2-Day Introduction to Agent-Based Modelling

Day 1: Session 1

Introduction, commands, loops, conditionals

Manchester Metropolitan University

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:

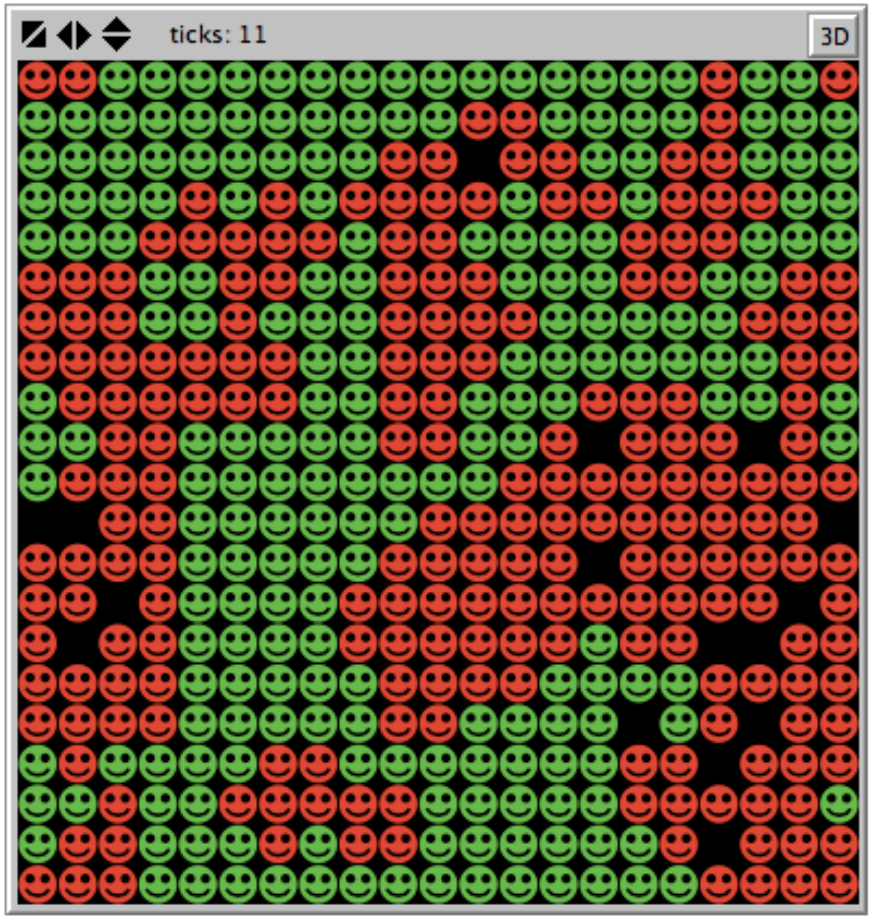
- The schedule which has pointers to:
  - The example models
  - These slides
  - Further material on the web
- The Facebook group for the course, which you can join and post discussion points as you go along,: flattery, useful links, etc. (linked to from the course website, you have to go to the page and request an invite!)

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

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

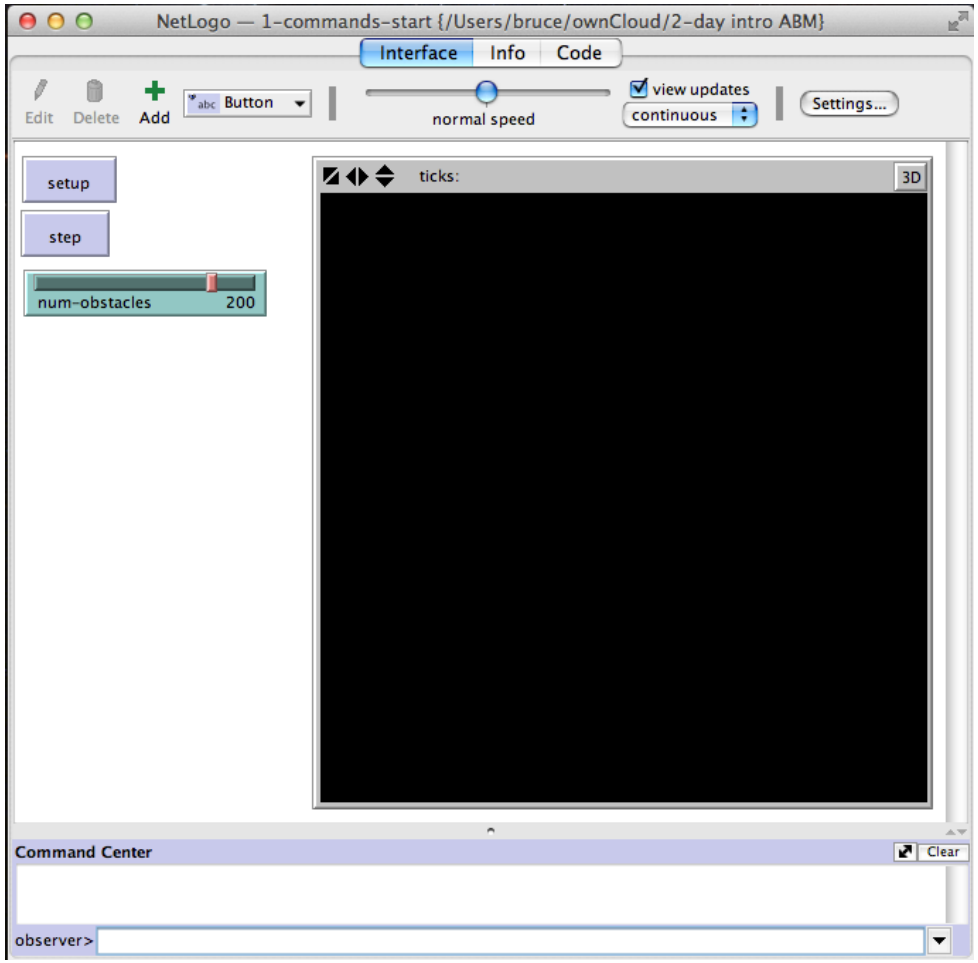
Segregation can result from wanting only a few neighbours of a like colour



Staring the first NetLogo Model

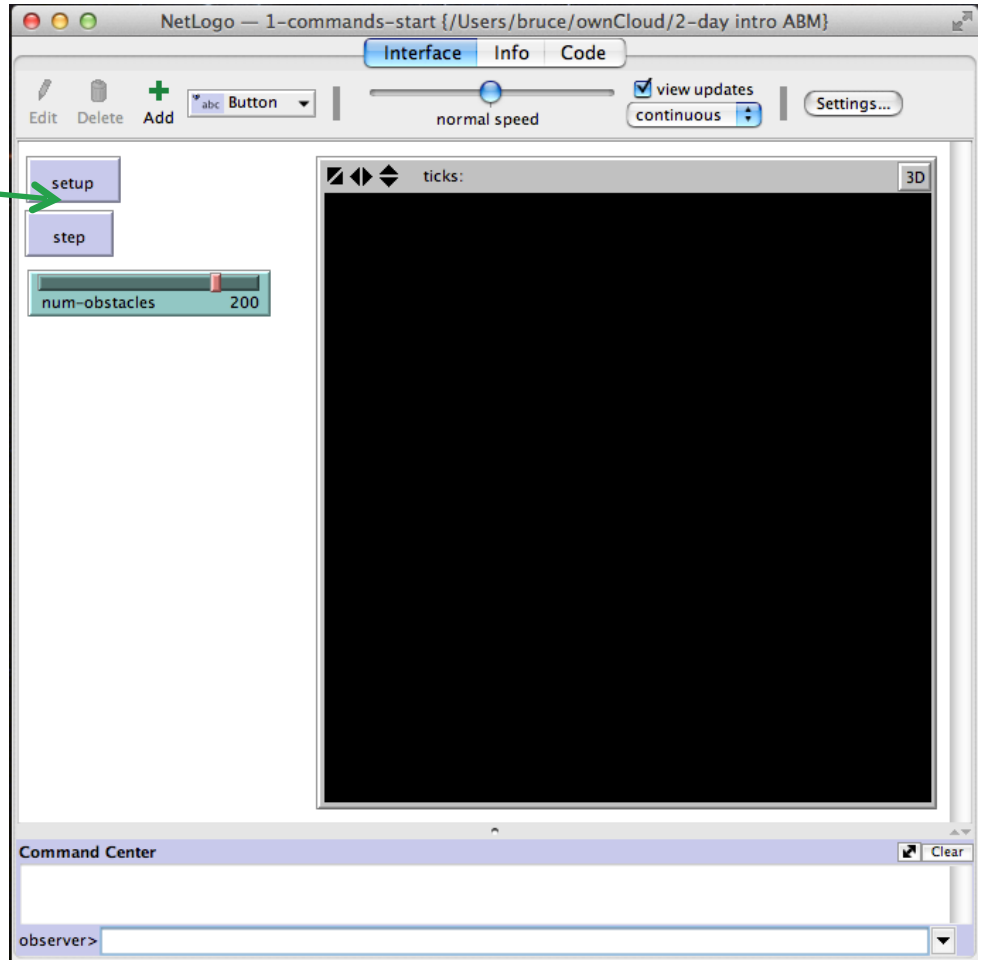
- If you have not installed NetLogo, please ask for help doing this now
- Download and run the "1-commands-begin.nlogo" model
- All example models are linked from the session page on the web, along with all other material for that session

NetLogo – Interface Panel



NetLogo – Interface Panel

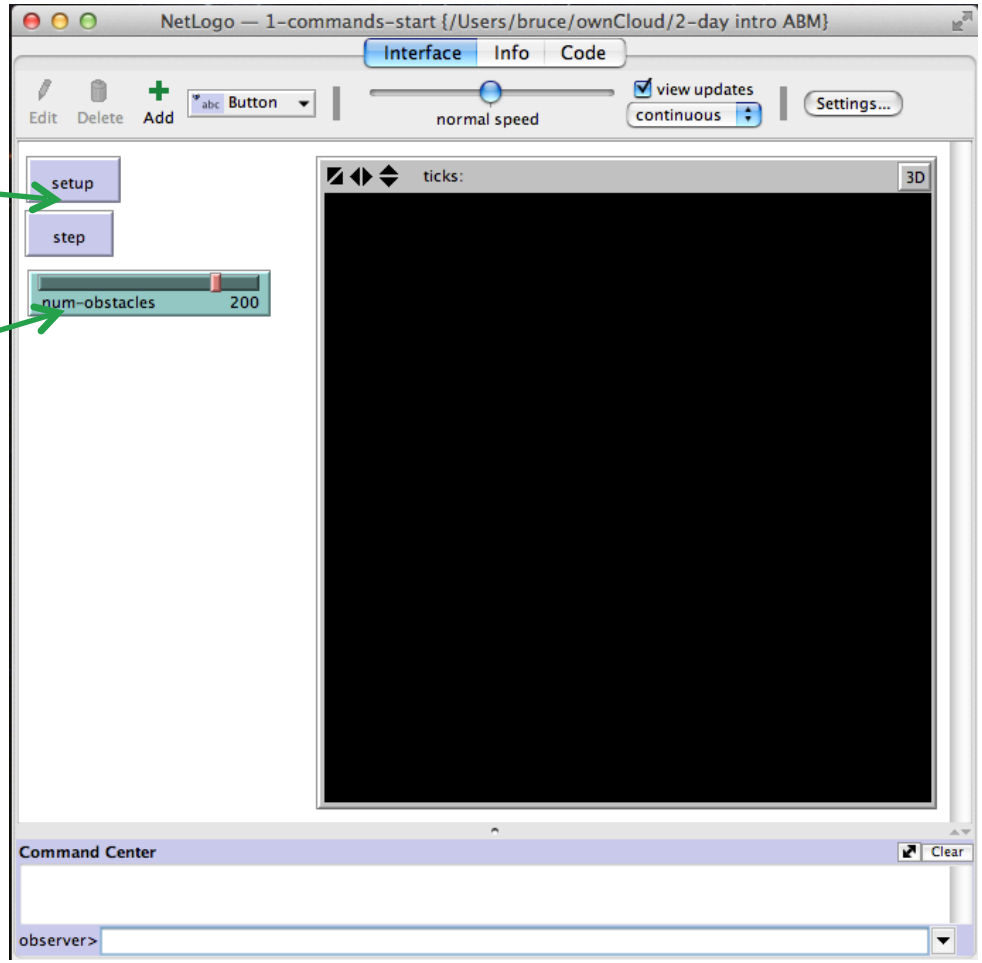
Command Buttons



NetLogo – Interface Panel

Command Buttons

Parameter Slider

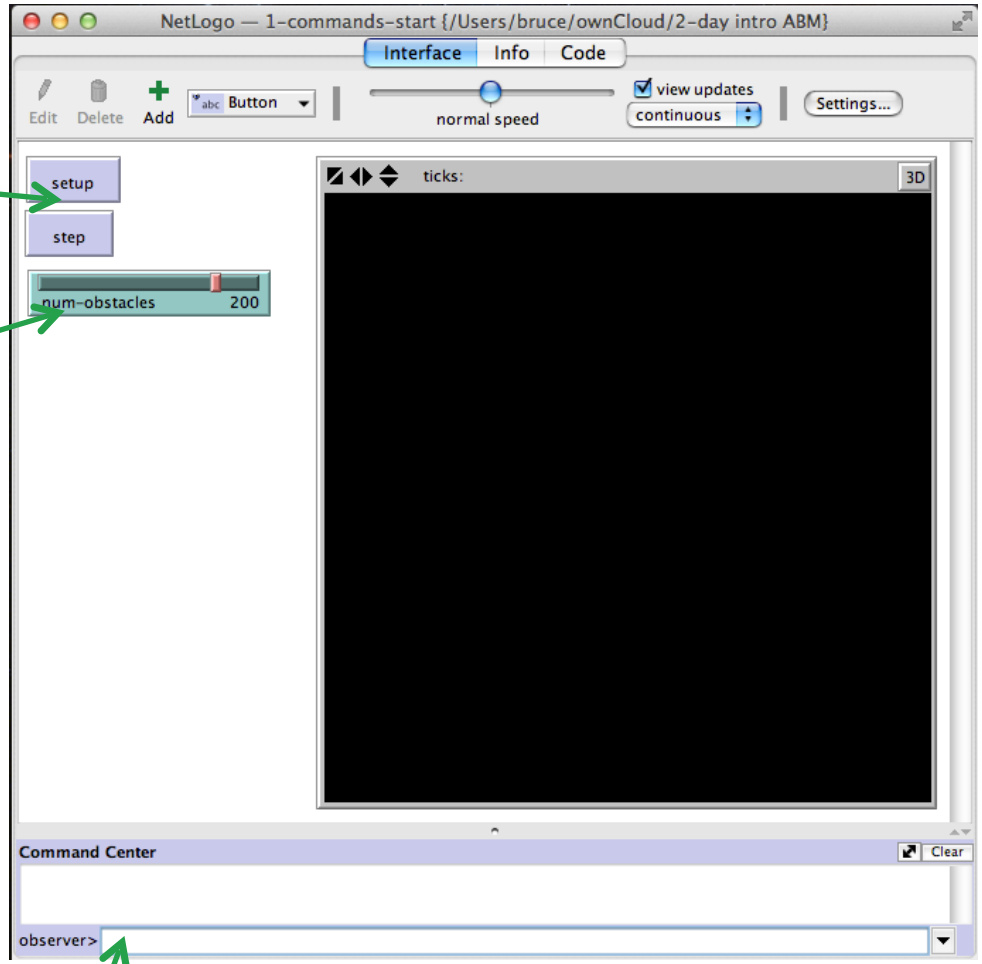


Manchester Metropolitan University

NetLogo – Interface Panel

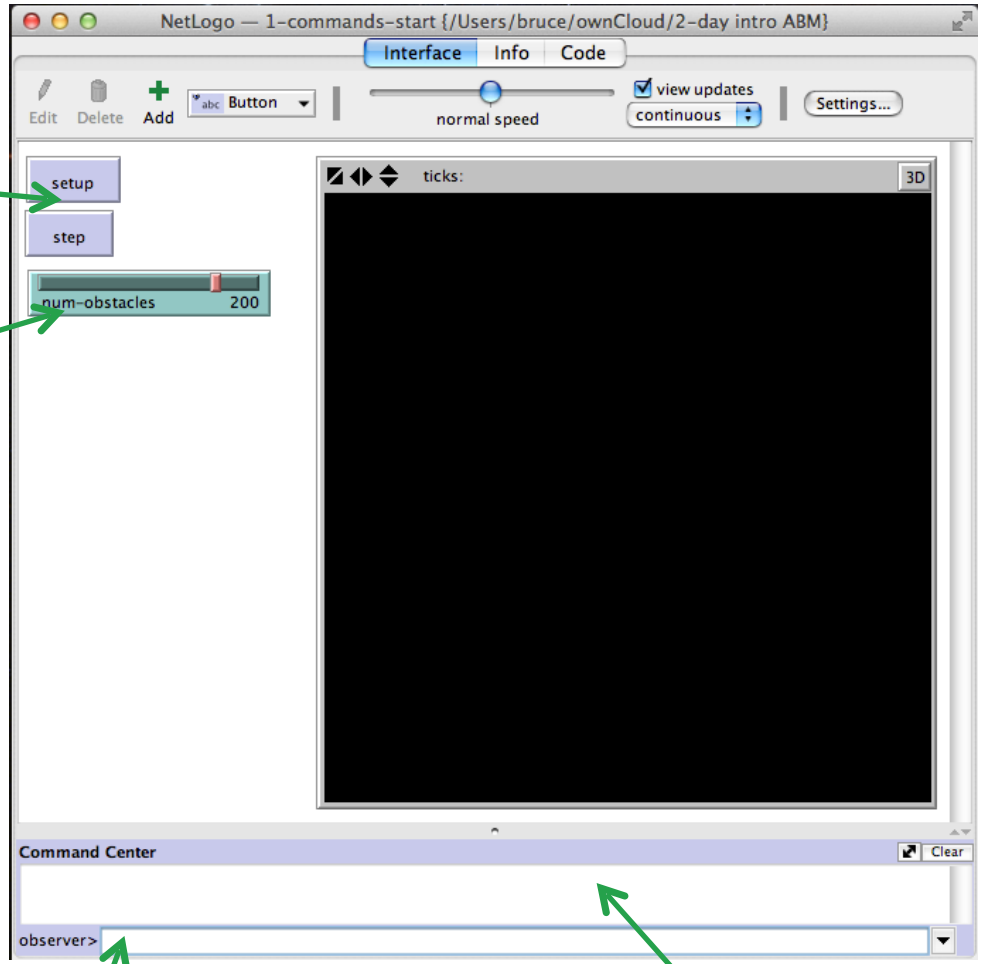
Command Buttons

Parameter Slider



Typed Direct Commands

NetLogo – Interface Panel



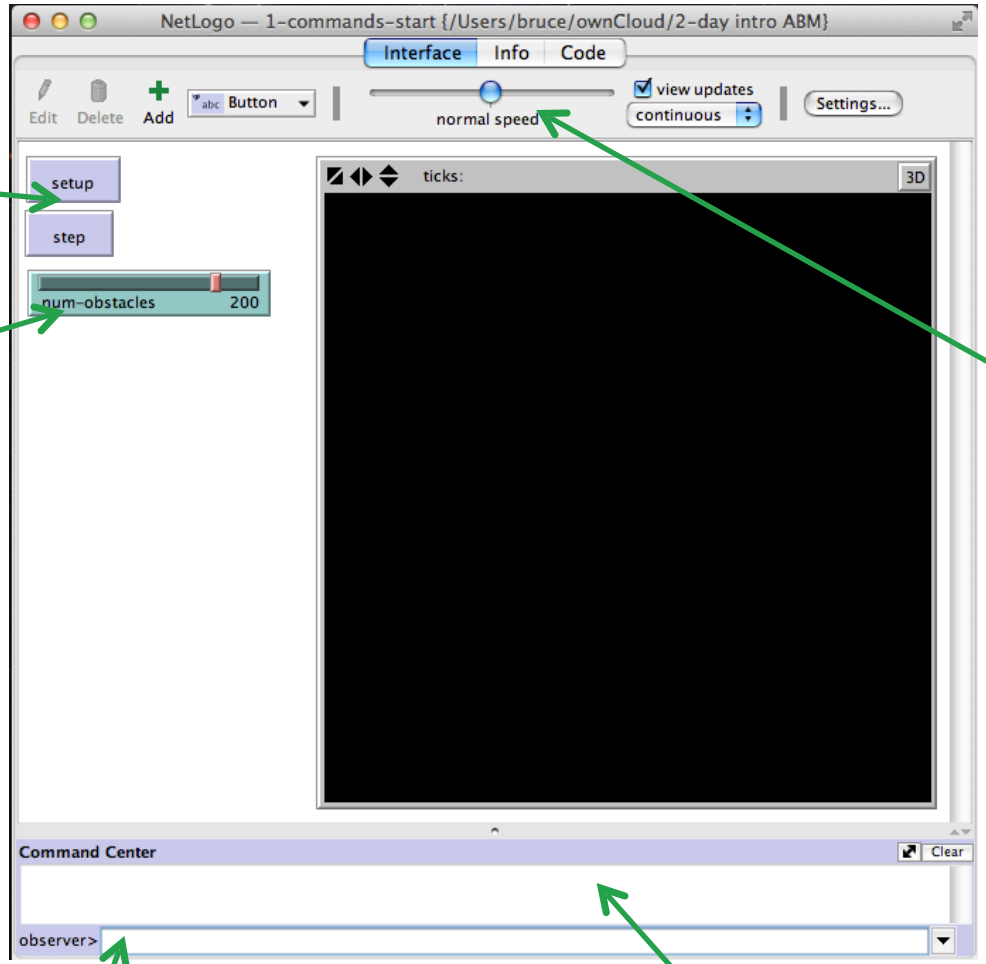
Command Buttons

Parameter Slider

Typed Direct Commands

Text Output

NetLogo – Interface Panel



Command Buttons

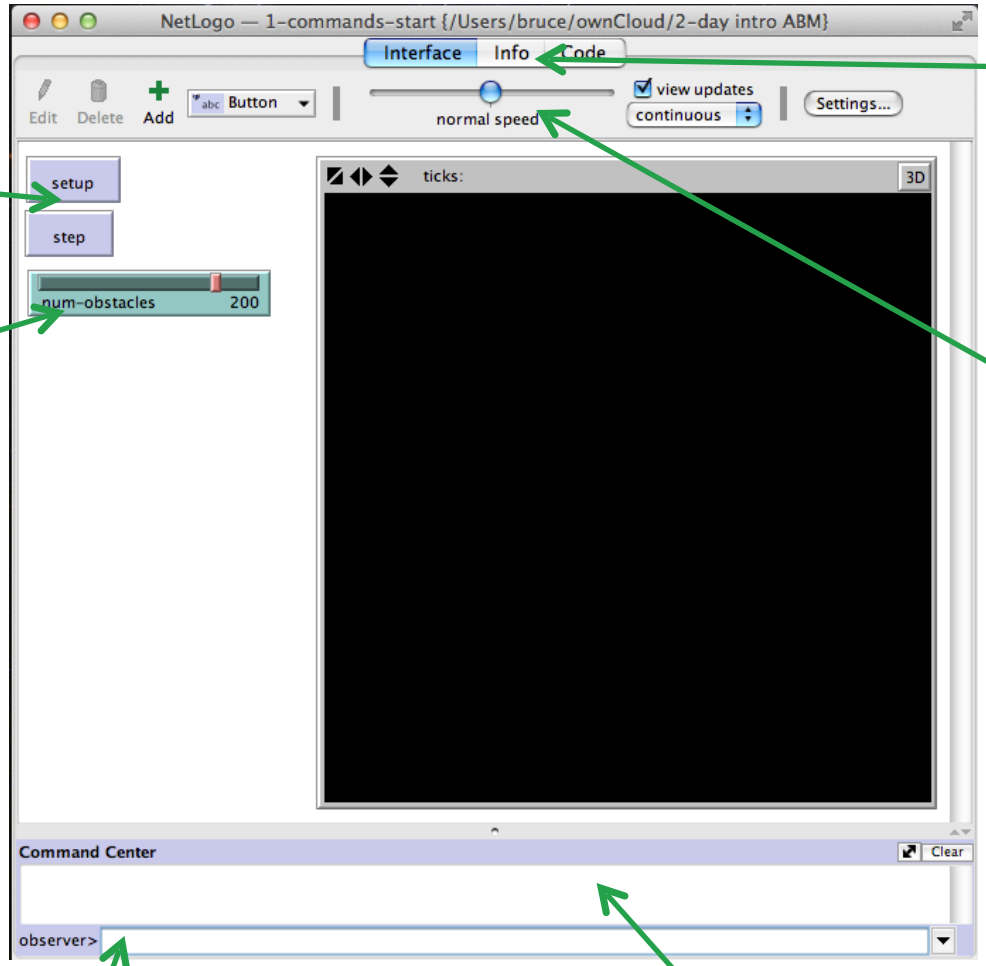
Parameter Slider

Speed Control

Typed Direct Commands

Text Output

NetLogo – Interface Panel



Command Buttons

Parameter Slider

Panel Selection (looks slightly different on Windows and Macs)

