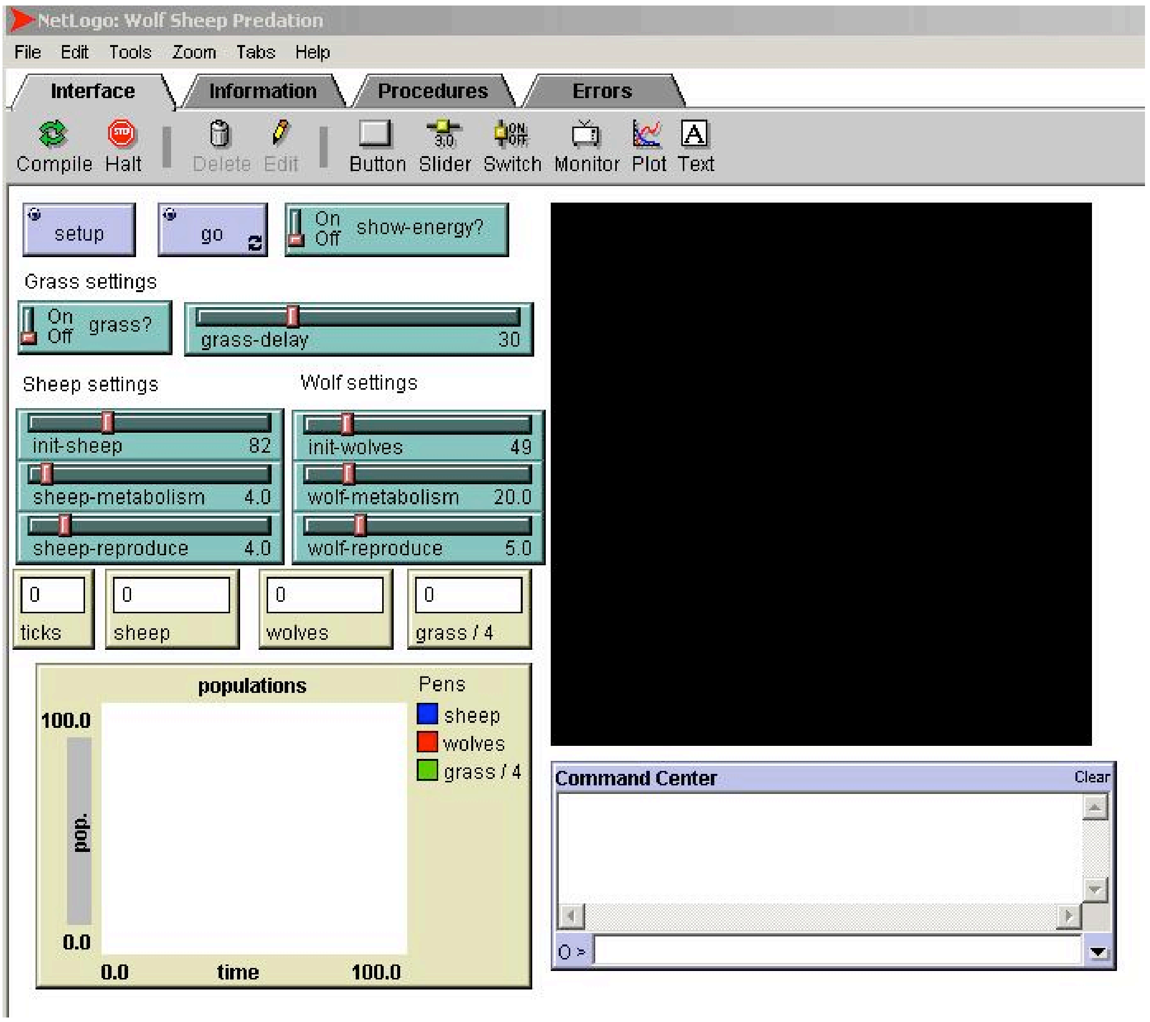
MODELING CHANGE WITH NETLOGO

NetLogo is similar to Agent Sheets in that it models behavior of individuals in a system. We can use the same type of logic we used in Agent Sheets, but with a slightly different implementation.

Messing About in NetLogo

First, the concept behind NetLogo. There are agents called turtles and there are agents called patches (the background). There are primitive (pre-defined) commands that control the characteristics and behaviors of the turtles and commands that impact the patches. Procedures can be used to control both. When you execute a command in NetLogo it always wants to know if the command should be executed by the Observer (you), the turtles (either some or all), or the patches (either some or all).

