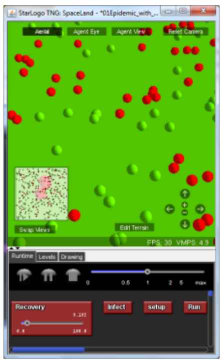
Although we are dealing principally with ‘ideas’ models, we often want to quantify the behaviour we observe. Using another simple StarLogo model we look at some of the tools that can help do that. We also introduce a similar text based coding environment to analyse a more complex model.

Preparation required:
Epidemic sample programs available to all. StarLogoTNG and NetLogo installed.

Pandemic Panic

We start by looking at a way to model the spread of an epidemic. There have been several pandemic alerts in recent years involving humans; the Ebola outbreak, swine flu, Zika virus or Avian Flu for example. There are many other examples in the natural world, particularly agricultural concerns. Think of Foot and Mouth, ‘Mad Cow’ disease or the current concern about TB. In all these cases, the very genuine concerns arise from modelling scenarios based on an understanding of how the particular infections spread.

We’ll use another exemplar model from the curriculum materials provided by the Scheller Teacher Education Partnership as our starting point. It is a simple model, but very effective for getting over the ideas of how a pandemic can quickly emerge (or be stopped).

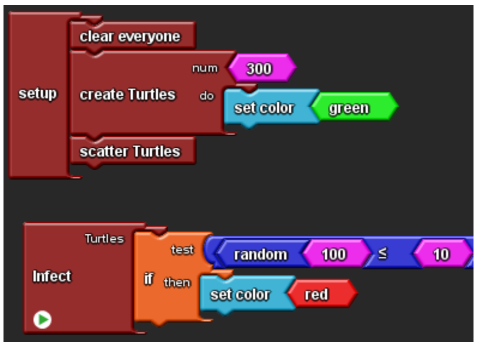


Open the model *01Epidemic_with_recovery.sltng*. As usual we have Setup and Run buttons, but you’ll notice we also have a third button, and a slider. The model is initialised as the diagram shows. Try setting the Recovery slider to about 8 in the first instance and watch the model.

We use red spheres to represent infected agents. When they collide with green agents the infection is passed on. But observe the infected agents – at some point they change back to green. They have recovered. It would be nice if we could plot the number of infected agents, and observe the effect of altering the recovery rate. We’ll do that later, but first let’s investigate how the model works.

How It Works

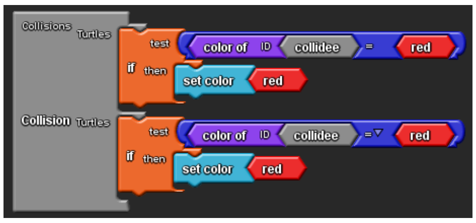
The model is quite simple – we should already be familiar with all the commands used. Again, reading the code and explaining it can be a good class exercise.



```
setup
    clear everyone
    create Turtles  num 300
                    do  set color green
    scatter Turtles

Infect   Turtles
    if test  random 100 ≤ 10
       then  set color red
```

Create the population: The Setup procedure creates and scatters our ‘turtles’. We’ve edited the breed, so it uses a sphere for the image, and we set those to green. To seed the infection we generate a random number for each agent, and if it is less than or equal to 10, we set its colour to red, indicating it is infected. So roughly 10% of the population will start infected – though the exact number is unknown.

Make them mobile: Our Run button instigates a familiar wiggle walk.



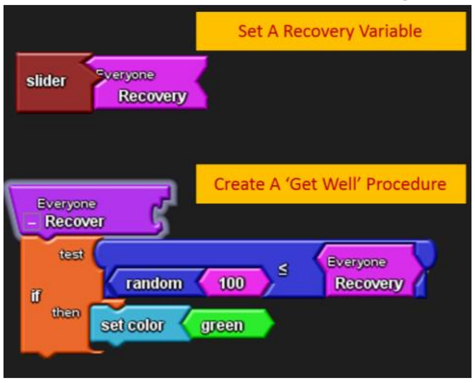
```
Collisions  Turtles
    if test  color of ID collidee = red
       then  set color red
Collision   Turtles
    if test  color of ID collidee = red
       then  set color red
```

Spread infection on contact: Note the code for a collision. Can anyone explain the top conditional statement? If the agent that bumps into the turtle is red, set this turtle to red, so passing on the infection. But why do we have a second conditional? Because we need to consider the collision from the perspective of both agents. So in each conditional, a different turtle is the collidee.

With just this simple code, we have the basis for a model. Children may be able to build this unaided, or it could be supplied and they could be challenged to say why it is flawed, and asked to improve it.

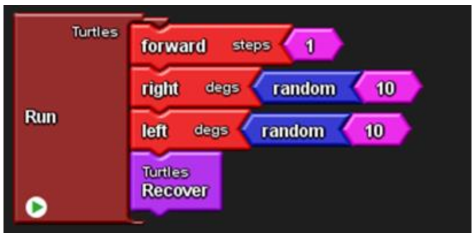
The important point to stress is that evaluation is an essential part of computational thinking. Identifying flaws and making judgements about what and how to improve their code is important. The video on the next slide demonstrates the problem. What is wrong? You may get a variety of answers – nobody dies for example, or sick people should stop moving and go to bed. These are likely to be valid points and a list could be made. Consideration could then be given to which are most important in an 'ideas' model that tries to represent the spread of infection.