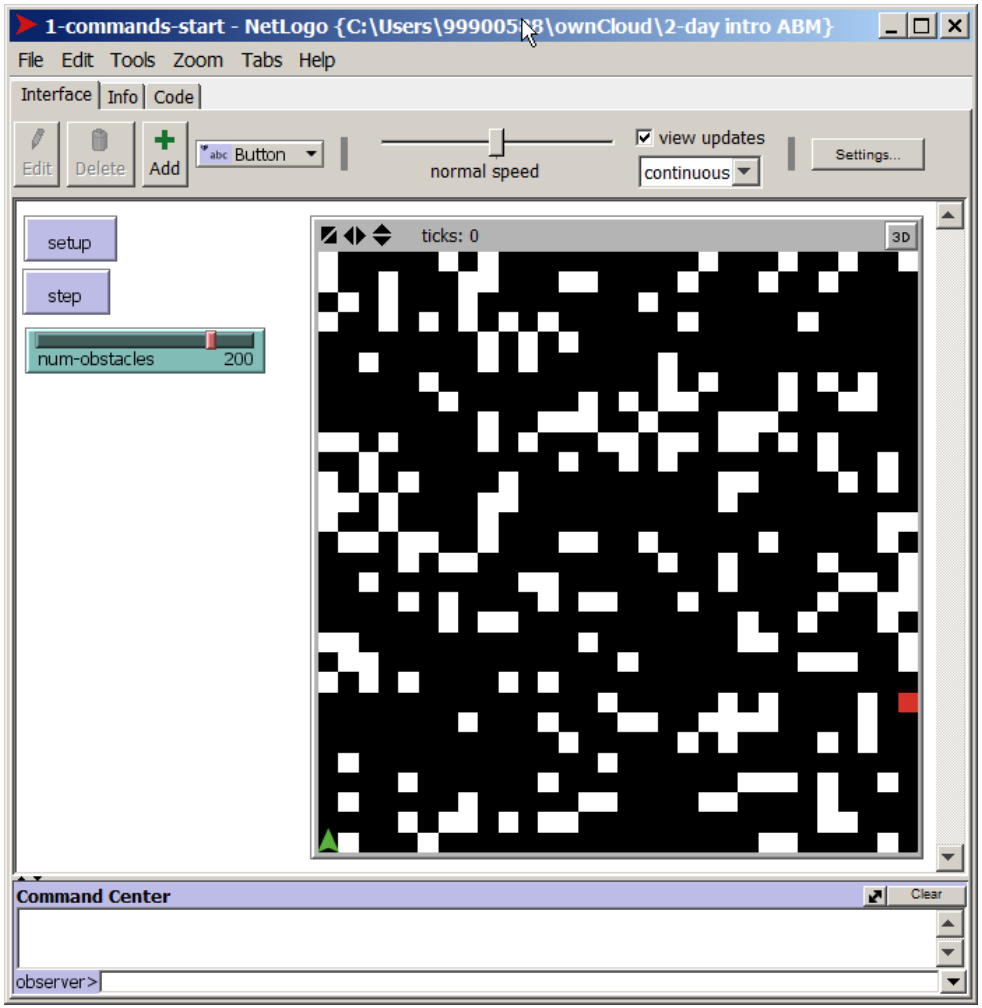
Speed Control

Typed Direct Commands

Text Output

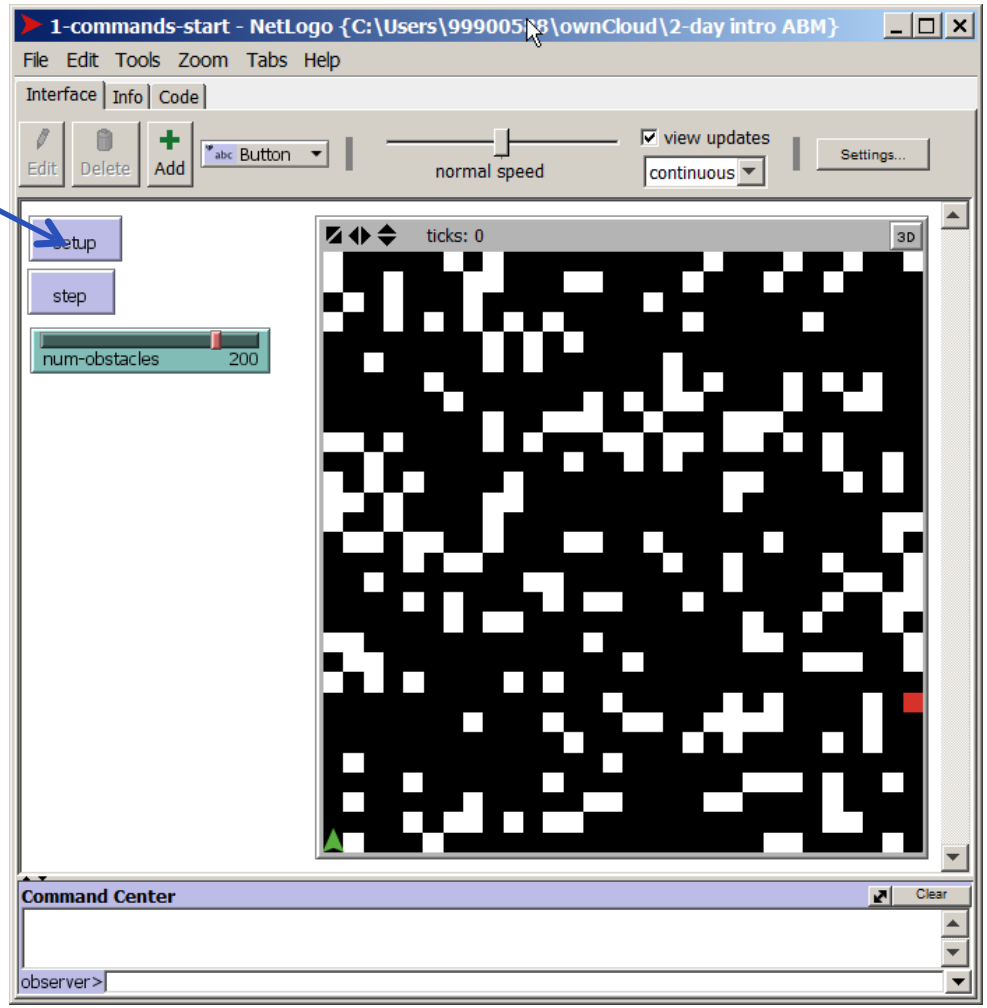
Manchester Metropolitan University

Typing in Commands



Typing in Commands

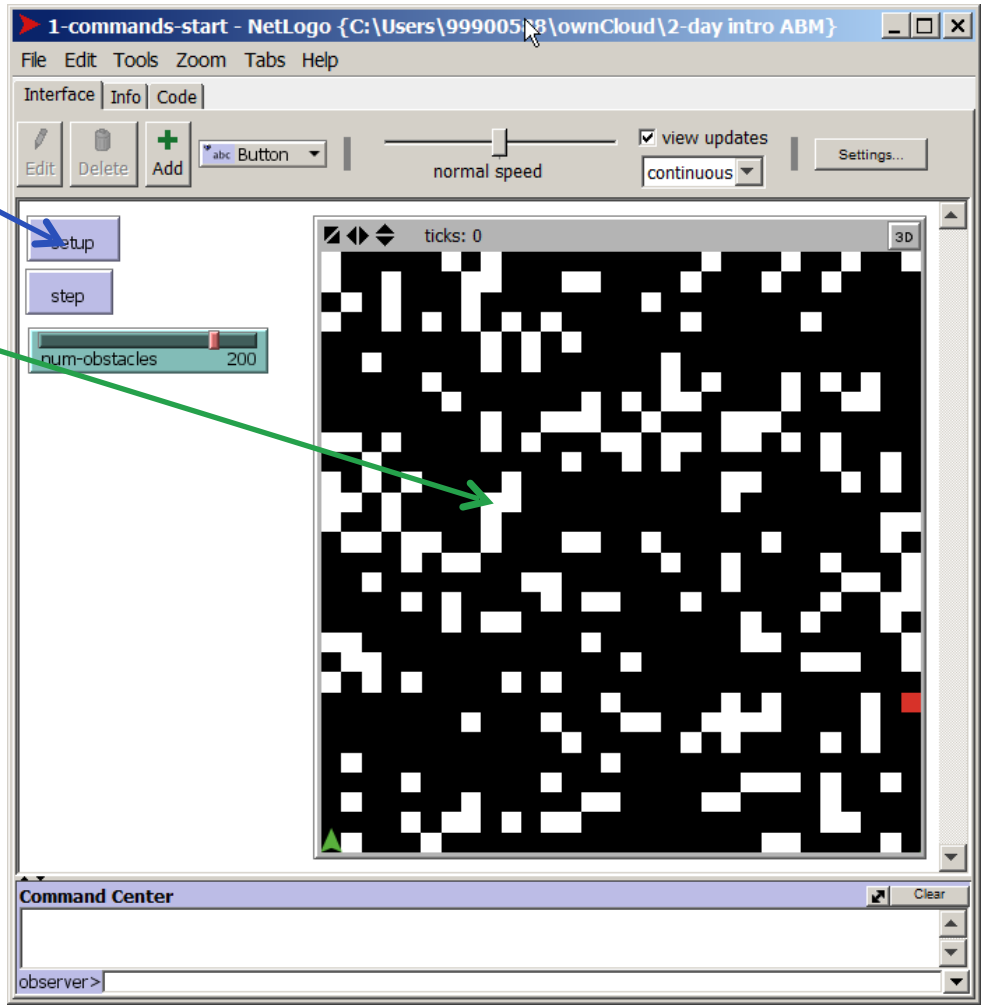
Press “setup” to initialise world



Typing in Commands

Press “setup” to initialise world

World with different colour patches

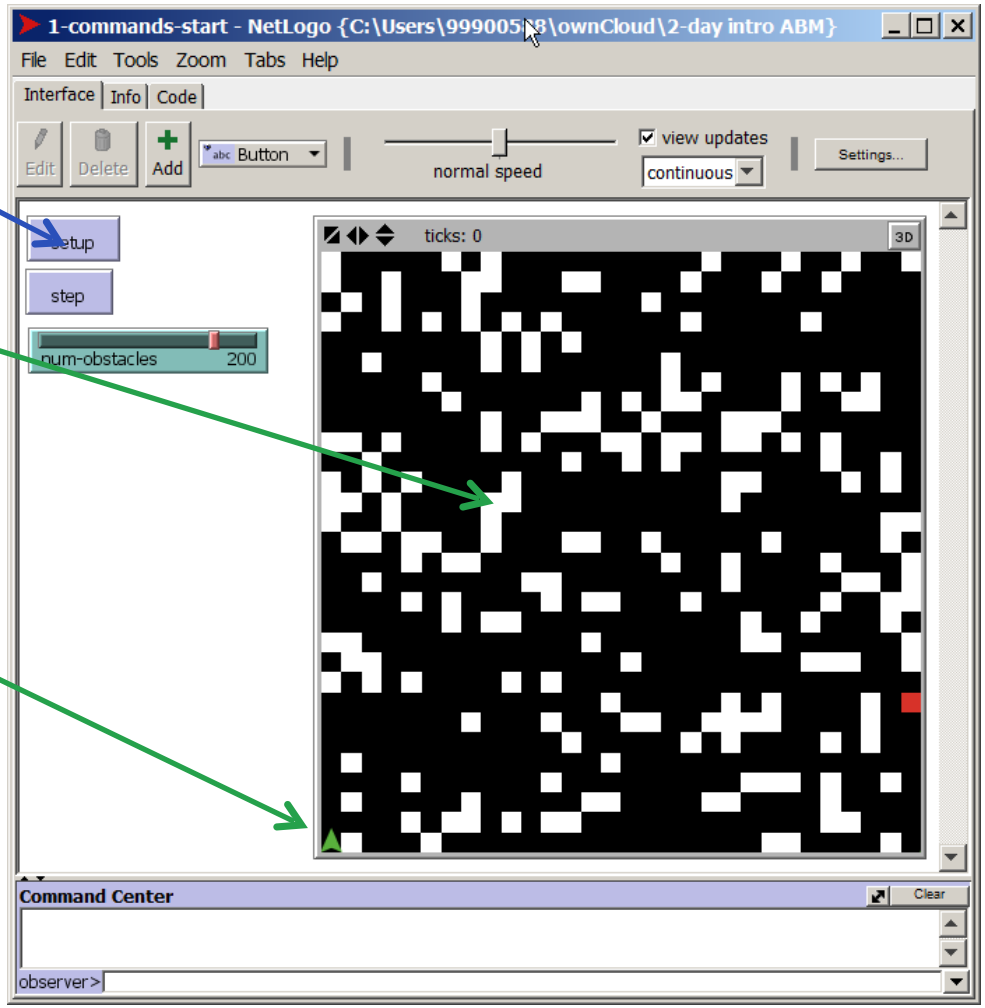


Typing in Commands

Press “setup” to initialise world

World with different colour patches

An agent!



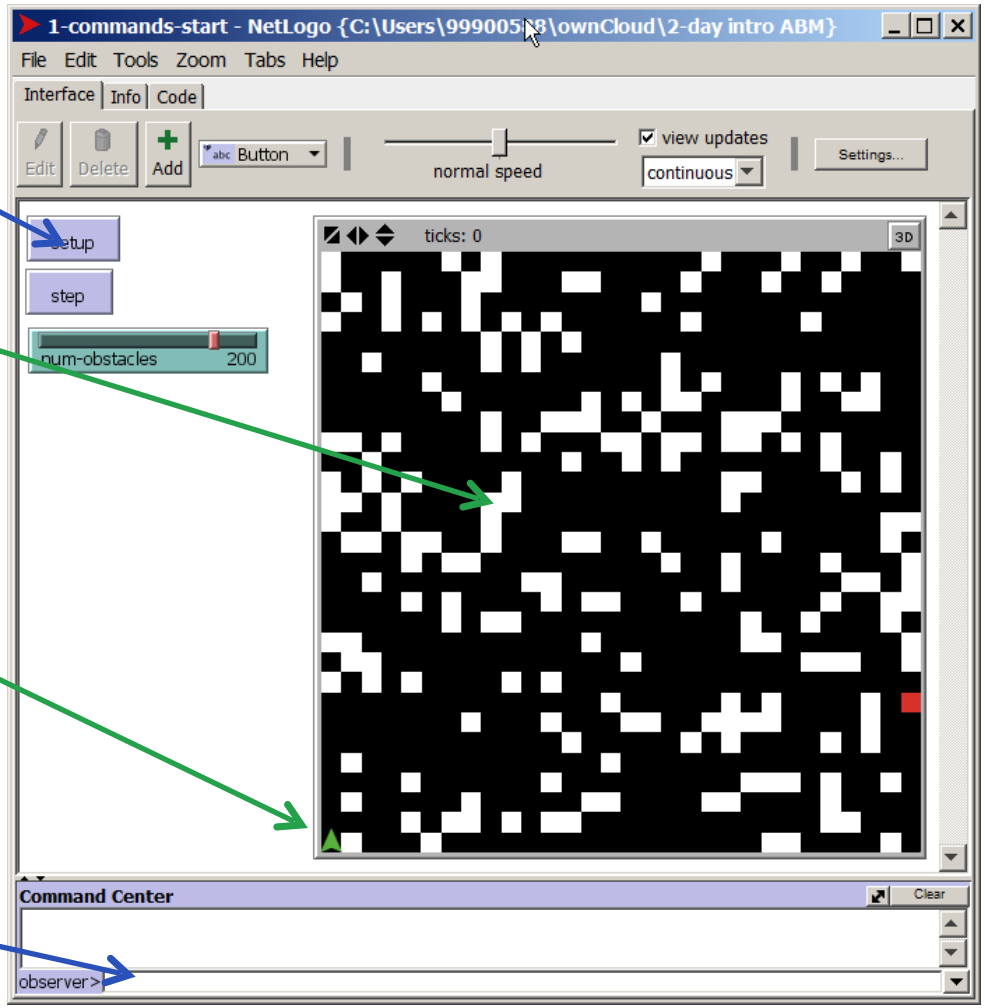
Typing in Commands

Press “setup” to initialise world

World with different colour patches

An agent!

Type commands in here as follows…



Manchester Metropolitan University

The command centre…

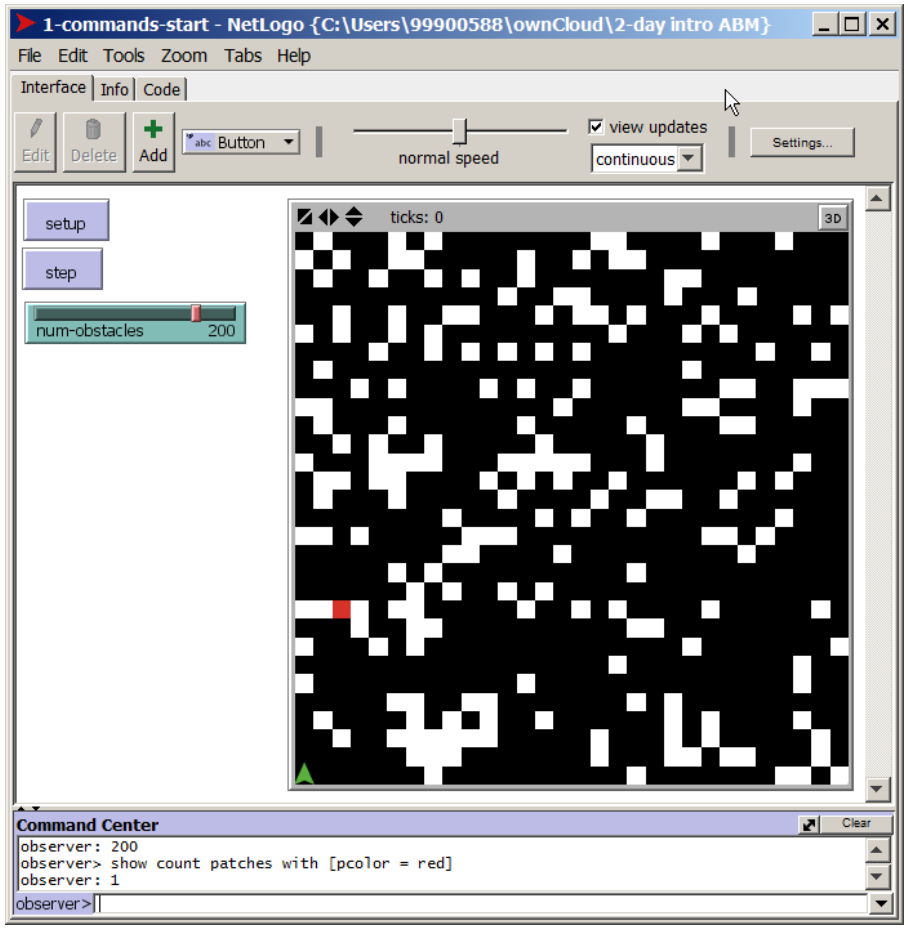
- "show" means show the result in the command centre

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]

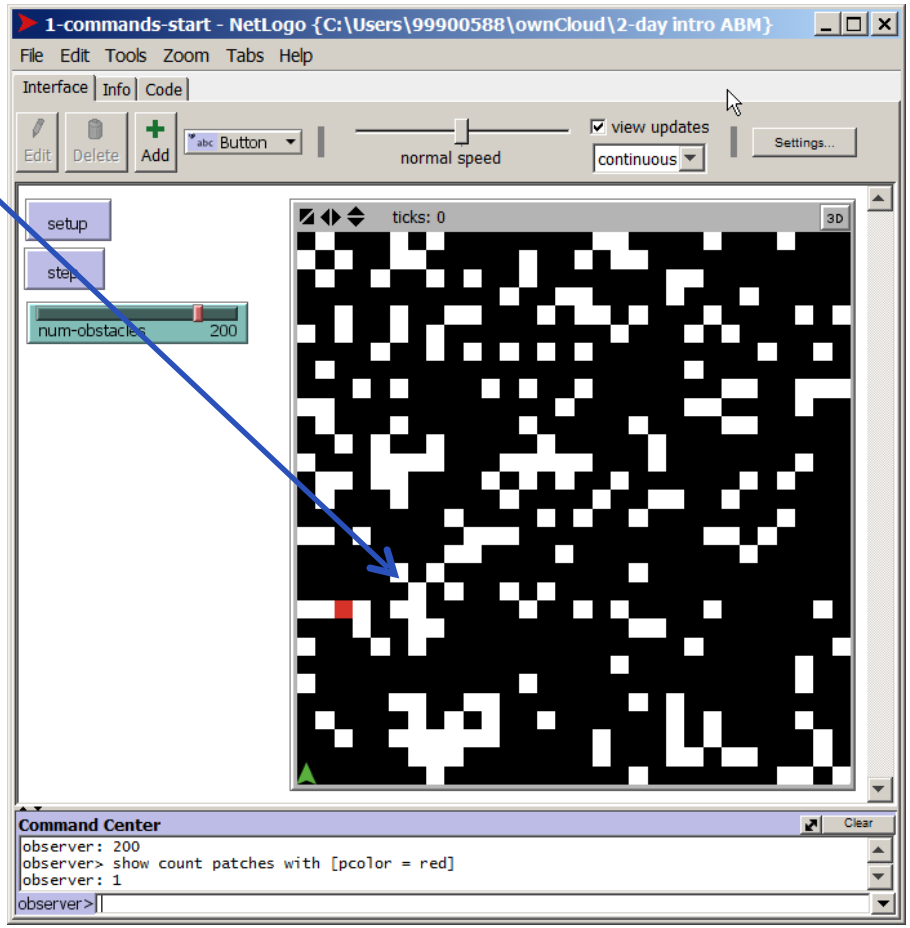
Anything typed into the command centre is from the "observer" point of view (yours!)

