Mathematics for Decisions

Modeling and Simulation: Introduction to Netlogo

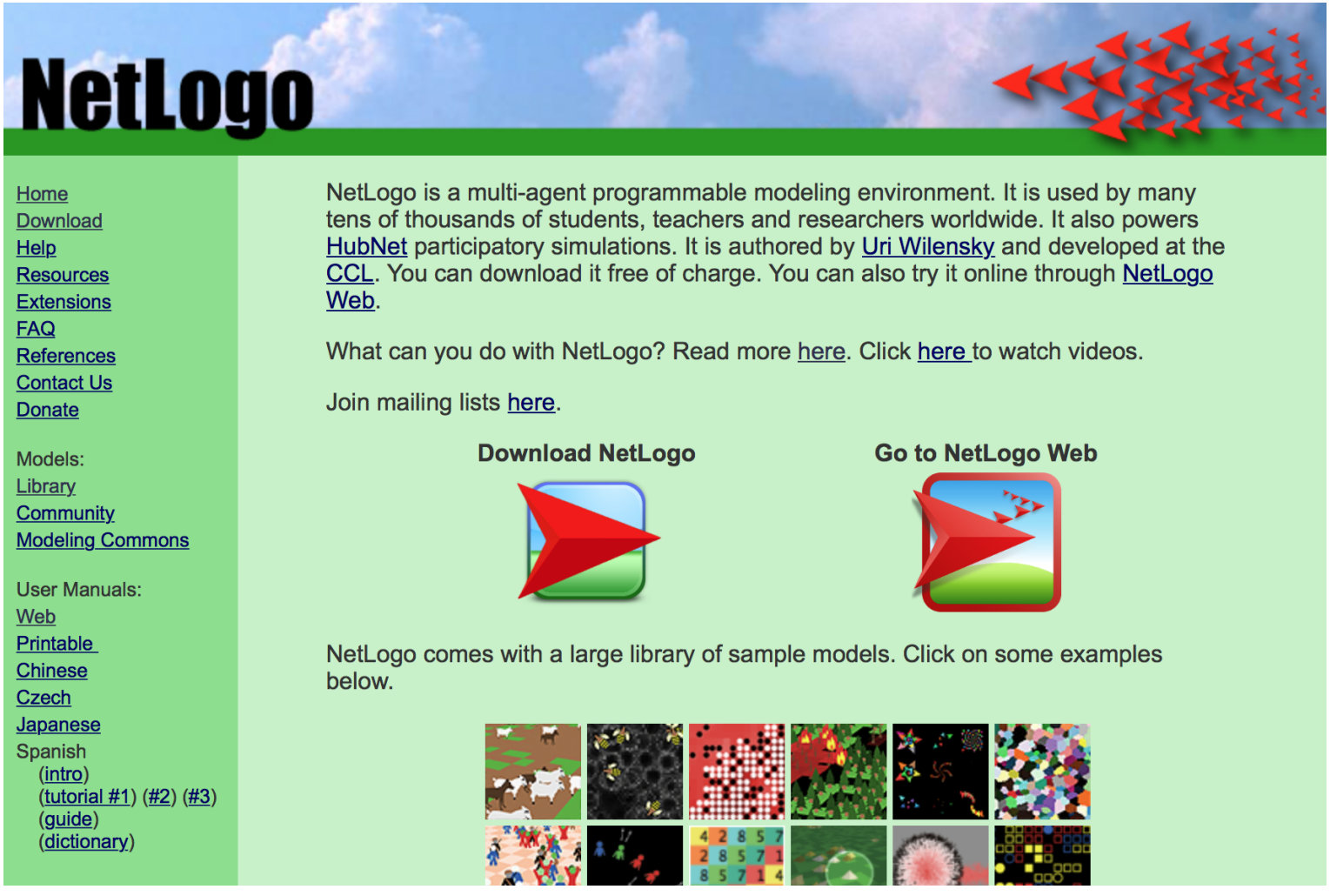
Romeo Rizzi, Alice Raffaele, Matteo Zavatteri

University of Verona

romeo.rizzi@univr.it, alice.raffaele@unitn.it, matteo.zavatteri@univr.it

January 12, 2021

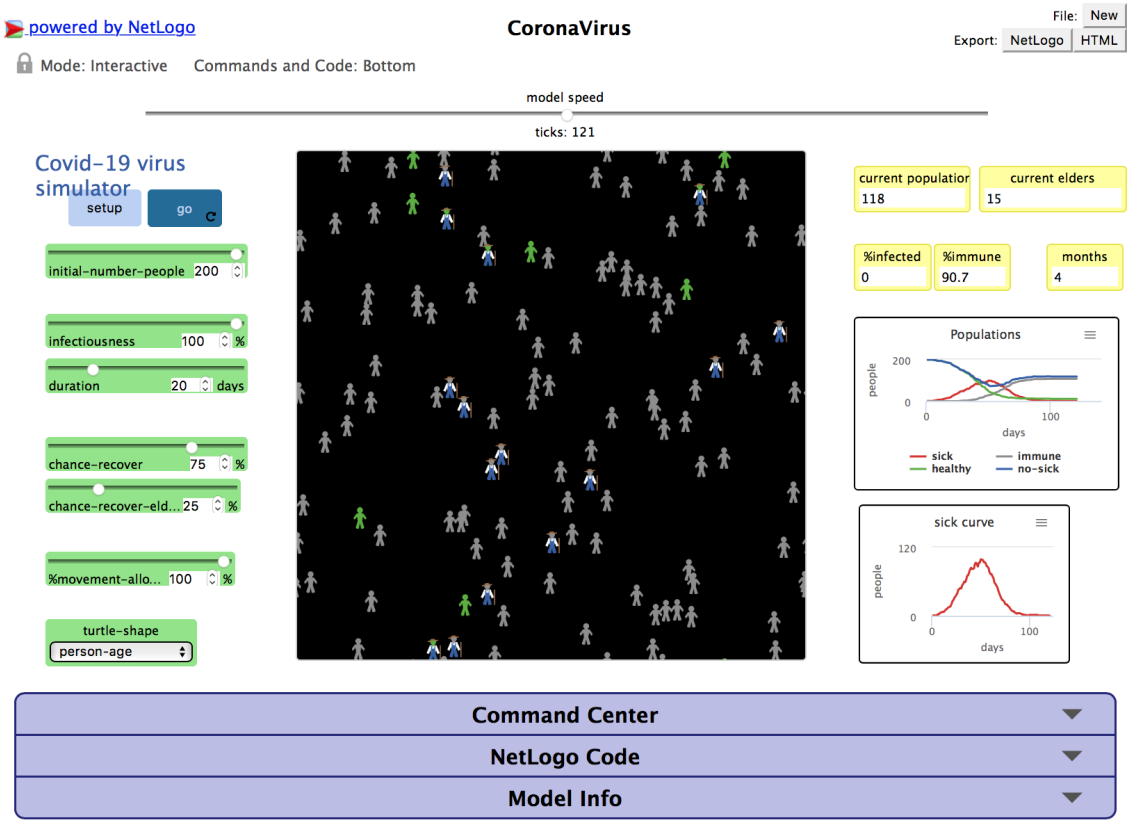
What is it?


NetLogo
Home
Download
Help
Resources
Extensions
FAQ
References
Contact Us
Donate
Models:
Library
Community
Modeling Commons
User Manuals:
Web
Printable
Chinese
Czech
Japanese
Spanish
(intro)
(tutorial #1) (#2) (#3)
(guide)
(dictionary)
NetLogo is a multi-agent programmable modeling environment. It is used by many tens of thousands of students, teachers and researchers worldwide. It also powers HubNet participatory simulations. It is authored by Uri Wilensky and developed at the CCL. You can download it free of charge. You can also try it online through NetLogo Web.
What can you do with NetLogo? Read more here. Click here to watch videos.
Join mailing lists here.
Download NetLogo
Go to NetLogo Web
NetLogo comes with a large library of sample models. Click on some examples below.

System requirements

- NetLogo runs on the **Java Virtual Machine**, so it works on all major platforms:
  - Windows: 10, 8, 7, Vista;
  - Mac: OS X 10.3 or newer;
  - Linux: standard Debian-based and Red Hat-based Linux distributions.
- Java 8 is already included in the download.
- It is run as a desktop application; command line operation is also supported.

Modelling a pandemia



http://www.maia.ub.es/~maite/CoronaVirus.html

At a party

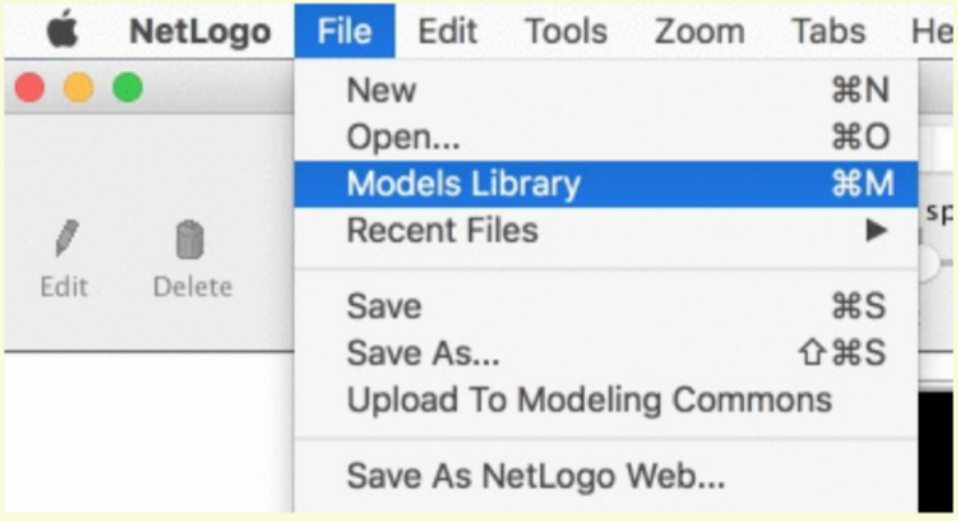
- People form groups and, as they circulate, groups change.
- Sociality depends on personalities and features: one can behave differently according to environment and others.

Is there any type of pattern to this kind of grouping?

Loading

What to do:

1. Start NetLogo.
2. Choose “Models Library” from the File menu.



3. Open the “Social Science” folder.
4. Click on the model called “Party”.
5. Press the “open” button.
6. Press the “setup” button.

