2-Day Introduction to Agent-Based Modelling

Day 1: Session 2

NetLogo Style, Documentation, kinds of agents, reacting to other agents

Manchester Metropolitan University

NetLogo Philosophy and Style

- Logo was originally designed by *Seymour Papert* (see his book "Mindstorms") who was a student of *Piaget*
- Underneath it is based on LISP, an old AI language that does everything with lists rather than numbers (but it does these too)
- Logo was originally designed as an environment for maths/computing exploration and creativity
- But NetLogo has been greatly extended to be an all-round simulation environment
- Instead of a few constructions which one uses to build everything, NetLogo has a *large* vocabulary of built-in "primitives" (the words built into NetLogo), so learning it is more like learning a natural language
- One programs by defining new procedures and functions using the "to… end" construct, which makes a new command in terms of a list of existing commands, which you can then use to make define further commands etc.
- So you essentially extend the built-in NetLogo primitives to make your own language
- Originally the agent was a physical robot on the floor which looked like a turtle, hence why agents are called turtles in NetLogo!

This means that…

…like a language there will be several phases you will go through:

1. Learning the syntax and basic words, where you are struggling to say anything, it seems confusing and you are a bit lost
2. Where you have some understanding of how to say some things, but are constantly looking things up and reading the manual to learn new bits, looking at other models for tips
3. Increasing expertise where the focus shifts to how to solve a programming problem, but one is still sometimes stumped due to things one did not understand and confused by one's own code!
4. Where it all just works – apparently this is a nice phase to be in, it is just that I have never met anyone who has reached it!

The NetLogo documentation

- NetLogo has a thorough documentation with (relatively) good explanations
- You will need to keep referring to it to get a handle on what it can do and does
- Even experienced programmers will not know it all, but are also referring to its documentation, learning new things
- To see the documentation:
  - Choose "**Help >> NetLogo User Manual**" from within Netlogo
  - or via http://ccl.northwestern.edu/netlogo/docs/

The Main page

The Main page

A simple, walk-through tutorial

The Main page

General introductions to features – good to browse, especially look at the "**Programming Guide**" to understand how NetLogo does things

The Main page

Advanced stuff, only read if you have got the basics and need bits from here

The Main page

But **THIS** is what you will keep referring to… the dictionary of all the Netlogo commands. ***Please click on this***

The NetLogo Dictionary

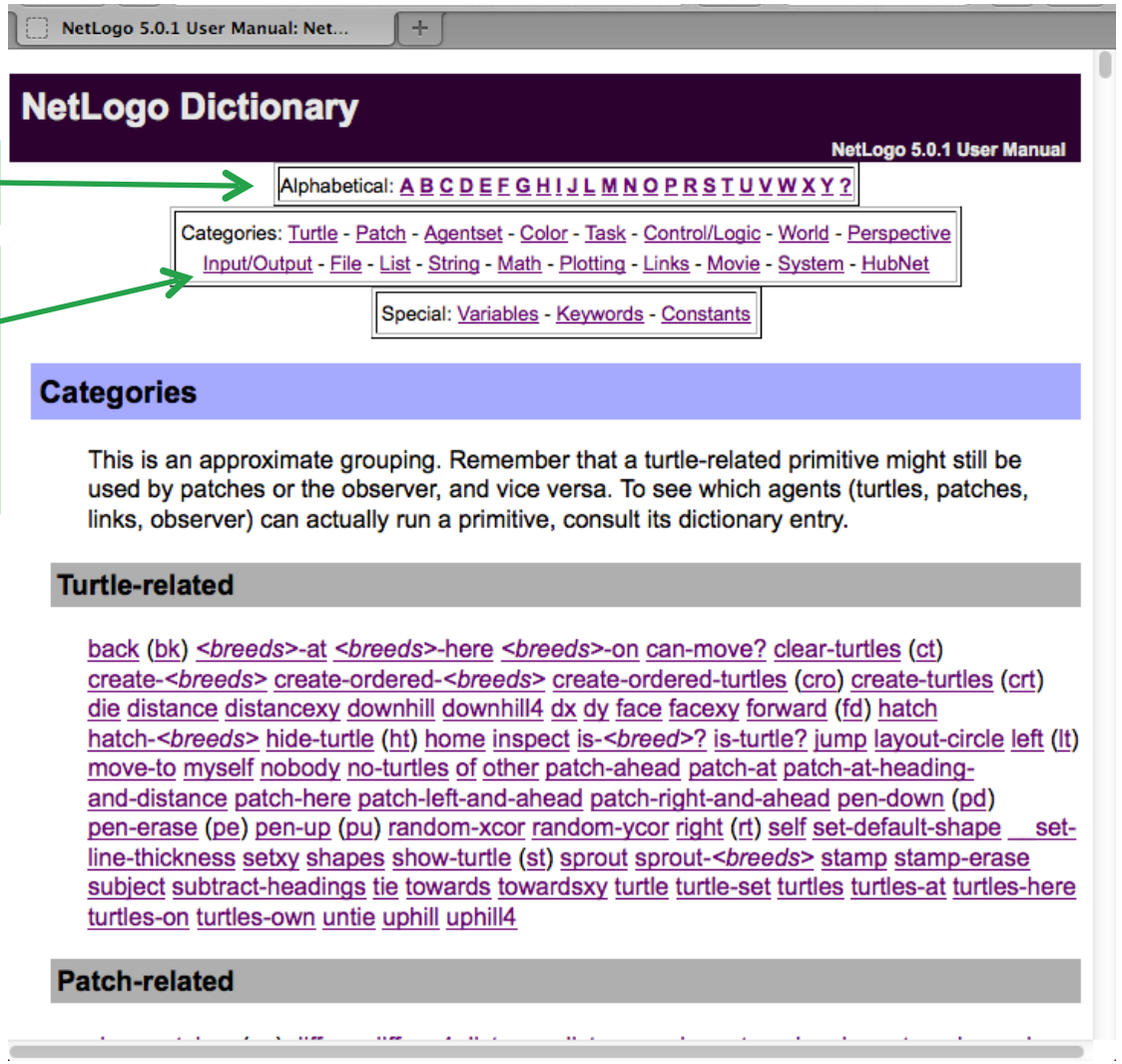
The NetLogo Dictionary

Alphabetic Index to Primitives

The NetLogo Dictionary

Alphabetic Index to Primitives

Primitives by functional category – good if you do not know the exact primitive you are looking for

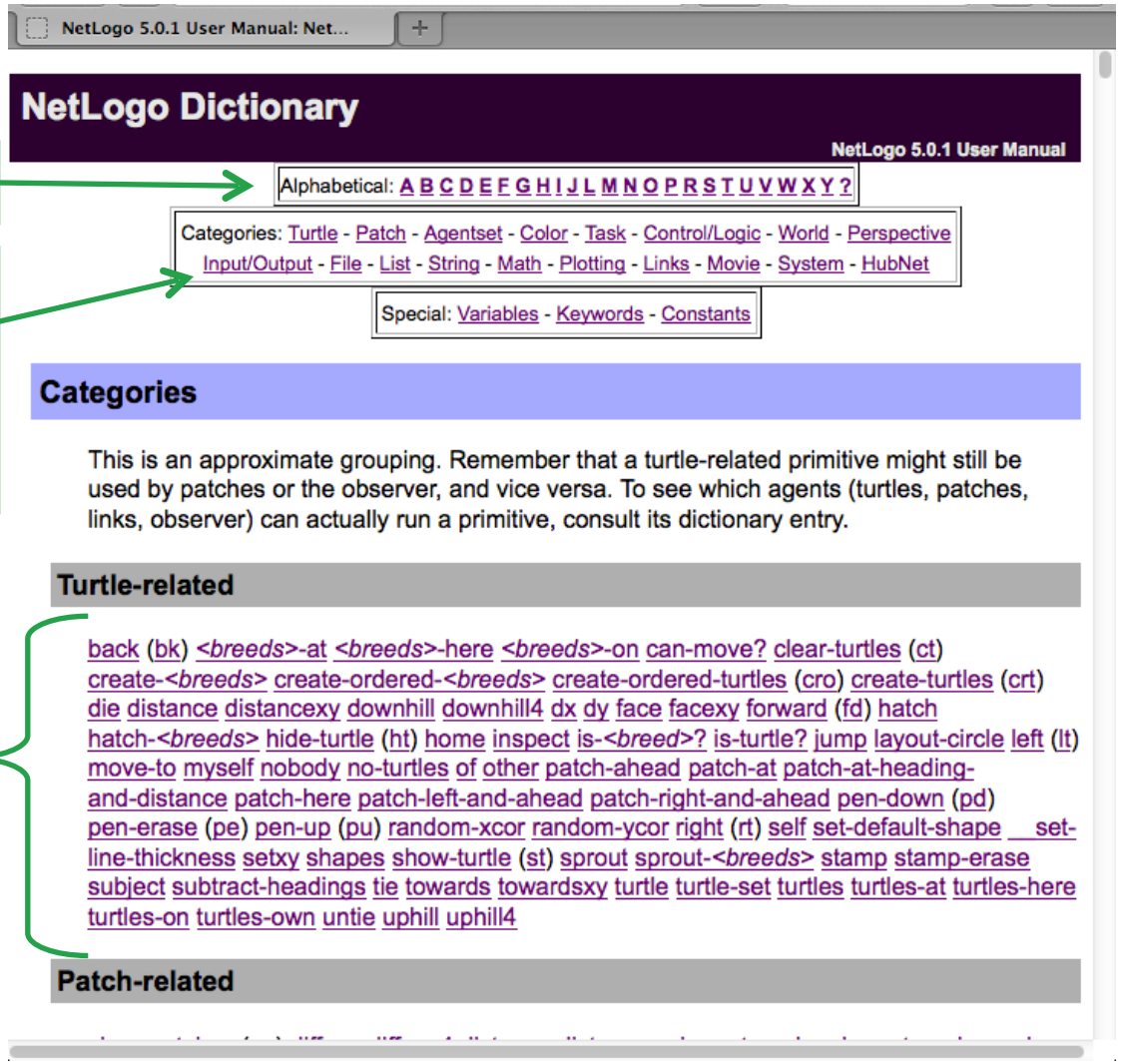


The NetLogo Dictionary

Alphabetic Index to Primitives

Primitives by functional category – good if you do not know the exact primitive you are looking for

Each category has a list of primitives to click on – this takes you to the definition with examples



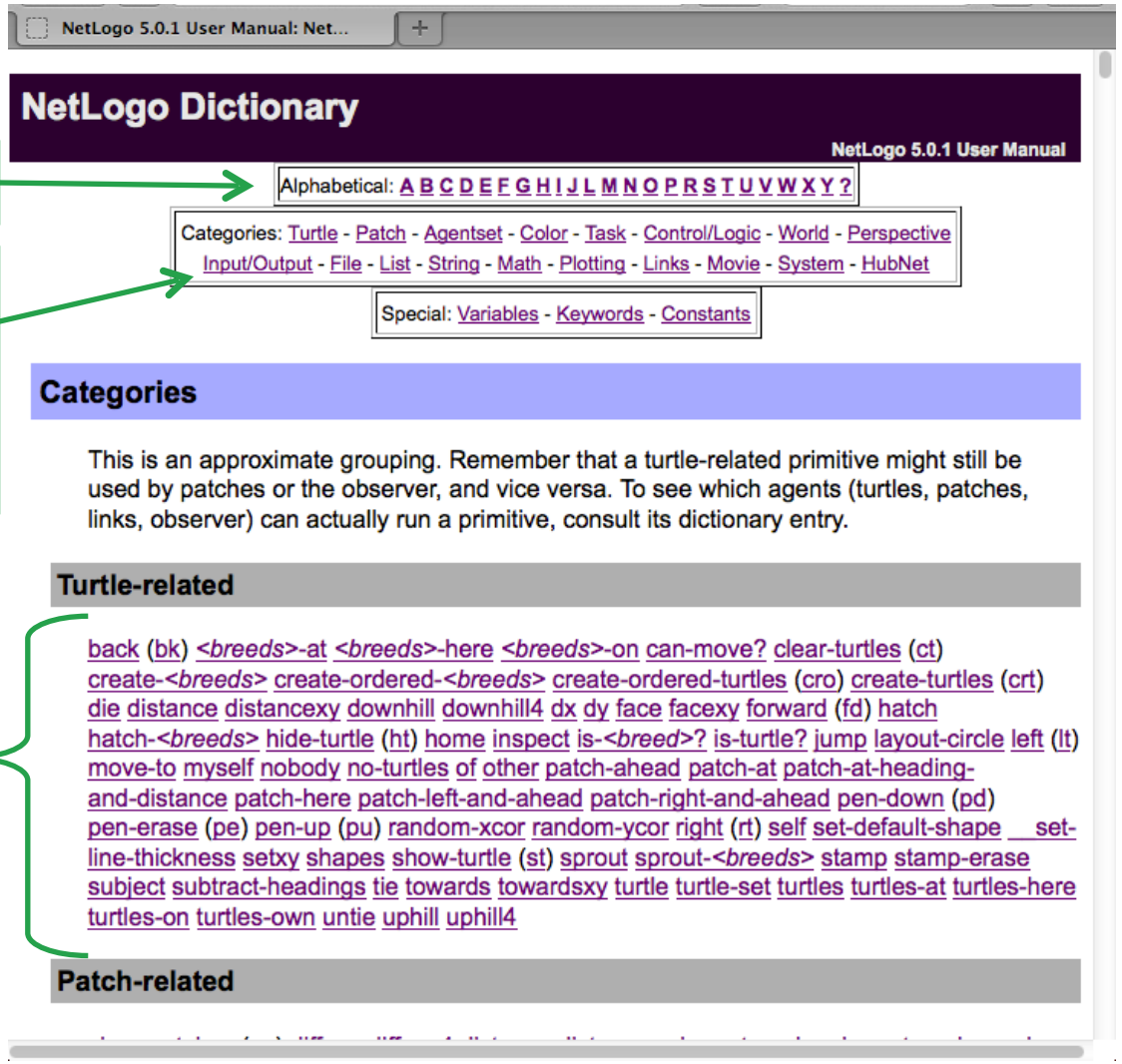
The NetLogo Dictionary

Alphabetic Index to Primitives

Primitives by functional category – good if you do not know the exact primitive you are looking for

