

# Agent-Based Modeling and Simulation

## From Animations to Science

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



Universidad Veracruzana

# Credits

- ▶ These slides are completely based on the book of Railsback and Grimm [2], chapter 5.
- ▶ Any difference with this source is my responsibility.



Universidad Veracruzana

# Using and documenting the model

- ▶ Beginners often believe that modeling is mainly about **formulating** and **implementing** models. But real work starts **after** a model has first been implemented.
- ▶ Then we use the model to find **answers** and **solutions** to the questions and problems we started our modeling project with, which almost always requires modifying the model formulation and software.
- ▶ Usually, the **iterative** modeling cycle is not documented. Instead, models are typically presented as **static entities** that were just produced and used. In fact, every model description is only a **snapshot** of a process.



Universidad Veracruzana

# The NetLogo case

- ▶ The Models Library of NetLogo has a similar problem: it presents the models and gives (on the Information tab) some hints of how to analyze them, but it cannot demonstrate **how to do science** with them.
- ▶ These models are very good at animation: letting us see what happens as their assumptions and equations are executed. But they do not show you how to explore **ideas and concepts**, develop and test **hypotheses**, and look for parsimonious and general **explanations** of observed phenomena.



Universidad Veracruzana

# The Butterfly model enhanced I

- ▶ We will illustrate how to make a NetLogo program into a **scientific model** instead of just a simulator, by taking the Butterfly model and adding the things needed to do science.
- ▶ Remember that the **purpose** of this model was to explore the emergence of “virtual corridors”: places where butterflies move in high concentrations even though there is nothing there that attracts butterflies.
- ▶ Our model so far simulates butterfly movement, but does not tell us anything about corridors and when and how strongly they emerge.
- ▶ Therefore, we will now produce **quantitative output** that can be analyzed, instead of just the visual display of butterfly movement.



Universidad Veracruzana

# The Butterfly model enhanced II

- ▶ We will also replace our very simple and artificial landscape with a **real one** read in from a topography file. In the exercises we suggest some scientific analyses for you to conduct on the Butterfly model.



Universidad Veracruzana

# Learning Objectives I

- ▶ Learn the importance of **version control**: saving documented versions of your programs whenever you start making substantial changes.
- ▶ Understand the concept that using an ABM for science requires producing **quantitative output** and conducting simulation experiments; and execute your first simulation experiment.
- ▶ Learn to define and initialize a global variable by creating a **slider** or **switch** on the Interface.
- ▶ Develop an understanding of what **reporters** are and how to write them.
- ▶ Start learning to **find and fix** mistakes in your code.
- ▶ Learn to create output by writing to an **output window**, creating a time-series plot, and exporting plot results to a file.



Universidad Veracruzana

# Learning Objectives II

- ▶ Try a simple way to **import data** into NetLogo, creating a version of the Butterfly model that uses real topographic data.



Universidad Veracruzana



# What is a corridor?

- ▶ Our Butterfly model is not ready to address its scientific purpose in part because it lacks a way to **quantitatively** observe the extent to which virtual corridors emerge.
- ▶ But how would we **characterize** a corridor? Obviously, if all individuals followed the same path (as when they all start at the same place and  $q$  is 1.0) the corridor would be very narrow; or if movement were completely random we would not expect to identify any corridor-like feature.
- ▶ But we need to quantify how the **width** of movement paths changes as we vary things such as  $q$  or the landscape topography. Let's do something about this problem.



Universidad Veracruzana

# Version Control

- ▶ But first, because we are about to make a major change in the program:
  - ▶ Create and save a new version of your butterfly software.
- ▶ Or use a version control system, like **git**:

<https://git-scm.com>



Universidad Veracruzana

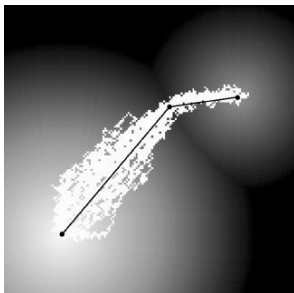
# Pe'er definitions revised

- ▶ Pe'Er, Saltz, and Frank [1] quantified corridor width by dividing the number of patches visited by all individuals during 1000 time steps by the distance between the start patch and the hill's summit.
- ▶ In our version of the model, different butterflies can start and end in different places, so we **slightly modify** this measure:
  - ▶ We will assume that each butterfly stops when it reaches a local hilltop (a patch higher than all its eight neighbor patches).
  - ▶ Then we will quantify the width of the corridor used by all butterflies as (a) the number of patches that are visited by any butterflies divided by (b) the mean distance between starting and ending locations, over all butterflies.
- ▶ This measure will be small (approaching 1.0) when all butterflies follow the same, straight path uphill; but should increase as they increasingly follow different paths.



