A Brief Introduction to Policy Modelling

Using agent-based simulation

Session 1

About agent-based simulation

Centre for Policy Modelling

Manchester Metropolitan University

Introduction:

some history and motivation, a first model to play with

Welcome

- This brief introduction is part of the EAEPE Conference, Manchester Metropolitan University, Nov. 2016.

It is organised and run by

- **Bruce Edmonds** and **Ruth Meyer** from the *Centre for Policy Modelling* at the Manchester Metropolitan University

- If you have not installed NetLogo (version 5.2.something), please do so now from: http://ccl.northwestern.edu/netlogo

Aims of the Sessions

To introduce you to:

- Agent-based simulation
- Its application to policy modelling

It ***will not*** get you to a point where you can start to program your own simulations, this takes quite a bit longer (just like any other technique).

However it ***will*** give you an idea of what this approach can do, some of its difficulties, and help you understand some of its key ideas

Manchester Metropolitan University

Outline of the Sessions

Session 1: *Agent-based simulation* (ABS)

- Background and introduction
- About ABS, including how a model is made
- Some examples to play with

Session 2: *Its application to policy issues*

- About policy modelling
- Some more applied examples to play with
- What to do next if you want to go further
- Concluding discussion and Q&A

The importance of formal models

Formal models are important because:

- they allow the sharing of representations (without being changed by re-interpretation), and so allow a community of researchers to critique, check, compare and improve such models collectively
- they can be indefinitely elaborated to ‘fit’ almost anything if complicated enough
- they are complementary to natural language accounts – allowing the properties and outcomes of processes and systems that are too complicated for the mind to track, to be reliably traced and inspected

Manchester Metropolitan University

Mathematical formal models

- Arithmetic, trignometry, difference equations, differential equations, statistics etc.
- Used to be the only kind of formal model practically available and thus were essential to any science of the measurable
- After WWII these models were applied to understanding social phenomena
- If one wanted to analytically solve them, one is limited to quite simple systems
- Which limited their use to: analogies of behaviour or aggregate summaries of systems

Manchester Metropolitan University

Economics

- By focusing on money (rather than value), preferences (rather than item properties), optimizing utility functions (rather than choice processes) etc. it was found that some interesting models of social phenomena involving exchange were solvable
- This became known as neo-classical economics
- It might have been that such models would approximate the behaviour of observed economic phenomena, in aggregate
- However, it became more interested in the mathematical properties of its models than whether it successfully captured observed phenomena

Simulation

- Numerical solutions to mathematical models have been around a long time
- Computers made this a feasible approach
- Alternative models of computation (such as cellular automata) appeared since WWII
- Simulation developed into an essential tool of science and engineering, allowing systems that were too complex for solvable mathematics to be modelled

Agent-Based Simulation

- In 1967 the computer language Simula 67 was specified and developed for organisational modelling
- This used separate computational entities for each thing modelled, and messages between the entities for interaction (this became object-oriented computing)
- One use is to model social systems, (such as the Schelling model in a following slide)
- (the computational entities are called 'agents' when the entities can be usefully interpreted as having cognition)
- It was no longer *necessary* to put up with simplistic or wrong assumptions just to get a useable formal model
- Solvable mathematical models and computational models each have different pros and cons…
- …but now can *choose* the most appropriate kind of formal model for the phenomena one is dealing with

Schedule, Course Material, etc.…

The course materials, examples etc. are all at

http://cfpm.org/eaepe

More materials for a fuller, 2-day course are freely available at:

http://cfpm.org/simulationcourse

This latter site has pointers to:

- The example models and the slides that explain them, divided into 8 sessions
- Further material on the web
- Pointers to books, links, tutorials etc.

Manchester Metropolitan University

About NetLogo

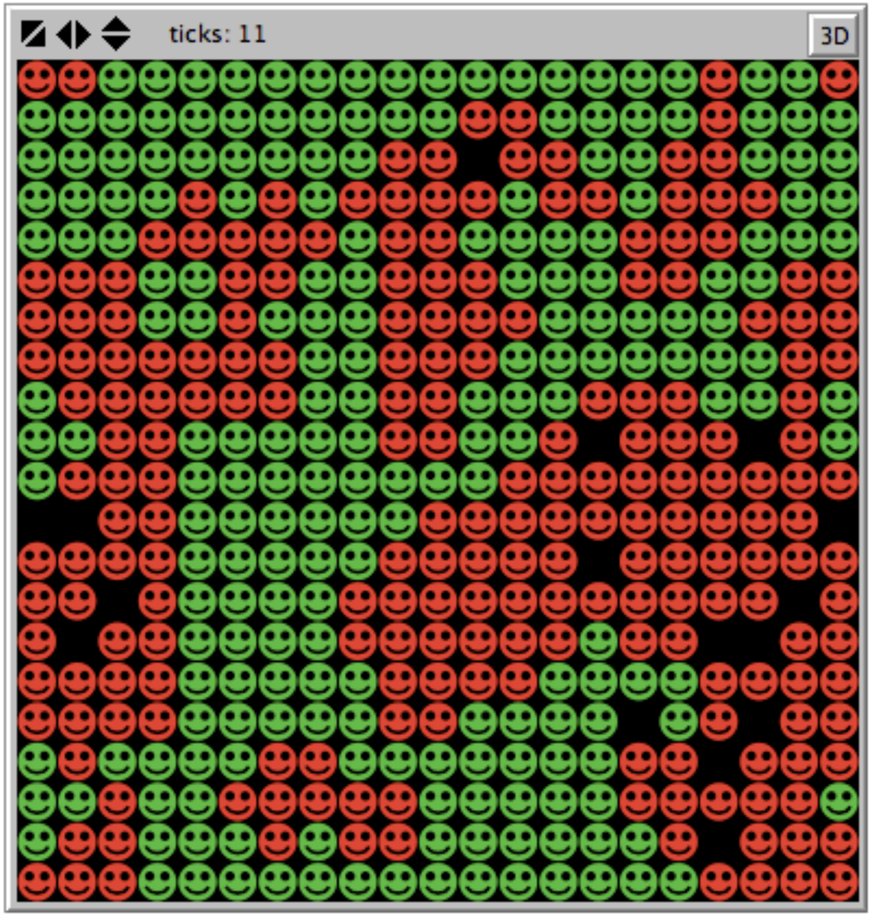
- "Logo" was a language invented by Semour Papert (a student of Piaget)
- Designed to be easy to write and read – 'english like' – a large built-in vocabulary and designed so one can progressively build up a personal set of user-defined words
- Designed as a language to explore ideas with in a playful manner
- NetLogo is a development of this, but with agents, patches etc. to facilitate the accessible construction, sharing and exploration of simulations
- But it is a full programming language and you could program any simulation in it if you wanted to
- Has become somewhat of a standard in the social sciences, with many available models (e.g. at http://OpenABM.org or bundled in the package)

A Classic Example of an Agent-Based Model: *Schelling's Segregation Model*

Schelling, Thomas C. 1971. Dynamic Models of Segregation. *Journal of Mathematical Sociology* **1**:143-186.

Rule: each iteration, each dot looks at its 8 neighbours and if, say, less than 30% are the same colour as itself, it moves to a random empty square

*This was a kind of **thought experiment** to look at the possible outcomes that result from the above rule*



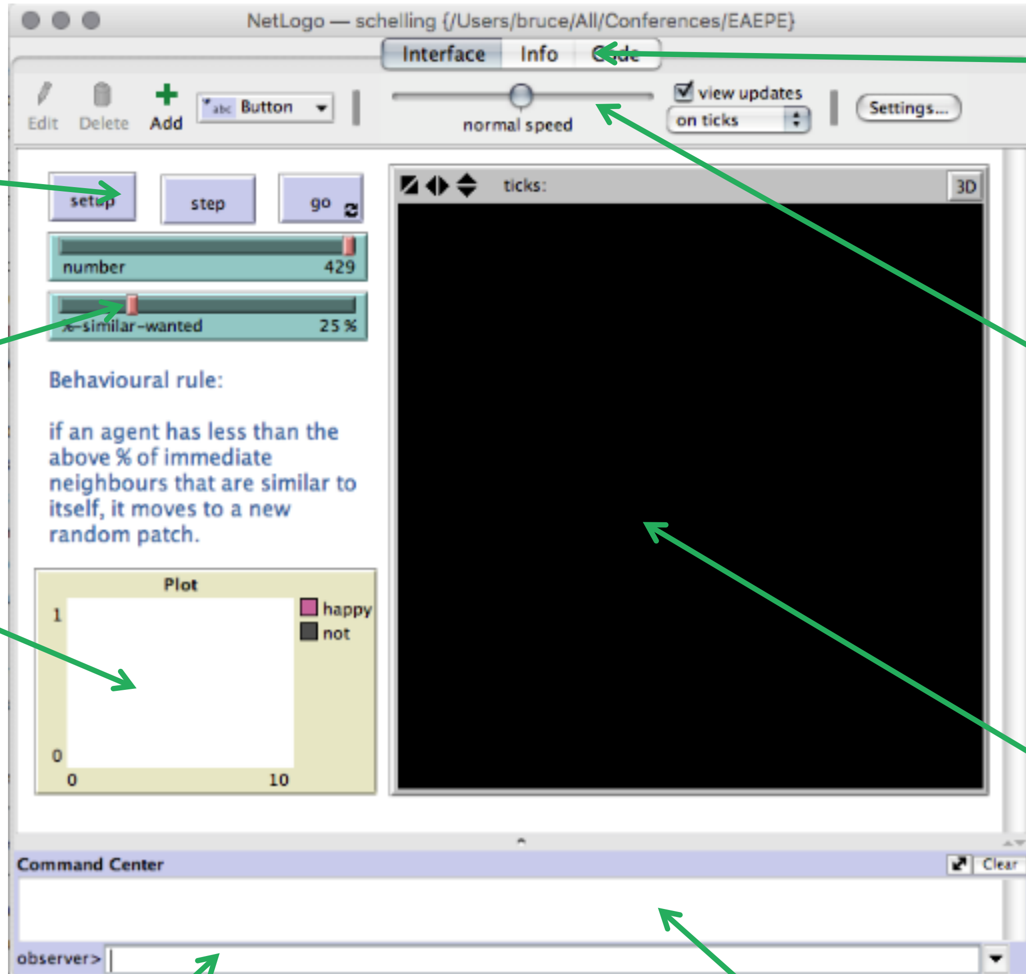
Manchester Metropolitan University

Staring the first NetLogo Model

- Goto http://cfpm.org/eaepe and download the file "schelling.nlogo" file
- If NetLogo is installed properly this should load and run when launched, if not you may have to start NetLogo and use "File" then "Open" etc. to load the model.

