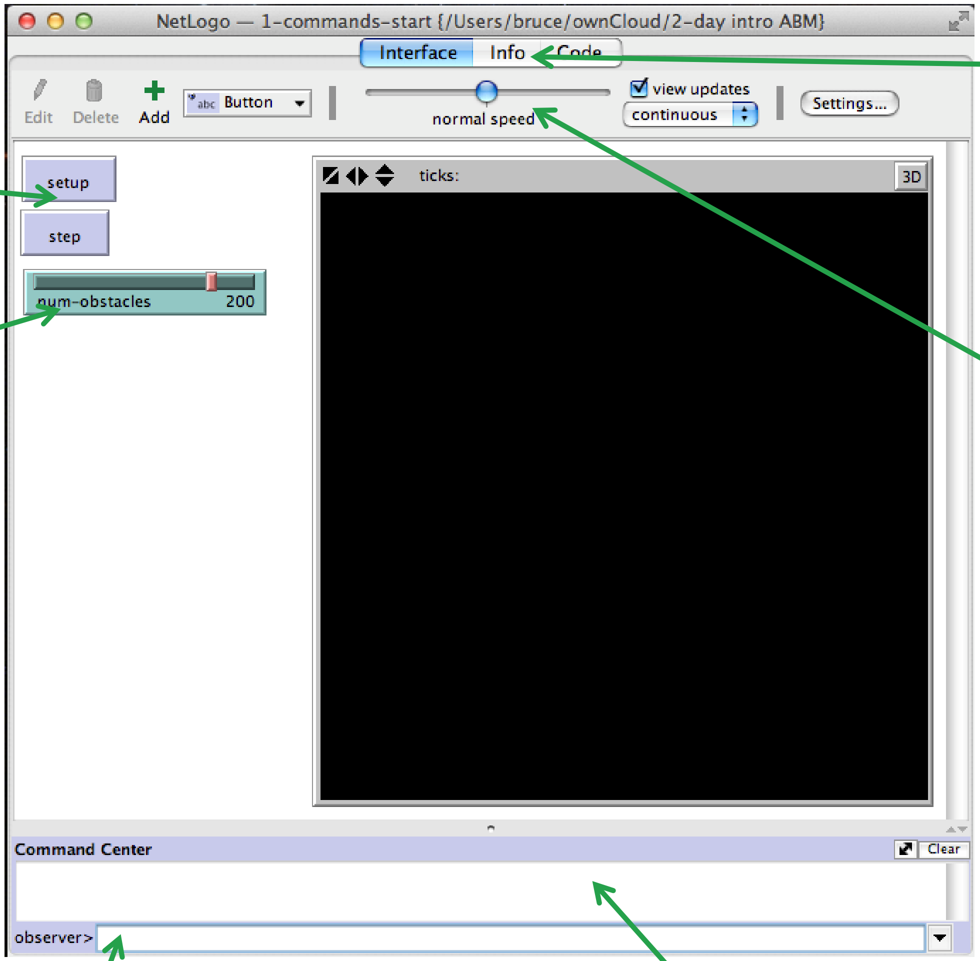
NetLogo

- Easy to use
- Programming language directed to interface
- “agents” are called “turtles” in documentation
- If you want to use complex (BDI) agents use Repast (Java based)

Manchester Metropolitan University

NetLogo – Interface Panel



Command Buttons

Parameter Slider

Panel Selection (looks slightly different on Windows and Macs)

