2-Day Introduction to Agent-Based Modelling

Day 1: Session 1

Introduction, commands, loops, conditionals
Welcome

It is organised and run by

– Bruce Edmonds from the Centre for Policy Modelling at the Manchester Metropolitan University
– and GertJan Hofstede from the XXX of Wageningen University
– With help from Gary Polhill of the James Hutton Institute in Aberdeen.
Course Aims

• To introduce you to programming and debugging agent-based modelling (ABM) through the NetLogo programming language
• To give you an idea of some of the things that ABM can represent and how this is done
• To give you an insight into the ABM way of thinking about social phenomena
• To help you understand the process of designing, programming and using an ABM
• To show you some examples of how ABM has been used in the social science literature and research
What it will not do…

• Is make you a full agent-based programmer
• An expert at NetLogo
• Able to rush off and immediately write a ‘blockbuster’ ABM paper
• Be able to immediately program what you have in mind to model

Sorry! These take time and some patience to achieve, but we hope to have started you on the road in these directions.
Course Style is…

• **Relaxed!** Please feel free to experiment, deviate from the course material, ask questions from the helpers, generally making the course maximally useful for you
• We will have quite a range of previous computer programming experience among the participants so it is inevitable that some will find some of this a bit slow (*if so experiment and extend your knowledge, making use of the helpers around and suggestions for extension*), and some will find parts a bit fast (*if so ask for lots of help from the helpers and simply don’t worry about it but go at your own pace*)
• Each session will start with an example model, with some explanation/directions from the front, but with suggestions for additional things to do within each model, a model with the additions is also provided
• It is in the fundamental nature of programming that not everything is obvious – *even when you have read the manual* – so do ask a helper when you get stuck
Schedule, Course Material, etc....

Are ALL freely available at:
http://cfpm.org/simulationcourse
(there will be a minimum of paper distributed)
That site has pointers to:
• The schedule which has pointers to:
  – The example models
  – These slides
  – Further material on the web
• The Facebook group for the course, which you can join and post discussion points as you go along,: flattery, useful links, etc. (linked to from the course website, you have to go to the page and request an invite!)
A Classic Example of an Agent-Based Model: 
*Schelling’s Segregation Model*


**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

*Segregation can result from wanting only a few neighbours of a like colour*
Staring the first NetLogo Model

- If you have not installed NetLogo, please ask for help doing this now
- Download and run the "1-commands-begin.nlogo" model
- All example models are linked from the session page on the web, along with all other material for that session
NetLogo – Interface Panel
NetLogo – Interface Panel
NetLogo – Interface Panel

- **Command Buttons**
  - setup
  - step

- **Parameter Slider**
  - num-obstacles: 200
NetLogo – Interface Panel

- **Command Buttons**
- **Parameter Slider**
- **Typed Direct Commands**
NetLogo – Interface Panel

Command Buttons

Parameter Slider

Typed Direct Commands

Text Output
NetLogo – Interface Panel

- Command Buttons
- Parameter Slider
- Speed Control
- Typed Direct Commands
- Text Output
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
Typing in Commands

Press “setup” to initialise world
Typing in Commands

Press “setup” to initialise world

World with different colour patches
Typing in Commands

Press “setup” to initialise world

World with different colour patches

An agent!
Typing in Commands

Press “setup” to initialise world

World with different colour patches

An agent!

Type commands in here as follows…
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
Inspecting Patches and Agents

Right-click (or ctrl click) on a patch, then "inspect" that patch.
Inspecting Patches and Agents
Inspecting Patches and Agents
Inspecting Patches and Agents
Inspecting Patches and Agents

Magnified View

Properties of patch

Type commands to patch here, e.g. `set pcolor red`
Inspecting Patches and Agents

Click on little “x” in corner to get rid of inspector

Magnified View

Properties of patch

Type commands to patch here, e.g. set pcolor red
Inspecting Patches and Agents

Click on little “x” in corner to get rid of inspector
Inspecting Patches and Agents
Inspecting Patches and Agents

Right-click (or ctrl click) on an agent, then “inspect” that agent.
Inspecting Patches and Agents
Inspecting Patches and Agents

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
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”
Using “ask”

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]
Running a simulation (the hard way!)

- 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.
Running a simulation \textit{(the hard way!)}

1. Move the slider to change parameter

- Each time “\textit{step}” is pressed the procedure called “\textit{go}” is caused to run – this is a list of commands, a \textit{program}.
- We will now look at this.
Running a simulation (the hard way!)