Manchester Metropolitan University

NetLogo – Interface Panel



Panel Selection (looks slightly different on Windows and Macs)

Command Buttons

Parameter Slider

Speed Control

Graph of outcomes

Visualisation of world

Typed Direct Commands

Text Output

Manchester Metropolitan University

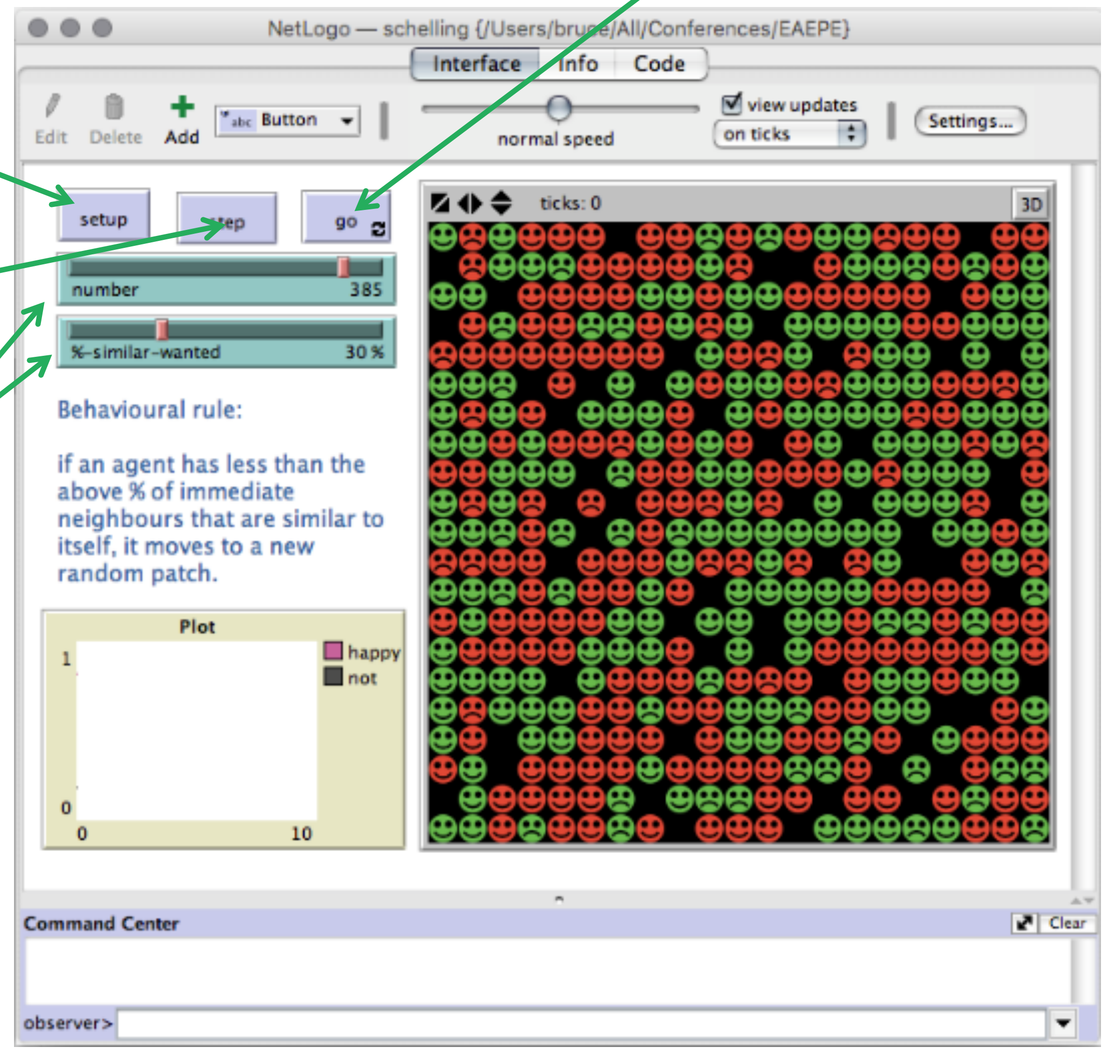
Play with the model!

4. The “go” button means “repeatedly do go”

2. Press “setup” to initialise model

3. Press “step” to make model do one set of instructions, press again…

1. Set the parameters



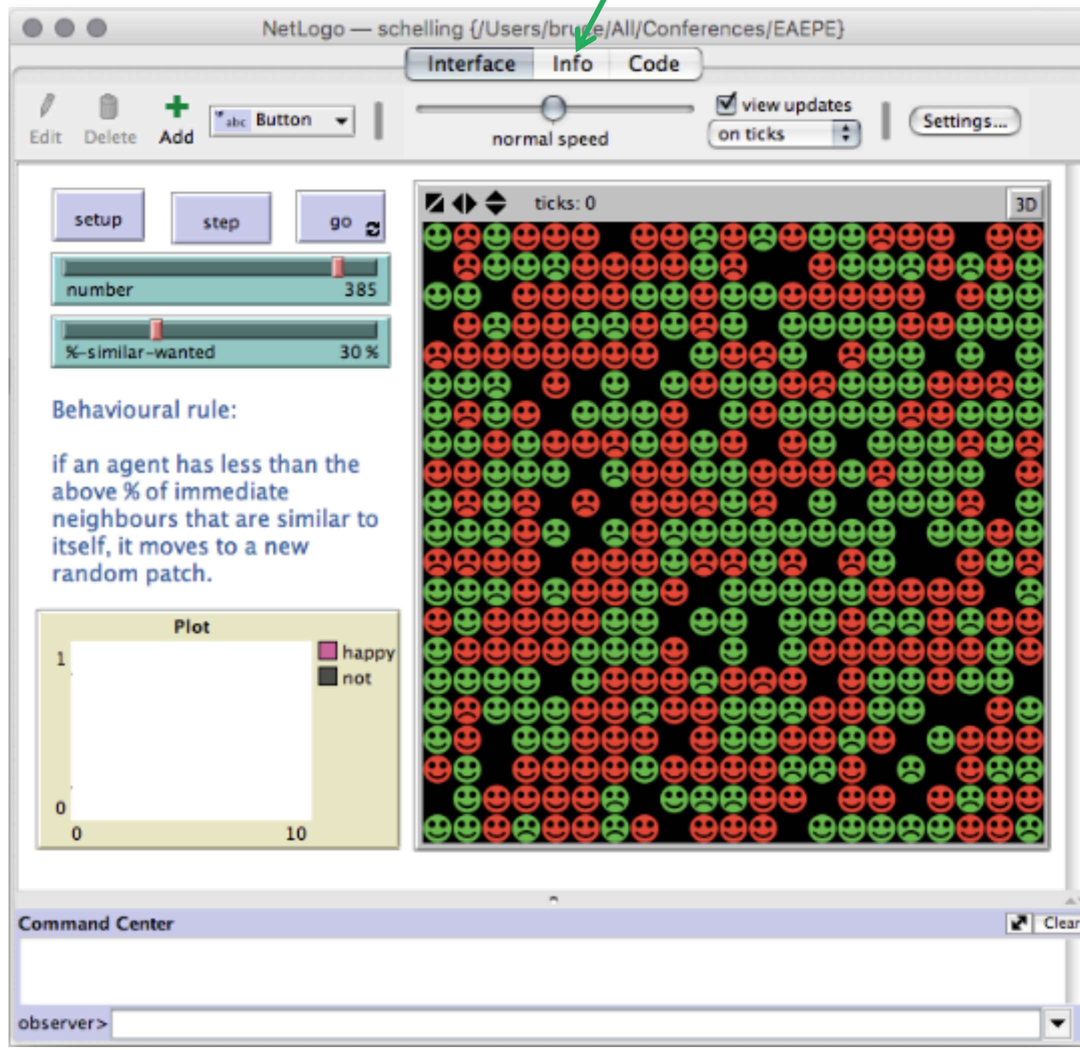
Play with the model, try different parameters, see what the results are, what do you conclude?

Manchester Metropolitan University

Play with the model!