Universidad Veracruzana

# Graphically



- ▶ 50 butterflies starting at 85,95 with  $q = 0.4$ .
- ▶ The number of white patches is 1956.
- ▶ The mean distance between butterfly starting and ending points is 79.2 patches.
- ▶ The corridor width is 24.7 patches.



Universidad Veracruzana

# Analysis

- ▶ We are going to produce a **plot of corridor width vs.  $q$** .
- ▶ This is important: When analyzing a model, we need to have a **clear idea** of that kind of plot we want to produce from what output, because this tells us what **kind of simulation experiments** we have to perform and what **outputs** we need the program to produce.



# Implementation

- ▶ Because it is now obvious that we need to conduct experiments by varying  $q$  and seeing its effect, create a **slider** for it on the Interface tab. Set the slider so that  $q$  varies from 0.0 to 1.0 in increments of 0.01.
- ▶ Change the setup procedure so that 50 individuals are created and start from the same position. Then vary  $q$  via its slider and observe how the **area** covered by all the butterfly paths changes.



# Some tips about using sliders

- ▶ Only global variables can be controlled by sliders, switches, and choosers; not turtle or patch variables.
- ▶ Once a controller is created for the global variable, it cannot still appear in the **globals** statement. The controller defines and initializes the variable.
- ▶ We recommend to comment the variable to remember it exists.
- ▶ The biggest potential problem is forgetting to remove statements setting the variable's value in the `setup` procedure. It will take always the same value.



# Implementation

- Modify the move procedure so that butterflies **no longer move** in they are at a hilltop, i.e., when the butterfly is in a patch with elevation higher than all its neighbors. Add this at the beginning of move:

```
1 | if elevation >= [elevation] of max-one-of neighbors  
   |   [elevation] ;; already in a hilltop  
2 | [stop]
```

- Observe that turtles can access the variables of its current patch.
- Observe the use of elevation as a reporter in **max-one-of**.



Universidad Veracruzana



# More changes

- ▶ Add a new variable to the **patches—own** statement: `used?`. This is a boolean variable, so we follow the NetLogo convention of ending such variables with a question mark. This variable will be true if any butterfly has landed in that patch.
- ▶ Add a variable called `start-patch` to the **turtles—own** statement.
- ▶ In the setup procedure, add statements to initialize these new variables: `used?` is set as false; `start-patch` can use the primitive **patch-here**.
- ▶ In NetLogo new variables have zero as value.



# Keeping track of visited patches

- ▶ Add a statement to the `move` procedure that causes the butterfly, once it has moved to a new patch, to set the patch's variable `used?` to true.



Universidad Veracruzana

# Calculating the corridor width

- ▶ In the go procedure's that stops the program after 1000 ticks, insert a statement that executes once before execution stops.
- ▶ This procedure uses the very important primitive `let` to create a new local variable `final-corridor-width` and give it the value produced by a new reporter `corridor-width`.
- ▶ Write the skeleton of a reporter procedure `corridor-width`, that reports the mean path width of turtles. Look up the keyword `to-report` in the NetLogo Dictionary and read about reporters in the Procedures section of the Programming Guide.



Universidad Veracruzana

# The corridor-width procedure

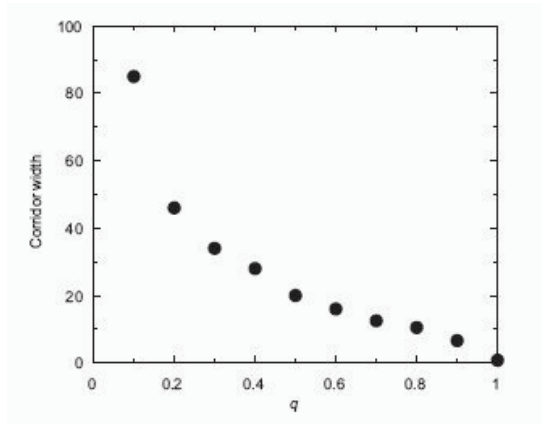
- ▶ Create a new local variable and set it to the number of patches that have been visited at least once. (Hint: use the primitive `count`.)
- ▶ Also create a new local variable that is the mean, over all turtles, of the distance from the turtle's current patch and its starting patch. (Look at the primitives `mean` and `distance`.)
- ▶ From the above two new local variables, calculate corridor width and report its value as the result of the procedure.
- ▶ In the `go` procedure, after setting the value of `final-corridor-width` by calling `corridor-width`, print its value in an `output window` that you add to the Interface. Do not only print the value of `final-corridor-width` itself, but also the text "Corridor width:".