The key flaw: without a way to recover, sooner or later everybody is infected. This is unrealistic. Building recovery into our model is a three step process.



Set a recovery variable: In which space is the Recovery variable created? It uses the Everyone space. We haven't used this before. We could have created it in the Turtle space. By creating it for Everyone, if we add other breeds to our model they too can access the variable. It acts as a 'global' rather than 'local' variable. Note the slider block. This is a built in 'widget'. It is automatically added to SpaceLand, allowing us to set and vary the Recovery variable at runtime.

Create a 'get well' procedure: Can anyone explain how the Recover procedure works? It uses the same technique used to seed the infection. We generate a random number for an agent between 1 and 100. If it is less than or equal to the Recovery variable, set by the slider, we set the agent green.



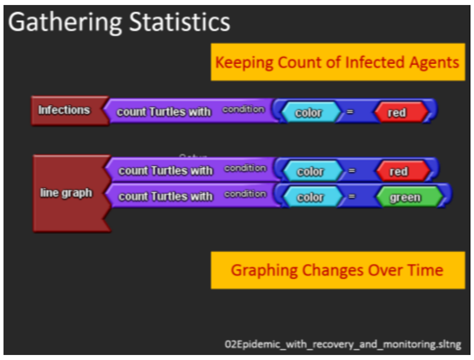
Call the procedure on each step: Having defined the Recover procedure, we call it on each repetition of the forever loop that runs the program. A good test of understanding might be to ask which Turtles it is applied to. Every Turtle will generate a procedure call. Since healthy agents are already green, it doesn't matter if they aren't set to green by the procedure.

Gathering Statistics

Now that we have a basic model we might want to experiment by altering either the starting numbers infected, or the recovery rate. Altering the recovery rate is achieved by moving the slider. How would we change our seeding of the infection? We would need to alter the number used in the Infect test.

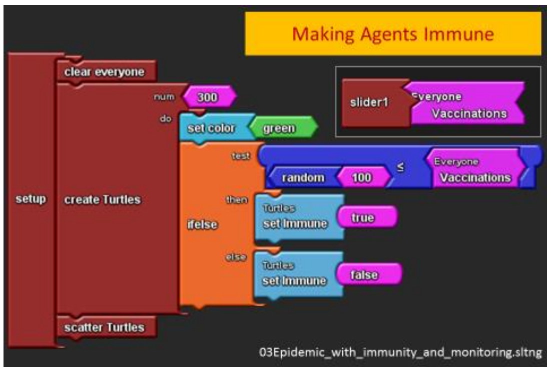
To make that easier, we could make this a variable value, and attach a slider to it. This could be set as a practical challenge. The widget is in the Setup and Run drawer. The default range is 0 to 10. Either parameter can be altered in SpaceLand. An example is included: *02Epidemic_with_recovery_and_seed_slider.sltng*

Sliders can be altered at runtime to observe the impact, but even so we have no accurate figures, just a visual sense of proportions. Two ways we could improve this might be to keep a count of infected agents, and graph the changes over time. Both are very simple to implement due to other widgets available in the Setup and Run drawer.



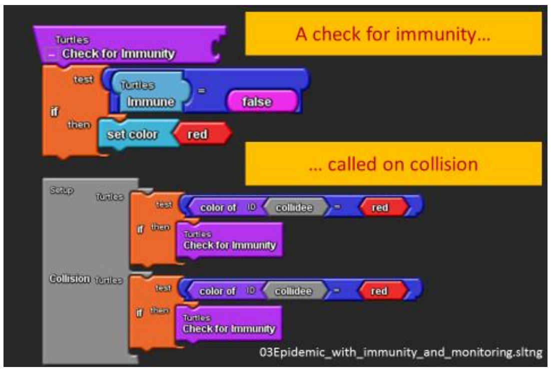
The line graph is particularly valuable in tracking how small local changes can impact on global values. The video clip on the next slide demonstrates this. As we alter the Recovery Rate during run time, we can see how the infected agents rises significantly in response to a small decrease in the Recovery, from 9/100 to 4/100. An example file is included: *02Epidemic_with_recovery_and_monitoring.sltng*. This has the Seed Slider removed to aid identification of the additional features. If participants are investigating the model in the session, they can experiment on their own models rather than show the video included in the presentation.

Making Agents Immune



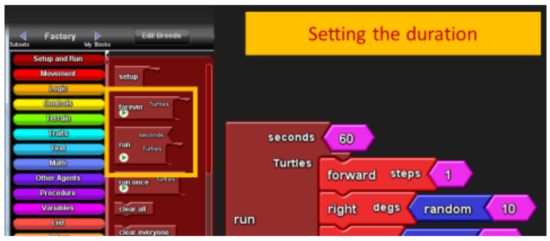
Now that we can monitor impact, let's add one further modification. We might want students to get an appreciation of the impact of a vaccination programme. To do this, we could add the facility to make some agents immune. A local Boolean variable would suffice which could be set when each agent is created.