Mingling groups

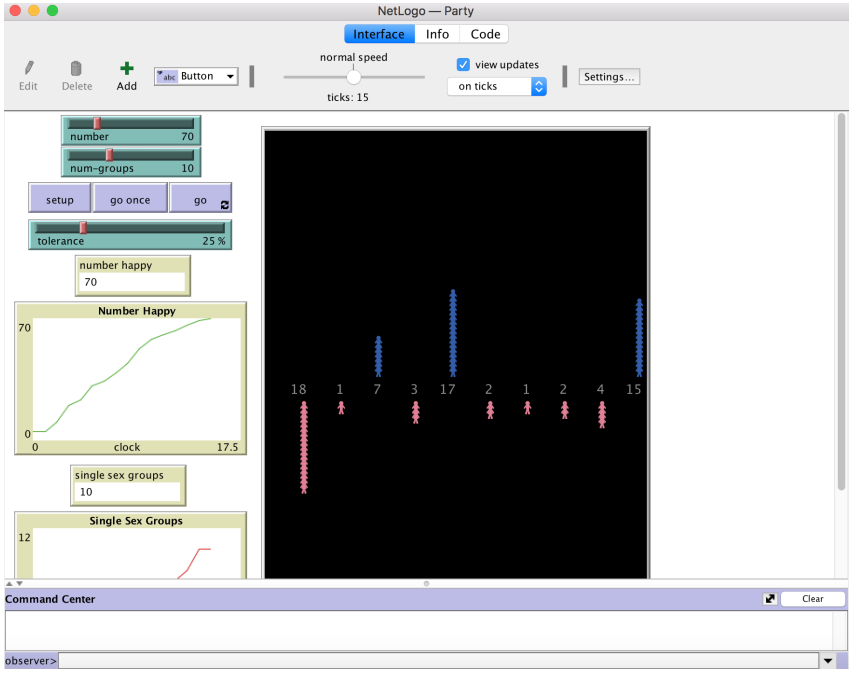
Suppose to have the following mingling groups at a party (men in blue, women in pink):

Questions:

- Do all the groups have about the same number of people?
- Do all the groups have about the same number of each sex?

Suppose $n = 150$ people: how will people gather together?

Simulation



What happens:

- People did not divide up evenly into the 10 groups.
- Over time, from all mixed groups to all single-sex groups.

Groups at parties do not just form randomly.

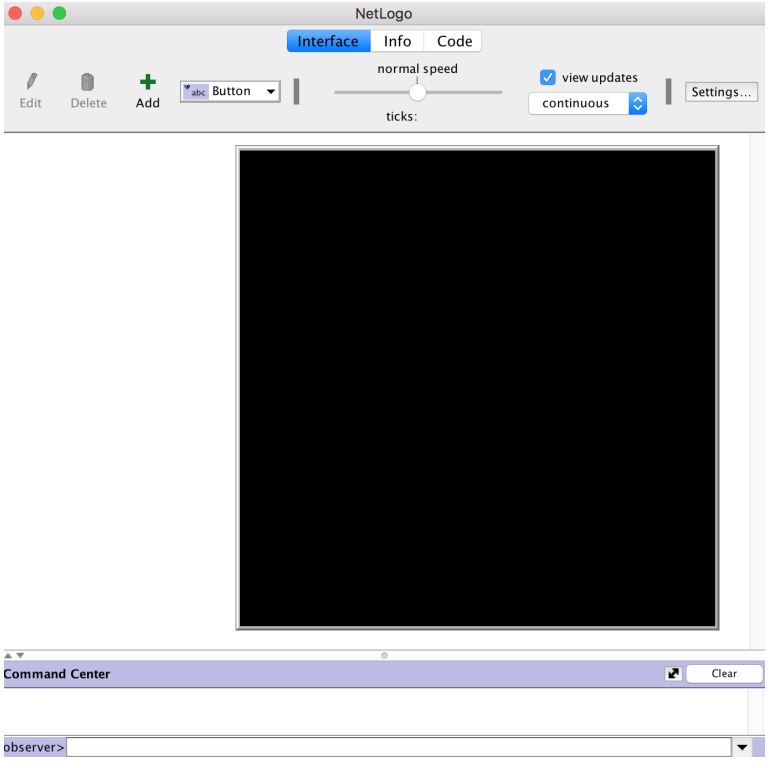
Playing with variables

- *Tolerance*: percentage of people of the opposite sex an individual is "comfortable" with;
- As individuals become "uncomfortable" and leave groups, they move into new groups, but this may cause some people in that group to become "uncomfortable" in turn and so on, until an equilibrium is reached.

Challenges

1. To make sure all groups of 10 have both sexes, at what level should we set the tolerance?
2. Can you see any other factors or variables that might affect the male to female ratio within each group?
3. How high does the tolerance value need to be before you get mixed groups?

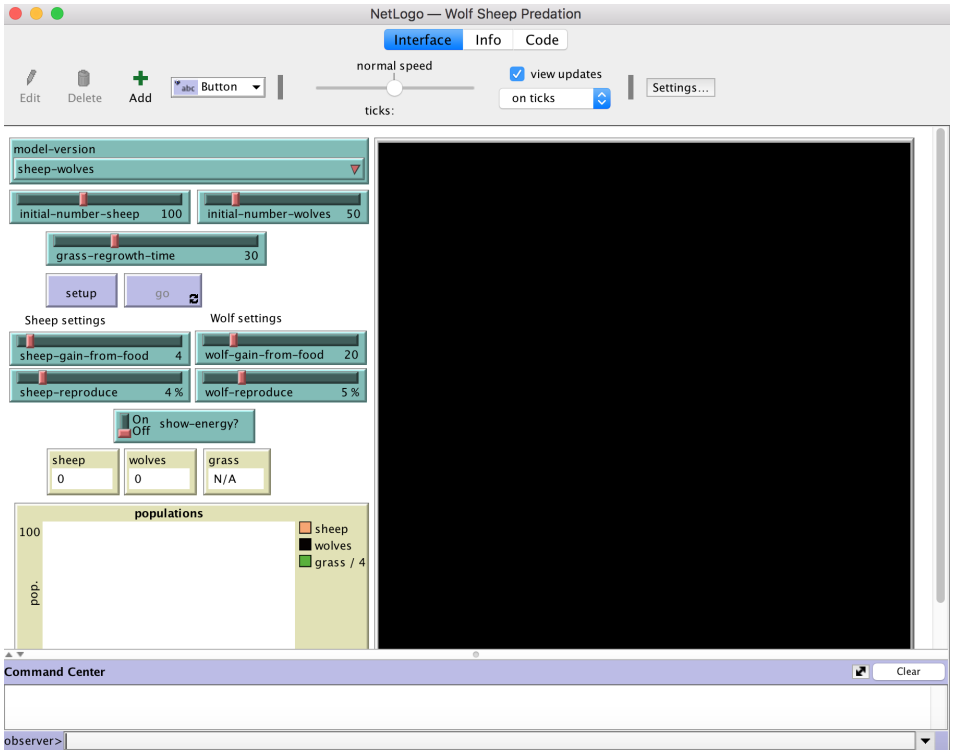
What can we do with NetLogo?

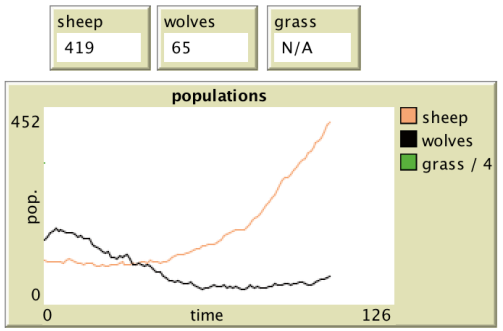


- Modeling a problem.
- Observing a situation or circumstance evolving dynamically.
- Experiment a scenario with a system in a rapid and flexible way.
- Surprise ourselves with *emergent* phenomena.

Running models

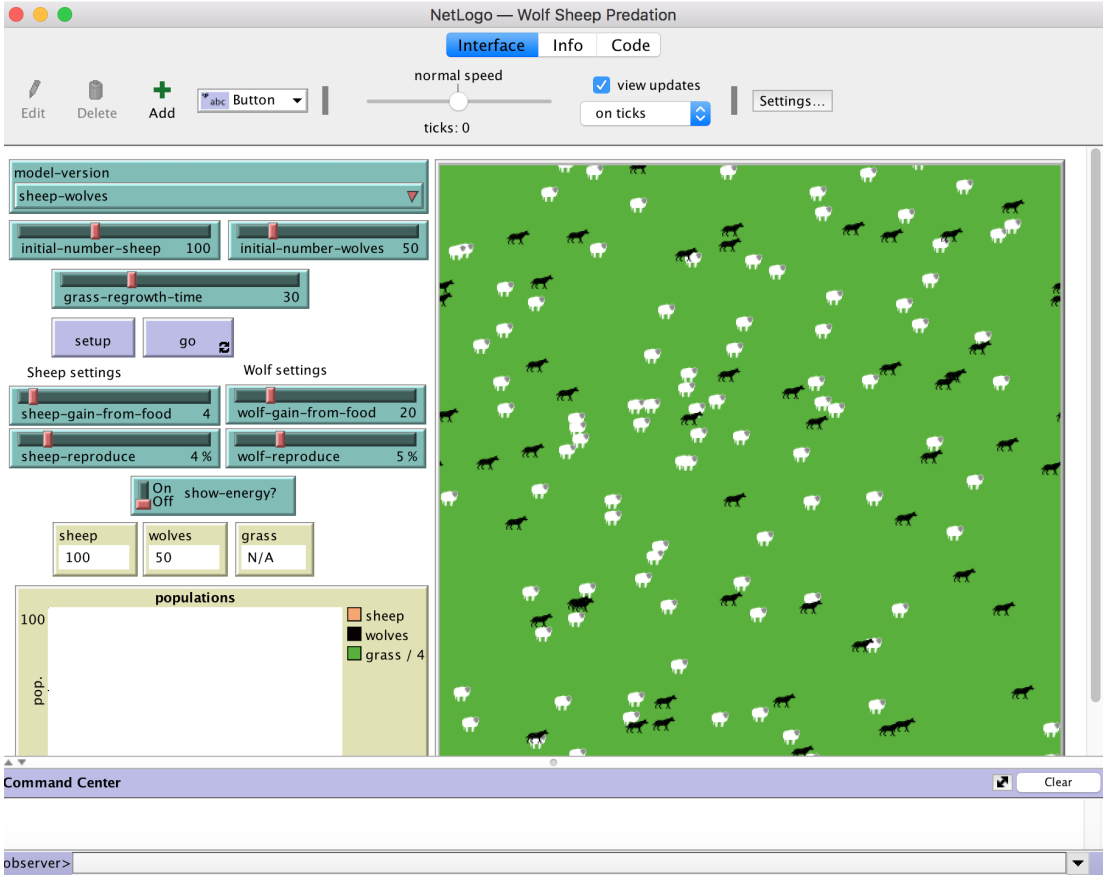
Let's use the **Wolf Sheep Predation** model: we can *interact with the model* (through switches, bars, buttons and sliders) and we can *see the results* (through monitors and views):