Each category has a list of primitives to click on – this takes you to the definition with examples

Click on "**Control/Logic**" then "**ask**"…



An example “definition”

NetLogo 5.0.1 User Manual: Net...

ask

ask *agentset* [*commands*]
ask *agent* [*commands*]

The specified agent or agentset runs the given commands.

```
ask turtles [ fd 1 ]
  ;; all turtles move forward one step
ask patches [ set pcolor red ]
  ;; all patches turn red
ask turtle 4 [ rt 90 ]
  ;; only the turtle with id 4 turns right
```

Note: only the observer can ask all turtles or all patches. This prevents you from inadvertently having all turtles ask all turtles or all patches ask all patches, which is a common mistake to make if you're not careful about which agents will run the code you are writing.

Note: Only the agents that are in the agentset *at the time the ask begins* run the commands.

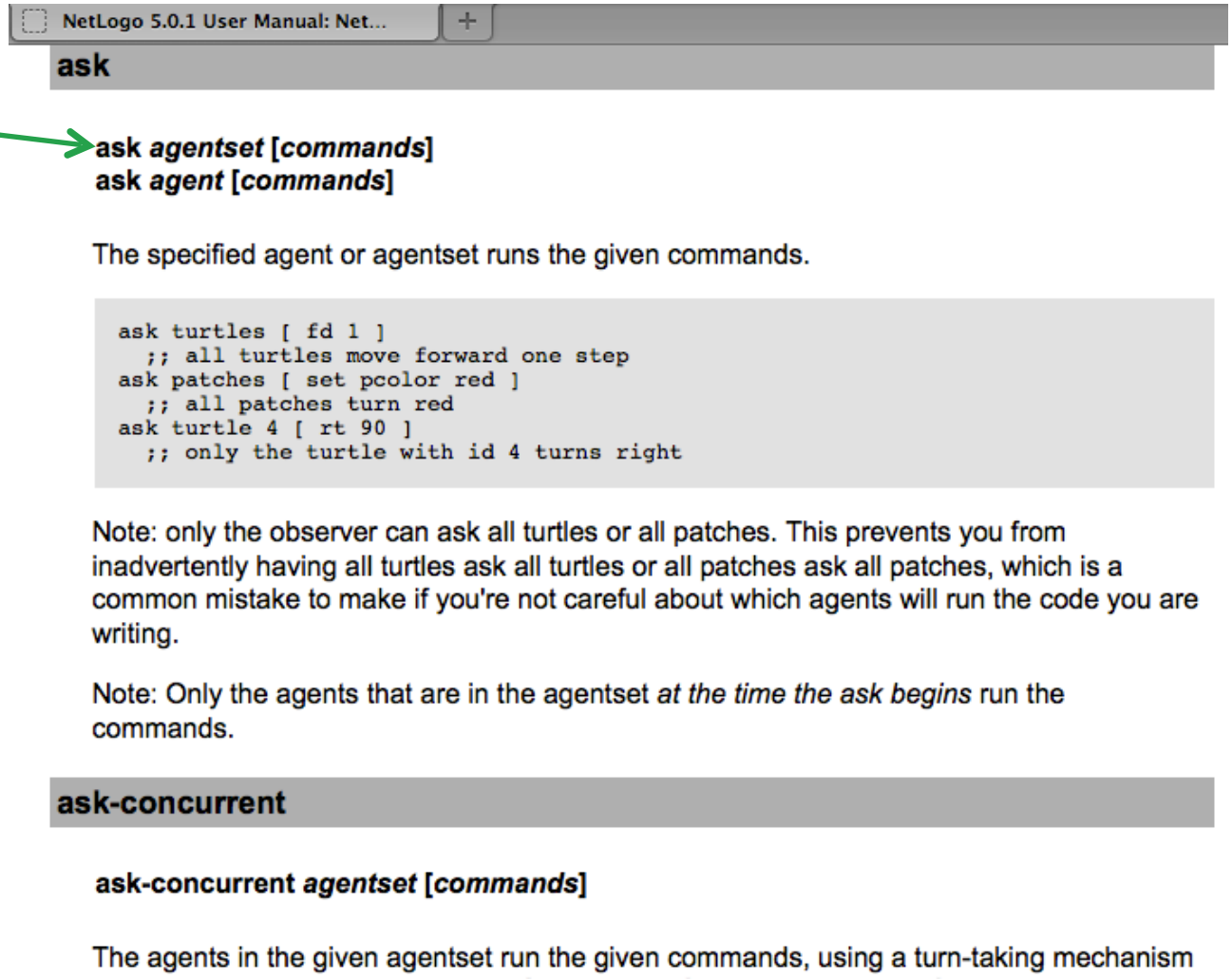
ask-concurrent

ask-concurrent *agentset* [*commands*]

The agents in the given agentset run the given commands, using a turn-taking mechanism

An example “definition”

The syntax of the primitive



NetLogo 5.0.1 User Manual: Net...

ask

ask *agentset* [*commands*]
ask *agent* [*commands*]

The specified agent or agentset runs the given commands.

```
ask turtles [ fd 1 ]
  ;; all turtles move forward one step
ask patches [ set pcolor red ]
  ;; all patches turn red
ask turtle 4 [ rt 90 ]
  ;; only the turtle with id 4 turns right
```

Note: only the observer can ask all turtles or all patches. This prevents you from inadvertently having all turtles ask all turtles or all patches ask all patches, which is a common mistake to make if you're not careful about which agents will run the code you are writing.

Note: Only the agents that are in the agentset *at the time the ask begins* run the commands.

ask-concurrent

ask-concurrent *agentset* [*commands*]

The agents in the given agentset run the given commands, using a turn-taking mechanism

An example “definition”

The syntax of the primitive

A brief explanation of the primitive

NetLogo 5.0.1 User Manual: Net...

ask

ask *agentset* [*commands*]
ask *agent* [*commands*]

The specified agent or agentset runs the given commands.

```
ask turtles [ fd 1 ]
  ;; all turtles move forward one step
ask patches [ set pcolor red ]
  ;; all patches turn red
ask turtle 4 [ rt 90 ]
  ;; only the turtle with id 4 turns right
```

Note: only the observer can ask all turtles or all patches. This prevents you from inadvertently having all turtles ask all turtles or all patches ask all patches, which is a common mistake to make if you're not careful about which agents will run the code you are writing.

Note: Only the agents that are in the agentset *at the time the ask begins* run the commands.

ask-concurrent

ask-concurrent *agentset* [*commands*]

The agents in the given agentset run the given commands, using a turn-taking mechanism

An example “definition”

The syntax of the primitive

A brief explanation of the primitive

Some examples of the primitive in use – ***these are particularly useful!***

NetLogo 5.0.1 User Manual: Net...

ask

ask *agentset* [*commands*]
ask *agent* [*commands*]

The specified agent or agentset runs the given commands.

```
ask turtles [ fd 1 ]
  ;; all turtles move forward one step
ask patches [ set pcolor red ]
  ;; all patches turn red
ask turtle 4 [ rt 90 ]
  ;; only the turtle with id 4 turns right
```

Note: only the observer can ask all turtles or all patches. This prevents you from inadvertently having all turtles ask all turtles or all patches ask all patches, which is a common mistake to make if you're not careful about which agents will run the code you are writing.

Note: Only the agents that are in the agentset *at the time the ask begins* run the commands.

ask-concurrent

ask-concurrent *agentset* [*commands*]

The agents in the given agentset run the given commands, using a turn-taking mechanism

An example "definition"

The syntax of the primitive

A brief explanation of the primitive

Some examples of the primitive in use – ***these are particularly useful!***

Notes – these explain potential "gotchas" and common mistakes

NetLogo 5.0.1 User Manual: Net...

ask

ask *agentset* [*commands*]
ask *agent* [*commands*]

The specified agent or agentset runs the given commands.

```
ask turtles [ fd 1 ]
  ;; all turtles move forward one step
ask patches [ set pcolor red ]
  ;; all patches turn red
ask turtle 4 [ rt 90 ]
  ;; only the turtle with id 4 turns right
```

Note: only the observer can ask all turtles or all patches. This prevents you from inadvertently having all turtles ask all turtles or all patches ask all patches, which is a common mistake to make if you're not careful about which agents will run the code you are writing.

Note: Only the agents that are in the agentset *at the time the ask begins* run the commands.

ask-concurrent

ask-concurrent *agentset* [*commands*]

The agents in the given agentset run the given commands, using a turn-taking mechanism

An example “definition”

The syntax of the primitive

A brief explanation of the primitive

Some examples of the primitive in use – ***these are particularly useful!***

Notes – these explain potential “gotchas” and common mistakes

NetLogo 5.0.1 User Manual: Net...

ask

ask *agentset* [*commands*]
ask *agent* [*commands*]

The specified agent or agentset runs the given commands.

```
ask turtles [ fd 1 ]
  ;; all turtles move forward one step
ask patches [ set pcolor red ]
  ;; all patches turn red
ask turtle 4 [ rt 90 ]
  ;; only the turtle with id 4 turns right
```

Note: only the observer can ask all turtles or all patches. This prevents you from inadvertently having all turtles ask all turtles or all patches ask all patches, which is a common mistake to make if you're not careful about which agents will run the code you are writing.