Notice we use the same technique of assigning a random number and testing it against a threshold. In this particular case, around 1% of the population are given immunity. But we can do better than that. By creating a 'Vaccinations' variable, attached to a slider like before, we can alter our level of vaccination at setup.


A check for immunity...
Turtles Check for Immunity
if
test
Turtles Immune = false
then
set color red
... called on collision
Setup
Collision
Turtles
color of ID collidee = red
Turtles Check for Immunity
03Epidemic_with_immunity_and_monitoring.sltng

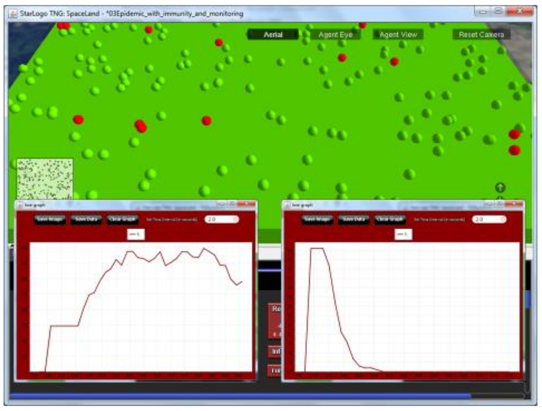
Once an agent has an Immune attribute, a procedure is needed to check it. The procedure is called when a collision occurs. If the agent collides with an infected agent, the check is called. Only if the agent is not immune, will they become infected.

There is just one more refinement before we can run our model. If we want to compare results we need to run simulations for the same length of time. Notice here we have changed the Forever block for a Fixed Time of 60 seconds. A final sample file is included, so attendees can experiment *03Epidemic_with_immunity_and_monitoring.sltng*.

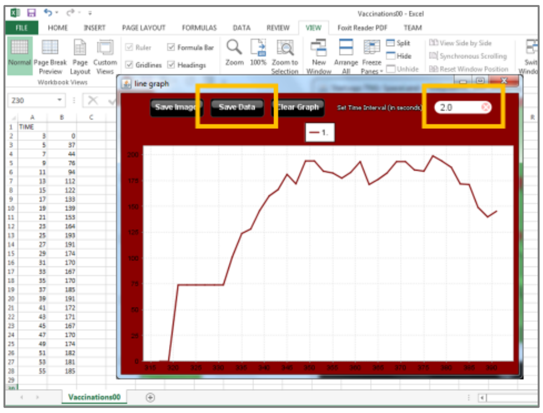


Investigating The Impact

The slide shows our simulation, ready to run using the sample file supplied. We have seeded the infection to around 10% of the population, have a recovery rate 5 and no vaccinations. We have set it to graph infections only. We can set the speed so it completes very quickly, and our result is shown. Infections peaked at around 200 – that's 2/3 of the 300 population we initially scattered.



If we now vaccinate 30% of the population and run the simulation again we can see a dramatic impact. Not only has the infection died away quickly, if you look closely at the axis, it peaked at around 60. The graphs scale automatically as they are being created. We have a clear visual representation of the potential impact of a vaccination programme … and hopefully stimulus to explore further.

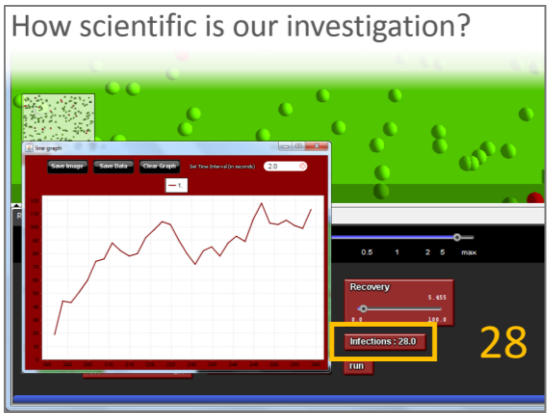


If we take a closer look at the graph widget, you'll see an option to Save the Data. This exports the data as a csv file, which can then be opened in Excel or a similar spreadsheet application. Down the left hand column are the time values in 2 second increments – a parameter that can be set when capturing the data. Although it would involve a degree of cutting and pasting, we have the basis for some simple analysis of the data captured. Children can be encouraged to think how they might plot the impact on the spread of an epidemic of various levels of vaccination take up.

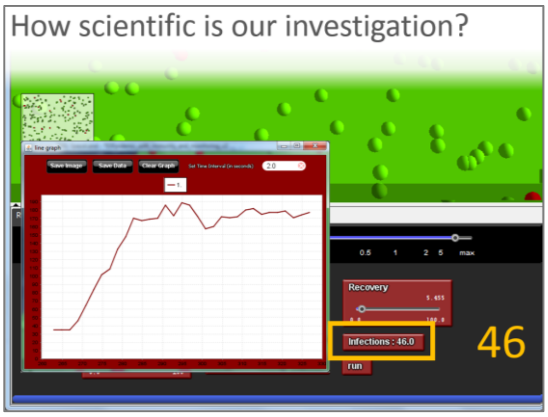
How Scientific Is The Investigation?