Inspecting Patches and Agents

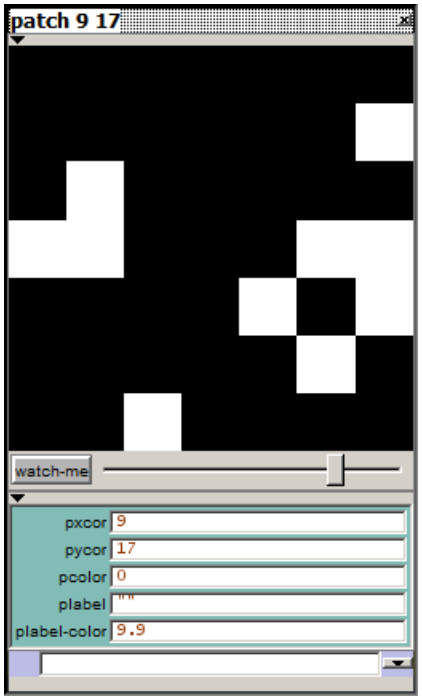


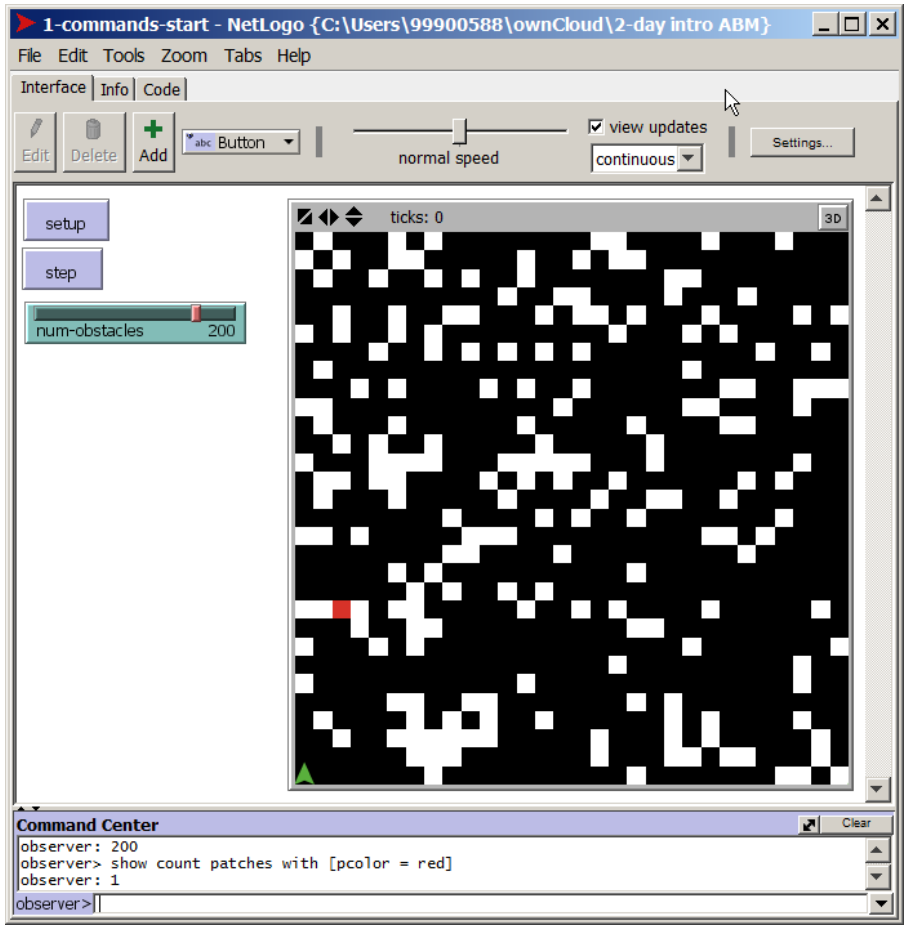
Inspecting Patches and Agents

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

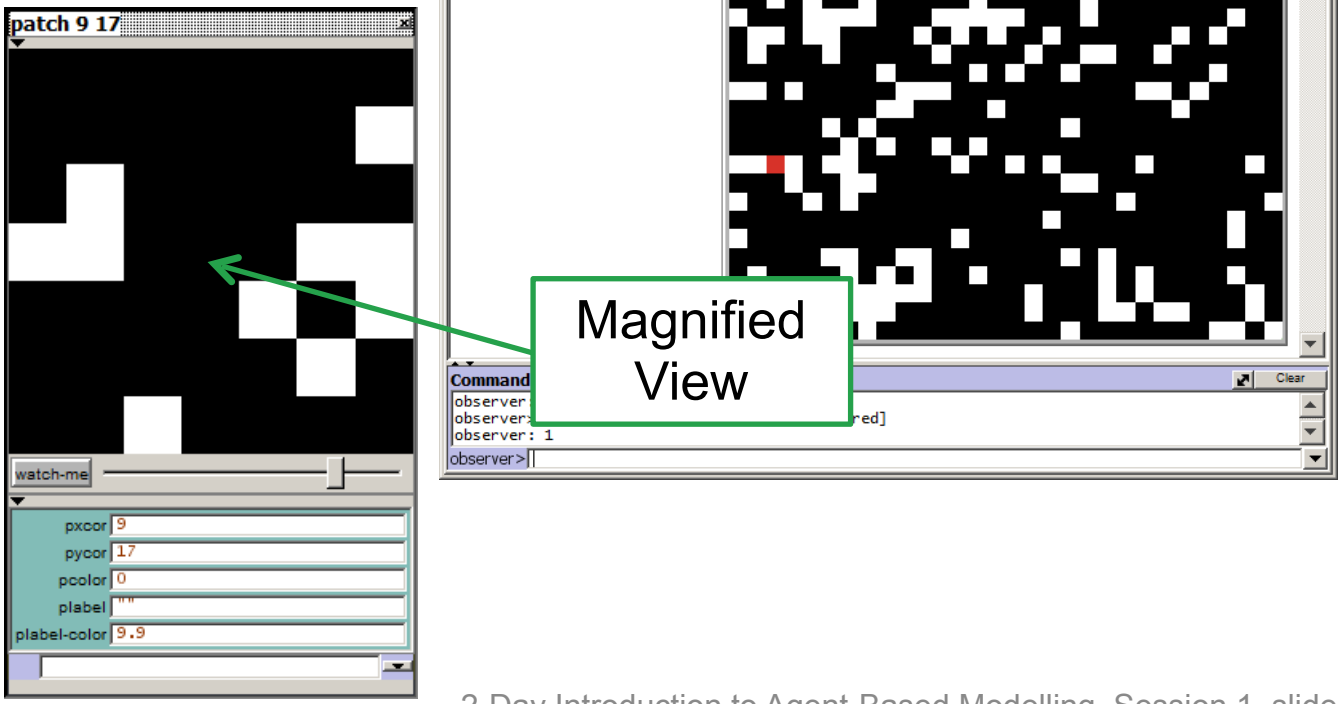


Inspecting Patches and Agents



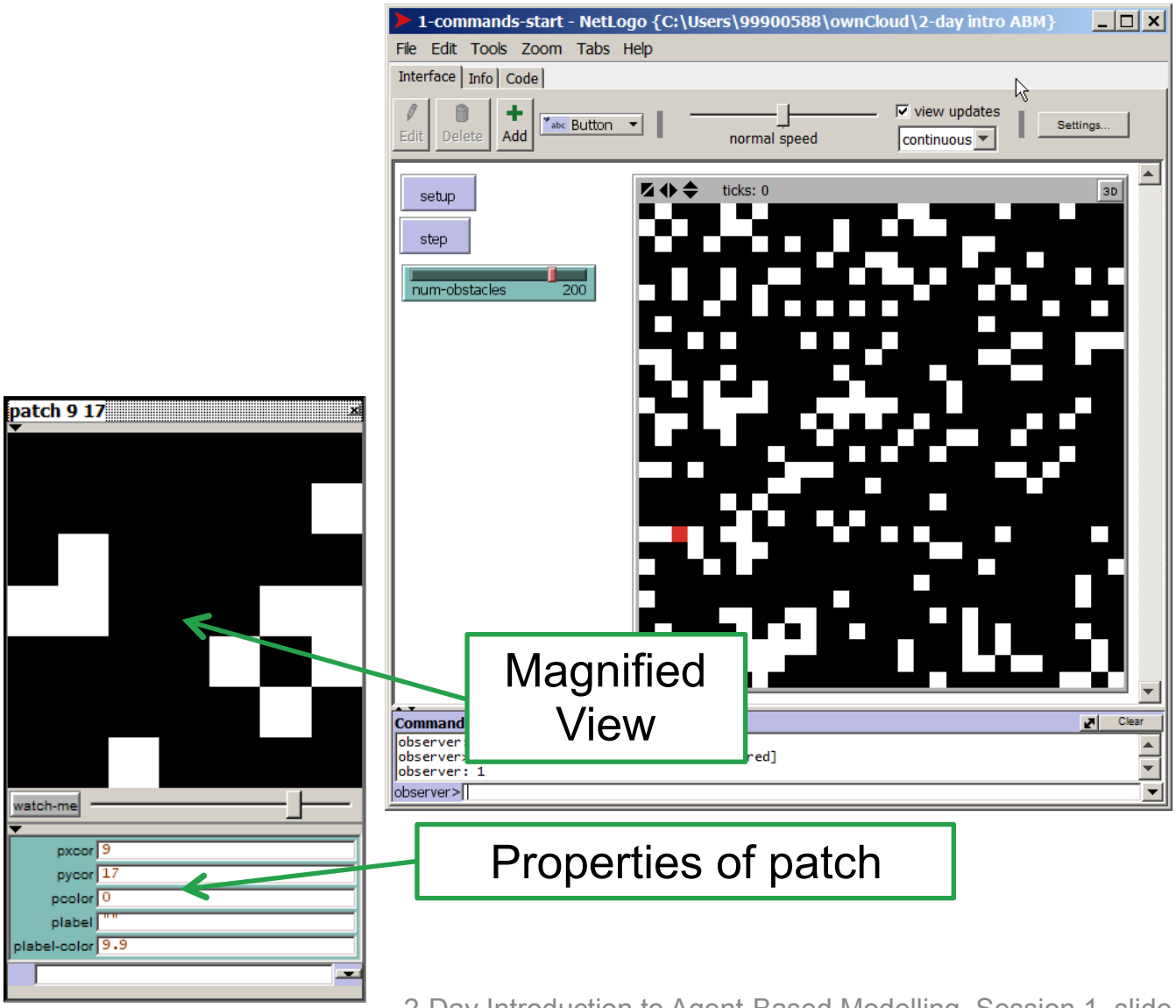


Inspecting Patches and Agents

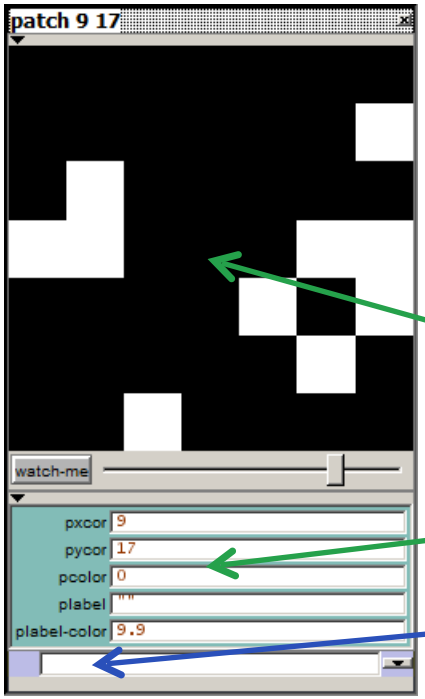


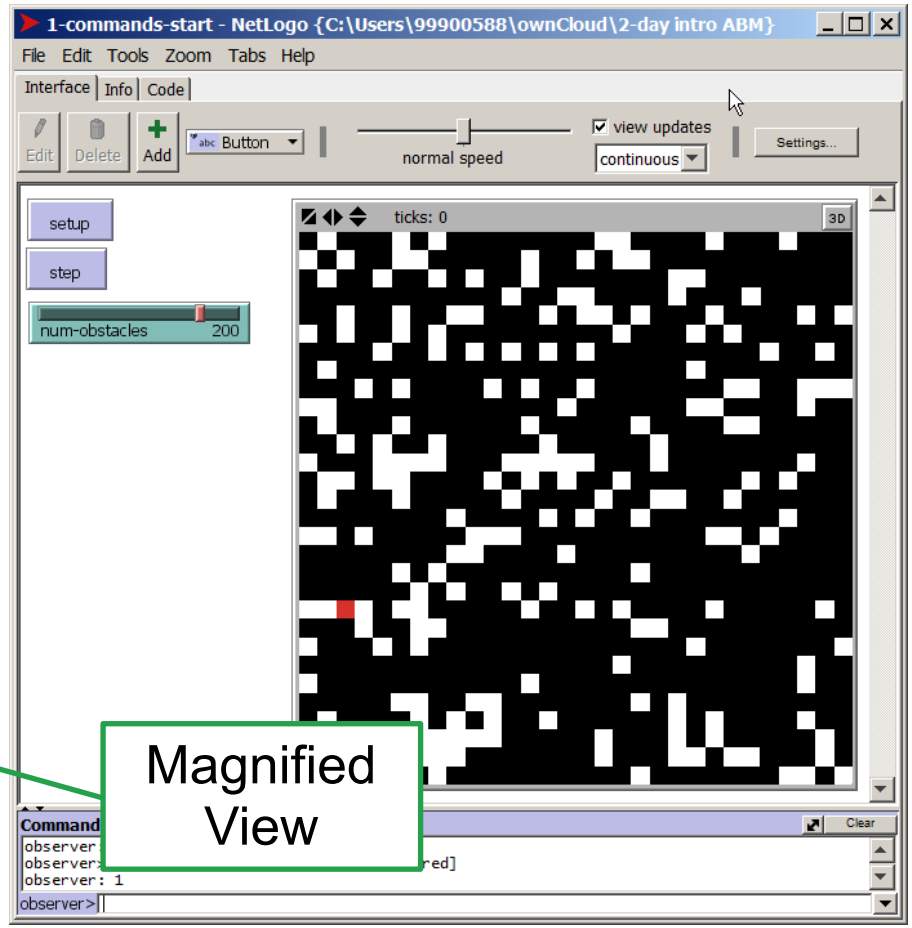
Manchester Metropolitan University

Inspecting Patches and Agents



Inspecting Patches and Agents



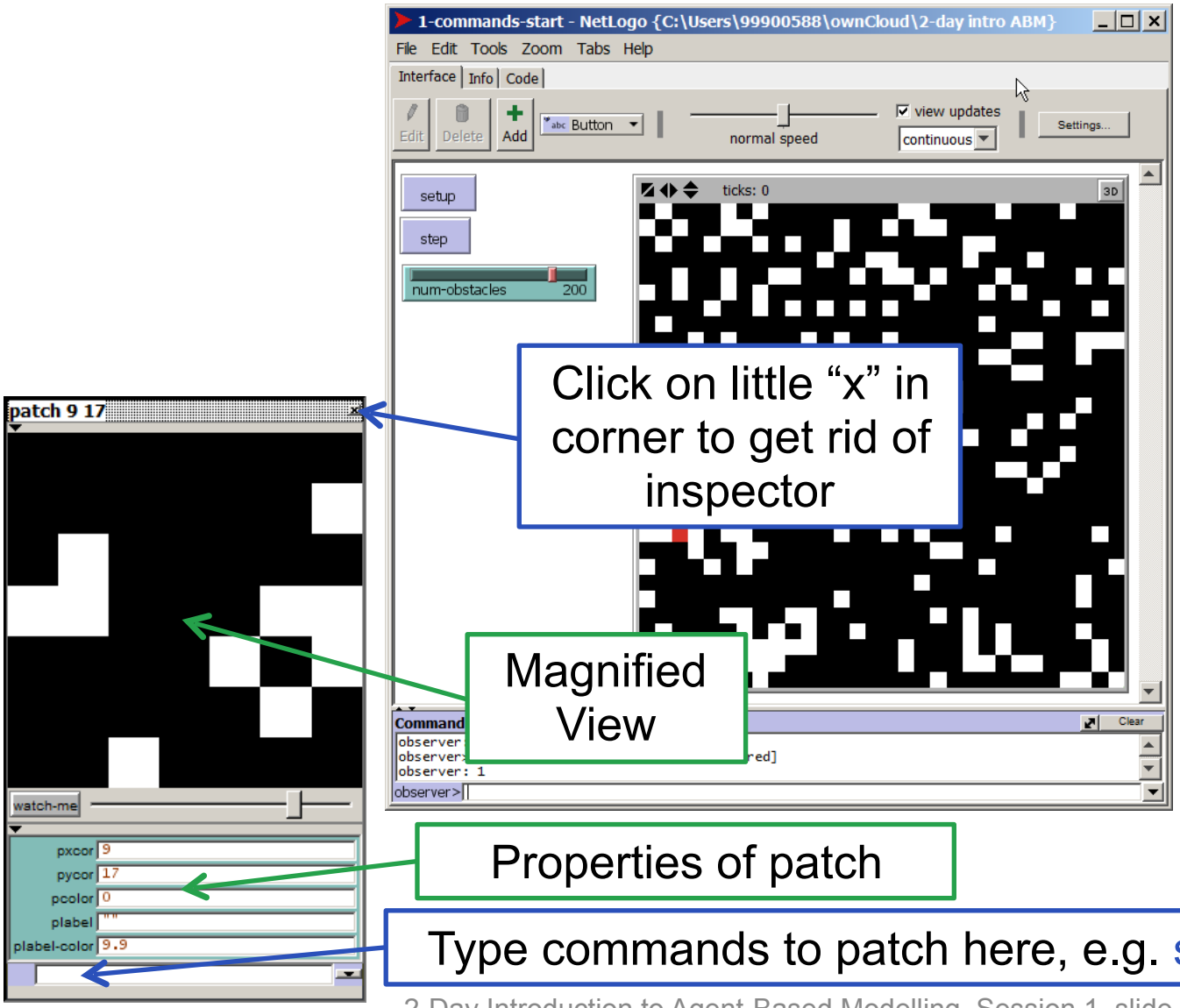


Magnified View

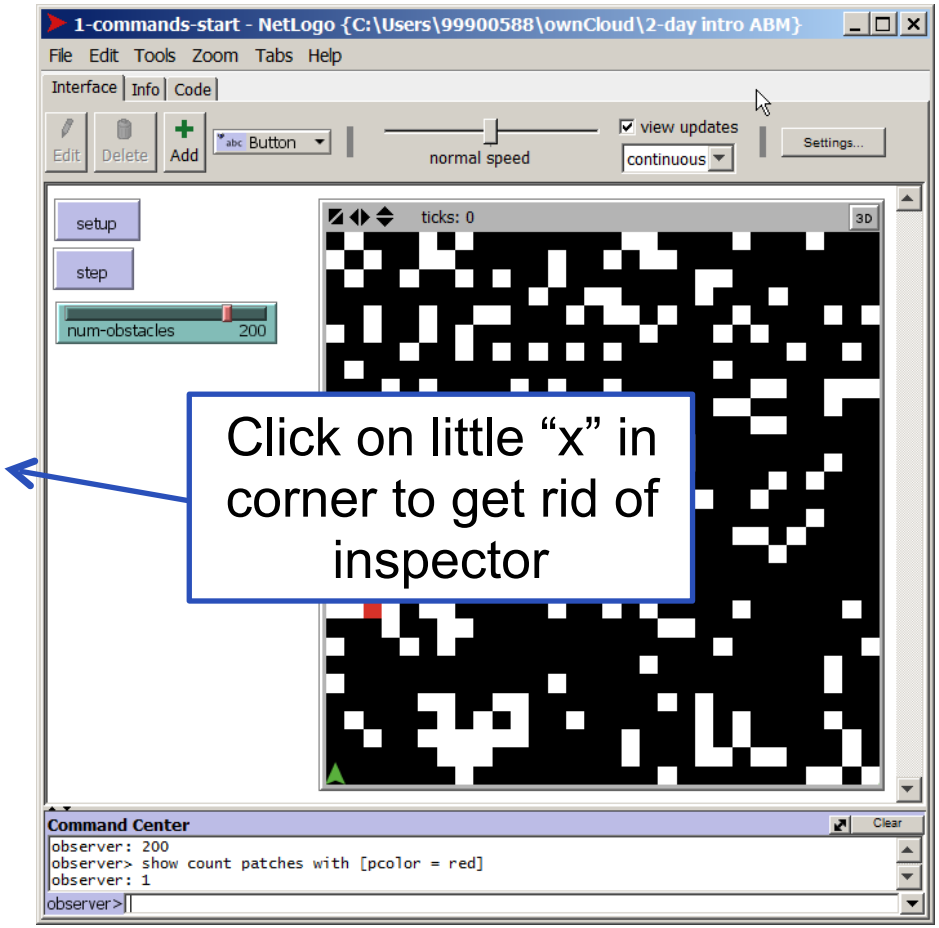
Properties of patch

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

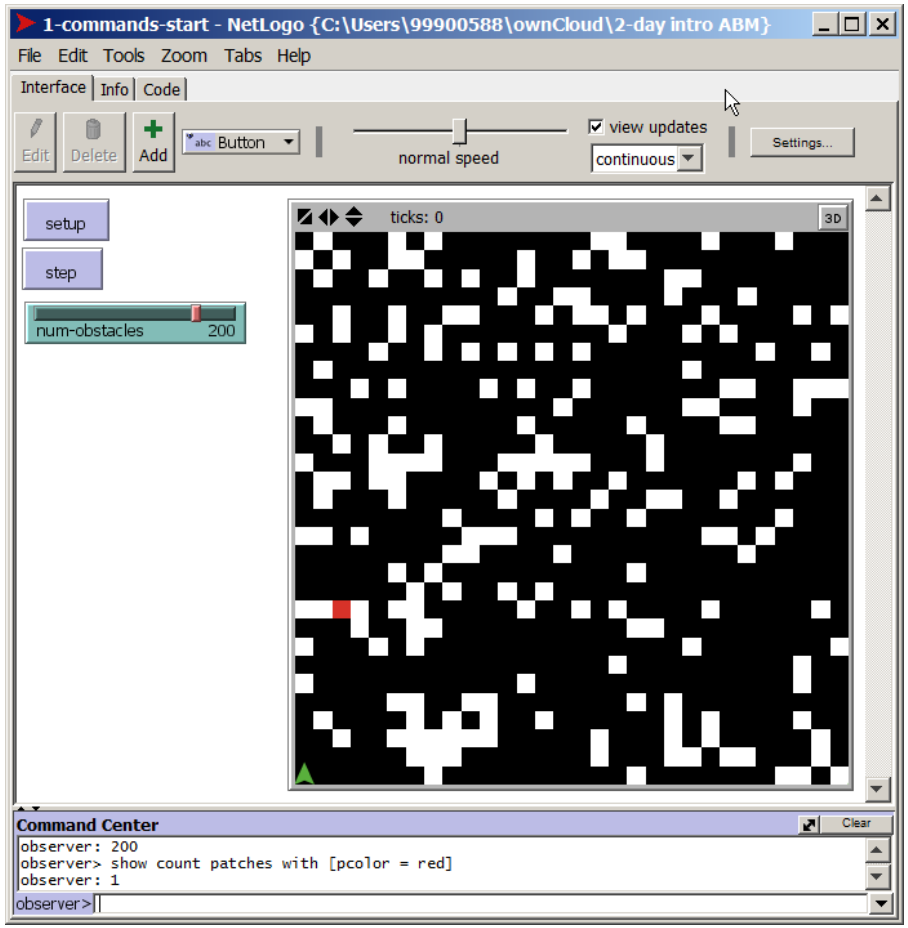
Inspecting Patches and Agents



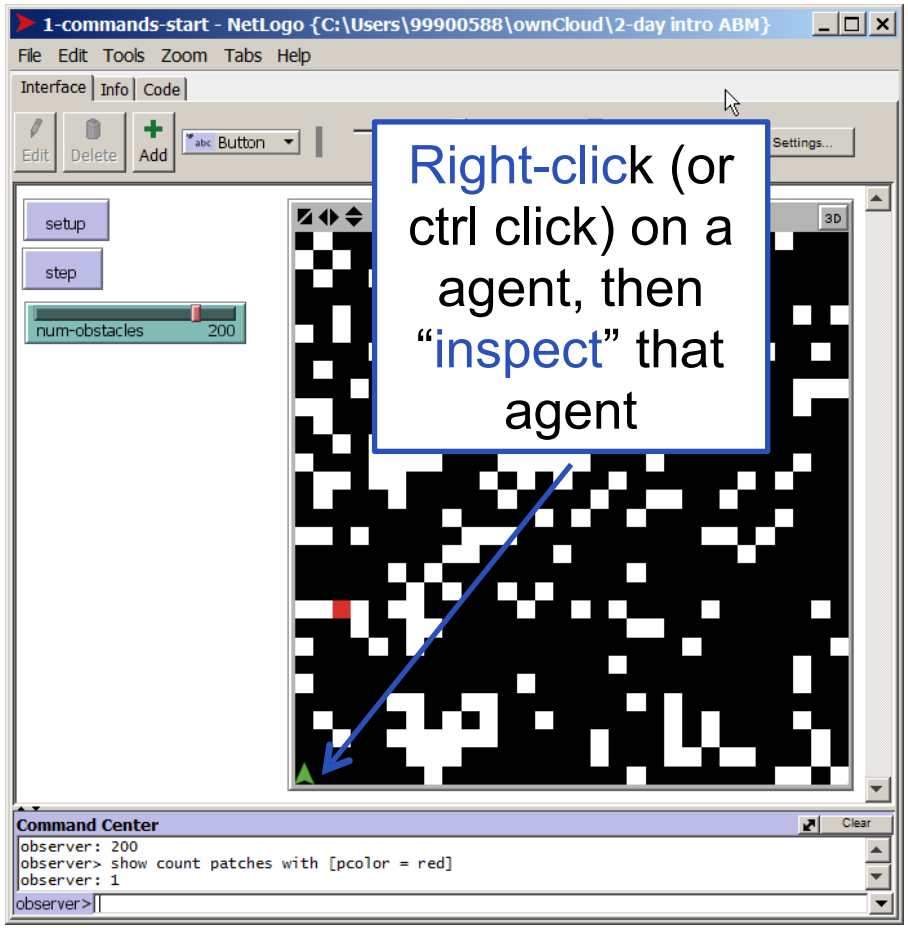
Inspecting Patches and Agents



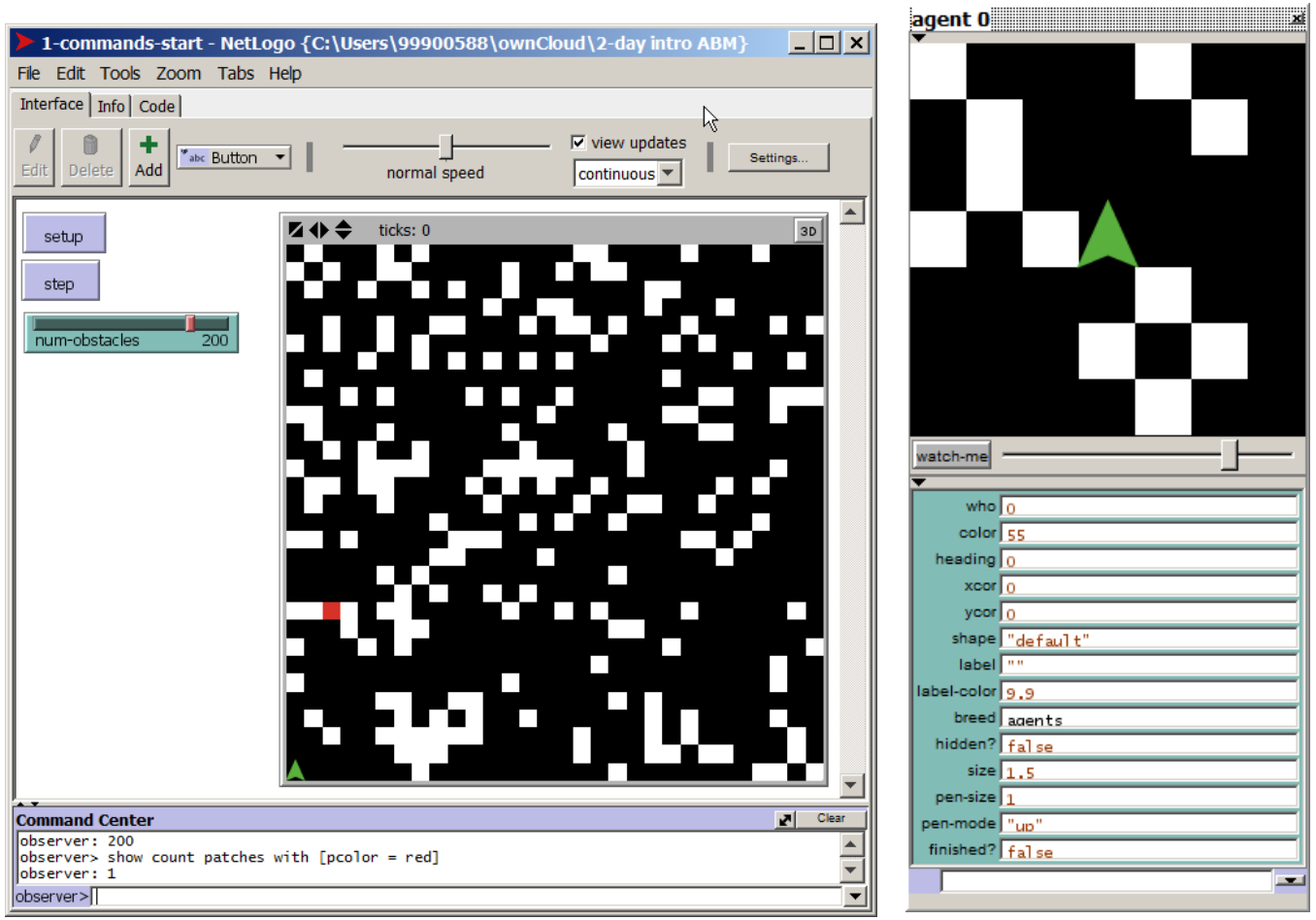
Inspecting Patches and Agents



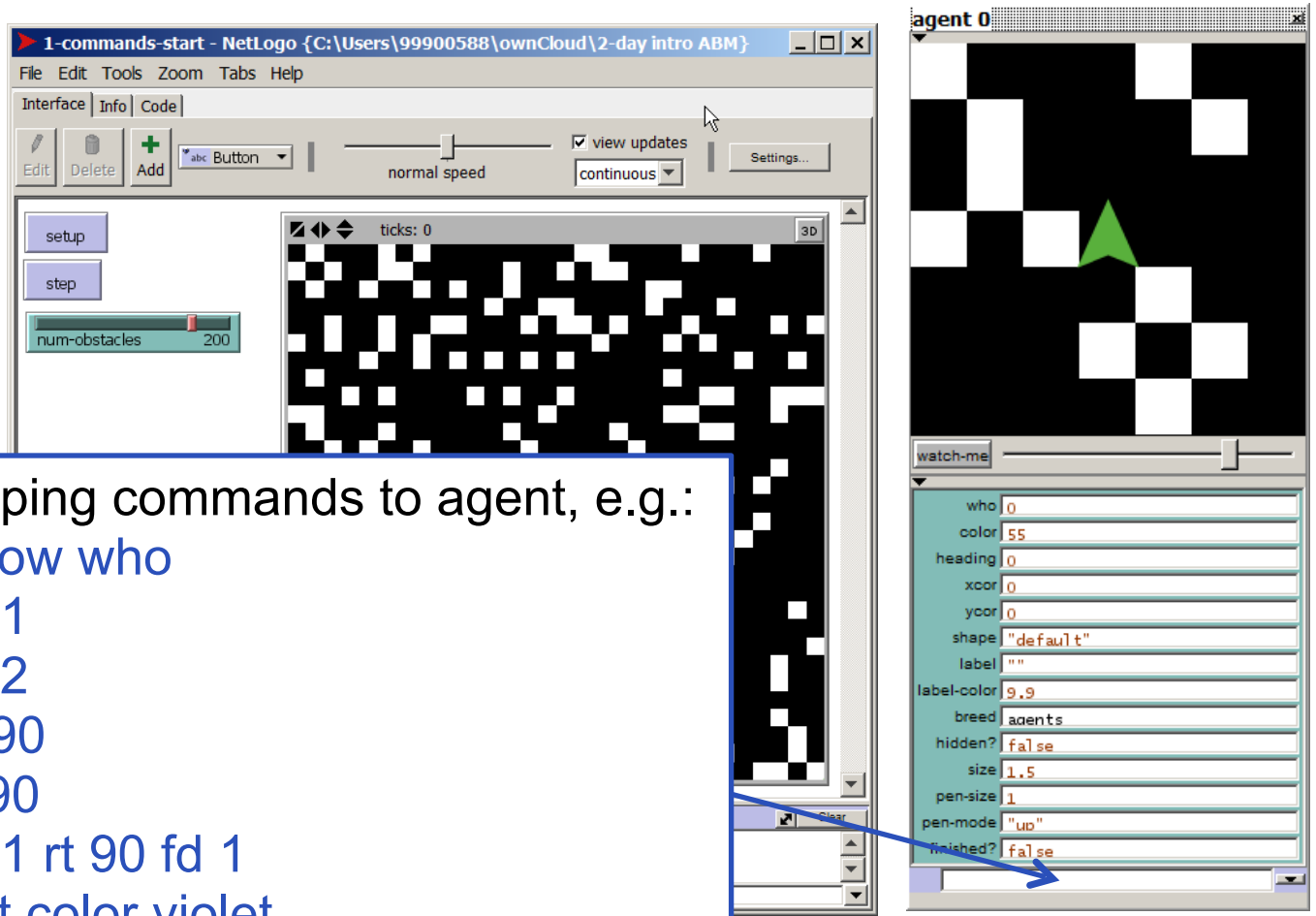
Inspecting Patches and Agents



Inspecting Patches and Agents