These change the view of the model

- Try swapping between "Interface", "Info" and "Code" panels
- Read the explanation
- Look at the code



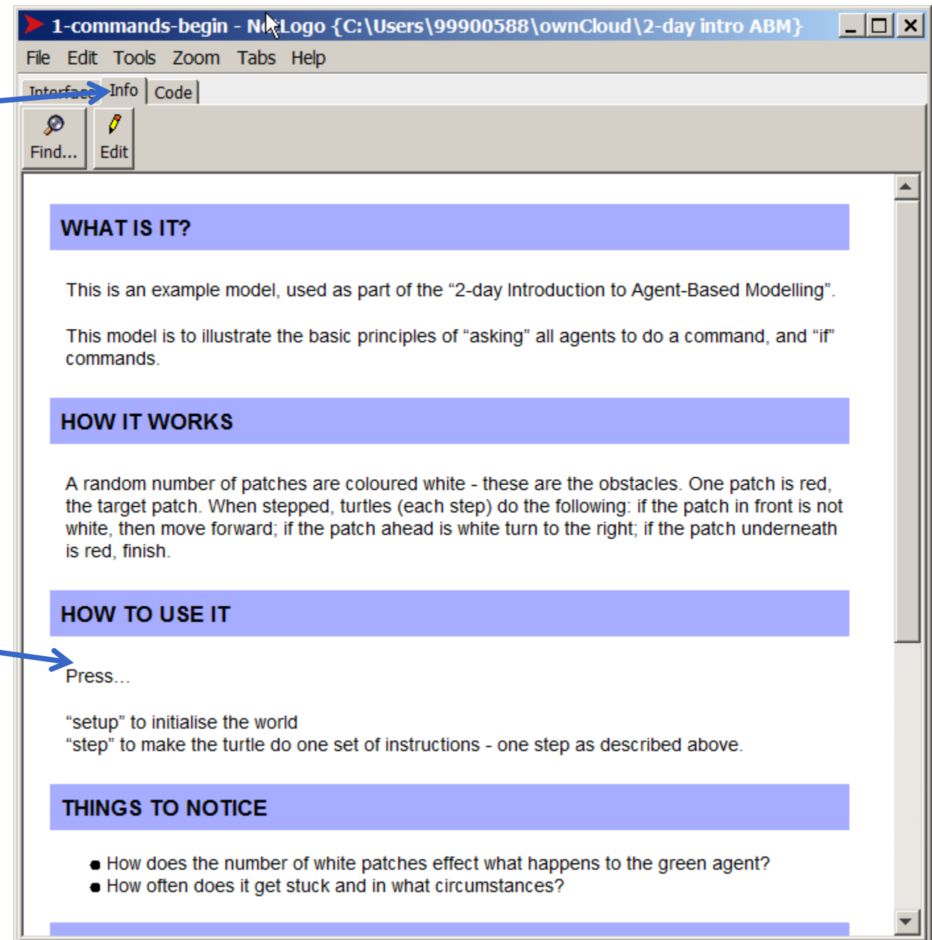
Manchester Metropolitan University

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



Manchester Metropolitan University

The Code Tab

Code is complex and it will take some time to learn to "read" it (but NetLogo code is a ***LOT*** easier than most)

Grey=comments (you can ignore these)

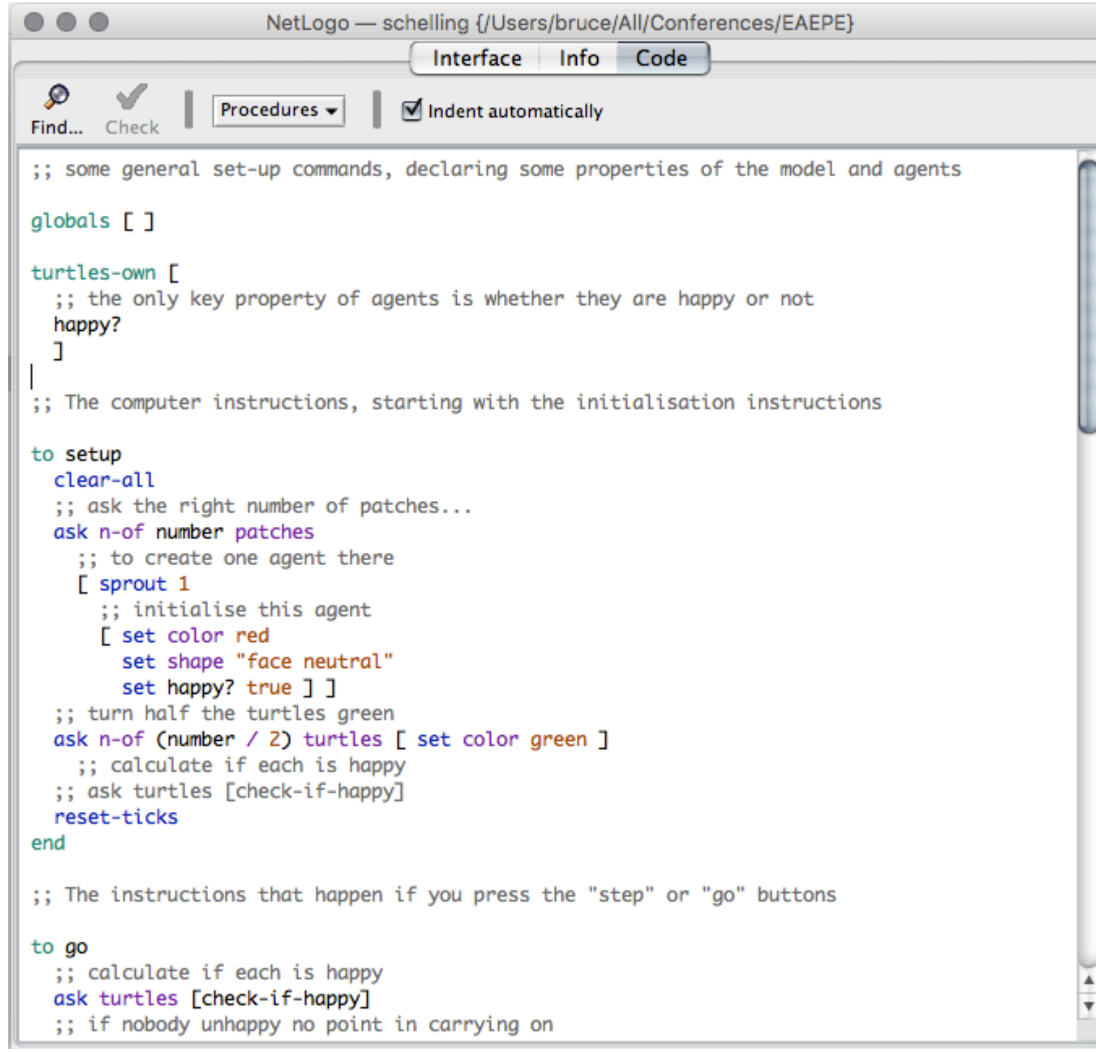
Green = special structural words

Blue = commands

Purple = NetLogo defined

Red = constants

Black = programmer defined words



Manchester Metropolitan University

NetLogo Documentation

- The NetLogo documentation is good – accessible, well written, gives examples and is extensive
- Access it either by using the menus:
  - ‘Help >> NetLogo User Manual’
- Particularly useful is the Dictionary got to:
  - Either from the Manual main page
  - Or ‘Help >> NetLogo Dictionary’
- Most experienced NetLogo programmers work with this open all the time, referring back and forth to it as they program

Questions about model outcomes

What do you notice about the segregation model – can you answer any of these?

- What happens if there are no spaces free?
- What happens if there are only a very few spaces free?
- What happens if there are a lot of spaces free?
- What happens with very low "%-similar-wanted"?
- What happens with very high "%-similar-wanted"?
- What happens if you gradually increase "%-similar-wanted" (0% then "go", 5% then "go", 10% then "go" etc.)?

Manchester Metropolitan University

Discussion

- Outcomes, even from very simple rules, are difficult to anticipate…
- ...until one has spent time playing with the model, then it may seem obvious
- Small changes in the rules or parameters can cause big changes in outcome – qualitatively as well as quantitatively
- But one can test these quite easily!
- The behavioural rules can be anything, and do not have to be restricted to any particular theory
- However the space of possible models and settings is HUGE and can not all be explored

Using NetLogo commands, changing code, adding to the interface

A very simple model to get an idea of how such simulations work

- Goto http://cfpm.org/eaepe, download the file "commands.nlogo" model and run it
- This is a very simple model to illustrate how commands are run in simulations – how NetLogo works