Speed Control

Typed Direct Commands

Text Output

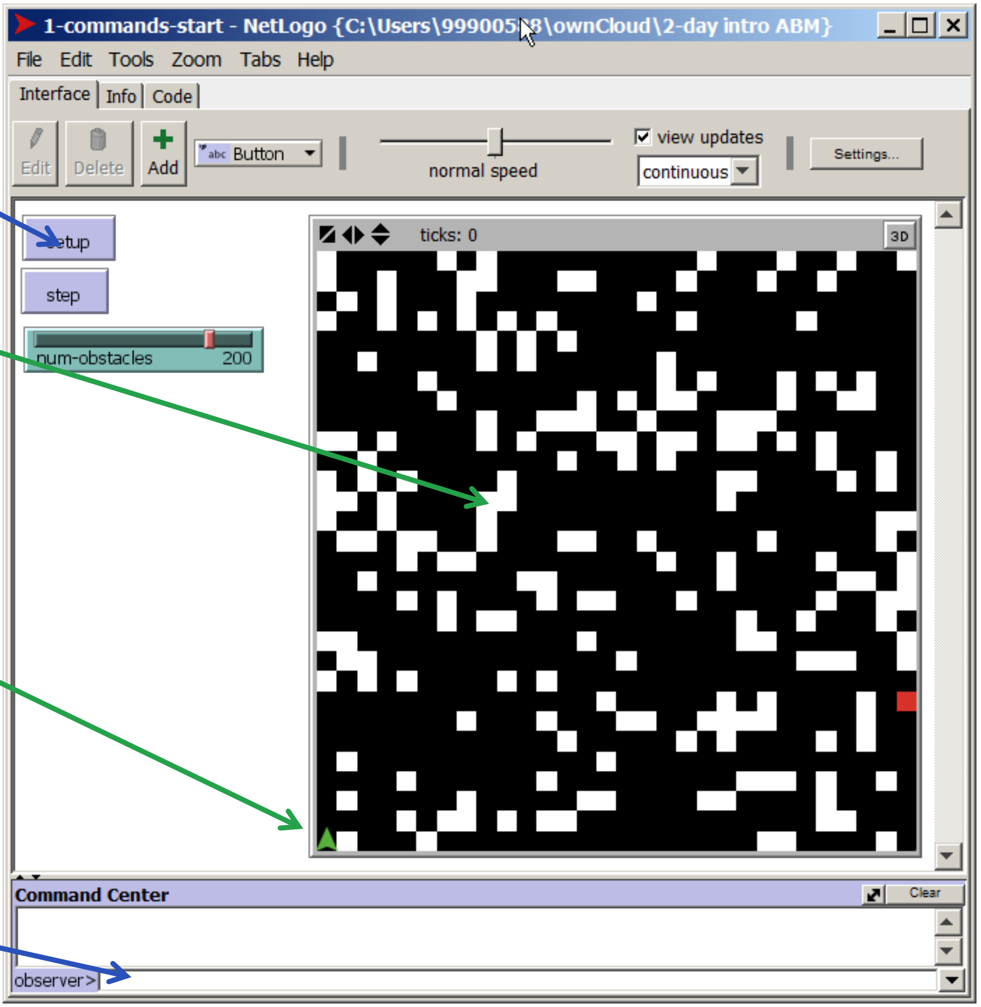
Typing in Commands

Press "setup" to initialise world

World with different colour patches

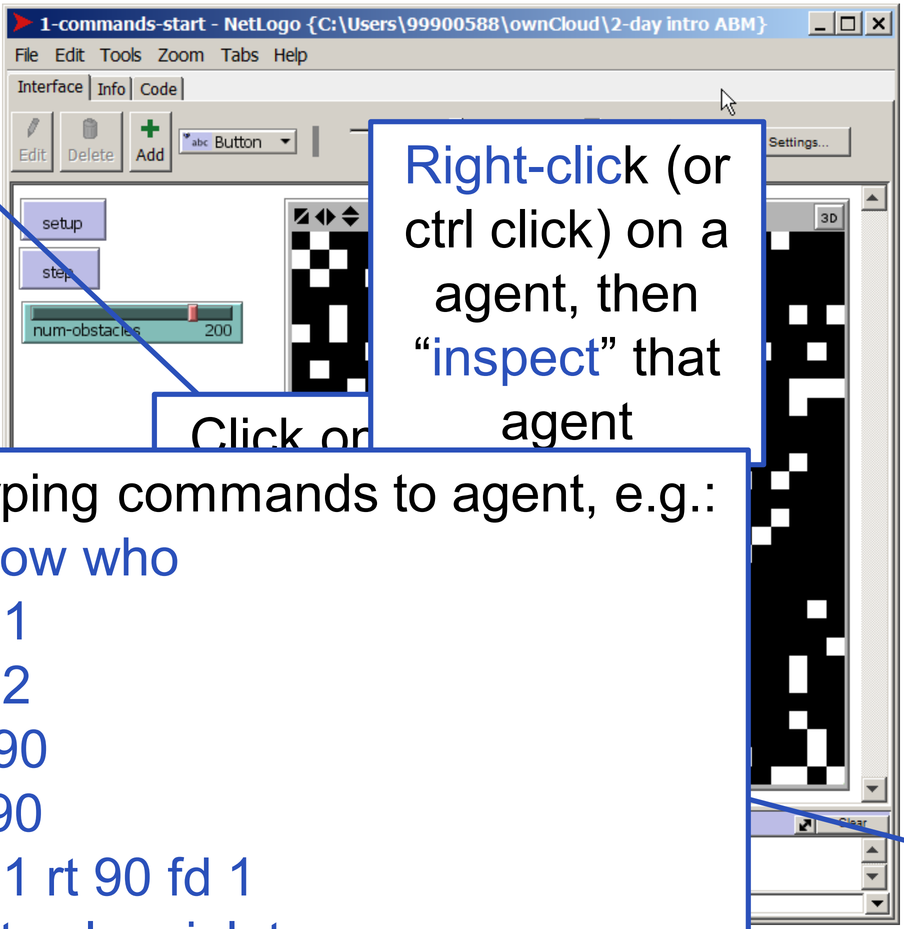
An agent!

Type commands in here as follows…



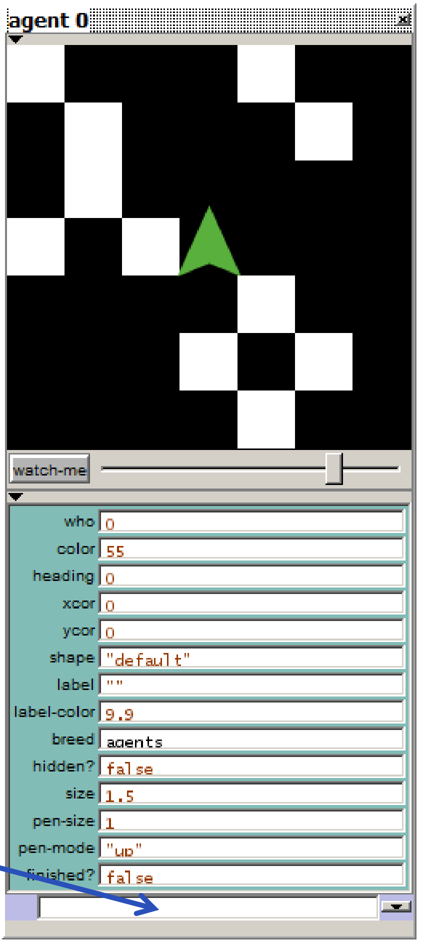
Inspecting Patches and Agents

Right-click (or ctrl click) on a patch, then "inspect" that patch

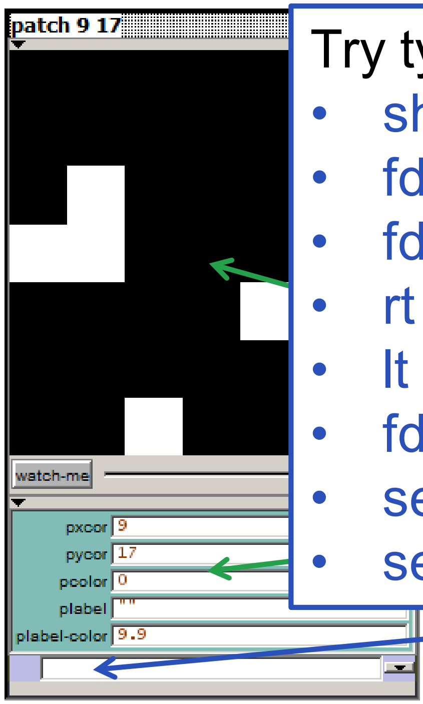


Click or

Right-click (or ctrl click) on a agent, then "inspect" that agent



who	0
color	55
heading	0
xcor	0
ycor	0
shape	"default"
label	""
label-color	9.9
breed	agents
hidden?	false
size	1.5
pen-size	1
pen-mode	"up"
finished?	false



pxcor	9
pycor	17
pcolor	0
plabel	""
plabel-color	9.9

Try typing commands to agent, e.g.:

- show who
- fd 1
- fd 2
- rt 90
- lt 90
- fd 1 rt 90 fd 1
- set color violet
- set size 4

Type commands to patch here, e.g. set pcolor red

Manchester Metropolitan University

Some important ideas

- The whole world, the turtles, the patches (and later the links) are "agents"
- That is, they:
  - have their own properties
  - can be given commands
  - can detect things about the world around them, other agents etc.
- But these are all ultimately controlled from the world (from the view of the observer)
- It is the world that is given the list of instructions as to the simulation, which then sends commands to patches, agents (and links) using the "ask" command