Inspecting Patches and Agents



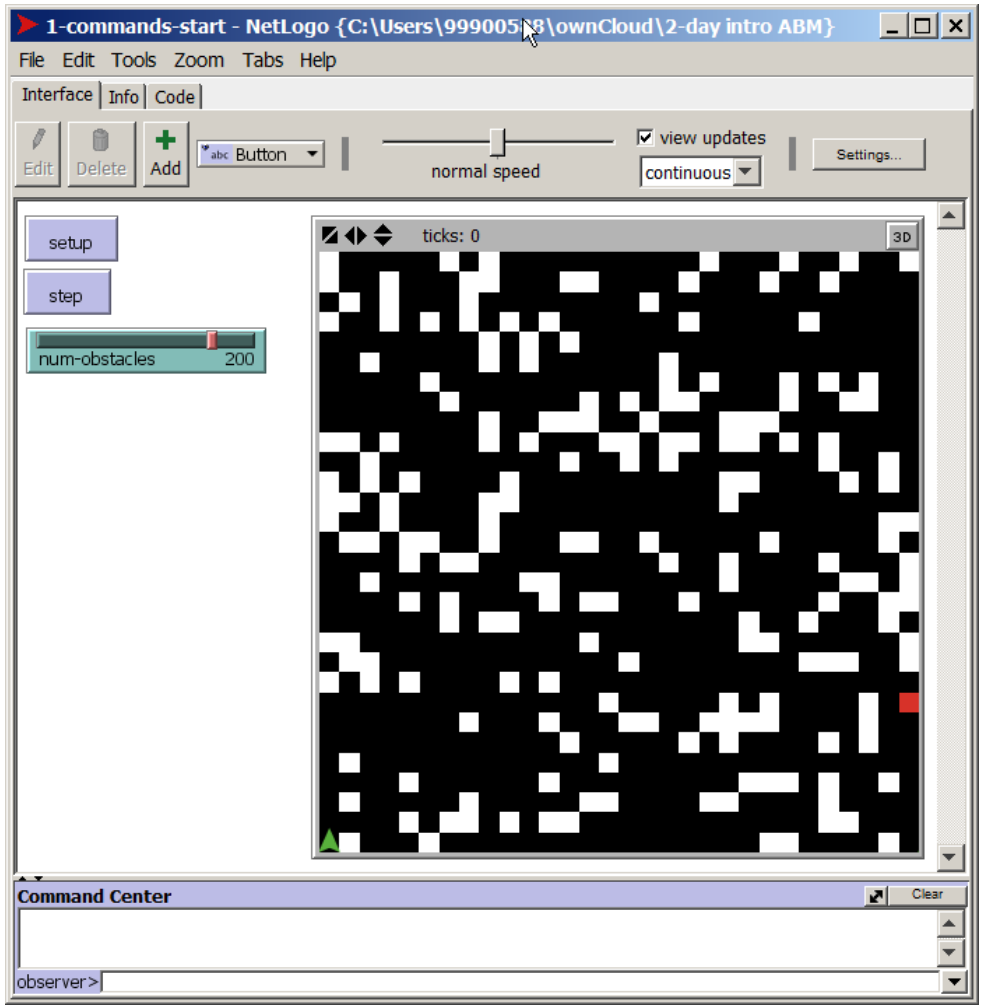
Try typing commands to agent, e.g.:

- 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”



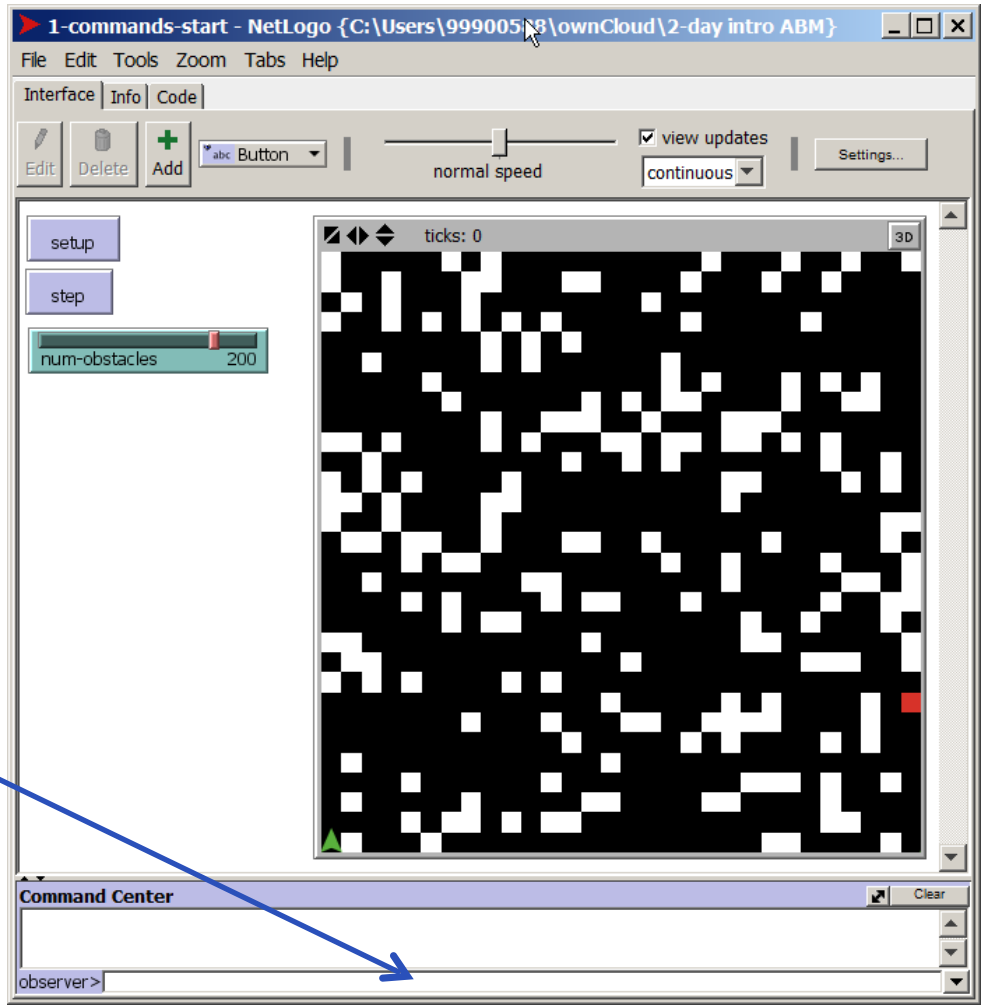
Using “ask”

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

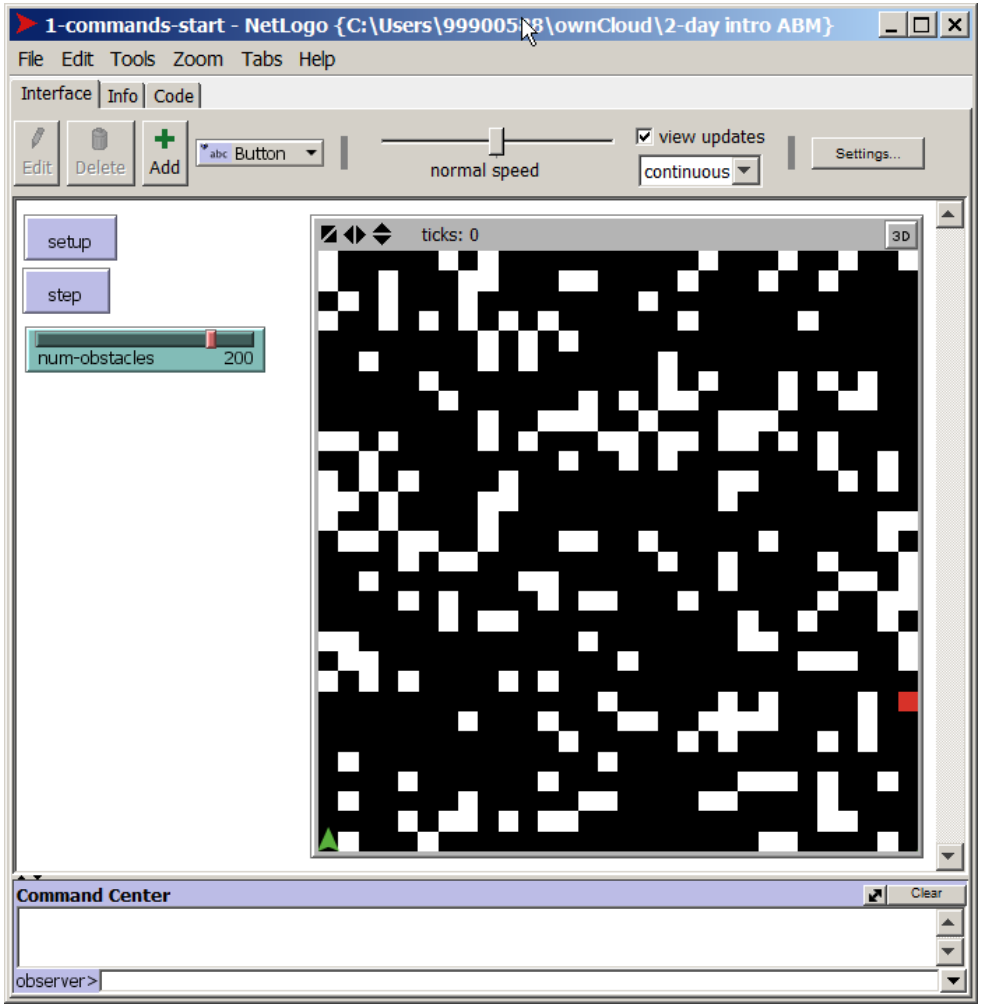
- ask agents [fd 1]
- ask agents [set color grey]
- ask agents [set shape “person”]
- ask agents [fd 1 rt 90 fd 1]
- ask agents [show patch-here]
- etc.

Can also ask patches:

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



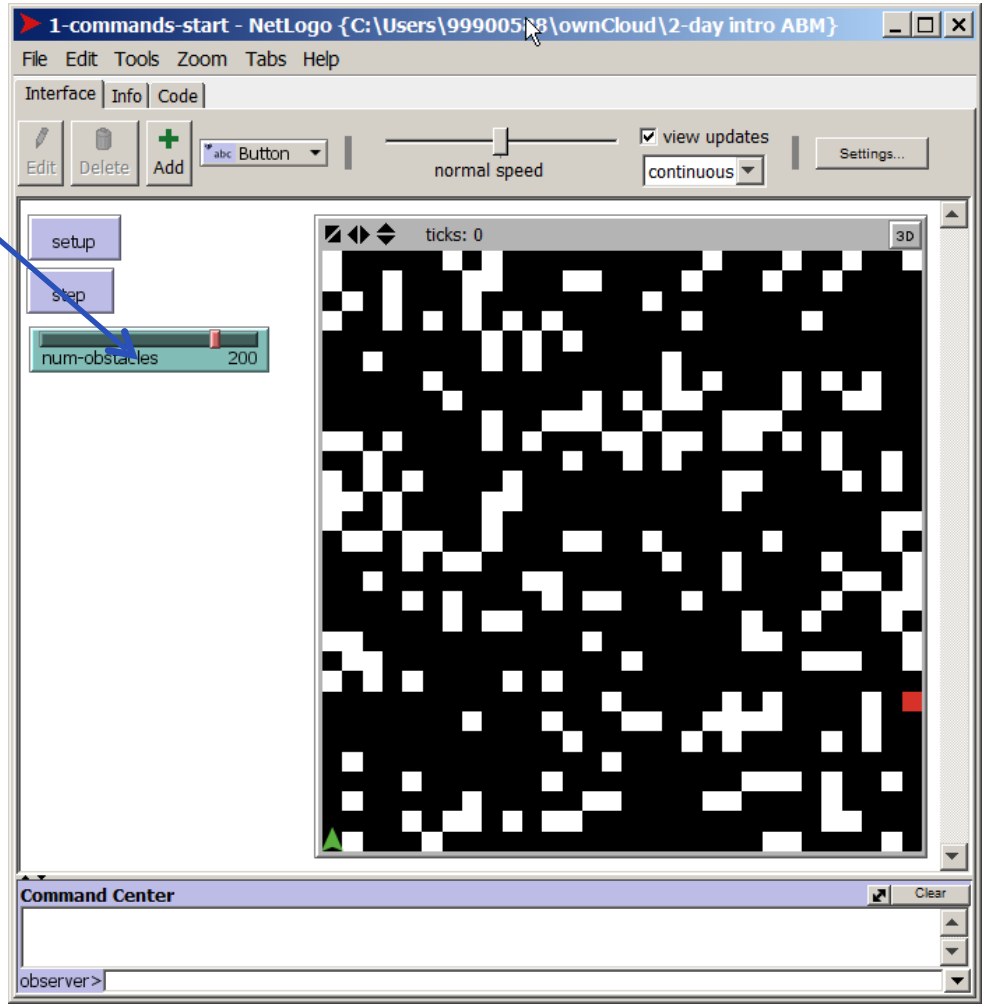
Running a simulation (the hard way!)



- 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 (the hard way!)

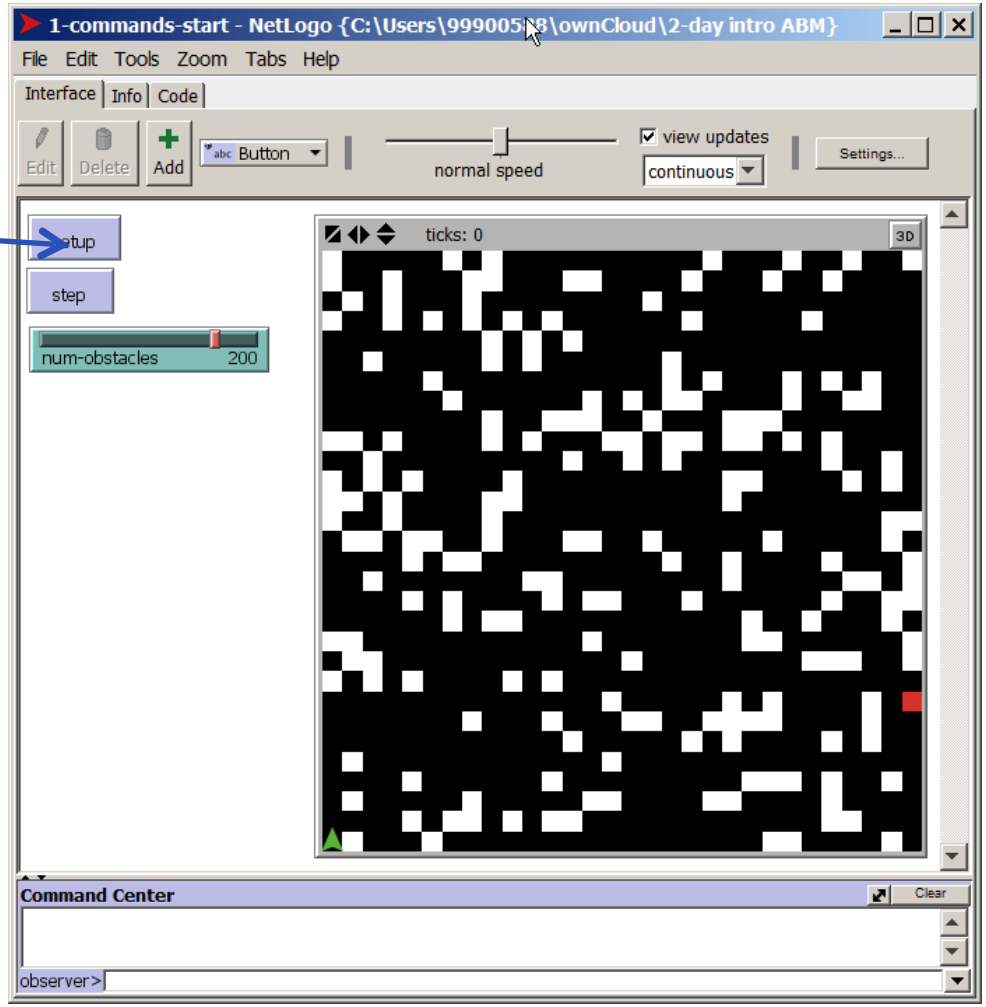
1. Move the slider to change parameter



- 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

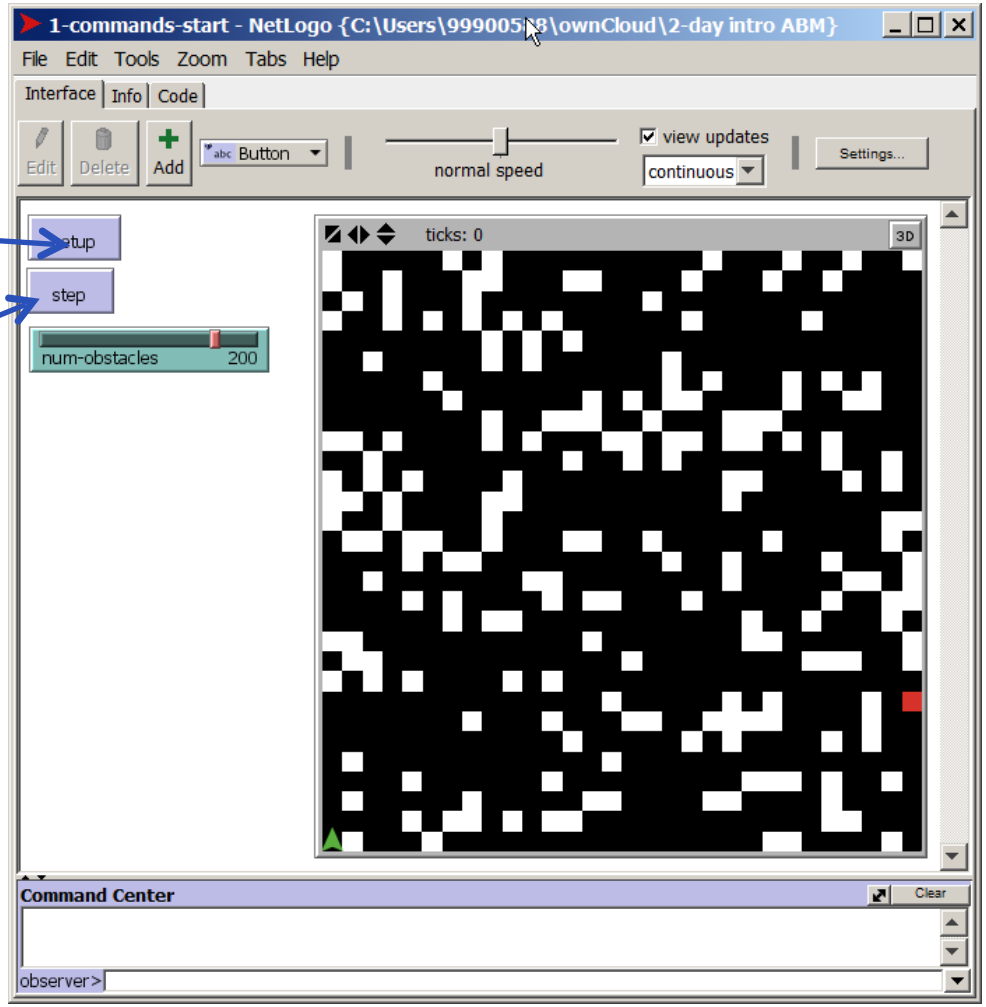


- Each time “step” is pressed the procedure called “go” is caused to run – this is a list of commands, a program.
- We will now look at this.