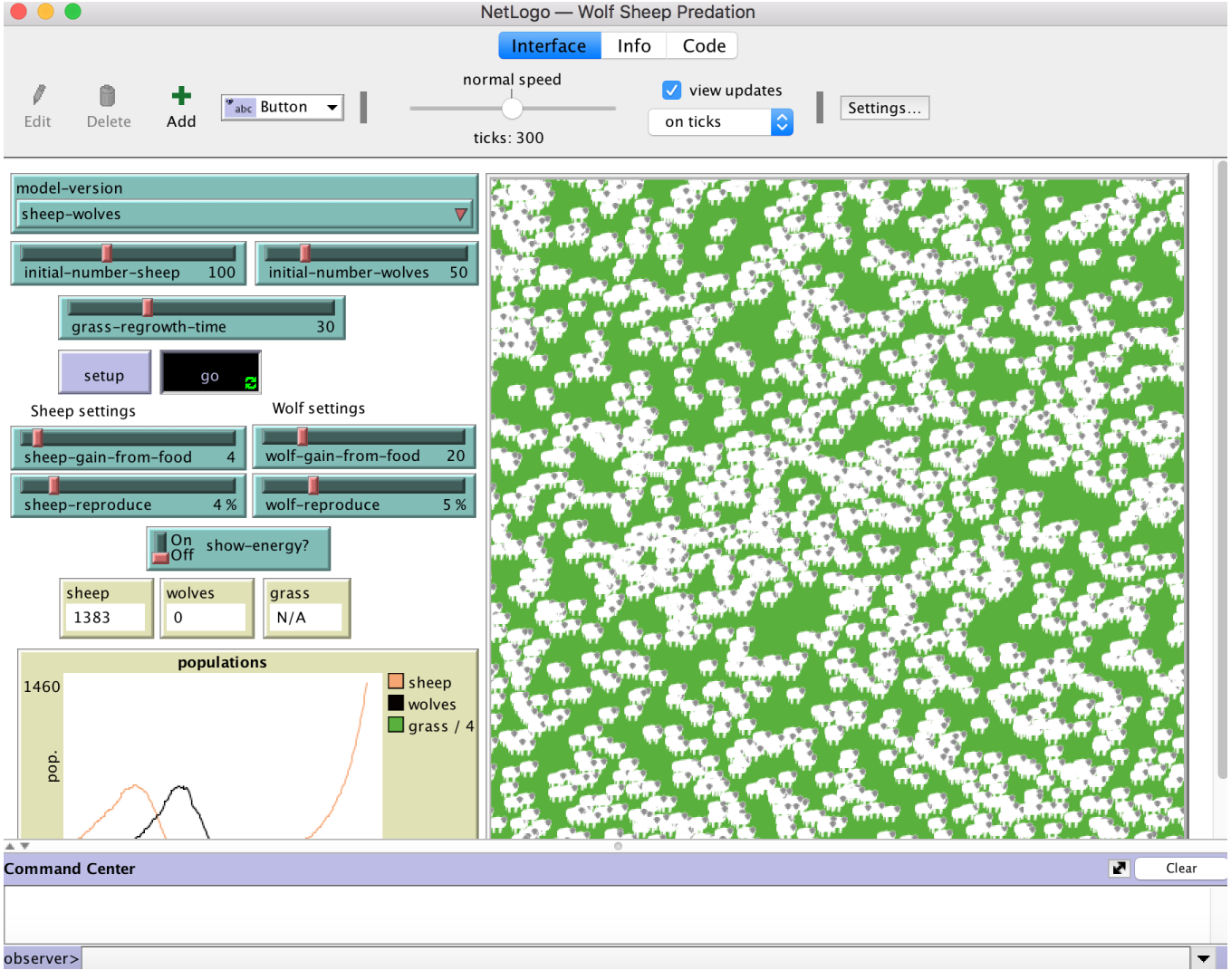


We can *edit*, *select*, *delete*, but also *copy image*, *export* or *clear*.

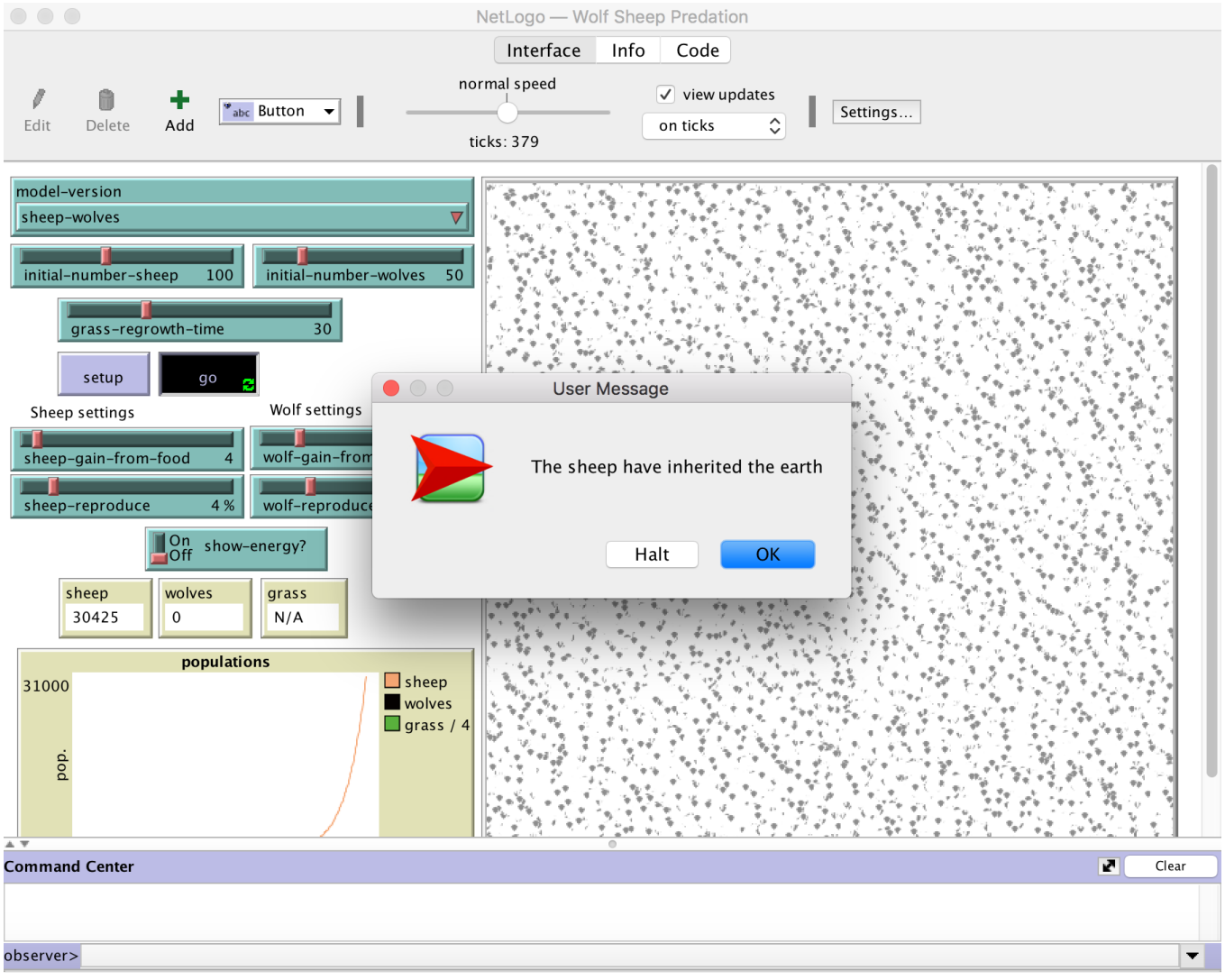
Pressing **setup**



Pressing **go**


NetLogo — Wolf Sheep Predation
Interface
Info
Code
Edit
Delete
Add
Button
normal speed
ticks: 300
view updates
on ticks
Settings...
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
Off
show-energy?
sheep
1383
wolves
0
grass
N/A
populations
1460
pop.
sheep
wolves
grass / 4
Command Center
Clear
observer>

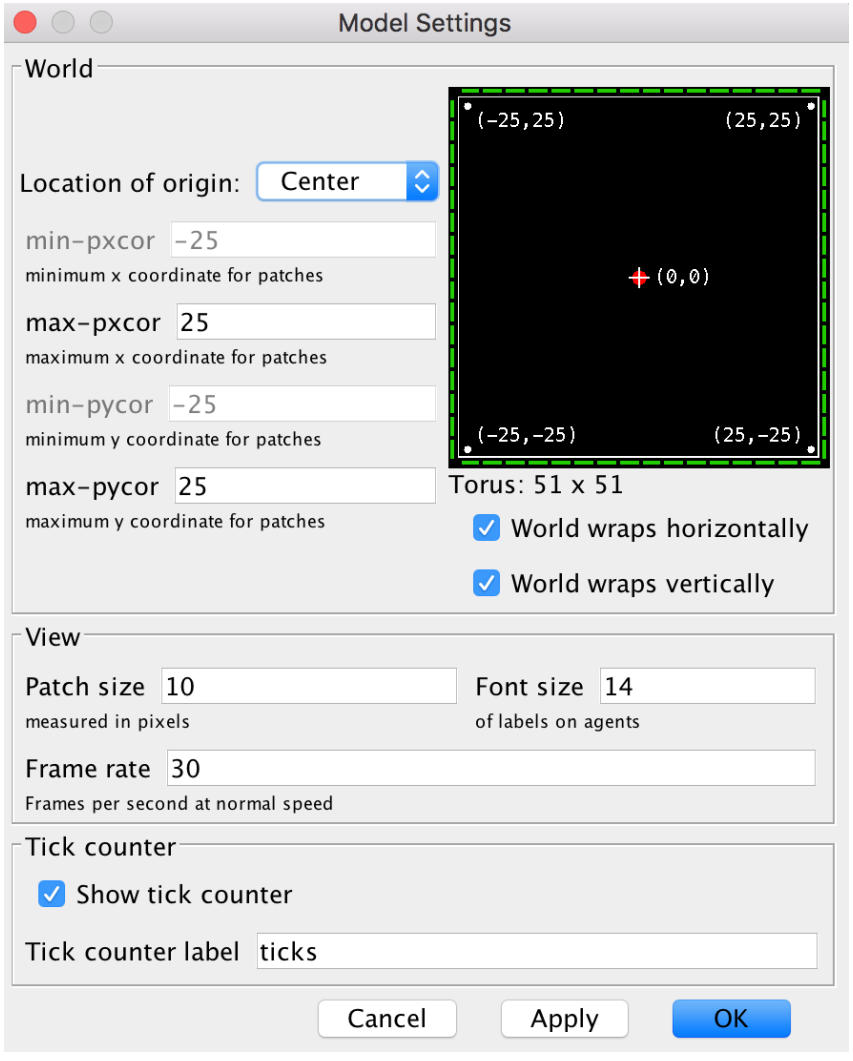
Results (with these settings)



Some tests

- Do you ever get different results if you run the model several times with the same settings?
- Try the speed slider and/or play with other switches;
- What would happen to the sheep population if there were more sheep and less wolves initially?
- What other sliders or switches can be adjusted to help out the sheep population?
- Set *initial-number-sheep* to 80 and *initial-number-wolves* to 50: what happened to the wolves in this run?