Note: Only the agents that are in the agentset *at the time the ask begins* run the commands.

ask-concurrent

ask-concu

The agents in the given agentset run the given commands, using a turn-taking mechanism

Try looking up the primitives: “**to**”, “**set**”, and “**if**”

Types of Agent

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

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

  create-people 1 [… *some commands* …]
- As well as being referred to directly, e.g.

  ask people [… *some commands* …]

Other Declarations

Interface | Info | Code

Find... Check | Procedures ▾ | ☑ Indent automatically

```
;; text, like this, that start with semi-colons are comments and do no
;;
;; First we have lists of general and individual properties/slots

globals [people-colours number-kinds me]

breed [people person]
breed [others another]

;; only attribute is age, all agents automatically have the attribute
others-own [age]
people-own [age]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  checkerboard-patches ;; colours backgroud (see below for code)
```


Manchester Metropolitan University

Other Declarations

Load the NetLogo model:
"**2-friends-begin.nlogo**"
and select the "**Code**" tab

```
;; like this, that start with semi-colons are comments and do no
;;
;; First we have lists of general and individual properties/slots

globals [people-colours number-kinds me]

breed [people person]
breed [others another]

;; only attribute is age, all agents automatically have the attribute
others-own [age]
people-own [age]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  checkerboard-patches ;; colours backgroud (see below for code)
```

Other Declarations

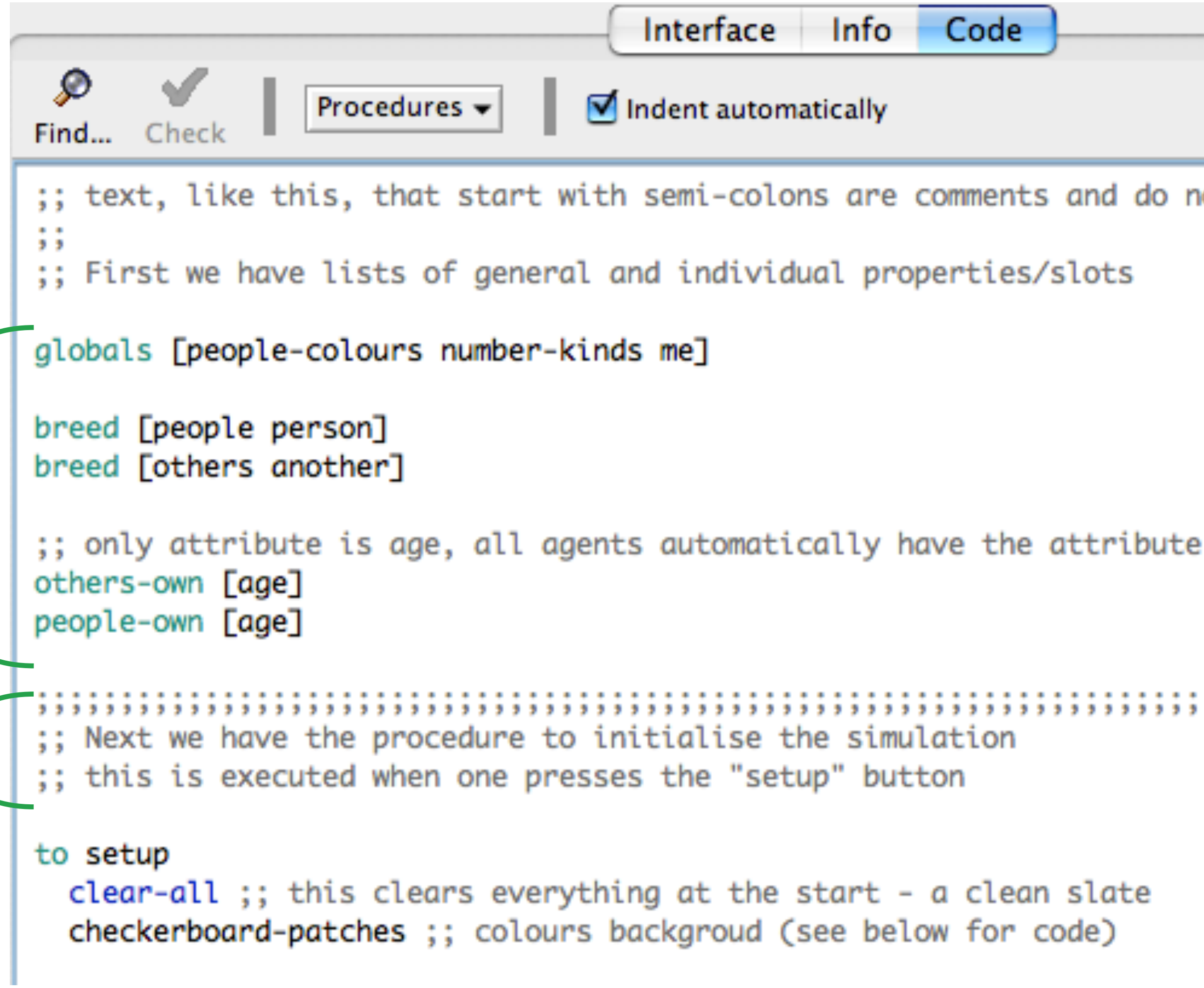
Interface | Info | Code

Find... Check | Procedures | Indent automatically

These are the various declarations

```
;; text, like this, that start with semi-colons are comments and do no
;;
;; First we have lists of general and individual properties/slots

globals [people-colours number-kinds me]

breed [people person]
breed [others another]

;; only attribute is age, all agents automatically have the attribute
others-own [age]
people-own [age]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  checkerboard-patches ;; colours backgroud (see below for code)
```

Other Declarations

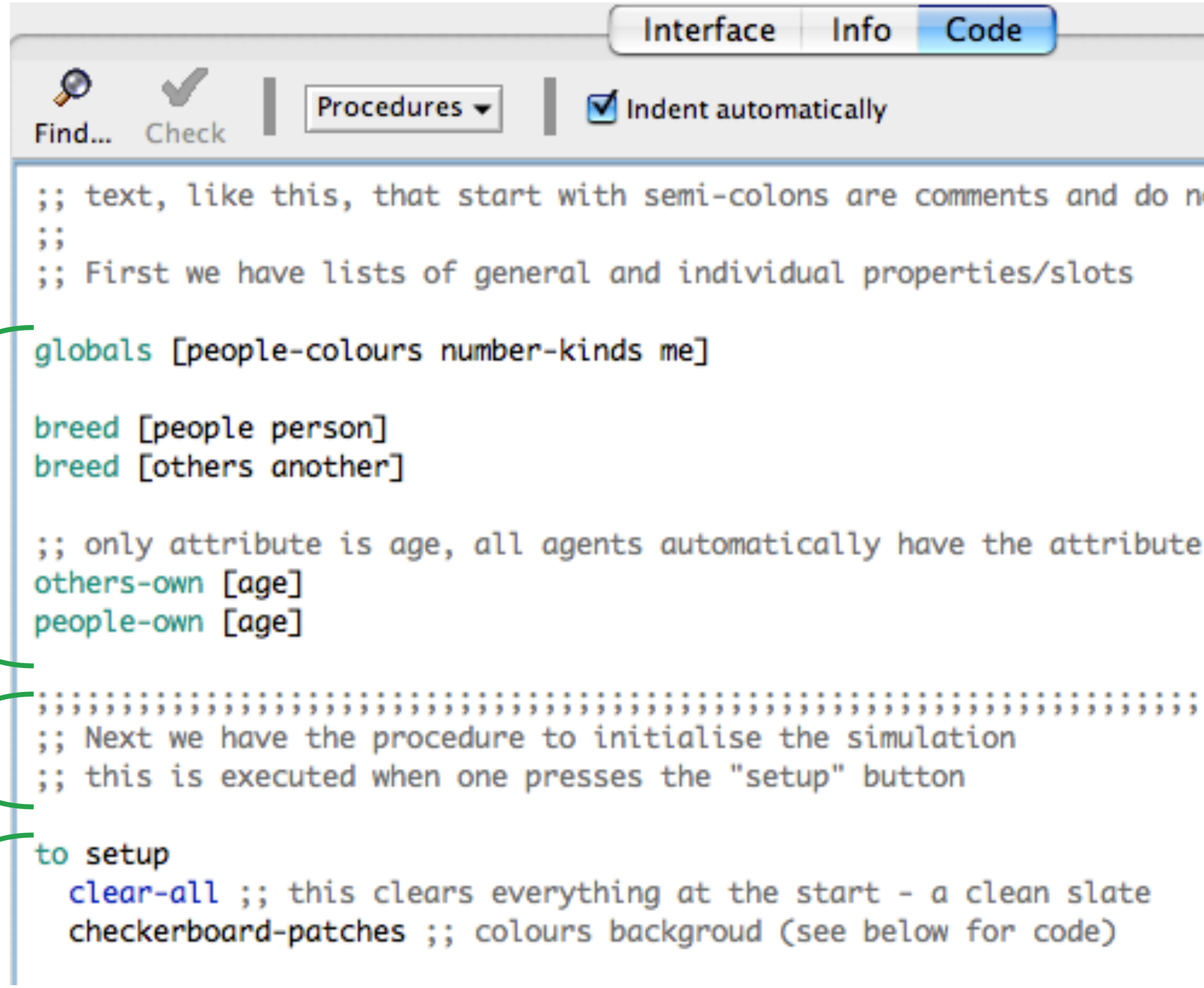


These are the various declarations

These just comments to help you understand the code

Manchester Metropolitan University

Other Declarations



These are the various declarations

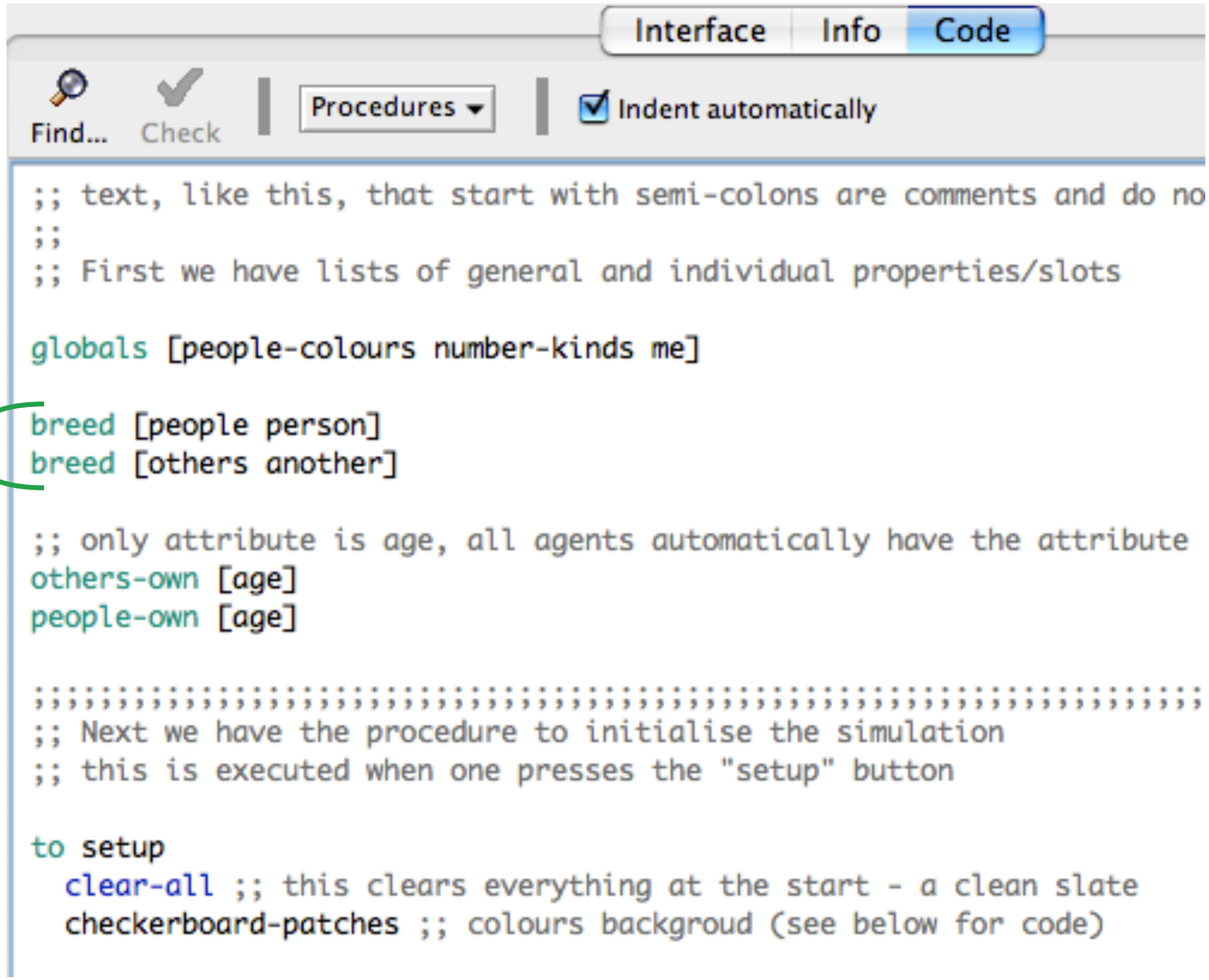
These just comments to help you understand the code