Manchester Metropolitan University

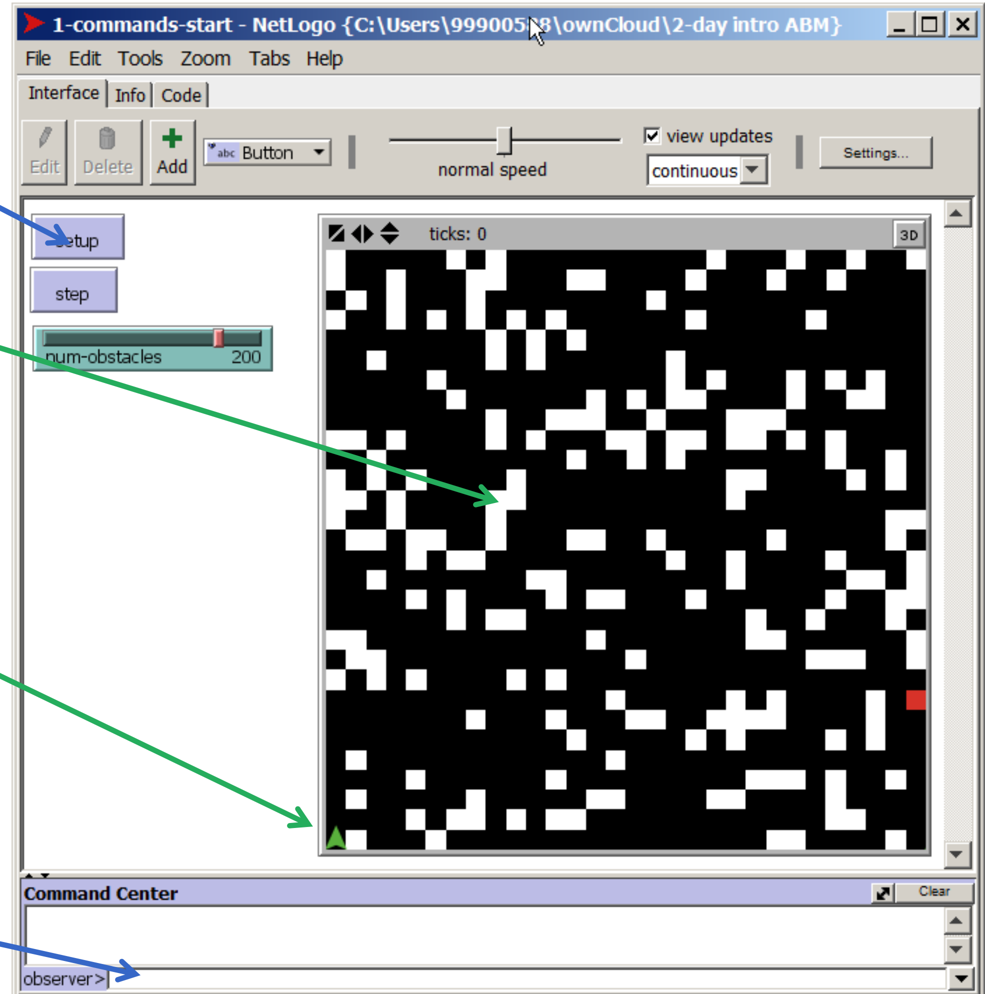
Typing in Commands

Press “setup” to initialise world

World with different colour patches

An agent!

Type commands in here as follows…



Manchester Metropolitan University

The command centre…

- “show” means show the result in the command centre

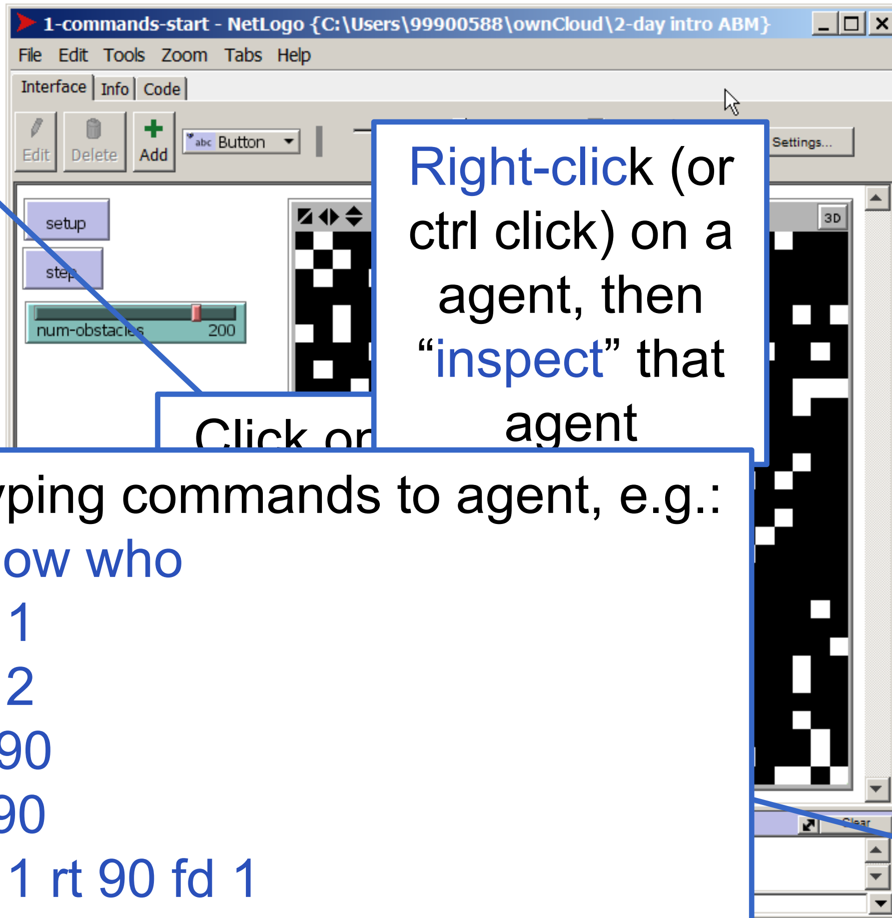
Try:

- show timer (and then try this again)
- show count agents
- show agents
- show sort agents
- show count patches
- show count patches with [pcolor = white]

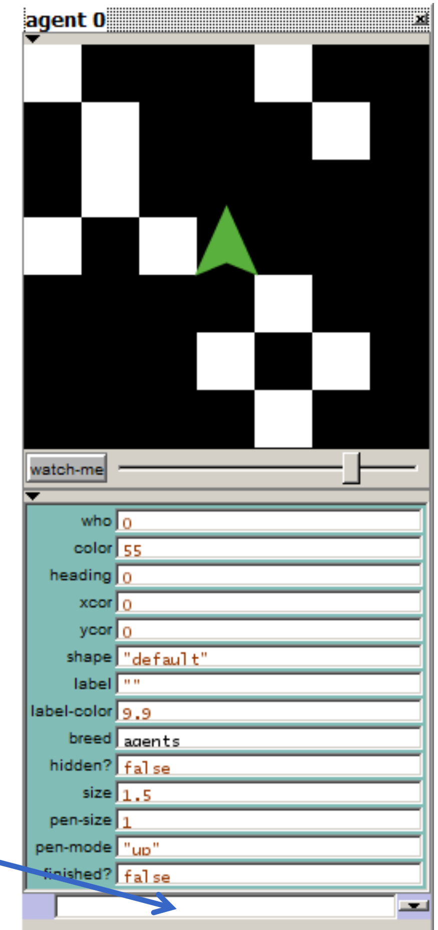
Anything typed into the command centre is from the “observer” point of view (yours!)

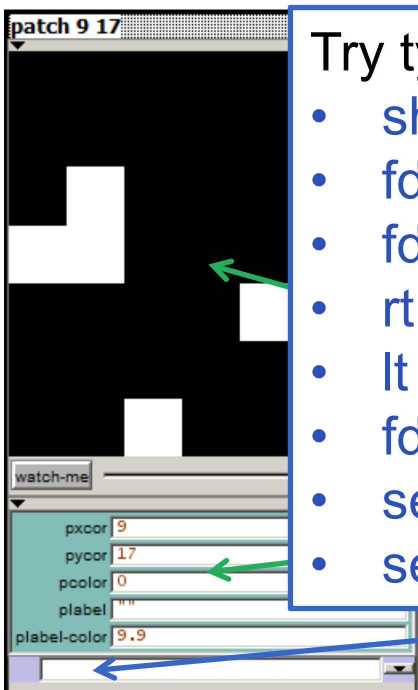
Inspecting Patches and Agents

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



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





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 purple
- set size 4

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

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

Manchester Metropolitan University

Using "ask"

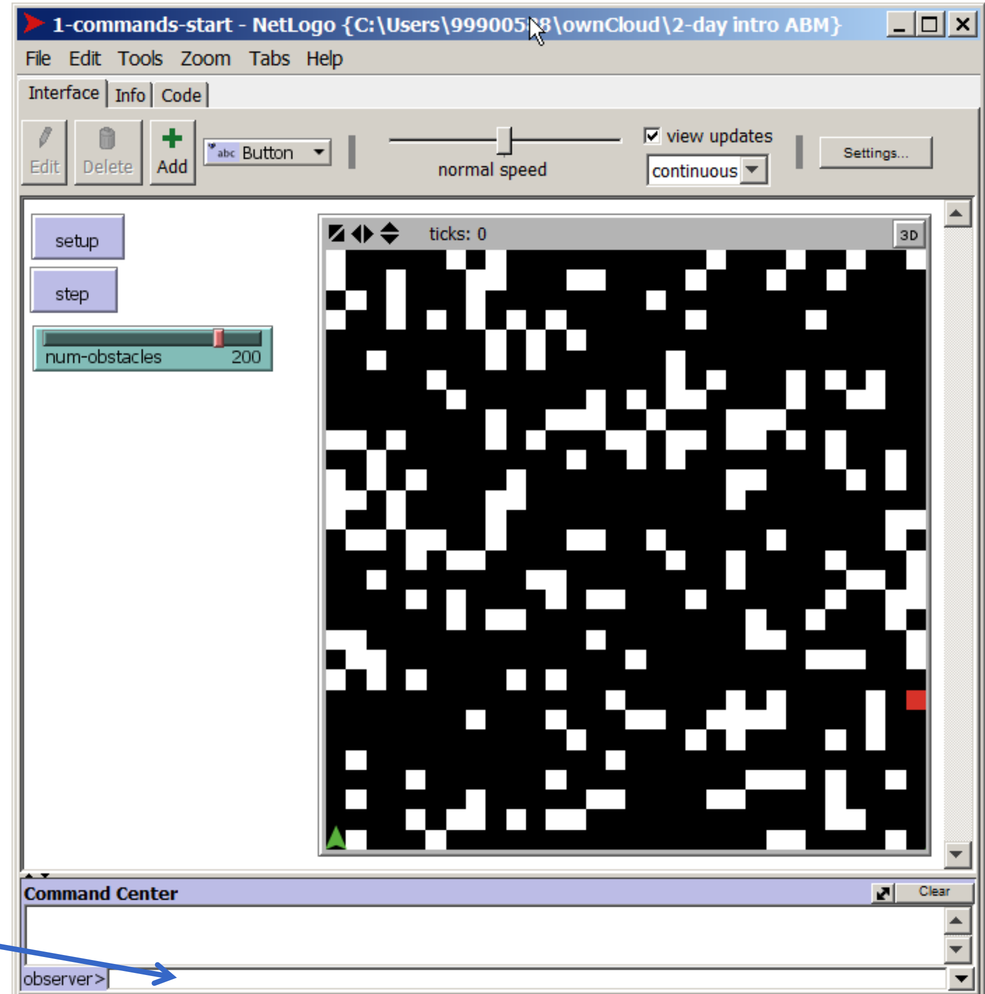
"ask" sends commands to a whole set of agents (one at a time in a random order)