Let's just consider our interface again for the moment. Whilst running a simulation a few times with different vaccination rates might seem like a scientific experiment, we need children to embrace a few more ideas. The question can be a good prompt for a class discussion. Points to draw out are that our model is only as good as the assumptions it is based on.

Whilst we keep time constant, we also have 2 other variables to consider as well as the vaccinate rate. The Recovery rate can represent the 'health' of the population – a well fed population is likely to recover quicker for example. So we have scope to investigate the impact of vaccination among populations of differing health. We can also modify the Seed to reflect other social factors such as overcrowding.



A final crucial element is for children to reflect on the impact of random behaviour. Although we can set a Seed variable, this only provides an upper boundary for generating a random number for each agent. As a result, the number of initial infections can vary significantly as can the subsequent behaviour of the model. Setting up the same experiment can result in wildly different starting rates of infection. And scattering the population results in a random distribution too. Here 28 initial infections lead to the first graph. Different initial infections generated different outcomes.



Recognising these factors should lead children to realise they need to carry out many runs of the same starting setup to derive an 'average' general view. Scope here for group planning perhaps, with different groups initiating multiple runs on the same data. Each group could have different parameters. Once collated, spreadsheets from each group can be swapped and combined.

Before moving on, it's worth considering ways to improve the model. Children will come up with all sorts of ideas. A key point to draw out is that ideas models are abstractions. Extra detail isn't always helpful so encourage pupils to focus on key ideas and simple ways to represent them.

An obvious one is to introduce doctors who heal people. They can be implemented as blue agents, who heal infected agents when then collide. Quarantine or hospitalisation might involve stopping the agent moving. Death might be another 'easy to implement' development, and birth, so we begin to explore the idea of a population over time. Passing on an infection at birth could also be considered.

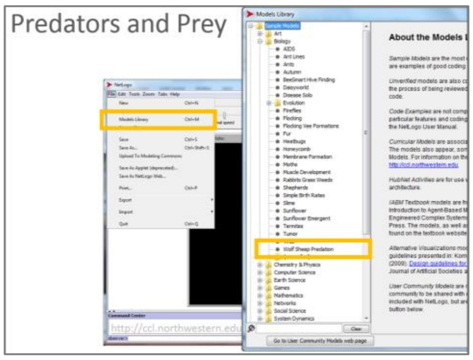
Allowing children to develop their own improvements is often very rewarding. This is constructionist pedagogy coming to the fore. If you wish to develop a more structured investigation the Scheller Teacher Education Program hosts a series of curriculum units including one on epidemics. This takes an investigation further, looking at the relative costs of different health interventions. All materials are freely downloadable from their site: goo.gl/VCLpu1.

Predators and Prey

For our second exploration into ways to analyse data we'll look at a model of a predator / prey environment. We'll use a different modelling tool this time – NetLogo. Another free download, it is widely used in academia: goo.gl/bm4CJL.

It follows the same principles as StarLogoTNG; simple local rules applied to many agents from which the global behaviour emerges. Developed by Uri Wilensky, co-author with Mitch Resnick of the paper outlining kinaesthetic activities earlier, he is another product of the MIT Media Lab, influenced by the educational ideas of Seymour Papert. So NetLogo is another Logo descendant, and a text based modelling environment. It is available for free download from the Centre for Connected Learning, which Wilensky directs. It is based at Northwestern University on the edge of Lake Michigan, Illinois.

Whilst we won't have time to delve into NetLogo in any great depth, if modelling grabs you, this environment offers a natural progression from StarLogo. Wilensky has written a comprehensive, if pricey book and the software is well supported and documented. Our friends from Project GUTS at SantaFe Institute, through Computer Science 4 All have provided a structured online introductory course. This exercise is taken from that course.



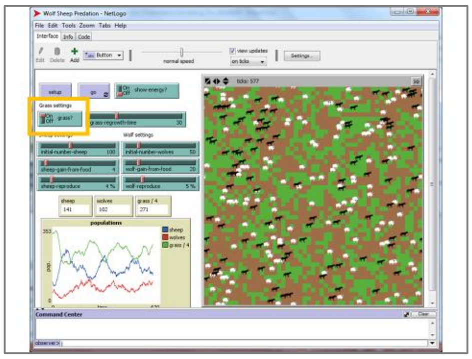
One of NetLogo's great strengths is the vast number of models in the Model Library. This gives students plenty of opportunity to explore models, and engage in text based coding by making simple modifications. Select the model library and you will see the many folders, each often containing many pre-built models. Expand the Sample models folder and locate the Biology folder. Expand that and begin to appreciate the amount of resources available. We will be working with the Wolf Sheep Predation model. If there are problems accessing the file from the model library it is included in the resources.

As the introductory notes say, this is a model to explore the stability of a simple predator / prey model. We're familiar now with many of the elements for a decentralised model such as this.

There are 3 elements:

The **observer (or user)** instantiates the world through the Setup and Go buttons. They can also change variable values via the sliders and settings to establish what is shown. Let's setup our world now, leaving all settings in their default state.