The code – the procedure definitions are here onwards

Manchester Metropolitan University

Other Declarations

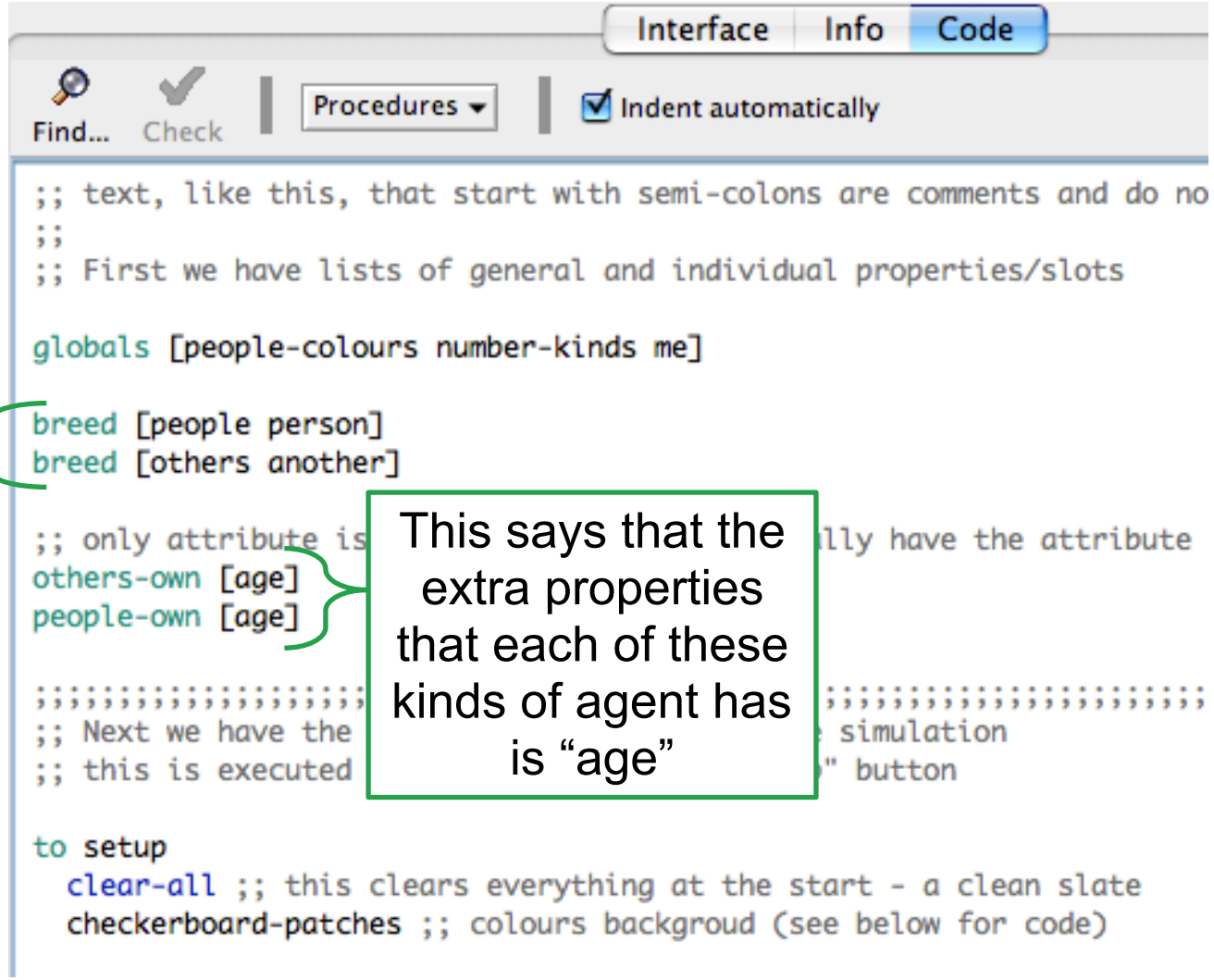
Two kinds of agent are defined: "**people**" and "**others**"



Manchester Metropolitan University

Other Declarations

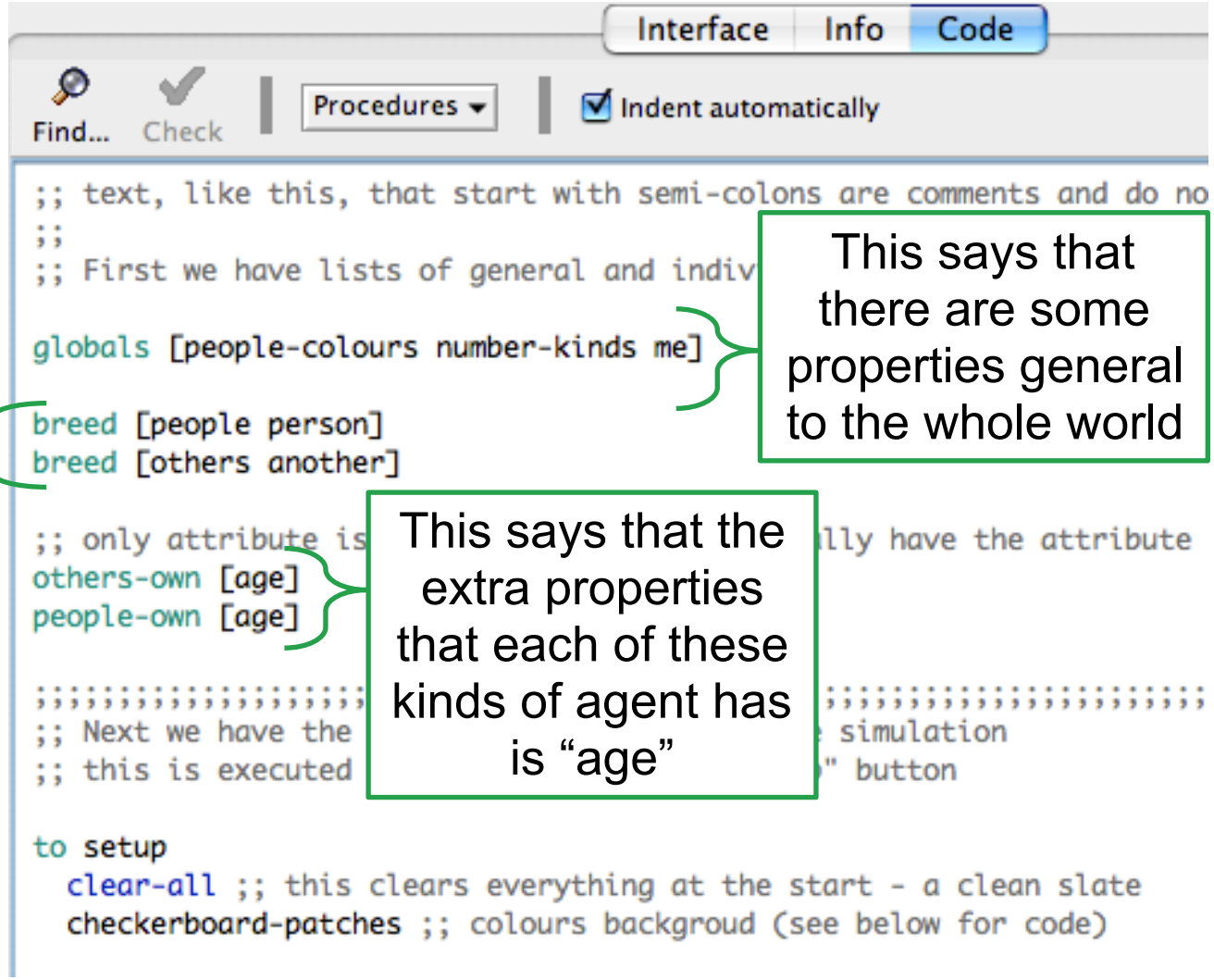
Two kinds of agent are defined: "**people**" and "**others**"



This says that the extra properties that each of these kinds of agent has is "age"

Other Declarations

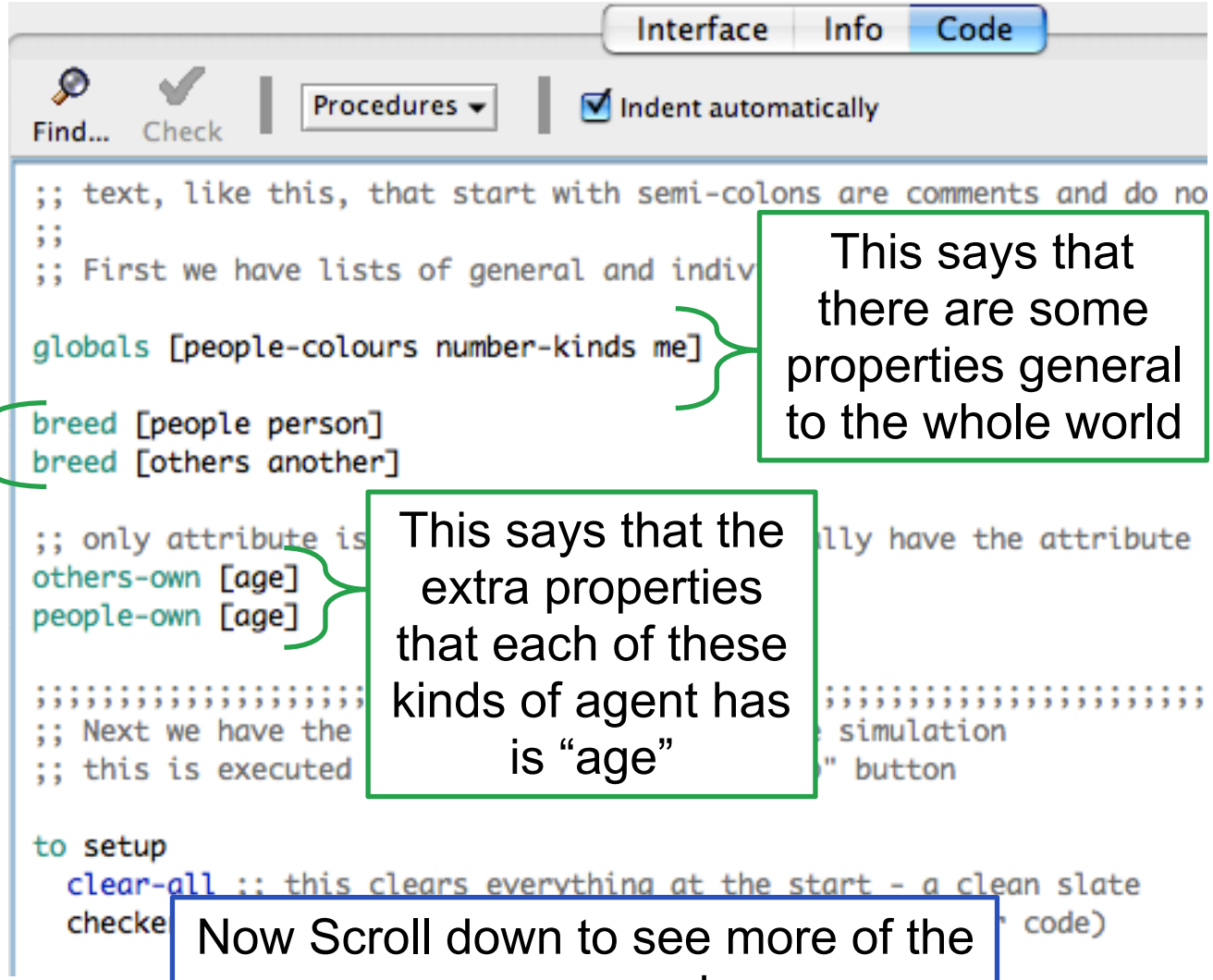
Two kinds of agent are defined: "**people**" and "**others**"



This says that there are some properties general to the whole world

This says that the extra properties that each of these kinds of agent has is "age"

Other Declarations



Two kinds of agent are defined: "**people**" and "**others**"

This says that there are some properties general to the whole world

This says that the extra properties that each of these kinds of agent has is "age"

Now Scroll down to see more of the program code

The “setup” procedure

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  checkerboard-patches ;; colours backgroud (see below for code)

  ;; the list of colors objects can be
  set people-colours [red blue green yellow orange violet cyan brown magenta white]
  set number-kinds length people-colours ;; records how many colors there are

  ;; first create the other people, shown as circles, put them on a random patch
  ;; the parameter "population" is determined by a slider
  create-others population [
    set shape "circle"
    set age random 100
    set size age / 1000
    set color random-kind
    shift-randomly
  ]

  ;; do the same for the focus person,
  create-people 1 [
    set shape "person"
    set size 0.2
    set age 50
    set size 0.2
    set color random-kind
    shift-randomly
  ]

  reset-ticks                                       ;; this initialises the simulation time sys
end
```

The “setup” procedure

All this defines what the “**setup**” command does.

So it is what the “setup” button causes to happen when you click on it.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  checkerboard-patches ;; colours backgroud (see below for code)

  ;; the list of colors objects can be
  set people-colours [red blue green yellow orange violet cyan brown magenta white]
  set number-kinds length people-colours ;; records how many colors there are

  ;; first create the other people, shown as circles, put them on a random patch
  ;; the parameter "population" is determined by a slider
  create-others population [
    set shape "circle"
    set age random 100
    set size age / 1000
    set color random-kind
    shift-randomly
  ]

  ;; do the same for the focus person,
  create-people 1 [
    set shape "person"
    set size 0.2
    set age 50
    set size 0.2
    set color random-kind
    shift-randomly
  ]

  reset-ticks                              ;; this initialises the simulation time sys
end
```

