

Agent-Based Modeling and Simulation

Getting Started with NetLogo


Dr. Alejandro Guerra-Hernández

Instituto de Investigaciones en Inteligencia Artificial
Universidad Veracruzana
*Campus Sur, Calle Paseo Lote II, Sección Segunda No 112,
Nuevo Xalapa, Xalapa, Ver., México 91097*
<mailto:aguerra@uv.mx>
<https://www.uv.mx/personal/aguerra/abms>

Doctorado en Inteligencia Artificial 2024



Credits

- ▶ These slides are based on the book of Railsback and Grimm [1], chapter 1.
- ▶ Any difference with this source is my responsibility.
- ▶ This work is licensed under [CC-BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/) 
- ▶ To view a copy of this license, visit:

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

Introduction

- ▶ We expect you to learn the basic elements of NetLogo mainly by using its excellent [User Manual](#), which includes tutorials, detailed guides to the interface and programming, and the all-important [NetLogo Dictionary](#).
- ▶ Keep in mind that this course is about [agent-based modeling](#), not just NetLogo.

Objectives

- ▶ Be familiar with the User Manual and its parts, including the **Interface and Programming Guides** and the **NetLogo Dictionary**.
- ▶ Complete the three **tutorials** in NetLogo's User Manual.
- ▶ Know what the **execute, information, and code tabs** are.
- ▶ Understand how to add **buttons** to the interface and to use the **world** and change its settings.
- ▶ Know the four built-in **types of agent**: observer, patches, turtles, and links.
- ▶ Be familiar with NetLogo's basic **code elements**: procedures, primitives, commands, and reporters; and the basic organization of a NetLogo program, including the **setup** and **go** procedures.
- ▶ Have **programmed** a first, very simple model.

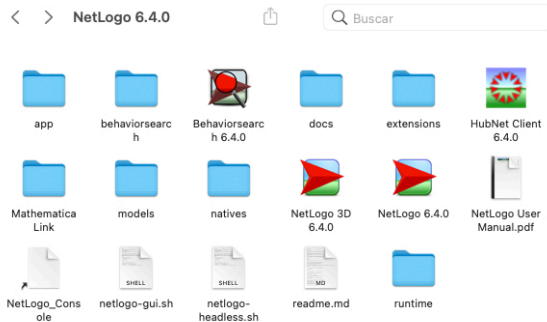


Installation

- ▶ Go to the NetLogo web site:

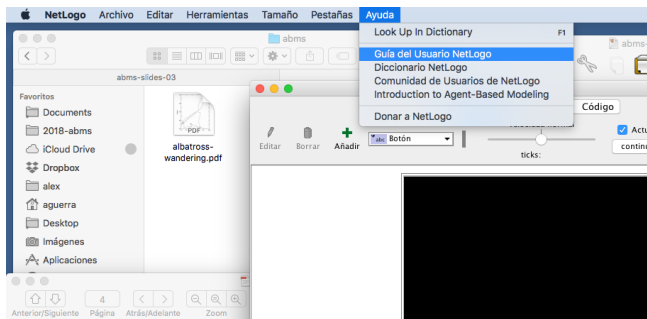
<https://ccl.northwestern.edu/netlogo/>

- ▶ Current version is 6.4.0



Help

- ▶ Start NetLogo, click on “Help” and then “NetLogo User Manual.” This opens NetLogo’s extensive documentation, which appears in your web browser.



User's manual

NetLogo
User Manual
version 6.0.4
June 4, 2018

Release Notes
System Requirements
Contacting Us
Copyright / License

Introduction
[What is NetLogo?](#)
[Sample Model: Party](#)

Learning NetLogo
[Tutorial #1: Models](#)
[Tutorial #2: Commands](#)
[Tutorial #3: Procedures](#)

Reference
[Interface Guide](#)
[Interface Tab Guide](#)
[Info Tab Guide](#)
[Code Tab Guide](#)
[Programming Guide](#)
[Transition Guide](#)
[NetLogo Dictionary \(en Español\)](#)

Features
[Shapes Editor](#)
[BehaviorSpace](#)
[System Dynamics](#)
[HubNet](#)
[HubNet Authoring](#)
[Logging](#)
[Controlling](#)
[Mathematica Link](#)
[NetLogo 3D](#)
[Save to Modeling Commons](#)

Extensions
[Extensions Guide](#)
[Archives](#)
[Array](#)
[Bitmaps](#)
[Control Flow](#)
[CSV](#)
[GIS](#)
[GoGo](#)
[LevelSpace](#)
[Matrices](#)
[Networks](#)
[Palette](#)
[Profiler](#)
[R](#)

What is NetLogo?