Universidad Veracruzana

# Playing with $q$

- ▶ If you use the slider to vary  $q$  over a wide range, write down the resulting corridor widths, and then plot the corridor width versus  $q$ :



Universidad Veracruzana

# Surprises

- ▶ These results are not very surprising because they mainly show that with **less randomness** in the butterflies' movement decisions, movement is straighter and therefore corridor width smaller.
- ▶ It seems likely that a **less-artificial** landscape would produce more interesting results.
- ▶ There is one little surprise in the results: corridor width is well below 1.0 (about 0.76) when  $q$  is 1.0. How is it possible that the number of patches visited is 24% less than the distance between starting and ending patches? Do the butterflies escape the confines of Euclidean geometry and find a path that's shorter than the straight-line distance? Is there something wrong with the distance primitive?



# Plotting

- ▶ Now let's try something we often need to do: examine results **over time** as the simulation proceeds instead of only at the end of the model run.
- ▶ **Plots** are extremely useful for observing results as a model executes.
- ▶ However, we still need results written down so we can analyze them, and we certainly do not want to write down results for every time step from a plot or the output window. Instead, we need NetLogo to write results out in a **file** that we can analyze.



Universidad Veracruzana

# Updating corridor width

- ▶ But before we can add any of these time-series outputs, we have to change the model so it produces results **every time** step.
- ▶ Currently we calculate corridor width only once, at the end of a simulation; now we need to calculate it each time step:

```
1 to go ; This is the master schedule
2   ask turtles [ move ]
3   plot corridor-width
4   tick
5   if ticks >= 1000 [ stop ]
6 end
```



Universidad Veracruzana



# Plot

- ▶ The `plot` statement does two things:
  - ▶ It calls the `corridor-width` reporter to **get the current value** of the corridor width, and then
  - ▶ It sends that value to be the **next point** of a plot of the Interface tab.
- ▶ Add a plot named "Corridor width" to the Interface tab.
- ▶ The code for plotting can be written in the plot object, but it is preferable to put in the code tab, where it is visible and easily modifiable.
- ▶ Test-run the model with several values of  $q$ .



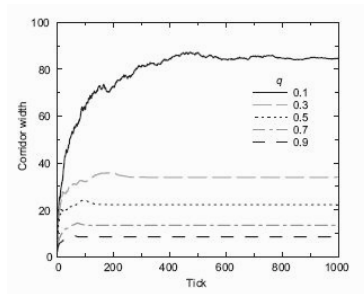
Universidad Veracruzana

# Exporting data

- ▶ Looking at the plot gives us an **idea** of how the corridor width changes with  $q$  and over time, but we need to **export** data:

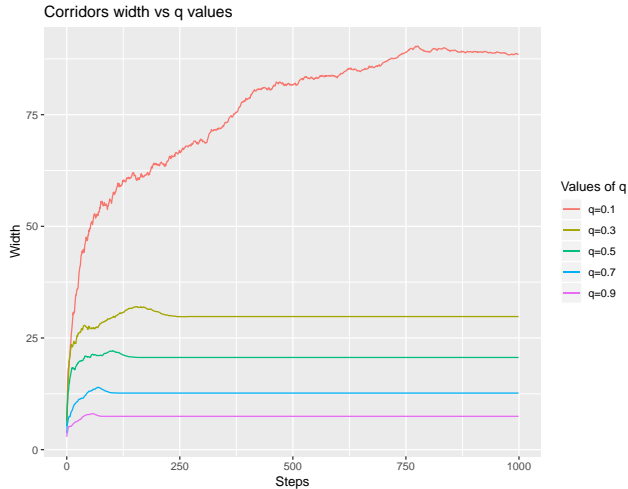
```
1 | export-plot "Corridor width" word  
   "Corridor-output-for-q-" q
```

- ▶ With a **spreadsheet** you can graph the changes over time for several values of  $q$ :



Universidad Veracruzana

# My graph with Excel + R



Universidad Veracruzana

# R Code

```
1 | qs <- read.csv(file = "corridor-width-qs.csv", header = TRUE, sep = ",")
2 |
3 | ggplot(qs,aes(step,q.0.1)) +
4 |   geom_line(aes(color="q=0.1")) +
5 |   geom_line(data=qs,aes(step,q.0.3,color="q=0.3")) +
6 |   geom_line(data=qs,aes(step,q.0.5,color="q=0.5")) +
7 |   geom_line(data=qs,aes(step,q.0.7,color="q=0.7")) +
8 |   geom_line(data=qs,aes(step,q.0.9,color="q=0.9")) +
9 |   labs(title="Corridors width vs q values", color="Values of q", x = "Steps",
         |     y = "Width")
```