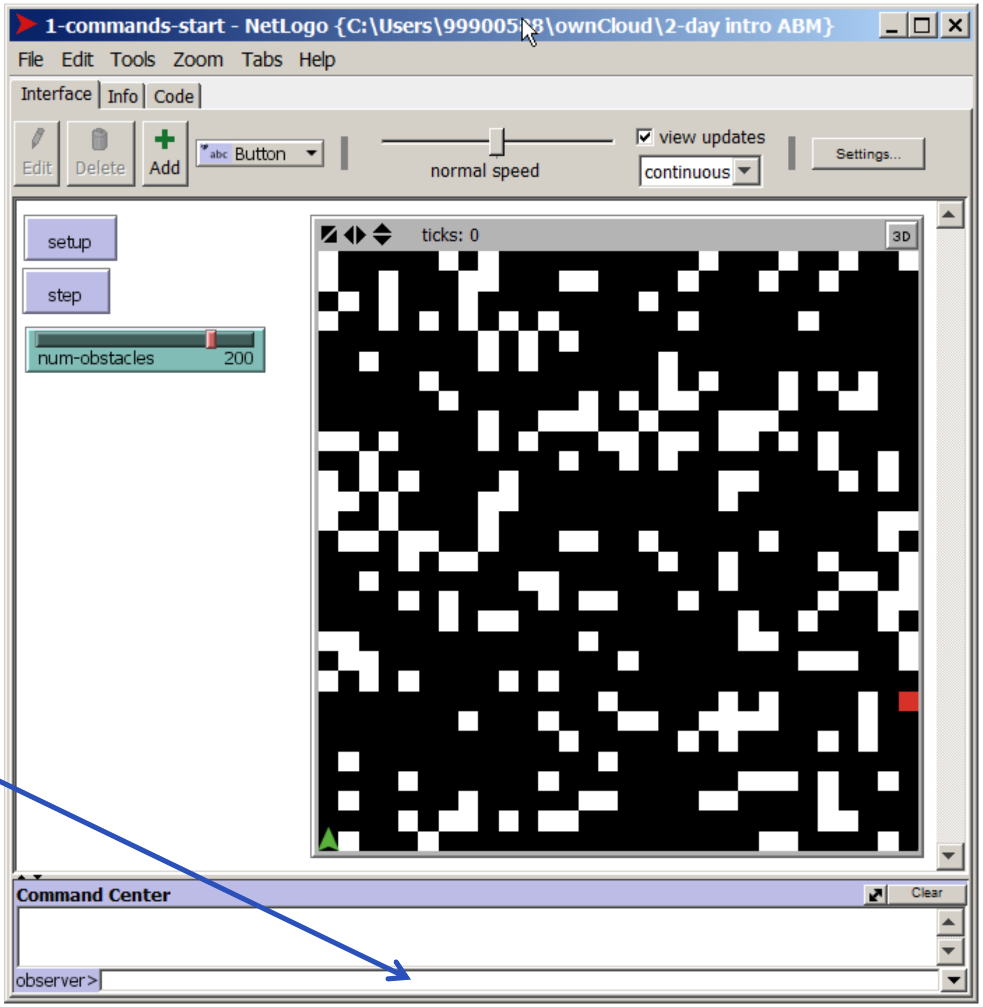
Using "ask"

Try typing commands to agents via the world, e.g.:

- ask inds [fd 1]
- ask inds [set color grey]
- ask inds [set shape "person"]
- ask inds [fd 1 rt 90 fd 1]
- ask inds [show patch-here]
- etc.

Can also ask patches:

- ask patches [show self]
- ask patches [set pcolor black]
- ask patch 0 0 [show agents-here]



Manchester Metropolitan University

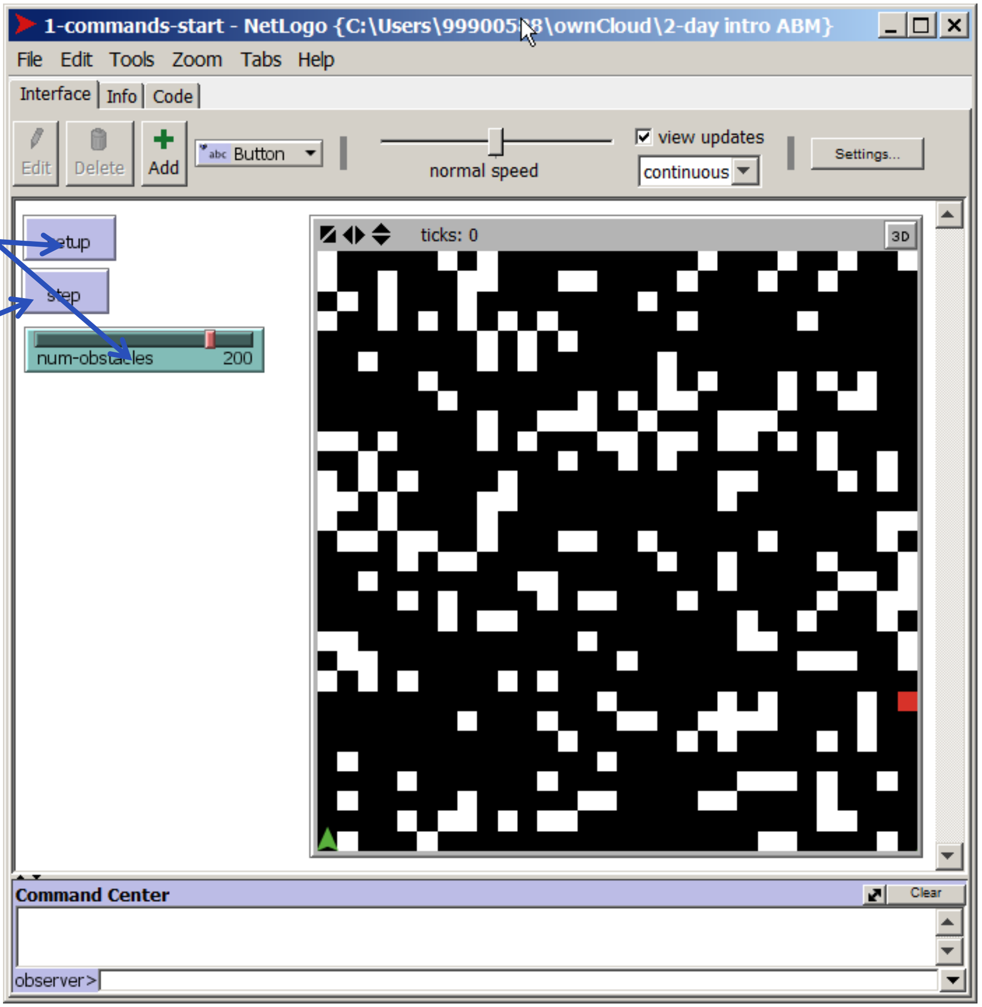
Running a simulation (the hard way!)

1. Move the slider to change parameter

2. Press "setup" to initialise world

3. Press "step" to make the program run one time step

4. Press "step" lots of times!!



- Each time "step" is pressed the procedure called "go" is caused to run – this is a list of commands, a program.
- We will now look at this.