The “setup” procedure

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Next we have the procedure t
;; this is executed when one pr

to setup
  clear-all ;; this clears ever
  checkerboard-patches ;; colou

  ;; the list of colors objects
  set people-colours [red blue green yellow orange violet cyan brown magenta white]
  set number-kinds length people-colours ;; records how many colors there are

  ;; first create the other people, shown as circles, put them on a random patch
  ;; the parameter "population" is determined by a slider
  create-others population [
    set shape "circle"
    set age random 100
    set size age / 1000
    set color random-kind
    shift-randomly
  ]

  ;; do the same for the focus person,
  create-people 1 [
    set shape "person"
    set size 0.2
    set age 50
    set size 0.2
    set color random-kind
    shift-randomly
  ]

  reset-ticks                          ;; this initialises the simulation time sys
end
```

This clears everything and then calls the procedure called “checkerboard-patches”

The “setup” procedure

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  checkerboard-patches ;; colours backgroud (see below

  ;; the list of colors objects can be
  set people-colours [red blue green yellow orange vio
  set number-kinds length people-colours ;; records ho

  ;; first create the other people, shown as circles,
  ;; the parameter "population" is determined by a sli
  create-others population [
    set shape "circle"
    set age random 100
    set size age / 1000
    set color random-kind
    shift-randomly
  ]

  ;; do the same for the focus person,
  create-people 1 [
    set shape "person"
    set size 0.2
    set age 50
    set size 0.2
    set color random-kind
    shift-randomly
  ]

  reset-ticks                         ;; this initialises the simulation time sys
end
```

This defines some global properties that may be used throughout the code

The "setup" procedure

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  checkerboard-patches ;; colours backgroud (see below for code)

  ;; the list of colors objects can be
  set people-colours [red blue gree
  set number-kinds length people-co

  ;; first create the other people,
  ;; the parameter "population" is
  create-others population [
    set shape "circle"
    set age random 100
    set size age / 1000
    set color random-kind
    shift-randomly
  ]

  ;; do the same for the focus pers
  create-people 1 [
    set shape "person"
    set size 0.2
    set age 50
    set size 0.2
    set color random-kind
    shift-randomly
  ]

  reset-ticks                              ;; this initialises the simulation time sys
end
```

This uses the value "population" (set by the slider) to create that many agents of the kind "others". It does the commands inside the **[…]** for each new agent as it is made.

Manchester Metropolitan University

The “setup” procedure

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  checkerboard-patches ;; colours backgroud (see below for code)

  ;; the list of colors objects can be
  set people-colours [red blue green yellow orange violet cyan brown magenta white]
  set number-kinds length people-colours ;; records how many colors there are

  ;; first create the other people, shown as circles, put them on a random patch
  ;; the parameter "population" is determined by a slider
  create-others population [
    set shape "circle"
    set age random 100
    set size age / 1000
    set color random-kind
    shift-randomly
  ]

  ;; do the same for the focus
  create-people 1 [
    set shape "person"
    set size 0.2
    set age 50
    set size 0.2
    set color random-kind
    shift-randomly
  ]

  reset-ticks                          ;; this initialises the simulation time sys
end
```

This does a similar thing but only creates 1 agent, this time of the kind “people”, then setting its shape, size, age etc.

Manchester Metropolitan University

The “setup” procedure

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  checkerboard-patches ;; colours backgroud (see below for code)

  ;; the list of colors objects can be
  set people-colours [red blue green yellow orange violet cyan brown magenta white]
  set number-kinds length people-colours ;; records how many colors there are

  ;; first create the other people, shown as circles, put them on a random patch
  ;; the parameter "population" is determined by a slider
  create-others population [
    set shape "circle"
    set age random 100
    set size age / 1000
    set color random-kind
    shift-randomly
  ]

  ;; do the same for the focus person,
  create-people 1 [
    set shape "person"
    set size 0.2
    set age 50
    set size 0.2
    set color random-kind
    shift-randomly
  ]

  reset-ticks                            ;; this initialises the simulation time sys
end
```

This command starts the simulation time going

Some Program Code

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  ;; things that happen to ALL entities in the model
  ask others [

    ;; examples of possible commands, remove ";;" to make active
    ;;if with-probability prob-of-moving [shift-randomly]

  ]

  ;; things that only happen to the focus turtle
  ask people [

    ;; if no others with same color as self then move
    if not any? other turtles-here with [color = [color] of myself]
      [shift-randomly]

    ;; examples of other commands, remove ";;" to make active
    ;; if any? other turtles-here with [age < 10] [shift-randomly]
  ]

  tick                                    ;; this progresses the tick cou
end
```

Some Program Code

All this defines what the "**go**" command does.

So it is what the "**step**" button does once when you click on it, or what the "**go**" button does repeatedly if you select it.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  ;; things that happen to ALL entities in the model
  ask others [

    ;; examples of possible commands, remove ";;" to make active
    ;;if with-probability prob-of-moving [shift-randomly]

  ]

  ;; things that only happen to the focus turtle
  ask people [

    ;; if no others with same color as self then move
    if not any? other turtles-here with [color = [color] of myself]
      [shift-randomly]

    ;; examples of other commands, remove ";;" to make active
    ;; if any? other turtles-here with [age < 10] [shift-randomly]
  ]

  tick                                    ;; this progresses the tick cou
end
```

Some Program Code

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  ;; things that happen to ALL entities in the model
  ask others [

    ;; examples of no                                          ive
    ;;if with-probabi

  ]

  ;; things that only happen to the focus turtle
  ask people [

    ;; if no others with same color as self then move
    if not any? other turtles-here with [color = [color] of myself]
      [shift-randomly]

    ;; examples of other commands, remove ";;" to make active
    ;; if any? other turtles-here with [age < 10] [shift-randomly]
  ]

  tick                                        ;; this progresses the tick cou
end
```

This asks all the "**others**" to do some commands (at the moment there are no commands)

Some Program Code

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  ;; things that happen to ALL entities in the model
  ask others [

    ;; examples of possible commands, remove ";;" to make active
    ;;if with-probability prob-of-moving [shift-randomly]

  ]

  ;; things that only happen to the focus turtle
  ask people [

    ;; if no others with same color as self then move
    if not any? other turtles-here with [color = [color] of myself]
      [shift-randomly]

    ;; examples of other commands, remove ";;" to make active
    ;; if any? other turtles-here with [age < 10] [shift-randomly]
  ]

  tick                                    ;; this progresses the tick cou
end
```

This asks all the "**people**" to do some commands (at the moment there is only one person).

Some Program Code

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  ;; things that happen to ALL entities in the model
  ask others [

    ;; examples of possible commands, remove ";;" to make active
    ;;if with-probability prob-of-moving [shift-randomly]

  ]

  ;; things that only happen to the focus turtle
  ask people [

    ;; if no others with same color as self then move
    if not any? other turtles-here with [color = [color] of myself]
      [shift-randomly]

    ;; examples of other commands, remove ";;" to make active

  ]

  ti                                                              cou
end
```

This asks all the "**people**" to do some commands (at the moment there is only one person).

The command inside the "ask" is an "**if**" command: it says *if there are no other turtles on the same patch with the same colour then do the "shift-randomly" procedure*

Some Program Code

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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
  ;; things that happen to ALL entities in the model
  ask others [

    ;; examples of possible commands, remove ";;" to make active
    ;;if with-probability prob-of-moving [shift-randomly]

  ]

  ;; things that only happen to the focus turtle
  ask people [

    ;; if no others with same color as self then move
    if not any? other turtles-here with [color = [color] of myself]
      [shift-randomly]

    ;; examples of other commands, remove ";;" to make active
    ;; if any? other turtles-here with [age < 10] [shift-randomly]
  ]

  tick                                    ;; this progresses the tick cou
end
```

This command progresses the simulation time one unit.

Scroll down some more for…

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Finally we have definitions of the various action words/commands we
;; this makes the code easier to read and so that chunks of code can b
;; DO NOT WORRY about the detail of these (yet)

to checkerboard-patches
  ;; colours patches depending on coordinates of patch
  ask patches [set pcolor (pxcor + pycor) mod 2]
end

to-report random-kind
;;  reports a random color from the list of set possible colors
  report item (random number-kinds) people-colours
end

to shift-randomly
  ;;  Move to a random patch
  setxy random (1 + max-pxcor) random (1 + max-pycor)
  ;; randomly shift a little from centre of patch so we can see them
  set xcor xcor + random-normal 0 0.1
  set ycor ycor + random-normal 0 0.1
end

to-report with-probability [prob]
  ;; returns value "TRUE" with probability determined by input
  report random-float 1 < prob
end
```


Manchester Metropolitan University

Scroll down some more for…

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Finally we have definitions of the various action words/commands we
;; this makes the code easier to read and so that chunks of code can b
;; DO NOT WORRY about the detail of these (yet)

to checkerboard-patches
  ;; colours patches depending on coordinates of patch
  ask patches [set pcolor (pxcor + pycor) mod 2]
end

to-report random-kind
;;  reports a random color from the list of set possible colors
  report item (random number-kinds) people-colours
end

to shift-randomly
  ;;  Move to a random patch
  setxy random (1 + max-pxcor) random (1 + max-pycor)
  ;; randomly shift a little from centre of patch so we can see them
  set xcor xcor + random-normal 0 0.1
  set ycor ycor + random-normal 0 0.1
end

to-report with-probability [prob]
  ;; returns value "TRUE" with probability determined by input
  report random-float 1 < prob
end
```


Manchester Metropolitan University

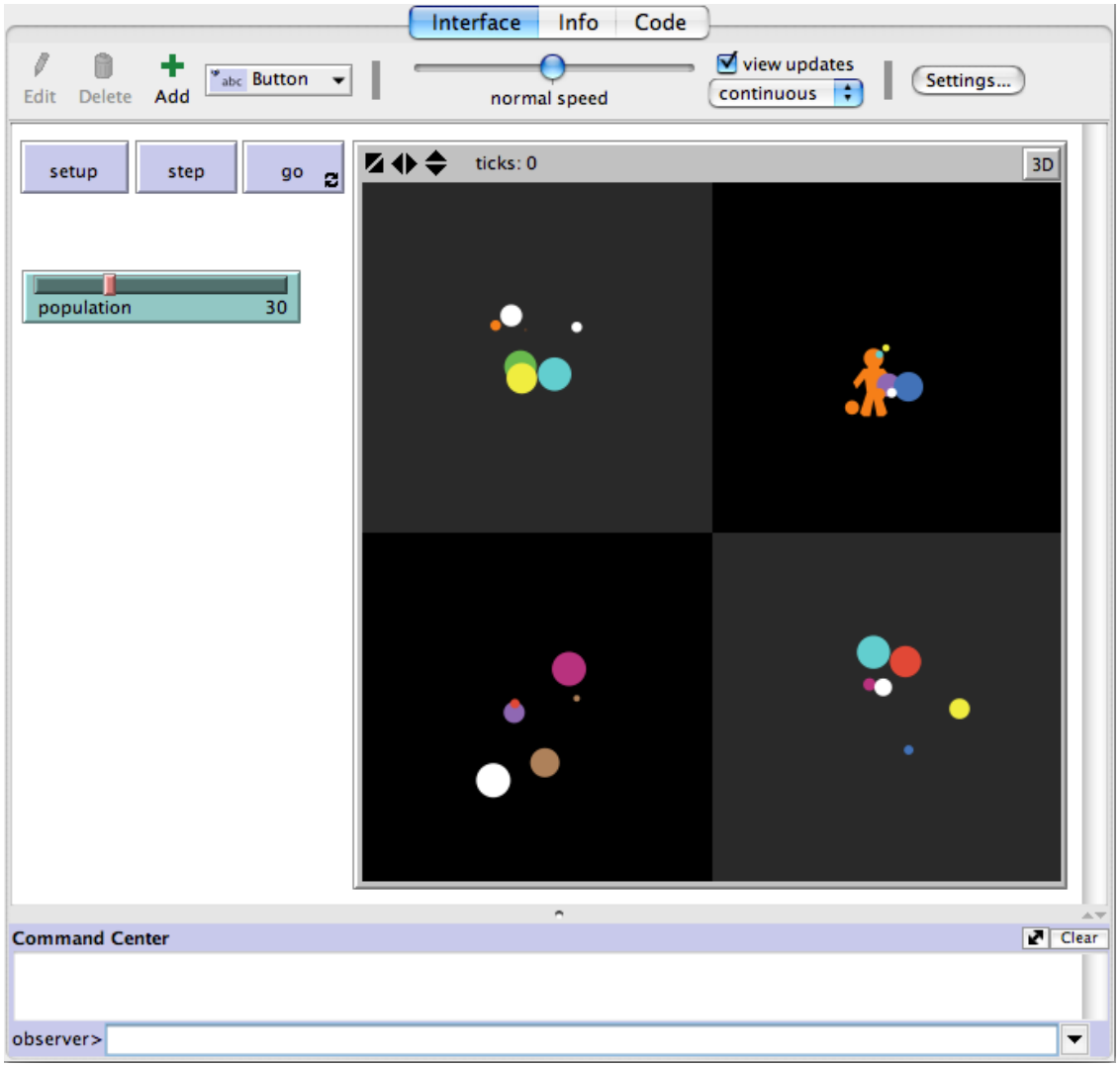
Scroll down some more for…

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Finally we have definitions of the various action words/commands we
;; this makes the code easier to read and so that chunks of code can b
;; DO NOT WORRY about the detail of these (yet)

to checkerboard-patches
  ;; colours patches depending on coordinates of patch
  ask patches [set pcolor (pxcor + pycor) mod 2]
end

to-report random-kind
;;  reports a random color from the list of set possible colors
  report item (random number-kinds) people-colours
end

to shift-randomly
  ;;  Move to a random patch
  setxy random (1 + max-pxcor) random (1 + max-pycor)
  ;; randomly shift a little from centre of patch so we can see them
  set xcor xcor + random-normal 0 0.1
  set ycor ycor + random-normal 0 0.1
end

to
  ;;                                                        input
  report random-float 1 < prob
end
```