Our **agents** will follow simple rules as before. How many different 'agents' have we in this model? (Two, sheep and wolves.) Let's run the model to see what happens. Everyone will get different results but you'll notice we have similar graphing capabilities as we had in StarLogo. Most model runs will probably lead to extinction of one or the other species. In the example shown, wolves have died out. What happens if one species becomes extinct? (If wolves die out, sheep population continues to expand. If sheep die down, wolves expand until they suddenly collapse). Either way, such a system is generally known as unstable.

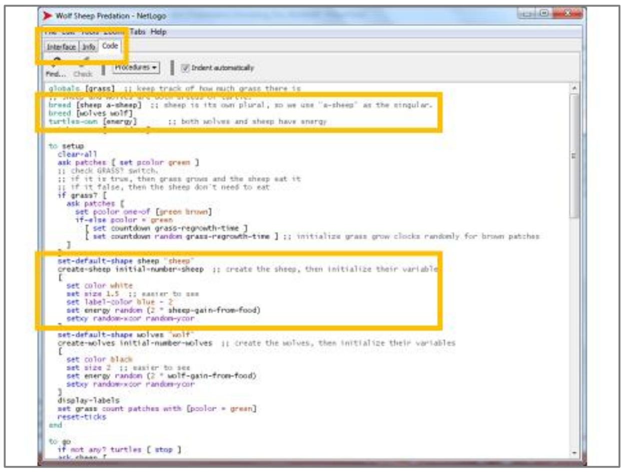


Let's make one alteration. The third element in a model is the agent's **environment** which is made up of patches. Switch the Grass on, which introduces many more variants into the behaviour of the model. Run the model and observe what happens. Notice now how we have a stable eco-system. As well as graphing sheep and wolves, we can monitor grass growth as well. Notice the way the populations of all go up and down, and notice too how they relate to each other. This is a familiar pattern in population dynamics which we will investigate later. First, let's understand how this model works.

How It Works

Notice the tabs along the top of the screen. Select the info tab, rather than the interface. Each model in the model library includes a comprehensive explanation. Scroll down and read the ‘things to notice’ and ‘things to try’. These are excellent resources for children to explore.

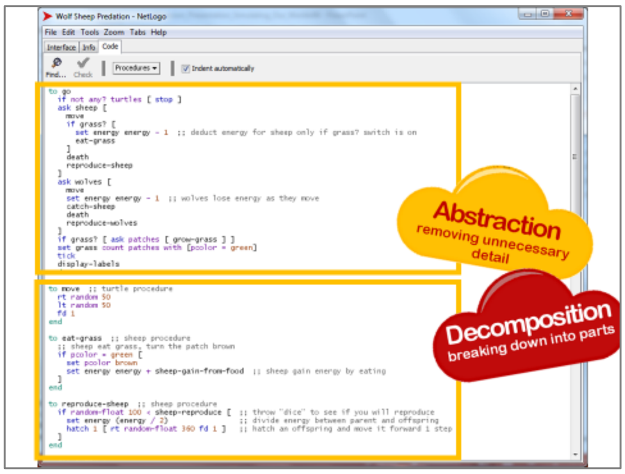
Finally, take a look at the code tab. It should be easy to identify the similarities with the sort of block based code we have used in our models. We have two breeds, sheep and wolves. These are subsets of a turtle object, and both have a variable energy. When we create our sheep, as part of the setup procedure, the energy value is assigned. Can anyone explain how it is calculated?


Wolf Sheep Predation - NetLogo
Tabs Help
Interface Info Code
Find... Check Procedures Indent automatically

```
globals [grass]  ;; keep track of how much grass there is
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

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 variable
  [
    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 the 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

to go
  if not any? turtles [ stop ]
```

Scrolling below the Setup procedure, you see the main Go procedure. Again, you could use this as an exercise in reading code as a group. Note how well commented and easy it is for children to understand, particularly if they have had experience with block based code.

Notice also, how ‘uncluttered’ the main procedure is – another example of procedural abstraction. The main ‘Run’ loop calls a series of procedures which are defined below it. Here the detail is spelt out, each action decomposing into a series of familiar steps. Again, ideal for assessing students ability to understand and interpret code. A code listing is included in the resources if you wish to develop further code reading exercises.


Wolf Sheep Predation - NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Find... Check Procedures Indent automatically
Abstraction
removing unnecessary detail
Decomposition
breaking down into parts

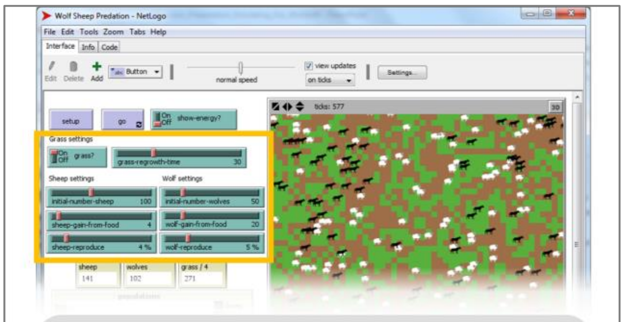
```
to go
  if not any? turtles [ stop ]
  ask sheep [
    move
    if grass? [
      set energy energy - 1  ;; deduct energy for sheep only if grass? switch is on
      eat-grass
    ]
    death
    reproduce-sheep
  ]
  ask wolves [
    move
    set energy energy - 1  ;; wolves lose energy as they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

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 an offspring and move it forward 1 step
  ]
end
```

Remember our pedagogy model? We’re still exploring models here, but engaging with a text based equivalent model deepens students understanding not only of how a model works, but how to analyse the simulation too.