2. Press “setup” to initialise world

• 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.
Running a simulation *(the hard way!)*

2. Press “setup” to initialise world

3. Press “step” to make the program run one time step

- 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.
Running a simulation \textbf{(the hard way!)}

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

```plaintext
;; 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 [agents agent]
;; only attribute is age, all agents automatically have the attribute of color and size

agents-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 poolor white
  
  ;; make one patch red - the target
  ask one-of patches [set poolor red]

  ;; creates one agent
  create-agents 1
    set size 1.5
    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

  ;; stuff that happens to any person
```
The Program Code

Click on the “Code” tab to see the program

```
;; 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 [agents agent]

;; only attribute is age, all agents automatically have the attribute of color and size
agents-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-agents 1 [ set size 1.1 ;; 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

    ;; stuff that happens to any person
```
The Program Code

This text is the program

```
;; 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 [agents agent]
;; only attribute is age, all agents automatically have the attribute of color and size
agents-own [finished]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Next we have the procedure to initialize 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 poolor white ]
  ;; make one patch red - the target.
  ask one-of patches [ set poolor red ]
  ;; creates one agent
  create-agents 1 [ set size 1.1 ;; 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

;; stuff that happens to any person
```
The Program Code

This text is the program

It has different parts

```plaintext
;; 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 [agents agent]
;; only attribute is age, all agents automatically have the attribute of color and size
agents-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 poolor white ]

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

  ;; creates one agent
create-agents 1 []
    set size 1.1 ;; 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 initializes the simulation time system and graphs
and

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
```
The Program Code

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.
The Program Code

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
The Program Code

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
Parts of the Code

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
Parts of the Code

```plaintext
; 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
end

;; this progresses the
```

Everything between “to” and “end” defines what “go” means
"ask agents" means to ask (all) agents to do some code, one after the other.

```
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
  end
  ;; this progresses the
```
Parts of the Code

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

What it is asking them all to do is between the square brackets “[ … ]”

```
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
    end
```

;; this progresses the
Parts of the Code

“if” statements are 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
end
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

end

;;; this progresses the
Parts of the Code

All the square brackets inside each other can be confusing, if you double-click \textit{just outside} a bracket, it shows what is in side between it and the matching bracket.
Parts of the Code

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.
All the square brackets inside each other can be confusing, if you double-click just outside a bracket, it shows what is inside between it and the matching bracket.
To change the program...

```plaintext
set finished? false ;; not finished to begin with

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

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 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 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 tick counter by one
end
```
To change the program...

Click within the text and type!
To change the program...

Click within the text and type!

*type the following:*

```plaintext
;; my bit!
if random-float 1 < 0.05 [lt 90]
```
To change the program...

Click within the text and type!

*type the following:*

`; ; my bit!`

if random-float 1 < 0.05 [lt 90]
To change the program...

You can press “Check” to see if you got the syntax of everything right!

Click within the text and type!

**type the following:**

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

Click within the text and type!

**type the following:**

```
;; my bit!
if random-float 1 < 0.05 [lt 90]
```
The Information Tab

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?
The Information Tab

Click on the “Info” tab to see a description of the model (or whatever the programmer has written, if anything!)
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
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!)
Adding a button and running the code (the fast way!)
Adding a button and running the code (the fast way!)

Click on the “Interface” tab to get back to the main view.
Adding a button and running the code (the fast way!)

Right-Click some empty space and choose "button"
Adding a button and running the code (the fast way!)

Right-Click some empty space and choose “button”
Adding a button and running the code (the fast way!)

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

Type the text “go” here and then check (to on) the “forever” switch then **OK**
Adding a button and running the code (the fast way!)
Adding a button and running the code (the fast way!)

Now when you press the “go” button it will keep doing doing the “go” procedure forever (until you “unpress” it)
Adding a button and running the code for only 10 steps
Adding a button and running the code for only 10 steps

Right-Click some different empty space and choose "button"
Adding a button and running the code for only 10 steps
Adding a button and running the code for only 10 steps

Type the text “repeat 10 [go]” here
Adding a button and running the code for only 10 steps

Type the text “repeat 10 [go]” here

Type the text “10 steps” here and then “OK”
Adding a button and running the code for only 10 steps
Adding a button and running the code for only 10 steps

Now when you press the “10 steps” button it will do the “go” procedure only 10 times.
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!)
Things to try…

Try to do the following:

• add a button to manually turn the agent right (using “rt 90”)

• add a slider for number of targets and add code to make this number of patches red

• try to change the “if” commands within the “go” procedure and see what happens

• add new “if” rules, for example to with a certain probability to turn left (using “if random-float 1 < 0.05 [.....]”)

• add a command within “setup” to place the agent at a random position at the start (using “setxy”, “random max-xcor” and random max-ycor”)
The End

2-Day Introduction to Agent-Based Modelling
http://cfpm.org/simulationcourse