Let's begin by getting comfortable with the NetLogo interface. Open NetLogo and open a sample model from the Models Library (file menu). Choose the Biology folder, and select Wolf Sheep Predation.



show-energy? Off

grass? Off

grass-delay 30

init-sheep 82

sheep-metabolism 4.0

sheep-reproduce 4.0

init-wolves 49

wolf-metabolism 20.0

wolf-reproduce 5.0

ticks 0

sheep 0

wolves 0

grass / 4 0

There are a few things to note about this interface:

Tabs. Across the top are four tabs:
<u>Interface</u> is the main tab where you interact with your model.
<u>Information</u> is a text window intended for background information about the model.
<u>Procedures</u> is the tab where you can modify a sample model's code or write your own.
<u>Errors</u> is where you can find information about errors in your code.

Buttons. In this model there are two buttons, setup and go. You are likely to find these two buttons in every sample model and will want them in your models as well. Setup initializes variables and creates agents (turtles) according to your initial conditions. Buttons are created and used in the Interface tab.

Switches. Switches are used to control settings; typically whether or not a feature is turned on. They are created and used in the Interface tab.

Slider bars. Slider bars are used as input devices to control variables that are initialized as part of setup. Slider bars are created and used in the Interface tab and automatically become global variables for your procedures.

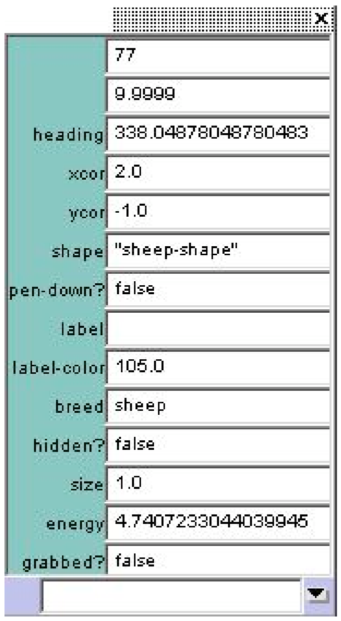
Monitors. Monitors are used to display the value of some variable, often used as counters. Monitors are created and used in the Interface tab, but the variable must already be defined.

Plot. Plot windows are used to plot graphs of variable values as they are changing. Plot windows are created in the Interface Tab, but the plotting functions must also be set up as a procedure.

Command Center. The command center allows you to enter primitive (pre-defined) commands on the fly, without having defined them in procedures. You can enter commands from the Observer, Turtle, or Patch perspective. The "O>" you see indicates that commands typed now will be executed as an observer. If you click on the "O>" you will see the option to choose Turtle or Patch.

First, let's look at what NetLogo can do. Click on Setup and then Go. When you want to stop the model, click on Go again. Note that the Go button has a pair of arrows in the lower right corner. This symbolizes that this button will continue to repeat the procedure it is calling over and over until you click on the button again. The setup button does not have these arrows and is used as a "do it once" button. If you click on the Go button again (after having stopped the model), it will continue the model where you stopped it. To restart the model, you need to first click on Setup again. Now change some of the slider bars and run the model again. Play with these variables (slider bars) to get the hang of using NetLogo.

At some point while your model is paused, right click (or CTRL click on a Mac) on one of the agents. Select "inspect turtle N". A pop up window (like the one here) displays all the pre-defined, as well as user-defined, information about that turtle. Note that each turtle has a "who" number (77 in this example); this is how individual turtles can be targeted and manipulated.



	77
	9.9999
heading	338.04878048780483
xcor	2.0
ycor	-1.0
shape	"sheep-shape"
pen-down?	false
label	
label-color	105.0
breed	sheep
hidden?	false
size	1.0
energy	4.7407233044039945
grabbed?	false

Now let's try a few commands to see how NetLogo works. In the Command Center (as Observer) type

```
ca           (press enter)
crt 10       (press enter)
```

`ca` is shorthand for Clear All. `crt 10` is shorthand for create-turtles 10. Notice that only one symbol appears on the screen. NetLogo places all new turtles at 0,0 (the center). Now type

```
ask turtles [fd 2]   (press enter)
```

fd 2 is short for forward 2. You should notice that the original placement pointed the turtles at regular intervals of 360 degrees. Now choose Turtles (instead of Observer) and type

```
fd 2
```

Notice that you *could* type all your commands as an Observer, but would need to always use "ask turtles" or "ask patches" to control the agents. Now, as an Observer, type

```
ca
crt 30
ask turtles [setxy random screen-size-x random screen-size-y set color blue]
```

Note that you may string together commands without any breaks or punctuation. Setxy is used to set the x,y coordinates of an individual agent. It expects two arguments (x and y). In this case "random screen-size-x" is the x argument. Screen-size-x is a primitive that is defined as the width of the screen (y is the height). Random expects one argument and is defined to choose a random number in the range of 0 to the number in the argument (in this case, screen-size-x).

We can use procedures to define various sequences of commands and buttons can be used to call the procedures.