Parameter Sweeping

NetLogo offers options for a more thorough exploration than those available in StarLogo. How many parameters interact in this model? There are 7 factors (or parameters) that can be altered to explore the behaviour of the simulation (3 for the sheep and wolves each plus the grass regrowth time). Like before, with the Bridge Design model, we might want to keep some factors constant whilst we sweep through a series of values for another factor.

So to analyse our model we need to perform a parameter sweep. But for each group of settings, because this is a stochastic model involving randomness, we also need to run the model multiple times. That is a lot of tests!

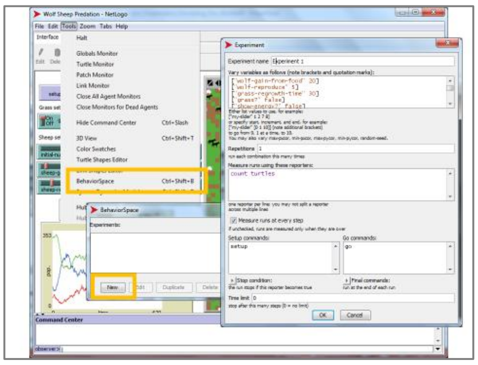


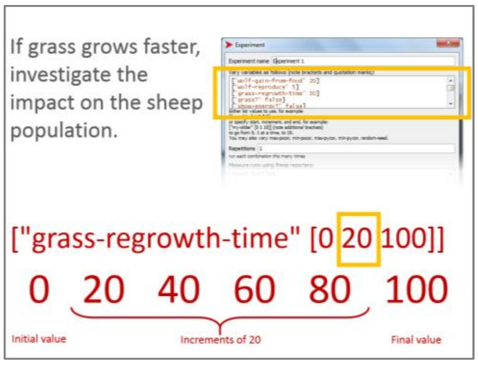
Grass settings
grass? On
grass-regrowth-time 30
Sheep settings
initial-number-sheep 100
sheep-gain-from-food 4
sheep-reproduce 4 %
Wolf settings
initial-number-wolves 50
wolf-gain-from-food 20
wolf-reproduce 5 %
sheep 141
wolves 102
grass / 4 271

Perform a parameter sweep
For each set of parameters:
Run the model multiple times

Like we suggested with StarLogo, we could approach this as a group task, but NetLogo has a wonderful tool to help us. It’s called behaviour space, and is accessed from the Tools menu. Instructions for setting up behaviour space and a sample spreadsheet output are also included in the resources. The following slides provide a walk through.

Select the new option and the dialogue window opens. Give your experiment a name, then look carefully at the variables list. Let's investigate the impact of faster grass growth on our sheep population. To do that we will first need to turn the grass on, so set "grass?" to true.



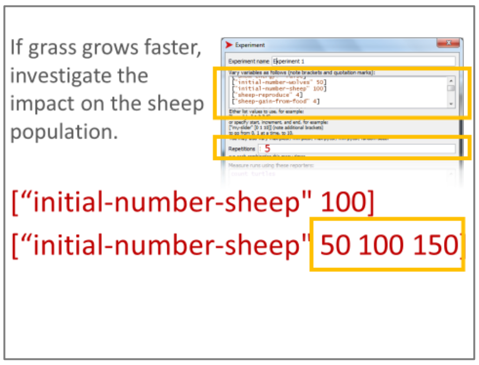

If grass grows faster, investigate the impact on the sheep population.

["grass-regrowth-time" [0 20 100]]

0 20 40 60 80 100

We might want to run the model at various rates of grass growth. Instead of a fixed rate of 30, let's try various rates. Replace 30 with the expression shown [0 20 100]. Note the extra brackets and three numbers separated by spaces. These denote the lower and upper bounds and the increment.

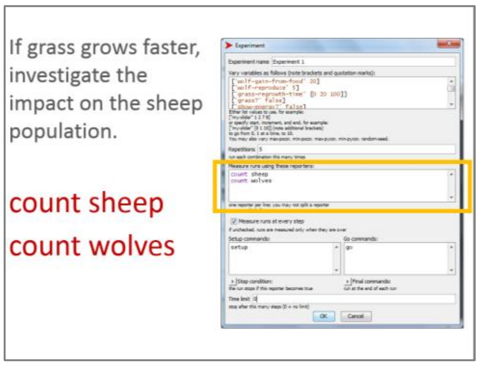
Scroll down the variable list and locate "initial-number-sheep". This is currently set to 100. Rather than a fixed population of sheep, we might want to investigate the impact of the grass change on several populations. Here we have set it to try starting populations of 50, 100 and 150. Note here that discrete values are simply separated by spaces, without any brackets.