NetLogo is a programmable modeling environment for simulating natural and social phenomena. It was authored by Uri Wilensky in 1999 and has been in continuous development ever since at the Center for Connected Learning and Computer-Based Modeling.

NetLogo is particularly well suited for modeling complex systems developing over time. Modelers can give instructions to hundreds or thousands of "agents" all operating independently. This makes it possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from their interaction.

NetLogo lets students open simulations and "play" with them, exploring their behavior under various conditions. It is also an authoring environment which enables students, teachers and curriculum developers to create their own models. NetLogo is simple enough for students and teachers, yet advanced enough to serve as a powerful tool for researchers in many fields.

NetLogo has extensive documentation and tutorials. It also comes with the Models Library, a large collection of pre-written simulations that can be used and modified. These simulations address content areas in the natural and social sciences including biology and medicine, physics and chemistry, mathematics and computer science, and economics and social psychology. Several model-based inquiry curricula using NetLogo are available and more are under development.

NetLogo is the next generation of the series of multi-agent modeling languages including StarLogo and StarLogo.T. NetLogo runs on the Java Virtual Machine, so it works on all major platforms (Mac, Windows, Linux, et al). It is run as a desktop application. Command line operation is also supported.

Features

- System:
 - Free, [open source](#)
 - Cross-platform: runs on Mac, Windows, Linux, et al
 - International character set support

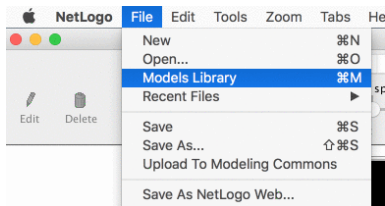


Relevant parts

- ▶ Introduction
- ▶ Learning NetLogo
- ▶ Interface Guide
- ▶ Programming Guide

Models Library

- ▶ Open the Models Library from the File menu:



- ▶ Choose "Wolf Sheep Predation" from the Biology section and press "Open".

The interface

NetLogo — Wolf Sheep Predation

Ejecutar Información Código

velocidad normal

ticks:

Actualizar de la Vista...

manualmente (ticks)

Configuración...

model-version
sheep-wolves

initial-number-sheep 100 initial-number-wolves 50

grass-regrowth-time 30

setup go

Sheep settings Wolf settings

sheep-gain-from-food 4 wolf-gain-from-food 20

sheep-reproduce 4% wolf-reproduce 5%

On show-energy?
 Off

sheep	wolves	grass
0	0	N/A

populations

pop. 100 0

time 0 100

sheep
wolves
grass / 4

Terminal de Instrucciones

Borrar

observador>

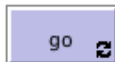


Excercise

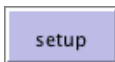
- ▶ Press the "setup" button.
- ▶ Q. What do you see appear in the view?
- ▶ Press the "go" button.
- ▶ Q. As the model is running, what is happening to the wolf and sheep populations?
- ▶ Press the "go" button to stop the simulation.

Controlling the Model: Buttons

- ▶ When a button is pressed, the model responds with an **action**.
- ▶ A button can be an **once** button, or a **forever** button. Forever buttons have two arrows in the bottom right corners, like this:



- ▶ Once buttons don't have the arrows, like this:



- ▶ Once buttons do one action and then stop. When the action is finished, the button pops back up.
- ▶ Forever buttons do an action over and over again. When you want the action to stop, press the button again.

Common buttons

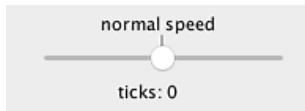
- ▶ Most models, including Wolf Sheep Predation, have a once button called **setup** and a forever button called **go**.
- ▶ Many models also have a once button called **go once** or **step once** which is like “go” except that it advances the model by one tick (time step) instead of over and over. Using a once button like this lets you watch the progress of the model more closely.
- ▶ Stopping a forever button is the normal way to pause or stop a model. After pausing you can make it resume by pressing the button again.
- ▶ Don't use **halt** unless necessary.