Running a simulation (the hard way!)

2. Press “setup” to initialise world

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



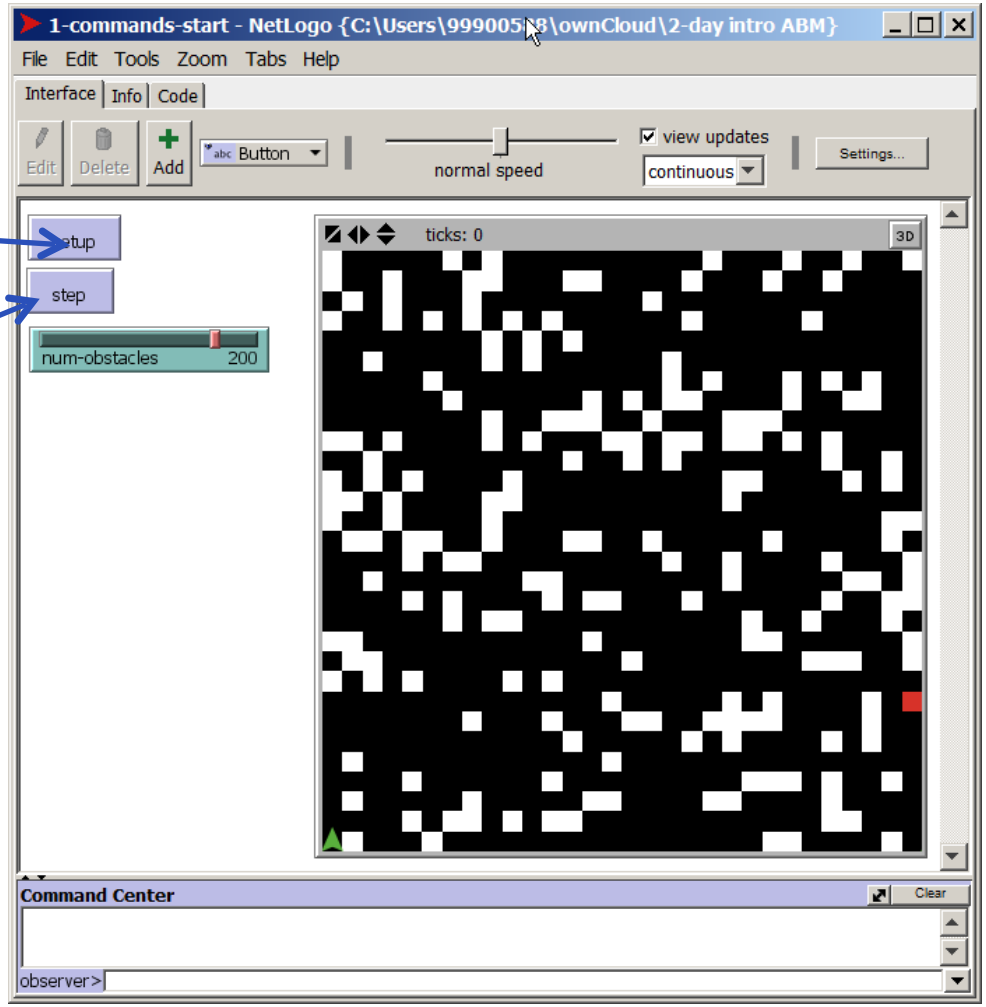
- Each time “step” is pressed the procedure called “go” is caused to run – this is a list of commands, a program.
- We will now look at this.

Running a simulation (the hard way!)

2. Press “setup” to initialise world

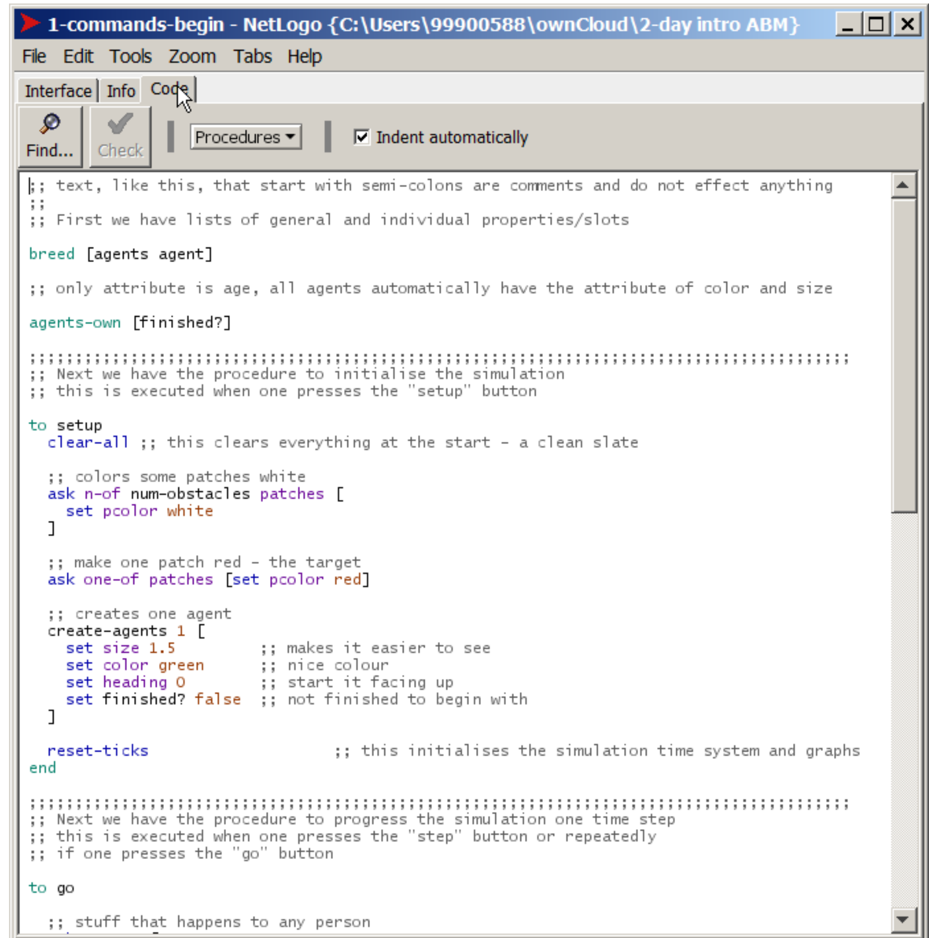
3. Press “step” to make the program run one time step

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.

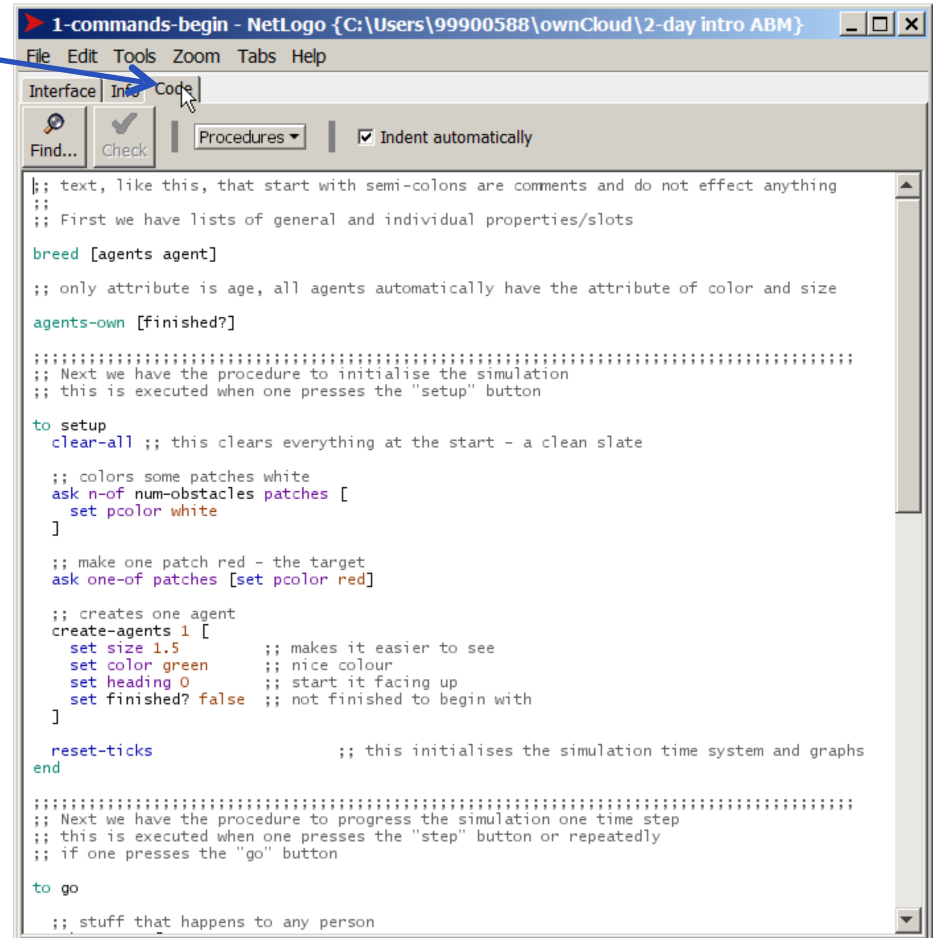
The Program Code



Manchester Metropolitan University

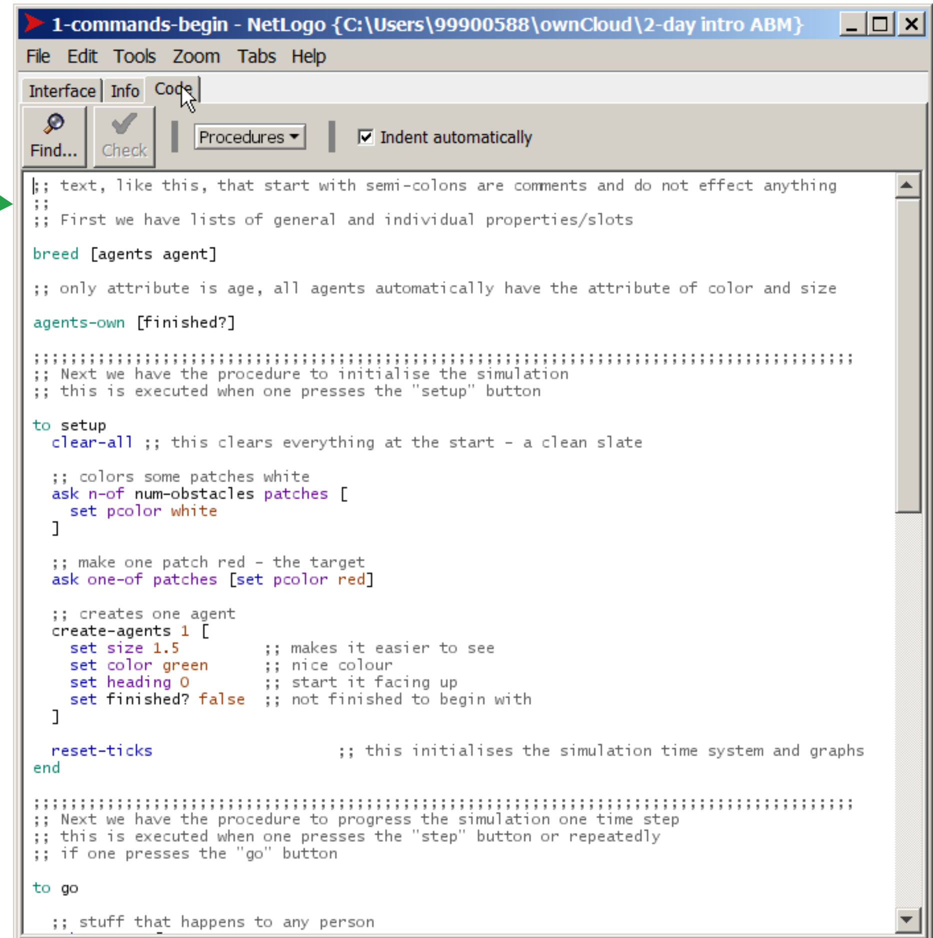
The Program Code

Click on the "Code" tab to see the program



The Program Code

This text is the program

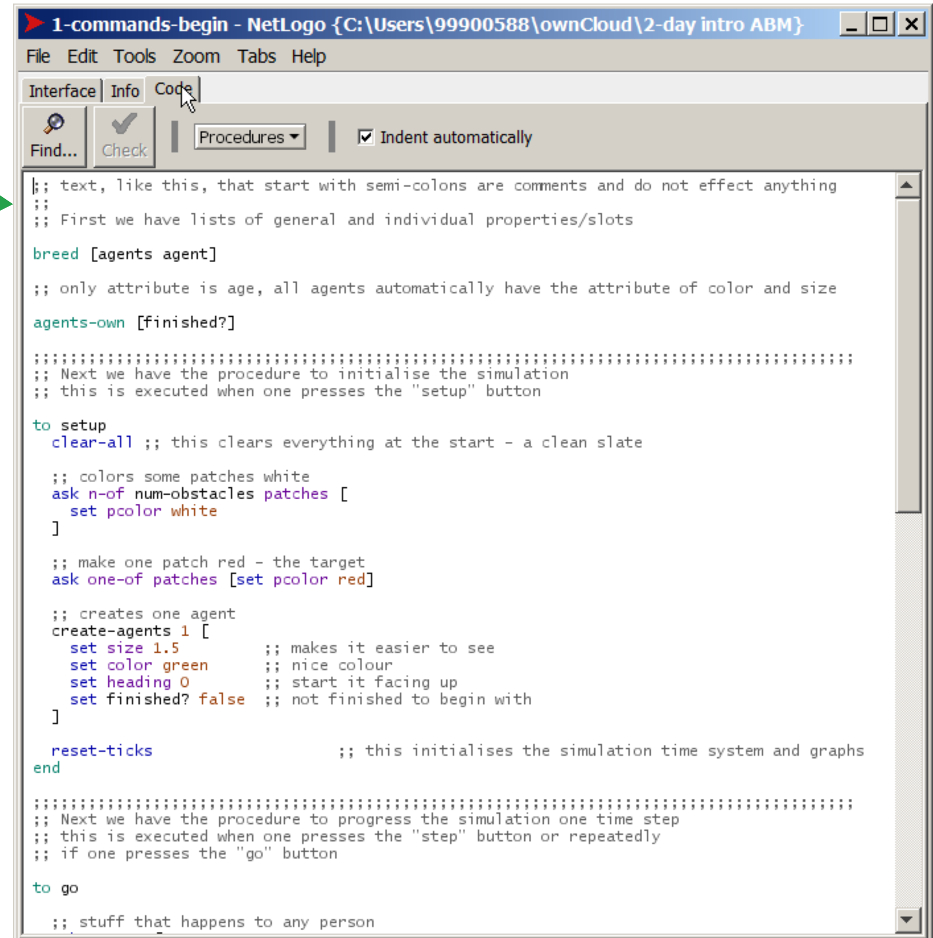


Manchester Metropolitan University

The Program Code

This text is the program

It has different parts



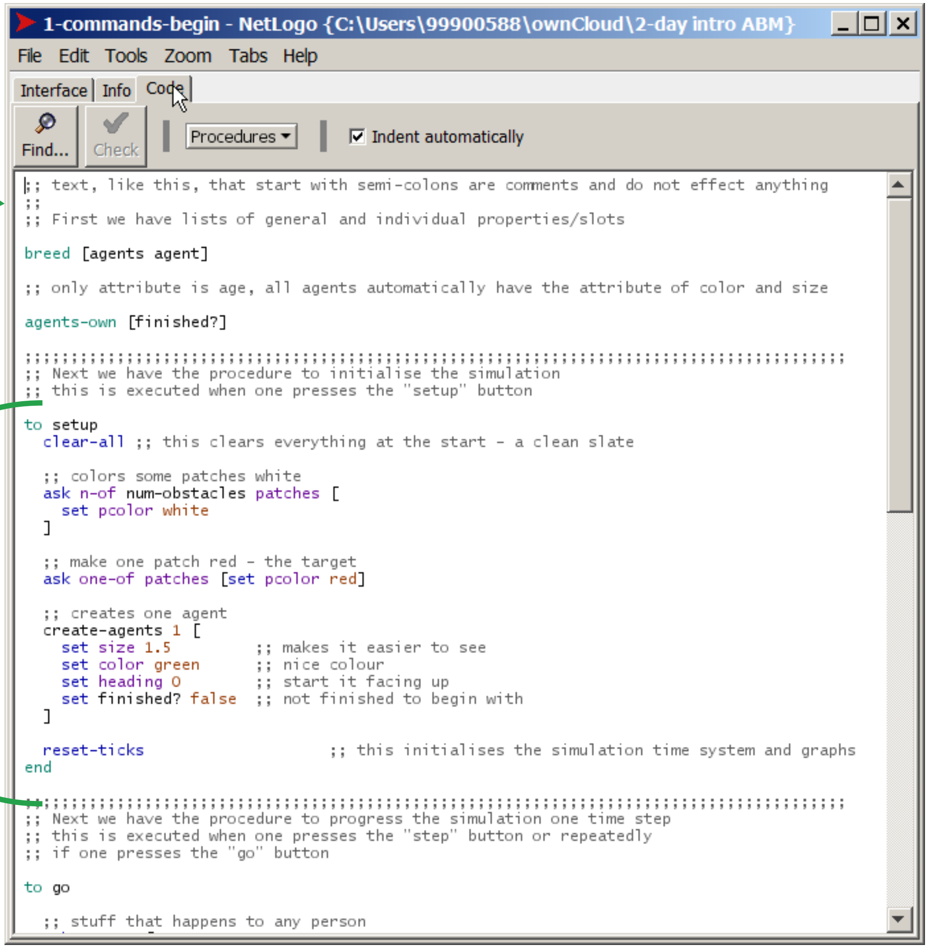
Manchester Metropolitan University

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



Manchester Metropolitan University

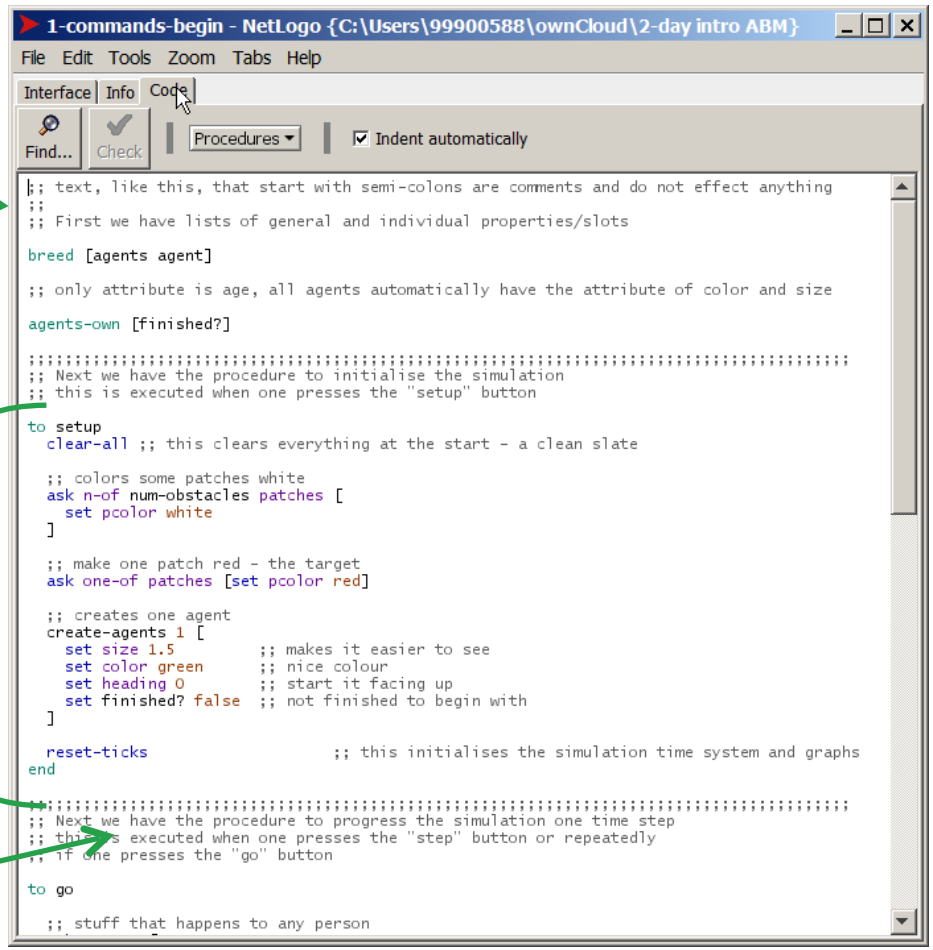
The Program Code

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



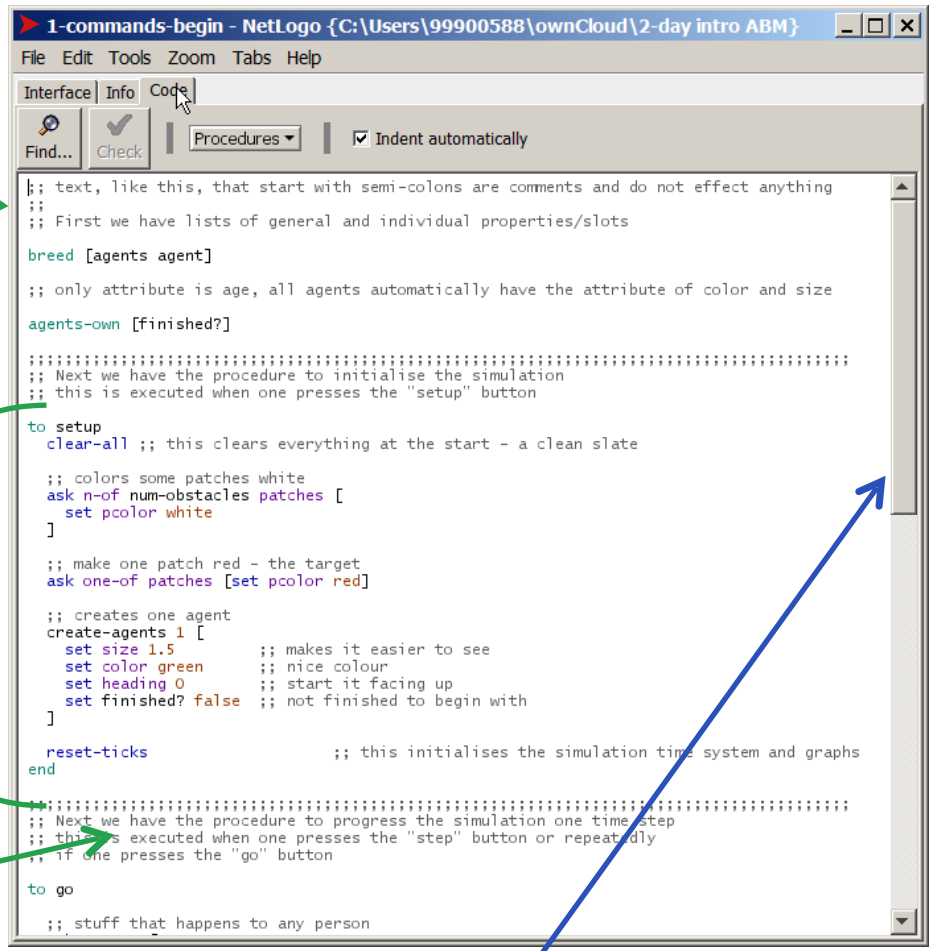
The Program Code

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

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

```
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

"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                                    ;; this progresses the
end
```

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                                        ;; this progresses the
end
```

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                                    ;; this progresses the
end
```

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

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 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