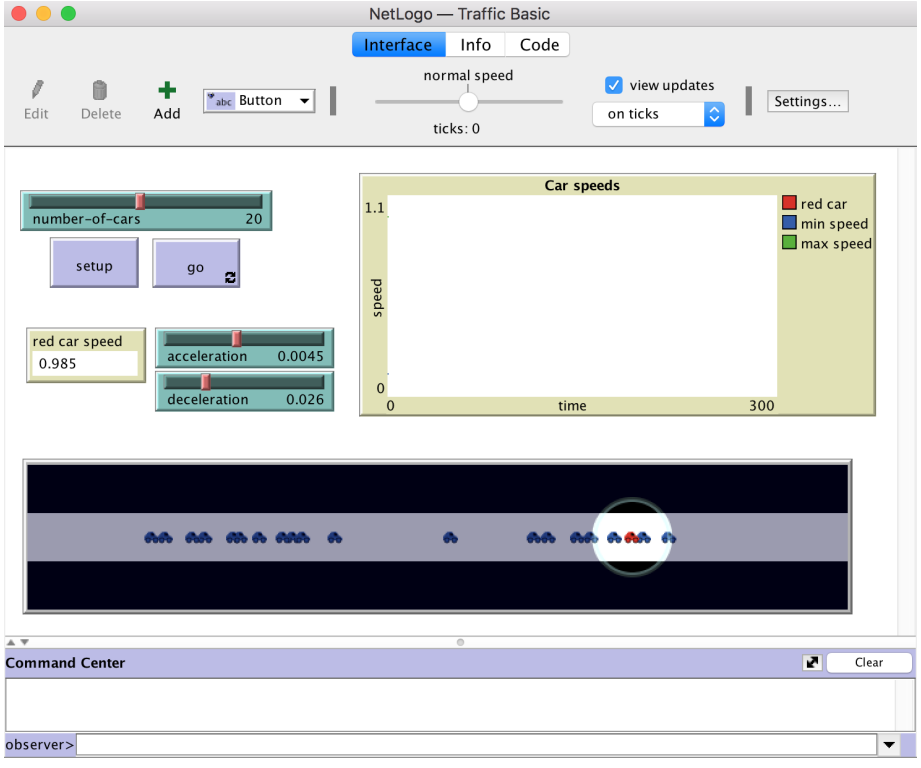
Press **Settings**



Commands

Let's shift from observing models to **manipulating** them, using the **Traffic Basic** model:

- One red car in a stream of blue cars.
- They all move in the same direction.
- Once in a while they "pile up" and an accident occurs.



Improving the model

Question: as you are using the Traffic Basic model, have you noticed any additions you would like to make to the model?

Improving the model

Question: as you are using the Traffic Basic model, have you noticed any additions you would like to make to the model?

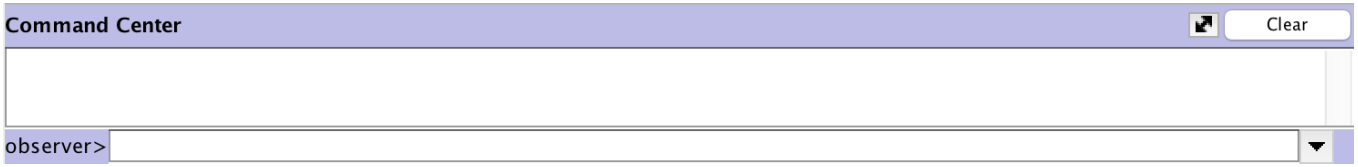
There are several improvements that could be implemented:

- *Visual*: changing color and shape of the cars.
- *Behavioral*: creating a stop light or another lane.

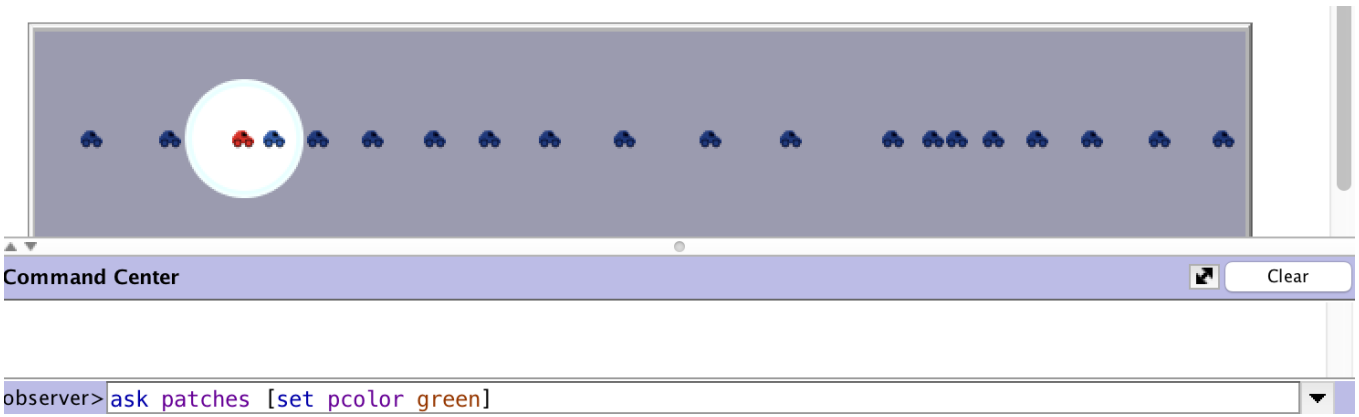
In this tutorial we'll focus just on visual changes.

Command Center

In the bottom of NetLogo application:



Let's try to tell the **observer** to do this, as indicate in the picture:



What happened to the View?
And why did not the cars turn green too?

Changing colors

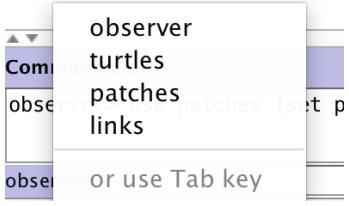
Let's examine together the command we've just written:

ask patches [set pcolor green]

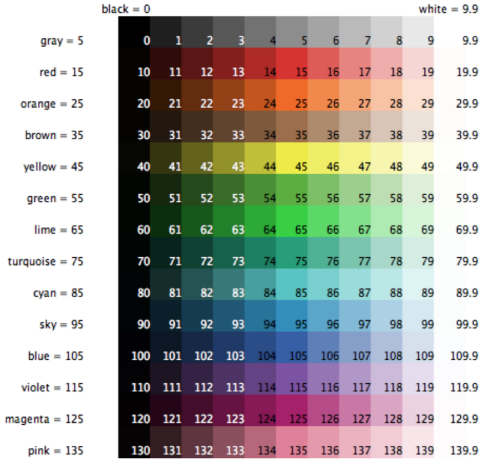
- *ask*, i.e., in this case, to order something to somebody;
- *patches* refers to an **agent** that is responsible of some objects in the models (but not the cars);
- *set pcolor green* determines exactly the action (i.e., to change the background color) we want the agent to do.

Agents

To see which agents are available, click on *observer*:



We can play with **color** (of a group of agents) and **pcolor** (a built-in patch variable which holds the color of the patch):

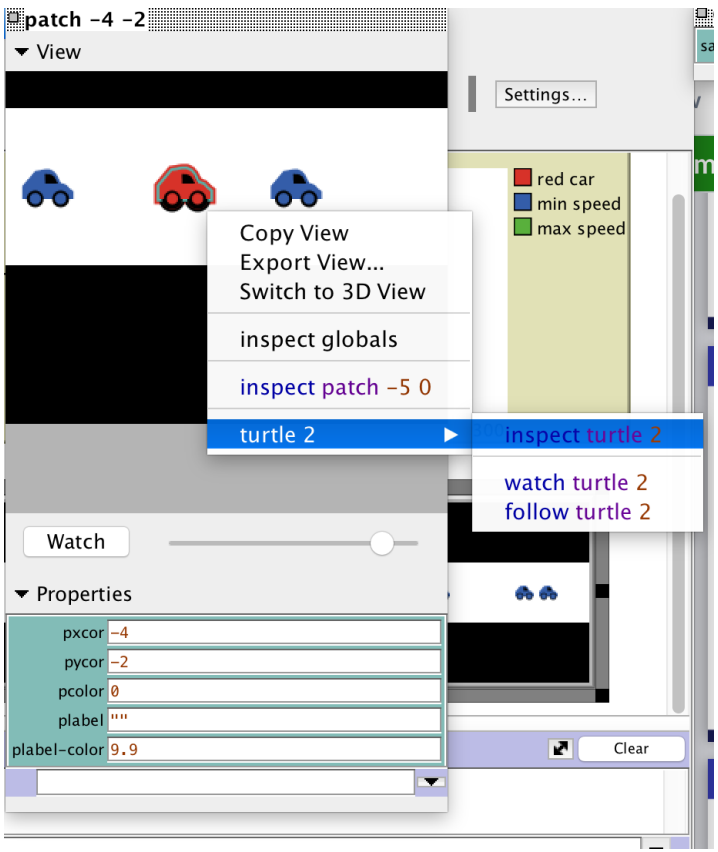


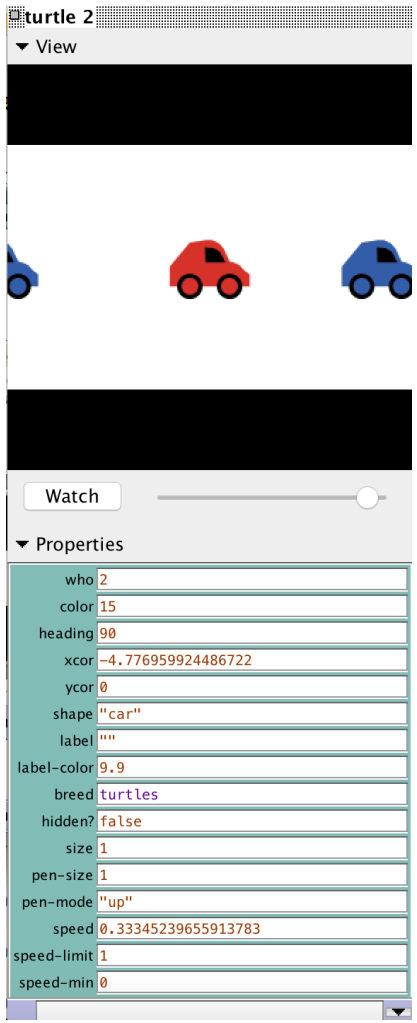
To get an intermediate shade, you can refer to it by a number instead, or by adding or subtracting a number from a name: e.g., **set pcolor red - 2**.