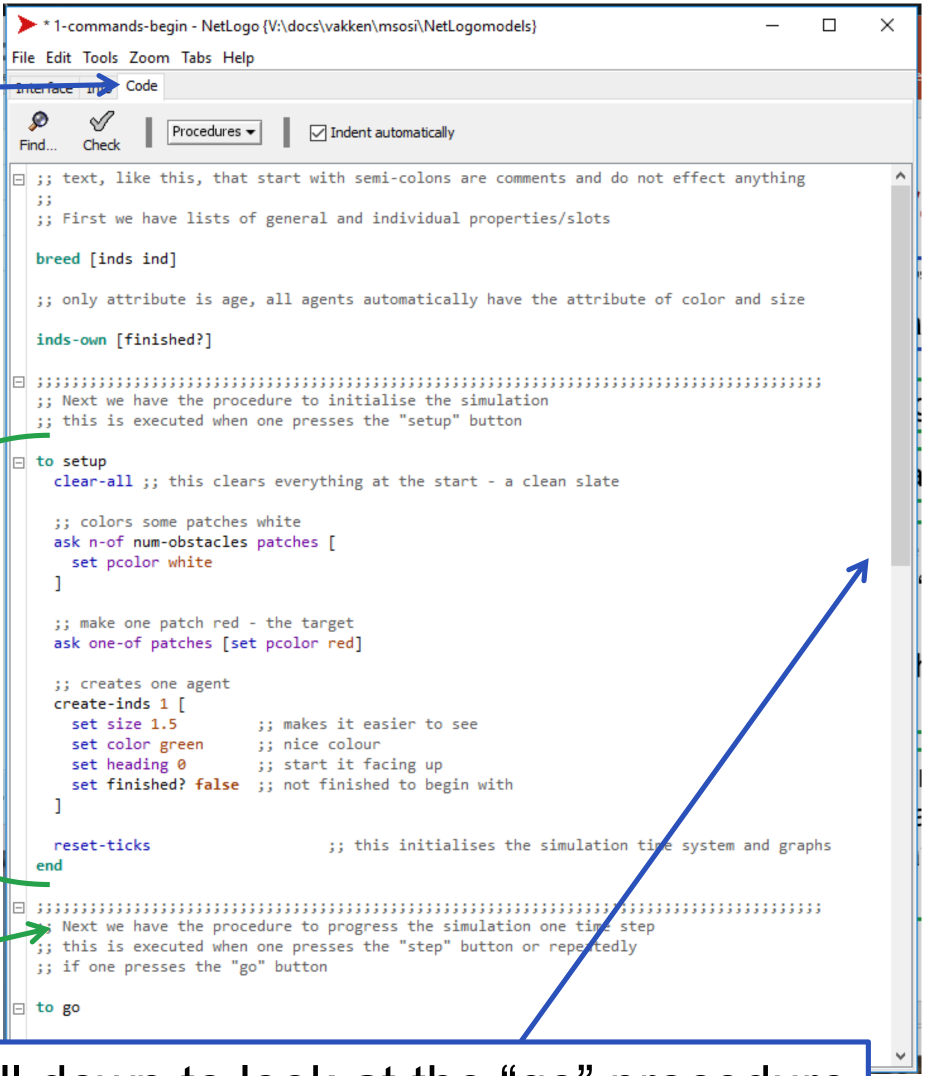
The Program Code

Click on the “Code” tab to see the program

This text is the program

This chunk of code (from “to” to “end”) is the “setup” procedure – what happens when you press the setup button

Text after a semi-colon “;” are comments and have no effect


* 1-commands-begin - NetLogo {V:\docs\vakken\msosi\NetLogomodels}
File Edit Tools Zoom Tabs Help
Interface Info Code
Find...
Check
Procedures
Indent automatically

```
;; text, like this, that start with semi-colons are comments and do not effect anything
;;
;; First we have lists of general and individual properties/slots

breed [inds ind]

;; only attribute is age, all agents automatically have the attribute of color and size

inds-own [finished?]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Next we have the procedure to initialise the simulation
;; this is executed when one presses the "setup" button

to setup
  clear-all ;; this clears everything at the start - a clean slate

  ;; colors some patches white
  ask n-of num-obstacles patches [
    set pcolor white
  ]

  ;; make one patch red - the target
  ask one-of patches [set pcolor red]

  ;; creates one agent
  create-inds 1 [
    set size 1.5        ;; makes it easier to see
    set color green     ;; nice colour
    set heading 0       ;; start it facing up
    set finished? false ;; not finished to begin with
  ]

  reset-ticks                   ;; this initialises the simulation time system and graphs
end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Next we have the procedure to progress the simulation one time step
;; this is executed when one presses the "step" button or repeatedly
;; if one presses the "go" button

to go
```