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

- ask agents [fd 1]
- ask agents [set color grey]
- ask agents [set shape "person"]
- ask agents [fd 1 rt 90 fd 1]
- ask agents [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

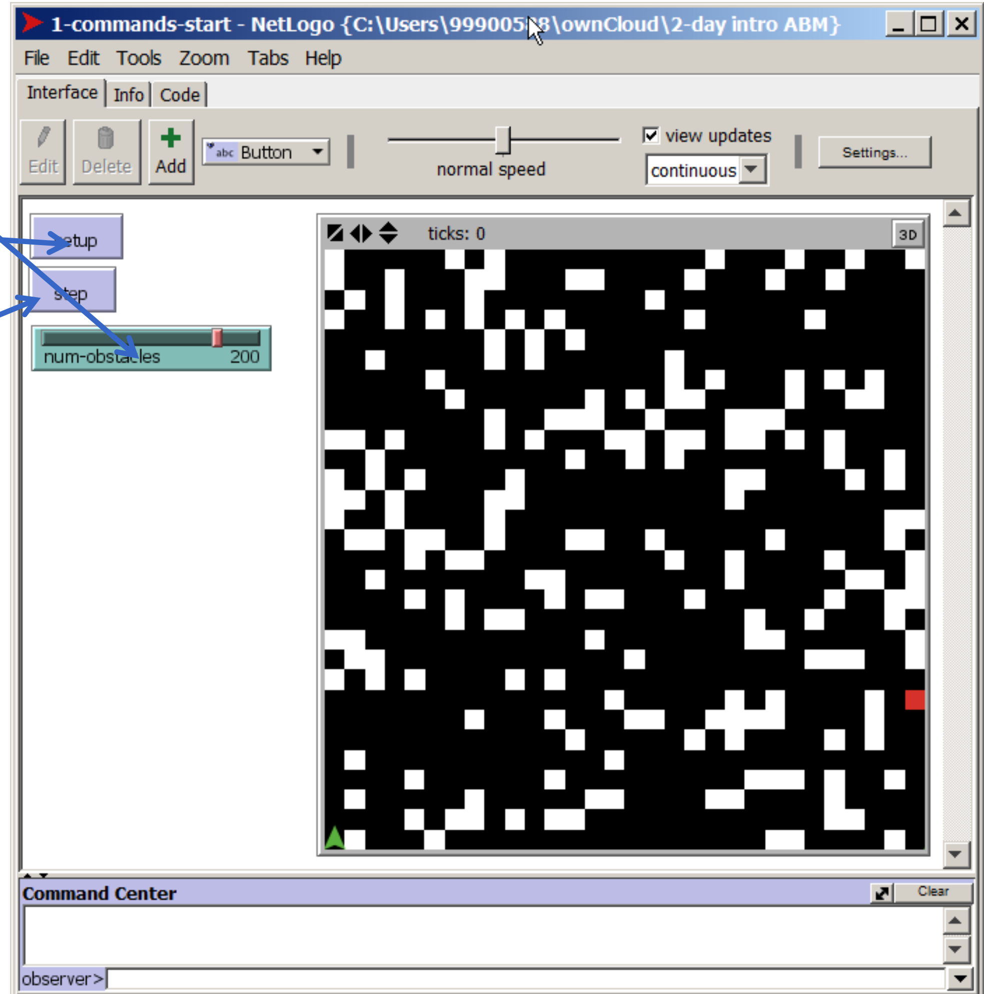
Running a simulation (the hard way!)

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 “step” is caused to run – this is a list of commands, a program.
- We will now look at this.

Manchester Metropolitan University

The Program Code

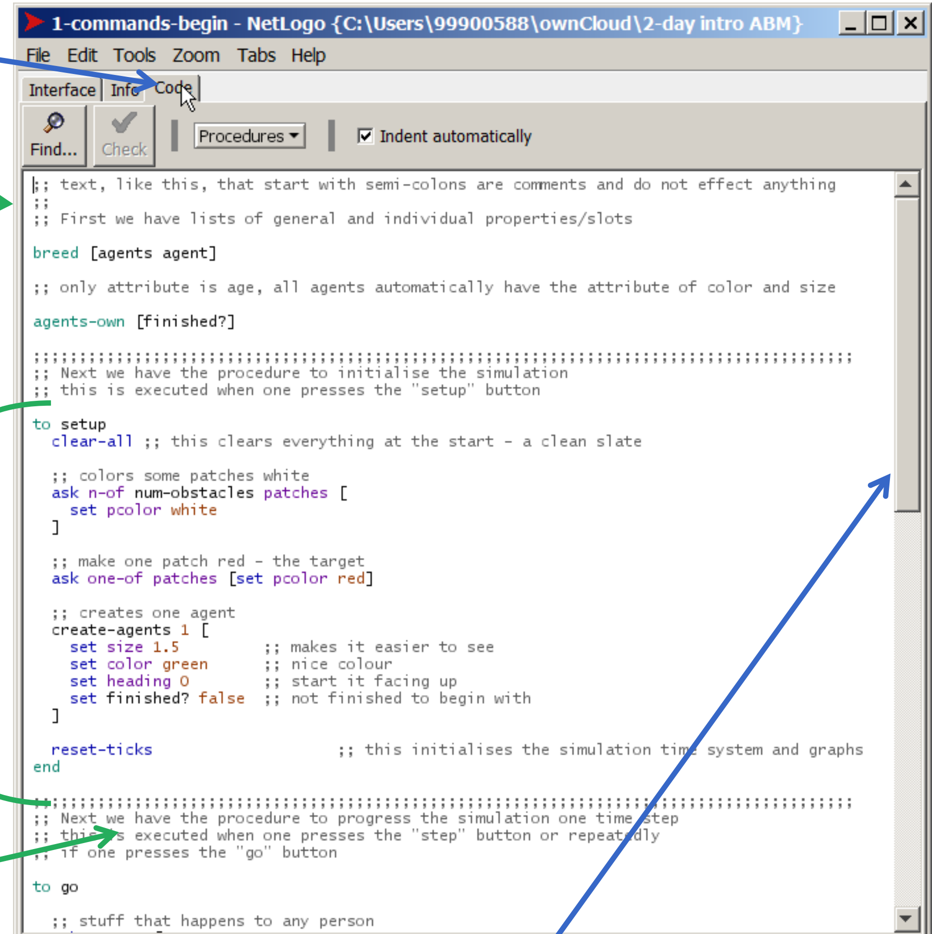
Click on the “Code” tab to see the program

This text is the program

It has different parts

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

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



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

Manchester Metropolitan University

Parts of the Code

"ask agents" means to ask (all) agents to do some

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

conditionals they have a condition and an action

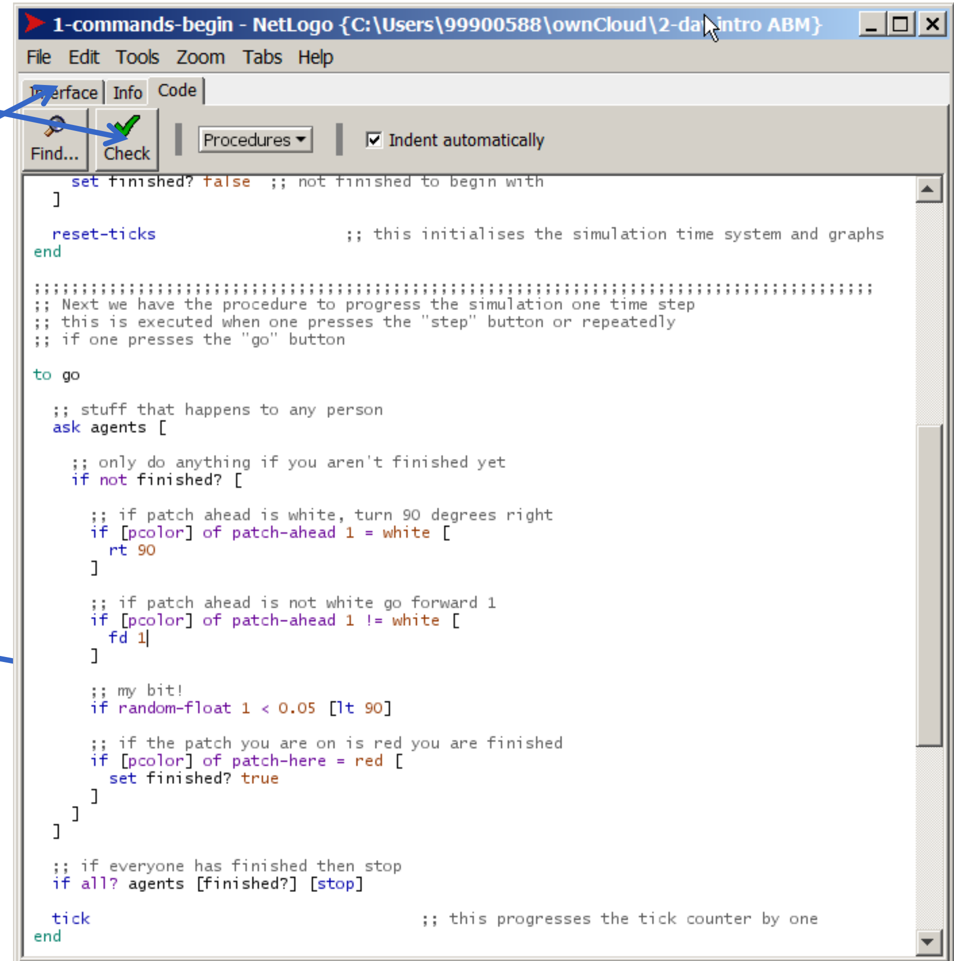
```
to go

  ;; stuff that happens to any person
  ask agents [
    ;; 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? agents [finished?] [stop]

  tick                                        ;; this progresses the
end
```