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

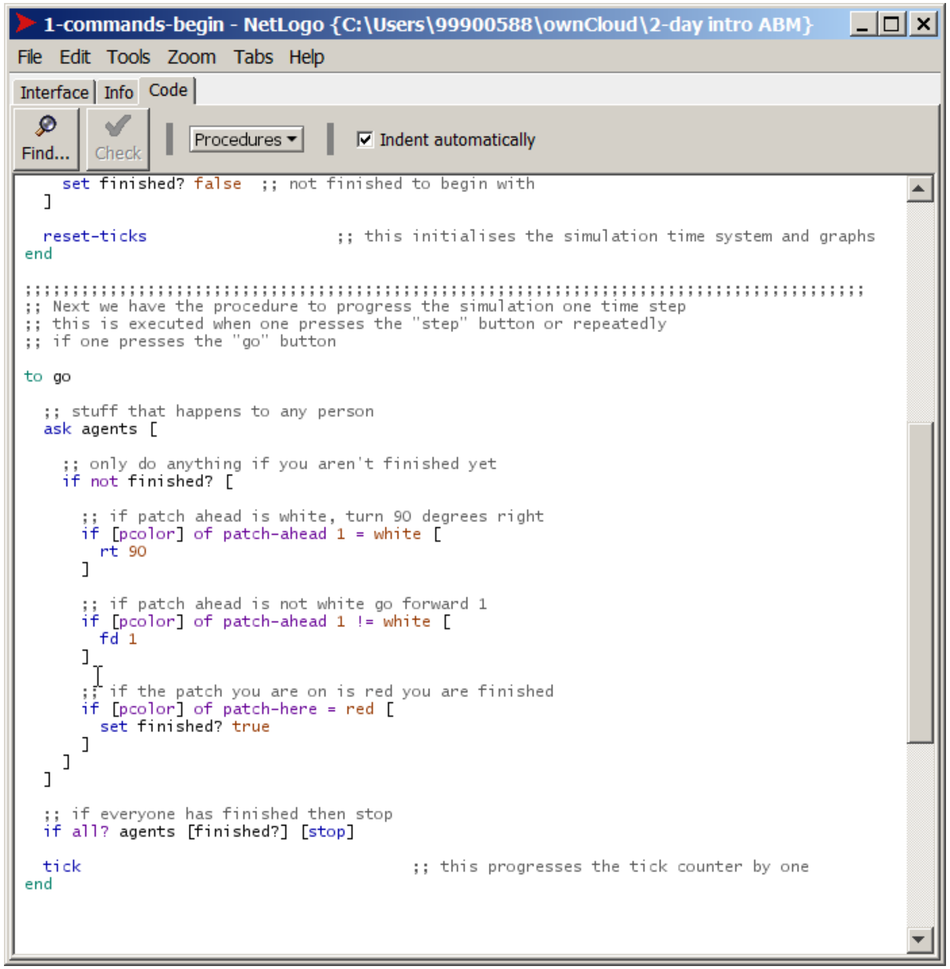
```
to go

  ;; stuff that happens to any person
  ask 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

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

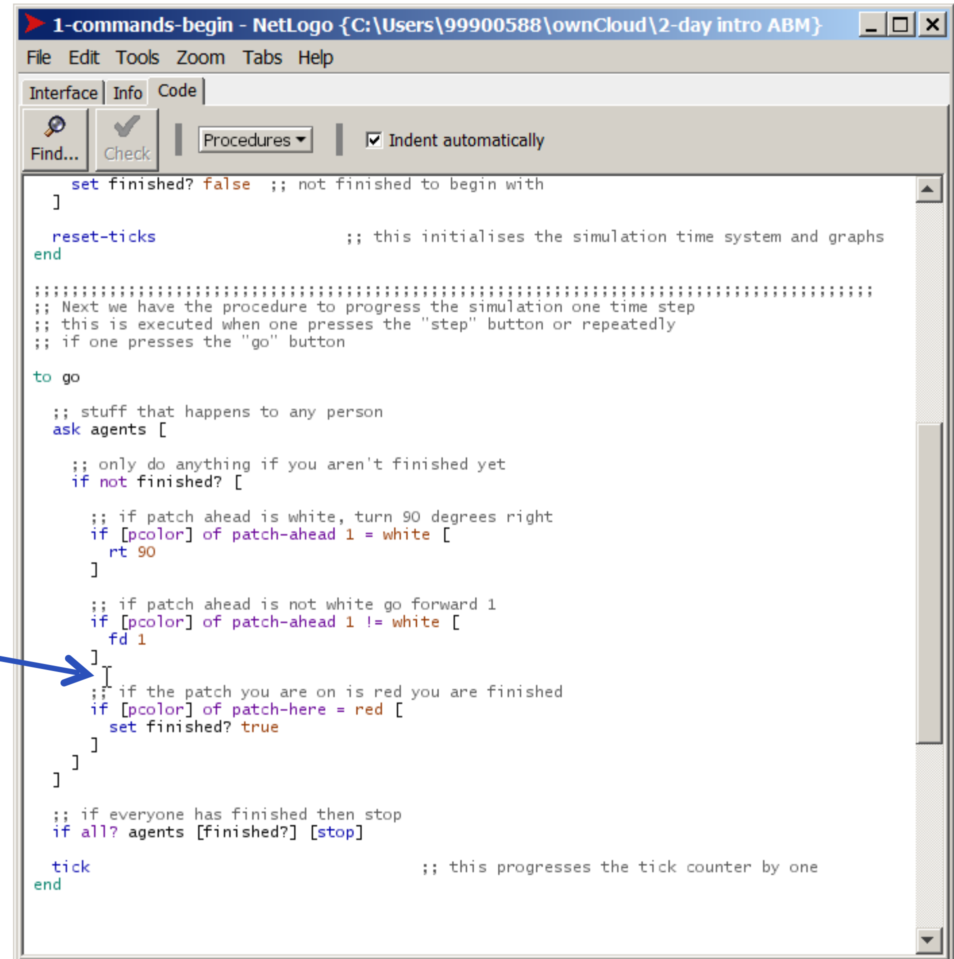
```
to go