Properties

Focus on the red car and discover the properties we can play with:

Right-click on the red car or type **inspect turtle 2** (or other who number), into the Command Center).





Procedures

In this part we see how we can alter and extend an existing model and, why not, create our own ones.

Types of agents:

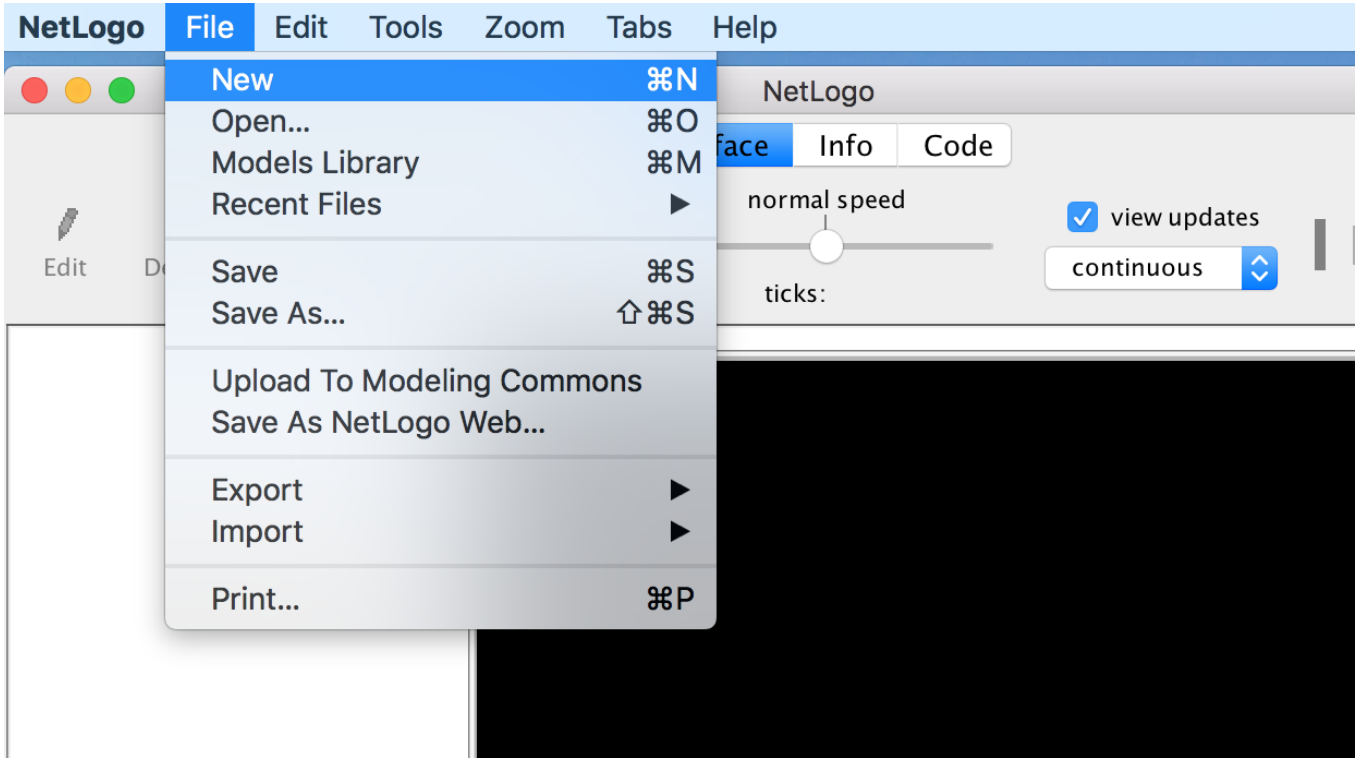
- patches – stationary and arranged into a grid;
- turtles – they move over that grid;
- links – they connect two turtles;
- the observer – it oversees everything that's going on and does whatever the turtles, patches and links cannot do for themselves.

They all can run commands and also procedures. A **procedure** combines a series of NetLogo commands into a single new command that we define by our own.

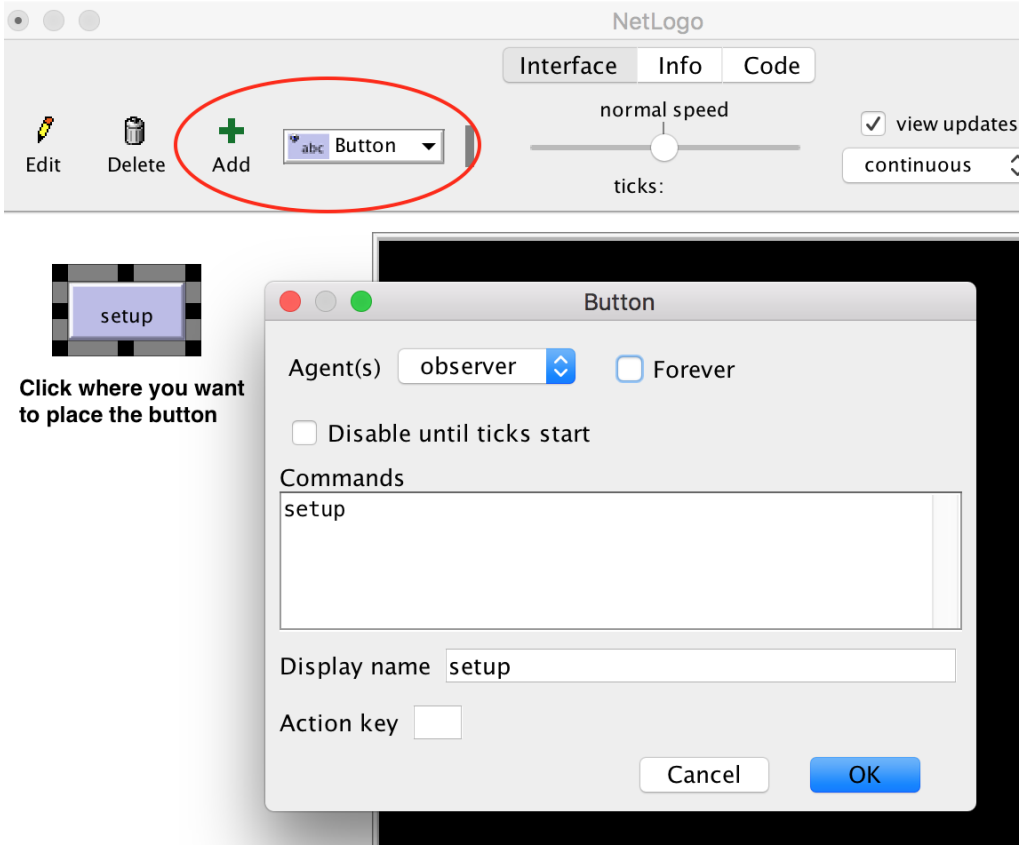
Main steps to build our ecosystem

1. Creating a new empty model
2. Making the setup button
3. Switching to tick-based view updates
4. Making the go button
5. Patches and variables
6. Turtle variables
7. Monitors
8. Switches and labels
9. More procedures
10. Plotting
11. Tick counter
12. Some more details

1. Creating a new empty model



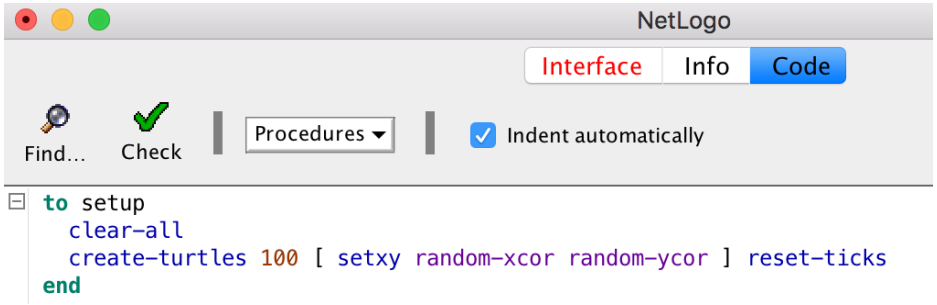
2. Making the setup button (I)



The label is red because we haven't written yet its related procedure, so let's do it.