Manchester Metropolitan University

To change the program…

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)

see if … thing



Click within the text and type!

type the following:

;; my bit!

if random-float 1 < 0.05 [lt 90]

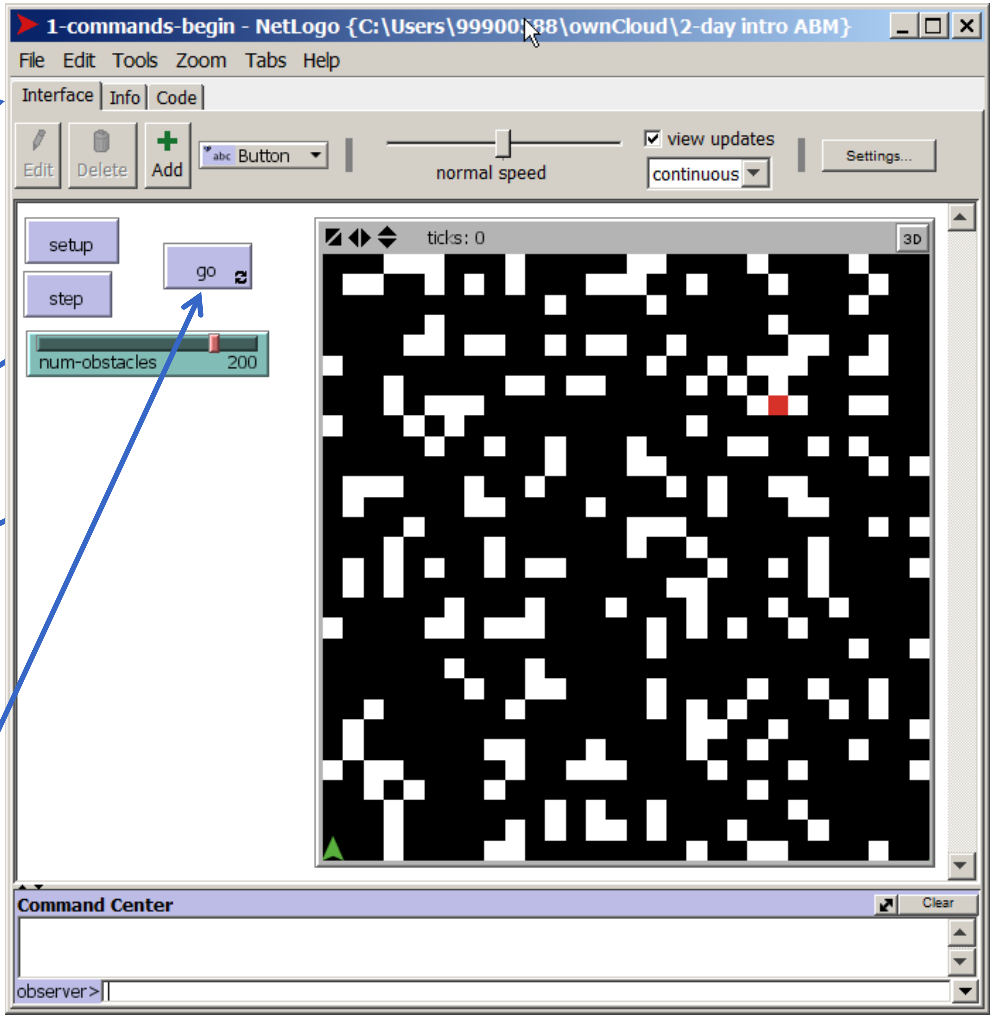
Adding a button and running the code (the fast way!)

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

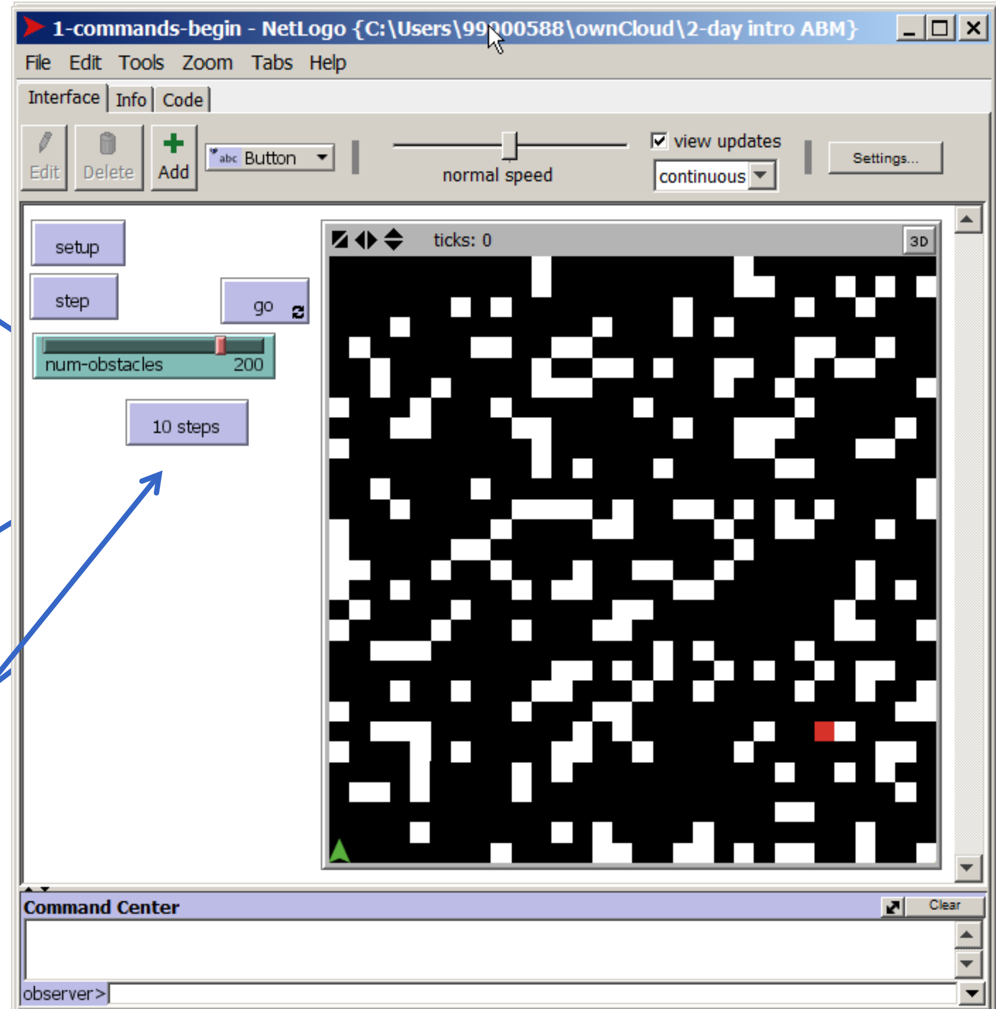
Adding a button and running the code for only 10 steps

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

Key facts

- Programs are lists of commands, in this case done one after another
- In NetLogo, there are different kinds of active 'agent' that can execute code, e.g. the 'turtles', patches, the observer context
- Some commands (e.g. 'ask') can pass control to other agents, so they can execute commands
- So in NetLogo (and many other languages) you have to remember who is doing it

The Experimentation Cycle

Often programming, especially in the exploratory phase, involves a cycle of:

- Writing some code
- Trying it out (as part of a program or as a direct command)
- Finding errors
- Reading the NetLogo documentation (more on this next session)
- Correcting Errors
- Until it works as you want it to (if ever!)