  ;; stuff that happens to any person
  ask 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
```

To change the program…



To change the program…

Click within the text and type!



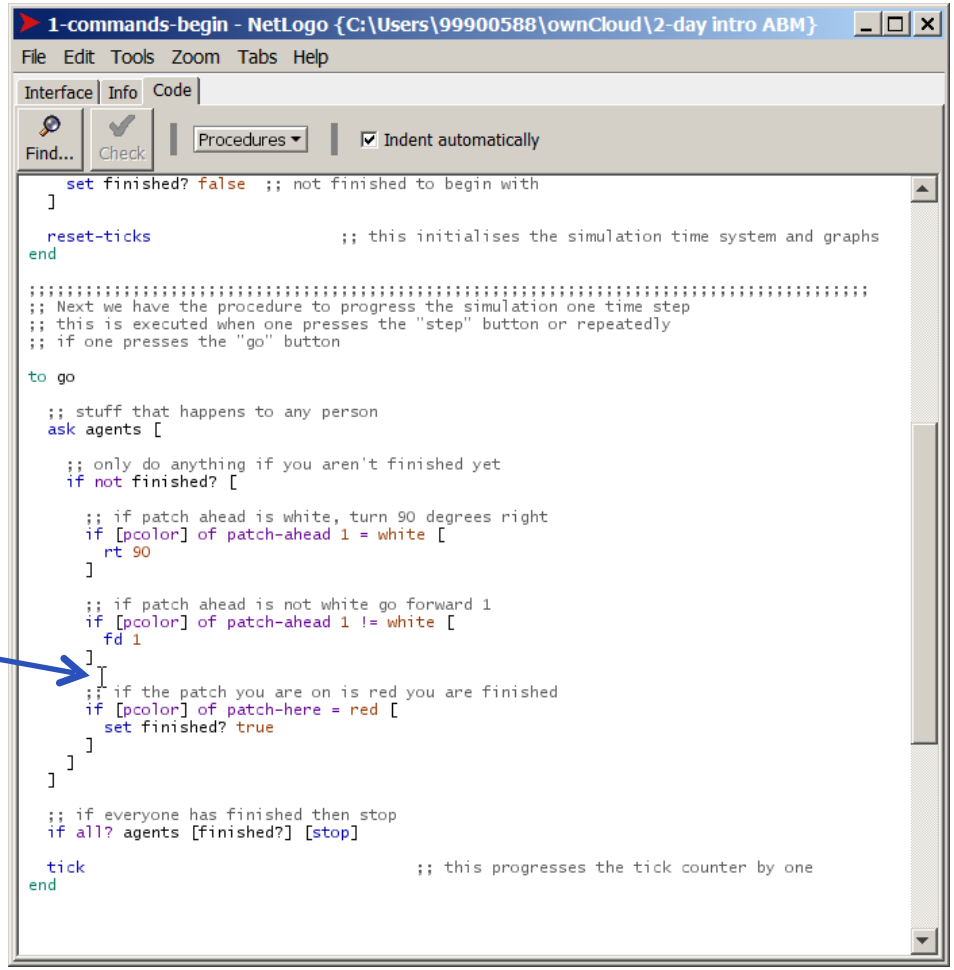
Manchester Metropolitan University

To change the program…

Click within the text and type!

type the following:

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

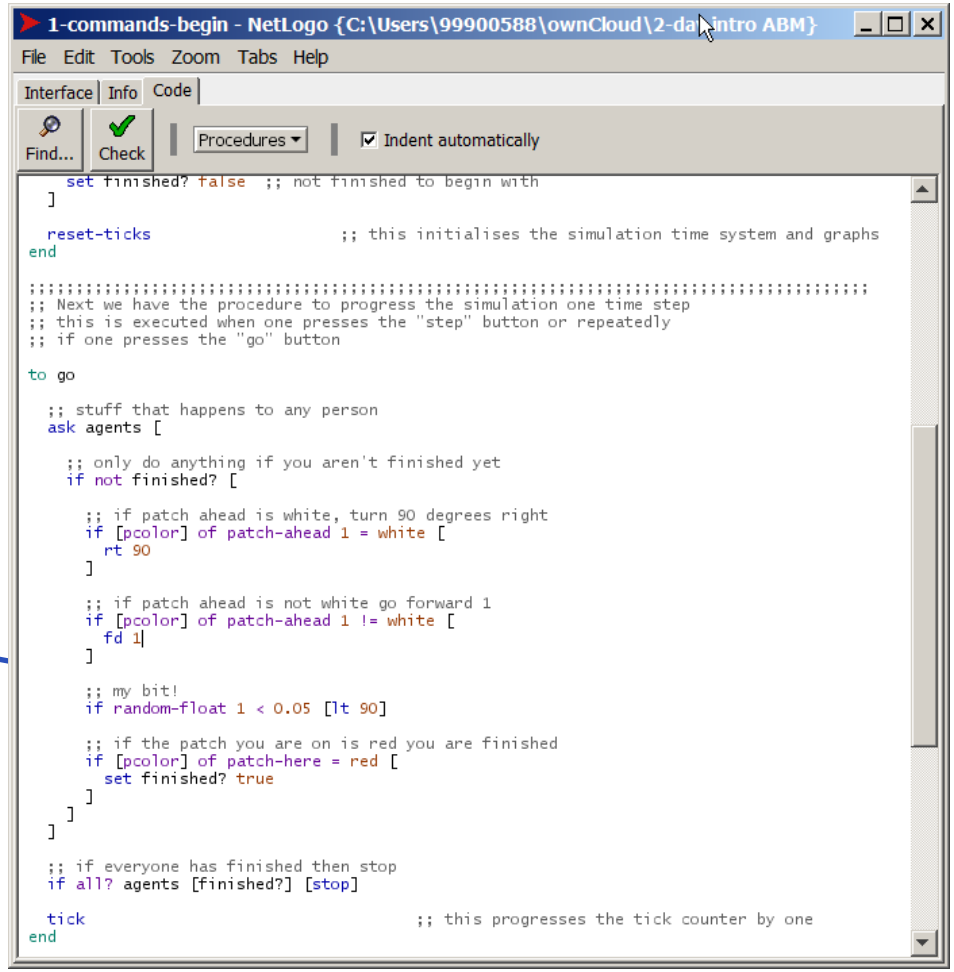


To change the program…

Click within the text and type!

type the following:

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



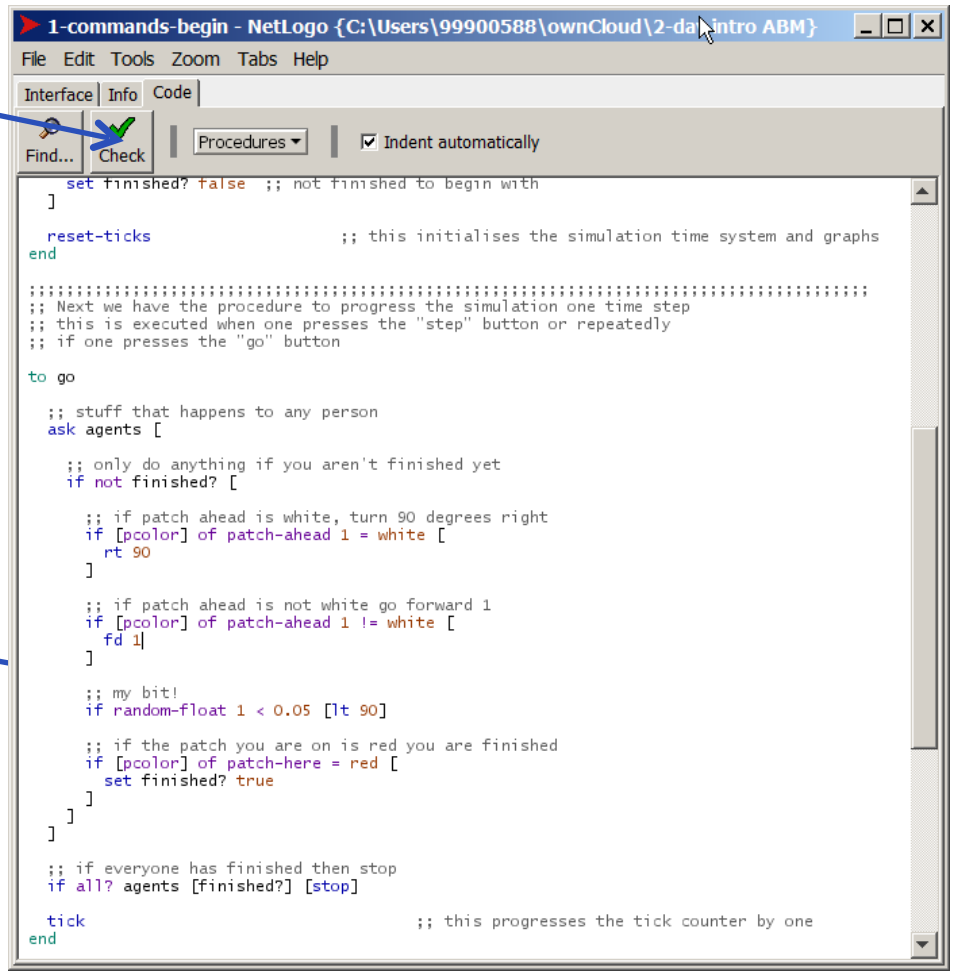
To change the program…

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]



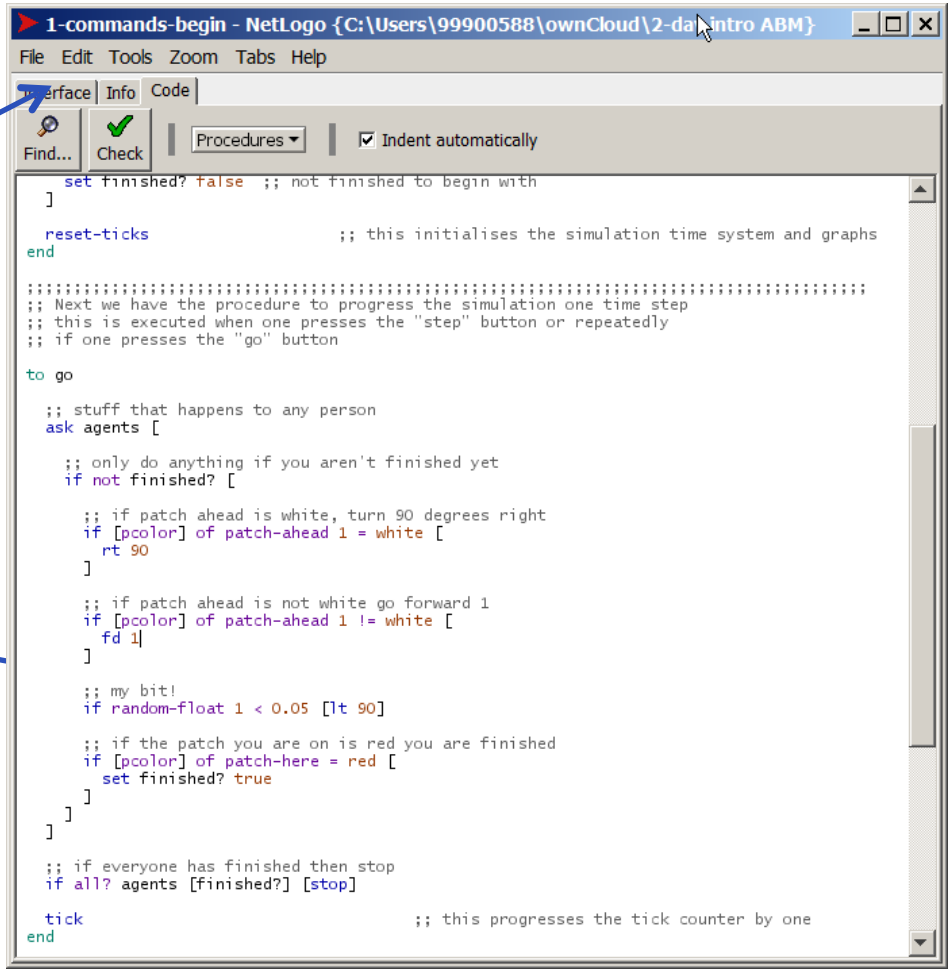
To change the program…

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)

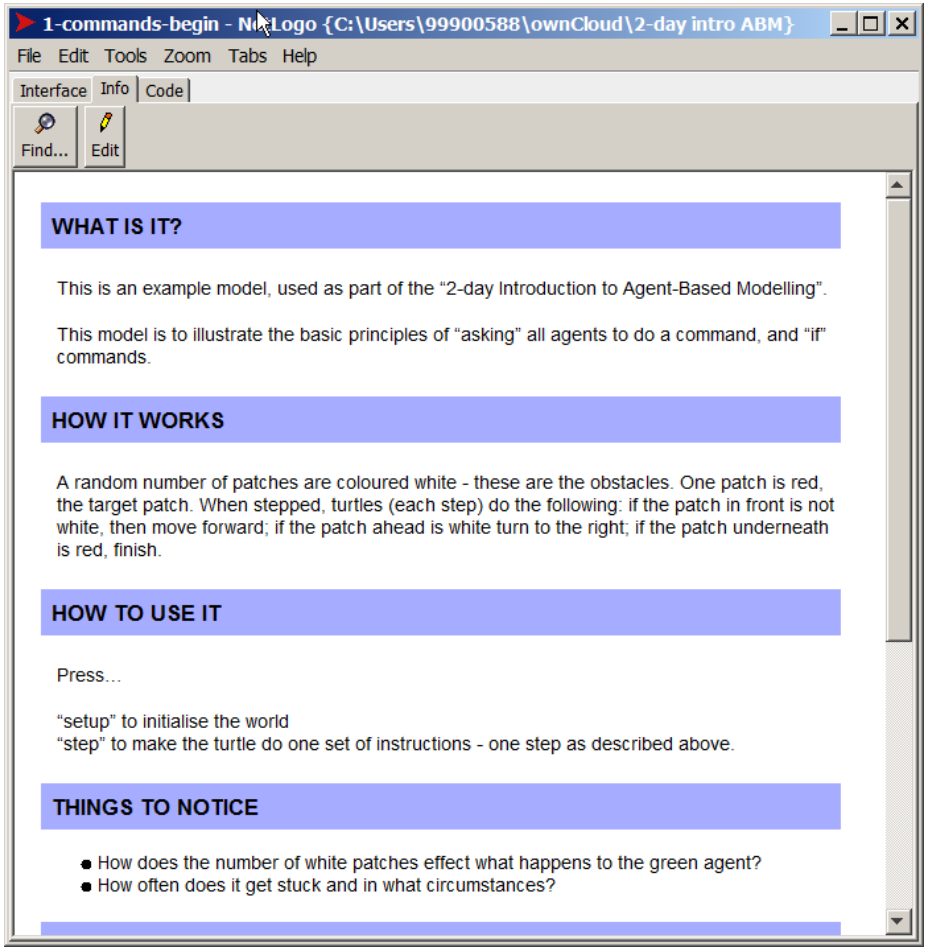
Click within the text and type!

type the following:

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

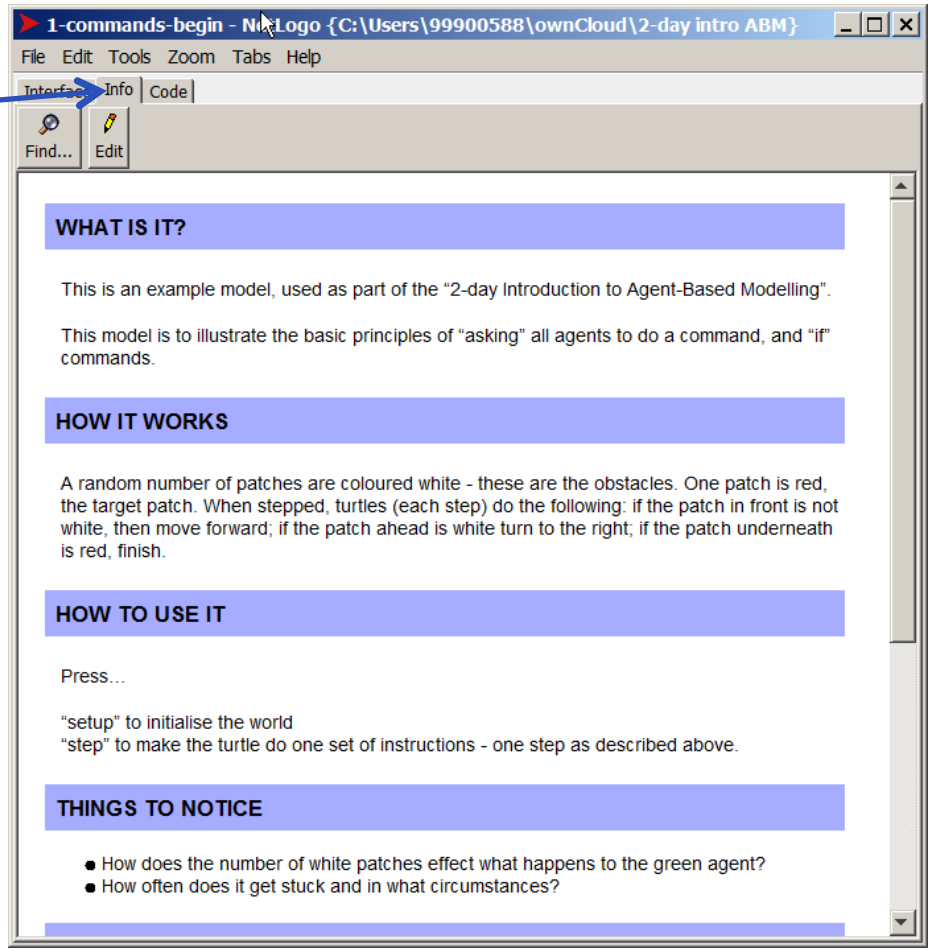


The Information Tab



The Information Tab

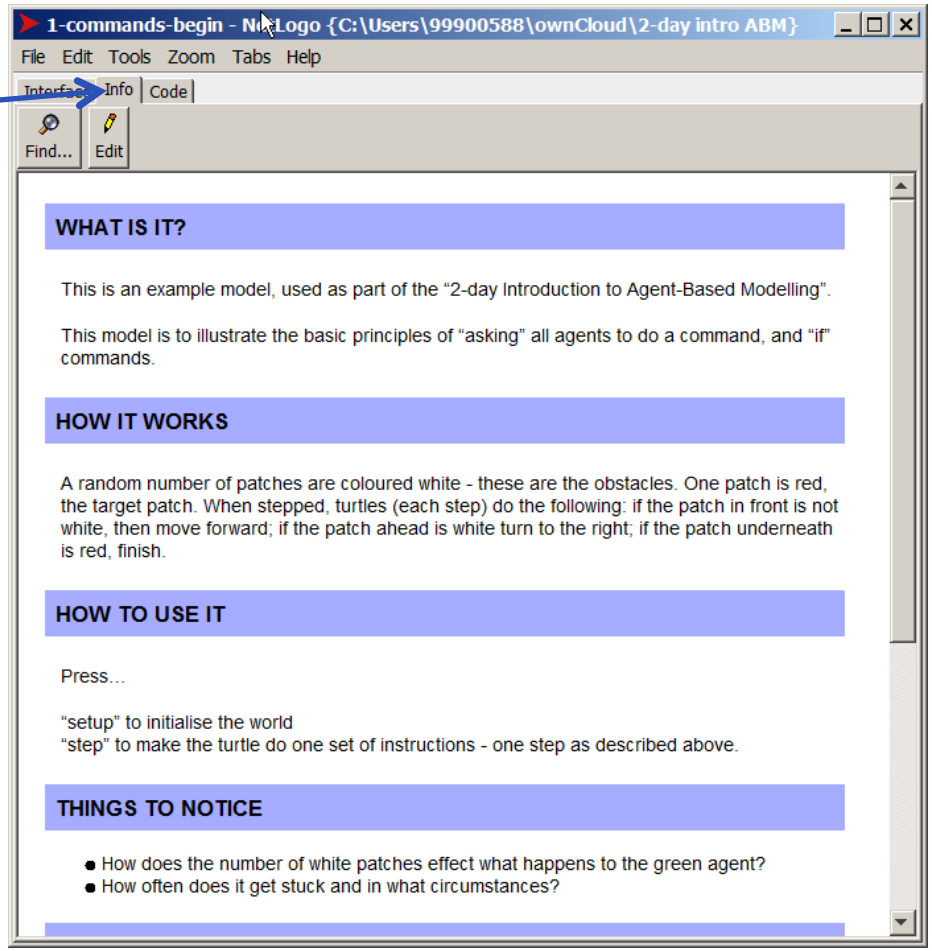
Click on the "**Info**" tab to see a description of the model (or whatever the programmer has written, if anything!)



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

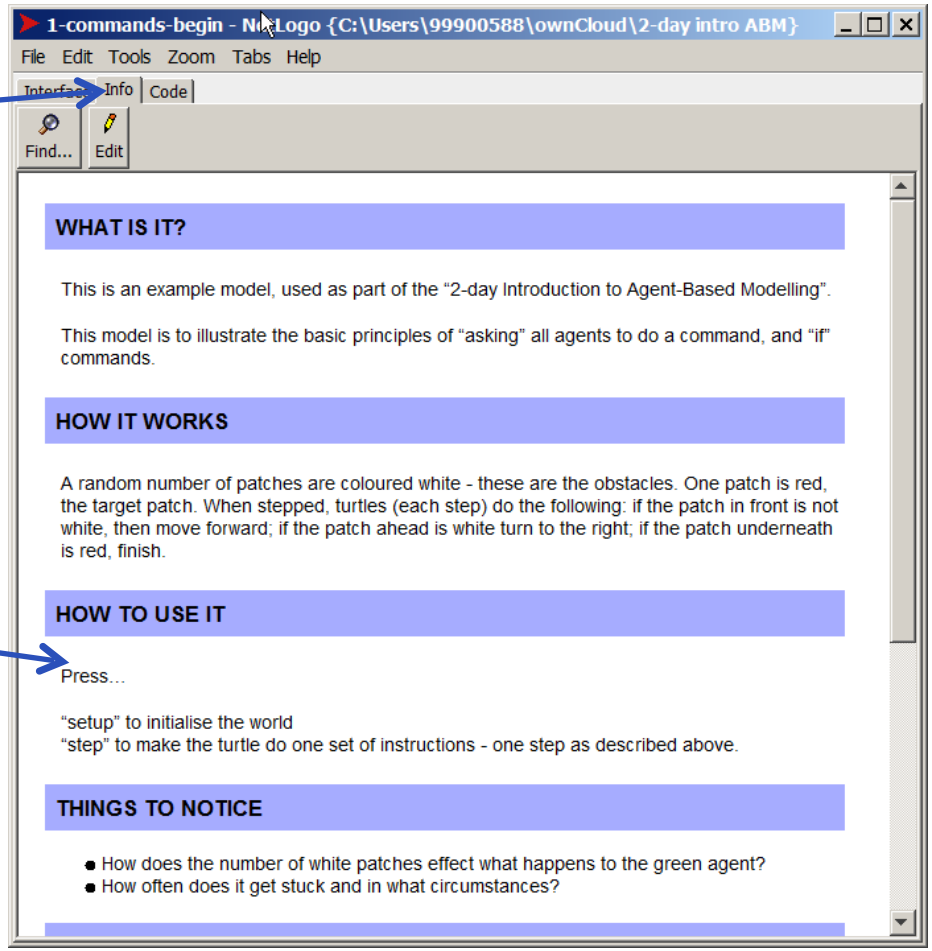


The Information Tab

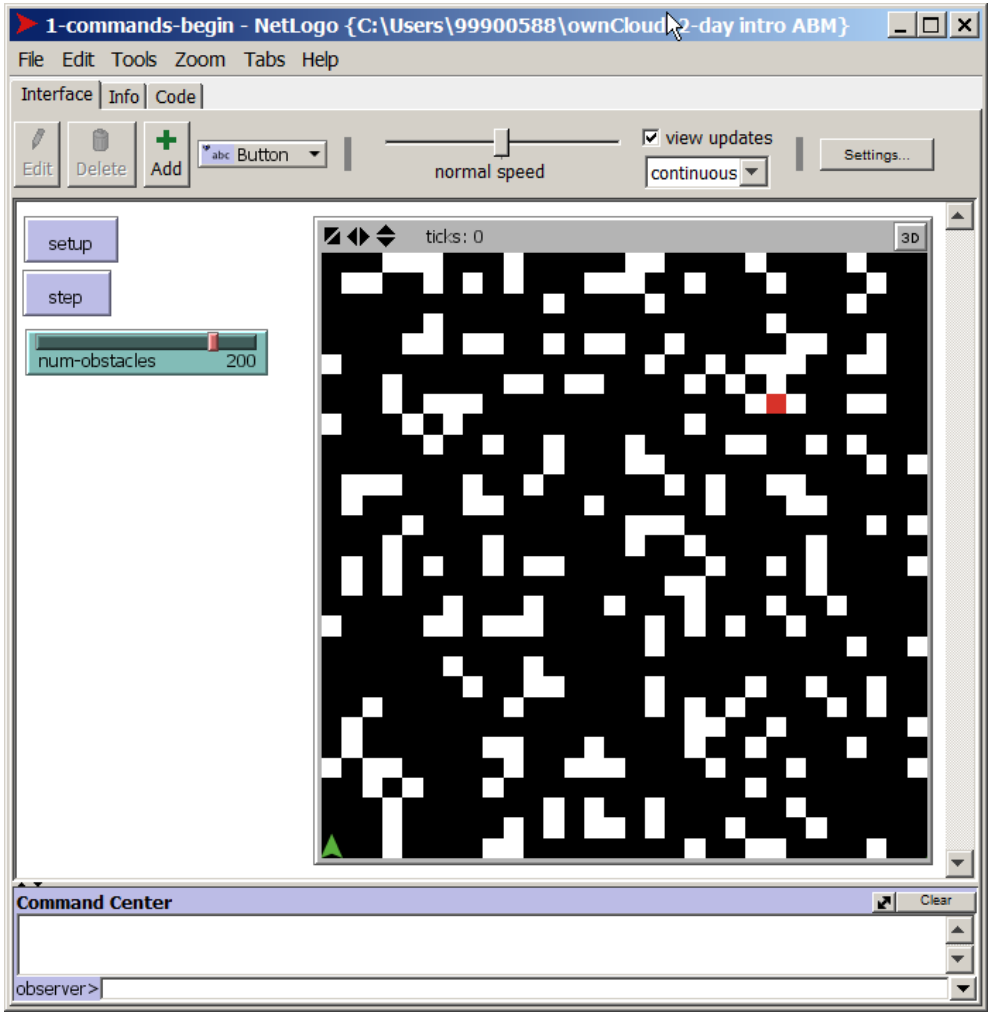
Click on the "**Info**" tab to see a description of the model (or whatever the programmer has written, if anything!)

Read it, scrolling down

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

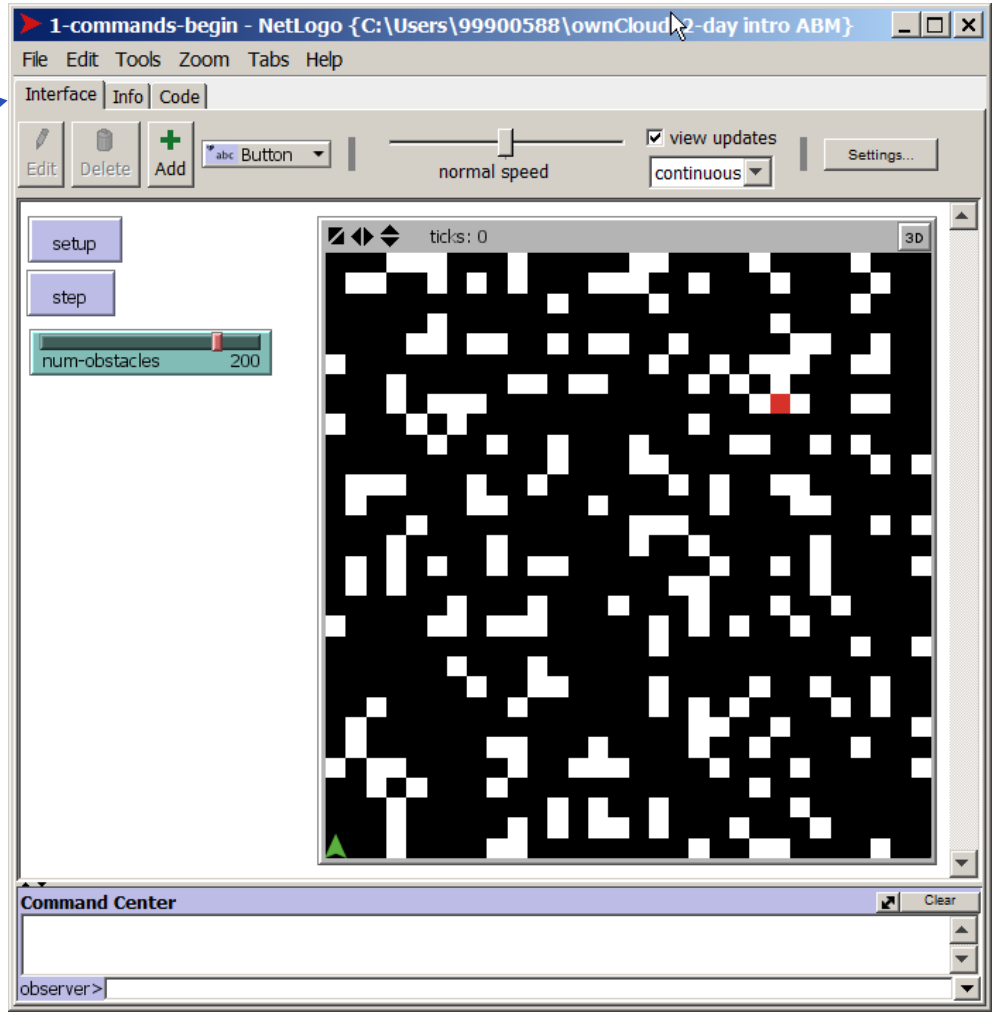


Adding a button and running the code (the fast way!)



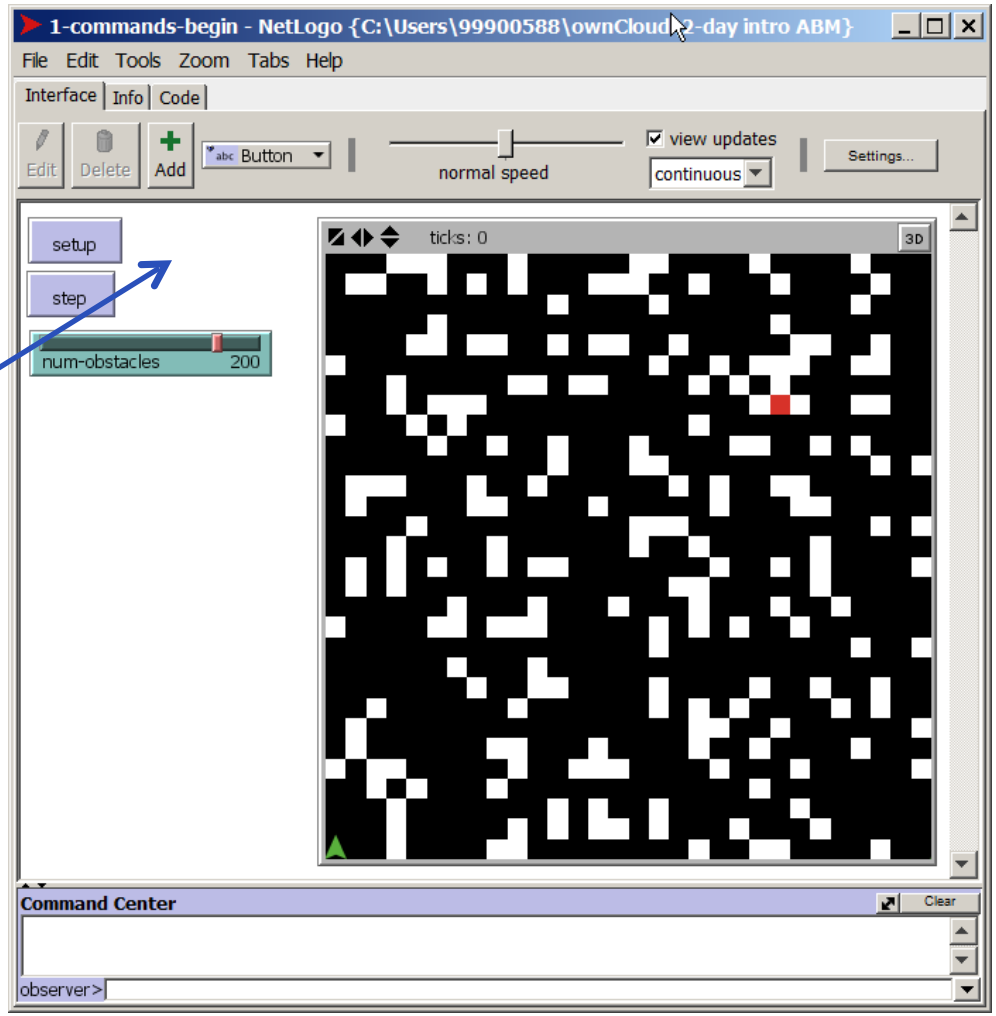