2. Making the setup button (II)

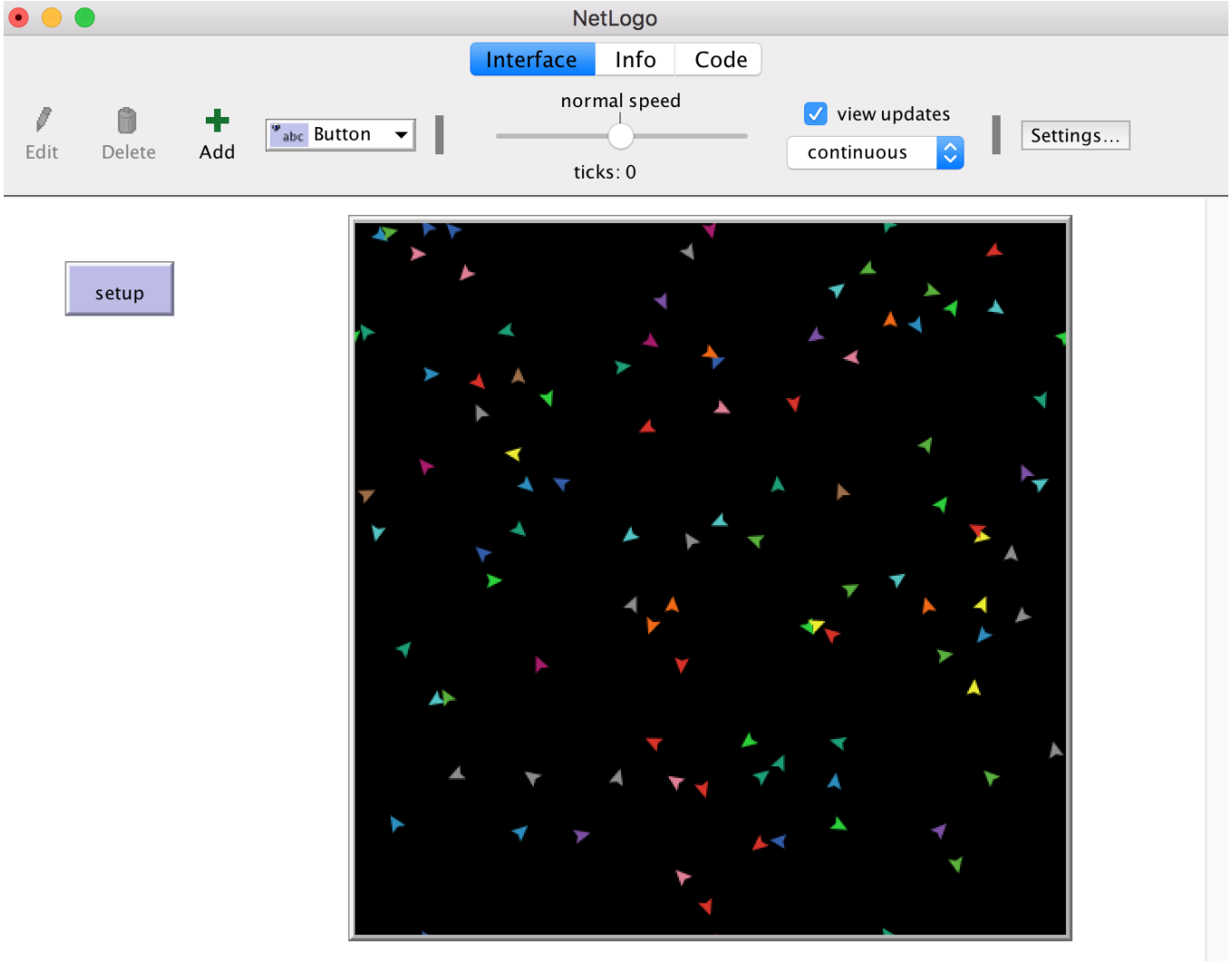
Switch to the Code tab and type:



Notes:

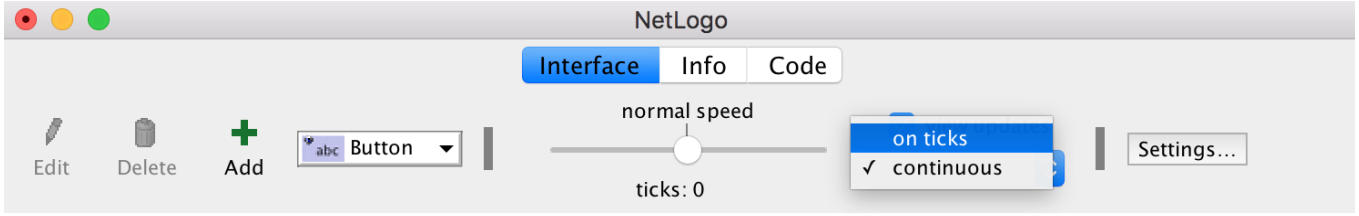
- Procedures begin with **to** and finish with **end**;
- **clear-all** resets the world to an initial empty state;
- **create-turtles 100** creates 100 turtles that start at the origin (i.e., the center of patch 0,0);
- **[. . .]** contain commands: *setxy random-xcor random-ycor* is a command using "reporters". A reporter, as opposed to a command, reports a result;
- **reset-ticks** starts the tick counter.

2. Making the setup button (III)



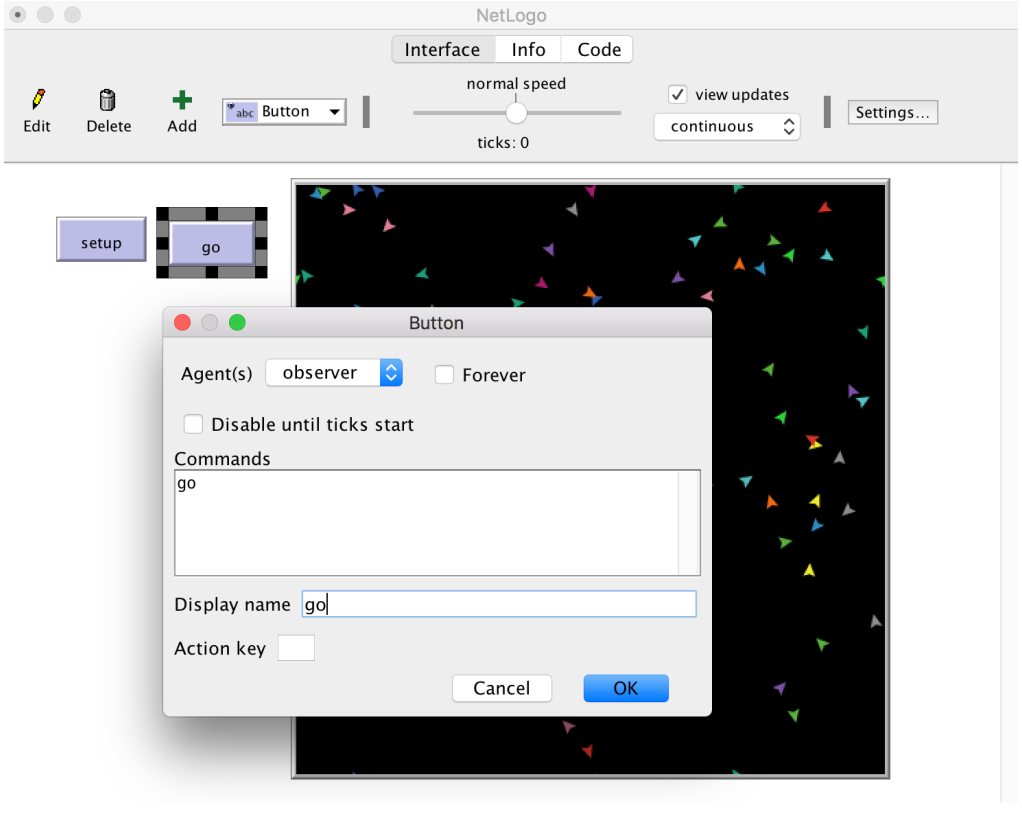
3. Switching to tick-based view updates

To update the view not continuously:



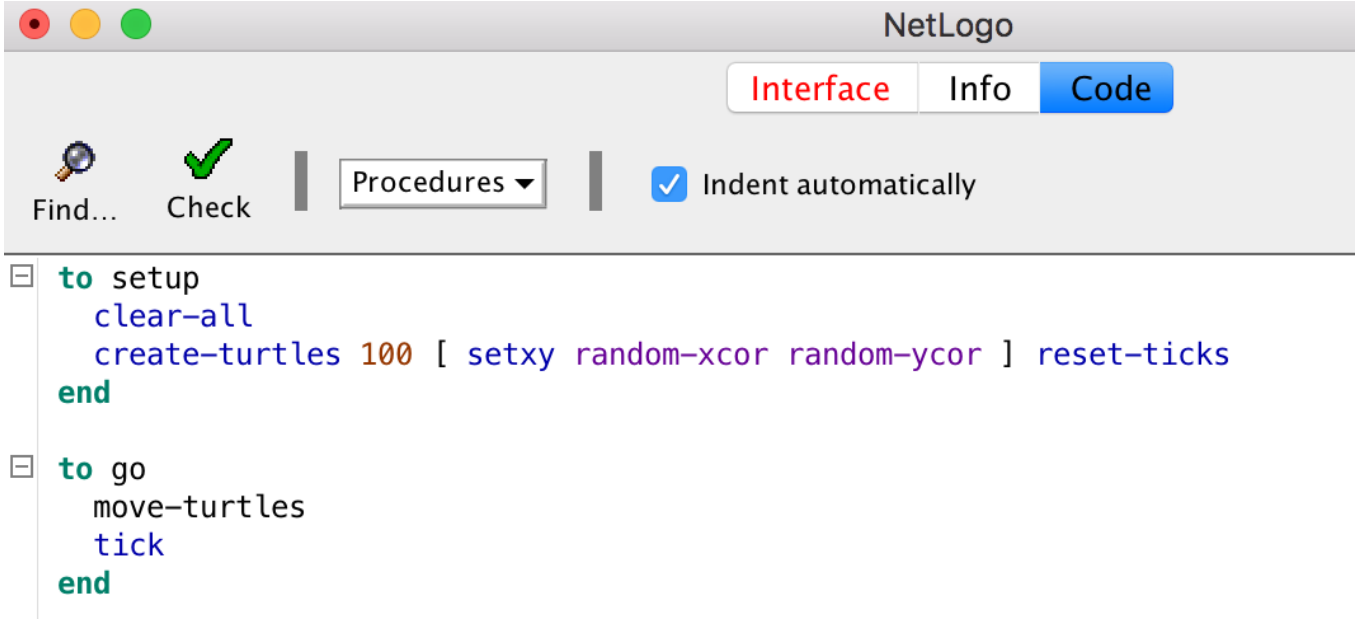
4. Making the go button (I)

Add the button as before



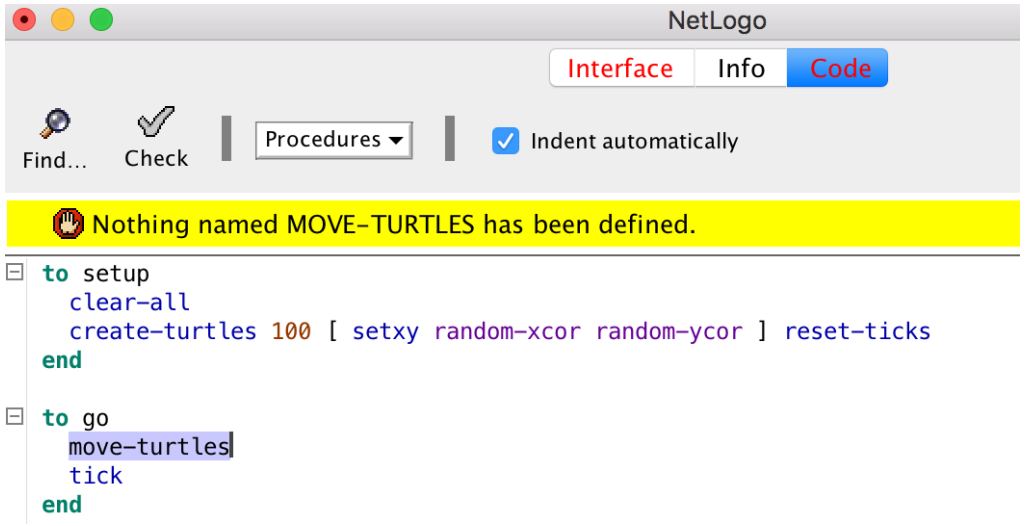
4. Making the go button (II)

And the code too, where **tick** increments the counter by 1:



But...