Scroll down to look at the “go” procedure – this is what the “step” button does

Manchester Metropolitan University

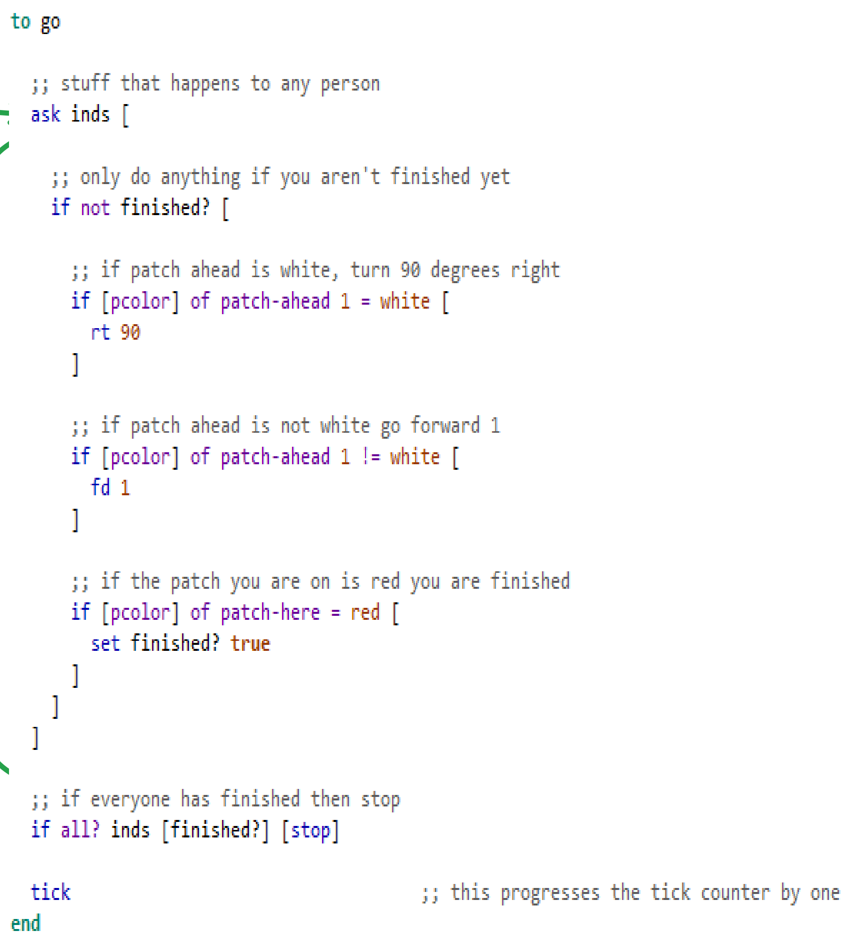
Parts of the Code

“ask inds” means to ask (all) agents to do some code, one after the other

Everything between “to” and “end” defines what “go” *means*

What it is asking them all to

All the square brackets inside each other can be confusing, if you double-click ***just outside*** a bracket, it shows what is in side between it and the matching bracket



```
to go

  ;; stuff that happens to any person
  ask inds [

    ;; only do anything if you aren't finished yet
    if not finished? [

      ;; if patch ahead is white, turn 90 degrees right
      if [pcolor] of patch-ahead 1 = white [
        rt 90
      ]

      ;; if patch ahead is not white go forward 1
      if [pcolor] of patch-ahead 1 != white [
        fd 1
      ]

      ;; if the patch you are on is red you are finished
      if [pcolor] of patch-here = red [
        set finished? true
      ]
    ]
  ]

  ;; if everyone has finished then stop
  if all? inds [finished?] [stop]

  tick                                    ;; this progresses the tick counter by one
end
```

To change the program…

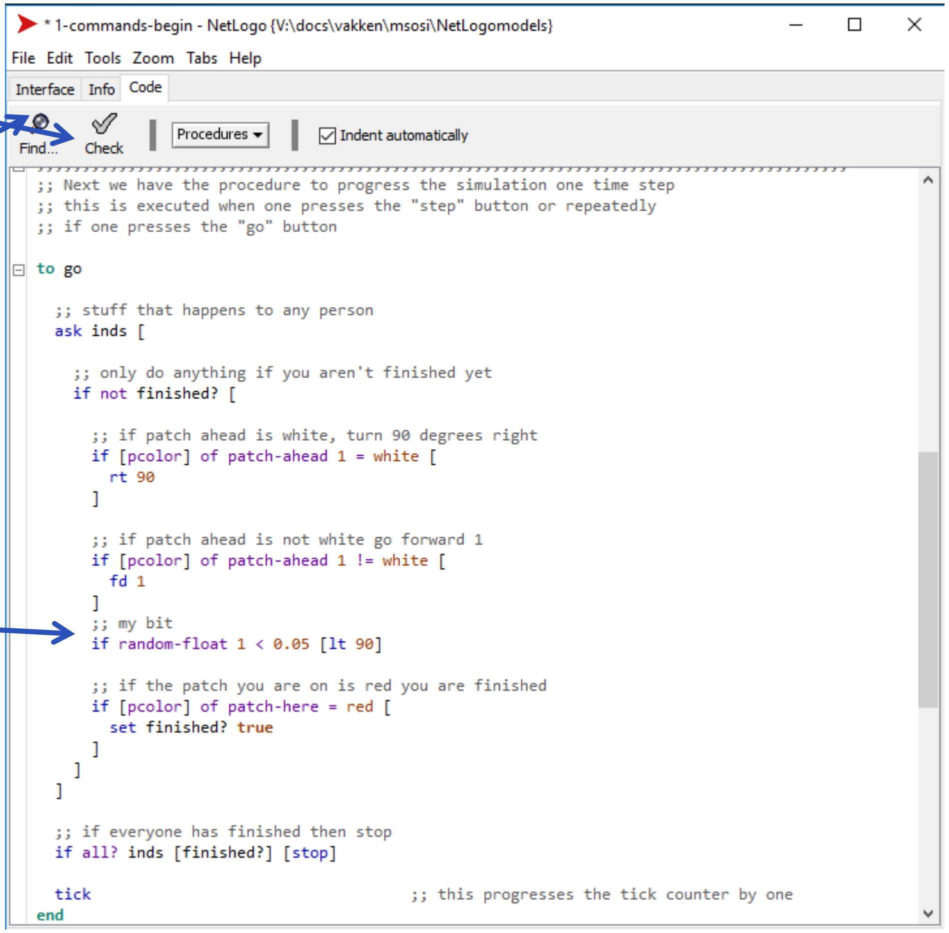
ee if
ning

If all is well you can then click on "**Interface**" to go back and try the effect of your change when running the code (pressing the "**step**" button)

Click within the text and type!

type the following:

;; my bit!
if random-float 1 < 0.05 [lt 90]


* 1-commands-begin - NetLogo {V:\docs\vakken\msosi\NetLogomodels}
File Edit Tools Zoom Tabs Help
Interface Info Code
Find... Check Procedures Indent automatically

```
;; Next we have the procedure to progress the simulation one time step
;; this is executed when one presses the "step" button or repeatedly
;; if one presses the "go" button

to go

  ;; stuff that happens to any person
  ask inds [

    ;; only do anything if you aren't finished yet
    if not finished? [

      ;; if patch ahead is white, turn 90 degrees right
      if [pcolor] of patch-ahead 1 = white [
        rt 90
      ]

      ;; if patch ahead is not white go forward 1
      if [pcolor] of patch-ahead 1 != white [
        fd 1
      ]
      ;; my bit
      if random-float 1 < 0.05 [lt 90]

      ;; if the patch you are on is red you are finished
      if [pcolor] of patch-here = red [
        set finished? true
      ]
    ]
  ]

  ;; if everyone has finished then stop
  if all? inds [finished?] [stop]

  tick                            ;; this progresses the tick counter by one
end
```