Adding a button and running the code (the fast way!)

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



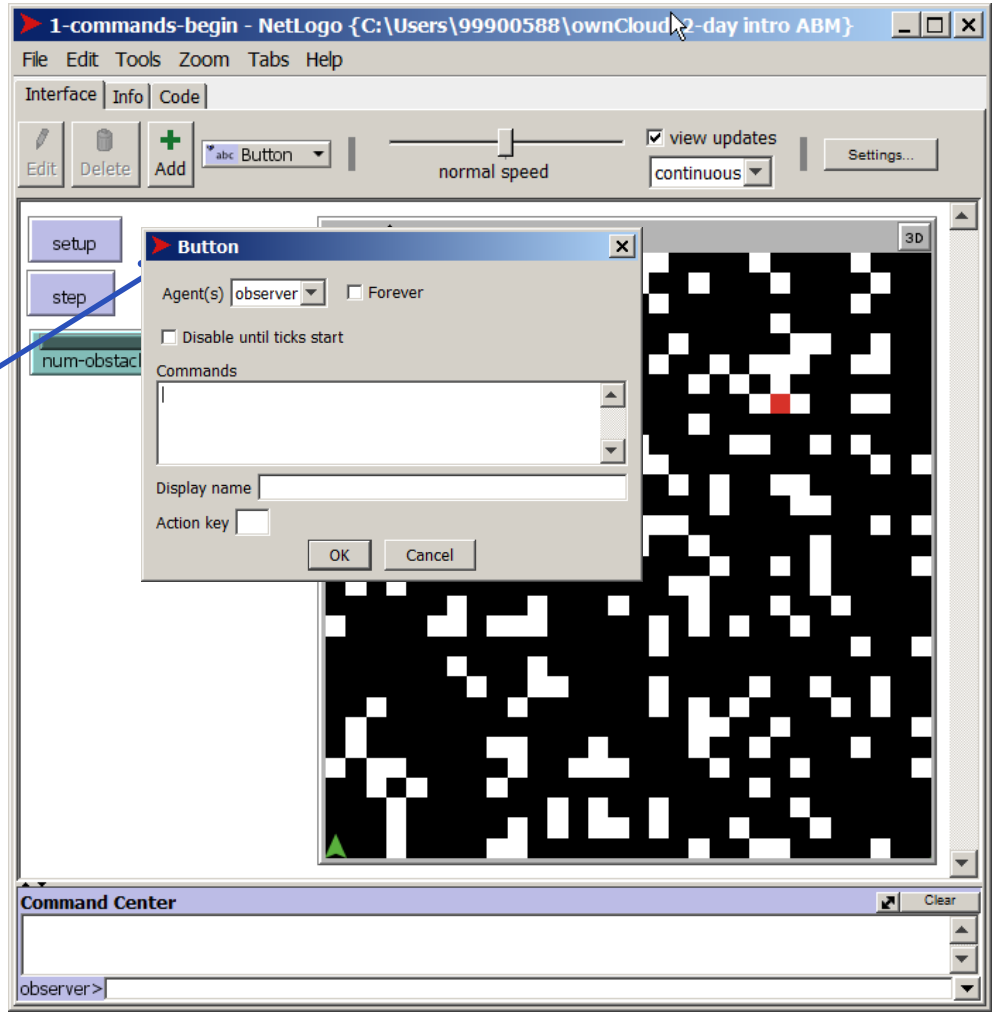
Adding a button and running the code (the fast way!)

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**”

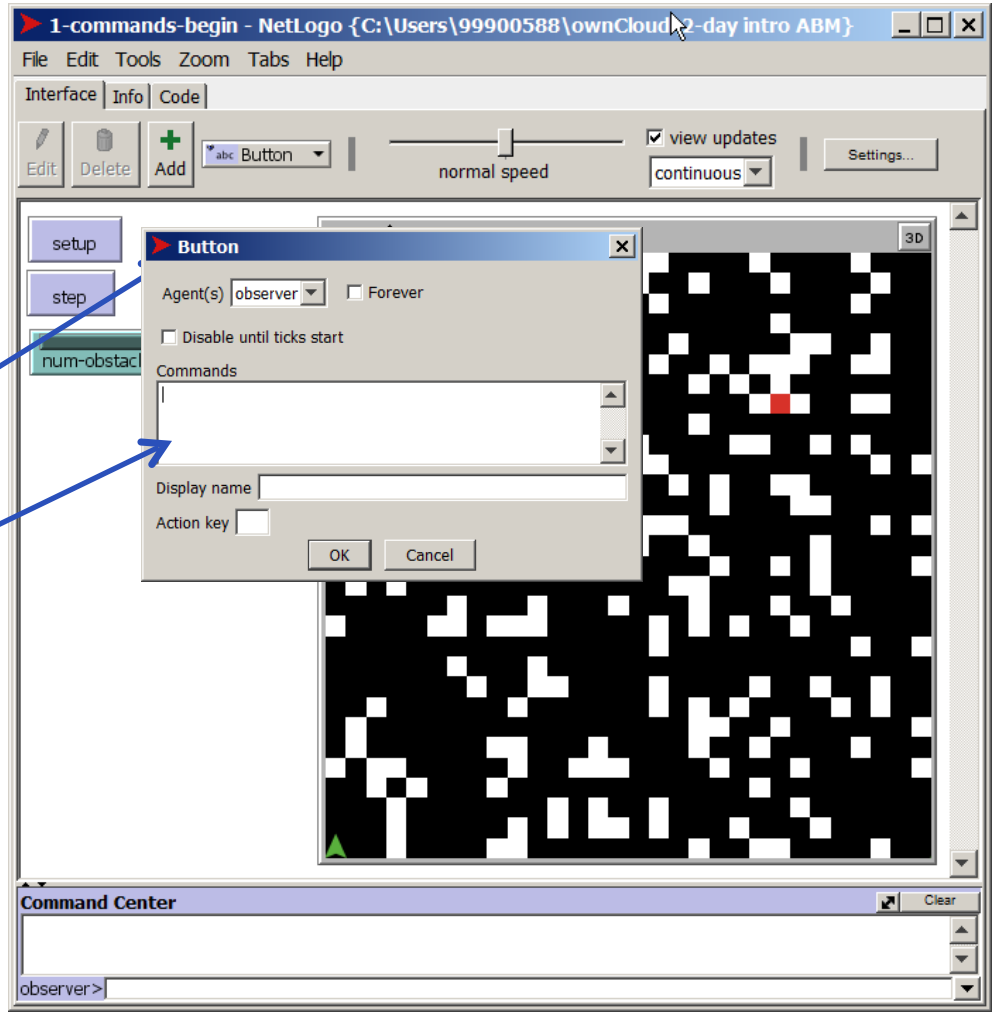


Manchester Metropolitan University

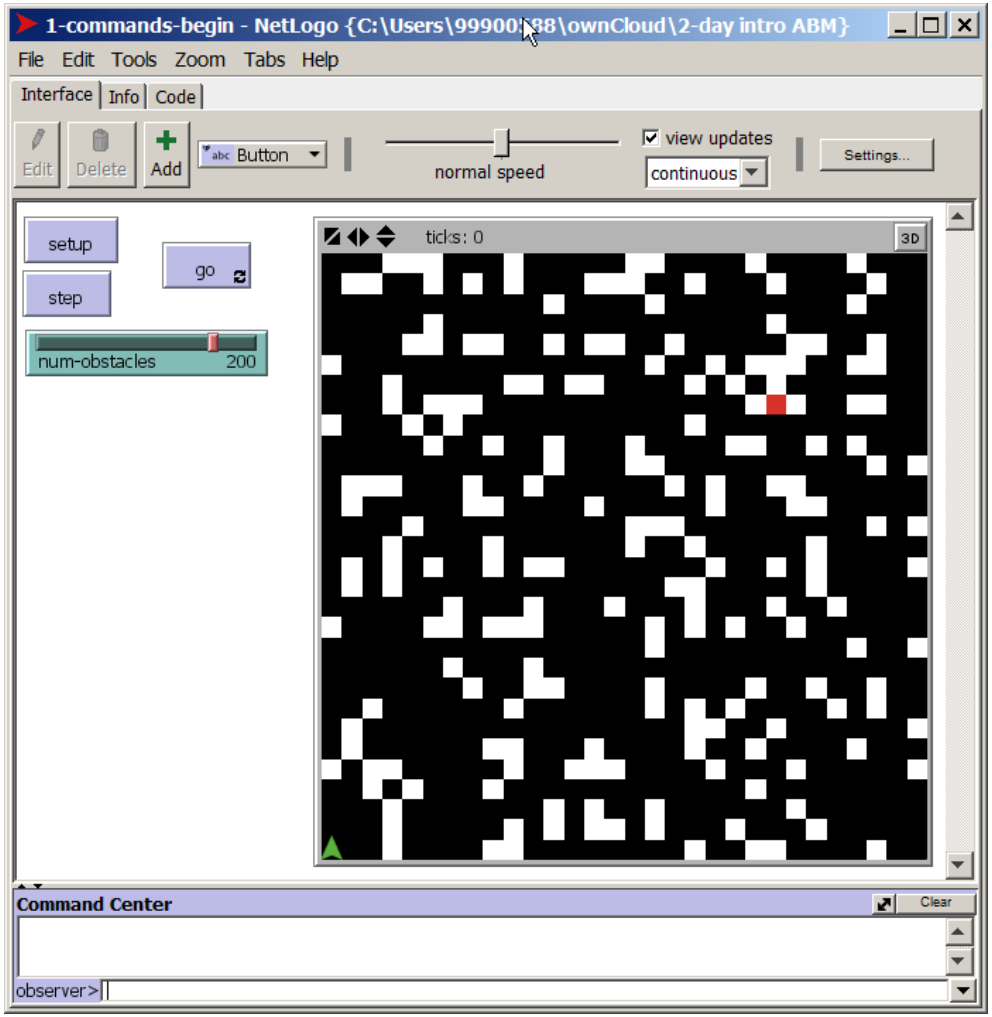
Adding a button and running the code (the fast way!)

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

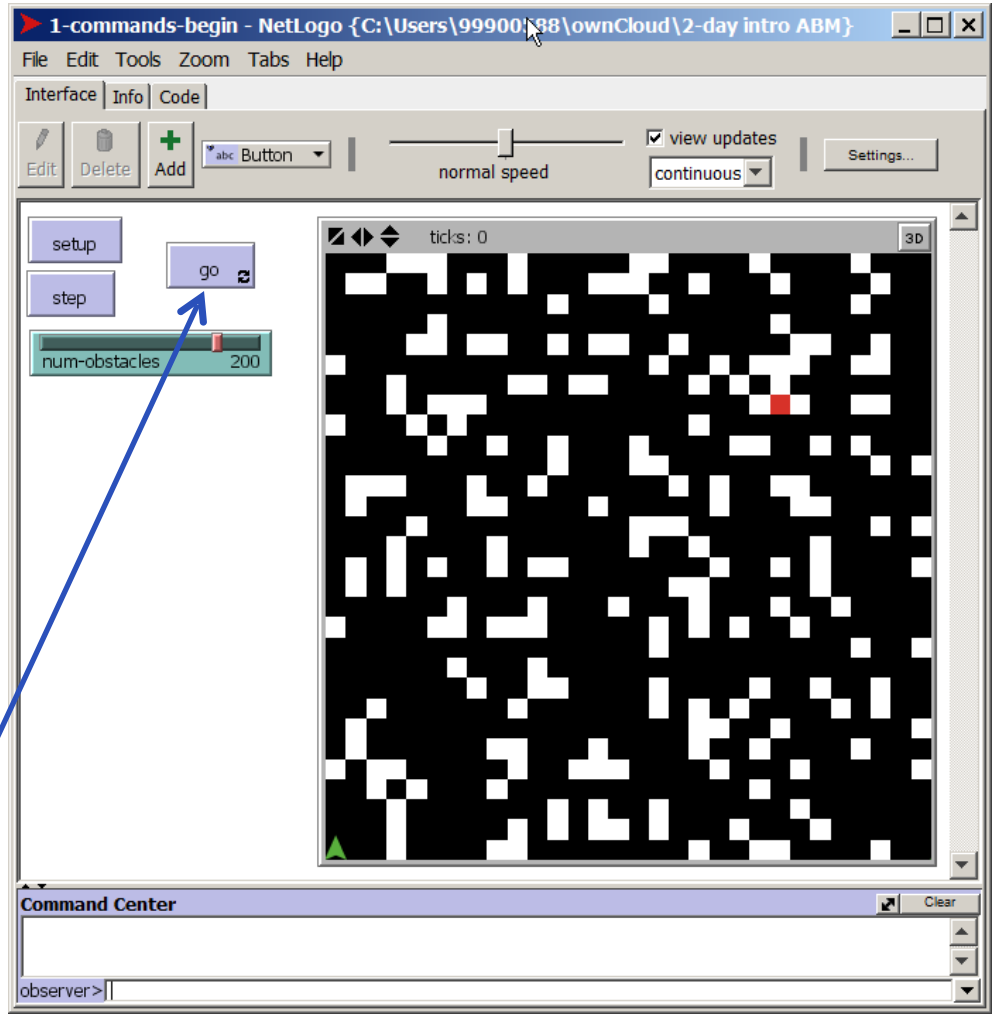
Type the text “go” here and then check (to on) the “forever” switch then “**OK**”



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



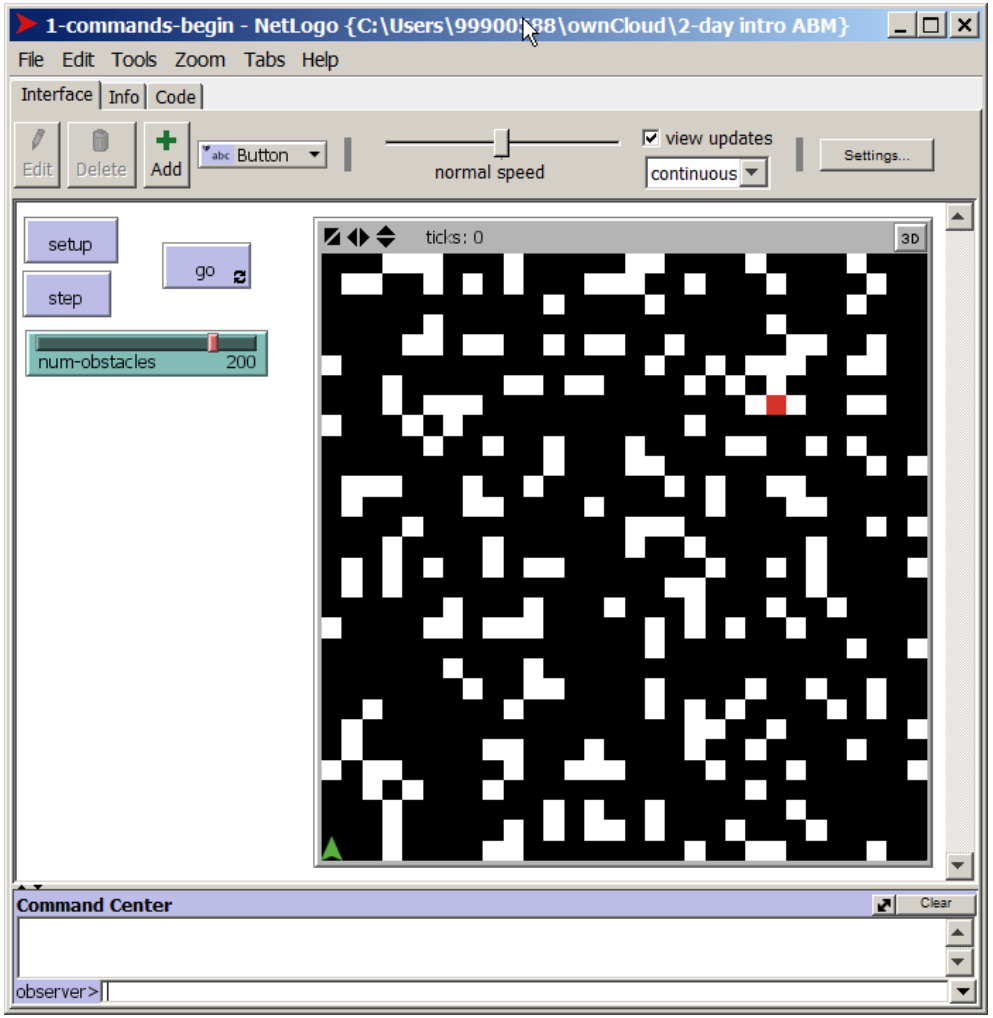
Adding a button and running the code (the fast way!)



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

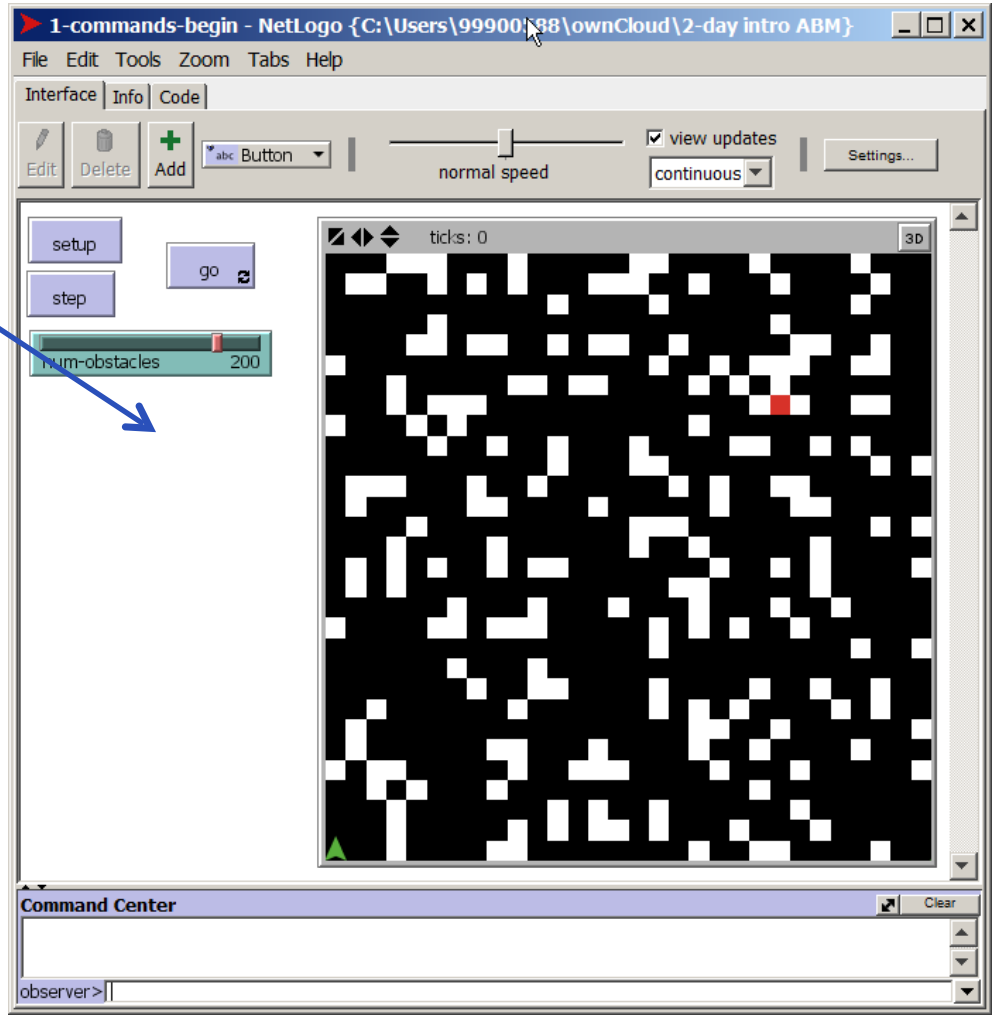
Manchester Metropolitan University

Adding a button and running the code for only 10 steps

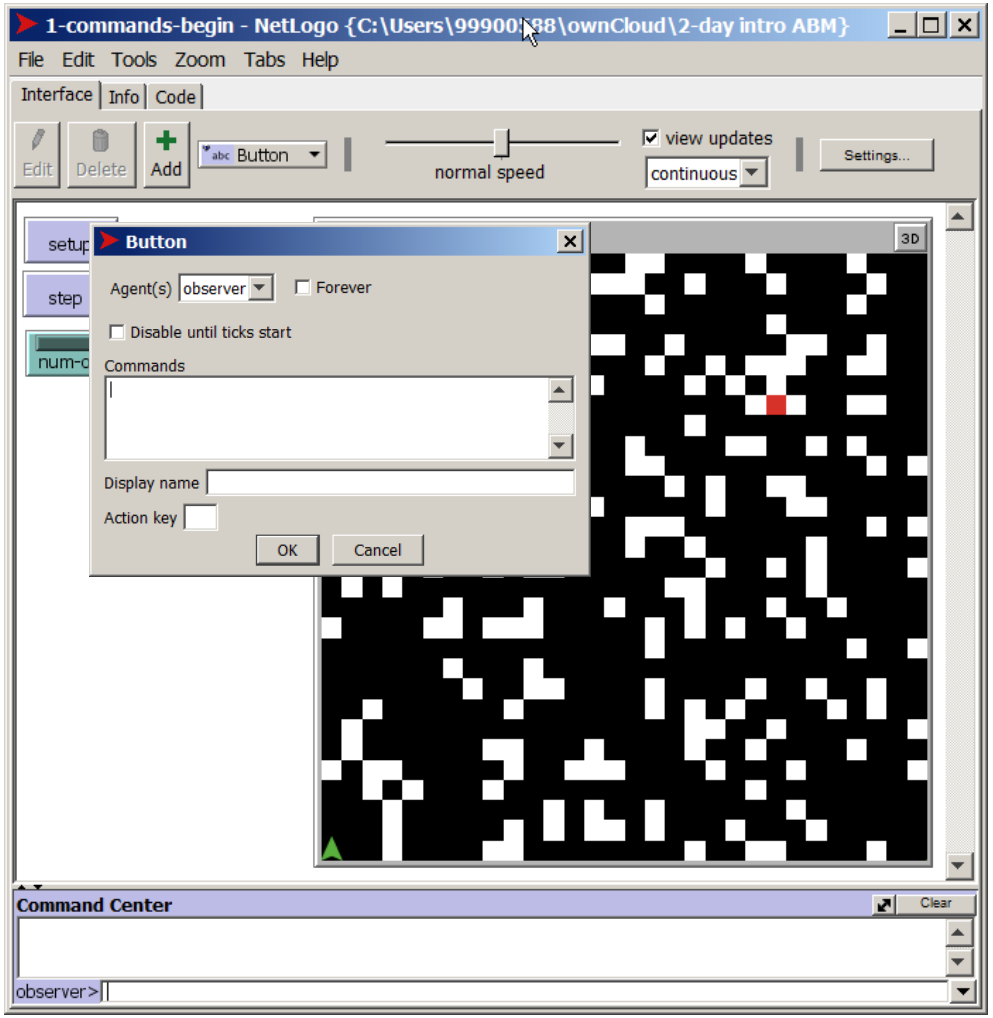


Adding a button and running the code for only 10 steps

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

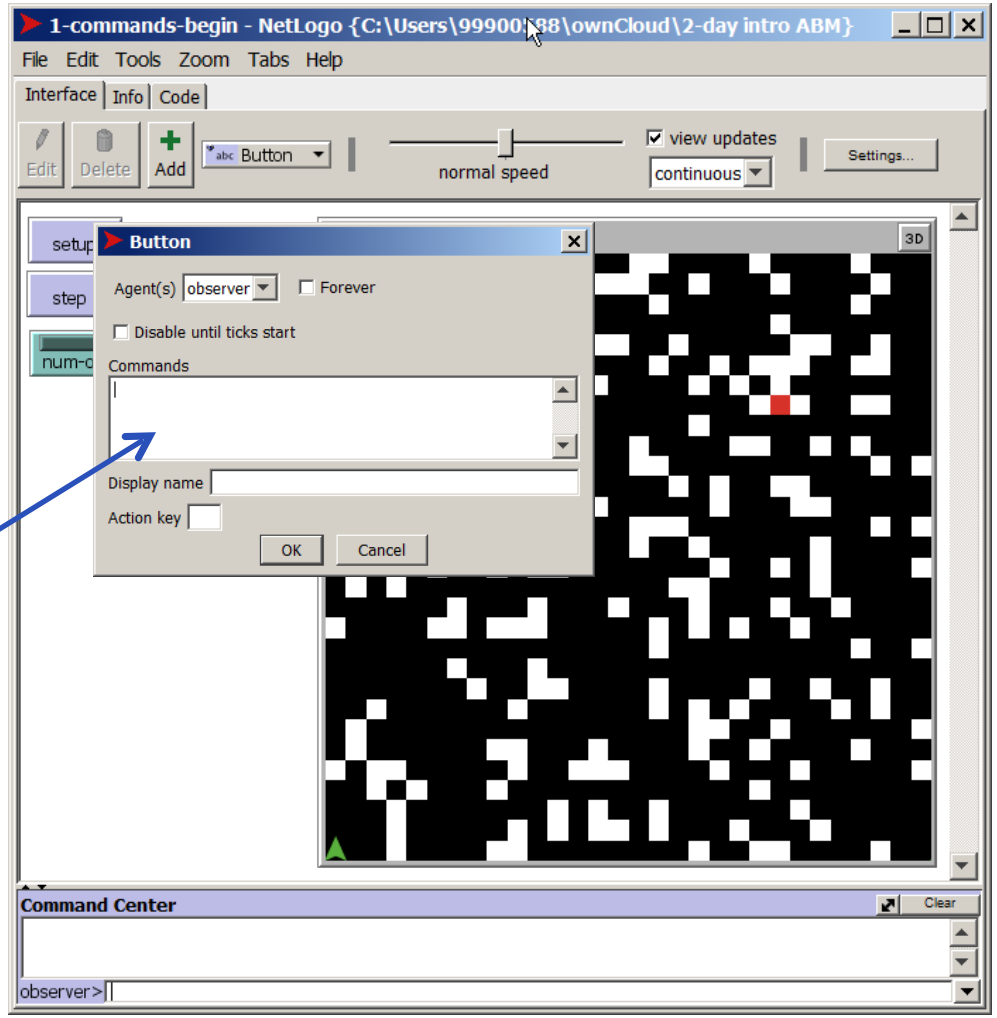


Adding a button and running the code for only 10 steps



Adding a button and running the code for only 10 steps

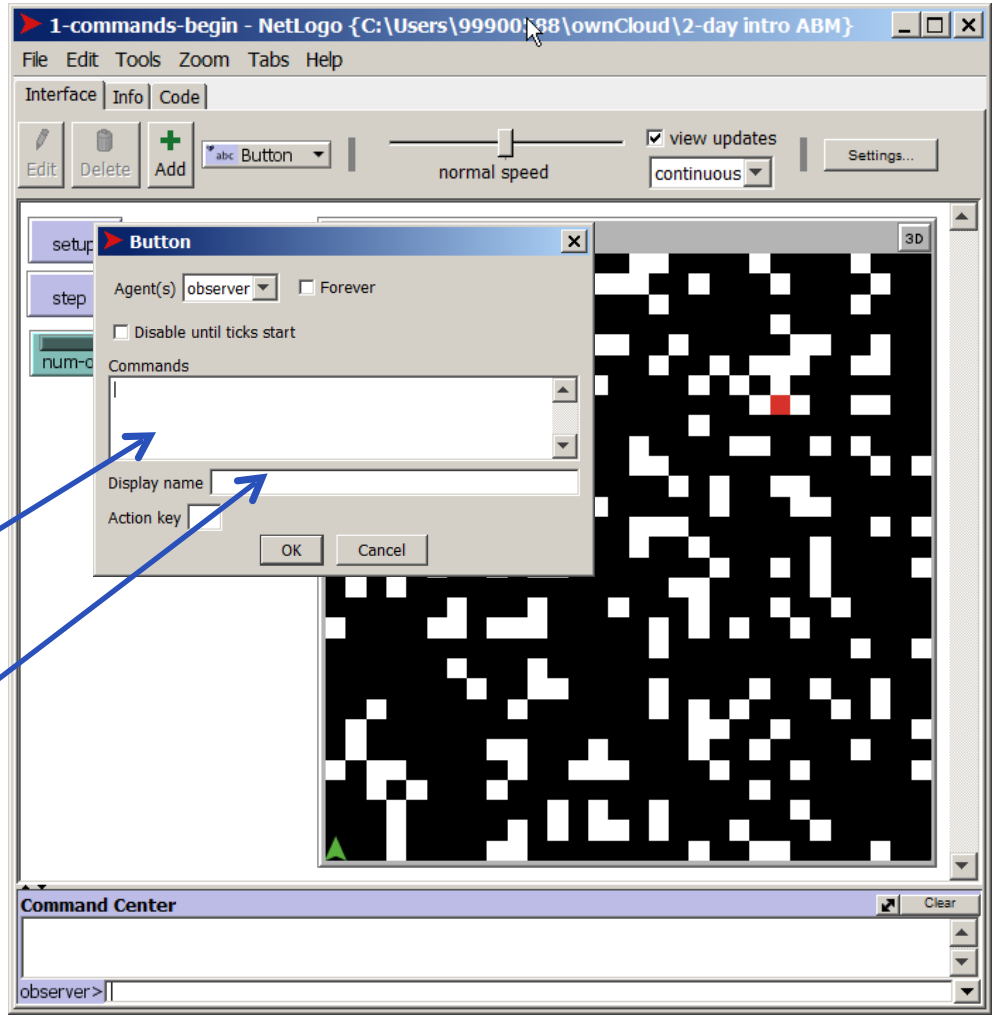
Type the text “repeat 10 [go]” here



Adding a button and running the code for only 10 steps

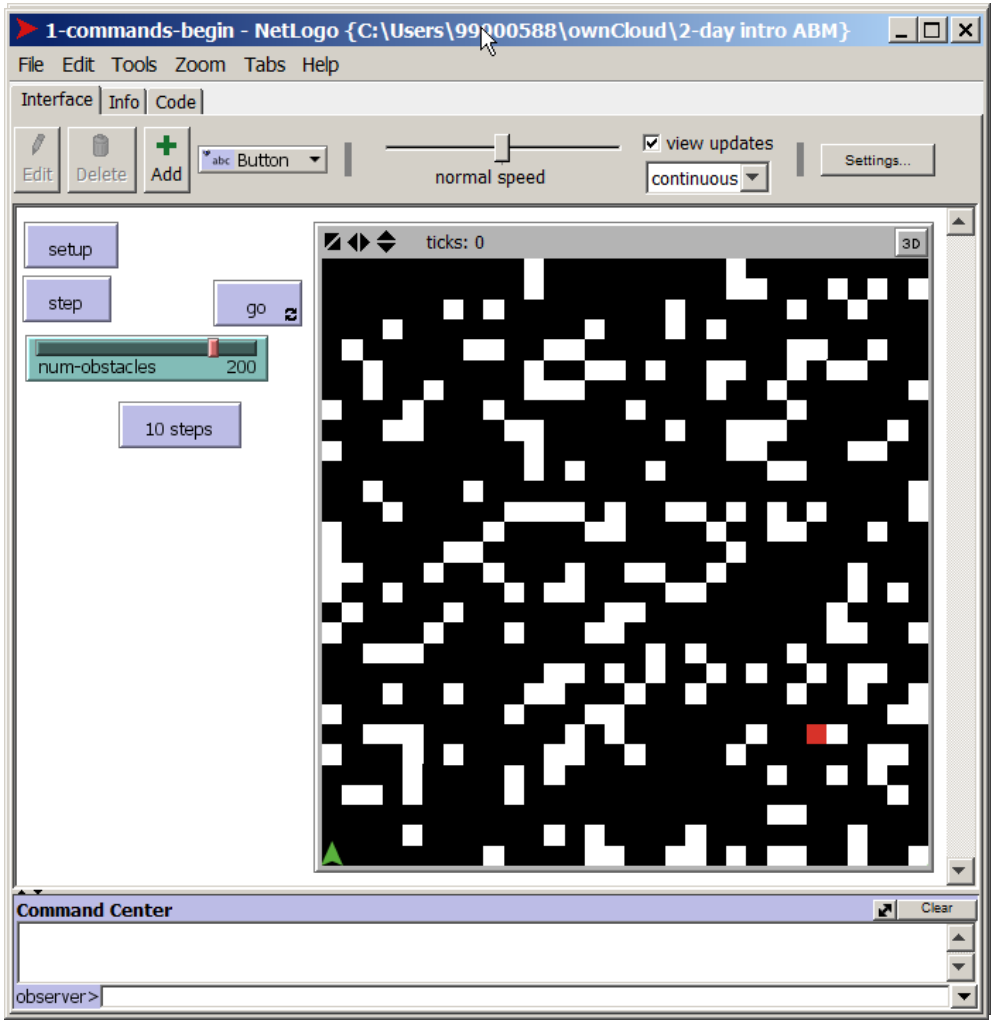
Type the text "repeat 10 [go]" here

Type the text "***10 steps***" here and then "**OK**"

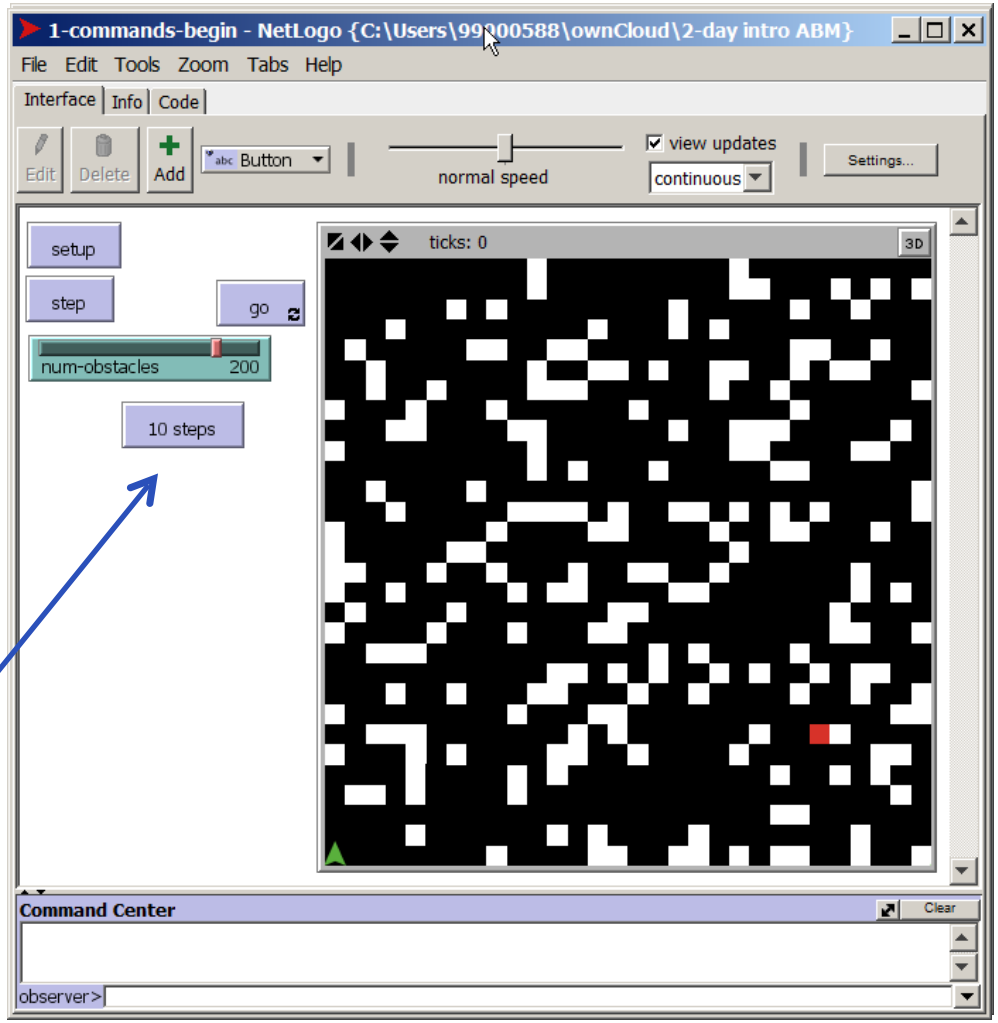


Manchester Metropolitan University

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

Manchester Metropolitan University