Excercise

- ▶ If you like, experiment with the “setup” and “go” buttons in the Wolf Sheep Predation model.
- ▶ Q. Do you ever get different results if you run the model several times with the same settings?

Controlling speed: Speed Slider

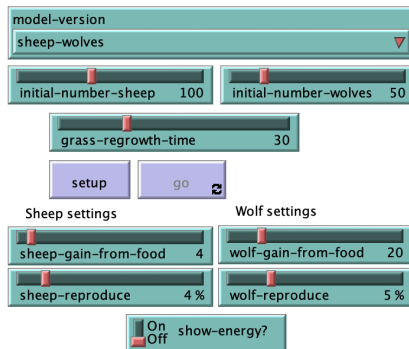
- ▶ The speed slider allows you to control the **speed** of a model, that is, the speed at which turtles move, patches change color, and so on.



- ▶ Note that if you push the speed slider well to the right, the view may update so infrequently that the model appears to have slowed down. It hasn't, as you can see by watching the tick counter race ahead. Only the frequency of view updates has lessened.

Adjusting Settings: Sliders and Switches

- ▶ A model's settings let you explore **different scenarios** or hypotheses. Altering the settings and then running the model to see how it reacts can give you a deeper understanding of the phenomena.
- ▶ Switches and sliders give you access to a model's **settings**.



Excercise

- ▶ Open Wolf Sheep Predation if it's not open already.
- ▶ Press “setup” and “go” and let the model run for about 100 ticks. (The tick count is shown above the view.) Stop the model by pressing the “go” button.
- ▶ Q. What happened to the sheep over time?
- ▶ Select the model version sheep-wolves-grass. Press “setup” and “go” and let the model run for a similar amount of time as before.
- ▶ Q What did the switch do? Was the outcome the same as your previous run?

Excercise

- ▶ Change from the Interface to the Info tab to learn what each of this models' sliders represents.
- ▶ Q. What would happen to the sheep population if there were more sheep and less wolves initially?

Excercise

- ▶ Q. What would happen to the sheep population if there were more sheep and less wolves initially?
 - ▶ Turn the “grass?” switch off.
 - ▶ Set the “initial-number-sheep” slider to 100.
 - ▶ Set the “initial-number-wolves” slider to 20.
 - ▶ Press “setup” and then “go”.
 - ▶ Let the model run for about 100 ticks.
 - ▶ Try running the model several times with these settings.
- ▶ Q. What happened to the sheep population?
- ▶ Q. Did this outcome surprise you? What other sliders or switches can be adjusted to help out the sheep population?

Excercise

- ▶ Set “initial-number-sheep” to 80 and “initial-number-wolves” to 50. (This is close to how they were when you first opened the model.)
- ▶ Set “sheep-reproduce” to 10.0%.
- ▶ Press “setup” and then “go”.
- ▶ Let the model run for about 100 time ticks.
- ▶ Q. What happened to the wolves in this run?

Plots

- ▶ Our plot contains three **lines**: sheep, wolves, and grass / 4. (So it doesn't make the plot too tall.)
- ▶ The lines show what's happening in the model over **time**. The plot legend shows what each line indicates: The population counts.
- ▶ When a plot gets close to becoming filled up, the horizontal axis is **compressed** and all of the data from before gets squeezed into a smaller space.
- ▶ If you want to save the data from a plot to view or analyze it in another application, use the “Export Plot” item on the File menu (**CSV** format).

Monitors

- ▶ Monitors are another means of **displaying** information from a model.

sheep	wolves	grass / 4
0	0	0

- ▶ The monitors show us the population of sheep and wolves, and the amount of grass. (Remember, the amount of grass is divided by four to keep the plot from getting too tall.)
- ▶ The numbers displayed in the monitors change as the model runs, whereas the plots show you data from the whole course of the model run.



Toolbar of Controls

- ▶ Let's experiment with the effect of these controls.
 - ▶ Press “setup” and then “go” to start the model running.
 - ▶ As the model runs, move the speed slider to the left.
 - ▶ Q. What happens?
- ▶ This slider is helpful if a model is running too fast for you to see what's going on in detail.
 - ▶ Move the speed slider to the middle.
 - ▶ Try moving the speed slider to the right.
 - ▶ Now try checking and unchecking the “view updates” checkbox.
 - ▶ Q. What happens?

