Exercise 4 – Wolf Sheep Predation model

Open NetLogo

From the menu select File>Models Library.

Then select Sample Models>Biology>Wolf Sheep Predation.

Click on the Info tab to see information about the model. It implements a simple form of biological ecosystem involving predators (wolves) and prey (sheep). It was developed for educational purposes with high school students[2].

After reading the Info page click back to the Interface screen and play with the model by running it a number of times with different parameters. Try running it with the "grass?" switch both on and off.

Select the Code tab. You should see something similar to:

```
globals [grass]  ;; keep track of how much grass there is
;; Sheep and wolves are both breeds of turtle.
breed [sheep a-sheep]  ;; sheep is its own plural, so we use "a-sheep" as the singular.
breed [wolves wolf]
turtles-own [energy]       ;; both wolves and sheep have energy
patches-own [countdown]

to setup
  clear-all
  ask patches [ set pcolor green ]
  ;; check GRASS? switch.
  ;; if it is true, then grass grows and the sheep eat it
  ;; if it false, then the sheep don't need to eat
  if grass? [
    ask patches [
      set pcolor one-of [green brown]
      if-else pcolor = green
        [ set countdown grass-regrowth-time ]
        [ set countdown random grass-regrowth-time ] ;; initialize grass grow clocks
                                                     ;; randomly for brown patches
    ]
  ]
  set-default-shape sheep "sheep"
  create-sheep initial-number-sheep  ;; create the sheep, then initialize their variables
  [
    set color white
    set size 1.5  ;; easier to see
    set label-color blue - 2
    set energy random (2 * sheep-gain-from-food)
    setxy random-xcor random-ycor
  ]
  set-default-shape wolves "wolf"
  create-wolves initial-number-wolves  ;; create wolves, then initialize their variables
  [
    set color black
    set size 2  ;; easier to see
    set energy random (2 * wolf-gain-from-food)
    setxy random-xcor random-ycor
  ]
  display-labels
  set grass count patches with [pcolor = green]
  reset-ticks
end
```

---

[2] Wilensky, U. & Reisman, K. (2006). Thinking like a Wolf, a Sheep or a Firefly: Learning Biology through Constructing and Testing Computational Theories – an Embodied Modeling Approach. Cognition & Instruction, 24(2), pp. 171-209. http://ccl.northwestern.edu/papers/wolfsheep.pdf

```
to go
  if not any? turtles [ stop ]
  ask sheep [
    move
    if grass? [
      set energy energy - 1  ;; deduct energy for sheep only if grass? switch is on
      eat-grass
    ]
    death
    reproduce-sheep
  ]
  ask wolves [
    move
    set energy energy - 1  ;; wolves lose energy as they move
    catch-sheep
    death
    reproduce-wolves
  ]
  if grass? [ ask patches [ grow-grass ] ]
  set grass count patches with [pcolor = green]
  tick
  display-labels
end

to move  ;; turtle procedure
  rt random 50
  lt random 50
  fd 1
end

to eat-grass  ;; sheep procedure
  ;; sheep eat grass, turn the patch brown
  if pcolor = green [
    set pcolor brown
    set energy energy + sheep-gain-from-food  ;; sheep gain energy by eating
  ]
end

to reproduce-sheep  ;; sheep procedure
  if random-float 100 < sheep-reproduce [  ;; throw "dice" to see if you will reproduce
    set energy (energy / 2)                ;; divide energy between parent and offspring
    hatch 1 [ rt random-float 360 fd 1 ]   ;; hatch offspring and move it forward 1 step
  ]
end

to reproduce-wolves  ;; wolf procedure
  if random-float 100 < wolf-reproduce [  ;; throw "dice" to see if you will reproduce
    set energy (energy / 2)               ;; divide energy between parent and offspring
    hatch 1 [ rt random-float 360 fd 1 ]  ;; hatch an offspring and move it forward 1 step
  ]
end

to catch-sheep  ;; wolf procedure
  let prey one-of sheep-here                    ;; grab a random sheep
  if prey != nobody                             ;; did we get one?  if so,
    [ ask prey [ die ]                          ;; kill it
      set energy energy + wolf-gain-from-food ] ;; get energy from eating
end

to death  ;; turtle procedure
  ;; when energy dips below zero, die
  if energy < 0 [ die ]
end

to grow-grass  ;; patch procedure
  ;; countdown on brown patches: if reach 0, grow some grass
  if pcolor = brown [
```

```
    ifelse countdown <= 0
      [ set pcolor green
        set countdown grass-regrowth-time ]
      [ set countdown countdown - 1 ]
  ]
end

to display-labels
  ask turtles [ set label "" ]
  if show-energy? [
    ask wolves [ set label round energy ]
    if grass? [ ask sheep [ set label round energy ] ]
  ]
end

; Copyright 1997 Uri Wilensky.
; See Info tab for full copyright and license.
```

Discussion of code:

Declarations:

At the start of the code one global variable is declared (grass). All turtles a given an additional variable (energy). All patches are given an additional variable (countdown).