Simple 'Flu Model: Healthy People Always Get Sick

Let's build the 'flu model again, from the individual (agent) point of view. We will have the following agents and associated behaviors:

Agent	**Behavior**
Healthy people	Move at random in the environment until they contact a sick person in an adjacent position. When this happens, the healthy person will become infected.
Sick people	Move at random in the environment.

First we need to start a new model, so select New in the File menu of NetLogo. Click on the Procedures tab; let's define some procedures, starting with "setup":

```
to setup
   ca
   crt 200
   ask turtles [
      if who = 0 [                     ;; choose one turtle, "0"
         set color red                 ;; set this turtle to red (sick)
         setxy (random screen-size-x) (random screen-size-y)
         ]                             ;; and randomly place it
      if who > 0 [                     ;; all other turtles
         set color blue                ;; set them to blue (healthy)
         setxy (random screen-size-x) (random screen-size-y)
         ]
      ]
   end
```

Now go to the Interface Tab to see what we have done. Click on the button tool, and click in the blank area of the tab to place the button. The code for this button is "setup". You may name the button anything you like; if you leave it blank it will be named "setup". Now click on the button and you should see one red and many blue "turtles" appear in the screen area, randomly placed.

Now let's create the main procedure that will call other procedures.

```
to run
   ask turtles [
      if color = blue [check_if_sick]   ;; check to see if turtle gets sick
      move                              ;; turtle will move randomly
     ]
end
```

We have named two procedures that do not yet exist (check_if_sick and move), so let's now define those:

```
to check_if_sick
   if color = blue [                       ;; only check if you are healthy
      if any turtles at-points [           ;; check for turtles next to you
         [-1 -1] [-1 0] [-1 1] [0 -1] [0 1] [1 -1] [1 0] [1 1]
         ]                                 ;; xy coordinates are relative
         with [color = red]                ;; if they are red (sick)
         [set color red]                   ;; then get sick
      ]
end

to move
   rt random 360                           ;; randomly turn
   fd 1                                    ;; move one step
end
```

We can now return to the Interface Tab to place a "run" button (the code should be whatever we named our main procedure, in this case, "run"). Click on the button. What happens? Do all the turtles get sick? How long does it take?

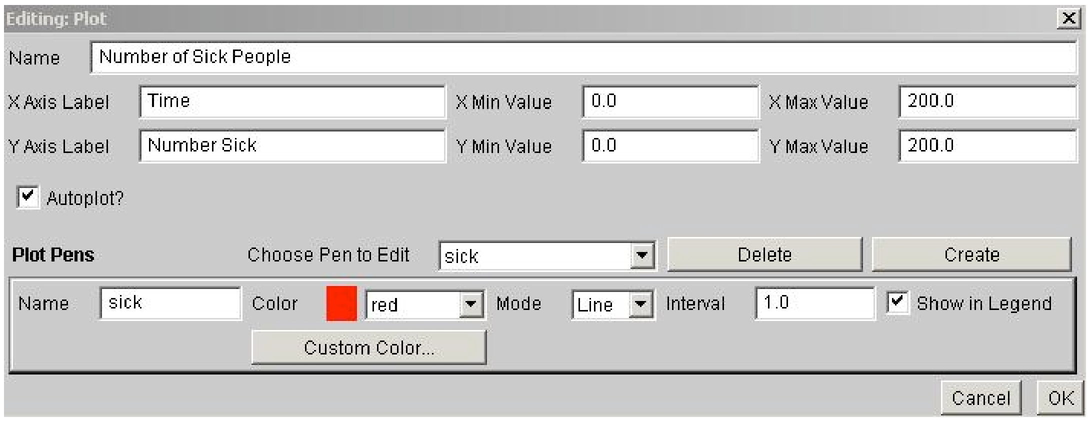
Now let's plot the number of sick turtles with plot procedure:

```
to plot_it
   set-current-plot "Number of Sick People"   ;; name our graph
   set-current-plot-pen "sick"                ;; define pens
   plot count turtles with [color = red]      ;; plot number of sick
   set-current-plot-pen "healthy"
   plot count turtles with [color = blue]
end
```

To run this procedure we can either create another button in the Interface Tab ("plot_it") or we can call it from the main run procedure by adding the following just before the end.

```
      plot_it                              ;; plot the number of sick turtles
```

We must also create a new plot in the Interface Tab, and must also define the pens ("sick" and "healthy") in that plot tool. Click on the Plot tool and place a plot, editing the Name, X and Y axis labels, and Plot Pens:



Name: Number of Sick People
X Axis Label: Time   X Min Value: 0.0   X Max Value: 200.0
Y Axis Label: Number Sick   Y Min Value: 0.0   Y Max Value: 200.0
✓ Autoplot?
Plot Pens   Choose Pen to Edit: sick
Name: sick   Color: 🟥 red   Mode: Line   Interval: 1.0   ✓ Show in Legend

Turtle Attributes: Counting the Number of Days A Person is Sick.

