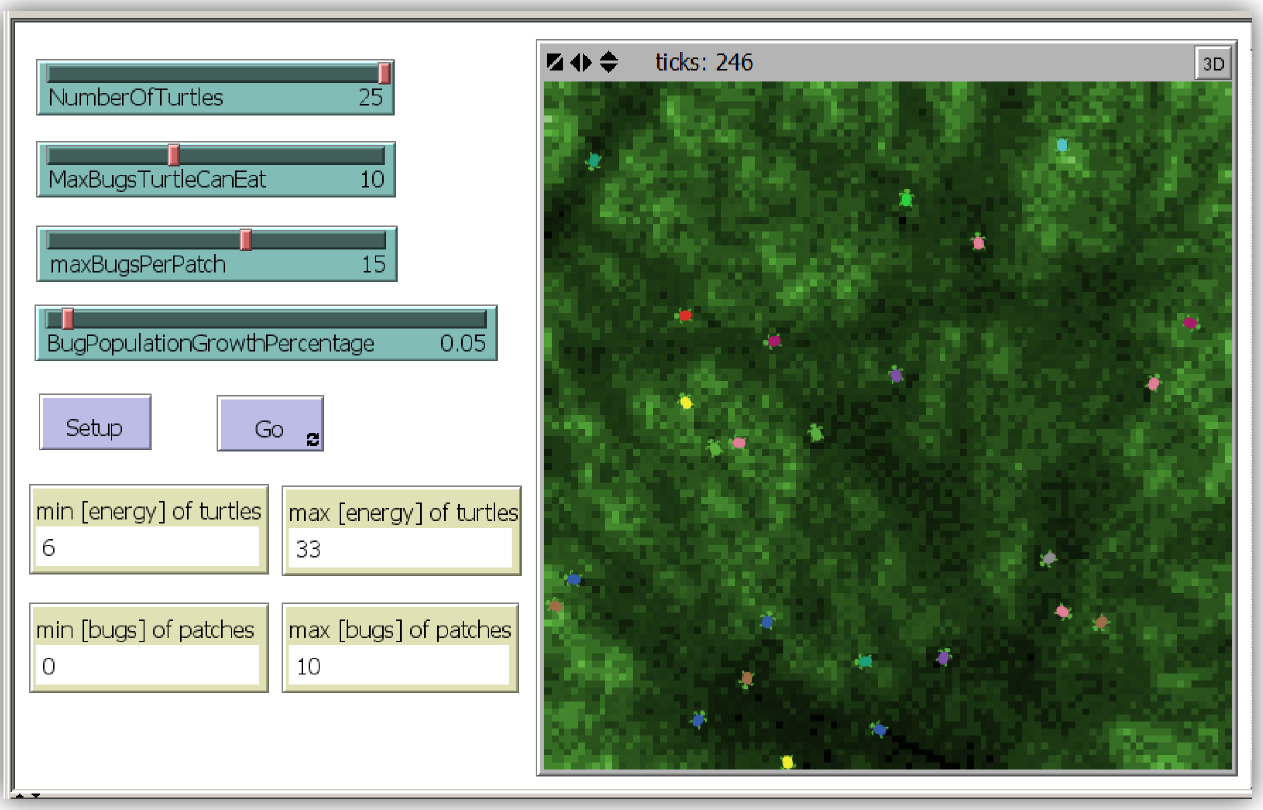
Lab 6: Energizer Turtles


NumberOfTurtles 25
MaxBugsTurtleCanEat 10
maxBugsPerPatch 15
BugPopulationGrowthPercentage 0.05
Setup
Go
min [energy] of turtles 6
max [energy] of turtles 33
min [bugs] of patches 0
max [bugs] of patches 10
ticks: 246

Screen capture showing the required components:
4 Sliders (as shown)
2 Buttons (as shown)
4 Monitors (as shown)
min-pxcor = -50, max-pxcor = 50, min-pycor = -50, max-pycor = 50

The fact that in the screen capture the turtles have a turtle shape is not a requirement. You may give the turtles any shape you want – including the default arrow shape.

Energizer Turtles is a model where, in addition to their usual properties, turtles have **energy** and patches have **bugs**. Just as each turtle in the model has its own location, its own color, its own pen state (up or down), and its own heading, each turtle also has a programmer defined custom property, called an ***agent variable*** in Netlogo. The programmer can tell Netlogo that each turtle has the agent variable called **energy**, by using the following statement:

```
turtles-own [energy]
```

Similarly, the programmer can tell Netlogo that each patch as the agent variable called bugs with the statement:

```
patches-own  [bugs]
```

In the above Netlogo statements, the words shown in teal, `turtles-own` and `patches-own`, are reserved words in the Netlogo language. The words shown in black, **`energy`** and **`bugs`**, are words made up by the programmer of this particular model. They can be any sequence of characters that follow Netlogo's naming rules.

