

Universidad Veracruzana

# Observations

- So far, the model only produces visual output, which let us look for obvious mistakes and see how the buttefly behaves.
- But for use the model for its scientific purpose –understanding the emergence of virtual corridors, we need additional outputs that quantify the width of the corridor used by a large number of butterflies.

Universidad Veracruzana

# Landscape

- In order to make more scientific we need a landscape model.
- It is good to start programming and model testing and analysis with artificial scenarios, but we do not want to restrict our analysis to such cases.

Universidad Veracruzana

# Analysis

- We have not yet done any analysis on this model, e.g., to see how the parameter $q$ affects butterfly movemente and the appearance of the virtual corridors.
- For now, play with the model asking What if...

Universidad Veracruzana

# Modeling and Coding

- Since this course is about ABMs, we revisited the ODD protocol for describing them,
- Showing how we can quiet directly translate an ODD description into a NetLogo program.
- In scientific modeling we start by thinking about and writing down the model design; ODD provides a productive, standard way to do this.
- Then, when we think we have enough of a design to implement on a computer, we translate it into code so we can start testing and revising the model.

Universidad Veracruzana

# Correspondance

- Although developed independently, NetLogo and the ODD protocol have many similarities and correspond quiet closely.
- Both of them were developed by looking for the key characteristics of ABMs in general and the basic ways that they are different from other kinds of model (and, therefore, ways that their software must be unique).
- These key characteristics were used to organize both ODD and NetLogo, so it is natural that they correspond with each other.

Universidad Veracruzana

# Development techniques

1. Start modeling with the simplest version of the model conceivable –ignore many, if not most, of the components and processes you expect to include later.
2. Develop programs in a hierarchical way: start with the skeletons of structures (procedures, ask commands, ifelse switches, etc.); test these skeletons for syntax errors; and only then, step by step, add "flesh to the bones" of the skeleton.
3. If your model will eventually include a complex or realistic environment, start with a simplified artificial one.
4. Formatting your code nicely and providing appropriate comments is well worth the tiny bit of time it takes.

Universidad Veracruzana

# Conceptual issues

- We wanted to develop a model that helps us understanding how and where in a landscape virtual corridors of butterfly movement appear.
- The hypothesis is that corridors are not necessarily linked to landscape features that are specially suitable for migration, but can emerge from the interaction between topography and the movement decisions of the butterflies.
- We represented these decisions in a most simple way: by telling the butterflies to move uphill, but with their variability in movement represented by the parameter $q$.
- The first results from a highly artificial landscape indicate that indeed our movement rule has the potential to produce virtual corridors, but we obviously have more to do.

Universidad Veracruzana

# Referencias I

SF Railsback and V Grimm. *Agent-Based and Individual-Based Modeling*. Princeton, New Jersey, USA: Princeton University Press, 2012.

Universidad Veracruzana