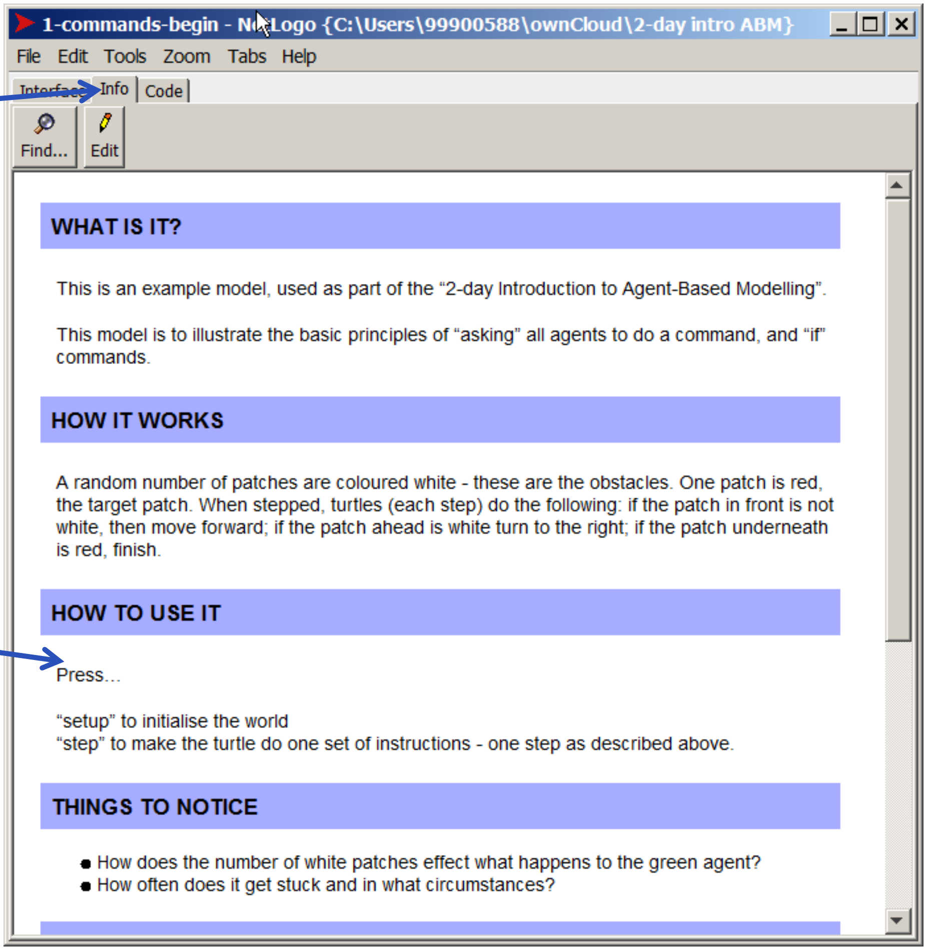
Manchester Metropolitan University

The Information Tab

Click on the "**Info**" tab to see a description of the model (or whatever the programmer has written, if anything!)

Read it, scrolling down

Here are some suggestions of bits of code to add and things to try (***in a bit***!)



1-commands-begin - NetLogo {C:\Users\99900588\ownCloud\2-day intro ABM}
File Edit Tools Zoom Tabs Help
Interface Info Code
Find... Edit

WHAT IS IT?

This is an example model, used as part of the "2-day Introduction to Agent-Based Modelling".

This model is to illustrate the basic principles of "asking" all agents to do a command, and "if" commands.

HOW IT WORKS

A random number of patches are coloured white - these are the obstacles. One patch is red, the target patch. When stepped, turtles (each step) do the following: if the patch in front is not white, then move forward; if the patch ahead is white turn to the right; if the patch underneath is red, finish.

HOW TO USE IT

Press...

"setup" to initialise the world
"step" to make the turtle do one set of instructions - one step as described above.

THINGS TO NOTICE

- How does the number of white patches effect what happens to the green agent?
- How often does it get stuck and in what circumstances?

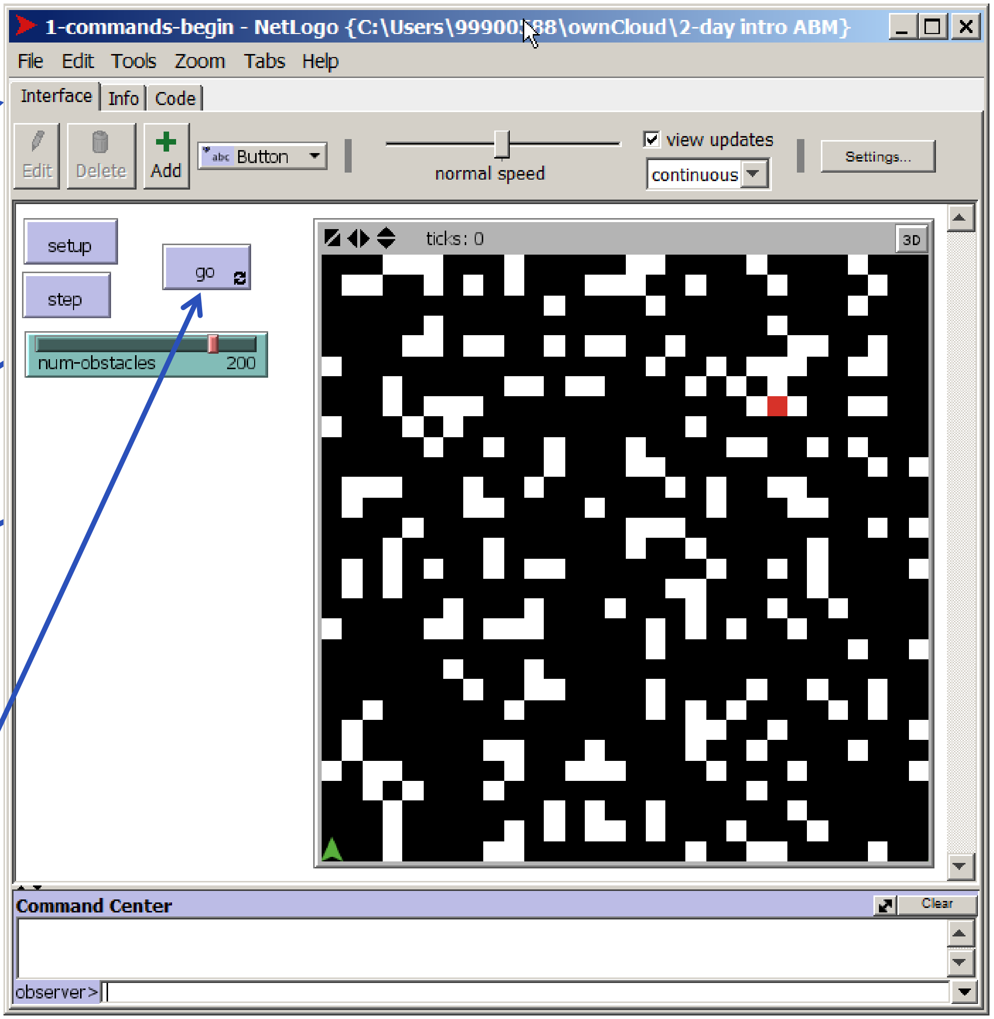
Adding a button and running the code (the fast way!)