Now click on the "Interface" tab

Running the “friends” simulation

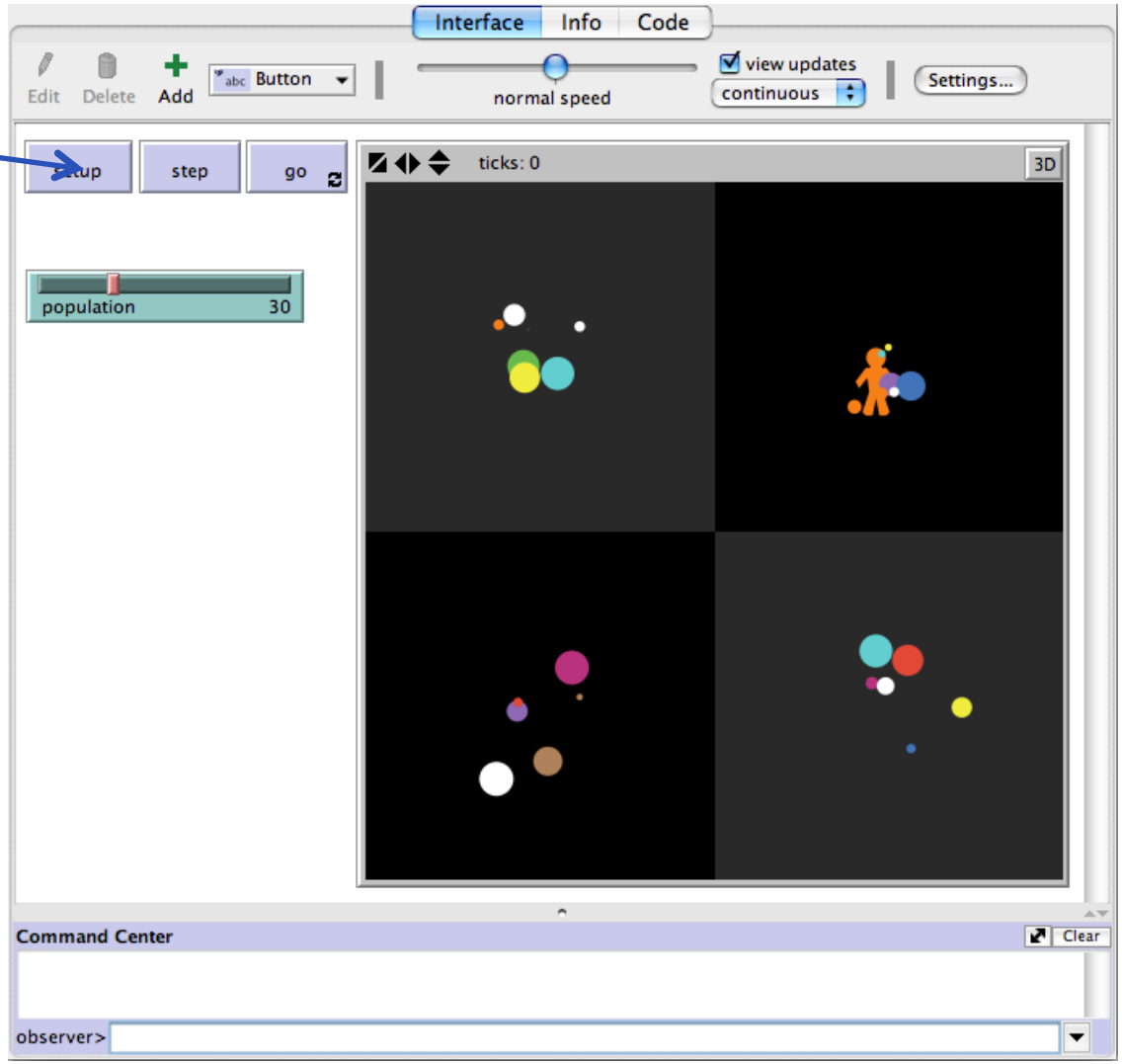
Flip back to the “setup” procedure on the Code tab and see if you can understand what this did.



Running the “friends” simulation

Press the “setup” button

Flip back to the “setup” procedure on the Code tab and see if you can understand what this did.

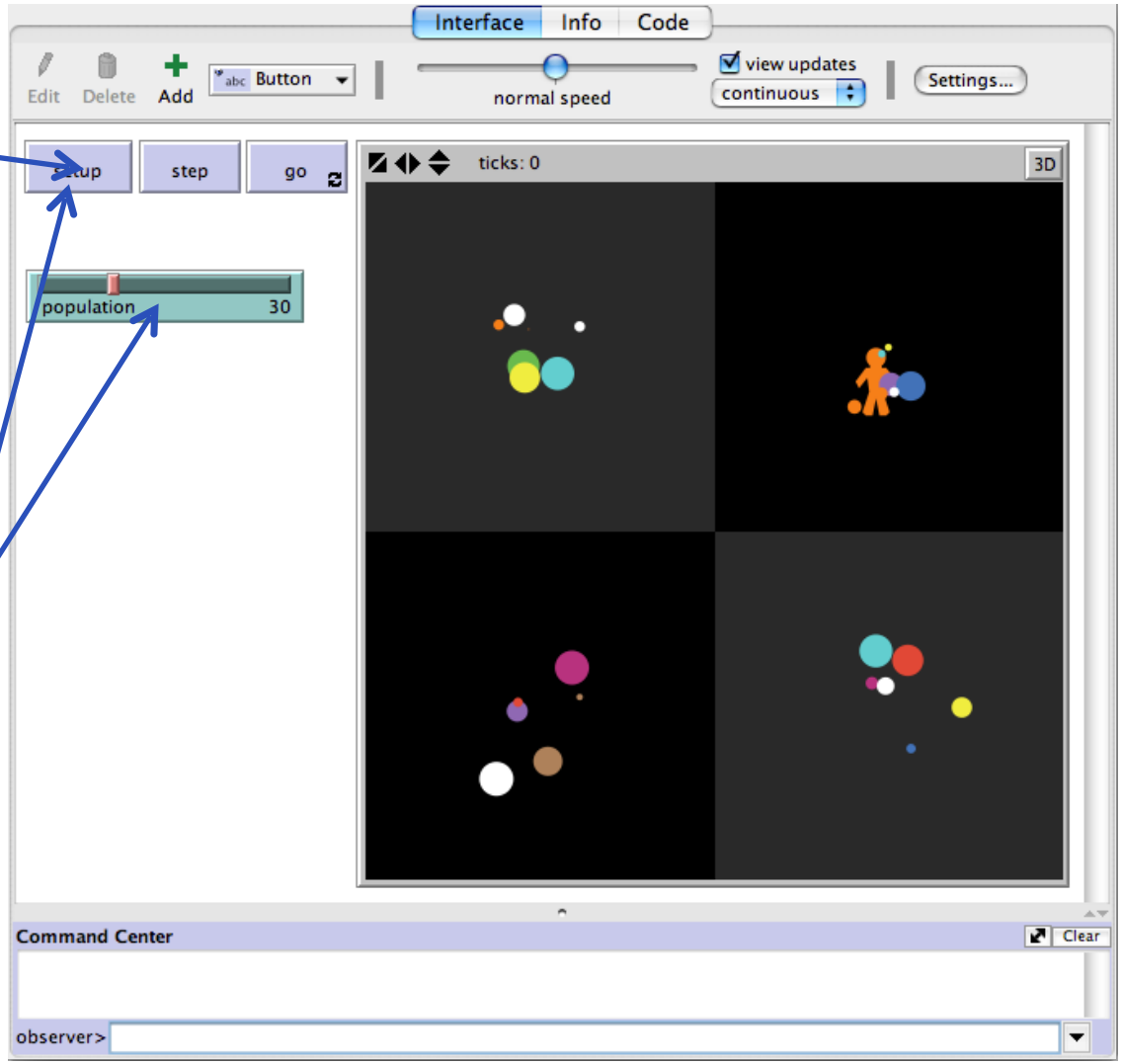


Running the “friends” simulation

Press the “setup” button

Flip back to the “setup” procedure on the Code tab and see if you can understand what this did.

Change the population slider and then press “setup” again.



Manchester Metropolitan University

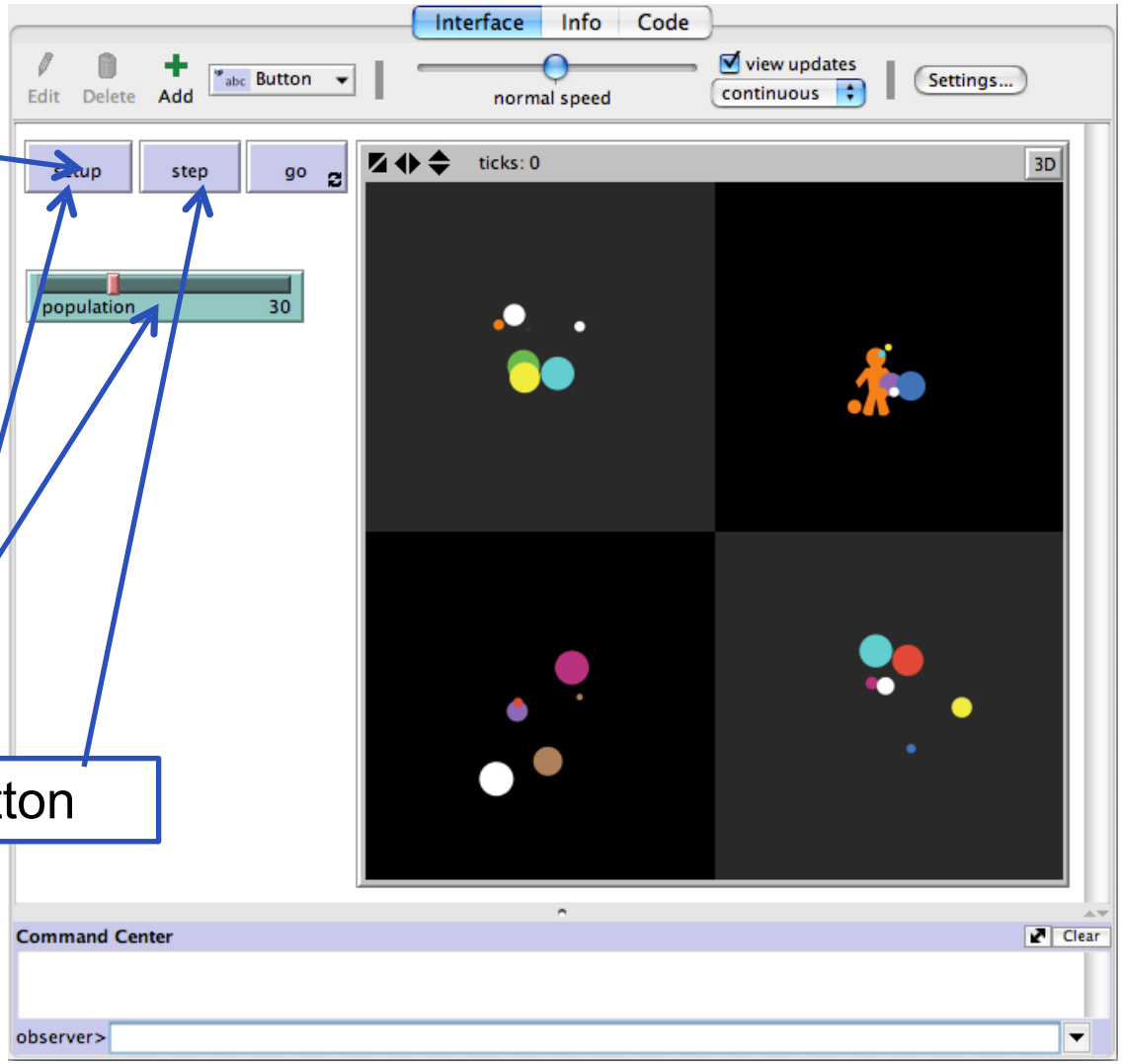
Running the “friends” simulation

Press the “setup” button

Flip back to the “setup” procedure on the Code tab and see if you can understand what this did.