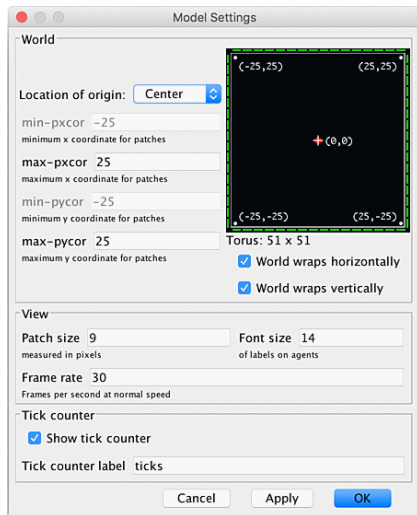
Comments

- ▶ Fast forwarding the model and turning off view updates are useful if you're impatient and want a model to run **faster**. Fast forwarding drops view updates so the model can run fast.
- ▶ When view updates are off completely, the model continues to run in the **background**, and plots and monitors still **update**. But if you want to see what's happening, you need to turn view updates back on by rechecking the box. Many models run much faster when view updates are off. For others, it makes little difference.

Properties of the View

- ▶ The size of the view is determined by five separate **settings**: min-pxcor, max-pxcor, min-pycor, max-pycor, and patch size.
- ▶ There are more model settings than there's room for in the toolbar. The **Settings...** button lets you get to the rest of the settings.

The settings editor

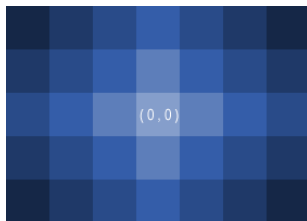


Excercise

- ▶ Q. What are the current settings for min-pxcor, max-pxcor, min-pycor, max-pycor, and patch size?
- ▶ Press “cancel” to make this window go away without changing the settings.
- ▶ Place your mouse pointer next to, but still outside of, the view. The pointer turns into a crosshair. Hold down the mouse button and drag the crosshair over the view. The view is now **selected**.
- ▶ Drag one of the square black “handles”. The handles are found on the edges and at the corners of the view. Unselect the view by clicking anywhere in the white background of the Interface tab. Press the “Settings. . .” button again and look at the settings.
- ▶ Q. What numbers changed?
- ▶ Q. What numbers didn't change?

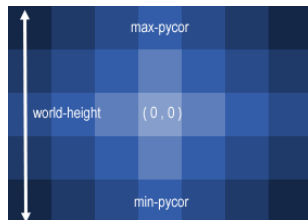
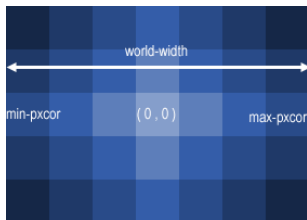
NetLogo's World

- ▶ The world is a two dimensional **grid** of patches –the individual squares in the grid.
- ▶ Think of the patches as square **tiles** in a floor. By default, exactly in the middle of the room is a tile labeled **(0,0)**; meaning that if the room was divided in half one way and then the other way, these two dividing lines would intersect on this tile. We now have a **coordinate system**:



Boundaries

- ▶ The number of tiles from right to left is called **world-width**. And the number of tiles from top to bottom is **world-height**. These numbers are defined by top, bottom, left and right **boundaries**.



Sections

- Sample models.** Organized by subject area and currently contains more than 200 models.
- Curricular models.** These are models designed to be used in schools in the context of curricula developed by the CCL at Northwestern University. Some of these are models are also listed under Sample Models.
- Code examples.** Simple demonstrations of particular features of NetLogo.
- Hubnet activities.** Participatory simulations for use with groups. e.g., in the classroom.



Types of Agents

- Mobile agents.** Referred as **turtles**. Later we will create **breeds**, i.e., kinds of turtles.
 - Patches.** Square cells that represent space. The **world** is a rectangular grid of patches.
 - Links.** Connections between two turtles, representing **relationships**, e.g., a network.
- The observer.** The overall **controller** of a model and its display.

Agents and "Agents"

- ▶ Usually, the term **agents** refers to what we have called mobile agents (turtles) and the controller.
- ▶ Patches and links are defined as agents **exclusively** in NetLogo.