Notice the lines:

```
breed [sheep a-sheep]
breed [wolves wolf]
```

The breed command creates new turtle types that are a subset of all turtles and inherit turtle properties (agent variables). After the breed command two words are supplied that refer to the plural and singular of the breed. Once defined, all the commands that can be applied to turtles can now be applied to individual breeds. This aids clarity when models contain several turtle types – as in this example where we have sheep and wolves.

Setup procedure:

Firstly all patches are set to green. Then if the “grass?” switch is set to on (in the Interface) patches are set to green or brown randomly. If the patches are green then the countdown patch variable is set the grass-regrowth-time (specified by a slider in the Interface). If the patches are brown then countdown is set to a random value between 0 and grass-regrowth-time.

Next the population of sheep are created and initialised. Notice the command:

```
create-sheep initial-number-sheep
```

This as an identical function to the “create-turtles” command except it only applies to the sheep breed that was declared. Essentially this amounts to creating a turtle with a special type variable containing the breed (in this case sheep). Sheep are initialised with a randomised amount of energy and a random position in addition to setting various display options.

In a similar way a population of wolves are initialised.

Next a call is made to the “display-labels” procedure (defined at the end of the code) which displays the energy level of sheep and wolves as a label depending on the settings of the “show-energy?” and “grass?” switches on the Interface screen. Finally the “grass” global variable is set to a count of the patches that are green.

Go procedure:

Firstly the program terminates if there are no turtles (i.e. if all wolves and sheep have died). Notice the use of the “any?” operator that returns true if what follows is non-empty.

Next each member of the sheep population is asked to move, eat grass, die and reproduce. Each of these actives is specified within a procedure defined later in the code. Notice that the eat grass activity is only activated if the “grass?” switch is on (set in the Interface screen).

The move procedure rotates the sheep in a random direction over a 100 degree range and then moves forward one patch length. Notice that “rt”, “lt” and “fd” are shorthand for “right”, “left” and “forward”.

The eat-grass procedure looks at the colour of the patch the sheep is on and if it is green then sets the colour to brown and increments the sheep energy variable by the sheep-gain-from-food value (which is set by a slider on the Interface). Notice that the “pcolor” patch variable can be used directly without reference to the patch that the sheep is on. When a turtle references a patch variable directly it is taken from the patch the turtle is currently on.

The death procedure causes the sheep to die (that is deleted from the population) if its energy value is less than zero.

The reproduce-sheep procedure probabilistically decides if to hatch a new sheep based on the value of “sheep-reproduce” (set in the Interface). By executing the “hatch 1” command the sheep produces a single clone of itself. By dividing the sheep energy values by 2 and then hatching this effectively divides the value between the parent sheep and the offspring. Note that new offspring are pointed in a random direction and move forward one patch length.

After all sheep have executed their commands the wolves are asked to perform a similar set of commands.

However wolves execute the catch-sheep procedure which simulates wolves catching and eating sheep to get energy. Firstly a local variable “prey” is initialised to a randomly selected sheep at the current patch:

```
let prey one-of sheep-here
```

The command “sheep-here” returns a set of all sheep on the current patch of the wolf. The “one-of” command selects one randomly from the set. Notice the use of “nobody” in the conditional:

```
if prey != nobody
```

If no sheep was found in the current patch then prey is set to the special value “nobody”. If a sheep is found then it is asked to die and the energy of the wolf is increased by the wolf-gain-from-food amount (set in the Interface).

After all wolves have executed their commands the grow-grass procedure is called for each patch (assuming the “grass?” switch is on, which is set in the Interface).

The grow-grass procedure implements a timed regrowth of any eaten grass. It checks if the patch colour (pcolor) is brown. If so it checks if the patch variable countdown has reached zero. If it has then the patch colour is set to green and the countdown variable set to the grass-regrowth-time (set in the Interface). Otherwise the countdown is decrement.

Finally the display-labels procedure is called which updates the labels attached to wolves and sheep to display their current energy. The “round” command forces the number to be displayed as the nearest whole number.

Task: Modify the wolf sheep predation model so you include another breed of turtles called “humans” which feed on wolves. In all other respects make humans behave like wolves the only difference is that they catch wolves rather than sheep. Add three sliders to the Interface to set the parameters for humans (in a similar way to the wolf and sheep sliders). In the same way as the sheep and wolves, in the Interface, add a line to the plot for the total number of humans and add a monitor to show the actual value of the total number of humans. For display purposes note that there is a shape called “person”.

Task: Experiment with parameter values using the sliders on the interface to determine if you can find values that keep humans, foxes and sheep coexisting in the environment. What did you find? Why?

Task: Modify the behaviour of humans so they feed on sheep not wolves but only eat a sheep with a probability set by a slider. If wolves and humans find themselves sharing a patch they fight but do not eat each other. If there are more wolves in the patch than humans the wolves win and the humans die otherwise the humans win and the wolves die What happens when you run it?