Model Overview:

The basic idea of the ***Energizer Turtles*** model is that turtles need energy. They move quickly when they have a lot of energy and cannot move at all when they do not have any energy. Turtles get energy from eating bugs on the patch where they are standing.

Turtle Setup:

1) The number of turtles created must be equal to the slider variable setting for **`NumberOfTurtles`**. This must be an integer no smaller than 1 and no larger than 25.
2) Each turtle must start in a uniformly distributed, random location within the Netlogo 2D world view.
3) Each turtle must start with a uniformly distributed random heading.
4) Each turtle must be of a color, size and shape that is clearly visible yet does not interfere with seeing other aspects of the model.
5) Each turtle must start with its **`energy`** set to 10 units.

Patch Setup:

1) All patches must be assigned an amount no less than 0 and no greater than the slider value **`maxBugsPerPatch`**. You have some option here. I assigned every patch a uniformly distributed, random amount of energy between 0 and 75% of **`maxBugsPerPatch`**. You may do the same or use a triangular distribution or place clusters of high bug concentrations or whatever you think works best. The only requirements are that each patches value must be

no less than 0, no greater than **maxBugsPerPatch** and that the values usually lead to interesting behavior in the model. For example, all cells having zero bugs makes a very uninteresting run of the model.

2) The color of each patch must indicate its bug population. In particular, a patch with zero bugs must be black. A patch with bugs **maxBugsPerPatch** must be white. Patches with in between numbers of bugs must be in between shades of green (or some other color). This is most easily done using Netlogo's `scale-color` command (see hint below).

Hint: `scale-color`

Syntax:

`scale-color` *color number lowValue highValue*

Reports a shade of color proportional to the value of *number*.

color can be **gray**, **red**, **orange**, **brown**, **yellow**, **green**, **lime**, **turquoise**, **cyan**, **sky**, **blue**, **violet**, **magenta** or **pink**.

If *number* is less than *lowValue*, then the darkest shade of color is chosen (generally, this is black or so near black that it looks black).

If *number* is greater than *highValue*, then the lightest shade of color is chosen (generally, this is white or so near white that it looks white).

Note: for color shade is irrelevant, e.g. green and green + 2 are equivalent, and the same spectrum of colors will be used.

Example:

Try to predict what the code below will do. Then type it into a NetLogo program and see what it does.

```
ask patches
[
  set pcolor scale-color red  pxcor  min-pxcor  max-pxcor
]
```

Each Turtle on Every Tick (Eat, Turn and Walk):

Eat

1) If a turtle is on a patch with some bugs, it gobbles them up. Keep in mind that in this model, bugs are NOT agents. In this model, each patch has an ***agent variable*** called **`bugs`**. Thus, having the turtle "eat" 10 bugs from a patch subtracts 10 from that patch's **`bugs`** variable.
2) A turtle will always eat as many bugs as it can (see below for the rules).
3) A turtle cannot eat more bugs in one tick than the slider value of **`MaxBugsTurtleCanEat`**.
4) A turtle cannot eat more bugs than are present on the patch on which it is standing.
5) A turtle cannot eat fractional bugs.
6) A turtle cannot eat less than zero bugs (turtles cannot puke bugs back into life).
7) A turtle gains one energy point for each bug that it eats.

Turn

1) Each turtle makes a wiggle turn. I used a wiggle angle of 5 degrees. You may use that or you may use a different angle if you thing it gives more interesting behavior. Note: if we were trying to create a model of real turtles of a particular species searching for insects in some real landscape, then we should observe the actual turtles or videos of the turtles. Then, we should try to find a movement model that matches the real-life observations. In this model, however, the goal is not to model anything real. In this model, we are simply looking for interesting emergent behaviors.

Walk

1) In this model, the world does NOT wrap horizontally nor does it wrap vertically. For info on how to set this and how to deal with it in code, see CS4All instructor Nick Bennett's excellent video titled "Conditional Control Flow" from week 5.
2) The reason for calling this model ***Energizer Turtles*** is that turtles move faster when they have more energy. In particular, the distance the turtle will try to move is equal to 1/10th its current energy. Thus, if a turtle has an energy of 24, then it will try to move forward 2.4 patches.
3) When a turtle cannot move, because it near the edge, it makes a 180 degree turn rather than moving forward (again, for details, see the video "Conditional Control Flow" from week 5).
4) When a turtle moves, it loses energy equal to the distance it moved.
5) A turtle may never have less than 0 energy.
6) If a turtle has zero energy, it does not move (0/10 = 0 distance).
7) A turtle only loses energy when it moves.
8) In this model, turtles are immortal (they never die).