Click on the "**Interface**" tab to get back to the main view

Right-Click some empty space and choose "**button**"

Type the text "go" here and then check (to on) the "forever" switch then "**OK**"

Now when you press the "go" button it will keep doing doing the "go" procedure forever (until you "unpress" it)



Manchester Metropolitan University

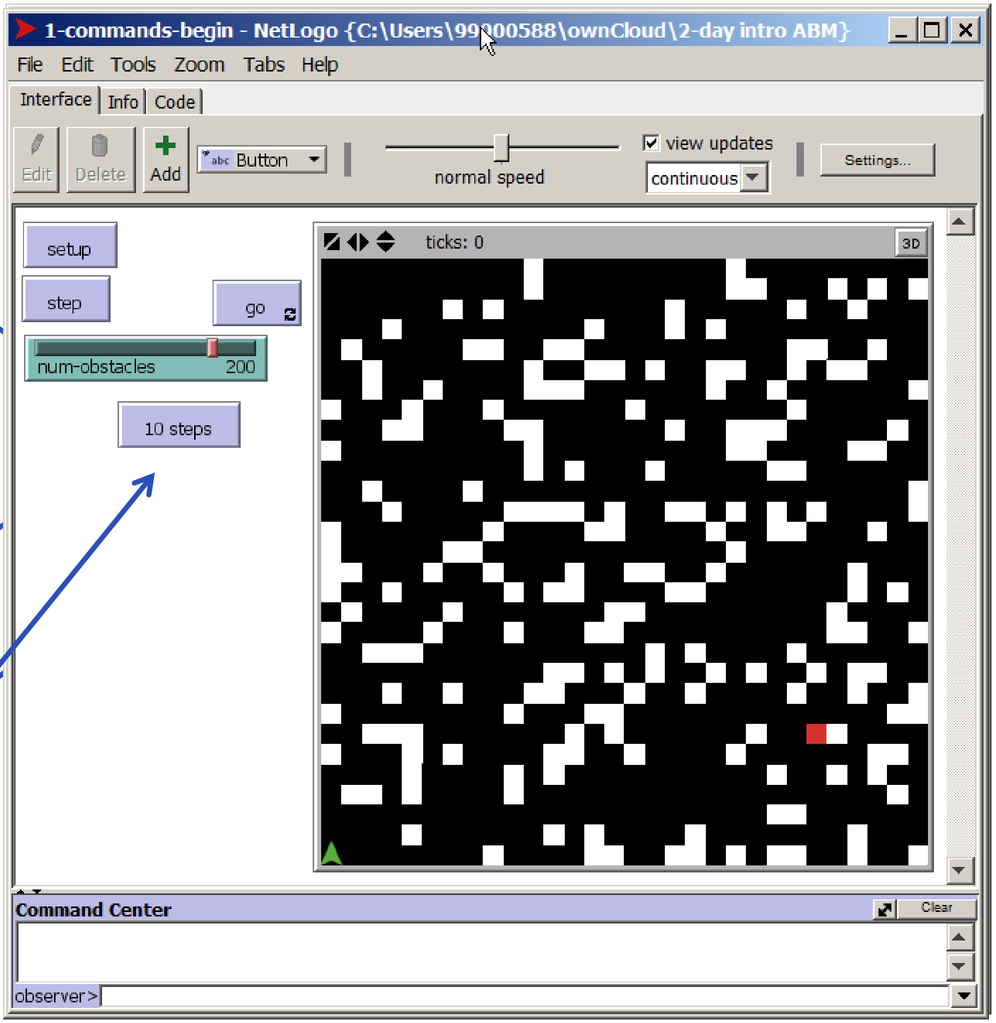
Adding a button and running the code for only 10 steps

Right-Click some different empty space and choose "**button**"

Type the text "repeat 10 [go]" here

Type the text "***10 steps***" here

Now when you press the "10 steps" button it will do the "go" procedure only 10 times



Manchester Metropolitan University

Types of Agent

- To make the programming clearer you can define different types of agent for different roles and purposes
- The built in general type “turtles” refers to ***all*** these kinds of agents
- (patches and links are of a different and fixed type)
- This is done in the declaration section at the top of the program code, e.g.

```
breed [people person]
```

- Once declared many commands use the breed name as part of the command, e.g.

  create-people 1 [… *some commands* …]

- As well as being referred to directly, e.g.

  ask people [… *some commands* …]

Manchester Metropolitan University

Reacting to other agents

- Reacting to and with other agents is at the core of most social ABMs
- Even simple mutual reaction can result in quite complex outcomes
- In simulations where it is hard to understand how the resultant patterns of the whole (the macro-level) come out of the behaviours of the agents (the micro-level) this is called "emergence"

Fixed vs. Reactive vs. Adaptive vs. Reflective Agents vs. …

How agents control behaviour is a matter of simulator choice, e.g…

- Behaviour might be *fixed* – an engrained habit, procedure, or built-in reflex
- It might be *reactive* – a certain response is 'triggered' under certain circumstances
- The agent might have internal memory/states that are changed by interaction and upon which future behaviour depends – this is *adaptive* behaviour
- The agent might do something more complicated… weighing up future alternatives, solving a puzzle, reasoning about possibilities etc. – these *reflective* actions are quite complex to program

Manchester Metropolitan University

The “voter” simulation

- This is a very simple simulation where votes and parties are spread over a political spectrum – voters vote for the party nearest in position to them, parties shift position if they do not win

Manchester Metropolitan University

“AgentSets” in NetLogo

One powerful facility in NetLogo is the ability to deal with sets of agents. Examples include:

- turtles – all agents
- parties – all agents of the breed “party”
- parties with [not won?] – the set of parties with the won? property set to false
- [color] of chosen-party – extracts the value(s) from a set of agents
- one-of voters – a random one from all in voters
- max-one-of parties [votes] – the agent in parties with the most of property: votes
- min-one-of parties [abs (political-position - [political-position] of myself)] – the agent in parties with the minimum value of abs (political-position - [political-position] of myself) in other words, the closest to its own political position

The category called “**Agentset**” in the NetLogo dictionary shows some of the primitives that can be used with these

The importance of visualisations

- Due to the fact that it is (relatively) easy to create a simulation you do not understand and that…
- …You can not rely on your intuitions and classic outputs such as aggregate measures/graphs
- Making good visualisations of what is happening is very important
- I often spend as much time on getting the visualisations of a model right as I do the original “core” programming
- And this can allow a “step change” in my understanding
- The NetLogo “world view” is ideal for this

Manchester Metropolitan University

Discussion – Interpreting an ABM

- Simulations (indeed any model) is meaningless without *some* interpretation of what things are meant to stand for to guide model development and investigation
- *How do **you** interpret your observations of the model with 100 voters and 3 parties?*
- The questions:
  - How meaningful is the simulation?
  - How empirically realistic is the simulation?
- Are not *quite* the same!

Manchester Metropolitan University

A change to the simulation setup

- In the setup procedure, where voters are created, change the command set political-position random-float 1 to: **set political-position random-normal 0.5 0.15**
- This changes the initial distribution of voters from a uniform one to a normal distribution
- Go back and re-investigate the behaviour of the simulation with this setup
- How much does it change the results? Just a bit? Qualitatively different?

Manchester Metropolitan University

Randomness!