Manchester Metropolitan University

Heterogeneous Adaptive Agents
– *a model of voting for parties*

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

The “voter” simulation

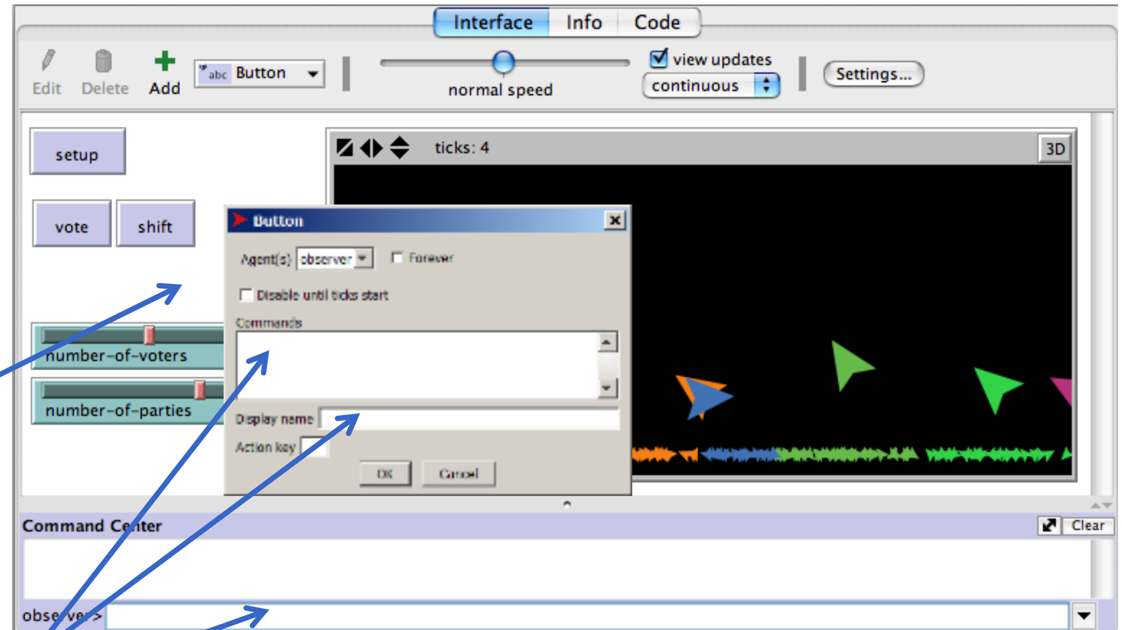
- This is a very simple simulation where votes and parties are spread over a political spectrum – voters vote for the party nearest in position to them, parties shift position if they do not win
- Load the simulation “voting.nlogo”
- Choose the number of voters and number of parties you want
- Initialise the simulation (“setup”)
- Then experiment with pressing the “vote” and “shift” buttons (the later causes all parties who did not win to shift their political position randomly)

Manchester Metropolitan University

Commands and Buttons

All buttons do is cause a given command to be executed when they are pressed – same as typing them in.

Right-Click (Mac: ctrl+click) on some empty space and choose “Button”



Type in the commands you want, in this case “vote shift” and the button name you want “Vote+Shift” then “**OK**”

ote” (followed by ift” here

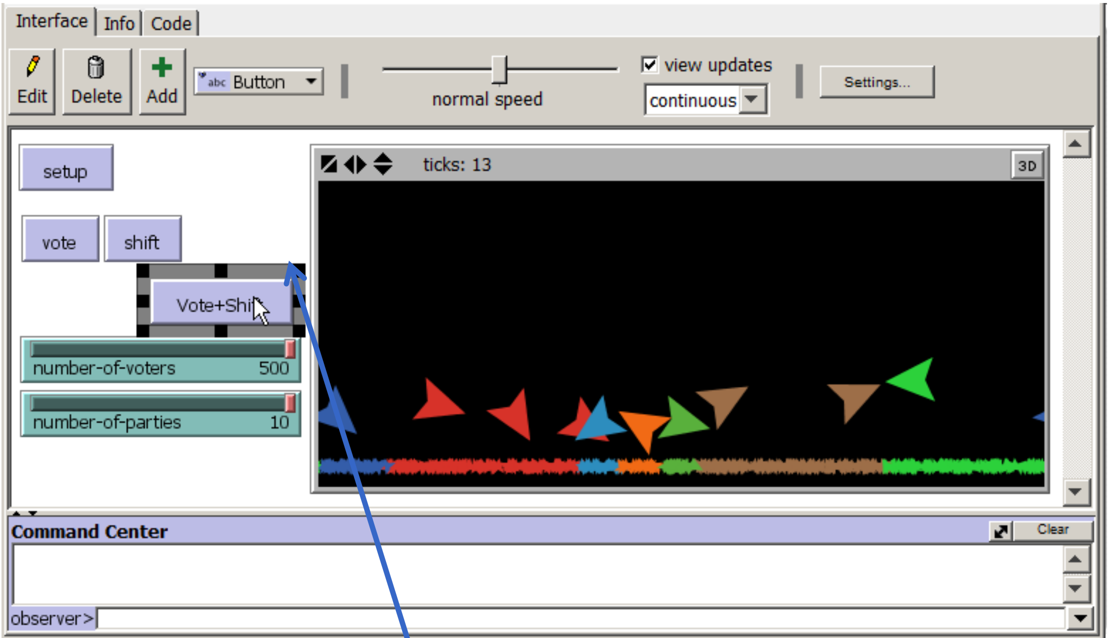
Manchester Metropolitan University

Commands and Buttons

Now try your new button.

Create a new button called "10xVote+Shift" that does the command "repeat 10 [vote shift]"

Now create a button called "go" that does the command "vote shift" but with the "**forever**" option selected



If you need to move/resize the button right-click and "Select" it then drag/adjust it

Manchester Metropolitan University

Improving the look

- Add the command set shape "person" within the **update-voter-appearance** procedure
- Add the command set color [color] of chosen-party within the "ask voters […]" within the "**vote**" procedure. after the "**set chosen-party…**" command
- go back and try the simulation now
- within setup within create-parties add the line set won? false just after the line set political-position…
- within the **update-party-appearance** procedure add the command:
- ifelse won?
  [set shape "face happy"]
  [set shape "face sad"]
- go back and try the simulation again
- experiment with changing the code so that the size of parties depends on how many votes they got

"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

Look at the code again and see if you identify when sets of agents are used and how the code works

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

An Investigation

- Set the number of voters to 100, the number of parties to 3
- Run it quite a few times
- Observed what tends to happen, e.g.
  - How do parties in the middle fare compared to parties on the wings
  - Under what sort of conditions does a party dominate for a period of time?
  - Under what sort of conditions does power switch rapidly between parties?

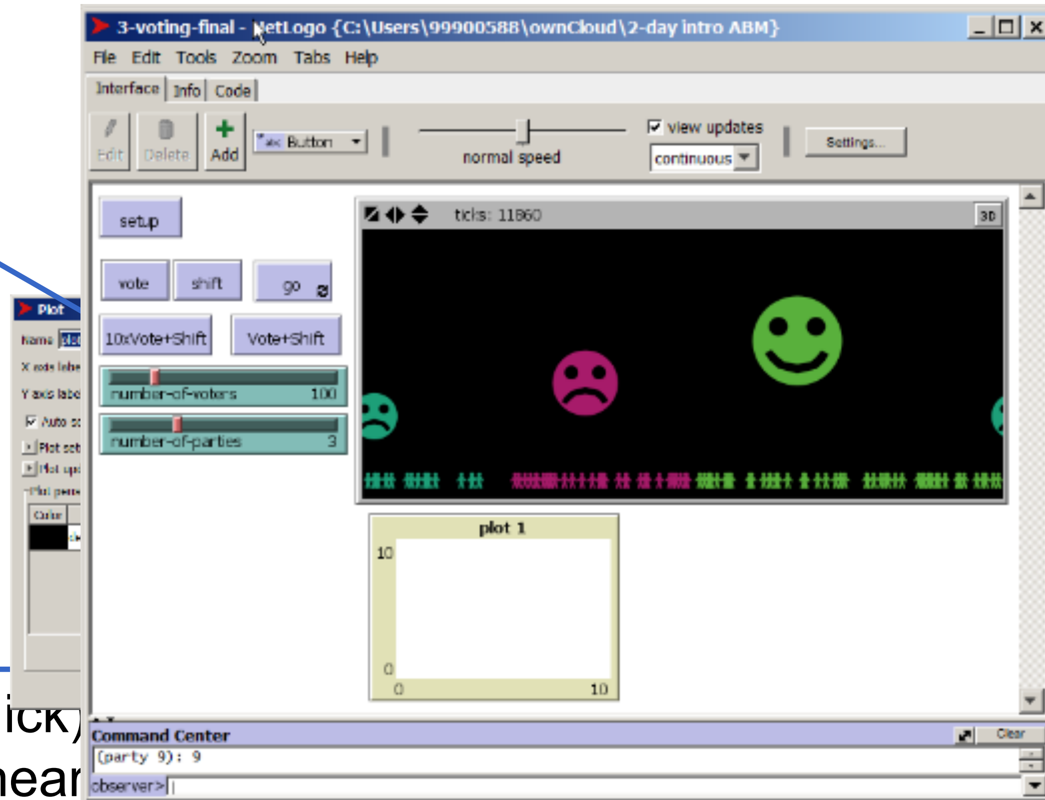
The importance of visualisations

- 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

Adding a graph

Replace the
“plot count turtles”
command there with:
plot [political-position] of
max-one-of parties [votes]
then “**OK**”

In other words to plot the political position of the winning party



Right-Click (Mac: ctrl+click
on some empty space near
the bottom and choose

If necessary, expand the NetLogo window to see the new plot window

Now re-run the simulation looking the political position of the ruling party

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!

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?

Other things to try

- Does changing the initial distribution of parties on the political spectrum change the behaviour of the simulation
- Can you try to change how the political parties adapt to losing?
- Can you add a rule so that voters change their political position as well?
- Can you change the simulation so that all parties somewhat adjust between elections but after an election there is a bigger or different shift?