# Grid-based Spatial Data

- ▶ Importing topographies of real landscapes (or any other spatial data) into a NetLogo program is **straightforward** –once the data are **prepared** for NetLogo.
- ▶ On this book's web site, we provide a **plain text** file that you can import to NetLogo and use as the landscape: `ElevationData.txt`
- ▶ The file contains the **mean elevation** of 25-meter patches.
- ▶ By **grid-based**, we mean data records that include an x-y coordinates and a variable value (elevation or anything else that varies over space) for points or cells evenly spaced in both x and y directions as NetLogo patches are.



Universidad Veracruzana

# Transformations

- ▶ Real data are typically in **coordinate systems** (e.g., UTM) that cannot be used directly by NetLogo because NetLogo requires that patch coordinates be integers and that the World includes the location 0,0.
- ▶ The **transformations** could be made via NetLogo code as the data are read in, but it is safest to do it in software such as a spreadsheet that lets you see and test the data before they go into NetLogo.
- ▶ These steps will prepare an input file that provides the value of a spatial variable to each patch. (It is easily modified to read in several patch variables at once.)



Universidad Veracruzana

# Origin

- ▶ NetLogo **requires** the point 0,0 to be in the World.
- ▶ It is easiest to put 0,0 at the lower left corner by identifying the **minimum** x-and y-coordinates in the original data, and then **subtracting** their values from the x-and y-coordinates of all data points.



Universidad Veracruzana

# Distances

- ▶ NetLogo patches are **one distance unit** apart, so the data must be transformed so points are exactly 1 unit apart.
- ▶ Divide the x-and y-coordinates of each point by the spatial resolution (grid size, the distance between points), in the same units that the data are in.
- ▶ **Example:** If the points represent squares 5 meters on a side and the coordinates are in meters, then divide the x-and y-coordinates by 5.
- ▶ As a result, all coordinates should now be **integers**, and grid coordinates should start at 0 and increase by 1.
- ▶ Save the data as a plain text file.



Universidad Veracruzana



# The World

- ▶ In NetLogo's Model Settings window, set the **dimensions** to match those of your data set (or, in the setup procedure, use the primitive `resize-world`).
- ▶ Turn **world wrapping off** (at least temporarily) so that NetLogo will tell you if one of the points in the input file is outside the World's extent.



Universidad Veracruzana

# Reading data

- The code is something like this:

```
1 file-open "ElevationData.txt"
2 while [not file-at-end?]
3 [
4   let next-x file-read
5   let next-y file-read
6   let next-elevation file-read
7   ask patch next-x next-y
8     [set elevation next-elevation]
9 ]
10 file-close
```



Universidad Veracruzana

# Exercise

- ▶ Create a new version of your model and change its setup procedure so it reads in elevations from the file `ElevationData.txt`.
- ▶ Look at the elevation data file to determine what dimensions your World should have.
- ▶ Change the statement that shades the patches by their elevation so the color is scaled between the minimum and maximum elevation (See, `min` and `max`).



# Last Changes

- ▶ Comment out the statement causing butterflies to stop if they are at a local hilltop. They will stop too soon in the real landscape.
- ▶ In the `crt` block, assign `xcor` and `ycor` randomly in a 10 by 10 neighborhood, located somewhere you choose. Butterflies won't start any more at the same place.



Universidad Veracruzana

# Small steps

- ▶ Modeling a system of multiple agents.
- ▶ Adding quantitative observations that we can analyse.
- ▶ Starting to use simulation experiments.
- ▶ Using real spatial data.



Universidad Veracruzana

# The Butterfly Model

- ▶ The model is extremely simple, but it already requires some difficult decision.
- ▶ **Example:** Several different ways you could define widths and corridor sizes, which likely would produce different results.
- ▶ It is thus important to learn how to analyse simple models before you turn to more complex ones.
- ▶ The model is good to start because people in all disciplines can understand it, even when it has been used in serious ecological research. But remember that ABMs are not restricted to organisms moving in landscapes.



Universidad Veracruzana

# Referencias I



G Pe'er, D Saltz, and K Frank. "Virtual corridors for conservation management". In: *Conservation Biology* 19.6 (2005), pp. 1997–2003.



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



Universidad Veracruzana