Variables and Commands

- ▶ Each **type** of agent has certain variables and commands.
- ▶ The NetLogo Dictionary (Reference) defines all of them.
- ▶ There are several important **built-in** variables for each agent type, e.g., location, color, etc.
- ▶ They are also defined in the Dictionary (Variables).
- ▶ You can define your **own** variables for each agent type.
- ▶ Variables belonging to the observer behave as **global** variables. They used to represent **environment characteristics** or **model parameters**.

Primitives

- ▶ Built-in procedures or commands for telling agents what to **do**.
- ▶ **Example.** `move-to` tells a turtle to move to the location of a patch or other turtle.
- ▶ There is even a primitive to move a turtle to the patch in his neighborhood that has the highest value of some variable.
- ▶ They are all defined, as usual, in the Dictionary.
- ▶ They are organized in two **categories**:
 - Commands.** Tell an agent to do something.
 - Reporters.** Calculate a value and report it for further use in the program.



Primitives and Kinds of Agents

- ▶ Primitives are kind sensitive. They are annotated in the Dictionary with an icon representing a turtle, a patch, a link, or the observer.
- ▶ A primitive can **only** be used by the kind of agent that annotates it.
- ▶ Each piece of code is a program in the **context** of one kind (ocasionally, several kinds) of agent.
- ▶ **Example.** You can not use `move-to` in the context of patches. It is only usable by turtles. Error: Using a turtle command in a patch context.
- ▶ Some primitives, e.g., `ask` begin new contexts within their brackets.
- ▶ **Example.** `ask patches [set pcolor red]`
- ▶ The same applies to variables.



New Project

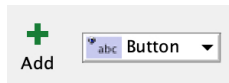
- ▶ We are ready to **program** our first model:
 1. Create, via File/New a new program. Save it, via File/Save as, in the directory of your choice with the name `mushroomHunt.nlogo`
 2. Click the Settings button.
 3. From the Interface tab, click the Settings button to open the Model Settings dialog, where you can check the World's geometry. For now we will use the default geometry with the origin (patch 0,0) at the center of the World, and both the maximum x-coordinate (max-pxcor) and maximum y-coordinate (max-pycor) set to 16.

Description

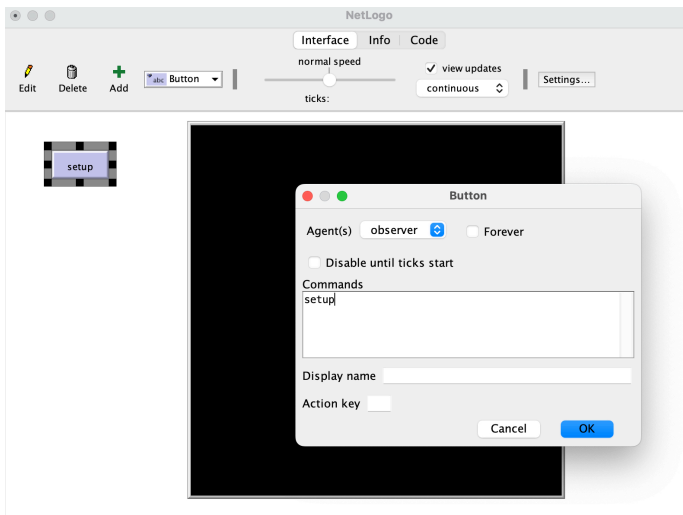
- ▶ For Mushroom Hunt, we want to create a world of black patches, with a few clusters of red patches (**mushrooms**) sprinkled in it;
- ▶ Create two turtles –our **hunters**;
- ▶ and then let the hunters **search** for the red patches.
- ▶ We therefore need to program how to **initialize** (create) the world and the hunters, and then the actions the hunters perform when the model runs.

Setup and Go

- ▶ We always use the convention of using the name `setup` for the procedure that initializes the World and turtles,
- ▶ and the name `go` for the procedure that contains the actions to be performed repeatedly as the model runs.
- ▶ To program these procedures, first we create the **buttons** using the tools bar:



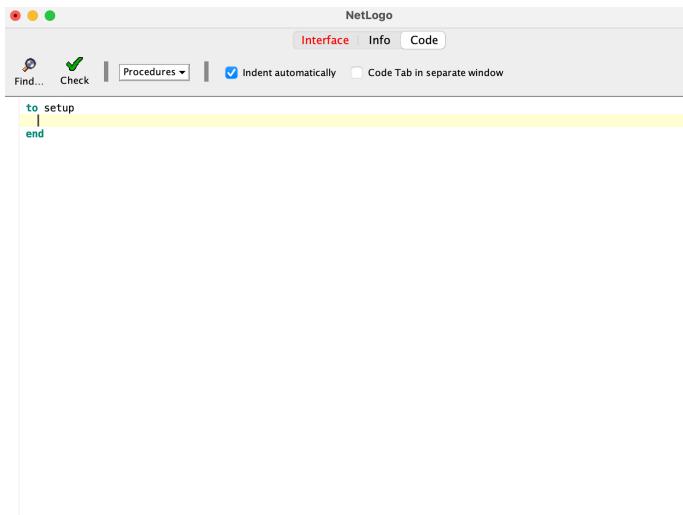
Adding the setup button



Going on

- ▶ The button's label and the interface tab are in red to indicate a **problem**: The corresponding command is not defined.
- ▶ Create a second button, assign it the command go, click the **forever** checkbox in the Button window, and click OK.

Writing the setup procedure



The screenshot shows the NetLogo software interface. At the top, there are three window control buttons (red, yellow, green) and the title "NetLogo". Below the title bar, there are three tabs: "Interface" (highlighted in red), "Info", and "Code". Under the "Code" tab, there is a toolbar with "Find..." (magnifying glass icon) and "Check" (checkmark icon). To the right of the toolbar is a dropdown menu set to "Procedures" and two checkboxes: "Indent automatically" (checked) and "Code Tab in separate window" (unchecked). The main area is a code editor with a light blue background. The code in the editor is:

```
to setup
|
end
```

The first two lines of code are highlighted in yellow. The "to setup" line is in blue, and the "end" line is in green.

Setup

- ▶ Change the setup procedure to say

```
1 | to setup  
2 |   ask patches [set pcolor red]  
3 | end
```

- ▶ This changes all patches to red!

A better setup

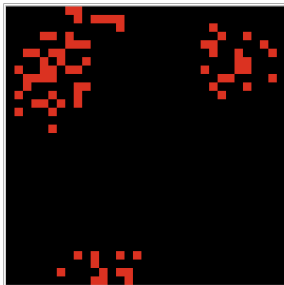
- ▶ But we want only some sparse red patches. Lets try

```
1 | to setup
2 |   ask n-of 4 patches
3 |   [
4 |     ask n-of 20 patches in-radius 5
5 |     [
6 |       set pcolor red
7 |     ]
8 |   ]
9 | end
```



Help

- ▶ Look up **n-of** and **in-radius** in the NetLogo Dictionary.
- ▶ You can go straight to the definition of any primitive by clicking within the word and pressing F1.
- ▶ Q. Test your procedure several times, what happens?
- ▶ Never, never forget to **clear all** at the beginning of your setup.



A better, better setup

- ▶ But we want only some sparse red patches. Lets try

```
1 | to setup
2 |   ca
3 |   ask n-of 4 patches
4 |   [
5 |     ask n-of 20 patches in-radius 5
6 |     [
7 |       set pcolor red
8 |     ]
9 |   ]
10 | end
```

- ▶ `ca` stands for `clear-all`.

Observations

- ▶ Sometimes it looks like fewer, larger clusters are created, because clusters **overlap**.
- ▶ Other times, it looks like part of a cluster is created at the **edge** of the World
- ▶ To understand why, go to the NetLogo Interface Guide section on the Views and read about **world wrapping** and see the Programming Guide's section called **Topology**.
- ▶ Understanding NetLogo's world wrapping is extremely important!
- ▶ Please **SAVE** your work.

Model parameters

- ▶ Lets **parameterize** the number of clusters of red patches.
- ▶ In our models, we usually try not to write numbers that control important things directly into the deep code.
- ▶ Instead, we make them parameters that are easy to find and change, i.e., we use **global variables**
- ▶ Add this to your code:

```
1 | globals [ num-clusters ]
```

New setup

- ▶ Modify setup accordingly:

```
1 | to setup
2 |   ca
3 |   set num-clusters 4
4 |   ask n-of num-clusters patches
5 |   [
6 |     ask n-of 20 patches in-radius 5
7 |     [
8 |       set pcolor red
9 |     ]
10 | ]
11 | end
```

- ▶ Later we'll use **sliders** to modify the value of the parameters.