Change the population slider and then press “setup” again.

Try pressing the “step” button



Running the “friends” simulation

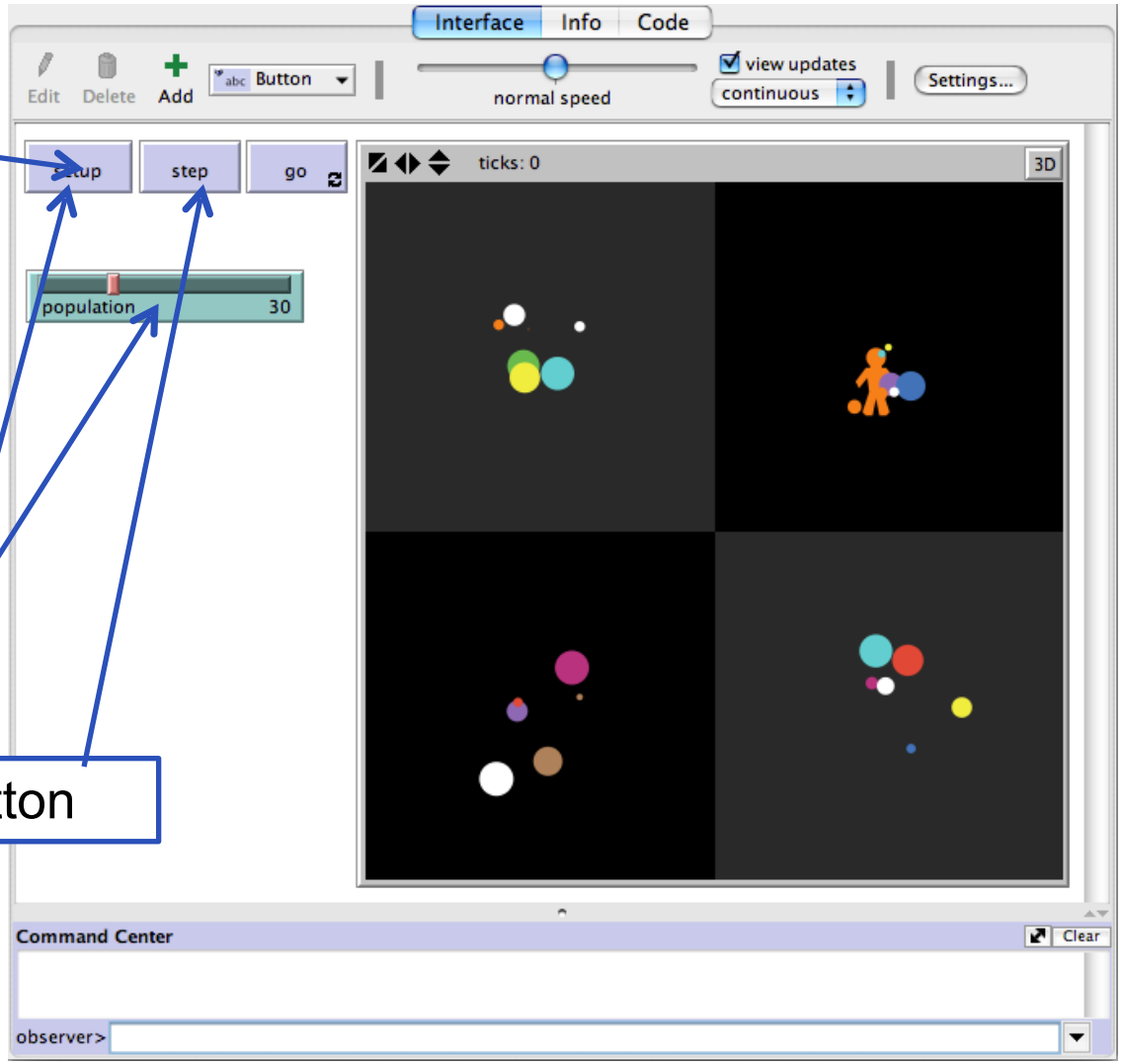
Press the “setup” button

Flip back to the “setup” procedure on the Code tab and see if you can understand what this did.

Change the population slider and then press “setup” again.

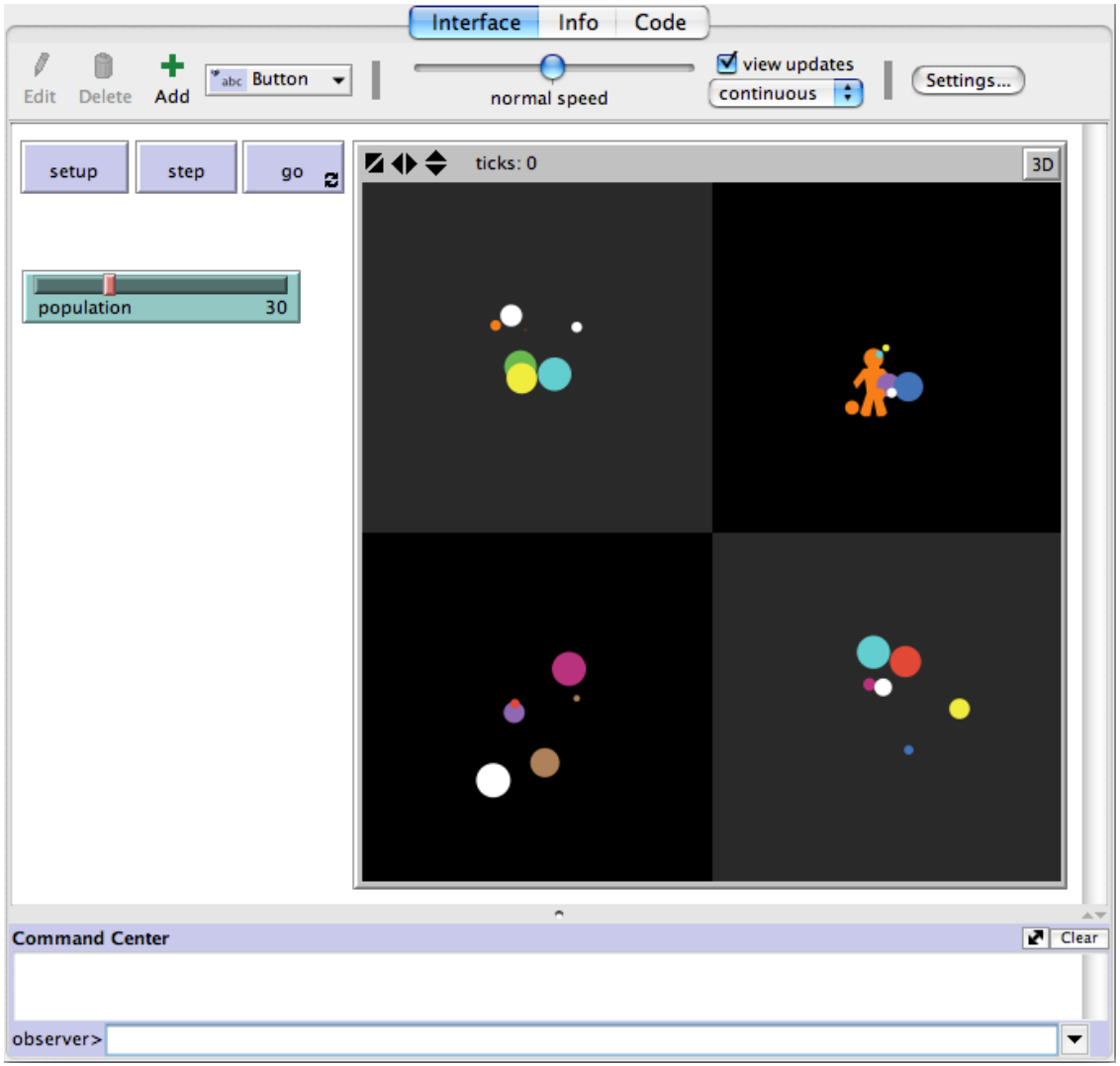
Try pressing the “step” button

Nothing much happens at the moment. Look at the “go” procedure and see if you can see why.