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

Mutual Adaption and Emergence

– a model of opinion change in a group setting

Mutual Adaption and Emergence

- Many interesting cases come about when agents are mutually adapting, so that the resultant organisation or social structure results from this mutual adaption
- However such a process can be difficult to predict from the initial conditions, this is called "emergence"
- Chance developments during the development of such organisation can determine which of several possible outcomes result
- Sometimes there are several, quite different, kinds of outcome that can occur from the same start
- In such situations, averaging the results from many runs is not helpful, indeed can be very misleading – better to try to characterise the different "phases"

Simulation of Influence with a Group

- Model originated from an EU project looking at how information disseminated to farmers
- They noticed that during meetings opinions often diverged into contrasting groups
- They made an abstract simulation to try and capture this phenomena
- Now a great family of related models along these lines, called "opinion dynamic" models
- This is a simplified version of one of these

Manchester Metropolitan University

Details of this Influence Simulation

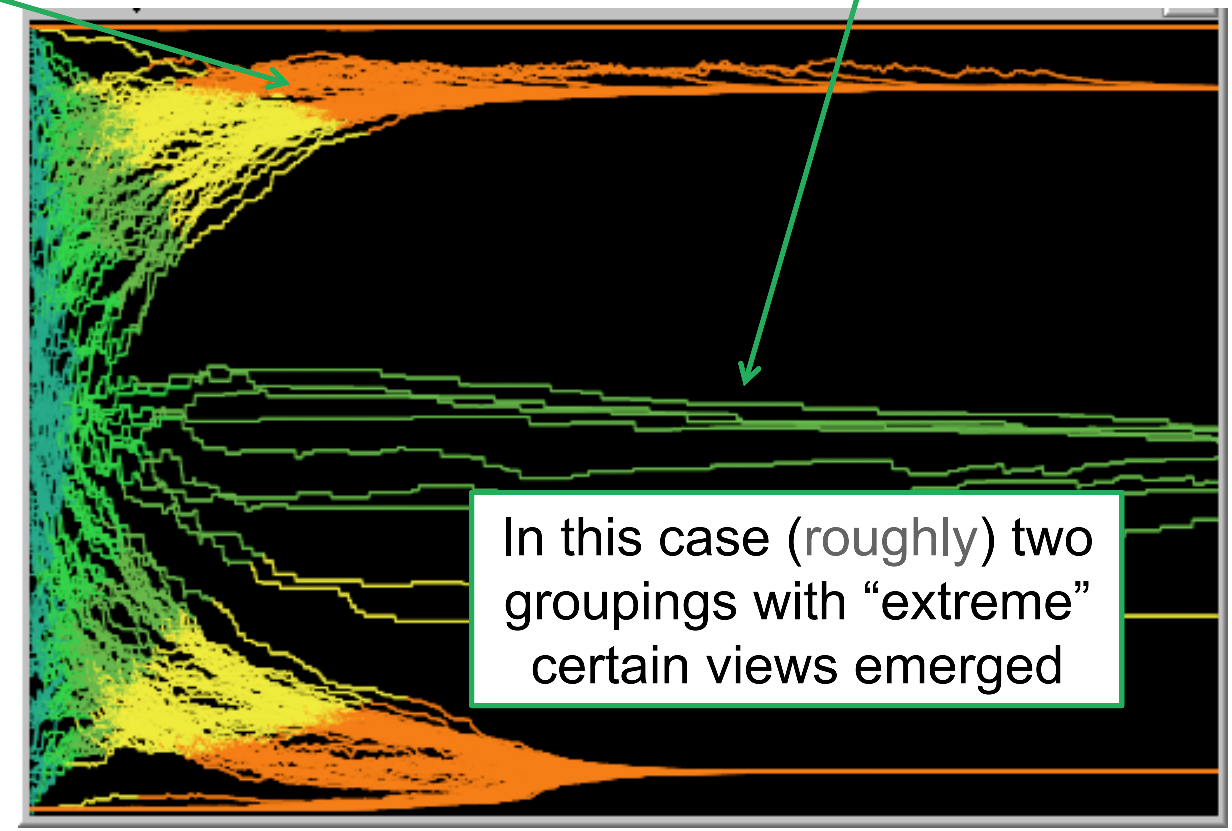
- Agents all have different levels of:
  - agreement on an issue, represented by a number -1 to 1
  - uncertainty about their opinion, represented by a number from 0 to 2
- Each iteration one (randomly picked) agent is randomly paired with another
- That other influences their opinion and uncertainty, but only if the other's opinion is sufficiently close to their own (difference is less than their uncertainty)
- There are some "extremists" who are divided between those with opinion 1 and -1 initially
- And "moderates" who have a random opinion initially
- This is a simple version of an existing model (see Info)
- There are many, many variants of these!

An example of what happens

The COLOUR of each is their level of uncertainty, from blue (maximally uncertain) to red (minimally uncertain)

Each line shows the "trajectory" of a single agent

The vertical scale represents the opinion of each, from -1 up to 1



In this case (roughly) two groupings with "extreme" certain views emerged

(Simulation) time is this axis

Manchester Metropolitan University

The Consensus Simulation

- Load the "opinion change.nlogo" simulation
- "prop-of-extremists" is the proportion of extremists in the initial population
- "uncert-of-moderates" is the initial uncertainty of the moderates (initial uncertainty of extremists is fixed at 0.05)
- "speed-uncertainty-change" is how much an agent's uncertainty is changed if influenced by another (opinion is always changed 5%)
- Play with the settings, run the simulations, see how many qualitatively different kinds of outcome there are and under what conditions they tend to occur

Procedures

- To organise code better, you can ‘bunch’ a whole lot of commands and give they a label you decide upon
- So they you can use this label and NetLogo will know this means to do the whole lot of commands it was defined with
- You can progressively define such labels using other labels etc., building up your own vocabulary of powerful commands

Go to the Code Tab and browse down to the bottom of the code

Between 'to' and 'end' there are sets of commands with a given label, each label can then be used elsewhere

```
to position-agent [tck]
  ;; move the turtle to the position corresponding to its opin
  ;; values need scaling to fit exactly on current world
  setxy
    (max-pxcor - min-pxcor) * (ceiling (tck / num-of-agents))
    0.5 * (opinion + 1) * (max-pycor - min-pycor) + min-pycor
end

to colour-agent
  ;; make the turtles (and hence their lines) a colour corresp
  ;; from 0 up to the max (the initial uncertainty of moderate
  ;; set up as a global variable
  let pos round (0.49 + 5 * uncertainty / uncert-of-moderates)
  if pos > length uncertainty-colours [set pos length uncertai
  set color item round (0.49 + 5 * uncertainty / uncert-of-mod
end

to plot-agent
  ;; once simulation is going, plotting involves (re)colouring
  colour-agent
  position-agent ticks
end
```


Manchester Metropolitan University

Different kinds of procedure

- Since the context of commands matters, *whether the commands are being done* ***within the context*** *of an agent, the observer, a patch (or even a link)* …
- …it is useful to keep track of which procedure (or chunk of code) is within which kind of context
- Some primitives and variables can only be used within an agent (turtle) context, others only within a patch context and others only within the observer context, etc.

Manchester Metropolitan University

Some Global Procedures in the code

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; global procedures ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;

to setup
  ;; note that parameter names have suffixes of the paramter name '

  clear-all

  ;; later used in colouring the plots
  set uncertainty-colours [red orange yellow green lime turquoise]
  set uncert-of-extremists 0.05

  ;; calculations as to the size of the various sub-populations
  let num-extremists round num-of-agents * prop-of-extremists
  let num-moderates num-of-agents - num-extremists
  let num-upper-extremists num-extremists / 2
  let num-lower-extremists num-extremists - n

  ;; create moderates with random opinions
  create-turtles num-moderates [
    set uncertainty uncert-of-moderates
    set opinion (random-float 2) - 1
  ]
```

The “**setup**” and “**go**” procedures are within the observer (the global) context

But some chunks of code are within the agent context

Agent procedures

```
;;;;;;;;;;;;;;;;;;;;;;;;
;;; agent procedures ;;;
;;;;;;;;;;;;;;;;;;;;;;;;

to be-influenced-from [oth]
  ;; only influenced if difference between opinion and other's opinion i
  ;; my uncertainty...
  if abs (opinion - [opinion] of oth) < uncertainty [
    ;; ...in which case shift my opinion and uncertainty towards other's
    set opinion 0.95 * opinion + 0.05 * [opinion] of oth
    set uncertainty (1 - speed-uncertainty-change) * uncertainty
                    + speed-uncertainty-change * [uncertainty] of oth
  ]
end

to initialise-agent-displ
  ;; what each turtle does in setup
  hide-turtle
  penup
  colour-agent
  position-agent 0
  pendown
end
```

A lot of the other procedures are within the agent context

Local agent variables are only usable within the agent context

Some primitives are only usable within the agent context

Things to try in this simulation

- Can you work out when one gets one, two or more groups out of the process?
- What might one add to help understand what is happening in the simulation?
- What happens if you change code in the procedure: "**be-influenced-from**"?
- What happens if everyone is only influenced by the nearest other?

If you have finished…

- … try playing with some of the other simulations
- Goto http://cfpm.org/eaepe/
- And read the document: “Other models to play with.pdf”
- And follow its instructions
- Each time: read the “Info Tab”, play with the simulation, look at the code (but only expect to understand small bits of the code as yet)

Manchester Metropolitan University

Conclusions of Session I

I hope you have got a little bit of an idea concerning agent-based simulation :

- What it is
- How it is programmed
- Its flexibility
- That the outcomes can be surprising
- That it can be hard to understand how one's own model works
- The space of possible models is huge

What we have not talked about is how to apply this to policy issues, and how to check if a model is (in any sense) correct!

The End

The Centre for Policy Modelling:
http://cfpm.org

Introduction to ABS and Policy Modelling materials:
http://cfpm.org/eaepe