Suppose we want to have each sick person keep track of how long it is sick. Since each sick person may be sick for different amounts of time, we need each turtle to keep track of its sick time. This is accomplished via a local attribute, which we will call "sicktime." How should this work? Here is the logic:

At the beginning we must add our newly defined attribute:

```
turtles-own [sicktime]
```

In setup, for each turtle set, set the sicktime to 0

```
to setup
  ca
  crt 200
  ask turtles [
  if who = 0 [
    set color red
    set sicktime 0
    setxy (random screen-size-x) (random screen-size-y)
    ]
  if who > 0 [
    set color blue
    set sicktime 0
    setxy (random screen-size-x) (random screen-size-y)
    ]
  ]
end
```

In run, as we look over the world each time step, when we see a sick person we need to add one to its sicktime attribute.

```
to run
    ask turtles [
        if color = red [
            set sicktime (sicktime + 1)
            ]
        if color = blue [check_if_sick]
        move
        ]
    plot_it
end
```

To see what the sicktime is for any sick agent on the screen, right click (or CTRL click) on the turtle. All attributes for that agent will be listed in the panel that appears.

Global Variables: Using Constants and Keeping Track of the Total Number Sick

Suppose you want to set a constant (e.g., infection rate or number of days to recover) or keep track of the total number of sick people at any given timestep. These are considered global variables and can be introduced either via slider bars in the Interface tab, or via the "globals" command in the Procedures tab. Suppose you'd like to use a turtle's sicktime to set a recovery rate, e.g., N timesteps to recover. Go to the Interface Tab and click on the slider tool and then click to place a slider in the window.

Name it "recovery_time"; now "recovery_time" can be used in your procedures:

```
to run
    set days (days + 1)
    ask turtles [
        if color = red [
            set sicktime (sicktime + 1)
            if sicktime > recovery_time [set color green]
            ]
        if color = blue [check_if_sick]
        move
        ]
    plot_it
end
```

Given that turtles can now recover, we might want to not only know the total number who are currently sick, but also the total number of turtles who have been sick at all. For this variable we don't need the user to set it, we just need to have it available for monitoring. For this we need to set global variables, which must be done in the Procedures Tab, before any procedures.

```
globals [total_sick]
```

To count the total sick turtles we would increment this variable each time a turtle gets sick:

```
set total_sick (total_sick + 1)
```

To monitor this number as the model runs we need to add a monitor window on the Interface Tab. Click on the monitor tool and place it in a blank area of the window. The "reporter" is the variable you want to monitor, in this case "total_sick". You can name the Monitor window anything you like.

Problems to Try

Not every contact results in sickness: Change the 'flu model to include the fact that a healthy person only gets the flu from a sick person 25% of the time. Experiment with the percentage to see how the behavior of the model changes.

Add death to the 'flu model: Suppose a sick person dies after 10 time steps. Change the 'flu model to reflect this. If the dead people are not contagious, do all of the healthy people get the 'flu? Don't forget to count these people and graph them, too.

Add recovery to the 'flu model: Suppose a sick person recovers after 20 time steps. Do all of the people become immune? What happens if 99% of the people recover and 1% die from the 'flu? Investigate how having the sick person become healthy (and susceptible) again differs from creating a new, immune depiction.

Add a stationary treatment facility to the 'flu model: Assume that if sick people find treatment, they recover and become immune to the 'flu.

Population Growth: Build a model of population growth given the following situation:

- There are males and females that reproduce 5% of the time when they come into contact.
- Half the time one male offspring is born and half the time one female offspring is born.
- When males and females have been around 10 years, they die off.

A more sophisticated population model: In the model above, suppose only adults (males and females older than 2 years) reproduce.

A model of crystallization: Molecules are moving around at random, with one "crystallized" molecule stationary in the middle. When molecules come in contact with the crystallized ones, they attach and become part of the stationary crystal.

Predator – Prey Interaction: Build a model in which prey and predators are born and die monthly as follows:

- 5% of the time when two prey come into contact, prey are born into all available adjacent locations.
- 1% of the time when two predators come into contact, one new predator is born if there is an adjacent location available.
- 20% of the time prey are eaten if the predators come in contact with them.
- predators die if they go without eating for more than 5 months.

How might you expand the model to include predators being satiated and so not hungry?

A Model Intersection: Build an intersection with a traffic light that cycles between green and red every 10 seconds. Build car agents that run on the roads and can recognize the light. Add a very small probability that a car will ignore the light and monitor all "crashes."