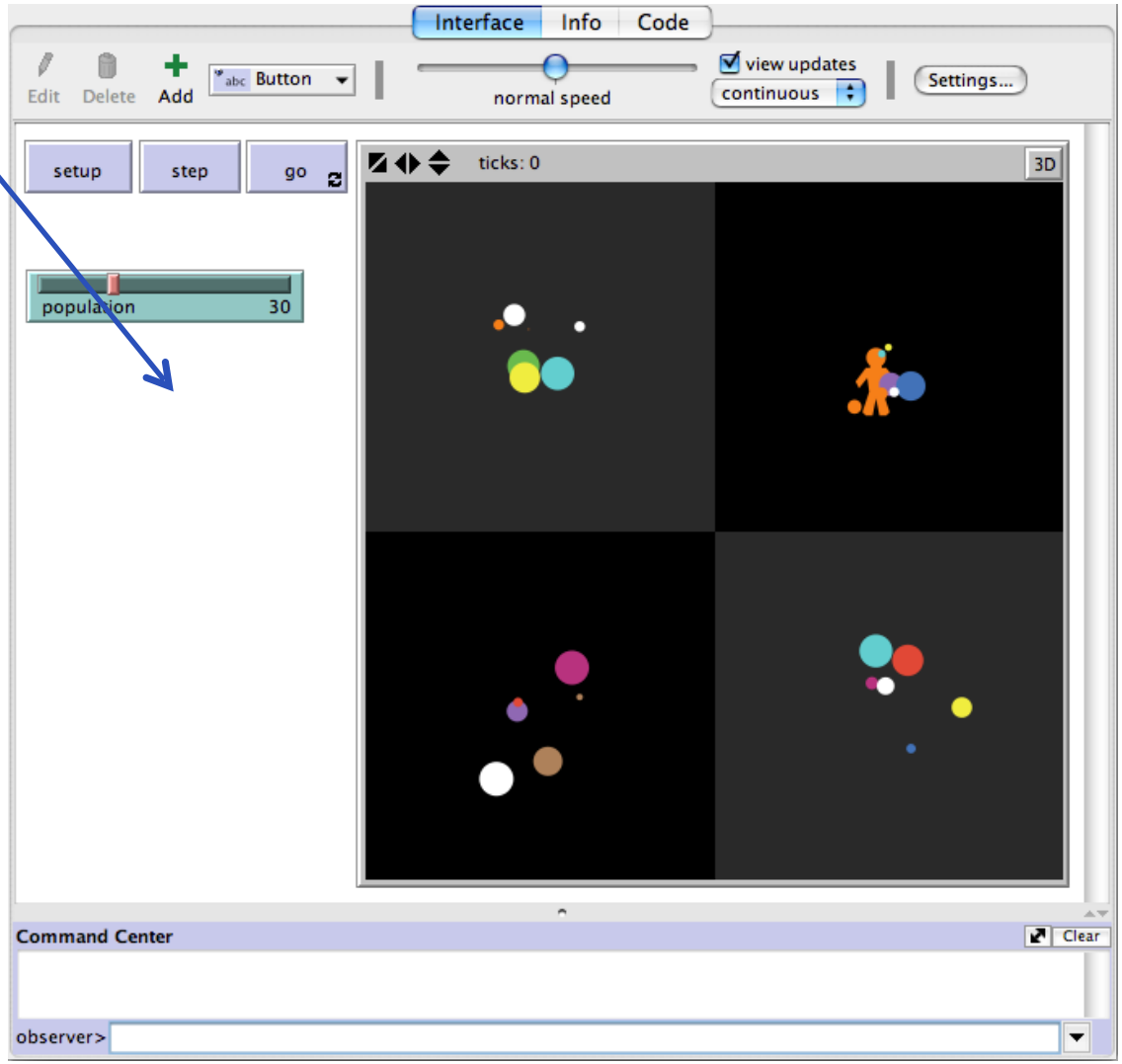
Manchester Metropolitan University

Adding a slider

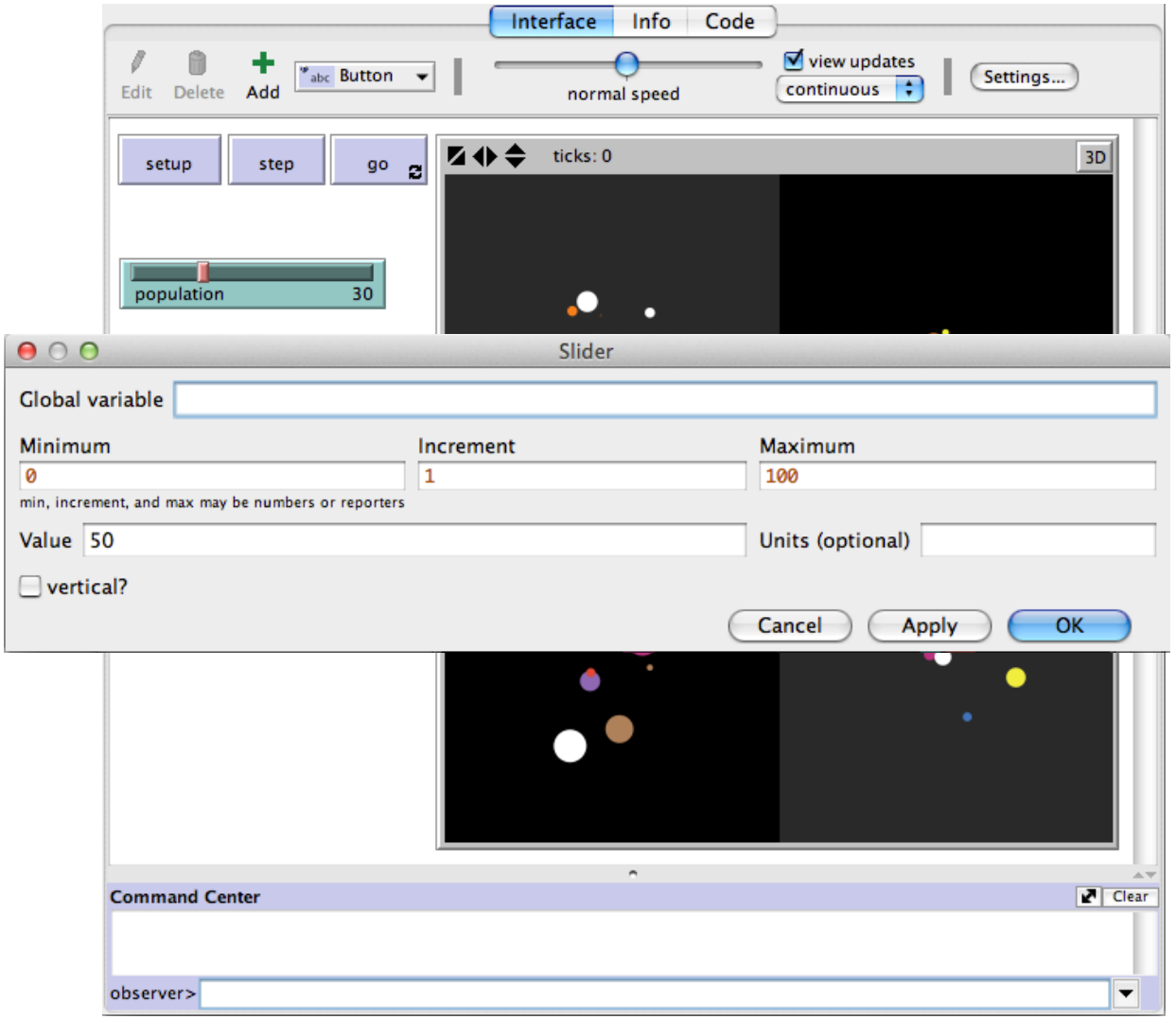


Adding a slider

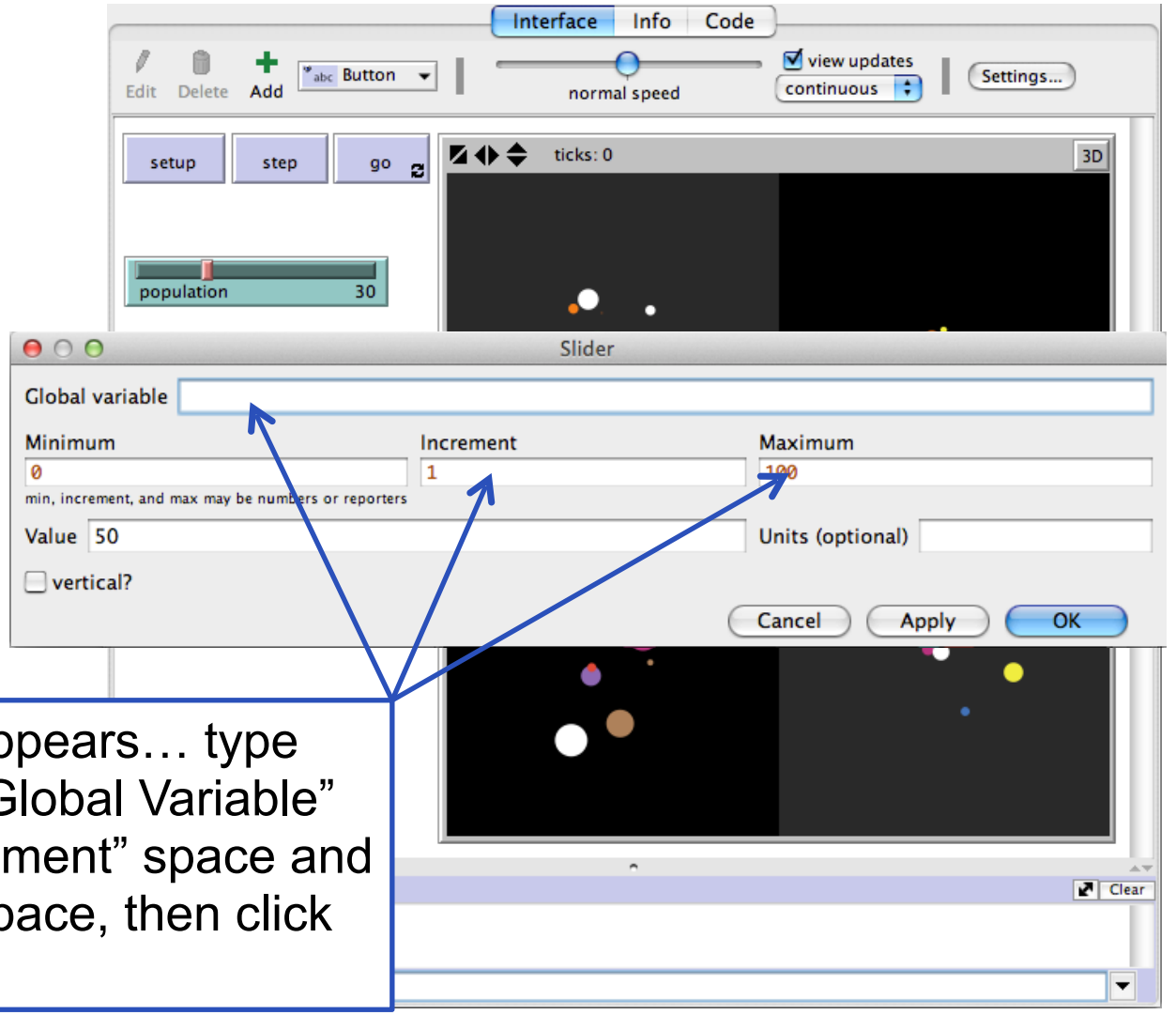
Right-click (Mac: crtl+click) on some empty space and select “Slider”



Adding a slider



Adding a slider



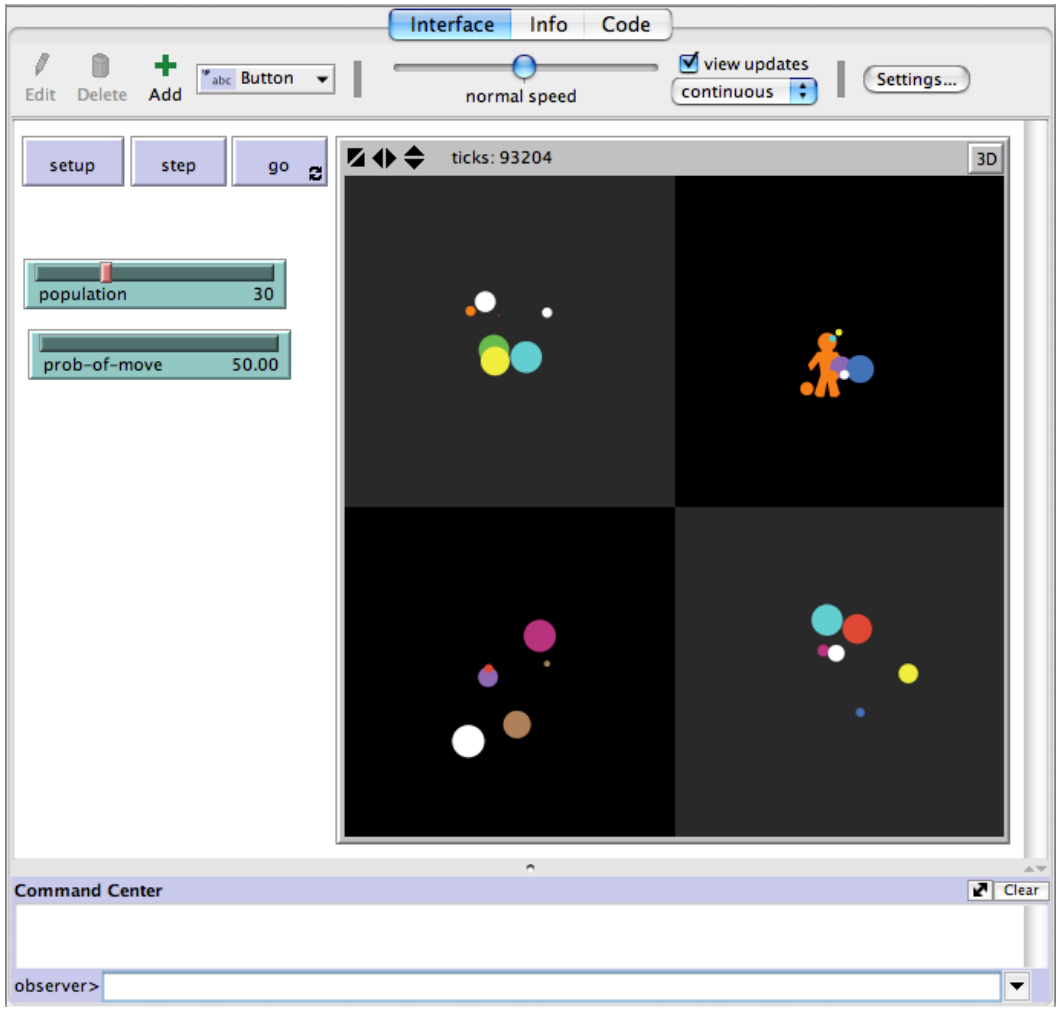
In the dialogue that appears… type "prob-of-move" in the "Global Variable" space, "0.01 in the "Increment" space and "1" in the "Maximum" space, then click "**OK**".

Changing the code to use the prob-of-move setting

Go back to the Code and go to the "go" procedure.

Then delete the two semi-colons in front of the ";;if with-probability prob-of-moving [shift-randomly]" statement to make it active.

Then go back to the Interface, select a "prob-of-move" setting and re-run the simulation. Try different settings. Work out what is happening, looking back at the code if necessary.



Other things to try

- Go back to the code, activate the ";; if any? other turtles-here with [age < 10] [shift-randomly]" statement by deleting the two semi-colons in front of it
- Add another slider to set "number-of-people" and change the code in the setup procedure to change the number of "people" created
- Add a statement to increase the age of "others" each simulation time click (using set age age + 1)
- Change the simulation so that there are only four colours (look at "people-colours") and then the code so that (eventually) all agents of the same color end up in the same quadrant
- Can you change the simulation so that all agents (eventually) sort themselves into similar ages
- Right-click (Mac: crtl+click) on the world view, then select "Edit…" then change the settings for "max-pxcor" (the maximum patch x coordinate) and "max-pycor" to "2" then OK. Re-run the simulation and see what happens.

Reacting to other agents

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

The End

2-Day Introduction to Agent-Based Modelling

http://cfpm.org/simulationcourse

Manchester Metropolitan University