- It is very tempting when some process is either complex or unknown to chuck in a random choice
- But this is as much a definite choice with consequences as any other and should be used with **caution**!
- It is OK when…
  - this is just a temporary 'stub' which will be replaced later (but then this needs to be declared if it is left in)
  - One just needs a variety of behaviours for exploratory/testing purposes (but then if you are publishing the results you have a different purpose)
  - One knows the behaviour IS random (check the evidence that this is so)
  - One is pretty sure that the behaviour is irrelevant to the outcome one is looking at (run the model with different kinds of behaviour and check it makes no difference)
- But otherwise it might be better to replace it with something more definite or more realistic

Manchester Metropolitan University

Interaction Structures

There are a number of possible ways of structuring agent interactions, including:

- Randomly – others are chosen from population at random
- Via space – those within a certain distance or in nearby/the same space
- Via a social network of links
- Only indirectly via the environment

All of these are relatively simple in NetLogo

Making & using a network in NetLogo

```
ask turtles [
  create-links-with n-of number-links-each other turtles
  ]
```

- For each node: make links with the set number of others, but not oneself (hence the "other")
- n-of returns that number of the set provided it (at random)
- Later any node can be asked to do something with all its "link-neighbours" – a set of all those it is connected with, e.g.:

```
 if any? link-neighbors with [color = red] [
     …do something or other…
    ]
```

Manchester Metropolitan University

Adding a choice of network I

- Right-click (or ctrl-click) on some empty space and choose “Chooser”
- In the dialogue that appears, enter network-type for “Global Variable” and…

```
"random"
"nearest”
```

- …in the “Choices” box, then press “OK”
- In the **setup** procedure change:

```
ask turtles [
    create-links-with n-of number-links-each other turtles
  ]
```

- to:

```
if network-type = "random" [
  ask turtles [
    create-links-with n-of number-links-each other turtles
  ]
 ]
```

- Now this method is only used to make the network if “random” is chosen in the Chooser dialogue

Manchester Metropolitan University

Adding a choice of network II

- Add the following into the setup procedure (immediately above or below the last bit of code we messed with):

```
if network-type = "nearest" [
  ask turtles [
    create-links-with min-n-of number-links-each other turtles
            [distance myself]
  ]
 ]
```

- If the *nearest* choice is made, for each node this links to the set number of nodes with smallest distance to itself (closest)
- Go back to the interface and try the simulation with this kind of network, evaluate the difference

To spread the graph display out a bit

- Add the command:

```
repeat 100 [ layout-spring turtles links 0.2 5 2]
```

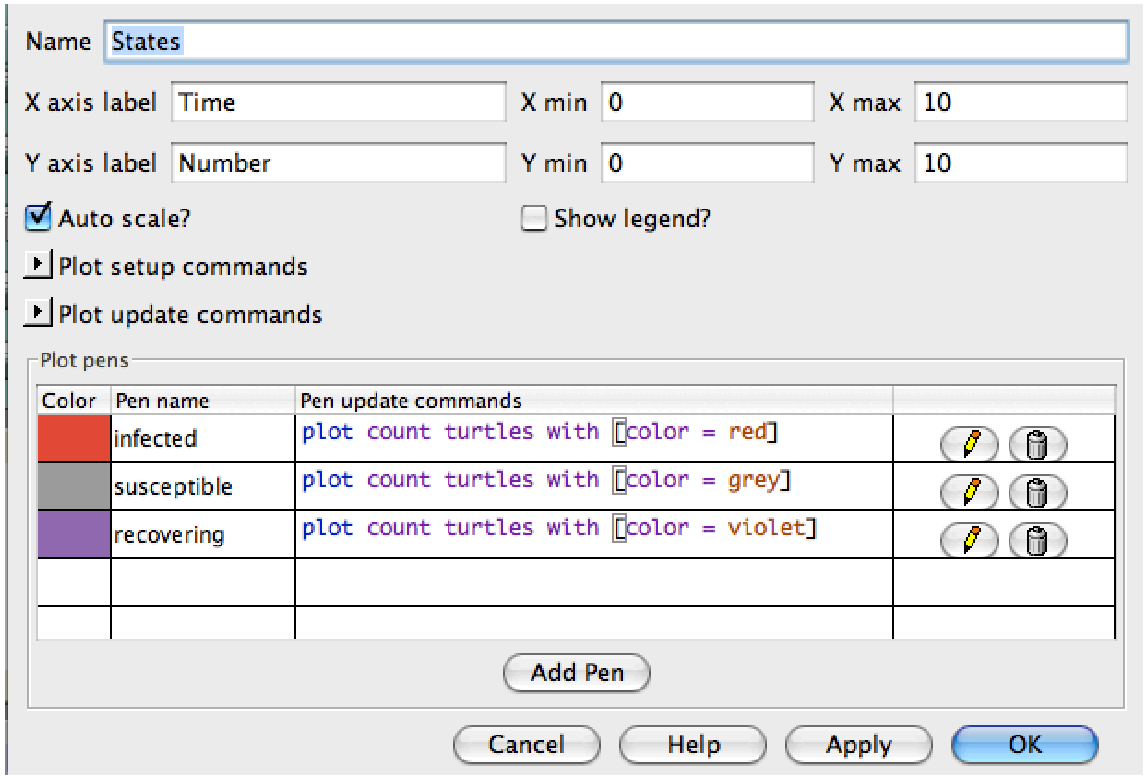
- Just before the "reset-ticks" command in the setup procedure
- "layout-spring" just adjusts the gaps between nodes as if they were connected with certain kinds of spring – doing this 100 times
- This just makes the network easier to see – NetLogo provides a number of varieties of these for different network display styles

Manchester Metropolitan University

Add a plot

- Right-Click on some empty space and choose "Plot", then enter the following information before pressing "OK"

To add a new plot line, press the "**Add Pen**" button. To change the pen name click on the name, delete the existing name and type your own. To change the colour, click on the colour, choose a colour and then "**OK**".



Name: States

X axis label: Time | X min: 0 | X max: 10

Y axis label: Number | Y min: 0 | Y max: 10

[✓] Auto scale? [ ] Show legend?

Plot setup commands

Plot update commands

Plot pens

Color	Pen name	Pen update commands
🟥	infected	`plot count turtles with [color = red]`
⬜	susceptible	`plot count turtles with [color = grey]`
🟪	recovering	`plot count turtles with [color = violet]`

Stage of Debugging

- Most coding time is not spent making the original simulation but in “fixing” it or adding more code in (e.g. plots, new aspects)
- More careful design can help reduce time spent debugging but it still dominates

There are different stages of debugging (in order of both occurrence and difficulty):

1. Fixing syntax errors so the code runs
2. Making the micro-level behaviours work as you intended them to (verification)
3. Making the resultant outcomes be as you want them to be (tuning and validation)

Manchester Metropolitan University

Strategies for debugging

- Keep adding more graphs, monitors, visualisations etc. so you better understand what is happening
- Turn parts of the behaviours “off” by temporarily changing the code (e.g. make all prices fixed) to see what happens
- Temporarily adding “show” statements into the code to show what is happening, e.g.
  - `show (word "Food of " self " is " food)`

Agent Case Studies

- Global outputs such as graphs, monitors, visualisations can only go so far…
- There is often nothing else for it but to follow a particular agent step by step checking what it does and its state *each time* compared to the code
- This is time-consuming and everybody tries to avoid doing it…
- …you often will simply not really understand your simulation unless you do!

Manchester Metropolitan University