4. Making the go button (III)



Let's add also the **move-turtles** procedure:

```
to move-turtles
  ask turtles [
    right random 360
    forward 1 ]
end
```

Go back to the Interface tab and edit the go button: select **Forever** to make the turtles move randomly for infinite ticks (at least, until you stop them).

4. Making the go button (IV)

About the commands:

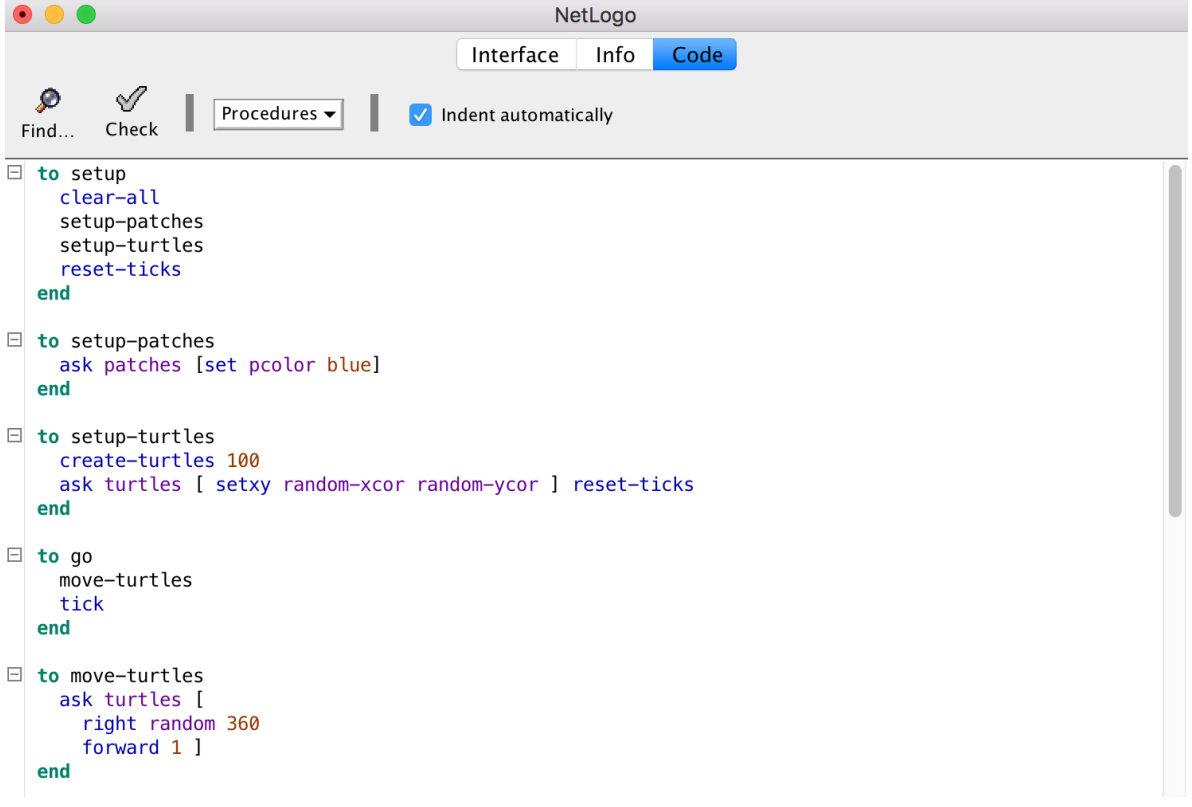
- **ask turtles [...]** says that each turtle should run the commands in the brackets;
- **right random 360** makes each turtle pick a random whole number between 0 and 359 to turn to the right;
- **forward 1** makes the turtle move forward one step.

Other notes:

- Modularity and simplicity (separate the procedures instead of writing everything in one block).
- By default behavior, if a turtle moves off the edge of the world, it "wraps", that is, it appears on the other side.
- Play around with other commands before you move forward with your model.

5. Patches and variables

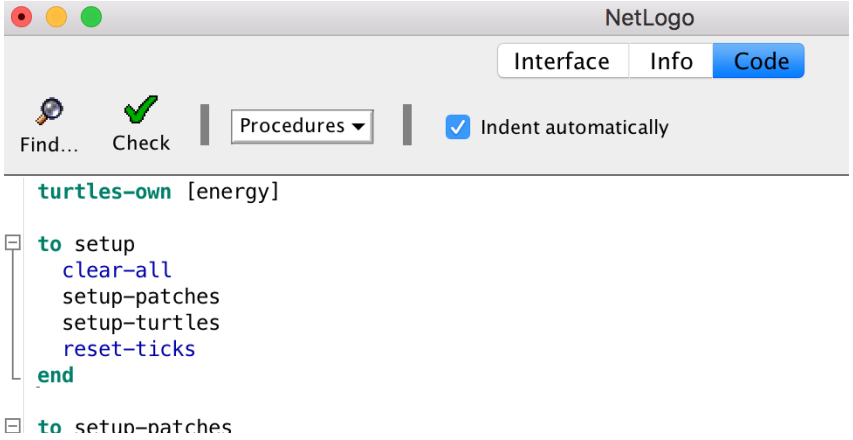
Let's add a setup procedure for patches and then rewrite the part about turtles:



6. Turtle variables (I)

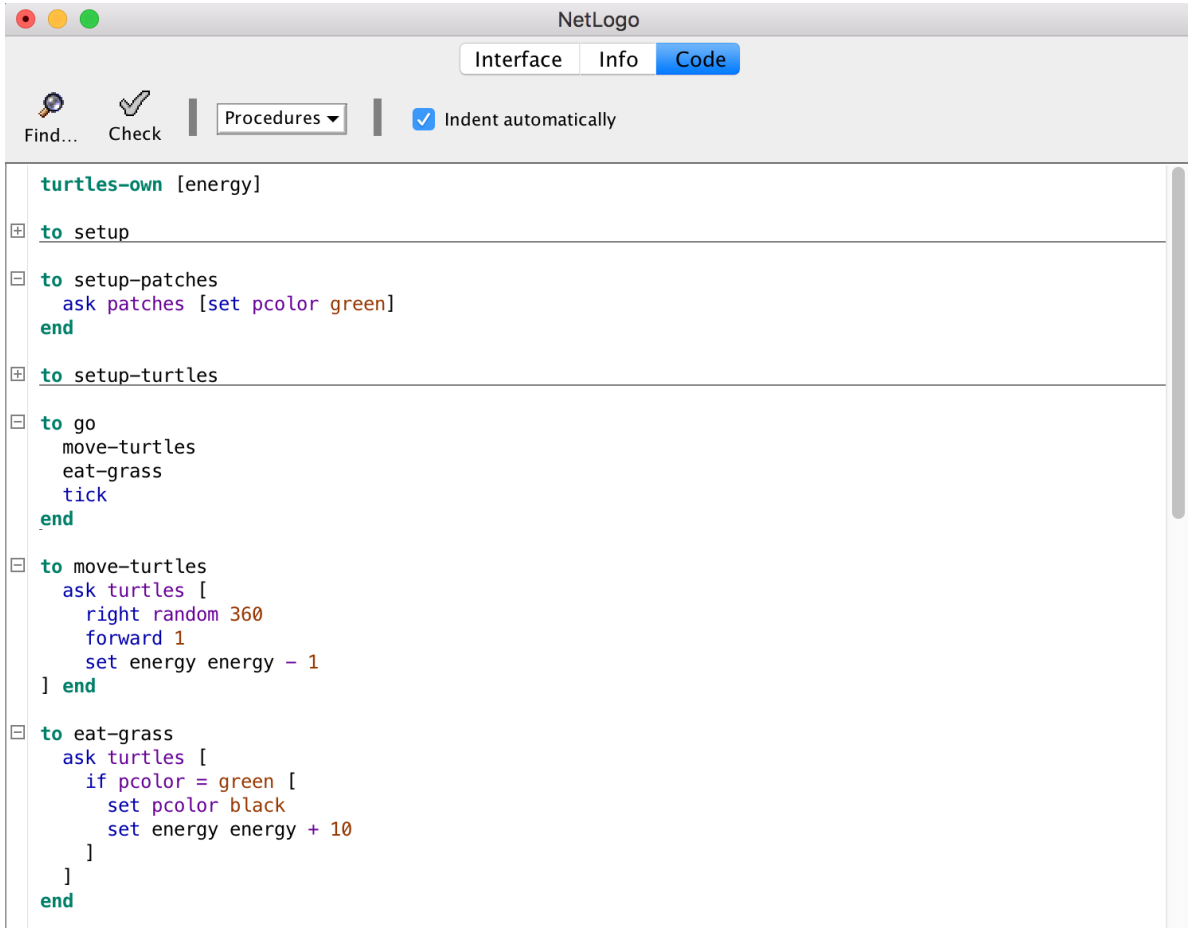
Now we add some interaction between turtles and patches:

- Turtles walk in the "grass" and eat it (i.e., the green patches), reproduce, and die.
- The grass will gradually grow again after it is eaten.
- We need a way to control when a turtle reproduces and dies, thus we introduce a turtle variable called **energy**:



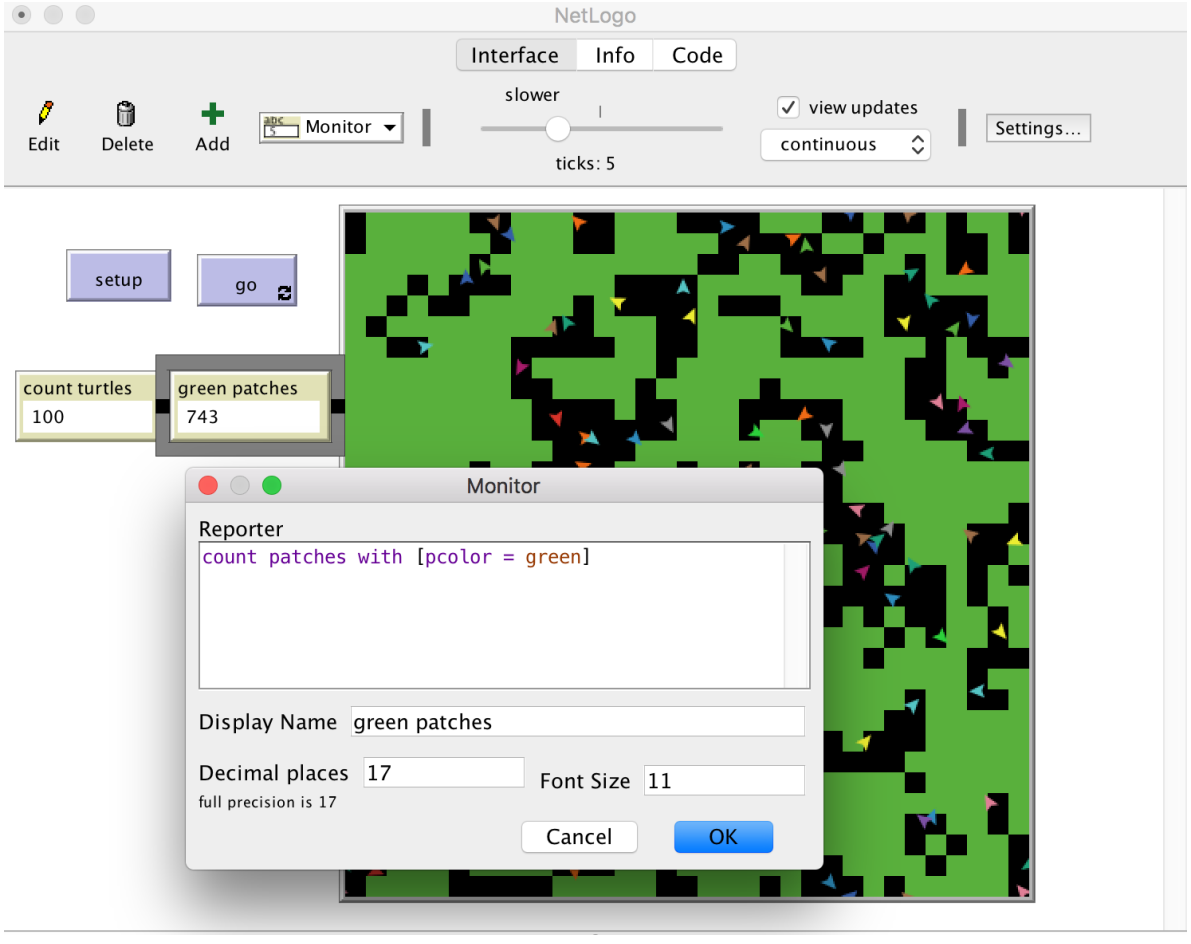
6. Turtle variables (II)

Then we modify the other turtles' procedures: when they move, they lose energy; if they eat grass, they increase it.



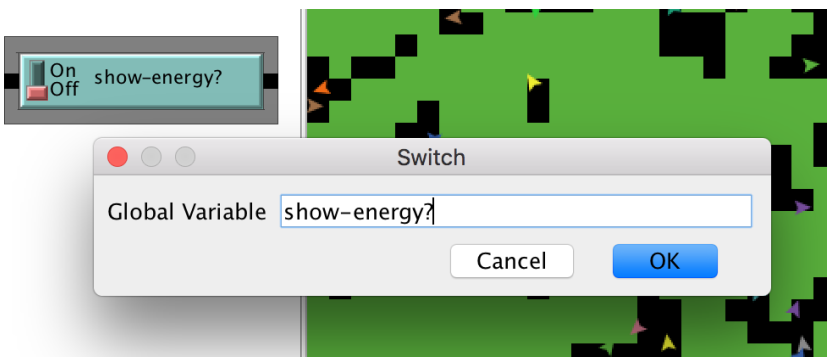
7. Monitors

We can create monitors to count our agents:



8. Switches and labels

It would be nicer if we could see every turtle's energy all the time, so we add a switch in the Interface tab and some changes in the Code tab:




```
to eat-grass
  ask turtles [
    if pcolor = green [
      set pcolor black
      set energy energy + 1
    ]
    ifelse show-energy?
      [ set label energy ]
      [ set label "" ]
  ]
end
```

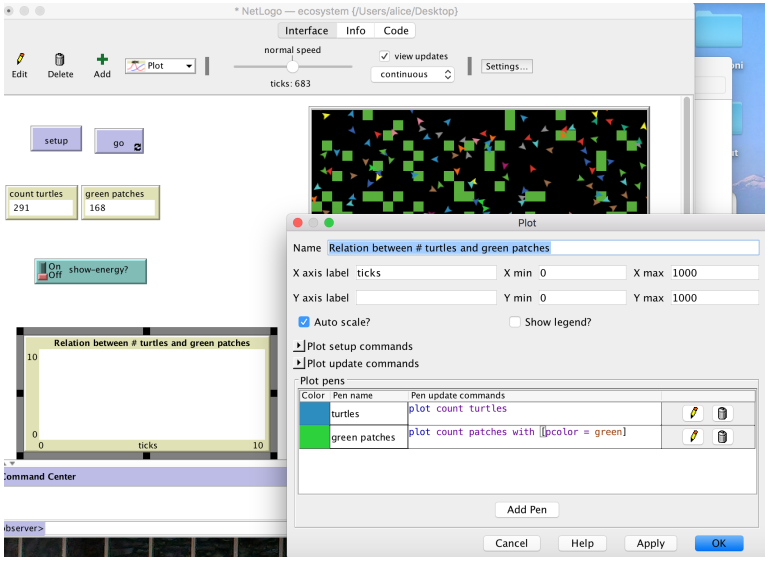
9. More procedures

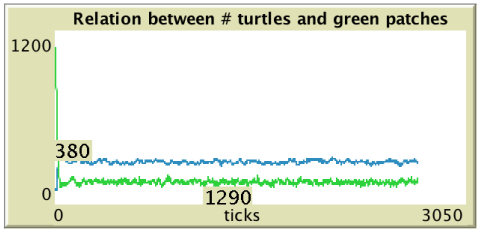
Let's add **reproduce**, **check-death** and **regrow-grass**:

```
to go
  move-turtles
  eat-grass
  reproduce
  check-death
  regrow-grass
  tick
end

to move-turtles

to eat-grass

to reproduce
  ask turtles [
    if energy > 50 [
      set energy energy - 50
      hatch 1 [ set energy 50 ]
    ]
]
end

to check-death
  ask turtles [
    if energy <= 0 [ die ] ]
end

to regrow-grass
  ask patches [
    if random 100 < 3 [ set pcolor green ] ]
end
```

10. Plotting

Now counts of turtles and green patches both fluctuate: is this pattern of fluctuation predictable? Is there a relationship between the variables? Let's add a **plot**:





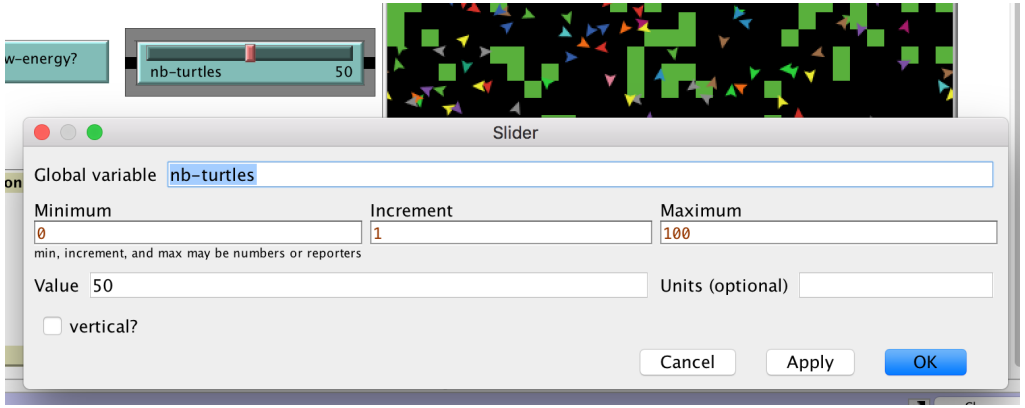
11. Tick counter

Let's stop the simulation at some point:

```
to go
  if ticks >= 500 [ stop ]
  move-turtles
  eat-grass
  reproduce
  check-death
  regrow-grass
  tick
end
```

12. Some more details

Finally we add a slider to decide each time the initial number of turtles and edit the code appropriately:




```
to setup-turtles
  create-turtles nb-turtles
  ask turtles [ setxy random-xcor random-ycor ] reset-ticks
end
```

What to bring home (I)

Resume of this first lecture:

- NetLogo offers several models from many disciplines to **simulate behaviors, actions and outcomes** of particular situations, according to parameters and variables set.
- Objects in the models are represented by **agents** we can define, change and interact with.
- NetLogo can also be used to **deploy and develop** models we prefer and are curious to investigate.

What to bring home (II)

In the next lectures:

- We'll reconstruct and discuss a model discussed during the mini-course by Prof. Benati (i.e., **Ethnocentrism**) and others new.
- If you are interested in modeling a pandemia (as we have been living since March 2020), take a look to Squazzoni et al. (2020), *Computational Models that Matter During a Global Pandemic Outbreak: A Call to Action* (you can find it in the Math Decisions repository on GitHub).

References

- Wilensky, U. (1999). *NetLogo*. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.
- Lopez-Sanchez M. (2020), *COVID-19 Virus Simulator – Adaptation of the Virus model from the NetLogo library*. http://www.maia.ub.es/~maite/CoronaVirus.html
- Squazzoni et al. (2020). *Computational Models that Matter During a Global Pandemic Outbreak: A Call To Action*. http://jasss.soc.surrey.ac.uk/23/2/10.html?fbclid=IwAR3tf1bGWi3SYs5ocKreEc6Z9PnqaO0C_20gjUJ1XrebRTa2UKDKfs1cF7A. Journal of Artificial Societies and Social Simulation 23 (2) 10