Each Patch on Every Tick (Multiply, Spread and Color):

Multiply

One thing bugs are good at is making more bugs. In this model, you will use the exponential growth equation:

```
set bugs (bugs + (bugs * BugPopulationGrowthPercentage / 100))
```

where,

`BugPopulationGrowthPercentage` is set with a slider.

Since the slider value is a percentage, it is divided by 100. Thus, if the setting were 100%, the equation would become:

```
set bugs (bugs + bugs)
```

That is, if the slider were set to 100%, the number of bugs in each patch would double every tick! This is way too fast for the turtles to keep up with. A good value for this growth rate is more like 0.05%!

Notice that the equation above will usually increase the bug population of a patch by a non-integer number. This might seem odd at first. This is necessary because increasing the population by 1 each tick is too fast a growth rate, yet increasing it by 0 is too slow. Think of these fractional populations as being still in the egg state. If in every tick, the bug population of a patch increases by 0.2, then after 5 ticks, one new full bug will emerge.

Note: Netlogo has a **`hatch`** command used to "hatch" new turtles or other agents. DO NOT use this for the bugs. Remember, in this model, the bugs are NOT agents. In this model **`bugs`** is just an agent variable of turtles NOT itself an agent.

Spread

1) A bugs life is not just having babies: Every tick, each patch uses Netlogo's **`neighbors`** reporter (see below).
2) If a patch has a neighbor with less than half the number of bugs that it has AND that patch has at least 2 bugs, then one of the bugs "moves" to its neighbor. I put "moves" in quotation marks because since in this model, bugs are NOT agents, they cannot actually move. The way to "move" a bug from one patch to another is to subtract 1 to the number of bugs in the first patch and add 1 to the number of bugs in the second patch.

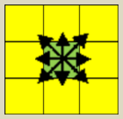
Color

After growth and spread, set each patch color the same as was done in setup.

Hint: Netlogo's neighbors reporter

neighbors

This reporter can only be used within a patch context. It reports the set of patches that are adjacent to the current patch. Usually, this is the surrounding patches, but when the patch is on the edge and world wrapping is turned off, then less than 8 neighbors will be reported.



Example: Add the code below to some model's setup button. Try to predict what will happen, then run the code and see what happens.

```
ask patches
[
  let neighborsInBox 0
  ask neighbors with [pxcor > 3 and pycor < 3]
  [ set neighborsInBox (neighborsInBox + 1)
  ]
  set pcolor green + (neighborsInBox)
]
```

Slider: NumberOfTurtles

Must have Minimum = 1, Maximum = 25, and Increment = 1.

Slider: MaxBugsTurtleCanEat

Must have Minimum = 1, Maximum = 25, and Increment = 1.

Slider: MaxBugsPerPatch

Must have Minimum = 1, Maximum = 25, and Increment = 1.

Slider: BugPopulationGrowthPercentage

Must have Minimum = 0, Maximum = 1, and Increment = 0.01.

Monitors:

The equation needed in each monitor is shown in the screen capture on the first page. Set each monitor to display 1 decimal place. If min [energy] or min [bugs] is ever less than 0, then your model has an error.

Grading Rubric [20 points total]:

[A: 1 points]: Submit Netlogo source code named: `W6.`*firstname*`.`*lastname*`.nlogo`.

[B: 1 points]: The first few lines of your code tab are comments including your name, the date, your school, and the assignment name.

[C: 2 points]: The code in the code tab of your program is appropriately documented with "inline comments".

[D: 2 point]: Your program's interface includes the required 4 sliders, 2 buttons and 4 monitors. Your layout must be neat, but can be organized as you like. The labels on your components must clearly indicate the component's function, but you may choose the words spacing, etc.

[E: 2 point]: Your setup procedure must set up the turtles as required above.

[F: 2 points]: Your setup procedure must set up the patches as required above

[G: 3 points]: Your go procedure must ***move*** the turtles as required.

[H: 1 point]: Your turtles must never have less than zero energy.

[I: 2 point]: Your go procedure must ***multiply*** the bugs as required.

[J: 2 point]: Your go procedure must ***spread*** the bugs as required.

[K: 2 point]: Your go procedure must ***color*** the patches required.

Extra credit: Collision Detection [+5]:

If a turtle collides with another turtle, then have your program do something that prevents one from passing through (or jumping over) the other. You can reverse their headings, or have them pick random headings (in such a way that they do not pass through each other). You could have one transfer some of its speed to the other or (if you want to get very fancy) have them move as though they had an elastic collision. For this, you could let all turtles have the same mass or you could make some larger turtles that have more mass.

Extra credit: Seeker Energizer Turtles [+5]:

Add a switch to your model to that when it is "on", your turtles become seeker turtles. A ***seeker turtle*** is one that moves the same as a normal energizer except for when it hits one of the outer edges. When a ***seeker turtle*** hit an edge, rather than turning 180 degrees, it turns to face the nearest patch with the most bugs.