If grass grows faster, investigate the impact on the sheep population.

["initial-number-sheep" 100]

["initial-number-sheep" 50 100 150]

So we now have 3 initial sheep populations, each being run in a simulation with 6 different rates of grass growth. There is one other factor still to consider. Because our model involves random elements we need to run each simulation several times. Let's change the 'Repetitions' to 5.

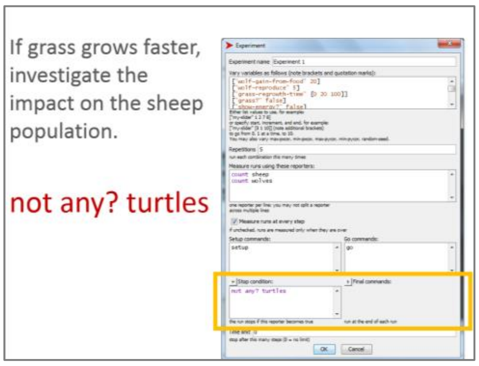

If grass grows faster, investigate the impact on the sheep population.

count sheep

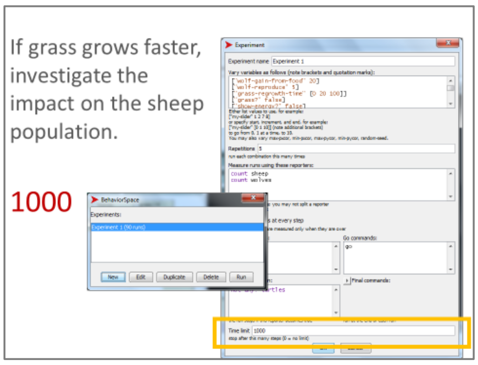
count wolves

With the parameters for the simulations set, we can consider what we want reported. The default is turtles, but we want the count of the sheep breed. It is probably worthwhile to have a count of the wolves too.

The default setting for behaviour space is to measure runs at every step, rather than just at the end. We'll leave that so you can see the effect, but if you had long model runs or to simplify the volume of data you may wish to just report the values at the end. Below this we can specify the procedures to call for setup, and run. In our case, the model uses just 'setup' and 'go'. The 'Stop condition' allows us to run models until certain conditions are met. In our case, we want to run each simulation for a certain amount of time, but if our breeds become extinct before that, there is little point continuing. So to demonstrate this capability, let us put in the condition "not any? turtles". There is another option to specify procedures to run once it has finished (Final commands). We have no need for that.


If grass grows faster, investigate the impact on the sheep population.

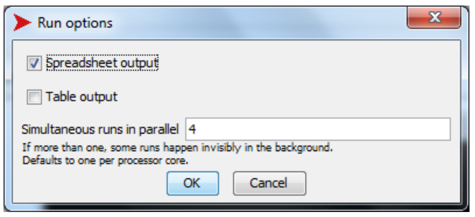
not any? turtles


If grass grows faster, investigate the impact on the sheep population.

1000

Our final task is to specify how long we want the simulation to run for. We'll set it for 1000. If you wish your simulation to run until a stop condition is met, this value can be left at 0. However, even if you wish your simulation to be controlled by a condition, rather than a set time, if there is a chance it may never meet the condition, it is wise to specify a final time limit. When we select OK, our Behaviour Space window lists the experiment(s) we have configured and the number of runs required.

When we hit Run, we are given the choice of output formats. Selecting Spreadsheet allows the data to be saved as a csv file. For efficiency and speed, several runs can be made in parallel – the default number being determined by the number of cores in the processor. Finally decide where you wish to save the file, and add a .csv extension. Behaviour Space will now run all the simulations automatically.

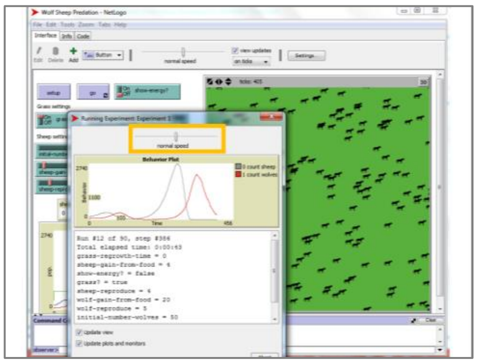


[✓] Spreadsheet output

[ ] Table output

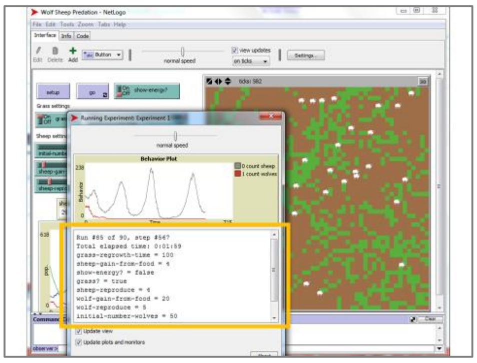
Simultaneous runs in parallel 4

If more than one, some runs happen invisibly in the background.
Defaults to one per processor core.