Creating our hunters

- ▶ We need to create two turtles.
- ▶ Add the following code to the setup procedure:

```
1 | crt 2
2 | [
3 |   set size 2
4 |   set color yellow
5 | ]
```

- ▶ Once created, the turtles will execute the code within the brackets.
- ▶ `crt` stands for `create-turtles`.

The go procedure

- ▶ It defines a model's schedule: The processes (procedures) that will be performed over and over agent, and their order.
- ▶ For the sake of **simplicity**, go uses to call other procedures where each action is programmed.
- ▶ **Example.** The hunter's search:

```
1 | to go
2 |   ask turtles [search]
3 | end
```

Our first search procedure

- ▶ Remember the strategy (long linear paths when nothing is found, short changing paths in the opposite case):

```
1 to search
2   ifelse time-since-last-found <= 20
3     [right (random 181) - 90]
4     [right (random 21) - 10]
5
6   forward 1
7 end
```

Problems!

- ▶ `time-since-last-found` has not been defined!
- ▶ First we declare it as a variable in the context of turtles:

```
1 | turtles-own
2 | [
3 |   time-since-last-found
4 | ]
```

- ▶ Then we initialize the value of the variable, e.g., when creating the turtles:


```
1 | crt 2
2 | [
3 |   set size 2
4 |   set color yellow
5 |   set time-since-last-found 999
6 | ]
```

The final search process

- ▶ We need to update time-since-last-found in the search process:

```
1 to search
2   ifelse time-since-last-found <= 20
3     [right (random 181) - 90]
4     [right (random 21) - 10]
5   forward 1
6   ifelse pcolor = red
7     [
8       set time-since-last-found 0
9       set pcolor yellow
10    ]
11    [
12      set time-since-last-found time-since-last-found + 1
13    ]
14 end
```

Agent monitors



turtle 0

▼ View

Watch

▼ Properties

who	0
color	45
heading	150
xcor	0.749061891972937
ycor	12.898282467711118
shape	"default"
label	""
label-color	9.9
breed	turtles
hidden?	false
size	2
pen-size	1
pen-mode	"up"
time-since-last-found	5

- ▶ Agent monitors are tools to **see, and change**, the variables of an agent.
- ▶ Move the cursor over one of the hunters and right-click. A panel appears that, at the end, lists turtle 0; move the cursor to this entry and select inspect turtle 0.
- ▶ Change the turtle size to 5.



The Command Center

- ▶ It appears at the **bottom** of the Interface tab.



- ▶ You have asked each hunter to create a second hunter, which then turns 160 degrees.
- ▶ Q. Select the observer to send commands to, and enter `show count turtles`



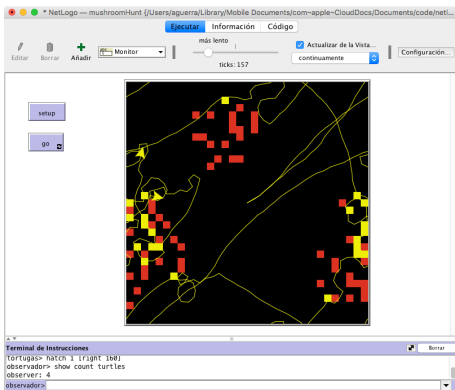
Time

- ▶ We haven't deal with time explicitly: we do not keep track of how many times the go procedure has executed, so we cannot determine **how much time** has been simulated.
- ▶ Add `reset-ticks` at the end of the setup procedure.
- ▶ Add `tick` at the end of the go procedure.
- ▶ Now you can follow the tick counter at the top of the World display.



Tracing the Turtles

- ▶ You can **trace** the routes of the mobile agents by adding **pen-down** when creating the turtles.



Conclusions

- ▶ To work **successfully** with NetLogo, you **must** use the User Manual **all the time**.
- ▶ The manual is available in **PDF** format for print copies.
- ▶ “Experienced programmers often quickly decide that they cannot program their models in NetLogo because they do not immediately see how to translate the logic of conventional languages into NetLogo. **Be patient**.”
- ▶ Section 2.4.4 lists the elements of NetLogo reviewed in this session.

References I

- [1] SF Railsback and V Grimm. *Agent-Based and Individual-Based Modeling*. Second. Princeton, NJ, USA: Princeton University Press, 2019.