Wolf Sheep Predation - NetLogo
Running Experiment: Experiment 1
normal speed
Behavior Plot
0 count sheep
1 count wolves
Time
Update view
Update plots and monitors

```
Run #12 of 90, step #386
Total elapsed time: 0:00:43
grass-regrowth-time = 0
sheep-gain-from-food = 4
show-energy? = false
grass? = true
sheep-reproduce = 4
wolf-gain-from-food = 20
wolf-reproduce = 5
initial-number-wolves = 50
```

Note that you can increase the speed, as well as observe one of the runs currently taking place. Observing the runs can stimulate discussion. With a grass-regrowth-time of 0, this run has resulted in sheep dying out and the population of wolves is falling dramatically. Why might that be the case? In a second example run, we seem to have a more stable eco-system. The slide highlights the settings – a grass-regrowth-time of 20.


Wolf Sheep Predation - NetLogo
Running Experiment: Experiment 1
normal speed
Behavior Plot
0 count sheep
1 count wolves
Time
Update view
Update plots and monitors

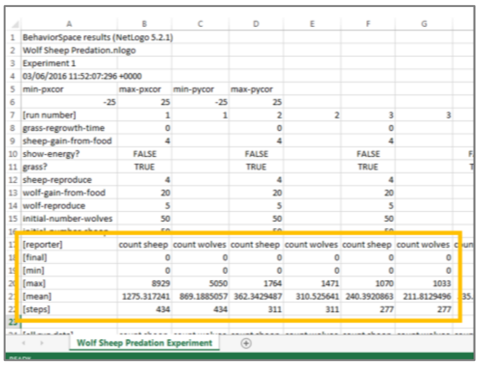
```
Run #85 of 90, step #567
Total elapsed time: 0:01:59
grass-regrowth-time = 100
sheep-gain-from-food = 4
show-energy? = false
grass? = true
sheep-reproduce = 4
wolf-gain-from-food = 20
wolf-reproduce = 5
initial-number-wolves = 50
```

With a longer regrowth time – 100 (this is run 85 of 90 shown right) we see a very different pattern. This time the wolves have died out and the sheep population is oscillating. So just the facility to run a series of simulations automatically can help stimulate further questions. Hopefully students will be motivated to analyse the spreadsheet data more thoroughly.

The data captured in the csv file, opened in Excel is displayed. It is included in the resources if the group wish to study it. The first thing you'll notice about the spreadsheet is that it is huge! Making sense of a large data set is part of the challenge for students. Let's spend a moment unpicking it.

The first six rows give details of the experiment: the model used, the date and the size of the world. That's what the min and max x and y co-ordinates indicate. Rows 7 to 16 list the starting parameters for each run. Check understanding by asking what the grass regrowth time is for run 3? What is the initial number of wolves and sheep for the first runs?

Rows 17 to 22report the final results from each run for the things we specified – in this case number of sheep and wolves. It also reports the minimum and maximum values achieved in each run and the mean. Finally the number of steps the simulation took is recorded. Question – we set our simulation to run for 1000 steps, why has this not happened in the runs displayed? Finally, point out that there will be several columns recording data for each run. Although the starting parameters are only stated once, there will be a column for each element we specified reporting on. If you scroll down, the count for each animal is recorded for every step of the simulation.


Wolf Sheep Predation Experiment

	A	B	C	D	E	F	G
1	BehaviorSpace results (NetLogo 5.2.1)						
2	Wolf Sheep Predation.nlogo						
3	Experiment 1						
4	03/06/2016 11:52:07:296 +0000						
5	min-pxcor	max-pxcor	min-pycor	max-pycor			
6	-25	25	-25	25			
7	[run number]	1	1	2	2	3	3
8	grass-regrowth-time	0		0		0	
9	sheep-gain-from-food	4		4		4	
10	show-energy?	FALSE		FALSE		FALSE	
11	grass?	TRUE		TRUE		TRUE	
12	sheep-reproduce	4		4		4	
13	wolf-gain-from-food	20		20		20	
14	wolf-reproduce	5		5		5	
15	initial-number-wolves	50		50		50	
17	[reporter]	count sheep	count wolves	count sheep	count wolves	count sheep	count wolves
18	[final]	0	0	0	0	0	0
19	[min]	0	0	0	0	0	0
20	[max]	8929	5050	1764	1471	1070	1033
21	[mean]	1275.317241	869.1885057	362.3429487	310.525641	240.3920863	211.8129496
22	[steps]	434	434	311	311	277	277

We could go on, but the point is to highlight the possibilities this offers for students to engage in systematic analysis. Once the data is in Excel, there are many more facilities for charting the data. Running simulations in NetLogo, analysing the data in Excel and creating a summary report, whether written, oral or a multimedia report offers lots of scope for satisfying other elements in the National Curriculum. More importantly, given the wide variety of models available, there are lots of possibilities for tying this work to that done in other departments, principally science.

Use
Explore
Build

Back to our pedagogical framework. We've spent a lot of time exploring 'computational abstractions' to get a good insight into what makes a model work. We've introduced several models that simulate real world systems at an 'ideas' level. Models such as these are ripe for evaluation and improvement. But we should now be in a position to build a model almost from first principles. That is the focus of the final session.